Volume 25 Number 2

ACTA CYBERNETICA

Szeged, 2021

ACTA CYBERNETICA

Information for authors. Acta Cybernetica publishes only original papers in the field of Computer Science. Manuscripts must be written in good English. Contributions are accepted for review with the understanding that the same work has not been published elsewhere. Papers previously published in conference proceedings, digests, preprints are eligible for consideration provided that the author informs the Editor at the time of submission and that the papers have undergone substantial revision. If authors have used their own previously published material as a basis for a new submission, they are required to cite the previous work(s) and very clearly indicate how the new submission offers substantively novel or different contributions beyond those of the previously published work(s). There are no page charges. An electronic version of the published paper is provided for the authors in PDF format.

Manuscript Formatting Requirements. All submissions must include a title page with the following elements: title of the paper; author name(s) and affiliation; name, address and email of the corresponding author; an abstract clearly stating the nature and significance of the paper. Abstracts must not include mathematical expressions or bibliographic references.

References should appear in a separate bibliography at the end of the paper, with items in alphabetical order referred to by numerals in square brackets. Please prepare your submission as one single PostScript or PDF file including all elements of the manuscript (title page, main text, illustrations, bibliography, etc.).

When your paper is accepted for publication, you will be asked to upload the complete electronic version of your manuscript. For technical reasons we can only accept files in LaTeX format. It is advisable to prepare the manuscript following the guidelines described in the author kit available at `https://cyber.bibl.u-szeged.hu/index.php/actcybern/about/submissions` even at an early stage.

Submission and Review. Manuscripts must be submitted online using the editorial management system at `https://cyber.bibl.u-szeged.hu/index.php/actcybern/submission/wizard`. Each submission is peer-reviewed by at least two referees. The length of the review process depends on many factors such as the availability of an Editor and the time it takes to locate qualified reviewers. Usually, a review process takes 6 months to be completed.

Subscription Information. Acta Cybernetica is published by the Institute of Informatics, University of Szeged, Hungary. Each volume consists of four issues, two issues are published in a calendar year. Subscription rates for one issue are as follows: 5000 Ft within Hungary, €40 outside Hungary. Special rates for distributors and bulk orders are available upon request from the publisher. Printed issues are delivered by surface mail in Europe, and by air mail to overseas countries. Claims for missing issues are accepted within six months from the publication date. Please address all requests to:

Acta Cybernetica, Institute of Informatics, University of Szeged
P.O. Box 652, H-6701 Szeged, Hungary
Tel: +36 62 546 396, Fax: +36 62 546 397, Email: `acta@inf.u-szeged.hu`

Web access. The above information along with the contents of past and current issues are available at the Acta Cybernetica homepage `https://cyber.bibl.u-szeged.hu/` .

Conference of PhD Students in Computer Science

Guest Editor:

Attila Kertész

University of Szeged, Hungary
keratt@inf.u-szeged.hu

Preface

The *12th Conference of PhD Students in Computer Science (CSCS)* was organized by the Institute of Informatics of the University of Szeged (SZTE) and held in Szeged, Hungary, between June 24–26, 2020.

The members of the *Scientific Committee* were the following representatives of the Hungarian doctoral schools in Computer Science: János Csirik (Co-Chair, SZTE), Lajos Rónyai (Co-Chair, SZTAKI, BME), Péter Baranyi (SZE), András Benczúr (ELTE), András Benczúr (SZTAKI), Hassan Charaf (BME), Tibor Csendes (SZTE), László Cser (BCE), Erzsébet Csuhaj-Varjú (ELTE), József Dombi (SZTE), István Fazekas (DE), Zoltán Fülöp (SZTE), Aurél Galántai (ÓE), Zoltán Gingl (SZTE), Tibor Gyimóthy (SZTE), Katalin Hangos (PE), Zoltán Horváth (ELTE), Márk Jelasity (SZTE), Zoltán Kása (Sapientia EMTE), László Kóczy (SZE), János Levendovszki (BME), Gyöngyvér Márton (Sapientia EMTE), Branko Milosavljevic (UNS), Valerie Novitzka (TUKE), László Nyúl (SZTE), Marius Otesteanu (UPT), Attila Pethő (DE), Vlado Stankovski (UNILJ), Tamás Szirányi (SZTAKI), Péter Szolgay (PPKE), János Sztrik (DE), János Tapolcai (BME), János Végh (ME), and Daniela Zaharie (UVT).

The members of the *Organizing Committee* were: Attila Kertész, Balázs Bánhelyi, Tamás Gergely, Judit Jász, and Zoltán Kincses.

There were more than 50 participants and 43 talks in several fields of computer science and its applications (11 sessions). The talks were going in sections in Graphs, Machine Learning, Security, Program Analysis, Healthcare, Simulation, Privacy, Computer Graphics I., Bugs, Computer Graphics II., and Distributed systems.

The talks of the students were completed by 2 plenary talks of leading scientists: Tibor Gyimóthy (University of Szeged, Hungary), and Gábor Tardos (Alfréd Rényi Institute of Mathematics, Hungary).

The open-access scientific journal Acta Cybernetica offered PhD students to publish the paper version of their presentations after a careful selection and review process. Altogether 29 manuscripts were submitted for review, out of which 22 were accepted for publication in the present special issue of Acta Cybernetica.

The full program of the conference, the collection of the abstracts and further information can be found at `https://www.inf.u-szeged.hu/~cscs/`.

On the basis of our repeated positive experiences, the conference will be organized in the future, too. According to the present plans, the next meeting will be held around the end of June 2022 in Szeged.

Attila Kertész
Guest Editor

Acta Cybernetica **25** (2021) 131–150.

Execution Time Reduction in Function Oriented Scientific Workflows*

Ali Al-Haboobi[ab] and Gabor Kecskemeti[ac]

Abstract

Scientific workflows have been an increasingly important research area of distributed systems (such as cloud computing). Researchers have shown an increased interest in the automated processing scientific applications such as workflows. Recently, Function as a Service (FaaS) has emerged as a novel distributed systems platform for processing non-interactive applications. FaaS has limitations in resource use (e.g., CPU and RAM) as well as state management. In spite of these, initial studies have already demonstrated using FaaS for processing scientific workflows. DEWE v3 executes workflows in this fashion, but it often suffers from duplicate data transfers while using FaaS. This behaviour is due to the handling of intermediate data dependency files after and before each function invocation. These data files could fill the temporary storage of the function environment. Our approach alters the job dispatch algorithm of DEWE v3 to reduce data transfers. The proposed algorithm schedules jobs with precedence requirements to primarily run in the same function invocation. We evaluate our proposed algorithm and the original algorithm with small- and large-scale Montage workflows. Our results show that the improved system can reduce the total workflow execution time of scientific workflows over DEWE v3 by about 10% when using AWS Lambda.

Keywords: scientific workflows, cloud functions, serverless architectures, makespan

1 Introduction

Over the recent years scientific workflows have been a major area of interest within the field of complex scientific applications. Large-scale scientific workflows consist

*This work was supported in part by the Hungarian Scientific Research Fund under Grant agreement OTKA FK 131793

[a]Institute of Information Technology, University of Miskolc, Miskolc, Hungary

[b]University of Kufa, Najaf, Iraq, E-mail: `al-haboobi@iit.uni-miskolc.hu`, `ali.alhaboobi@uokufa.edu.iq`, ORCID: `0000-0001-7632-2485`

[c]School of Computer Science and Mathematics, Liverpool John Moores University, Liverpool, UK, E-mail: `kecskemeti@iit.uni-miskolc.hu`, `g.kecskemeti@ljmu.ac.uk`, ORCID: `0000-0001-5716-8857`

DOI: 10.14232/actacyb.288489

of a significant number of dependent jobs that rely on the output of other jobs (i.e., precedence constraints). Each job can be executed independently when its precedence constraints are met. Montage [11], CyberShake [10], and LIGO [1] are examples of scientific workflow applications. Workflow Management Systems (WMSs - such as Pegasus [8] and Kepler [2]) are used to ensure the precedence execution order and data constraints of every job in a scientific workflow are met during their runtime.

Cloud computing is fast becoming a key instrument in executing workflows. FaaS is a recent development in the field of cloud computing, and it has already incited significant interest in processing workflows. It promises a simple function-oriented execution environment for non-interactive tasks of web applications. Just like with other cloud computing technologies, there are commercial platforms (such as AWS Lambda and Google Cloud Functions) that were developed to provide FaaS functionalities. These allow functions to be executed in environments with a few limitations. First, there are resource limits on CPU, RAM, and temporary storage use. Second, the implemented functions are expected to have stateless behaviour: the execution environment will newly instantiate and terminate for each function invocation (i.e., will not remember state from previous invocations unless some persistence technology is applied). In addition, Amazon Kinesis shard acts as an independent queue that can send workflow tasks to its own function instance.

A number of studies [12, 18, 15] have proved the ability of cloud functions to execute small- and large-scale workflows. In spite of the previously discussed limitations, DEWE v3 have executed workflows even using functions. To avoid the temporary storage use limitation, it uses Amazon S3 to store intermediate workflow data. Therefore, the workflow data needs to be downloaded/uploaded for each function invocation when dependent jobs rely on the output data of other jobs. A large amount of transfer of dependent data can occur during workflow execution between S3 and the FaaS execution environment. Consequently, this could lead to an increased communication costs and a longer makespan.

In this paper, we propose to reduce the dependency transfers in workflows using FaaSs by improving the scheduling algorithm of DEWE v3. Our proposed algorithm exploits the internal queueing mechanisms of Amazon Kinesis shards that feed into AWS Lambda function instances. We choose to move some simple WMS behaviours inside the FaaS. Our approach schedules some dependent jobs on the same shard where their preceding jobs were scheduled. As a result, these dependent jobs can utilise the output files that generated from their precedence constrains in the same invocation. As there is no need for transfers, this step reduces the total workflow execution time as well. Due to Lambda's limitations in terms of temporary storage, the larger files cannot be processed in functions, these we scheduled in a sufficiently sized VM.

We evaluated the proposed and original algorithms with small- and large-scale Montage workflows. The large one is a 6-degree Montage workflow with over eight thousand jobs requiring the transfer of 38 GBs of inputs and outputs. This workflow size was chosen because the original DEWE v3 exhibits a significant amount of re-transfer data behaviour with this workflow. To show the limitations of our approach, we also used a smaller workflow (0.1-degree Montage) that does not have significant amounts of re-transfers even with the original approach.

The proposed algorithm outperforms the original in most cases. Our results show that the proposed approach can reduce the total workflow execution time over the original DEWE v3 approach by about 10%. Our improved scheduling algorithm schedules jobs with precedence constraints on the same shard to be executed in the same Lambda invocation. As a result, it can improve the execution time of scientific workflows on the Lambda platform. In contrast, our approach does not show significant differences in the performance when testing with smaller workflows.

The rest of this paper is organized as follows: the next section presents the background knowledge and related works. Section 3 includes the explanation of DEWE v3 and the proposed algorithm. Section 4 involves the evaluation of our approach with the original algorithm of DEWE v3. Section 5 concludes the paper and suggests some future works.

2 Background Knowledge and Related Works

This section first reviews scientific workflows for scheduling and challenging of real-world experiments as well as simulation frameworks. Then an overview is presented on the most popular FaaS platforms. Finally, the section concludes with a problem statement for the current related works.

2.1 Background Knowledge

A workflow can be formulated as a Directed Acyclic Graph (DAG) that contains a collection of atomic tasks. The nodes are a set of tasks $\{T_1, T_2, ..., T_n\}$ while the edges represent data dependencies among these tasks.

Workflow scheduling is an increasingly important area regarding WMSs. It plays a critical role to achieve an optimal resource allocation for all tasks. The problem of scheduling in distributed environments is known to be NP-hard [20]. Therefore, no algorithms can achieve an optimal solution within polynomial time while some algorithms can provide approximate results in polynomial time.

Running real-world experiments for workflows is a challenge and especially for execution of large-scale. Therefore, WMS simulation has been studied by many researchers using different simulator extensions such as WorkflowSim [7] and WRENCH [6]. WorkflowSim extends the CloudSim [3] simulator, while WRENCH extends the SimGrid [5] framework. However, to date, FaaSs are not simulated in these simulator extensions for running scientific workflows. As a result, we need to restrict our experiments to smaller-scale and larger-scale with considering data transfers, but real-world executions of workflows on commercial FaaSs like Lambda.

Lambda[1] has been presented by AWS in 2014 while cloud functions (GCF[2]) have introduced by Google in 2016. In [12] they stated that Google Cloud Functions, in

[1]`https://aws.amazon.com/lambda/`
[2]`https://cloud.google.com/functions/`

its current form, is not suitable for executing scientific workflow applications due to its limited inbound and outbound socket data quota. There are two benefits when workflows are executed on FaaS systems. First, resource management is provided by the platform in a scalable way. It means the number of concurrent invocations into the infrastructure can more closely follow the actual workflow's demands without the burden on the WMS to deal with the infrastructure's management. Second, due to the nature of the lightweight functions used, the user pays for the much less overheads on computing resource consumption in contrast to more traditional Infrastructure as a Service systems. Lambda functions are stateless, thus their execution environment is initialized and ended for each function invocation. In addition, other commercial solutions also appeared on the FaaS landscape, like Microsoft Azure Functions[3] and IBM OpenWhisk Functions[4].

The above mentioned four FaaS providers were evaluated in [16, 9]. The authors proposed multiple hypotheses concerning the expected performance of cloud functions and designed several benchmarks to confirm them. Their function platforms have tested by invoking CPU, memory, and disk-intensive functions. In addition, data transfer times were also measured for these function providers. They observed different resource allocation policies at the providers. The execution performance of Lambda and GCF is based on the size of memory that is allocated for the invocation. They identified that at the time of writing, Amazon's was more flexible and performant. Moreover, they also reported that computing with cloud functions is more cost-effective than virtual machines due to practically zero delay in booting up new resources. They also indicated that due to the more fine grained invocation patterns to functions virtual machines would have to sit idle in between invocations. This behaviour results in more costs incurred by virtual machine based function oriented solutions. Consequently, we expect more users would prefer Lambda based workflows due to its efficiency and effectiveness comparing with other platforms.

2.2 Related Works

Nowadays, most scientific workflows have been processed in clouds, especially on IaaSs. Only a few related works have studied the use of FaaS platforms to execute workflows. In [17], Malawski et al. proposed five architectural alternatives to run scientific workflows on clouds. One of them introduced a system for serverless computing that integrated the HyperFlow engine with GCFs and AWS Lambda. They examined the viability of running large-scale scientific workflows on cloud functions by evaluating their implementation with a 0.25-degree and a 0.4-degree Montage workflow. They found the approach highly promising. In addition, in [18], they further tested the prototype a 0.6-degree Montage workflow as well. They stopped their experiment at a 0.6-degree workflow as they had faced problems with the temporary storage's 500 MB limitation. However, their approach already exhibits the deficiency of increased transfer of dependent data on these workflows.

[3]`https://docs.microsoft.com/en-us/azure/azure-functions/functions-overview`

[4]`https://cloud.ibm.com/docs/openwhisk?topic=openwhisk-getting-started`

In [12], Jiang et al. designed a WMS called DEWE v3 that can process scientific workflows on three various modes: (*i*) traditional clusters, (*ii*) cloud functions, and (*iii*) a hybrid mode that combines the two. It was tested with large-scale Montage workflows. They have proven that cloud functions can be used in large-scale scientific workflows with complex precedence constrains. However, their job dispatch algorithm schedules jobs to Lambda without considering on their precedence constraints to be executed in the same Lambda invocation. Consequently, more transfer of dependent data can occur during the execution between the storage service and the Lambda invocation's execution environment. This can lead to increased communication costs.

Next, Kijak et al. [15] summarized the challenges for running scientific workflows on a serverless computing platform. They presented a serverless Deadline-Budget Workflow Scheduling (SDBWS) algorithm that was transformed to support function platforms. It was tested with a small-scale 0.25-degree Montage workflow on AWS Lambda. The algorithm used different memory sizes for Lambda based on the deadline and budget constraints assigned by the user. In addition, the function resource is selected depending on the combination of cost and time. This approach was only tested on small scale and likely exhibits transfer of dependent data issues.

In contrast to the above works, [19] proposed an approach which utilised three different cloud function platforms which were Lambda, GCF, and OpenWhisk. They evaluated the platforms with a large-scale (over 5000 jobs in parallel) bag-of-tasks style workflow. The experimental results showed that Lambda and GCF can provide more computing power if one requests more memory, while OpenWhisk's performance is indifferent from this factor. Consequently, they have shown that cloud functions can provide a high level of parallelism for workflows with a large number of parallel tasks at the same time. However, they experimented with a bag-of-tasks approach where they did not consider transfer of dependent data.

In [4], they built Wukong, a new serverless parallel computing framework. It's a cost-effective, serverless, decentralized, locality-aware parallel computing framework. Its key insight is that partitioning the work of a centralized scheduler (i.e., tracking task completions, identifying and dispatching ready tasks, etc.) across a large number of Lambda executors, can greatly improve performance by permitting tasks to be scheduled in parallel, reducing resource contention during scheduling, and making task scheduling data locality-aware, with automatic resource elasticity and improved cost effectiveness. However, their approach already exhibits the deficiency for the data transfers of the precedence constraints between the different jobs of workflow.

3 Our DEWE v3 extension

To uncover the possibilities in dependency transfer optimisation, we have chosen DEWE v3 as a base WMS for our work. Our choice was due to three factors: (*i*) its scheduling technique was closest to our envisioned approach, (*ii*) it is an open source WMS, and (*iii*) it already has the implementation of Lambda as our target execution environment. To understand our extension, we first give a general overview of DEWE v3's behaviour in the following few paragraphs.

DEWE v3 can execute scientific workflows on three different approaches (traditional clusters, cloud functions, and a hybrid mode that combines the two). The FaaS platform supports AWS Lambda and Google Cloud Functions. It has executed large-scale workflows on a hybrid approach that combines traditional clusters with the FaaS platform. DEWE v3 runs a workflow engine on virtual machine. When using AWS Lambda, DEWE v3 reads the workflow definition from an XML file and based on the information found in them loads the job binaries and input files to the object storage Amazon S3. Given that Lambda has a temporary storage limit of 500MBs in the execution environment, some jobs cannot be sent to Lambda due to their large size. Jobs that are ready for execution (i.e., according to their precedence constraints) are scheduled to Amazon Kinesis shards.

Each shard acts as an independent queue that can send tasks to its own function instance. The number of tasks that a function can process in a single invocation is determined by the batch size of Kinesis. This can be configured before the workflow's execution. Next, the Lambda function will pull a batch of tasks from its own shard to execute them sequentially in a single function invocation. The number of running function instances and accompanying kinesis shards are also configurable before the workflow's runtime and this directly influences the maximum level of parallelism the workflow's execution can exhibit.

When a function instance starts to process a job, DEWE v3 needs to download its input data from Amazon S3. Similarly, when the job's processing has finished this must be also uploaded to S3 to make sure other jobs in the workflow can be scheduled due to their input data being ready. This could result in a large amount of transfer dependent data during the execution of the workflow. The transfers take place between S3 and the FaaS environment and directly increase the workflow's communication costs.

To avoid these transfers, we have focused our improvement on the scheduling algorithm of DEWE v3 which targets the Lambda platform as its execution environment. In order to reduce data transfers, during the scheduling, we not only considered the currently ready jobs, but also their successors allowing their sequential execution in a single function instance given that they would not violate Lambda's temporary storage limitation. The next subsection discloses our changes in details.

3.1 The Proposed Scheduling Algorithm

To enhance DEWE v3's data transfers, we moved some workflow management system behaviours inside Amazon's FaaS platform. We exploited the sequencing behaviour of shards and Lambdas. First, some jobs and their successors are scheduled to the same shard and function instance. The ordering of the schedule in the shard is kept in line with the job order in the workflow as prescribed by job precedence constraints. Additionally, we used the *SequenceNumberForOrdering* parameter that guarantees the order of jobs on a shard[5]. This will allow the consecutive jobs to be executed in the same Lambda invocation avoiding the need to transfer outputs and inputs if they are only used in between the given jobs. This behaviour is due to Lambda pulling a batch of jobs based on the batch size of Kinesis to execute them sequentially in an invocation. When the first job in the batch starts its processing, it will read its input data from Amazon S3. We used Amazon S3 because it makes data available through an Internet API that can be accessed anywhere. The intermediate data will be uploaded to S3 that might be needed by other jobs out of batch jobs. Finally, the Lambda will finish processing the batch by uploading the final datafiles to S3 as well.

We have extended the *LambdaWorkflowScheduler* class of DEWE v3 [6]. Our proposed algorithm mainly focuses its changes to the *setJobAsComplete* method, and our changes are depicted in algorithm 1. This algorithm changes the decision on which jobs to schedule at a particular time, while it also alters the shard selection for the jobs that have predecessors. First, we discuss these new choices through the algorithm, then we will disclose two illustrative examples which help to clarify the behaviours even further.

Algorithm 1 shows the pseudo-code of the proposed scheduling algorithm for scientific workflows. We assume that before the application of this algorithm, all jobs without predecessors were scheduled to shards already. Then, this function is invoked by each completed job (T) to release its successor jobs. In step 5, we initialise $jobsNum$ to make sure our allocations of any given shard are balanced in step 10. In step 6, we initialise $alertMax$ which will be used to determine if the current shard received sufficient jobs to fill a complete Lambda invocation batch. Next, in step 7 we initialise the array ($loadBalancing$) that will maintain the job counts on each shard. This will allow us to see if a particular shard is less used and prioritize it for future occasions to equalise the load on all of our lambda instances. Step 9 is the basic behaviour of DEWE v3, where it forgets about jobs that have been completed (called T in our case). This step allows us to determine what job is available to schedule at the moment as jobs without predecessors will become eligible to schedule. In step 10, we choose a shard that has received the minimal number of jobs so far. In step 12, the algorithm checks if the successor job T_i has no more predecessor jobs, then in step 13, the algorithm will schedule T_i to the Kinesis shard determined in the previously discussed step 10. Next, we process all successor jobs (T_j) of our just scheduled T_i. Step 16 checks if T_j has no other predecessor job but T_i. If so, then in Step 17 the algorithm will remove T_i as a predecessor job from T_j (to allow its premature schedule to the same shard that we used for T_i - this is disclosed in Step 18). To ensure the balanced use of all our function instances, step 21-24 checks if we have scheduled sufficient jobs for the next lambda invocation (i.e., the currently selected shard is allocated a complete batch worth of jobs). If so, we don't pursue scheduling any further successors to T_i. We will also remember that we exceeded the batch size of the shard, so the next

[5]`https://docs.aws.amazon.com/kinesis/latest/APIReference/API_PutRecord.html`
[6]`https://github.com/Ali-Alhaboby/DEWE.v3`

Algorithm 1 The proposed scheduling algorithm.

```
Function jobCompleted($T$)
 1: $T_i$ = successor job, $T_j$ = dependent job, $KS$ = Kinesis shard
 2: $L$ = Lambda instance, $batchSize$ = the batch size of jobs in Lambda
 3: $n$ = the number of Lambda instances equals the number of Kinesis shards
 4: $m$ = the shard number that has received the minimum number of jobs
 5: $jobsNum$ = the number of scheduled jobs to shard.
 6: $alertMax$ = alerting the number of scheduled jobs equals to $batchSize$
 7: $loadBalancing[n]$ := an array to count the number of sent jobs to each shard
 8: for $i = 1, 2, \ldots, p$ do          // $p$ is the number of successors of $T$
 9:   Remove $T$ as a predecessor job from $T_i$
10:   $m$ := find the shard number that has received the minimum number of jobs
11:   $jobsNum := 0$
12:   if $T_i$ has no precedence constraints then
13:     Schedule $T_i$ to $KS_m$ to run in $L_m$
14:     for $j = 1, 2, \ldots, q$ do   // $q$ represents the number of successor jobs of $T_i$
15:       $jobsNum := jobsNum + 1$
16:       if $T_j$ has only $T_i$ as a precedence constraint then
17:         Remove $T_i$ as a predecessor job from $T_j$
18:         Schedule $T_j$ to $KS_m$ to run in $L_m$
19:         $jobsNum := jobsNum + 1$
20:       end if
21:       if $jobsNum == batchSize$ then
22:         $alertMax$ := true
23:         break
24:       end if
25:       if $alertMax$==true then
26:         $loadBalancing[m] := loadBalancing[m] + jobsNum$
27:         $m$ := find the shard number that has received the minimum number
            of jobs
28:         $alertMax$ := false
29:         $jobsNum := 0$
30:       end if
31:     end for
32:   end if
33: end for
```

shard's schedule can be influenced according to our load balancing rules denoted by steps 26-29. Step 26 maintains the *loadBalancing* array, while step 27 selects a new shard that has received the minimum number of jobs to proceed with the scheduling of further jobs.

To further clarify how the proposed algorithm works, we apply its steps on two simple but carefully selected and crafted sample workflows. Although these

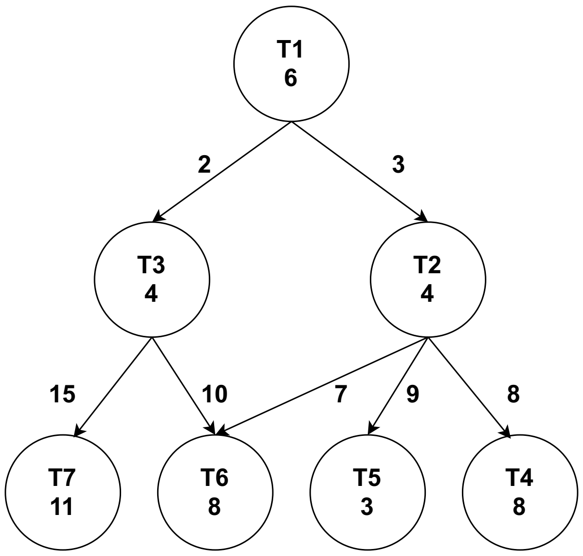


Figure 1: A sample workflow

workflows are simplified, they capture well known DAG patterns that often occur in more complex workflows. As a result, through them, we can demonstrate the applicability of our algorithm to other more complex workflows.

3.2 First illustrative example

In this subsection, we will discuss the workflow fragment, shown in Figure 1. This consists of seven tasks in the graph's nodes: $T1 - T7$. The number inside each task's node represents its estimated execution time (in seconds). On the edges between the nodes, we have also depicted the estimated data transfer time between the storage service (Amazon S3) and the FaaS execution environment.

In the following paragraphs, we will discuss how the original and our new algorithms would be applied to execute the workflow. Before we begin, we will assume the following: (*i*) there are two Kinesis shards with two Lambda function instances behind that can execute the workflow's jobs; (*ii*) each invocation downloads/uploads data files sequentially from/to Amazon S3; (*iii*) Amazon S3 will be used to store all workflow data.

First, the original algorithm would schedule T1. Once T1 completes, it will enable the schedule of T2 and T3 using both available shards. Once they complete, T4, T5, T6 and T7 will be scheduled on two shards as two invocations. Table 1 shows our analysis of the expected execution time with the original algorithm. The colouring of the Table also shows concurrent invocations (i.e., steps coloured the same execute in parallel). When we have parallel invocations, the largest execution

Table 1: The Execution Time (ET) and Transfer Time (TT) of each Lambda invocation of the original algorithm on the sample workflow of Figure 1.

Step	Tasks	ET	TT S3 to FaaS	TT FaaS to S3	Total Time
1	T1	6	-	5	11
2	T2	4	3	24	31
2	T3	4	2	25	31
3	T4, T5	11	17	-	28
3	T6, T7	19	32	-	51
					83

time of the parallel steps will be the component to be considered for the total workflow execution time (i.e., 11s for the white-, 31s for the yellow- and 51s for orange-steps). Finally, for DEWE v3's original algorithm, the Table also discloses our estimated total execution time of 83s in bold.

Now let's compare this approach to our improved scheduling algorithm. We first schedule all tasks that have no predecessor tasks such as T1 which is the same behaviour as before. The commonalities stop here though. Next, when T1 completes, T2 and T3 will become ready. Then, to reduce data transfers, our algorithm will schedule their successor tasks (T4, T5, and T7) as well. It will schedule T2, T4, and T5 on the same shard to be executed in the same function invocation. Also, it will schedule T3 and T7 on the same shard to run on the same invocation. At this time T6 is still left out of schedule because it has two predecessor tasks and we would need both of their outputs before we could start executing T6. Finally, when T2 and T3 complete, they will release T6 to be ready. In Table 2, we computed the Transfer Time (TT) FaaS to S3 in Step 2 because T2 and T3 have a child task T6 which is not scheduled. Therefore, all the data dependency files generated from T2 and T3 need to be uploaded to Amazon S3 in order to make them available to T6. Due to our algorithm's load balancing behaviour, T6 will execute in the same shard T3 and T7 did (as that shard executed the fewest jobs thus far). Similarly to the original algorithm's analysis Table, we have presented our analysis for the new algorithm as well in Table 2. We have concluded that the total workflow execution time of our improved algorithm on this workflow is expected to be significantly better at 68s.

3.3 Second illustrative example

In this subsection, we will discuss the workflow fragment, shown in Figure 1. This fragment has taken from a 0.1-degree Montage workflow that we used in our experiment.

In our second illustrative example, we explain how the proposed algorithm relies on the structure of workflow. We used a workflow fragment that has taken from a 0.1-degree Montage workflow that we used in our experiment. This workflow

Table 2: The Execution Time (ET) and Transfer Time (TT) of each Lambda invocation of the proposed algorithm on the sample workflow of Figure 1.

Step	Tasks	ET	TT S3 to FaaS	TT FaaS to S3	Total Time
1	T1	6	-	5	11
2	T2, T4, T5	15	3	24	42
2	T3, T7	15	2	25	42
3	T6	8	7	-	15
					68

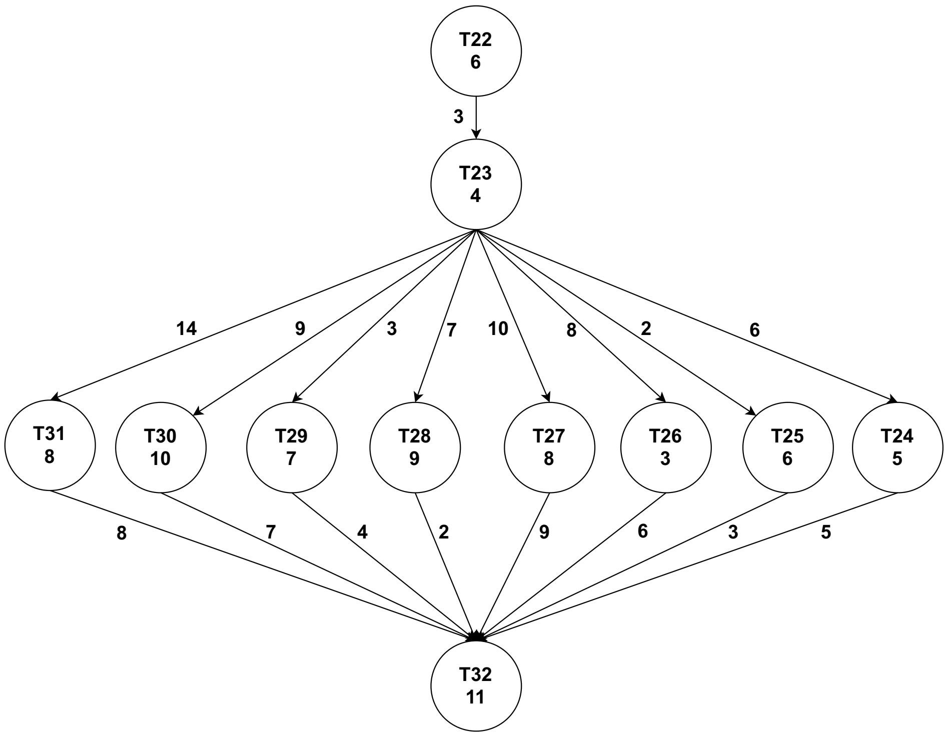


Figure 2: A workflow fragment of a 0.1-degree Montage workflow

(shown in Figure 2) consists of eleven tasks ($T22 - T32$). We will use the same assumptions of the previous example, while also having a batch size of ten. Now we apply both algorithms as follows.

Again, the original algorithm schedules T22 then waits for its completion. Afterwards, it will schedule T23 on one of the two shards. Next, when this task completes, T24-31 will be scheduled on one of the two shards because the batch

Table 3: The Execution Time (ET) and Transfer Time (TT) of each Lambda invocation of the original algorithm on the sample workflow of Figure 2.

Step	Tasks	ET	TT S3 to FaaS	TT FaaS to S3	Total Time
1	T22	6	-	3	9
2	T23	4	3	59	66
3	T24, T25, T26, T27, T28, T29, T30, T31	56	59	44	159
4	T32	11	44	-	55
					289

Table 4: The Execution Time (ET) and Transfer Time (TT) of each Lambda invocation of the proposed algorithm on the sample workflow of Figure 2.

Step	Tasks	ET	TT S3 to FaaS	TT FaaS to S3	Total Time
1	T22	6	-	3	9
2	T23, T24, T25, T26, T27, T28, T29, T30, T31	60	3	59	122
3	T32	11	44	-	55
					186

size of each Lambda instance is 10. Finally, when they complete, they will release T32 to be ready. The total workflow execution time of the original algorithm is estimated to be 289s based on our analysis of Table 3.

With the proposed algorithm a few steps change again. First, as T22 does not have a predecessor, we proceed as the original algorithm. Once it completes, T23-31 will be notified of the completion of one of their predecessors. As our algorithm also schedules successor tasks, T24-31 will also be scheduled to reduce data dependency transfers. All the tasks will be allocated to one of the shards because the batch size of each Lambda instance is 10. They will allocate to the same shard. Finally, when they complete, they will release T32 to be ready. In Table 4, we estimate the total workflow execution time of our algorithm to be 186s which is a significant improvement over the original approach.

With these two illustrative examples we have demonstrated the potential of our algorithm. In the following section, we will evaluate it on both smaller and larger scale real-life workflow executions.

4 Scheduling experiment

In our experiment, we have evaluated our proposed algorithm as well as the original from DEWE v3 on three different approaches (with/without data dependencies on smaller and larger scale). In all three cases, we choose to evaluate through the well known Montage workflow as this makes our results comparable to the previous studies in the related works. Montage is a compute-intensive astronomy workflow for generating custom mosaics of the sky. Montage was also used for different benchmarks and performance evaluation in the past [13]. To ensure good quality data collection, we have repeated all experiments described in this section three times and we reported the average measurement result for each experiment. Each experiment was repeated three times because we obtained the relative consistency of the results by three executions. In addition, we calculated the boxplot visualization that displays the data distribution based on five-number summary (i.e., minimum, first quartile, median, third quartile and maximum) on Figures 3, 4, 5 and 6.

4.1 Evaluation without processing data transfers

First, we have evaluated both algorithms with 2.0 and 4.0 degree Montage workflows (these are medium and larger scale workflows). In this first experiment, we wanted to demonstrate that our algorithmic changes have only negligible influences on the execution time when data transfers play little or no role in a workflow's makespan. Without data transfers our approach should not be able to make its gains. As a result, this experiment can only differ due to execution time circumstances or due to algorithmic changes. This experiment will show the variance of the results without any influence from data transfers. Consequently, we can use the observed differences between the original and the new algorithm as the baseline (i.e., if we see proportionally similar results for the later experiments then the later results would not be significant). The configurations of the experiment are as follows:

1. The Lambda Memory sizes were 512, 1024, 1536, 2048 and 3008 MB
2. The Lambda execution duration limit was 900 seconds.
3. The batch size of the Lambda function was 30.
4. The number of Kinesis shards was set to 5.
5. The VM was t2.micro instance as a free tier with 1 vCPU 2.5 GHz, Intel Xeon Family, and 1 GiB memory.

Figure 3 shows the total execution time of both systems with 2.0-degree workflow on five different memory sizes of Lambda. The differences between the original and the new algorithms have a mean absolute percentage error (MAPE) of 9.96%. While Figure 4 illustrates the total execution time of both systems with 4.0-degree workflow on five different memory sizes of Lambda. In the second case, the MAPE of the total execution time have been calculated as 2.19%. Thus we can conclude

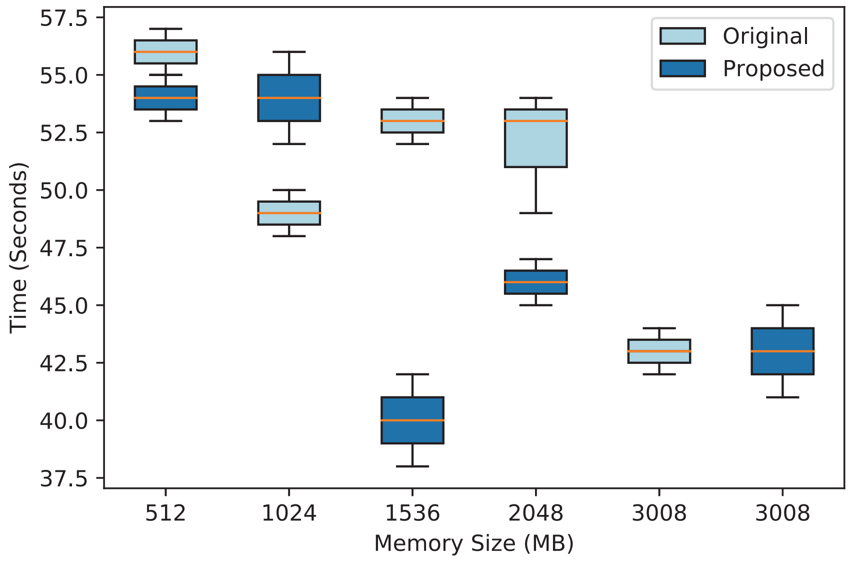


Figure 3: The boxplot visualization of total Execution Time (ET) of both systems with a 2.0-degree Montage workflow without data transfers running on different Lambda memory sizes.

that our changes could manifest in a $\sim$ 6% (average) MAPE. Therefore, in the rest of our experiments results with higher average error values than 6% show that our measurements can be considered as a significant difference. We repeated some memory sizes on the X-axis of Figures 3 and 4 because the boxplot visualization has similar results for both systems.

4.2 Small-scale evaluation

Next, we have evaluated both the original and the new algorithm with a 0.1-degree Montage workflow that also processed its data transfers. We have selected the 0.1 degree one to validate that testing with smaller Montage workflows does not show significant differences with regards to the total execution time (i.e., we show that our approach does not introduce execution time penalties even on smaller workflows where transfers are marginal). The 0.10-degree Montage workflow is sufficiently small for this as it consists of 33 tasks only. The configurations of the experiment are as follows:

1. The Lambda Memory sizes were: 512, 1024, 1536, 2048 and 3008 MB
2. The Lambda execution duration was 900 seconds.
3. The batch size of the Lambda function was 10.

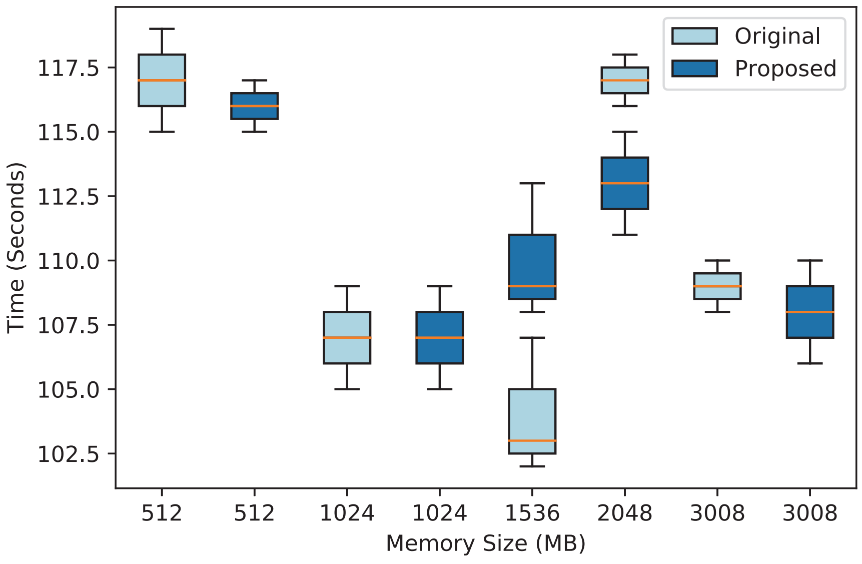


Figure 4: The boxplot visualization of total Execution Time (ET) of both systems with a 4.0-degree Montage workflow without data transfers running on different Lambda memory sizes.

4. The number of Kinesis shards was set to 2.

5. The VM was t2.micro instance as a free tier with 1 vCPU 2.5 GHz, Intel Xeon Family, and 1 GiB memory.

Figure 5 shows the total execution time of both systems with five different memory sizes of Lambda. The MAPE for this series of measurements was 13.95%. This shows that our algorithm has some minimal positive effects already for small-scale workflows as we have arrived to a MAPE value which is over 10% that we have seen in our control experiment in the previous subsection. The results about the lambda with the smallest memory configuration are inconclusive and needs further experimentation to clarify the exact reasons, however it is likely to be caused by the significantly weaker computing performance of those lambda memory configurations.

4.3 Large-scale evaluation

Finally, we have concluded our experiments by evaluating both systems with a 6.0-degree Montage workflow with processing data transfers. This workflow has over eight thousand jobs requiring total data transfers with the size of 38GBs. We have selected this workflow size because in our past analysis, DEWE v3 has already

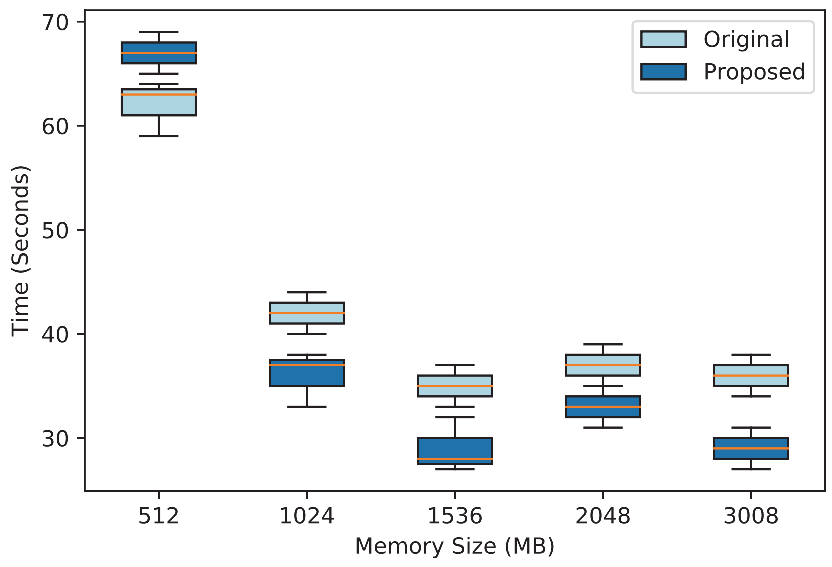


Figure 5: The boxplot visualization of total Execution Time (ET) of both systems with a 0.1-degree Montage workflow with data transfers running on different Lambda memory sizes.

shown a large amount of re-transfer data behaviour. Ideally, our improved DEWE v3 does not have this issue with such large-scale re-transfer-prone workflows. Due to the large expected dependency files of some of the workflow's jobs (namely mAdd), this experiment also used a larger Virtual Machine (VM) alongside the usual lambda functions (as such, all mAdd jobs were executed on the VM). The configurations of the experiment are as follows:

1. The Lambda Memory size was 3008 MB
2. The Lambda execution duration was 900 seconds.
3. The batch size of the Lambda function was 20.
4. The number of Kinesis shards was set to 30.
5. The virtual machine was t2.xlarge that has the following features: 16 GiB of memory and 4 vCPUs.

Figure 6 shows the total execution time of both systems. The proposed algorithm has reduced the total execution time of the large-scale workflow over DEWE v3 by approximately 10%. Thus, this experiment demonstrates that our algorithm is beneficial to be applied for larger scale workflows where the typical data dependency files are still within the 500 MB limit of the Lambda temporary storage limit

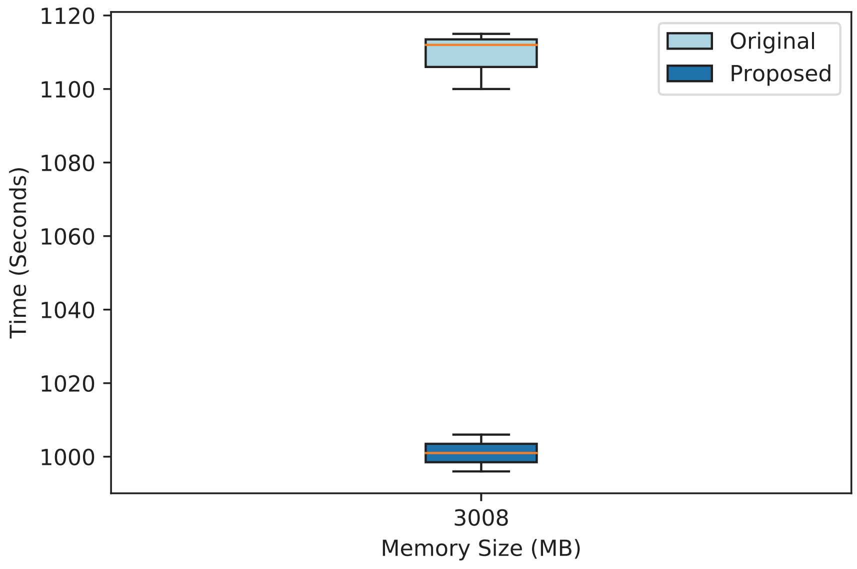


Figure 6: The boxplot visualization of total Execution Time (ET) of both systems with a 6.0-degree Montage workflow with data transfers running on Lambda.

(if this limit would be often breached, the virtual machine count would need to be extended and the cost and elasticity benefits of FaaS systems would be mostly lost). In conclusion both data transfer inducing measurements demonstrate a significantly better result over the original algorithm when we consider the control experiment in subsection 4.1.

5 Conclusion

In this paper, we have changed the job dispatch algorithm of DEWE v3 to reduce its data transfers. The main issue was that DEWE v3 has duplicated data transfers when it executes workflows on FaaSs. It was due to the uploading of intermediate data dependency files after the completion of each function invocation to allow the deletion of temporary files. Otherwise it would fill the Lambda temporary storage space over time because it has an Amazon 500 MB limit. Our proposed algorithm schedules jobs with precedence requirements on the same shard to run in the same function invocation. As a result, the dependent jobs can use the intermediate files that are produced from their predecessor jobs in the same function invocation. We have evaluated our proposed- and the original algorithms with small- and large-scale Montage workflows. Our results show that the improved system can reduce the total workflow execution time of scientific workflows over the original DEWE v3 approach by about 10% when targeting FaaS systems.

In our future work, we will extend the improved system to run on heterogeneous memory sizes of cloud functions to reduce the execution time and cost. In addition, we will study the behaviour of other scientific workflows to make the results more generally applicable. Moreover, we will introduce a Workflow Management System (WMS) simulation for the DISSECT-CF [14] simulator in order to enable the simulation and the execution of scientific workflows on different, reproducible environments. This would foster the creation of more efficient, multi target (i.e., cloud, FaaS, fog etc) workflow scheduling. Finally, we will consider Amazon Elastic File System (EFS) instead of Amazon S3 for storage workflows' data to investigate it in terms of performance, availability, and cost.

References

[1] Abramovici, Alex, Althouse, William E, Drever, Ronald WP, Gürsel, Yekta, Kawamura, Seiji, Raab, Frederick J, Shoemaker, David, Sievers, Lisa, Spero, Robert E, Thorne, Kip S, et al. LIGO: The laser interferometer gravitational-wave observatory. *Science*, 256(5055):325–333, 1992. DOI: `10.1126/science.256.5055.325`.

[2] Altintas, Ilkay, Berkley, Chad, Jaeger, Efrat, Jones, Matthew, Ludascher, Bertram, and Mock, Steve. Kepler: An extensible system for design and execution of scientific workflows. In *Proceedings of the 16th International Conference on Scientific and Statistical Database Management*, pages 423–424. IEEE, 2004. DOI: `10.1109/SSDM.2004.1311241`.

[3] Calheiros, Rodrigo N, Ranjan, Rajiv, Beloglazov, Anton, De Rose, César AF, and Buyya, Rajkumar. CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and experience*, 41(1):23–50, 2011. DOI: `10.1002/spe.995`.

[4] Carver, Benjamin, Zhang, Jingyuan, Wang, Ao, Anwar, Ali, Wu, Panruo, and Cheng, Yue. Wukong: A scalable and locality-enhanced framework for serverless parallel computing. In *Proceedings of the 11th ACM Symposium on Cloud Computing*, pages 1–15, 2020. DOI: `10.1145/3419111.3421286`.

[5] Casanova, Henri, Giersch, Arnaud, Legrand, Arnaud, Quinson, Martin, and Suter, Frédéric. Versatile, scalable, and accurate simulation of distributed applications and platforms. *Journal of Parallel and Distributed Computing*, 74(10):2899–2917, 2014. DOI: `10.1016/j.jpdc.2014.06.008`.

[6] Casanova, Henri, Pandey, Suraj, Oeth, James, Tanaka, Ryan, Suter, Frédéric, and da Silva, Rafael Ferreira. Wrench: A framework for simulating workflow management systems. In *2018 IEEE/ACM Workflows in Support of Large-Scale Science (WORKS)*, pages 74–85. IEEE, 2018. DOI: `10.1109/WORKS.2018.00013`.

[7] Chen, Weiwei and Deelman, Ewa. Workflowsim: A toolkit for simulating scientific workflows in distributed environments. In *2012 IEEE 8th international conference on E-science*, pages 1–8. IEEE, 2012. DOI: `10.1109/eScience.2012.6404430`.

[8] Deelman, Ewa, Blythe, James, Gil, Yolanda, Kesselman, Carl, Mehta, Gaurang, Patil, Sonal, Su, Mei-Hui, Vahi, Karan, and Livny, Miron. Pegasus: Mapping scientific workflows onto the grid. In *European Across Grids Conference*, pages 11–20. Springer, 2004. DOI: `10.1007/978-3-540-28642-4_2`.

[9] Figiela, Kamil, Gajek, Adam, Zima, Adam, Obrok, Beata, and Malawski, Maciej. Performance evaluation of heterogeneous cloud functions. *Concurrency and Computation: Practice and Experience*, 30(23):e4792, 2018. DOI: `10.1002/cpe.4792`.

[10] Graves, Robert, Jordan, Thomas H, Callaghan, Scott, Deelman, Ewa, Field, Edward, Juve, Gideon, Kesselman, Carl, Maechling, Philip, Mehta, Gaurang, Milner, Kevin, et al. Cybershake: A physics-based seismic hazard model for southern California. *Pure and Applied Geophysics*, 168(3-4):367–381, 2011. DOI: `10.1007/s00024-010-0161-6`.

[11] Jacob, Joseph C, Katz, Daniel S, Berriman, G Bruce, Good, John, Laity, Anastasia C, Deelman, Ewa, Kesselman, Carl, Singh, Gurmeet, Su, Mei-Hui, Prince, Thomas A, et al. Montage: A grid portal and software toolkit for science-grade astronomical image mosaicking. *International Journal of Computational Science and Engineering*, 4(2), 2009. DOI: `10.1504/IJCSE.2009.026999`.

[12] Jiang, Qingye, Lee, Young Choon, and Zomaya, Albert Y. Serverless execution of scientific workflows. In *International Conference on Service-Oriented Computing*, pages 706–721. Springer, 2017. DOI: `10.1007/978-3-319-69035-3_51`.

[13] Juve, Gideon and Deelman, Ewa. Resource provisioning options for large-scale scientific workflows. In *2008 IEEE Fourth International Conference on eScience*, pages 608–613. IEEE, 2008. DOI: `10.1109/eScience.2008.160`.

[14] Kecskemeti, Gabor. DISSECT-CF: a simulator to foster energy-aware scheduling in infrastructure clouds. *Simulation Modelling Practice and Theory*, 58:188–218, 2015. DOI: `10.1016/j.simpat.2015.05.009`.

[15] Kijak, Joanna, Martyna, Piotr, Pawlik, Maciej, Balis, Bartosz, and Malawski, Maciej. Challenges for scheduling scientific workflows on cloud functions. In *11th IEEE International Conference on Cloud Computing (CLOUD)*, pages 460–467. IEEE, 2018. DOI: `10.1109/CLOUD.2018.00065`.

[16] Lee, Hyungro, Satyam, Kumar, and Fox, Geoffrey. Evaluation of production serverless computing environments. In *11th IEEE International Conference*

on Cloud Computing (CLOUD), pages 442–450. IEEE, 2018. DOI: `10.1109/CLOUD.2018.00062`.

[17] Malawski, Maciej. Towards serverless execution of scientific workflows-hyperflow case study. In *WORKS 2016 Workshop*, pages 25–33. CEUR-WS.org, 2016.

[18] Malawski, Maciej, Gajek, Adam, Zima, Adam, Balis, Bartosz, and Figiela, Kamil. Serverless execution of scientific workflows: Experiments with hyperflow, AWS lambda and Google cloud functions. *Future Generation Computer Systems*, 2017. DOI: `10.1016/j.future.2017.10.029`.

[19] Pawlik, Maciej, Figiela, Kamil, and Malawski, Maciej. Performance considerations on execution of large scale workflow applications on cloud functions. *arXiv preprint arXiv:1909.03555*, 2019. `https://arxiv.org/abs/1909.03555`.

[20] Ullman, Jeffrey D. NP-complete scheduling problems. *Journal of Computer and System sciences*, 10(3):384–393, 1975. `https://core.ac.uk/reader/82723490`.

Acta Cybernetica **25** (2021) 151–169.

Symbolic Regression for Approximating Graph Geodetic Number*

Ahmad T. Anaqreh[ab], Boglárka G.-Tóth[ac], and Tamás Vinkó[ad]

Abstract

In this work, symbolic regression with an evolutionary algorithm called Cartesian Genetic Programming, has been used to derive formulas capable to approximate the graph geodetic number, which measures the minimal-cardinality set of nodes, such that all shortest paths between its elements cover every node of the graph. Finding the exact value of the geodetic number is known to be NP-hard for general graphs. The obtained formulas are tested on random and real-world graphs. It is demonstrated how various graph properties as training data can lead to diverse formulas with different accuracy. It is also investigated which training data are really related to each property.

Keywords: symbolic regression, cartesian genetic programming, geodetic number

1 Introduction

Geodetic number is the minimal-cardinality set of nodes, such that all shortest paths between its elements cover every node of the graph [16]. Calculating the geodetic number proved to be an NP-hard problem for general graphs [5]. The integer linear programming (ILP) formulation of geodetic number problem was given in [16], containing also the first computational experiments on a set of random graphs.

The trivial upper bound for the geodetic number is $g(G) \leq n$. Chartrand *et al.* [10] proved that $g(G) \leq n - d + 1$, where d is the diameter of G. Other

*The project has been supported by the European Union, co-financed by the European Social Fund (EFOP-3.6.3-VEKOP-16-2017-00002), by grant NKFIH-1279-2/2020 of the Ministry for Innovation and Technology, Hungary and by the grant SNN-135643 of the National Research, Development and Innovation Office, Hungary.

[a]Department of Computational Optimization, Institute of Informatics, University of Szeged, Hungary

[b]E-mail: ahmad@inf.u-szeged.hu, ORCID: 0000-0002-3971-2684

[c]E-mail: boglarka@inf.u-szeged.hu, ORCID: 0000-0002-0927-111X

[d]E-mail: tvinko@inf.u-szeged.hu, ORCID: 0000-0002-3724-4725

DOI: 10.14232/actacyb.276474

upper bounds are also given in [6, 30, 31], but these are concerning specific graph structures.

Chakraborty *et al.* [9] proposed an algorithm to approximate the geodetic number on edge color multigraph. A polynomial algorithm to compute the geodetic number of interval graphs has been proposed in [12]. Greedy-type algorithms are developed in [3] to find upper bound of the geodetic number on general graphs based on shortest paths information.

There are varied applications of geodetic sets and geodetic number. Clearly, they can be applied in computational sociology as it is hinted in [7, 31]. The definition of convexity of set of nodes in a graph [18] is a somewhat converse property to geodetic set. Related notions are the graph hull number [14] and the domination number [15]. All these concepts have practical applications, e.g., in public transportation design [9], in achievement and avoidance games [8], in location problems [25], in maximizing the switchboard numbers on telephone tree graphs [23], in mobile ad hoc networks [26], and in design of efficient typologies for parallel computing [24].

Graph properties are certain attributes that could make the structure of the graph understandable. Occasionally, standard methods to calculate exact values of graph properties cannot work properly due to their huge computational complexity, especially for real-world graphs. In contrast, heuristics and metaheuristics are alternatives which have proved their ability to provide sufficient solutions in a reasonable time. However, in some cases even heuristics fail to succeed, particularly when they need some less easily obtainable global information of the graph. The problem thus should be dealt with in a completely different way by trying to find features that are related to the property, and based on these data build a formula which can approximate the graph property.

Topological representation is the simplest way to represent graphs, where the graph is a set of nodes and edges. However, the spectral representation (e.g., adjacency matrix, Laplacian matrix) can significantly help to describe the structural and functional behavior of the graph. Adjacency matrix is a square matrix in which a non-zero element indicates that the corresponding nodes are adjacent. Implementations of well known algorithms like Dijkstra's or Floyd–Warshall algorithm usually use the adjacency matrix to calculate the shortest paths for a given graph. The diameter of a graph is the length of its longest shortest path. It is known that the diameter of a given graph is small if the absolute value of the second eigenvalue of its adjacency matrix is small [11]. Laplacian matrix is a square matrix which can be used to calculate, e.g., the number of spanning trees for a given graph. The eigenvalues of the Laplacian matrix are non-negative, less than or equal to the number of nodes, and less than or equal to twice the maximum node degree [4]. Considering these important relations between the graph properties, eigenvalues of spectral matrices and more parameters (to be discussed in the forthcoming sections), which can be calculated easily even for complex graphs, symbolic regression is one of the good choices to verify the connection between graph parameters and properties, and use such parameters for approximating hard to compute network properties.

Symbolic regression (SR) is a mathematical model which attempts to find a simple formula such that it fits a given output in term of accuracy based on a set of inputs. In conventional regression techniques, a pre-specified model is proposed, while symbolic regression avoids a particular model as a starting point to give a formula. Instead, in SR, initial formulas are formed randomly by combining the inputs: parameters, operators, and constants. Then, new formulas are assembled by recombining previous formulas by using one of the evolutionary algorithms, which is the genetic programming in our work. Symbolic regression practically has infinite search space, hence infinite formulas to assemble. Nevertheless, this can be considered as an advantage when symbolic regression uses an evolutionary algorithm called genetic programming, which requires diversity to efficiently explore the search space, ensuring a highly accurate formula.

The inputs are predefined parameters and constants. SR combines these parameters and constants by a set of given arithmetic operators (such as $+, -, \times, \div$, etc.) to assemble a formula. In the papers by Schmidt and Lipson, symbolic regression was used to find physical laws based on experimental data [28], and then they used it to find analytical solutions to iterated functions of an arbitrary form [29]. Even though there are some algorithms in the literature that use symbolic regression apart from genetic programming [21], essentially genetic programming is considered as one of the most popular algorithms applied by symbolic regression [19].

The rest of the paper is structured as follows. Section 2 discusses the specific genetic programming approach we used together with the list of graph properties. Section 3 discusses the methodology used to approximate the graph geodetic number. Section 5 reports the numerical results to show the efficiency of the formulas we obtained. The conclusion of our work is presented in Section 6. In the Appendix, we report all the formulas we obtained during this work.

2 Preliminaries

2.1 Cartesian Genetic Programming

One of the most famous genetic programming tools is called Cartesian Genetic Programming (CGP) developed by Miller [22]. CGP is an iteration-based evolutionary algorithm and works as it follows. CGP begins by creating a set of initial solutions, from which the best solution is chosen by evaluating these solutions based on the fitness function. Then these solutions will be used to create the next generation in the algorithm. The next generation's solutions will be a mixture of chosen solutions from the previous generation's, where the new generation's solutions should not be identical to the previous ones', which can be done by mutation. Mutation is used to change small parts of the new solutions and it usually occurs probabilistically for CGP. The mutation rate is the probability of applying the mutation on a specific new solution. Eventually, the algorithm must terminate. There are two cases in which this occurs: the algorithm has reached the maximum number of generations, or the algorithm has reached the target fitness. At this point, a final solution is selected and returned.

Cartesian Genetic Programming has several parameters to set up, which certainly have effects on its performance. The specific parameters used in this paper are detailed later in Section 4.3.

2.2 Geodetic Number

A simple connected graph is denoted by $G = (V, E)$, where V is the set of nodes and E is the set of edges.We have $N = |V|$ and $M = |E|$. Geodetic number is the minimal-cardinality set of nodes, such that all shortest paths between its elements cover every node of the graph [16]. The formal description is as follows. Given $i, j \in V$, the set $I[i, j]$ contains all $k \in V$ which lies on any shortest paths between i and j. The union of all $I[i, j]$ for all $i, j \in S \subseteq V$ is denoted by $I[S]$, which is called as *geodetic closure* of $S \subseteq V$. Formally,

$$I[S] := \{x \in V : \exists i, j \in S, x \in I[i, j]\}.$$

The *geodetic set* is a set S for which $V = I[S]$. The *geodetic number* of G is

$$g(G) := \min\{|S| : S \subseteq V \text{ and } I[S] = V\}.$$

2.3 Graph Properties

Adjacency matrix. The adjacency matrix is a square $|N| \times |N|$ matrix A such that its element A_{ij} is equal to one when there is an edge from node i to node j, and zero when there is no edge.

Shortest path. The series of nodes $u = u_0, u_1, \ldots, u_k = v$, where u_i is adjacent to u_{i+1}, is called a *walk* between the nodes u and v. If $u_i \neq u_j$ $(\forall i, j)$, then it is called a *path.* The path's length is k. Given all paths between nodes u and v, a *shortest path* is a path with the fewest edges. Shortest paths are usually not unique between two nodes.

Diameter. Graph diameter is the length of the longest path among all the shortest paths in the graph.

Degree, degree-one node. The *degree* of a node is the number of edges linking the node to other nodes in the graph, denoted by $\deg(v)$. If $\deg(v) = 1$, which means there is only one edge connecting the node, this node is called a *degree-one node.* It is known from the literature that degree-one nodes are always part of the geodetic set, see [17]. The number of degree-one nodes in the graph is denoted by δ_1.

Laplacian matrix. The Laplacian matrix is a square $|N| \times |N|$ matrix L such that $L = D - A$, where A is the adjacency matrix and D is the degree matrix, i.e., the elements in its main diagonal are defined as $D_{ii} = \deg(v_i)$, where $v_i \in V$ $(i = 1, \ldots, N)$.

Simplicial node. node v is called a *simplicial node* if its neighbors form a clique (complete graph), namely, every two neighbors are adjacent. If G is a nontrivial connected graph and v is a simplicial node of G, then v belongs to every geodetic set of G, see [1]. The number of simplicial nodes in the graph is denoted by σ.

Betweenness centrality. Betweenness centrality (BC) for a specific node v is the proportion of all the shortest paths pass through this node. It is shown in [17] that if G is a star graph with n nodes then $g(G) = n - 1$, where the central node with the highest BC, that all the shortest paths passing through, will never be in the geodetic set. Moreover, in the tree graph G with k leaves $g(G) = k$, that means the leaves with low BC are geodetic nodes while the root and the parents with higher BC are not part of the geodetic set.

3 Methodology

Although there are not many papers proposing the idea of using symbolic regression for approximating graph properties, the work by Martens *et al.* [20] was a good starting point for us. They used the eigenvalues of the Laplacian matrix and of the adjacency matrix as inputs for CGP, the experiments made on real-world networks to optimize the diameter and isoperimetric number. In our case, we aim at obtaining results for the geodetic number on random and real-world graphs. Thus, we investigated graph properties that are strongly related to the geodetic number, which have been discussed in Section 2.3.

We have used CGP-Library which is a cross-platform Cartesian Genetic Programming implementation developed by Andrew Turner[1]. The library is written in C and it is compatible with Linux, Windows and MacOS.

In order to use CGP a set of training data is needed. Each training data will contain instances and each instance contains two parts: (i) parameters of graph properties and chosen constants as inputs, (ii) exact value of the graph property as output. Thus, CGP will attempt to join the parameters and constants by using arithmetic operators to achieve the output. The set of arithmetic operators we have used in all the cases is $\{+, -, \times, \div, \sqrt{x}, x^2, x^3\}$. For the graph properties we have used the ones discussed in Section 2.3: eigenvalues of the adjacency matrix and Laplacian matrix, number of degree-one nodes, number of simplicial nodes, etc. It will be shown that these parameters strongly related to the graph geodetic number so they can be beneficial inputs for CGP. The classification of these inputs into categories will be shown in the following section which reports the results of our numerical experiments.

[1]`http://www.cgplibrary.co.uk/`

4 Parameters of the Numerical Experiments

The main goal of our experiments was to investigate the graph geodetic number for random graphs and real-world graphs. Since the most related paper to our work of Märtens *et al.* [20] contains results for the graph diameter (which is, similarly to the geodetic number, also based on shortest paths) we report our results obtained for the diameter and compare these values. The metrics used to measure the goodness of a formula are mean absolute error and mean relative error.

In the following subsection we describe the graphs used for the training as well as for the validation.

4.1 Random Graphs

Set of 120 random graphs created by using the three well-know generative models: Erdős-Rényi [13], Watts-Strogatz [32], and Barabási-Albert [2]. Regarding the number of nodes and edges the following approach were used:

- the number of nodes were $n = 10, 20, 30, 40, 50, 60, 70, 80, 90, 100$, and
- for the number of edges we followed the scheme as in [16]:
  - for each case one can have maximum $n \cdot (n-1)/2$ edges,
  - and we took 20%, 40%, 60% and 80% of this maximum number of edges.

4.2 Real-World Graphs

As a set of real-world graphs we used 10 graphs from the Network Repository[2] [27]. For the training part, 120 connected sub-graphs of these networks with different sizes ($14 \le N \le 140$) were created from this set by using the following simple procedure. For a given real-world graph $G(V, E)$, first, a random set $W \subset V$ of nodes were selected. Then, the induced sub-graph of G with node set W is taken. This sub-graph $\hat{G}$ might not be connected, so, as a final step, the largest connected component of $\hat{G}$ is selected.

4.3 CGP Parameters

CGP needs predefined parameters to work properly. Table 1 summarizes the values of the parameters we have used in the experiments. The details of the parameters used are the following.

Evolutionary Strategy The evolutionary strategy uses selection and mutation as search operators. The usual version used by CGP is the one which we also apply in this paper, which is called $(1+4)$-ES. Here, the procedure selects the fittest individual as the parent for the next generation, from the combination of the current parent and the four children.

[2]`http://networkrepository.com/`

Table 1: Parameters of CGP

Parameter	**Value**
Evolutionary Strategy	$(1 + 4)$-ES
Node Arity	2
Mutation Type	Probabilistic
Mutation Rate	0.05
Fitness Function	Supervised Learning
Target Fitness	0.1
Selection Scheme	Select Fittest
Reproduction scheme	Mutation Random Parent
Number of generations	$200,000$
Update frequency	100
Threads	1
Function Set	`add sub mul div sqrt sq cube`

Node Arity Each node is assumed to take as many inputs as the maximum node arity value, namely, the maximum number of inputs connected to a specific node.

Mutation Type The mutation, as basic search operator of the evolutionary strategy, is performed by adding a random vector to the current solution. In our paper this is done probabilistically.

Mutation Rate The probability of applying mutation on a specific solution.

Fitness Function The supervised learning fitness function applies to each solution and assigns a fitness value to how closely the solution output match the desired output. Based on that, the solutions with better fitness value will be chosen for next generations.

Target Fitness The fitness function used in this work is the absolute differences (absolute error) between the generated and predefined outputs, where the best solution is the one with absolute difference less than or equal to the given value.

Selection Scheme The applied fittest selection schemes select the best solutions based on the closest fitness obtained by the solution.

Reproduction scheme There are two ways in which new children can be created from their parents. In the first method the child is simply a mutated copy of the parent. In the second method the child is a combination from both parents with or without mutation. This latter method is referred to recombination. Usually, CGP-Library uses the random parent reproduction scheme which simply creates each child as a mutated version of its parents.

Number of generations How many iterations CGP will apply before termination, unless one of the solutions obtained the target fitness.

Update frequency The frequency at which the user is updated on progress, where the progress details shown on the terminal.

Threads The number of threads the CGP library will use internally.

Function Set the arithmetic operators used by CGP to combine the inputs.

4.4 Training data parameters

The list of parameters used as input in the training data, separated into different sets as follows.
For random graphs:

1) $N, M, \lambda_N, \lambda_i$ $(i = 1, 2, 3)$
2) N, M, μ_{N-1}, μ_i $(i = 1, 2, 3)$
3) $N, M, \lambda_i, \lambda_{N-i-1}$ $(i = 1, \dots, 5)$
4) N, M, μ_i, μ_{N-i-1} $(i = 1, \dots, 5)$
5) $N, M, \lambda_i, \lambda_{N-i-1}$ $(i = 1, \dots, 5)$ and constants $1, 2, 3, 4, 5$
6) N, M, μ_i, μ_{N-i-1} $(i = 1, \dots, 5)$ and constants $1, 2, 3, 4, 5$

where N is the number of nodes, M is number of edges, λ_i is the i-th eigenvalue of adjacency matrix, μ_i is the i-th eigenvalue of Laplacian matrix.

For real-world graphs:

1) $N, M, \delta_1, \sigma,$ and constants $1, 2, 3, 4, 5$
2) $N, M, \delta_1, \sigma, \lambda_i, \lambda_{N-i-1}$ $(i = 1, \dots, 5)$
3) $N, M, \delta_1, \sigma, \mu_i, \mu_{N-i-1}$ $(i = 1, \dots, 5)$
4) $N, M, \delta_1, \sigma, \lambda_i, \lambda_{N-i-1}$ $(i = 1, \dots, 5)$ and constants $1, 2, 3, 4, 5$
5) $N, M, \delta_1, \sigma, \mu_i, \mu_{N-i-1}$ $(i = 1, \dots, 5)$ and constants $1, 2, 3, 4, 5$

where δ_1 is the number of nodes with degree one in the graph, σ is the number of simplicial nodes in the graph.

Note that in Section 2.3 the betweenness centrality was also discussed as shortest path based graph centrality measure, which has relation to the geodetic number. In the conducted experiments we were trying to involve the betweenness values of the nodes by putting them into categories. However, none of the best approximating formulas we have obtained by the symbolic regression included this information.

5 Results

To obtain the formulas for either random graphs or real-world graphs, we have run the CGP dozen times for each different category (see Section 4.4 for the list of these categories). Amongst all the formulas we choose the best ones according to its output's absolute error and relative error compared to the exact value. Hence, the best formulas gave the smallest error. The full list of the chosen formulas are given in the Appendix. In the following we report and discuss the top formulas for each case.

Both the diameter and the geodetic number are of course integers. However, the obtained formulas by the symbolic regression usually result in non-integer number. Hence, in the tables which report the results, first we rounded the values given by the formulas and then the errors were calculated.

Consequently, the results are reported in two types of tables. For the random graphs, only the summary of the approximation errors are shown. Regarding the real-world graphs, the full details are given, i.e., the calculated values of the diameter as well as the geodetic number using the best formulas are presented.

5.1 Diameter

5.1.1 Random graphs

Table 2 summarizes the results obtained for the different random graphs, where (6) gives the best approximation:

$$\frac{N + \lambda_{N-2} + 4}{\sqrt{M}}.$$

For the investigated set of random graphs, λ_{N-2} is in the range $[-7, -1]$, which is, on average,cancelled out by the constant factor $+4$ in formula (6). Moreover, for these graphs we have $M = O(N)$, which means that formula (6) is roughly $O(\sqrt{N})$. The square root function, at least in the range where our experiments were done, is close to the logarithm function. It is well know that the diameter of (random) graphs can be estimated by $\log(N)$. The symbolic regression did not use the log

Table 2: Diameter validations on random graphs

	formula	(2)	(3)	(4)	(5)	(6)	(7)
ER	mean absolute error	1	1.5	0.6	6.05	**0.4**	0.9
	mean relative error	0.4	0.53	0.19	2.46	**0.1**	0.33
BA	mean absolute error	1.3	0.8	0.35	4.2	**0.1**	0.55
	mean relative error	0.52	0.28	0.14	1.86	**0.03**	0.2
WS	mean absolute error	1.7	1.7	0.5	6.15	**0.35**	1.15
	mean relative error	0.57	0.57	0.18	2.48	**0.1**	0.4

function, as it was not in its function set, see Table 1. Nevertheless, it is interesting to see that it found formula (6), which is close to the logarithm of the number of nodes in the graphs.

5.1.2 Real-world graphs

For the diameter of real-world graphs, as it is shown in Table 3, the formula (15) was the best by giving very close values to the exact diameter:

$$\frac{2M}{\lambda_1\lambda_2^2} + \frac{\lambda_5^2 + 2(\lambda_N - \lambda_3) + 50}{\lambda_1} + \frac{2}{\lambda_1\lambda_2}$$

Closer inspection reveals that the last term in the formula usually has very small values, below 0.1. The other parts of (15) contribute by roughly equal quantity to the final result. The formula includes the first three, the fifth and the last eigenvalue of the adjacency matrix, as well as the number of edges. Thus, it is a nice demonstration of the surprising power of symbolic regression that it can find non-trivial combination of graph features which can well approximate a graph measure such as diameter. On the other hand, the computational cost is $O(N^3)$ due to the need of calculating the eigenvalues. This means that it has the same cost as directly applying an exact algorithm such as Floyd-Warshall to obtain the diameter.

Table 3: Diameter validations on real-world graphs

network	N	M	D	[20]	(8)	(9)	(10)	(11)	(12)	(13)	(14)	(15)
`ca-netscience`	379	914	17	21	13	9	14	19	4	17	12	10
`bio-celegans`	453	2025	7	7	5	4	8	12	3	8	6	4
`rt-twitter-copen`	761	1029	14	16	13	14	14	19	12	17	20	11
`soc-wiki-vote`	889	2914	13	10	11	8	11	15	7	11	12	6
`ia-email-univ`	1133	5451	8	6	9	9	7	12	9	8	13	10
`ia-fb-messages`	1266	6451	9	7	10	8	8	12	7	9	11	6
`bio-yeast`	1458	1948	19	19	14	28	15	20	18	18	39	18
`socfb-nips-ego`	2888	2981	9	52	14	14	16	23	3	20	21	7
`web-edu`	3031	6474	11	36	14	11	15	22	13	19	16	8
`inf-power`	4941	6594	46	98	14	38	17	24	71	20	53	48
			mean absolute error:	13.3	5.6	4	5.2	6.9	6.2	5.2	6.4	**3.3**
			mean relative error:	0.92	0.28	0.27	0.27	0.53	0.37	0.31	0.48	**0.27**

As we can see, formulas (9) and (10) resulted the same mean relative error than (15), however, they were worse by the mean absolute error. Formula (10) involves some of the eigenvalues of the Laplacian matrix, and some constants. Formula (9) uses some of the eigenvalues of the adjacency matrix, number of nodes and it also uses the number of simplicial nodes. Thus, these formulas, although not giving as precise approximations as (15), are built up by some other graph parameters compared to (15).

Note that in the 5th column of Table 3 we included the results reported in [20] for the same set of graphs. Clearly, all the formulas we found gave smaller errors than the best solution from [20].

5.2 Geodetic number

In order to compare the approximations given by the formulas found by symbolic regression, the computation of the exact geodetic number of the input graphs were needed. For that, we used the integer linear programming formulation proposed in [16].

5.2.1 Random graphs

The results for the geodetic number of random graphs can be seen in Table 4. Formula (16) gave the best approximations for the ER and WS graphs:

$$\sqrt{\frac{N^{3/2}}{\lambda_1}} - \frac{\lambda_{N-4}N^{3/2}}{\lambda_1^2 + N^{3/2}}.$$

In case of BA graphs formula (17) resulted in the lowest error:

$$\frac{\mu_4^2}{\mu_2\mu_{N-3}} + \sqrt{N - \mu_3}$$

Practically, both formulas need the computation of all eigenvalues, thus their computational cost is $O(N^3)$. The exact computation of the geodetic number is NP-hard, whereas formula (16) and (17) can be evaluated in polynomial time.

Note that overall, formula (16) gives the best approximation for all three types of random graphs. Investigating the values one obtains by evaluating formula (16) on random graphs, it turns out that the second part is roughly half of the first part. Thus, a simpler formula would be

$$\frac{3}{2}\sqrt{\frac{N^{3/2}}{\lambda_1}}.$$

Table 4: Geodetic number validations on random graphs

	formula	(16)	(17)	(18)	(19)
ER	mean absolute error	**0.92**	1.31	1	1.07
	mean relative error	**0.1**	0.16	0.16	0.13
BA	mean absolute error	2.15	**1**	1.775	2.92
	mean relative error	0.18	**0.08**	0.17	0.26
WS	mean absolute error	**0.54**	1.38	0.92	0.69
	mean relative error	**0.04**	0.19	0.12	0.08

On average, this gives a bit more pessimistic approximation (namely, mean average error = 1.89, and mean relative error = 0.1). However, it needs the computation of the first dominant eigenvalue only, which costs $O(N^2)$.

5.2.2 Real-world graphs

Table 5 shows the results for the real-world graphs. It is important to emphasize here that since the real-world graphs in Table 5 have hundreds of nodes and thousands of edges, the calculation of the exact geodetic number, using the integer linear programming formulation proposed in [16], requires enormous computational time. For the three largest graphs (`socfb-nips-ego`, `web-edu` and `inf-power`) we were unable to compute the exact geodetic numbers due to time constraints, so they are left out from the comparison.

In this case the best approximation was obtained by the surprisingly compact formula (27):

$$\delta_1 + \sigma + \sqrt{M} - 2.$$

The number of degree-one nodes and the number of simplicial nodes appear in formula (27) because these nodes must be part of the geodetic set, as it was already mentioned in Section 2.3. In fact, these two factors appear in all the best formulas we have found, see Appendix. In the `ca-netscience` collaboration network and in the `bio-celegans` there are lots of simplicial nodes and not many degree-one nodes. For the other graphs it is just the other way around, i.e., the number of simplicial nodes is not more than 10. The remaining part of the geodetic number is approximated by $\sqrt{M} - 2$, which contributes to the approximation on these graphs 1/3 at most. The computational cost of formula (27) is $O(NM)$.

Table 5: Geodetic number validation on real-world graphs.

network	N	M	$g(G)$	(20)	(21)	(22)	(23)	(24)	(25)	(26)	(27)
`ca-netscience`	379	914	253	208	151	190	198	194	206	195	200
`bio-celegans`	453	2025	172	213	115	119	195	188	225	203	146
`rt-twitter-copen`	761	1029	459	436	437	438	439	428	446	442	444
`soc-wiki-vote`	889	2914	275	247	212	220	222	231	247	259	245
`ia-email-univ`	1133	5451	244	225	182	194	181	192	208	196	233
`ia-fb-messages`	1266	6451	318	266	254	264	276	280	296	313	311
`bio-yeast`	1458	1948	784	763	761	766	761	751	775	762	773
		mean absolute error:		32.7	56.1	44.9	39.9	39.0	29.7	28.1	**21.9**
		mean relative error:		0.12	0.21	0.17	0.14	0.13	0.12	0.11	**0.08**

5.2.3 Improvement

We have listed the best formulas and we verified them with specific random and real-world graphs. Our aim is to derive a general formula for the geodetic number that can give good approximation for any real-world graph. For that we wanted to try and make formula (27) even sharper. One of the possible ways is to use linear regression.

For linear regression the generalized formula containing multipliers as variables has the form

$$a \cdot \delta_1 + b \cdot \sigma + c \cdot \sqrt{M} - d$$

The variables were initialized as $a = 1, b = 1, c = 1, d = 1$. The linear regression finds the values of the variables a, b, c and d minimizing the mean absolute error of the approximated value.

As a result, linear regression found that $a = 0.99, b = 0.79, c = 0.97, d = 0.99$, so the formula can be written as

$$0.99 \cdot \delta_1 + 0.79 \cdot \sigma + 0.97 \cdot \sqrt{M} - 0.99. \tag{1}$$

5.2.4 Validation of improved formula

For validating the quality of the formula (1), 120 sub-graphs (where $31 \leq N \leq 485$) from real-world graphs in Table 5 have been used. These graphs were created by the same procedure described in Section 4.2. Then the geodetic number was calculated twice: the exact value by using the ILP formulation [16], and then the approximation using the formula (1) obtained by linear regression. Figure 1 shows a comparison between the two values for the sub-graphs. It is clear that the approximations are close to the exact $g(G)$ values. For all the 120 graphs we obtained mean absolute error = 12.27 and mean relative error = 0.18 by using formula (1). This is just a slight improvement though, since formula (27) gives mean absolute error = 12.37 and the same relative error as (1).

There are two gaps in the figure indicating that for some graphs the approximation is much less than the exact value. For these graphs, the number of simplicial nodes was zero. Since formula (1) is the summation of the number of simplicial

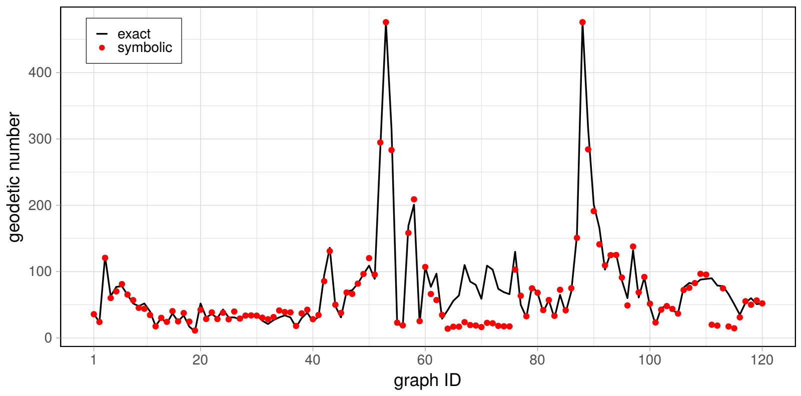


Figure 1: Exact $g(G)$ and values given by the optimized formula (1)

nodes, the number of degree-one nodes, and the number of edges, if one of these values is zero that will cause these gaps. For this type of graphs, where σ and δ_1 are close to zero, it might be more beneficial to use one of the formulas we found for the random graphs. For example, using formula (16) we get mean absolute error = 39.87 and mean relative error = 0.57 for these graphs, while formula (1) on the same graphs gives mean absolute error = 40.87 and mean relative error = 0.6.

6 Conclusion

Our work reports that symbolic regression is successfully applicable to derive optimized formulas for graph geodetic number $g(G)$. The best formula we found is very simple and it can estimate the value of $g(G)$ if the number of edges, the number of degree-one nodes and the number of simplicial nodes are known. Thus, the approximation of the geodetic number can be obtained in a reasonable computational time, even for graphs with thousands of nodes and edges, while obtaining the exact geodetic number is an NP-hard problem. We demonstrated how different training sets will lead to different formulas with different accuracy so that we can claim that finding good training data is essential. Hence, finding the best parameters of training graphs, where these parameters are highly related to the graph property are the most important part for symbolic regression to approximate in a better manner.

Acknowledgements

We thank our reviewers for their critical comments and valuable suggestions that highlighted important details and helped us in improving our paper.

References

[1] Ahangar, H A, Fujie-Okamoto, F, and Samodivkin, V. On the forcing connected geodetic number and the connected geodetic number of a graph. *Ars Combinatoria*, 126:323–335, 2016.

[2] Albert, R and Barabási, A-L. Statistical mechanics of complex networks. *Reviews of Modern Physics*, 74(1):47–97, 2002. DOI: `10.1103/RevModPhys.74.47`.

[3] Anaqreh, A T, G.-Tóth, B, and Vinkó, T. Algorithmic upper bounds for graph geodetic number. *Central European Journal of Operations Research*, 2021. DOI: `10.1007/s10100-021-00760-7`.

[4] Anderson, W and Morley, T. Eigenvalues of the Laplacian of a graph. *Linear and Multilinear Algebra*, 18(2):141–145, 1985. DOI: `10.1080/03081088508817681`.

[5] Atici, M. Computational complexity of geodetic set. *International Journal of Computer Mathematics*, 79(5):587–591, 2002. DOI: 10.1080/00207160210954.

[6] Brešar, B, Klavžar, S, and Horvat, A T. on the geodetic number and related metric sets in cartesian product graphs. *Discrete Mathematics*, 308(23):5555–5561, 2008. DOI: 10.1016/j.disc.2007.10.007.

[7] Brešar, Boštjan, Kovše, Matjaž, and Tepeh, Aleksandra. Geodetic sets in graphs. In *Structural Analysis of Complex Networks*, pages 197–218. Springer, 2011. DOI: 10.1007/978-0-8176-4789-6_8.

[8] Buckley, F and Harary, F. Geodetic games for graphs. *Quaestiones Mathematicae*, 8(4):321–334, 1985. DOI: 10.1080/16073606.1985.9631921.

[9] Chakraborty, D, Foucaud, F, Gahlawat, H, Ghosh, S K, and Roy, B. Hardness and approximation for the geodetic set problem in some graph classes. *Conference on Algorithms and Discrete Applied Mathematics*, 12016:102–115, 2020. DOI: 10.1007/978-3-030-39219-2_9.

[10] Chartrand, G, Harary, F, and Zhang, P. On the geodetic number of a graph. *Networks: An International Journal*, 39(1):1–6, 2002. DOI: 10.1002/net.10007.

[11] Chung, F. Diameters and eigenvalues. *Journal of The American Mathematical Society*, 2(2):187–196, 1989. DOI: 10.1090/S0894-0347-1989-0965008-X.

[12] Ekim, T, Erey, A, Heggernes, P, van 't Hof, P, and Meister, D. Computing minimum geodetic sets of proper interval graphs. *LATIN 2012: Theoretical Informatics*, 7256:279–290, 2012. DOI: 10.1007/978-3-642-29344-3_24.

[13] Erdős, P and Rényi, A. On random graphs I. *Publicationes Mathematicae Debrecen*, 6:290–297, 1959.

[14] Everett, Martin G and Seidman, Stephen B. The hull number of a graph. *Discrete Mathematics*, 57(3):217–223, 1985. DOI: 10.1016/0012-365X(85)90174-8.

[15] Hansberg, A. and Volkmann, L. On the geodetic and geodetic domination numbers of a graph. *Discrete Mathematics*, 310(15):2140–2146, 2010. DOI: 10.1016/j.disc.2010.04.013.

[16] Hansen, P and van Omme, N. On pitfalls in computing the geodetic number of a graph. *Optimization Letters*, 1(3):299–307, 2007. DOI: 10.1007/s11590-006-0032-3.

[17] Harary, F, Loukakis, E, and Tsouros, C. The geodetic number of a graph. *Mathematical and Computer Modeling*, 17(11):89–95, 1993. DOI: 10.1016/0895-7177(93)90259-2.

[18] Harary, F and Nieminen, J. Convexity in graphs. *J. Differential Geom*, 16(2):185–190, 1981. DOI: `10.4310/jdg/1214436096`.

[19] Koza, J R. Genetic programming: On the programming of computers by means of natural selection. *MIT Press, Cambridge, USA*, 1992.

[20] Märtens, M, Kuipers, F, and Van Mieghem, P. Symbolic regression on network properties. *Genetic Programming*, pages 131–146, 2017. DOI: `10.1007/978-3-319-55696-3_9`.

[21] McConaghy, T. Ffx: Fast, scalable, deterministic symbolic regression technology. *Genetic Programming Theory and Practice IX*, pages 235–260, 2011. DOI: `10.1007/978-1-4614-1770-5_13`.

[22] Miller, J. Cartesian genetic programming. *Springer*, 2011. DOI: `10.1007/978-3-642-17310-3`.

[23] Mitchell, S L. Another characterization of the centroid of a tree. *Discrete Mathematics*, 24(3):277–280, 1978. DOI: `10.1016/0012-365X(78)90098-5`.

[24] Peper, Ferdinand. *Efficient network topologies for extensible massively parallel computers*. PhD thesis, TU Delft, 1989. `http://resolver.tudelft.nl/uuid:2e1e7c7e-b8b6-4883-bfcf-2b00d7aa5db8`.

[25] Prakash, Veeraraghavan. An efficient g-centroid location algorithm for cographs. *International Journal of Mathematics and Mathematical Sciences*, 2005(9):1405–1413, 2005. DOI: `10.1155/IJMMS.2005.1405`.

[26] Prakash, Veeraraghavan. Application of g-convexity in mobile ad hoc networks. In *6th International Conference on Information Technology in Asia*, pages 33–38, 2009.

[27] Rossi, R A and Ahmed, N K. An interactive data repository with visual analytics. *SIGKDD Explor*, 17(2):37–41, 2016. DOI: `10.1145/2897350.2897355`.

[28] Schmidt, M and Lipson, H. Distilling free-form natural laws from experimental data. *Science*, 324(5923):81–85, 2009. DOI: `10.1126/science.1165893`.

[29] Schmidt, M and Lipson, H. Solving iterated functions using genetic programming. *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation*, pages 2149–2154, 2009. DOI: `10.1145/1570256.1570292`.

[30] Soloff, J A, Márquez, R A, and Friedler, L M. Products of geodesic graphs and the geodetic number of product. *Discussiones Mathematicae Graph Theory*, 35(1):35–42, 2015. DOI: `10.7151/dmgt.1774`.

[31] Wang, F H, Wang, Y L, and Chang, J M. The lower and upper forcing geodetic numbers of block–cactus graphs. *European Journal of Operational Research*, 175(1):238–245, 2006. DOI: `10.1016/j.ejor.2005.04.026`.

[32] Watts, D J and Strogatz, S H. Collective dynamics of 'small-world' networks. *Nature*, 393:440–442, 1998. DOI: `10.1038/30918`.

Appendix

The best formulas, with respect to mean absolute deviation, found by CGP are listed for the different graph types and graph properties.

Formulas for random graphs diameter. The results obtained with these formulas are shown in Table 2.

$$\sqrt{\frac{N}{\lambda_1}} + \sqrt{\frac{2N}{\lambda_1} - \lambda_3} + 1 \tag{2}$$

$$\frac{N(M + \mu_{N-2})^3(2N^3 + (M + \mu_{N-2})^3(N + \mu_1))}{(N^3 + \mu_1(M + \mu_{N-2})^3)^2} \tag{3}$$

$$\frac{N + 2\sqrt{\lambda_1} - 2\lambda_3}{\lambda_1} \tag{4}$$

$$\frac{N(M + N - \mu_{N-2})}{2(N + \mu_1\mu_4)} \tag{5}$$

$$\frac{N + \lambda_{N-2} + 4}{\sqrt{M}} \tag{6}$$

$$\frac{\mu_{N-2}(3N - \mu_1)}{N\mu_N + \mu_1\mu_{N-2}} \tag{7}$$

Formulas for real-world graphs diameter. The results obtained with these formulas are shown in Table 3.

$$\mu_{N-3} + \mu_{N-4} - 2 + 4(\mu_{N-4} - 2)^2 - \frac{\mu_{N-1}(\mu_{N-3} + \mu_{N-4} + \sigma)}{5} \tag{8}$$

$$\sqrt{\frac{2\lambda_4^3}{(\lambda_1 + \lambda_{N-4})^3} + \frac{(\lambda_1 + N + \sigma)}{(\lambda_1 + \lambda_{N-4})} + \lambda_N} \tag{9}$$

$$\left((\mu_{N-4} - 2)^2 + \frac{(\mu_{N-4}^2 + 1)}{\frac{\mu_2^2}{125} + \mu_5 - (\mu_{N-4} - 2)^2}\right)^2 \tag{10}$$

$$((\sqrt{(3-\sqrt{\mu_{N-4}})})^2((3-\sqrt{\mu_{N-4}})^2-2)) \tag{11}$$

$$\frac{N+2\mu_{N-3}\delta_1+\mu_2+\mu_5+\sigma-2(\mu_{N-4}-5)^3}{\mu_{N-3}\delta_1+\mu_2+\mu_3+\mu_{N-4}+\sigma} \tag{12}$$

$$\mu_{N-3}-2\mu_{N-4}^2-((\mu_{N-4}-1)^3-\sqrt{3})^3 \tag{13}$$

$$\sqrt{2}\sqrt{\lambda_N+\frac{\lambda_1^3}{(\lambda_1+\lambda_{N-4})^3}+\frac{\lambda_1+N+\sigma}{\lambda_1+\lambda_{N-4}}} \tag{14}$$

$$\frac{2M}{\lambda_1\lambda_2^2}+\frac{\lambda_5^2+2(\lambda_N-\lambda_3)+50}{\lambda_1}+\frac{2}{\lambda_1\lambda_2} \tag{15}$$

Formulas for random graphs geodetic number. The results obtained with these formulas are shown in Table 4.

$$\sqrt{\frac{N^{3/2}}{\lambda_1}}-\frac{\lambda_{N-4}N^{3/2}}{\lambda_1^2+N^{3/2}}. \tag{16}$$

$$\frac{\mu_4^2}{\mu_2\mu_{N-3}}+\sqrt{N-\mu_3} \tag{17}$$

$$\frac{N\lambda_4}{3\lambda_1}+\sqrt{5} \tag{18}$$

$$\frac{5(N\mu_4+5\mu_1-5\mu_{N-3})}{\mu_4^2+10\mu_4+25} \tag{19}$$

Formulas for real-world graphs geodetic number. The results obtained with these formulas are shown in Table 5.

$$\delta_1+\sigma+\sqrt{\delta_1}-\lambda_2-\lambda_3\lambda_{N-1}-\lambda_{N-2}\lambda_{N-4}+\frac{N}{\delta_1} \tag{20}$$

$$\frac{\delta_1^2(\delta_1+\sigma)+\delta_1 N}{(\delta_1^2+N)}+\sqrt{\delta_1+\sigma+\lambda_N+\frac{N}{\delta_1}}+\frac{N}{\delta_1}+\frac{N}{\delta_1^2}+1 \tag{21}$$

$$\delta_1 + \sigma + \lambda_{N-4} + \sqrt{N} + \frac{\lambda_N}{\delta_1} + \frac{N(\delta_1 + 1)}{\delta_1^2 + \delta_1 \sigma} \tag{22}$$

$$\delta_1 + \sigma + \sqrt{\delta_1} - \frac{\mu_2 \mu_{N-3}}{\mu_{N-4}} + \mu_2 + \frac{N}{\delta_1} - \sqrt{\frac{\delta_1^2}{\mu_2 N} + \frac{\delta_1 \sigma}{\mu_2 N}} \tag{23}$$

$$\delta_1 + \sigma - \frac{\delta_1}{\sqrt{N}} - \mu_2 \mu_{N-2} + \mu_2 \mu_{N-4} + 2\mu_4 - 2\mu_5 - \mu_{N-2} + \mu_{N-4} + \sqrt{N} \tag{24}$$

$$\delta_1 + \sigma - 2\mu_{N-2} + \sqrt{-2\mu_5 + N} - 8 + \frac{\mu_2}{\mu_{N-2} + 3} + \frac{N}{\delta_1} \tag{25}$$

$$\delta_1 + \sigma + \frac{\mu_1}{\delta_1 + \sigma} - \mu_2 \mu_{N-2} + \mu_4 + 9\mu_{N-2}^2 \mu_{N-3}^2 + \sqrt{\mu_{N-4}}(\mu_2 - \mu_5) \tag{26}$$

$$\delta_1 + \sigma + \sqrt{M} - 2 \tag{27}$$

Quadric Tracing: A Geometric Method for Accelerated Sphere Tracing of Implicit Surfaces[*]

Csaba Bálint[ab] and Mátyás Kiglics[ac]

Abstract

Sphere tracing is a common raytracing technique used for rendering implicit surfaces defined by a signed distance function (SDF). However, these distance functions are often expensive to compute, prohibiting several real-time applications despite recent efforts to accelerate them. This paper presents quadric tracing, a method to precompute an augmented distance field that significantly accelerates rendering. This novel method supports two configurations: (i) accelerating raytracing without losing precision, so the original SDF is still needed; (ii) entirely replacing the SDF and tracing an interpolated surface. Quadric tracing can offer 20% to 100% speedup in rendering static scenes and thereby amortizes the slowdown caused by the complexity of the geometry.

Keywords: computer graphics, sphere tracing, signed distance function

1 Introduction

Most implicit surface representations require conversion to a triangle list for efficient, real-time visualization. Distance-based representations are such an exception where sphere tracing [8] enables raytracing for the whole scene within milliseconds, using the GPU. The representation supports a range of set operations, such as union, intersection, subtraction, and real-time offsets [9]. This allows any CSG tree to be constructed from a large set of primitives, set operations, and transformations [6]; however, the function evaluation time and convergence slow drastically with scene complexity. This paper aims to amortize the scene complexity dependence of raytracing an implicit surface defined by a signed distance function and thereby greatly accelerate render times.

[*]EFOP-3.6.3-VEKOP-16-2017-00001: Talent Management in Autonomous Vehicle Control Technologies — The Project is supported by the Hungarian Government and co-financed by the European Social Fund. Csaba Bálint was also supported by the ÚNKP-20-3 New National Excellence Program of the Ministry for Innovation and Technology.

[a]Eötvös Loránd University, Budapest, Hungary

[b]E-mail: `csabix@inf.elte.hu`, ORCID: `0000-0002-5609-6449`

[c]E-mail: `kiglics@caesar.elte.hu`, ORCID: `0000-0003-4957-7495`

DOI: `10.14232/actacyb.290007`

For this, we generate a cache structure that allows the scene to be traversed with rays more quickly, yet it can slow down near the surface for the utmost accuracy. This structure and the accelerating algorithms must possess the following three properties to improve on existing methods:

- The structure must be concise since three-dimensional data can quickly fill up the available GPU memory.
- Query operations have to be efficient for significant render time reduction during raytracing.
- The preprocessing step should be fast enough to accelerate raytracing, even when called in every frame for dynamic scenes.

Our solution relies on special quadrics, that is, second-degree algebraic surfaces. We define these quadrics pivoted around a given point and an arbitrary direction in space. Each of the enclosed quadratic regions we generate will either contain the entirety of the scene geometry or none of it, so we call these bounding and unbounding quadrics, respectively. Our preprocessing step consists of generating quadric parameters for a 3D grid of values, while the rendering step accelerates raytracing by intersecting the rays with these quadrics first.

This paper focuses on presenting the quadric tracing algorithm and applying it to accelerate sphere tracing utilizing a uniform grid of quadric parameters.

Previous work Since the first appearance of the sphere tracing algorithm [8], several variants of accelerations and enhancements have been released. For clarity, we refer to the sphere tracing method published in [10] as the relaxed sphere tracing algorithm, and we call enhanced sphere tracing the algorithm presented in [3]. The relaxed sphere tracing [10] takes larger steps determined by a scalar, and if the radius is too large, it reverts to the basic sphere tracing size. The more recently published enhanced sphere tracing [3] uses the radii calculated in the two previous steps to efficiently approximate planar surfaces. These methods mainly attempt to increase the size of the steps taken on the ray during rendering; hence, the number of SDF evaluations are minimized. The algorithm applications include light visibility computation such as soft shadows [5], ambient occlusion [18], and global illumination. Various utilizations are also known, like room impulse response estimation [11], or deep learning based implicit signed distance functions [13]. Most of the algorithms above can be improved by quadric tracing.

2 Preliminaries

An $f : \mathbb{R}^3 \to \mathbb{R}$ function is a signed distance function (SDF) if it is continuous and

$$|f(\boldsymbol{p})| = d(\boldsymbol{p}, \{f = 0\}) \quad (\boldsymbol{p} \in \mathbb{R}^3) \ ,$$

as stated in [4, 15]. Where $d(\boldsymbol{p}, \{f = 0\}) = \inf\{\|\boldsymbol{p} - \boldsymbol{x}\|_2 : \boldsymbol{x} \in \{f = 0\}\}$ is the Euclidean distance from the $\{f = 0\} = \{\boldsymbol{x} \in \mathbb{R}^3 : f(\boldsymbol{x}) = 0\}$ surface. The $\{f <$

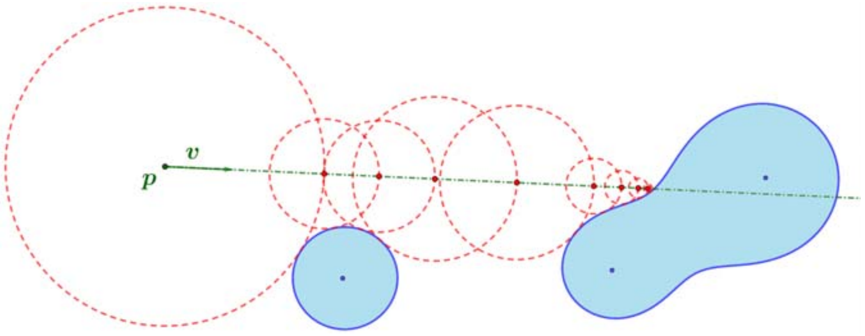


Figure 1: The sphere tracing algorithm takes distance sized steps along the ray. These distance sized steps visualized here in 2D by the red unbounding circles surrounding each signed distance function evaluation point. This image is reused from [4] with the permission of the authors.

$0\} = \{\boldsymbol{p} \in \mathbb{R}^3 : f(\boldsymbol{p}) < 0\}$ region is the inside, while the $\{f > 0\}$ region is considered the outside of the represented geometry. Set operations can be performed on objects defined by SDFs by taking the point-wise minimum or maximum of the operand functions for union or intersections, respectively [8].

Various sphere tracing algorithms exist for surface visualization. These raytracing techniques often start a ray through each pixel of the virtual camera and march along the ray, taking distance sized steps [8]. This is because, for any $\boldsymbol{p} \in \mathbb{R}^3$ point, there are no surface points within $f(\boldsymbol{p})$ distance: this is called the unbounding sphere. Hence, sphere tracing is often visualized as a series of unbounding spheres or circles along a ray as in Figure 1. When the point-to-surface distance becomes negligible, the surface is reached.

However, all sphere tracing algorithms slow down near the surface regardless of the direction taken [10, 3, 5]. The only exception is when the derivative of f is known, and the geometry is convex, in which case huge steps can be taken [8]. This is because, instead of an unbounding sphere, we can draw a separating plane with normal $\nabla f(\boldsymbol{p})$ and surface point $\boldsymbol{p} - f(\boldsymbol{p}) \cdot \nabla f(\boldsymbol{p})$. Note that if $f : \mathbb{R}^3 \to \mathbb{R}$ is an exact signed distance function, then if the derivative exists at $\boldsymbol{p} \in \mathbb{R}^3$, it is of unit length: $\|\nabla f(\boldsymbol{p})\|_2 = 1$. This can often produce long steps, even when the evaluation point is close to the surface. This method can be extended to the union of concave objects, but the average step size will rapidly decrease, and function evaluation time will increase.

Nonetheless, the surface is often concave, or the derivative might be undefined or unknown. For this reason, we generalize the unbounding sphere and separating plane approach to unbounding quadrics [12] for any SDF. Our method consists of two steps. During precomputation, we store distance values in a regular grid. For each cell, the normal vector $\nabla f(\boldsymbol{p})$ is stored along with a $k \in [-1, 1]$ parameter describing the shape of the quadric. Therefore, the quadric is parameterized with a

3D direction vector and a scalar value. During the rendering step, *quadric tracing* intersects the ray with the precomputed quadric to accelerate tracing convergence.

3 Conic sections of $k \in [-1, 1]$

We chose to use the revolution of conic sections for the quadratic proxy surfaces such that a single scalar value $k \in [-1, 1]$ we can encode a large variety of surfaces around the axis defined by $\nabla f(\boldsymbol{p})$. This section details how the 2D conics are constructed to generate both bounding and unbounding regions. Let the y-axis denote the axis of revolution such that the symmetric curve going through the origin has the following form:

$$A(k) \cdot x^2 + B(k) \cdot y^2 + C(k) \cdot y = 0 \qquad (A, B, C : [-1, 1] \to \mathbb{R}). \tag{1}$$

The coefficients are functions of the $k \in [-1, 1]$ parameter, and these control the shape, eccentricity, and curvature of the conic [17]. We found the following three functions to change the conic section in the desired, continuous way from a circle centered at $(0, -0.5)$ to another around $(0, 0.5)$ as k increased from -1 to 1.

$$A := A(k) := k^2 \ , \ B := B(k) := 2\left(|k| - \frac{1}{2}\right) \ , \ C := C(k) := -k \tag{2}$$

This choice allows the conic sections seen on Figure 2 to turn inside out at $k = 0$, and describe both bounded $|k| > \frac{1}{2}$ and unbounded $|k| \leq \frac{1}{2}$ regions visualized. For brevity, we omit the function notation.

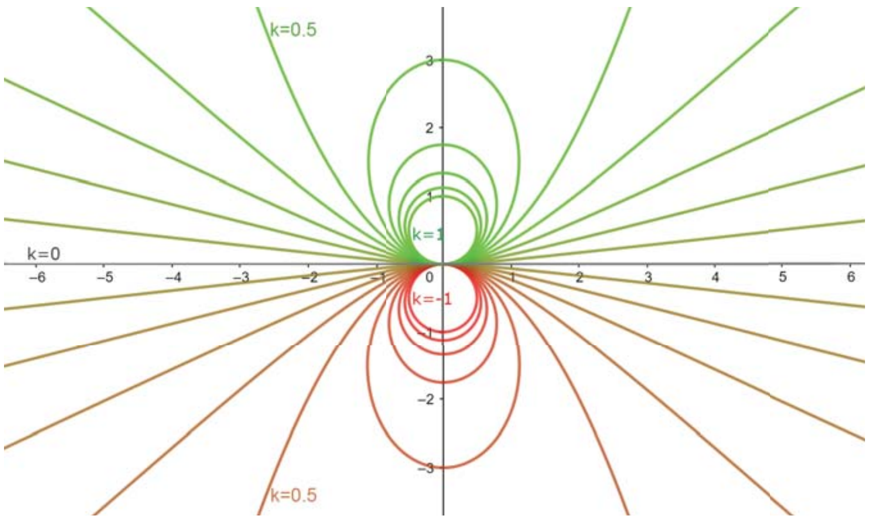


Figure 2: Conic sections with varying $k \in [-1, 1]$. For $k = \pm 1$, the conic section is a circle; for $k = \pm 0.5$, a parabola; for $k \in (0.5, 1)$ or $k \in (-1, -0.5)$, an ellipse; and for $k \in (0, 0.5)$ or $k \in (-0.5, 0)$, it is a branch of a byperbola.

We obtain the polar parametrization of these conics by intersecting them with $t \in [-\pi, \pi)$ angled rays from the origin. The distance from the origin to the intersection point will be a function of this angle, $r(t)$; so by substituting the polar coordinates into the implicit form in (1), we can solve for this $r(t)$:

$$A \cdot (r(t) \cdot \cos t)^2 + B \cdot (r(t) \cdot \sin t)^2 + C \cdot r(t) \sin t = 0 \ .$$

Assuming that $r(t) \neq 0$ yields the polar parametrization $\boldsymbol{s} : [-\pi, \pi) \to \mathbb{R}^2$ of the conic section:

$$r(t) = \frac{-C \cdot \sin t}{A \cdot \cos^2 t + B \cdot \sin^2 t} \quad \Longrightarrow \quad \boldsymbol{s}(t) := \begin{bmatrix} \cos t \cdot r(t) \\ \sin t \cdot r(t) \end{bmatrix} \qquad \big(t \in [-\pi, \pi)\big) \tag{3}$$

For values of $k \in \left(-\frac{1}{2}, \frac{1}{2}\right)$, the $\boldsymbol{s}(t)$ describes a hyperbola with an unwanted branch. Let $L := L(k) \in [-\pi, \pi)$ denote the value where $r(t)$ has a singularity. Thus, we can restrict $\boldsymbol{s}(t)$ to the $[-L, L)$ interval where

$$L := L(k) := \begin{cases} \frac{\pi}{2}, & \text{if } A(k) \cdot B(k) \geq 0, \\ \arctan \sqrt{\frac{-A}{B}}, & \text{otherwise} \end{cases} \qquad (k \in [-1, 1]) \ .$$

Note that for all of the equations above, the heuristic $A(k), B(k)$, and $C(k)$ functions may readily be redefined if needed. We have experimented with several sets of functions. Simplicity, numerical stability, and continuity in terms of k were the deciding factors. Even though the parametric form has a singularity at $k = 0$, the algorithms in this paper are based on the implicit equation, extending to the $k = 0$ line or plane.

4 Unbounding quadrics

We parameterize quadrics of revolution by rotating $\boldsymbol{s}(t) = [\boldsymbol{s}_1(t), \boldsymbol{s}_2(t)]^T$ from (3) around the vertical axis:

$$P(u, v) := \begin{bmatrix} \cos(v) \cdot \boldsymbol{s}_1(u) \\ \sin(v) \cdot \boldsymbol{s}_1(u) \\ \boldsymbol{s}_2(u) \end{bmatrix} = r(u) \cdot \begin{bmatrix} \cos v \cdot \cos u \\ \sin v \cdot \cos u \\ \sin u \end{bmatrix} \qquad (u \in [0, L(k)), v \in [0, 2\pi)).$$

These quadrics can be seen on Figure 3. Similarly, the implicit equation becomes

$$A \cdot (x^2 + y^2) + B \cdot z^2 + C \cdot z = 0.$$

Applying the above, we can calculate the intersection of a ray and the quadric surface. The ray is given by a point $\boldsymbol{p}_0 = (x_0, y_0, z_0) \in \mathbb{R}^3$ and a vector $\boldsymbol{v} = (\boldsymbol{v}_x, \boldsymbol{v}_y, \boldsymbol{v}_z) \in \mathbb{R}^3, \|\boldsymbol{v}\| = 1$, so we can substitute any $\boldsymbol{p}_0 + t \cdot \boldsymbol{v}$ $(t > 0)$ point on the ray and solve the resulting quadratic equation with the coefficients in the $at^2 + bt + c = 0$ equation:

$$\begin{aligned} a &= A \cdot \boldsymbol{v}_x^2 + A \cdot \boldsymbol{v}_y^2 + B \cdot \boldsymbol{v}_z^2 \\ b &= 2A \cdot x_0 \boldsymbol{v}_x + 2A \cdot y_0 \boldsymbol{v}_y + 2B \cdot z_0 \boldsymbol{v}_z + C \boldsymbol{v}_z \\ c &= A \cdot (x_0^2 + y_0^2) + B \cdot z_0^2 + C \cdot z_0 \end{aligned}$$

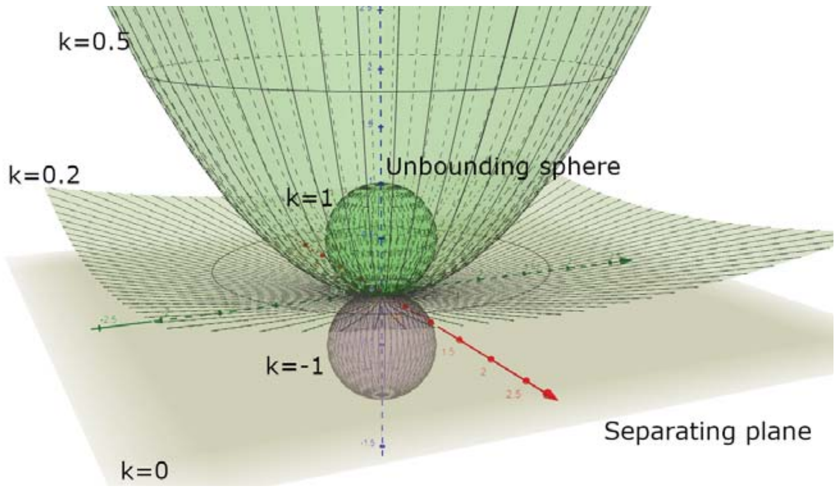


Figure 3: Surfaces of the unbounding quadrics for $k \in [-1, 1]$ separate the space into two regions. The region that contains the $(0, 0, 1)$ point must contain the entirety of the surface once the quadric is transposed and rotated into the scene.[16]

If there are real roots, let them be $t_1 \leq t_2$ and let $I := [t_1, t_2]$. If there are no real roots, let $I := [t_1, t_2] := [-\infty, +\infty]$ [14]. However, if $0 \neq |k| < \frac{1}{2}$, then the solution corresponding to the unwanted branch of the hyperbola needs to be omitted:

$$I := [t_1, t_2] := \begin{cases} [-\infty, t_2] & \text{if } (z_0 + t_1 \boldsymbol{v}_z) \cdot k < 0 \\ [t_1, +\infty] & \text{if } (z_0 + t_2 \boldsymbol{v}_z) \cdot k < 0 \end{cases}$$

If none of the conditions apply, I left as defined before. Then, the first intersection between the ray and the quadric may be computed by evaluating the following four conditional assignments in order:

$$\begin{array}{lll} t := t_2 & \text{if} & t_1 = -\infty \\ t := t_1 & \text{if} & 0 < t_2 < +\infty \\ t := \infty & \text{if} & t_1 < 0 \\ t := 0 & \text{if} & \text{otherwise} \end{array} \tag{4}$$

The resulting $t \geq 0$ is the intersection location along the $\boldsymbol{p}_0 + t \cdot \boldsymbol{v}$ ray.

5 Preprocessing

The accelerator data structure consists of a single grid that stores the distance values and the quadrics of revolutions in four 32 bit floating-point values. We multiply the normalized axis direction $\boldsymbol{n} \in \mathbb{R}^3$ ($\|\boldsymbol{n}\|_2 = 1$) with $2 + k$ to store the quadric using 96 bits while the final 32 bits are reserved for the signed distance function values $d \in \mathbb{R}$. The quadrics are defined relative to each grid point $\boldsymbol{p}_c \in \mathbb{R}^3$ which becomes the origin in Figure 3. The quadric is scaled and rotated such that the axis of revolution points towards is $\boldsymbol{n}$.

During the preprocessing steps, the regular grid is populated with values. The axis we store is the $-\frac{\nabla f(\boldsymbol{p})}{\|\nabla f(\boldsymbol{p})\|}$ gradient vector because it mostly points towards the surface and proves to be a good heuristics. The following four steps detail the computation of the $k \in [-1, 1]$ values. These are executed for every cell in parallel using the GPU:

1. Starting from the origin of the quadric $\boldsymbol{p}_c$, we shoot rays in uniformly sampled directions $\boldsymbol{v}_i$ ($i = 1, \ldots, N$).
2. Each $\boldsymbol{p}_c + t\boldsymbol{v}_i$ ray is then sphere traced until the surface or the bounding box is reached.
3. For each ray-surface intersection $(x_i, y_i, z_i) := \boldsymbol{p}_c + t_i\boldsymbol{v}_i$, we compute the $k_i \in [-1, 1]$ value that defines the unique quadric of revolution that touches the surface at that intersection point.
4. To obtain the unbounding quadric within the cell, we need the narrowest candidate unbounding quadric region from all rays; therefore, $k := \min\{k_i \mid i \in \{1, \ldots, N\}\}$.

The third step warrants more explanation. Note that we only need to operate in the plane defined by $\boldsymbol{v}_i$ and the $\boldsymbol{n}$ normal vector because the quadric is symmetric around $\boldsymbol{n}$. Let $\boldsymbol{q} = (\boldsymbol{q}_x, \boldsymbol{q}_y) \in \mathbb{R}^2$ be the rotated projection into this plane of the (x_i, y_i, z_i) intersection point where $\boldsymbol{n}$ corresponds to the y-axis in 2D. We can solve the 2D implicit equation in (1) of the conic for the k parameter after substituting (2):

$$k^2\boldsymbol{q}_x^2 + 2\left(|k| - \frac{1}{2}\right)\boldsymbol{q}_y^2 - k\boldsymbol{q}_y = 0$$

Depending on the sign of k, the entirety of the conic section lies in the upper or lower half planes, that is: $\operatorname{sgn}\boldsymbol{q}_y = \operatorname{sgn} k$. Substituting $|k| = k \operatorname{sgn} k$, and rearranging yields the quadratic equation

$$\boldsymbol{q}_x^2 \cdot k^2 + (2\operatorname{sgn}\boldsymbol{q}_y \cdot \boldsymbol{q}_y^2 - \boldsymbol{q}_y) \cdot k - \boldsymbol{q}_y^2 = 0 \ .$$

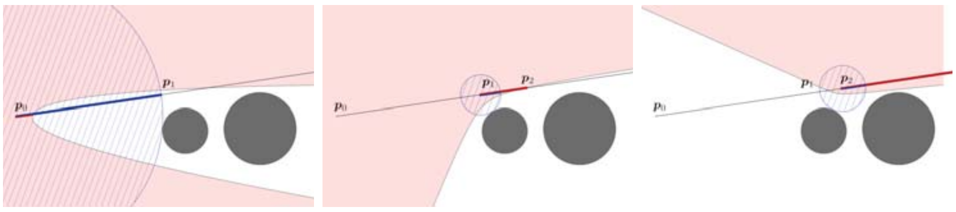
The smaller solution for k is the desired parameter k_i of the quadric in the third step.

6 Quadric tracing

The benefit of quadric tracing is that it takes much larger steps along the ray, as illustrated in Figure 4. The quadrics are stored in grid cells, so for each evaluation point along the ray, the closest quadric is queried and intersected as seen in Section 4. This often yields a step size that skips several grid cells and still does not skip ray surface intersections.

However, sometimes the quadric does not contain the whole cell that it is assigned to, so the acceleration is zero or negligible for some positions. In this case, we take advantage of the stored distance values to advance the iteration by the larger of the two proposed step size. When the computed distance value dips below a certain threshold, our quadric tracing iteration stops. The remaining iterations are then used for the enhanced sphere tracing method [3] that converges quickly close to the surface.

The C++ and OpenGL implementation was developed using the Dragonfly framework [2, 1]. The preprocessing step, the sphere tracing methods, and quadric tracing are implemented on the GPU using compute shaders and appropriate memory barriers. The CSG trees are stored as structures on the CPU, but the shader code for the SDF can be readily generated on-the-fly. Thus, upon the change of parameters or desired algorithms, the implementation generated the SDF and recompiled its shaders to preprocess and render the new surface.



(a) A sphere tracing step was taken because it was larger

(b) Quadric tracing step is now greater then the distance

(c) Quadric step to infinity terminates the raytracing

Figure 4: Consecutive quadric tracing of two circles in 2D. The preprocessing step created the pink unbounding regions to accelerate raytracing whenever the quadric step is larger (b, c). Sometimes, the sphere tracing step is larger because the quadrics are confined to a grid (a). However, the ray does not intersect the quadric in the last case, meaning the algorithm halts in the third iteration.

7 Results

Our implementation featured and compared classic sphere tracing, relaxed sphere tracing [10], enhanced sphere tracing [3], quadric tracing. For the method we call relaxed sphere tracing from [10], the recommended 1.6 step size increase was used. The enhanced sphere tracing had to be slowed down to accommodate smooth concave surfaces, so the recommended 0.95 step size reduction was applied as detailed in [3]. During our experiments, we implemented nine test scenes presented in Figure 5. These test models were also used in [7] for their measurements.

Measuring errors for sphere tracing algorithms can be problematic because there are hit rays that intersect the surface, and there are miss rays that do not. In both cases, most algorithms cannot ever reach the desired value, only converge towards it. Thus, we have a distance-to-surface error tolerance threshold that changes

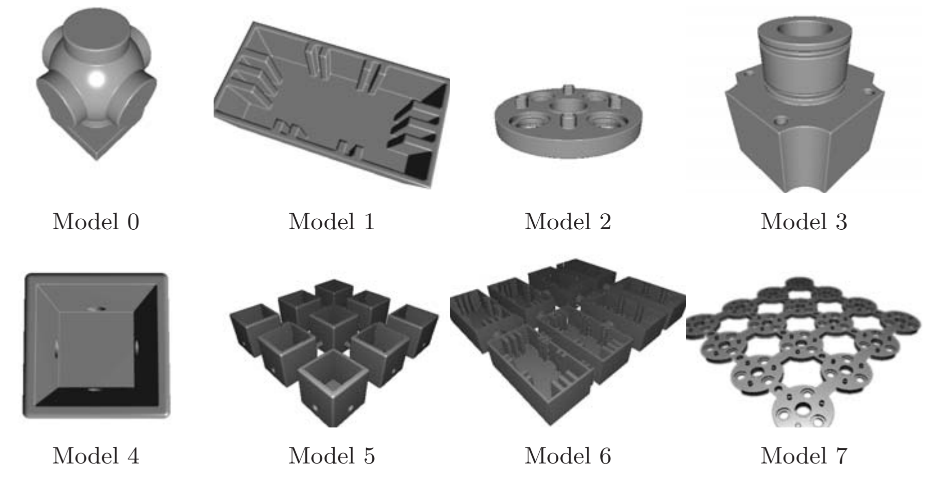


Figure 5: Performance measurements were based on these test scenes from [7] for which we have our own implementation.

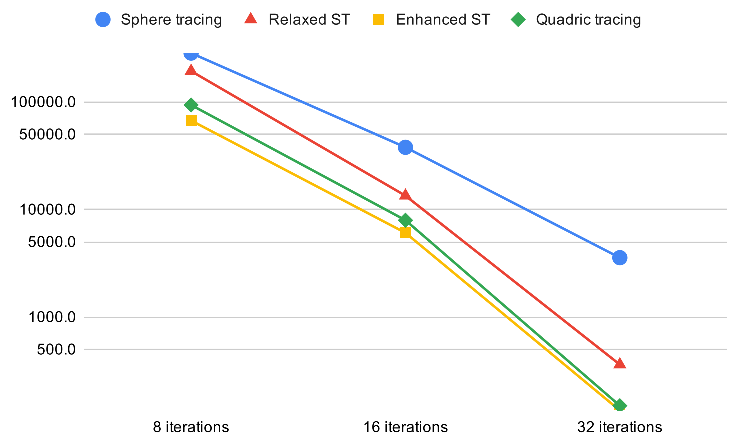
along the ray, effectively, a cone trace. This means that the absolute error can be arbitrarily large and still be acceptable. Also, if the algorithm takes larger steps, it tends to overstep the error threshold by a larger amount making the error smaller compared to another method with a smaller step size.

For our measurements, we generated a $16 \times 16 \times 16$ quadric field by shooting 70×70 rays for each cell. The preprocessing took around three to four times as long as producing a single frame.

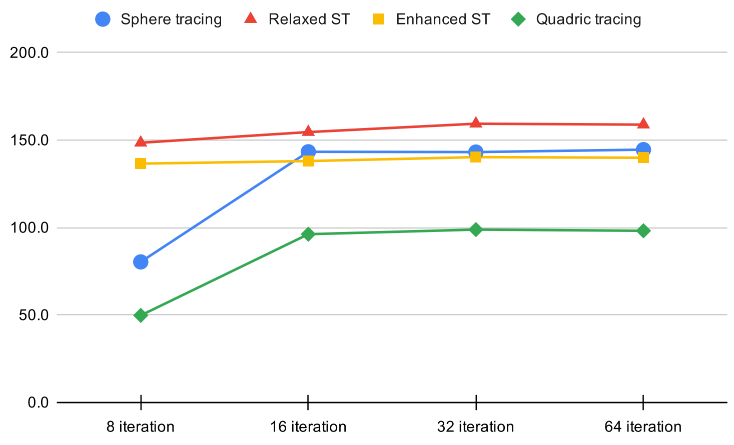
Data for ground truth were produced by sphere tracing the scene for a thousand iterations. This causes the error of relaxed, enhanced, and quadric tracing methods to plateau for high iterations, while the sphere tracing will be exact compared to itself, clearly seen above iteration 64 on Figure 6a.

We made several error metrics, such as the absolute error of divergent and convergent rays, the ratio of rays that did not converge, and the number of SDF evaluations. For the first one, we compared rays that missed or converged to the surface in the ground truth data but were still converging in the test case. Error values could not be aggregated across test scenes easily; thus, we provide in-depth data for the run with 32 iterations on Table 1. Table 1 presents average frame render times and the measured sum of absolute raytracing errors for each model. Quadric tracing performed better for more complex scenes, and such a case is presented in Figure 6. We also plotted the number of rays that did not converge before a given iteration number on Figure 8a.

Our measurements also confirmed the results of [3]: enhanced sphere tracing decreases the error at a higher rate than the relaxed or the original method. Enhanced sphere tracing is most efficient at proximity to the surface, while our quadric tracing accelerates the raytracing through the vast distances between the objects,



(a) Error on a logarithmic scale



(b) Render time in milliseconds

Figure 6: Measured error and render time values as a function of the iteration for Model 7 test scene. Quadric tracing has a slightly higher error for a given iteration count but yields much higher performance across.

as exemplified by Figure 6. The average render time improvement is similar, as demonstrated by Figure 7. Quadric tracing becomes much faster than other methods as the function evaluation time increases because the cost of memory read operations are unaffected by scene complexity. Signed distance function evaluations are decreased by 20% on average when rendering with quadric tracing, as seen in Figure 8a.

Table 1: Relative error and render time comparison of the novel quadric tracing compared to the state of the art methods, each taking 32 iterations. The relative values are divided by the error of the basic sphere tracing, taking the same number of iterations. The enhanced method is slightly more accurate but takes longer to execute for complex scenes.

	Relative error w.r.t. sphere tracing			**Render time in ms**			
	relaxed	enhanced	quadric	basic	relaxed	enhanced	quadric
model 0	0.0945	**0.0362**	0.0488	8	8	**8**	8
model 1	0.2920	**0.1227**	**0.1227**	15	14	**10**	12
model 2	0.1259	0.0347	**0.0327**	18	18	**14**	16
model 3	0.0463	**0.0034**	**0.0034**	13	13	**10**	11
model 4	**0.0003**	**0.0003**	**0.0003**	8	**8**	8	9
model 5	0.1883	**0.0879**	0.0982	71	70	59	**40**
model 6	0.0842	**0.0142**	0.0152	263	276	230	**162**
model 7	0.0574	**0.0101**	0.0137	616	695	626	**363**
model 8	0.0054	**0.0003**	**0.0003**	288	330	293	**262**

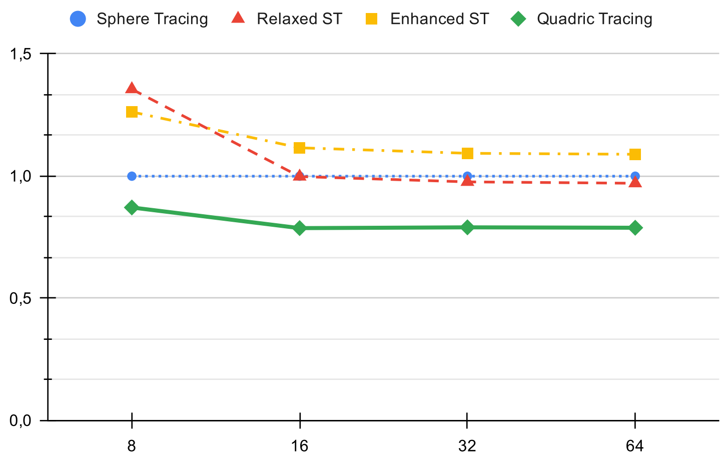
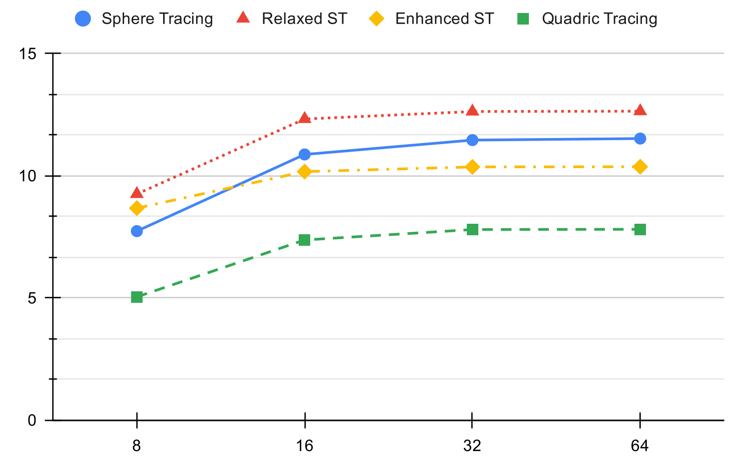
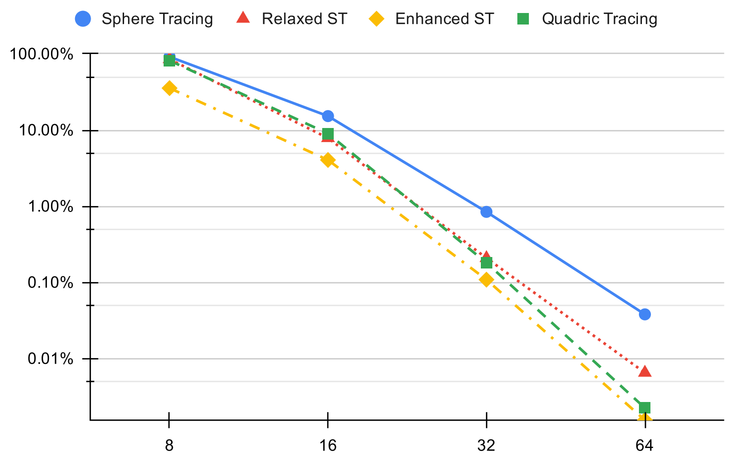


Figure 7: Average relative render time with respect to sphere tracing. Each runtime is divided by that of sphere tracing and then the results are averaged. The time overhead of the enhanced sphere tracing at the end of quadric tracing is included.



(a) Average SDF evaluations per ray



(b) Percentage of rays that did not converge

Figure 8: Number of function evaluations per pixel, and the number of rays that neither missed or hit the surface yet. About 20% of the expensive signed distance function evaluations can be avoided with quadric tracing.

8 Conclusion

This paper presented a novel algorithm to accelerate the raytracing of implicit surfaces by fitting special quadratic volumes from grid points to the surface and raytracing the precomputed proxy geometry instead. Thus, instead of a potentially expensive function evaluation, quadric tracing mostly reads from memory when rendering the scene. Even though our method can visualize the surface from the precomputed data, the acceleration structure is coarse and would seriously limit the resolution and remove surface features. Hence, we opted to continue sphere tracing on the exact signed distance values after our quadric tracing iteration. Therefore, the surface remains unchanged while a significant amount of function evaluations were avoided.

Implementation of the quadric tracing idea necessitated several difficulties to be solved. First, the unbounding volumes had to be compactly stored, efficiently computed, queried, and intersected with, for which revolutions of conic sections were designed with specific coefficient functions. Second, both the parametric and implicit representations were needed in 2D and 3D to write efficient implementations for these on the GPU. This paper focused more on the rendering part rather than the quadric generation because raytracing efficiency determines the competitiveness of quadric tracing. Finally, we implemented all of the presented methods in C++ and OpenGL, utilizing the massive parallelization of the GPU for preprocessing, sphere tracing, and rendering.

Our results show that quadric tracing can efficiently mitigate the slowdown caused by large CSG trees. In such cases, the method is up to two times faster then enhanced sphere tracing [3] whilst being slightly less precise, as shown in Figure 6. On average, quadric tracing was 40% faster than the current state of the art method.

However, quadric tracing requires the accelerator data structure to be computed and populated, taking from a few milliseconds to seconds to complete. Therefore, every time the scene changes, every quadric in every cell needs to be updated, which renders quadric tracing impractical for dynamic scenes. Although intuition suggests otherwise, executing the preprocessing step and the quadric tracing in every frame can be faster than the enhanced method, as our early experiments suggested. However, this was not the focus of this paper, so further research is required with a suitable data structure that can be updated efficiently.

References

[1] Bálint, Csaba and Bán, Róbert. Dragonfly: A C++17 OpenGL framework. 7th Winter School of PhD Students in Informatics and Mathematics, 2020. `http://www.doszmito.hu/wsps7.pdf`.

[2] Bálint, Csaba and Bán, Róbert. Dragonfly: A high level low overhead OpenGL framework. The 11th International Conference on Applied Informatics, 2020.

[3] Bálint, Csaba and Valasek, Gábor. Accelerating Sphere Tracing. In Diamanti, Olga and Vaxman, Amir, editors, *EG 2018 - Short Papers*. The Eurographics Association, 2018. DOI: `10.2312/egs.20181037`.

[4] Bálint, Csaba, Valasek, Gábor, and Gergó, Lajos. Operations on signed distance functions. *Acta Cybernetica*, 24(1):17–28, May 2019. DOI: `10.14232/actacyb.24.1.2019.3`.

[5] Bán, Róbert, Bálint, Csaba, and Valasek, Gábor. Area Lights in Signed Distance Function Scenes. In Cignoni, Paolo and Miguel, Eder, editors, *Eurographics 2019 - Short Papers*. The Eurographics Association, 2019. DOI: `10.2312/egs.20191021`.

[6] Foley, James David. *Constructive Solid Geometry.* In *Computer Graphics: Principles and Practice*, pages 533–558. Addison-Wesley Professional, 1990.

[7] Friedrich, Markus, Roch, Christoph, Feld, Sebastian, Hahn, Carsten, and Fayolle, Pierre-Alain. A flexible pipeline for the optimization of CSG trees. *Computer Science Research Notes*, 3001, 2020. DOI: `10.24132/csrn.2020.3001.10`.

[8] Hart, John. Sphere tracing: A geometric method for the antialiased ray tracing of implicit surfaces. *The Visual Computer*, 12, 1995. DOI: `10.1007/s003710050084`.

[9] Íñigo Quílez. Rendering Worlds with Two Triangles with raytracing on the GPU in 4096 bytes. In NVScene, 2008. `https://uploads.gamedev.net/monthly_2017_07/rwwtt_pdf.dde5c198ccab31d95a41093666ffaad1`.

[10] Keinert, Benjamin, Schäfer, Henry, Korndörfer, Johann, Ganse, Urs, and Stamminger, Marc. Enhanced Sphere Tracing. In Giachetti, Andrea, editor, *Smart Tools and Apps for Graphics - Eurographics Italian Chapter Conference.* The Eurographics Association, 2014. DOI: `10.2312/stag.20141233`.

[11] Lechner, Patrik. Room impulse response estimation using signed distance functions. In *DAFx-2020 - Vienna.* 23rd International Conference on Digital Audio Effects, September 2020.

[12] Levy, Silvio. Geometry formulas and facts. In *30th Edition of CRC Standard Mathematical Tables and Formulas.* CRC Press, 1995.

[13] Liu, Shaohui, Zhang, Yinda, Peng, Songyou, Shi, Boxin, Pollefeys, Marc, and Cui, Zhaopeng. DIST: rendering deep implicit signed distance function with differentiable sphere tracing. *CoRR*, abs/1911.13225, 2019. `http://arxiv.org/abs/1911.13225`.

[14] Oden, J. Tinsley and Demkowicz, Leszek. Linear algebra. In *Applied Functional Analysis (3 ed.).* Chapman and Hall/CRC, 2018.

[15] Osher, Stanley and Fedkiw, Ronald. Signed distance functions. In *Level Set Methods and Dynamic Implicit Surfaces*, pages 17–22. Springer, 2003.

[16] Rodrigues, Olinde. Des lois géometriques qui régissent les déplacements d'un système solide dans l'espace, et de la variation des coordonnées provenant de ces déplacement considérées indépendant des causes qui peuvent les produire. *Journal de Mathématiques Pures et Appliquées*, 5:380–440, 1840.

[17] Thomas, George B. and Finney, Ross L. *Curves in the Plane.* In *Calculus and Analytic Geometry (5th ed.).* Addison-Wesley, 1979.

[18] Wright, Daniel. Dynamic occlusion with signed distance fields. In *Advances in Real-Time Rendering in Games.* Epic Games (Unreal Engine), SIGGRAPH course, 2015.

Acta Cybernetica **25** (2021) 187–203.

Geometric Distance Fields of Plane Curves[*]

Róbert Bán[a] and Gábor Valasek[a]

Abstract

This paper introduces a geometric generalization of signed distance fields for plane curves. We propose to store simplified geometric proxies to the curve at every sample. These proxies are constructed based on the differential geometric quantities of the represented curve and are used for queries such as closest point and distance calculations. We derive the theoretical approximation order of these constructs and provide empirical comparisons between geometric and algebraic distance fields of higher order. We validate our theoretical results by applying them to font representation and rendering.

Keywords: computer graphics, signed distance fields, plane curves

1 Introduction

Signed distance functions (SDF) are special implicit representations of shapes. They map a real number to every point in space and this scalar encodes two attributes of the query position: (i) its distance to the boundary of the geometry represented by the SDF and (ii) whether the query point is inside, outside, or on the boundary of the geometry. The former is the magnitude of the scalar mapped to the point and the latter is determined by its sign.

The construction and evaluation of the exact SDF of a complex scene is computationally expensive. As such, most applications settle on using discrete samples and various reconstruction filtering techniques to infer an approximate signed distance value for every query point in space. We refer to these as discrete signed distance fields (DSDF) and our present work is a generalization of this approach.

In a recent work [3], we considered the algebraic generalization of a signed distance sample. We proposed the use of degree one Taylor approximations to the signed distance function and showed that this allows considerable reductions in storage. That is, even though the size of a single sample increased, the approximation properties of the field itself have improved enough so that in total less scalars were needed to retain a prescribed accuracy.

[*]EFOP-3.6.3-VEKOP-16-2017-00001: Talent Management in Autonomous Vehicle Control Technologies – The Project is supported by the Hungarian Government and co-financed by the European Social Fund.

[a]Eötvös Loránd University, Budapest, Hungary, E-mail: {rob.ban, valasek}@inf.elte.hu, ORCID: 0000-0002-8266-7444, 0000-0002-0007-8647

DOI: 10.14232/actacyb.289248

The generalization of this approach, i.e. increasing the degree of the Taylor approximation is hindered by the coefficient explosion of Taylor polynomials. Since a degree n polynomial in $\mathbb{R}^d$ is represented by $\binom{n+d}{n}$ coefficients, a naive representation of a degree 1 and 2 Taylor polynomial in the plane requires 3 and 6 scalars respectively. Unfortunately, exceeding the per sample storage capabilities of GPU texture formats limits the immediate applicability of texture filtering based approaches, so even degree 2 polynomials need additional techniques to retain their practical value in real-time use cases.

In this paper, we propose an alternative higher order sample construction for planar DSDFs. This technique uses per sample geometric proxies of the boundary curves. These proxies are based on the differential geometric properties of the closest boundary point and they are a generalization of the approach presented in [16].

The intuition comes from recognizing that in the plane, a degree 1 Taylor polynomial is a line in $\mathbb{E}^2$ that also coincides with the tangent line at the closest boundary point to the sample position.

As such, a second order geometric approximation to the boundary is an osculating circle. Clearly, this does not coincide with a second order algebraic sample, whose zero level set determines a conic section in the plane. Moreover, a circle can be represented by its center and radius, i.e. 3 scalars, whereas a degree two polynomial in two variables is determined by 6 scalar coefficients.

Our main theoretical contribution is that this storage reduction does not cost us approximation power: the signed distance function of the osculating circle is a similarly second order approximation to the signed distance function of the original geometry as a second degree Taylor polynomial. This is proven in Section 6.

More generally, we show that entities possessing an order n geometric contact have equal SDF derivatives up to order n.

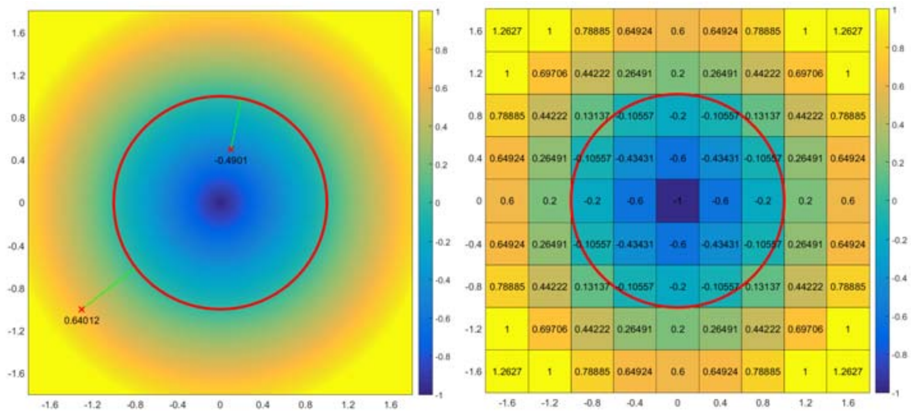
We validate our theoretical results by applying this representation to the storage and rendering of vector fonts in Section 9.

2 Prior work

Discrete signed distance field constructions have found uses in many applications, including font rendering [7], collision detection [6], and various other areas [11, 17, 5, 1].

Our focus is in the planar use cases of DSDFs, and more specifically, font and vector art rendering. DSDF based techniques have received much attention in this venue and they have been improved both in terms of performance and quality.

A notable work is that of Loop and Blinn [9], that takes GPU architecture specific considerations into account and contributes in both areas of improvements. They propose a general framework for the rendering of vector art composed of up to cubic Bézier curves. For anti-aliasing, they approximate the signed distance to the boundary using a first order expansion and show how these data can be stored at the primitive vertices such that hardware bilinear interpolation yields correct



	-1.6	-1.2	-0.8	-0.4	0	0.4	0.8	1.2	1.6
1.6	1.2627	1	0.78885	0.64924	0.6	0.64924	0.78885	1	1.2627
1.2	1	0.69706	0.44222	0.26491	0.2	0.26491	0.44222	0.69706	1
0.8	0.78885	0.44222	0.13137	-0.10557	-0.2	-0.10557	0.13137	0.44222	0.78885
0.4	0.64924	0.26491	-0.10557	-0.43431	-0.6	-0.43431	-0.10557	0.26491	0.64924
0	0.6	0.2	-0.2	-0.6	-1	-0.6	-0.2	0.2	0.6
-0.4	0.64924	0.26491	-0.10557	-0.43431	-0.6	-0.43431	-0.10557	0.26491	0.64924
-0.8	0.78885	0.44222	0.13137	-0.10557	-0.2	-0.10557	0.13137	0.44222	0.78885
-1.2	1	0.69706	0.44222	0.26491	0.2	0.26491	0.44222	0.69706	1
-1.6	1.2627	1	0.78885	0.64924	0.6	0.64924	0.78885	1	1.2627

Figure 1: The signed distance function of the unit circle (left) and a signed distance field of the same geometry (right).

approximations.

A particular difficulty with DSDF approaches is that they cannot deal with hard corners as bilinear interpolation tends to smooth them out. More general approaches, such as edge-aware sampling [2] or feature based textures [13] can improve on this, but there are font-specific solutions to this problem as well. For example [10] replaces the Bézier segments by linear and circular arcs that also improves performance. This latter approach is the most similar work to ours but our argument is based on the functional approximation of the SDF and the local differential geometry of the boundary.

Recently, results have been published about a more accurate anti-aliasing of vector based fonts in 3D scenes [4].

3 Theoretical background

Notation We denote the n-dimensional Euclidean space by $\mathbb{E}^n$ and $\|\cdot\|_2$ is the Euclidean norm. The partial derivatives of an $f : \mathbb{E}^2 \to \mathbb{R}$ function are $\partial_1 f, \partial_2 f$ or f_x, f_y. The scalar product of vectors $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{R}^n$ is written as $\langle \boldsymbol{a}, \boldsymbol{b} \rangle = \boldsymbol{a}^T \boldsymbol{b}$.

Definition 1. *Let $\boldsymbol{\alpha} = (\alpha_1, \dots, \alpha_n) \in \mathbb{N}^n$ be a multi-index. Then we define the following operations:*

- $|\boldsymbol{\alpha}| = \alpha_1 + \alpha_2 + \cdots + \alpha_n$
- $\boldsymbol{\alpha}! = \alpha_1! \cdot \alpha_2! \cdot \ldots \cdot \alpha_n!$, *where* $0! = 1$
- $\boldsymbol{x}^{\boldsymbol{\alpha}} = x_1^{\alpha_1} \cdot x_2^{\alpha_2} \cdot \ldots \cdot x_n^{\alpha_n} \qquad (\boldsymbol{x} = (x_1, x_2, \dots, x_n) \in \mathbb{E}^n)$
- $\partial^{\boldsymbol{\alpha}} f = \partial_1^{\alpha_1} \partial_2^{\alpha_2} \dots \partial_n^{\alpha_n} f \qquad (f : \mathbb{E}^n \to \mathbb{R})$

Definition 2 (Regular curves). *An $\boldsymbol{r} \in \mathbb{R} \to \mathbb{E}^n$ parametric curve is regular iff $\forall t \in \mathcal{D}_{\boldsymbol{r}} : \boldsymbol{r}'(t) \neq \boldsymbol{0}$.*

Definition 3 (Natural parametrization). *$\hat{\boldsymbol{r}} \in \mathbb{R} \to \mathbb{E}^n$ is the natural or arc-length parametrization of $\boldsymbol{r} \in \mathbb{R} \to \mathbb{E}^n$, which means $\forall s \in \mathcal{D}_{\hat{\boldsymbol{r}}} : \|\hat{\boldsymbol{r}}'(s)\|_2 = 1$.*

Definition 4 (Arc-length function). *$s : [a,b] \to [0,L]$ is called the arc-length function of $\boldsymbol{r} : [a,b] \to \mathbb{E}^n$, if $s(t) = \int_a^t \|\boldsymbol{r}'(x)\|_2 \, dx$ and $L = \int_a^b \|\boldsymbol{r}'(x)\|_2 \, dx$.*

Corollary 1 (Natural parametrization). *Let $\boldsymbol{r} : [a,b] \to \mathbb{E}^n$ be a regular parametric curve, then $\boldsymbol{r} = \hat{\boldsymbol{r}} \circ s$ and $\hat{\boldsymbol{r}} = \boldsymbol{r} \circ s^{-1}$.*

Definition 5 (G^n continuity). *Two curves are $G^n, n \geq 1$ at a common point $\boldsymbol{x}$ iff there exits a regular parametrization with respect to which they are C^n at $\boldsymbol{x}$.*

Definition 6 (Signed distance function). *The signed distance function $f : \mathbb{E}^2 \to \mathbb{R}$ of an $F \subset \mathbb{E}^2$ two dimensional shape is defined as $f(\boldsymbol{x}) = \operatorname{sgn}(\boldsymbol{x}) \cdot d(\boldsymbol{x}, \partial F)$, where $\partial F := \overline{F} \cap \overline{(\mathbb{E}^2 \setminus F)}$ is the boundary of F, $d(\boldsymbol{x}, G) = \inf_{\boldsymbol{y} \in G} \|\boldsymbol{x} - \boldsymbol{y}\|_2$ and $\operatorname{sgn}(\boldsymbol{x})$ determines if $\boldsymbol{x}$ is inside or outside of F:*

$$\operatorname{sgn}(\boldsymbol{x}) = \begin{cases} -1 & \text{if } \boldsymbol{x} \in F, \\ 1 & \text{if } \boldsymbol{x} \notin F. \end{cases}$$

Definition 7 (Footpoint parameter relation). *Let $\boldsymbol{p} : [a,b] \to \mathbb{E}^2$ be a regular parametric curve. Then $r \subset \mathbb{E}^2 \times [a,b]$ contains all $(\boldsymbol{x}, t)$ pairs, where t is a parameter for a closest point on the curve for $\boldsymbol{x}$:*

$$r = \left\{ (\boldsymbol{x}, t) \in E^2 \times [a,b] \;\middle|\; \|\boldsymbol{p}(t) - \boldsymbol{x}\|_2 = \min_{u \in [a,b]} \|\boldsymbol{p}(u) - \boldsymbol{x}\|_2 \right\}$$

The above relation is usually almost a function. The points $\boldsymbol{x}$ that have multiple parameters associated with are the ones where there is no unique closest point, such as the center of a circle. These points lie on the cut locus of the curve.

Definition 8 (Footpoint parameter mapping). *A $t : \mathbb{E}^2 \to [a,b]$ footpoint parameter mapping of a regular parametric curve is any narrowing of the footpoint parameter relation of the curve, so that it is a function on the whole domain: $t \subset r : \forall \boldsymbol{x} \in \mathbb{E}^2 \; |\{u \in [a,b] : (\boldsymbol{x}, u) \in t\}| = 1$.*

4 Algebraic SDFs

Algebraic signed distance fields are based on the Taylor approximation theorem [3]. Instead of simple function values, we store higher order Taylor polynomial approximations to the shape in the samples.

Definition 9 (Taylor polynomials). *Let $f : \mathbb{R}^n \to \mathbb{R}, \boldsymbol{a} \in D_f \subset \mathbb{R}^n, f \in C^k[\boldsymbol{a}]$. The order k Taylor polynomial of f around $\boldsymbol{a}$ is*

$$T_f^{(k)}(\boldsymbol{x}) = \sum_{|\boldsymbol{\alpha}| \leq k} \frac{\partial^{\boldsymbol{\alpha}} f(\boldsymbol{a})}{\boldsymbol{\alpha}!} (\boldsymbol{x} - \boldsymbol{a})^{\boldsymbol{\alpha}} \qquad (\boldsymbol{x} \in \mathbb{R}^n)$$

Taylor polynomials have a good local approximation property characterized as

Theorem 1 (Taylor Approximation Theorem)**.** *Let $f : \mathbb{R}^n \to \mathbb{R}, S \subset \mathbb{R}^n$ open and convex, $f \in C^{k+1}[S]$. If $\boldsymbol{a}, \boldsymbol{a}+\boldsymbol{h} \in S$, then*

$$f(\boldsymbol{a}+\boldsymbol{h}) = T_f^{(k)}(\boldsymbol{a}+\boldsymbol{h}) + R_{\boldsymbol{a},k}(\boldsymbol{h})$$

where the residual $R_{\boldsymbol{a},k}$ can be expressed using an adequate $c \in (0,1)$:

$$R_{\boldsymbol{a},k}(\boldsymbol{h}) = \sum_{|\boldsymbol{\alpha}|=k+1} \partial^{\boldsymbol{\alpha}} f(\boldsymbol{a}+c\cdot\boldsymbol{h}) \frac{\boldsymbol{h}^{\boldsymbol{\alpha}}}{\boldsymbol{\alpha}!}$$

or with an integral form:

$$R_{\boldsymbol{a},k}(\boldsymbol{h}) = (k+1) \sum_{|\boldsymbol{\alpha}|=k+1} \frac{\boldsymbol{h}^{\boldsymbol{\alpha}}}{\boldsymbol{\alpha}!} \int_0^1 (1-t)^k \partial^{\boldsymbol{\alpha}} f(\boldsymbol{a}+t\boldsymbol{h}) dt \ .$$

In the two dimensional case, the Taylor polynomials are written in the following way. Let $\boldsymbol{x} = [x,y]^T, \boldsymbol{a} = [a,b]^T \in \mathbb{E}^2$, then

$$T_f^{(k)}(x,y) = \sum_{i=0}^{k} \sum_{j=0}^{i} \frac{\partial_1^j \partial_2^{i-j} f(a,b)}{j!(i-j)!} (x-a)^j (y-b)^{i-j}$$

In algebraic signed distance fields, these local approximations are used as samples. In the simplest construction, all of the samples share a common polynomial degree. This degree is referred to as the order of the algebraic distance field, since it is the approximation order of the stored polynomials.

An order 1 algebraic distance field uses the distance function value and the first partial derivatives. In the local polynomial basis, it is written as

$$T_f^{(1)}(\boldsymbol{x}) = f(\boldsymbol{a}) + f_x(\boldsymbol{a})(x-a) + f_y(\boldsymbol{a})(y-b)$$

Local refers to that the basis functions $x-a$, $y-b$ are relative to the sample position $\boldsymbol{a} = [a,b]^T$, and the global basis would be independent of it. Similarly, an order 2 algebraic distance field sample is derived from the distance function value, the first and the second order derivatives:

$$\begin{aligned} T_f^{(2)}(\boldsymbol{x}) =& f(\boldsymbol{a}) + f_x(\boldsymbol{a})(x-a) + f_y(\boldsymbol{a})(y-b) + \\ & \frac{1}{2} f_{xx}(\boldsymbol{a})(x-a)^2 + f_{xy}(\boldsymbol{a})(x-a)(y-b) + \frac{1}{2} f_{yy}(\boldsymbol{a})(y-b)^2 \end{aligned}$$

Note that the classical signed distance field coincides with the order 0 algebraic signed distance field, as the degree 0 polynomial approximation of a function is a constant function value, i.e. $T_f^{(0)}(\boldsymbol{x}) \equiv f(\boldsymbol{a})$. This way, the algebraic signed distance fields can be viewed as a generalizations of the classical one.

Also note that an algebraic signed distance field sample is not necessary represented directly by the constant function value and the derivatives; see Section 5.

5 Representation of algebraic samples

The samples in an algebraic signed distance field are polynomials. The order zero case coincides with traditional signed distance fields, which store signed distance values. Polynomials are usually represented with their power basis coefficients, but any basis of representation can be chosen. The two main options are local and global basis.

By local polynomial basis we mean that the basis is relative to the sample position. In a global basis all of the basis functions are given in the same coordinate system, i.e. same axes and origin. For a single sample, there is no real difference between the two options. The practical difference appears when multiple samples interact in some way. This is the case when querying SDFs, which is usually done using hardware accelerated linear blending. We show that the global basis is invariant under affine combinations, which helps leveraging this GPU texture filtering.

To prove this, let $\lambda_i(\boldsymbol{x})$ be an arbitrary collection of barycentric weight functions, i.e. $\lambda_i : \mathbb{E}^n \to \mathbb{R}$ such that $\forall \boldsymbol{x} \in \mathbb{E}^n : \sum_i \lambda_i(\boldsymbol{x}) = 1$. Let $b_j(\boldsymbol{x})$ be a polynomial basis and $P_i(\boldsymbol{x}) = \sum_j a_{ij} b_j(\boldsymbol{x})$ arbitrary polynomials. Let us now consider the barycentric combination of these polynomials as

$$\begin{aligned}\sum_i \lambda_i(\boldsymbol{x}) P_i(\boldsymbol{x}) &= \sum_i \lambda_i(\boldsymbol{x}) \left(\sum_j a_{ij} b_j(\boldsymbol{x}) \right) = \\ &= \sum_i \sum_j \lambda_i(\boldsymbol{x}) a_{ij} b_j(\boldsymbol{x}) \\ &= \sum_j \underbrace{\left(\sum_i \lambda_i(\boldsymbol{x}) a_{ij} \right)}_{\hat{a}_j(\boldsymbol{x})} b_j(\boldsymbol{x}) \\ &= \sum_j \hat{a}_j(\boldsymbol{x}) b_j(\boldsymbol{x})\end{aligned}$$

This means that if the polynomials are all stored in the same global basis, we can first interpolate the coefficients of the polynomials and then evaluate one polynomial. If the texture stores polynomial coefficients, the GPU bilinear interpolation returns $\hat{a}_j(\boldsymbol{x})$. This implicitly uses a linear λ_i weighting function, but we can achieve different non-linear λ_i weights by modifying the texture lookup coordinates.

6 Geometric interpretation

The main issue with the algebraic approach is that the number of coefficients grow quadratically with the order. For order n, the number of coefficient is $\binom{n+2}{n} = \frac{(n+2)(n+1)}{2}$. Thus a second order algebraic sample consists of six coefficients which already do not fit into a single texture sample. This motivated us to investigate

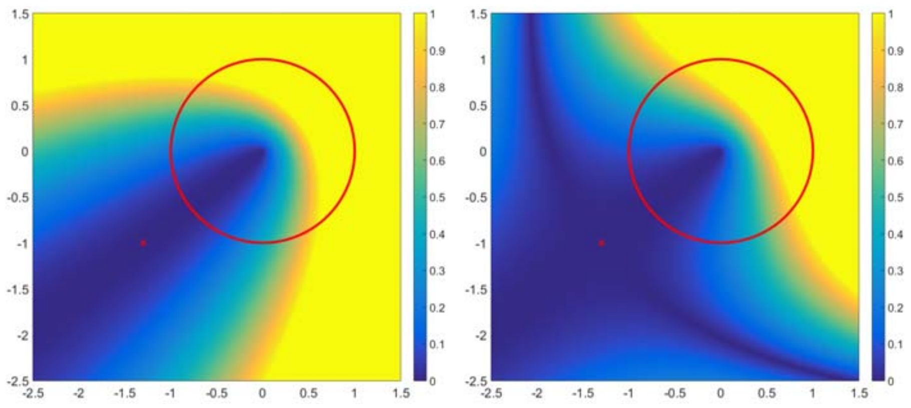


Figure 2: Error heat map of the signed distance function inferred from a first and a second order algebraic sample – Taylor polynomial – approximating the SDF of the unit circle. The sample position is the red x and the geometry is the red circle.

alternative approximations with a lower scalar footprint that still retain a proven approximation order.

The geometric interpretation of the order 0 algebraic sample is already established in the literature: a value $d = f(\boldsymbol{x})$ sampled from the distance function is the signed distance from the closest point to the curve (or surface in 3D) and defines an unbounding circle (sphere) around the sample position with a radius of $|d|$. This unbounding volume contains no surface point.

The first order algebraic sample is a linear polynomial with three coefficients, which naturally describes a line with its zero level set. However, in our case, this line has a geometric interpretation: this line is the tangent line of the curve at the closest point to the sample point. Moreover, the polynomial is the exact signed distance function of the represented line, since the gradient of the function is unit length and constant for the entire domain.

A second order algebraic sample is a second order polynomial with six coefficients, which means that the level sets are quadrics or, in special cases, degenerate quadrics. However the polynomial here – unlike in the order one case – is not the signed distance function of the zero level set, but only a local approximation to it. The nearest point on the curve is usually part of the zero level set of the second degree polynomial and the close neighbourhood matches the approximated curve.

These geometric interpretations open a way to using geometric objects as approximations. To see what kind of geometries could replace the algebraic samples, we show that two curves connecting with G^n continuity have a C^n continuous signed distance function around the connection. For this let us first prove

Theorem 2. *The signed distance function of any regular parametric curve is independent of the parametrization of the curve.*

Proof. Let us start with an arbitrary parametrization, and show that the signed distance function is the same as for the natural parametrization. Let $\boldsymbol{p} : [a, b] \to \mathbb{E}^2$ be a regular parametric curve, and consider one of its footpoint parameter mappings as defined in Definition 8. A closest point function $\boldsymbol{p}^* : \mathbb{E}^2 \to \mathbb{E}^2$ is then $\boldsymbol{p}^* = \boldsymbol{p} \circ t$. This maps all points in the plane to the closest point – or one of the closest points – on the curve. With the help of the closest point function, we can define the (unsigned) distance function $\tilde{f} : \mathbb{E}^2 \to \mathbb{R}_0^+$ of the curve as $\tilde{f}(\boldsymbol{x}) = \|\boldsymbol{p}^*(\boldsymbol{x}) - \boldsymbol{x}\|_2$. For the signed distance function, we also need to partition the plane into inside and outside sets. Let $F \subset \mathbb{E}^2$ be the inside of our shape (bounded by $\boldsymbol{p}$), then the signed distance function $f : \mathbb{E}^2 \to \mathbb{R}$ is

$$f(\boldsymbol{x}) = \operatorname{sgn}(\boldsymbol{x}) \cdot \tilde{f}(\boldsymbol{x}) \qquad (\boldsymbol{x} \in \mathbb{E}^2).$$

Now let $\hat{\boldsymbol{p}} : [0, L] \to \mathbb{E}^2$ be the natural paramteriztaion of $\boldsymbol{p}$. As we have noted in Corollary 1, the arc-length function $s : [a, b] \to [0, L]$ links $\boldsymbol{p}$ to the natural parametrization: $\boldsymbol{p} = \hat{\boldsymbol{p}} \circ s$. Similarly, the footpoint parameter mapping $\hat{t} : \mathbb{E}^2 \to [0, L]$ of $\hat{\boldsymbol{p}}$ is related to t through s: $t = s^{-1} \circ \hat{t}$. These show that $\boldsymbol{p}^*$ is independent of parametrization and thus the signed distance function as well since

$$\begin{aligned}\boldsymbol{p}^* &= \boldsymbol{p} \circ t \\ &= \hat{\boldsymbol{p}} \circ s \circ s^{-1} \circ \hat{t} \\ &= \hat{\boldsymbol{p}} \circ \hat{t}\end{aligned}$$

□

Another interpretation of Theorem 2 is that the signed distance function is invariant under regular reparametrizations of the represented curve. For example, the first partial derivatives with respect to the X and Y coordinate axes are

$$\begin{aligned}\partial_k f(\boldsymbol{x}) &= \partial_k \left(\operatorname{sgn}(\boldsymbol{x}) \cdot \|\boldsymbol{p}^*(\boldsymbol{x}) - \boldsymbol{x}\|_2\right) \\ &= \operatorname{sgn}(\boldsymbol{x}) \cdot \frac{(\boldsymbol{p}^*(\boldsymbol{x}) - \boldsymbol{x})^T}{||\boldsymbol{p}^*(\boldsymbol{x}) - \boldsymbol{x}||_2} \cdot \left(\partial_k \boldsymbol{p}^*(\boldsymbol{x}) - \partial_k \boldsymbol{x}\right) \qquad (k \in \{1, 2\})\end{aligned}$$

Corollary 2. *If two curve segments connect G^n continuously, then the distance fields of the curves are C^n continuous at the connection point.*

This is true because G^n continuity is equivalent to the two connecting curves having the same natural parametrization [15].

Remark 1. This continuity extends to the line segment starting from the foot point through the sample, until the normal line intersects the cut locus. See Figure 3 for an example.

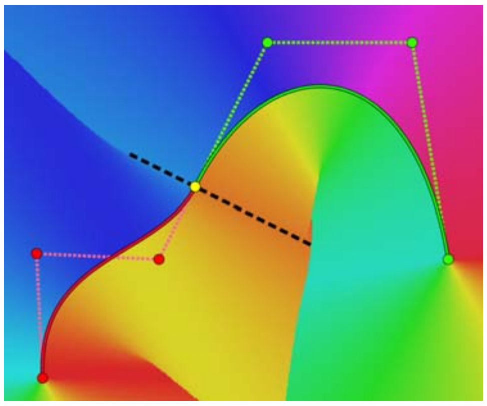


Figure 3: The red and green cubic Bezier-segments connect G^1 continuously at the shared yellow point. The background color encodes the gradient vector of the unsigned distance function of the curves. The black dashed line partitions the plane by which curve segment provides the foot point. As we note in Remark 1, the gradient is continuous on this line segment.

7 Geometric proxies

We have seen in Corollary 2 that G^n continuous curves have C^n continuous signed distance fields outside the cut locus. As a consequence, curves with proper connectivity properties can be used to approximate the signed distance field of a shape instead of algebraic polynomials. We have a few expectations towards the geometric proxies. (i) The proxy has to be a theoretically proven approximation to the original SDF. This is fulfilled by any G^n continuously connected curve at the footpoint of the desired order. (ii) It has to encode the inside/outside partitioning. This will give us the sign of the distance field, which is a binary information but it can be often encoded naturally into the representation of the proxy. (iii) The SDF of the proxy should be easy to evaluate. It is important since it will be used several times when the distance field is sampled. (iv) It should have a low memory cost, as we want a representation that is at least as memory efficient as the algebraic distance fields are.

The order zero geometric proxy is the footpoint itself. In practice, this construction is suboptimal, as the classical signed distance field uses a single scalar value as a sample, but a 2D point is represented by two scalars and the sign needs additional handling.

In first order, the simplest curve is the line which is in our case the tangent line of the approximated curve at the footpoint. Note that this is equivalent to the algebraic first order sample, however in the next chapter we will show a more efficient encoding. As seen with the polynomial encoding, the sign is naturally present in the linear function and the inside/outside partitioning is trivially inferred from a sample. For the tangent line to exist, the curve must be G^1 continuous at the footpoint. This means that the direction of the gradient of the curve changes continuously. The length of the gradient, however, is allowed to have discontinuities.

The second order geometric proxy is the circle. The approximating circle is again touching the curve at the footpoint and in this case it will be the osculating circle. This differs from the algebraic order two representation, so a comparison is necessary. The inside of the circle represents our 2D shape if the shape is locally convex and the outside if the shape is locally concave. The criterion for the existence of the osculating circle is that the curve is G^2 continuous. This means, that it is G^1 continuous and the osculating circles change continuously. It is important to note that the representation has to fall back to the first order case if the curvature is zero, since that would mean an infinitely large circle, that is a straight line, that are very common in modelling and especially vector art.

Geometric distance fields are sampled similarly to algebraic DSDFs, but this time we cannot rely on automatic GPU-interpolation, because the stored geometric data cannot be interpolated trivially into a new correct proxy geometry. First we take the proxies at the corners of the sampled cell, then calculate the distance from all four proxies and finally, bilinearly interpolate the received values.

8 Encoding of geometric samples

8.1 First order geometric sample

The first order geometric sample is a half plane – or an oriented line. We have already seen that the first order algebraic sample describes the same line with a first order polynomial.

With a geometric approach, we could store the footpoint and the direction vector or normal vector of the line. If we save these, the footpoint would consume two scalars and the line direction/normal one additional scalar. However, note that the tangent line is always perpendicular to the footpoint vector, i.e the vector starting at the sample point and pointing to the footpoint. Using this fact, we can discard the direction component and only store the footpoint.

To be able to differentiate between the inside and outside, we propose to use a modified polar coordinate system. The change is that the radial coordinate is allowed to have a sign, encoding the insideness of the sample point. For consistency, the angular coordinate has to be rotated 180 degrees if the radial coordinate is negative.

A first order geometric proxy is then encoded by a pair of signed distance and modified polar angle values $(d, \theta) \in \mathbb{R} \times [0, 2\pi)$, and the curve normal $\boldsymbol{n} \in \mathbb{R}^2$ at the footpoint and the footpoint $\boldsymbol{t} \in \mathbb{E}^3$ are computed upon query as

$$\boldsymbol{n} = \begin{bmatrix} \cos(\theta) \\ \sin(\theta) \end{bmatrix} \tag{1}$$

$$\boldsymbol{t} = \boldsymbol{x} - d \cdot \boldsymbol{n} \tag{2}$$

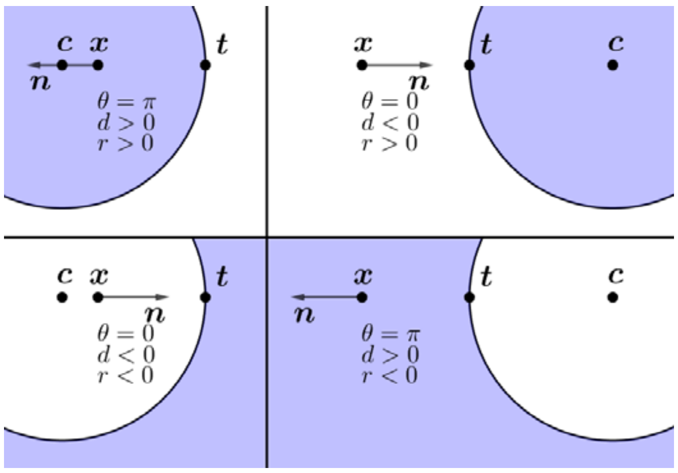


Figure 4: An order 2 geometric sample (d, θ, r) can represent four circles, depending on the signs of d and r. The figure illustrates these four corresponding to the same $\boldsymbol{x}$ sample position and $\boldsymbol{t}$ footpoint. The inside of the shape is in blue, the outside is in white.

Note that this representation is an orthogonal extension of the ordinary signed distance field, as the first coordinate is simply the signed distance, and the second, angular coordinate is independent of it.

8.2 Second order geometric sample

The second order geometric proxy is the osculating circle and there are various options for the representation of a circle.

The three parameters representing the circle proxies are $(d, \theta, r) \in \mathbb{R} \times [0, 2\pi) \times \mathbb{R}$, which is an orthogonal extension of d and θ, with the new signed radius parameter, r. The footpoint $\boldsymbol{t} \in \mathbb{E}^2$ can be decoded the same way as in Equation (2). And the center $\boldsymbol{c} \in \mathbb{E}^2$ is calculated as

$$\boldsymbol{c} = \boldsymbol{t} + r \cdot \boldsymbol{n} \tag{3}$$

Here, we took advantage of the special geometric setting and noted that the osculating circle is tangent to the tangent line at the footpoint. By displacing the footpoint along the normal line by the signed radius, we can reconstruct the correct osculating circle. Figure 4 shows the four cases that can correspond to an order 2 geometric sample.

The last problem to solve is when the osculating circle degenerates into the tangent line, i.e. the case of zero curvature. We might encode this as either positive or negative infinity in r if the number format allows it. For example most of the floating point formats have infinities. In a uniformly quantized representation the minimum and maximum values could be treated as such. Another way could be to use signed curvature $\kappa \in \mathbb{R}$ instead of the radius. The relation between the two quantities is $\kappa = \frac{1}{r}$. This way, the zero curvature sample has zero as a third parameter instead of infinity. In this case, the decoder ignores the curvature and it falls back to the first order mode and the line.

8.3 Evaluation of a sample

To query a value from a sample of the geometric distance field at a query point $\boldsymbol{p} \in \mathbb{E}^2$, first, the properties of the geometric proxy has to be calculated. Then the signed distance is calculated at the query point to the proxy geometry. In first order, calculate $\boldsymbol{n}$ and $\boldsymbol{t}$ using Equations (1) and (2), and the distance as

$$f(\boldsymbol{p}) = \langle \boldsymbol{p}, \boldsymbol{n} \rangle - \langle \boldsymbol{t}, \boldsymbol{n} \rangle. \tag{4}$$

For the second order, the first step is to decode the parameters of the osculating circle using Equations (1), (2) and (3). If the sample represents a circle – and not a line as a fallback, i.e. $\kappa \neq 0$ – then the final distance is

$$f(\boldsymbol{p}) = -\operatorname{sgn}(r) \cdot \left(\|\boldsymbol{p} - \boldsymbol{c}\|_2 - |r| \right). \tag{5}$$

Blending can be used to combine multiple samples but unlike for algebraic samples, we cannot use hardware accelerated texture filters, as the samples first have to be decoded from their encoded form.

9 Results

9.1 Font representations

We tested our signed distance field constructions by font rendering. We used various TrueType fonts [8] in our tests. Distance fields were generated for each glyph, which were combined into a single 2D texture.

A TrueType glyph consists of one or multiple outlines. The winding direction of the outline defines if the outline is an outside border or a border of an inside hole. Outlines consist of a closed loop of line segments and quadratic Bezier segments, given with their control points. We used the FreeType library [14] to load the font data.

These outlines are the input for the DSDF generation, given as a series of segments. For simplicity, we used a brute force method for finding the closest outline point for the samples. An analytic nearest point solution is calculated for every relevant segment, and the closest one is selected. Then the insideness is decided by the direction of the nearest segment, any other features needed for the different DSDF constructions (derivatives, curvature) are extracted analytically. We implemented the DSDF generation for the GPU, and even though it is a brute force algorithm, the run time for the generation is negligible. Also note, that the generation step is usually only needed to be done once.

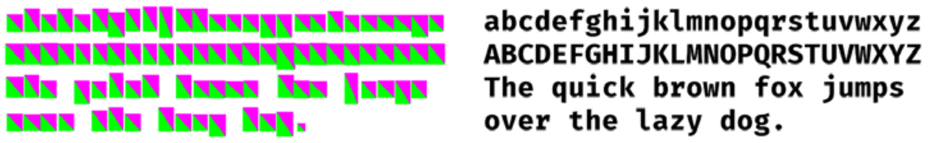

abcdefghijklmnopqrstuvwxyz
ABCDEFGHIJKLMNOPQRSTUVWXYZ
The quick brown fox jumps
over the lazy dog.

Figure 5: Individual rectangles are rendered for every character. The fragment shader samples the DSDF and determines the alpha level of the pixel. The used font is FiraCode Bold [12].

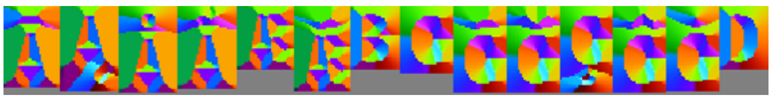


Figure 6: Visualization of a part of an algebraic first order texture atlas. The three color components encode the gradient vector of the distance field.

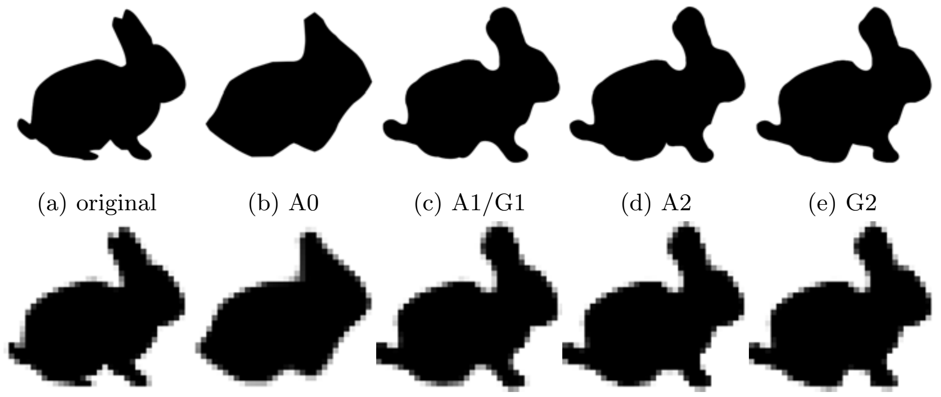
The generated DSDF is saved in a 2D texture and used as an atlas for rendering. The font characters are stored in rectangular regions in the atlas, see for example Figure 6. At render time each character to be rendered is covered with a rectangle (consisting of two triangles – see Figure 5), and a custom fragment shader is used to sample and calculate the signed distance from the outline of the character. The distance is then mapped to the alpha value of the fragment. The mapping can be simply 1 and 0 for negative and positive values respectively, but a better anti-aliased result is achieved by setting a 1-2 pixel wide band with a gradient between the two values. The gradient is tuned to represent the coverage of a hard edge passing through the pixel in the given distance.

Font rendering is a challenging test case for DSDFs, since they often operate with corners which do not satisfy our assumed higher order continuity properties. Nonetheless, higher order DSDFs provide a better use of memory, as it is shown in Section 9.2.

9.2 Test results

We tested the signed distance field constructions on different fonts and vector arts. The tests rasterized the DSDFs as high resolution classical distance fields (containing only signed distance values) and compared them to the signed distance function values of the original vector image. On the tables and figures of this section we refer to the zero, first and second order algebraic distance fields as A0, A1 and A2. Similarly G1 and G2 are the first and second order geometric distance fields. A0 distance fields are the traditional distance fields.

A1 and G1 theoretically encode the same information and therefore reconstruct the same distance values for any query position apart from a small numerical error.



(a) original (b) A0 (c) A1/G1 (d) A2 (e) G2

(f) original (g) A0 (h) A1/G1 (i) A2 (j) G2

Figure 7: Test case for DSDF representations on a bunny silhouette. (a) shows the original vector art, (b)-(e) are high resolution rasterizations of the tested DSDF representations. (f)-(j) show the corresponding rasterizations at the intended display size (34x30px). The DSDFs have a four times larger sample spacing compared to the display resolution, therefore having 16 times less samples/pixels. (c) and (h) contain both A1 and G1 as their reconstruction always matches exactly.

The rasterized images of A1 and G1 are thus indistinguishable in practice.

The DSDFs queried at their sample positions are exact. This means that if the pixels are aligned with the field samples, the rasterized image is the same for all algebraic and geomertic fields. If the pixels and field samples are offset or the pixels are sparser than the samples, the queries of the DSDFs are so close to the true distance function value that the resulting image is stable, giving a robust rendering method.

Our new signed distance field constructions proved to be a useful tool for font and vector art rendering. Example renders of a bunny silhouette can be seen on Figure 7. The shown distance fields have four times lower resolution than the intended display size, meaning that they contain 16 times less samples. This extreme setting is presented here to show that even these sparse fields – with much lower resolution – reconstruct the original vector art closely apart form really fine details. Rasterized at the intended display resolution, these lower resolution higher order distance fields (excluding A0) only differ in a few pixels near the most curved parts.

Table 1 shows the results of two accuracy tests. Test case #1 was some text similar to Figure 5 and test case #2 was calculated on renderings of the bunny seen on Figure 7 but with twice the resolution. The first two columns show the median and mean absolute error of the reconstructed signed distance field values. The third column is the error ratio of the reconstructed sign, i.e. when the inferred inside/outside partitioning is incorrect. Note that A1 and G1 are equal as stated before. G2 usually performs at the same level or better as A2. The first order DSDFs do not seem to lower the absolute distance error compared to the classical

Table 1: Error metrics of the different DSDF representations.

	test #1			test #2		
	median	mean	sign	median	mean	sign
A0	0.0399	0.1053	2.14%	0.0399	0.0871	1.23%
A1	0.0411	0.1119	1.84%	0.0413	0.0861	0.98%
G1	0.0411	0.1119	1.84%	0.0413	0.0861	0.98%
A2	0.0029	0.1002	1.52%	0.0121	0.0785	**0.83%**
G2	**0.0015**	**0.0869**	**1.39%**	**0.0098**	**0.0743**	0.87%

Table 2: Average render times of full screen test texts.

A0	A1	G1	A2	G2
0.306 ms	0.325 ms	0.319 ms	0.370 ms	0.349 ms

distance field (A0), but they always improve the sign correctness metric. This can be seen on Figure 7 as well: the average error values might be close, but visually the first order fields perform better. Similarly, the second order fields always outperform the first order ones. Other test cases have shown similar relative[1] numbers for all constructions.

Table 2 shows averaged render times for the different DSDF constructions from FullHD full screen tests. The sampling of higher order distance fields always costs more than the traditional fields, but they provide the possibility to use lower resolution fields or have better precision at the same resolution. G2 rendering is faster than A2 despite the extra calculations needed. This can be explained by the fact that A2 needs two textures for its 6 coefficients.

10 Conclusion

We proposed a geometric generalization of higher order signed distance fields. We have proven that these constructs have the same approximation order as their Taylor-based algebraic counterparts. These theoretical results were also validated by empirical measurements.

The geometric distance field representations proved to be valid and efficient tools for font and vector art representation and rendering. Our empirical tests have shown that the geometric signed distance fields are as good as the algebraic ones, in fact, in most of the cases the second order geometric construction is more precise than the second order algebraic. This comes with a performance cost of $10-20\%$

[1]Because the error in value and sign both depend on scale

in render times compared to using traditional signed distance fields, assuming the same field resolution.

In the future, we plan to extend our geometric construction to three dimensional signed distance fields. Three dimensional first order algebraic distance fields already proved their applicability [3].

References

[1] Aaltonen, Sebastian. GPU-based clay simulation and ray-tracing tech in Claybook, Game Developers Conference 2018. In *Game Developers Conference*, San Francisco, CA, March 2018.

[2] Bala, Kavita, Walter, Bruce, and Greenberg, Donald P. Combining edges and points for interactive high-quality rendering. *ACM Trans. Graph.*, 22(3):631–640, July 2003. DOI: `10.1145/882262.882318`.

[3] Bán, Róbert and Valasek, Gábor. First order signed distance fields. In Wilkie, Alexander and Banterle, Francesco, editors, *Eurographics 2020 - Short Papers*. The Eurographics Association, 2020. DOI: `10.2312/egs.20201011`.

[4] Ellis, A., Hunt, W., and Hart, J. Real-time analytic antialiased text for 3-D environments. *Computer Graphics Forum*, 38(8):23–32, 2019. DOI: `10.1111/cgf.13757`.

[5] Evans, Alex. Learning from failure: a survey of promising, unconventional and mostly abandoned renderers for 'Dreams PS4', a geometrically dense, painterly UGC game. In *Advances in Real-Time Rendering in Games*. MediaMolecule, SIGGRAPH, 2015.

[6] Fuhrmann, Arnulph, Sobottka, Gerrit, and Groß, Clemens. Distance fields for rapid collision detection in physically based modeling. In *Proceedings of the International Conference Graphicon*, 2003.

[7] Green, Chris. Improved alpha-tested magnification for vector textures and special effects. In *ACM SIGGRAPH 2007 Courses*, SIGGRAPH '07, pages 9–18, New York, NY, USA, 2007. ACM. DOI: `10.1145/1281500.1281665`.

[8] Kaasila, Sampo. Method and apparatus for moving control points in displaying digital typeface on raster output devices, 1994. United States Patent No. US5155805A.

[9] Loop, Charles and Blinn, Jim. *Rendering Vector Art on the GPU.* In *GPU Gems 3*, chapter 25. Addison-Wesley Professional, 2007.

[10] Michail, Ashraf A., Teitlebaum, David B., and Furtwangler, Brandon C. Arc spline GPU rasterization for cubic Bezier drawing, 2010. United States Patent No. US8624899B2.

[11] Osher, S. and Fedkiw, R. *Level Set Methods and Dynamic Implicit Surfaces.* Springer Verlag, 2003. DOI: `10.1007/b98879`.

[12] Prokopov, Nikita. Fira code: free monospaced font with programming ligatures. GitHub URL: `https://github.com/tonsky/FiraCode`.

[13] Ramanarayanan, G., Bala, K., and Walter, B. Feature-Based Textures. In Keller, Alexander and Jensen, Henrik Wann, editors, *Eurographics Workshop on Rendering.* The Eurographics Association, 2004. DOI: `10.2312/EGWR/EGSR04/265-274`.

[14] Turner, David, Wilhelm, Robert, and Lemberg, Werner. The FreeType Project. URL: `https://www.freetype.org/index.html`.

[15] Valasek, Gábor. *Nonlinear Geometric Models.* PhD thesis, Eötvös Loránd University, 2016. DOI: `10.15476/ELTE.2015.123`.

[16] Valasek, Gábor. Generating distance fields from parametric plane curves. In *Annales Mathematicae et Informaticae 48*, pages 83–91, 03 2018. `http://publikacio.uni-eszterhazy.hu/id/eprint/3316`.

[17] Wright, Daniel. Dynamic occlusion with signed distance fields. In *Advances in Real-Time Rendering in Games.* Epic Games (Unreal Engine), SIGGRAPH, 2015.

Acta Cybernetica **25** (2021) 205–221.

Towards Version Controlling in RefactorErl*

Jenifer Tabita Ciuciu-Kiss[ab], Melinda Tóth[ac], and István Bozó[ad]

Abstract

Static source code analyser tools are operating on an intermediate representation of the source code that is usually a tree or a graph. Those representations need to be updated according to the different versions of the source code. However, the developers might be interested in the changes or might need information about previous versions, therefore, keeping different versions of the source code analysed by the tools are required. RefactorErl is an open-source static analysis and transformation tool for Erlang that uses a graph representation to store and manipulate the source code. The aim of our research was to create an extension of the Semantic Program Graph of RefactorErl that is able to store different versions of the source code in a single graph. The new method resulted in 30% memory footprint decrease compared to the available workaround solutions.

Keywords: Erlang, RefactorErl, graph version control, optimisation

1 Introduction

Static program analysis [3] is a method of debugging of the source code of a program performed before the execution. It is used early in development before any other testing. The benefits of static analysis need to be mentioned: speed (compared to manual code reviews automated tools are way faster and the early found coding errors are less costly to fix); depth (static code analysers can cover every possible code execution path); accuracy (human errors can be eliminated).

*The research has been supported by the project "Integrált kutatói utánpótlás-képzési program az informatika és számítástudomány diszciplináris területein (Integrated program for training new generation of researchers in the disciplinary fields of computer science)", No. EFOP-3.6.3-VEKOP-16-2017-00002. The project has been supported by the European Union and co-funded by the European Social Fund. The research is part of the "Application Domain Specific Highly Reliable IT Solutions" project that has been implemented with the support provided from the National Research, Development and Innovation Fund of Hungary, financed under the Thematic Excellence Programme TKP2020-6 (National Challenges Subprogramme) funding scheme.

[a]ELTE, Eötvös Loránd University, Budapest, Hungary

[b]E-mail: `kuefmz@inf.elte.hu`, ORCID: `0000-0002-3170-6730`

[c]E-mail: `toth_m@inf.elte.hu`, ORCID: `0000-0001-6300-7945`

[d]E-mail: `bozo_i@inf.elte.hu`, ORCID: `0000-0001-5145-9688`

DOI: `10.14232/actacyb.289386`

Version control [18] (also known as revision control or source control) is a component of software configuration management. It is used for managing changes to programs, documents and other data. The idea of storing the differences between different versions of a document is as old as writing is but it became more important in the last two decades when the area of computing has started. In software engineering, version control is any kind of tracks that provides control over the changes between the different version of source code. Version controlling tools make the process easier and much faster, so it is very convenient to use them. One of the most famous tools for that is Git [8], which is known by most software developers.

RefactorErl [4, 19] is a static source code analyzer and transformation tool for Erlang programs. Beyond that RefactorErl does thorough static analysis on the given source code, it has many other features to support code comprehension: it finds the dependencies between the modules, provides a query language to gather semantic information about the source code, performs clustering, etc. Some source code analysers have their own built-in version controlling systems. RefactorErl does not provide version controlling, it works on a given snapshot of a software.

The internal representation of RefactorErl gives us the opportunity to build the version control over that. The tool represents the source code in a Semantic Program Graph (SPG) [12] which will be explained in detail later. The graph is stored in a database to make the results of the source code analysis permanently available. The tool can work with different databases (i.e. the distributed, relational Mnesia database [14], or the key-value Kyoto database [10]), but it builds the same internal representation, the SPG, in any cases. All the analyses and the transformations are performed on the graph layer, thus our version controlling is defined on the top of the SPG and not on the database layer.

The main contribution of this paper are the algorithms as an extension of RefactorErl to handle different versions of the source code. Our first tests on open source repositories provided a memory saving for about 30%.

In Section 2, we introduce some information about the RefactorErl tool and the data structure used by the tool. In Section 3, the algorithm of finding the differences between different versions is presented which is followed by Section 4, where the algorithm of storing the found differences is discussed. Then, in Section 5, we present some usage of the version control. In Section 6 the evaluation is explained. Section 7, contains information about the related work. Finally, Section 8 concludes the paper.

2 Background

In our work we focus on the RefactorErl framework that was designed to analyse Erlang source code.

Erlang The Erlang [6, 1] programming language is a general-purpose, competitive, dynamically typed programming language. Other important features include being a greedily evaluated, non-purely functional and open source language. It is designed to develop systems with high fault tolerance and real-time robust size. Erlang programs run inside a virtual machine (Erlang VM or *node*), so programs written in Erlang are platform-independent. Erlang's standard library is called OTP (Open Telecom Platform [13]), which was originally designed to write telecommunications software, but can be used more widely. The Erlang runtime environment and OTP are collectively referred to as Erlang/OTP.

RefactorErl Static source code analyser tools are heavily used in software development and maintenance processes. RefactorErl [4, 17, 19] is an open-source static source code analyzer and transformer tool for Erlang source codes. Despite the fact that RefactorErl is still considered as a prototype tool, its usefulness in industrial usage has been proved already. It was initially developed at the request of Ericsson and it is still used by developers there. Apart from static analysis it has many other features: it is compatible with the most famous code editors; it can find the dependencies between the modules and it also represents this information in well designed graphs; it gathers many different kind of information about the source code; the found bugs can be investigated. These features manifest the significance of RefactorErl.

Semantic Program Graph During the initial analysis, the tool builds a Semantic Program Graph (SPG) [12]. In order to make the source code analysis results available for further use, we need to save the graph. The SPG is stored in a database. Different databases can be used to store the SPG, but all the analyses and transformations are manipulating the SPG itself. If the source code is changed, the tool updates the graph, so in the new graph, only the new information is available. Any kind of backups about different versions are really expensive at the current implementation, because we can use the backuping features of the used database only to backup the whole SPG. Therefore we would like to design a much more effective way of version handling on the SPG level. An adequate solution to this problem is to save the differences in the same graph. In this way, we do not need to store database backups and load them once an older version is needed.

The SPG is a three-layered rooted, directed, labelled graph structure that contains lexical, syntactic and semantic information about the source code. The different kinds of data are stored in the nodes of the graph (in the attributes of the node) that are linked through tagged edges. These edges contain structural information about the nodes. The graph has a specified structure. It starts with a root node, which is followed by a node containing the information about the file (the name, absolute path, etc.). They are linked with an edge tagged with the file. The file node is linked to its forms: the function definitions and attributes. The function definition forms are linked to its clauses, and so on. In addition, there are module and function semantic nodes to represent the semantic information as well. Based on the structure of the graph, we were able to define an algorithm to find the differences between two versions of the source code by traversing the representing syntax trees simultaneously.

Motivation During a software development in industry, versions occur quickly and many times the difference between them are important since the new change could lead to some error or the previous version was more effective, etc. Storing the versions in a static analyzer tool helps the developers to fasten the process of debugging, not to mention the analyses of the relations between the versions that the tool can provide.

As it was previously mentioned, any kind of backups are expensive since RefactorErl can store only one version of a source code at once. It is possible to manually create backups of the different versions on database level, but a single running instance of RefactorErl can handle only one version at the same time. Loading an old version of the source code requires to start a new RefactorErl instance and load the whole old SPG stored in the backups. That is an inconvenient, time-, memory- and resource-consuming task.

3 Detecting differences

As it was mentioned before, RefactorErl generates an SPG for each module and it stores the analyzed source code in this structure. When we re-analyze a module that has already been analyzed, the tool recognizes this and replaces the subgraph belonging to the modified source code part[1] with a new subgraph, but the old and the new version of the software have not been related to each other so far. The old subgraph is deleted after the new subgraph is created, however, there is a point in the analysis when the information for both graphs can still be found in the tool. We use this point to analyze the two subgraphs, detect differences, and perform version control.

In our version controlling, the primary goal is to somehow explore the differences between the two graphs, and then, in some way, to represent the information stored by the two subgraphs in a merged subgraph.

The differences can be categorised into three major groups that need to be recognized and addressed in different ways. These three groups are:

- Update
- Deletion
- Insertion

Let us look at how to handle each case properly. For that we will use an example seen on Figure 1. The example contains two function definitions. The first one, `foo`, checks whether its argument is `0` and returns the term zero. Function `op` multiplies its arguments.

[1]RefactorErl performs an incremental analysis on form level: once a form (e.g. function, attribute, etc.) changed in the source code, it reanalyses only the changed form

```
foo(0) -> zero.
op(A, B) -> A * B.
```

Figure 1: Example Erlang function

3.1 Update

We talk about an update when we had something in a version, and we changed something in it, without inserting a new part or deleting a whole part of the source code.

For a better understanding we make a modification on the example presented in Figure 1. As it can be seen in Figure 2, we made an update on the return value of the function **op**, the multiplication operation (`*`) has been changed to plus (`+`).

```
foo(0) -> zero.
op(A, B) -> A + B.
```

Figure 2: Example Erlang function with update

The algorithm of finding the updates is the most important, because after resolving those updates we will consider the remaining changes either as a deletion or an insertion depending on where the extra elements on the graph were found.

As a first step we make all combinations of function nodes and store them in a list[2]. We will go through these function pairs and check if they are the same function in different versions. For that we check whether the name and the arity of the functions are the same. If so, we say that these two functions are the same and we look for the differences between those functions. This algorithms works well for real code examples as the name of the function rarely changes after a commit[3].

The recursive function of finding the differences gets two nodes, one of them is from the SPG of the previous version and the another one is from the new version. The algorithm can be found in Figure 3. The list of pairs that the algorithm initially gets is the list of the investigated form nodes. The `TableOfDifferences` variable refers to the ETS table [7] in which the difference nodes are stored. While analyzing a new version of a file RefactorErl recognizes the new, deleted and modified forms of the source code based on the hash value of the forms. At this point we save the changed forms[4] in ETS tables for further analysis. The `equivalent` function checks whether the two nodes are the same in the required attributes (number and type of attributes, number of children, etc.), and if not, we say that we found an update at that node and we insert the node and the subgraph bellow it into the version

[2]We need to generate all function pairs because at this point we do not know their names yet, as the name is stored in a lower level in the graph.

[3]In the current implementation, we recognise renaming of function definitions as newly inserted and deleted functions. Some heuristics can be applied to change this behaviour.

[4]We save the sub-syntaxtrees representing the changed forms.

controlled graph with `insertUpdate` function. If we did not find an update, we prepare the list of pairs of the children nodes with `makePairs` function and check whether the children contain updates. Each edge label has a name and a serial number in the SPG representing the order of the syntax tree elements. Thus the `children` function organizes the nodes in this way, so we are able to create the pairs for the next step in the recursive investigation.

```
selectUpdate(ListOfNodePairs)
    for NodePair in ListOfNodePairs do
        Children1 = children(first(NodePair))
        Children2 = children(second(NodePair))
        if NodePair in TablesOfDifferences then
            if !equivalent(Children1, Children2) then
                insertUpdate(Children1, Children2)
            else
                ListOfChildrenPairs =
                     makePairs(Children1, Children2)
                selectUpdate(ListOfChildrenPairs)
            end if
        end if
    end for
    return ok
```

Figure 3: `selectUpdate` function

3.2 Deletion

A deletion is when we had something in the source code in a previous version, and we deleted it in a version after that.

For finding the deletions, we go through the nodes that have been present in the previous version of the source code but they are not present any more. That means that we go through the SPG and if we find a node that does not have a version in the new graph then we conclude that a deletion is found.

In Figure 4, one can see that we made a deletion by deleting the function `foo`.

```
op(A, B) -> A * B.
```

Figure 4: Example Erlang function with deletion

3.3 Insertion

An insertion is, when we insert a brand new part in the source code, which did not exist in the previous version.

We use a similar algorithm here as for deletion, the difference is that when we look for insertions we look for those nodes that are present in the new graph but they had not been present in the old graph.

As an example, we made an insertion to the code seen in Figure 4, so after making a deletion we do an insertion on that code. We add the function `bar` and the result code can be seen on Figure 5.

```
op(A, B) -> A * B.
bar(1) -> one.
```

Figure 5: Example Erlang function with insertion

4 Updating the graph

After finding and identifying the different types of changes, we need to represent them in the SPG. The basic idea is that we introduce two different types of tags for the edges. Updates, deletions and insertions are handled in a similar way. Thanks to RefactorErl, we can easily and efficiently insert new nodes and edges. In the following, we will demonstrate the algorithm for each case.

Let us consider the examples used in the previous section to present the method we used for updating the graphs. To make the differences more illustrative, first we take a look at the non version controlled graph which is generated by the RefactorErl. You can find that in Figure 6.

4.1 Update

In the case presented in Figure 2, we find the difference presented in the previous section. We insert the whole subgraph under the found difference between the operations with a versioned edge. An edge is versioned when it has the same name as before but it has a v at the beginning of its name. In the resulting graph under the difference node, we can find the content of both versions in the different subgraphs. One can see in Figure 7 the edge *vbody* which marks the discussed update.

In this way, we can easily find the versioned part in the graph and we can keep all the previous features of the tool. An obvious question may arise, whether it is possible to store more than two versions in this way. Yes, it is possible, but we need to make it customisable by the users.

4.2 Deletion

The handling of this difference is very similar to the algorithm from the previous subsection. Let us consider the modified example in Figure 7 again. We can see that the difference seen in Figure 4 discussed in the previous section is released.

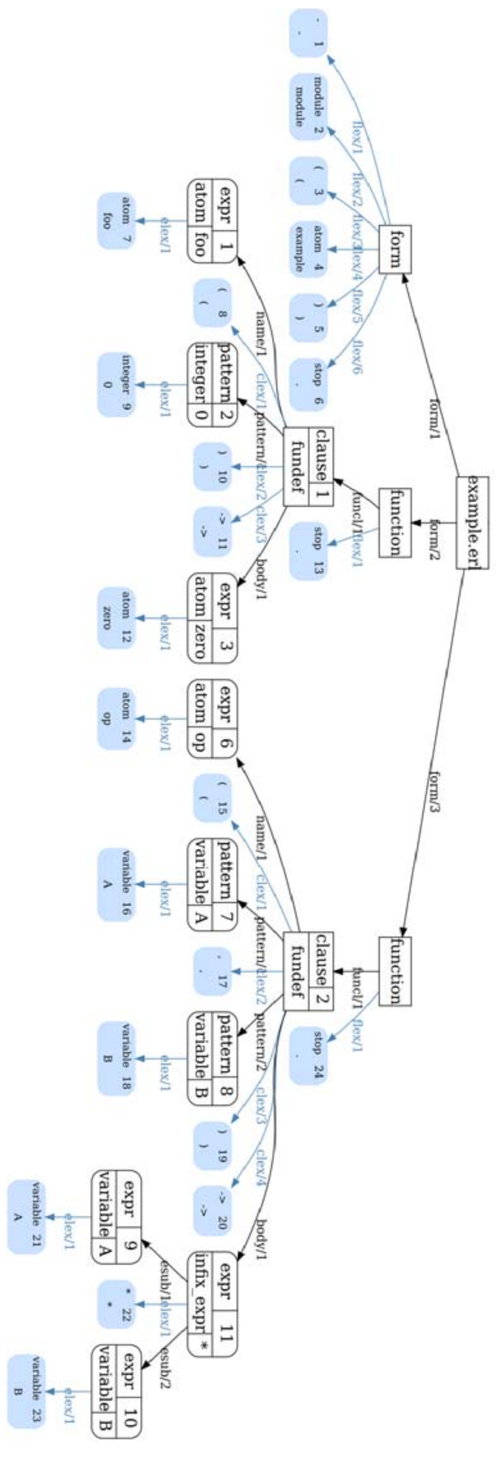


Figure 6: The not version controlled SPG

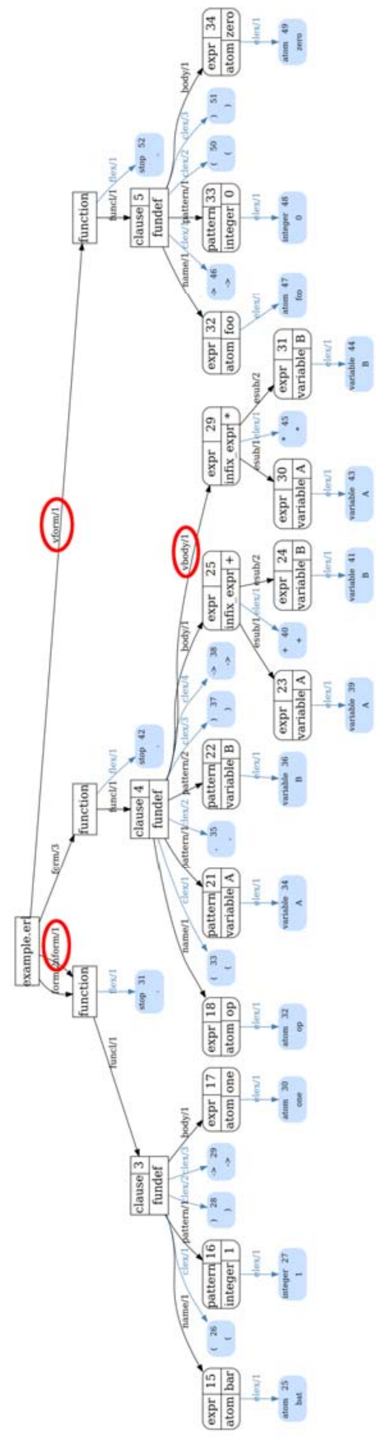


Figure 7: The version controlled SPG

To handle this, we insert the deleted subgraph under its parent node (the original function node) with a *vform* versioned edge seen on Figure 7.

4.3 Insertion

Considering the example seen on Figure 5, we have an inserted function in the code and we want to represent that in the version controlled graph. For that, we insert the difference in the same way but for the tag of the edge, we insert an n at the beginning instead of the v, one can see the inserted *nform* edge on the Figure 7. When we do that, we keep the nonversioned edge as well, because it does not bother us and it is needed to have a syntactically correct graph. Now, we can easily find the insertions since it had a different concept than the deletions and updates.

5 Gathering information

At this point we have the version controlled SPG saved in the tool. We would like to extract information from the graph and for that we use the query language which was already part of the tool and we extended it for the version controlled graph.

The semantic query language [11] was designed to query information about the analyzed Erlang code. The concepts of the query language are defined according to the semantic units and relationships of the Erlang language, e.g. functions and function calls, records and their usage, etc.

The elements of the language are the following entities: module, function, variable, etc. Each of them has several selectors and properties. A selector selects a set of entities that meet the given requirement. A property describes some properties of an entity type.

One of the needed information pieces could be to define whether the modules are version controlled or not. For that we performed the query seen in Figure 8. The query returns whether the uploaded modules are version controlled or not, in other words, its return value is a Boolean which is true if a module contains information from a previous version and false otherwise.

```
ri:q("mods.is_versioncontrolled").
ri:q("mods.is_versioned").
```

Figure 8: "Is the module version controlled" query

For a huge repository, this information might not be needed for all the modules or we would like to know which specific functions are version controlled. For that we realized a query found in Figure 9. It is quite similar with the first one apart from that it defines whether the functions in the modules are version controlled or not in the same way.

```
ri:q("mods.funs.is_versioncontrolled").
ri:q("mods.funs.is_versioned").
```

Figure 9: "Is the function version controlled" query

Also we might need the names of the version controlled functions and the query presented in Figure 10. It is built on the query presented in Figure 9. The return value of this query is a string which contains the name of the version controlled function in case that it is version controlled and `not_versioncontrolled` otherwise.

```
ri:q("mods.funs.versname").
ri:q("mods.funs.versionname").
ri:q("mods.funs.versionedname").
ri:q("mods.funs.vname").
```

Figure 10: "Name of version controlled functions" query

It is possible that we are not sure about the stored version on the SGP. This problem can be solved using the query presented in Figure 11. This query gives the number of the included versions on graph for each module. The returned value is an integer which is one if only the actual version is stored, two if the previous version is also included on the graph, etc.

```
ri:q("mods.version_counter").
ri:q("mods.vercount").
ri:q("mods.versions_number").
ri:q("mods.vernum").
```

Figure 11: "Number of stored versions" query

6 Evaluation

We made some measurement for the algorithm using a public GitHub repository [5] and two of its versions, the used old version can be found with change-id `499de032d7c6fc294aff977bea1405e09a1a3eae` and the new version can be found with change-id `5b7363c14071abe92d4039a7fc96371bd6eee91e`. The number of lines that differ between the two versions is 646.

To investigate the measurement, we check the tool's memory footprint[5] in five different states of the tool: when the tool is empty; when only the first version is loaded into the tool; when only the second version is loaded to the tool; when both

[5]We used the *erlang* : *memory*(). function.

Table 1: Results for version controlling a public repository

	ets	**total**
Empty	1399208	**23318920**
Old version	90707888	**119560720**
New version	94655240	**123790536**
Version controlled	106574824	**177795384**
Old- + New- version	183963920	**220032336**

of the version are loaded to the tool separately; when the versions are loaded to the tool using version control.

It is important to mention that the results are strongly dependent from the analyzed data. If there are almost no differences between the versions, we will get almost 100% efficiency. Meanwhile, when the source code has totally changed, the version control is not efficient at all, as both of the source codes have to be saved.

The results for the measurements presented above can be seen in Table 1 and Figure 12. As one can see, the tool's memory footprint is the smallest when the tool is empty. When only the first version is loaded, it has a smaller footprint than the second version. It is generally true as the code is developed over time. When both of the versions are loaded separately, the memory requirements are around twice as big as for the versions separately. When the algorithm of version control is used we can see that the memory requirement for the tool is less than for the case without using version control. This means that the algorithm works as it was expected at the first place. Quantifying the results in this case $32,52\%$ memory gain was achieved.

At the same time we also checked the overhead of the version controlling algorithm on the runtime: we experienced 4% overhead.

7 Related work

Version control of trees [2] presents everywhere in software development; generally it is solved by serializing the tree, not via methods operating on the tree itself.

Ralph Hinze and Ross Paterson have published a data structure called finger tree [9], which is purely functional. It can be used to efficiently implement other functional data structures. The finger tree, amortized, provides access to the "fingers" (leaves) of the tree where it stores the data. Depending on the size of the given part, we can concatenate or split data in logarithmic time in the size of the smaller piece. Internal nodes store what associative action their descendants have performed. The data stored in internal nodes can be used to provide its functionality to other non-tree data structures.

In the finger tree, the finger is the part where you can access a part of the data structure, in imperative languages this is called pointers. Fingers are structures

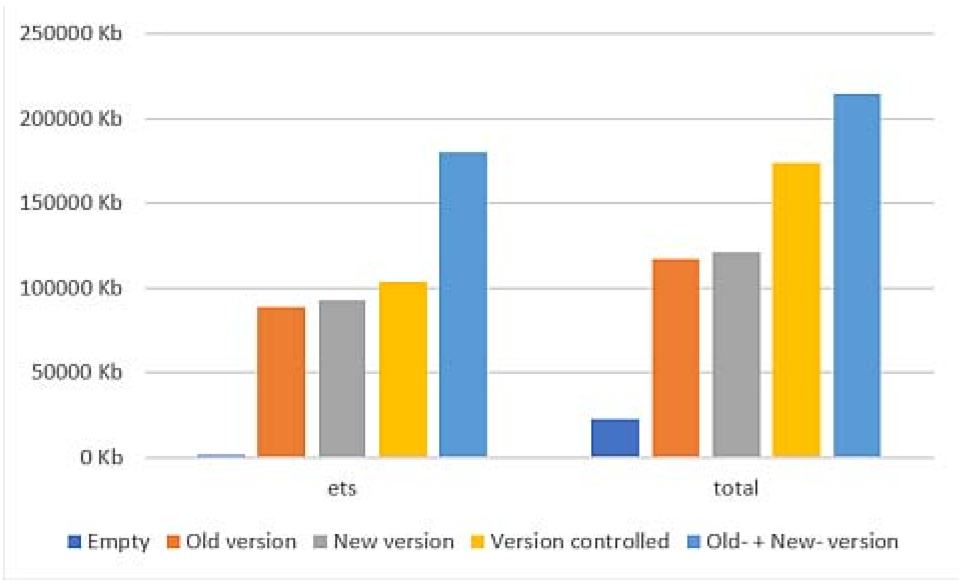


Figure 12: Diagram for results version controlling a public repository

that point to the end of a series or a letter to *node*. The fingers are part of the original wood, thus ensuring constant access over time. It also stores in each internal node the result of applying some associative operation to its descendants. This ”summary” data stored in the internal nodes can be used to provide the functionality of data structures other than trees. Finger trees are inherently persistent which means that older versions of the tree are always preserved. This data structure is often used for for handling the undo/redo operation in editors because it makes easy to find the differences in the text. The undo/redo actions can be interpreted as version control, but there some issues that need to be mentioned. The inside representation of RefactorErl does not make it possible to use the undo/redo method for version control without any changes, because the SPG (Semantic Program Graph) structure does not have the ”summary” data property, thus the finger tree change detection mechanism cannot be adopted easily for our use case.

UML diagrams also have version control issues [15]. The logic of the version control solution for UML diagrams is very similar to the version control implemented in RefactorErl. First, let us look at how a UML diagram is built. When designing UML diagrams, we distinguish between two types of diagrams, those that are semantically relevant and those that are not. It is important that UML diagrams also have a tree structure. *Node* has a well-defined type, and there can be different types of connections between different types of *node*. Some *node* is complex, they can carry several different types of information, but one can also store an entire subchart.

Differences need to be classified into different groups, as each difference needs to be treated differently. Let us look at what types of differences there can be between UML diagrams. First, we divide the differences into two groups, there are differences within *node* when only the content of *node* has changed, and there are structural differences when there has already been a change in the structure itself. Differences within *node* can be simple when only one attribute has changed, it can be multivalued when multiple attributes have changed, or it can be a change within an attribute reference when only the reference has changed.

In the event that there has also been a change within the structure, we must also divide them into several groups.

- Internal design difference: when there was no change within the elements, only the relationships between them changed.

- Shifting within a structure: when the structure remains the same as it was, only one or more edges moved.

- Internal node shifting: when we put operations or attributes from one class to another.

- Position shifting: when rearranging the order of attributes or operations within a class.

As the inside representation of RefactorErl does not force us to divide the handling of version control in as many different groups, it is important to know how different data structures are version controlled.

The difference computation of the version control algorithm of the UML diagrams works in a very similar way as in RefactorErl. Both of the algorithms go through the nodes in a breadth-first order and compare them to each other. The difference is that we consider changes inside nodes and for UML diagrams the differences are identified comparing the subgraph for each node. This would not be effective for us as the elements can be reorganized easily without making any significant change so we decided to compare each node pair only.

8 Conclusion

In this paper we presented how version control works in RefactorErl. After defining the different cases of modifications between different versions we explained how those differences can be found and represented in the tool. We also showed some use-cases and the results of the analysis of its efficiency.

The proof of concept implementation of the algorithm is integrated within the RefactorErl environment. In general, it has an overhead on the run-time, but it has a smaller memory footprint. The previous graphs for previous versions do not need to be saved, because the annotated new graph is storing all the required information about the different versions. The algorithm can be optimized for different resources and different projects. We are investigating the possible usage of the versioned graph now.

For treating more than one version, an extra parameter needs to be introduced for counting the version number next to versioned edges. By this we can easily differ the different versions from each other and it also makes possible to store all the version in one graph.

8.1 Future work

The next step is the extension of the implementation to handle more than two versions. For this an index could be added for the version controlled edges (i.e. $vbody_1, vbody_2, \dots$). This way we can differ the versions easily. We would like to note here that only the representation of the changes need to be changed at this stage, the defined algorithm for finding and categorizing the differences is the same in all cases.

The algorithm could also we extended with some extra heuristic algorithms, so we could find the differences in as low level as possible. The heuristic algorithms could be defined after collecting some data from industrial usage.

The designed generic difference detection algorithm can be used in other projects as well. It would be a nice proof of the reusability of our approach to try it on the representation of the SSQSA [16] language independent source code analyser framework.

References

[1] Armstrong, Joe. Erlang. *Commun. ACM*, 53(9):68–75, September 2010. DOI: `10.1145/1810891.1810910`.

[2] Asenov, Dimitar, Guenat, Balz, Müller, Peter, and Otth, Martin. Precise version control of trees with line-based version control systems. In *International Conference on Fundamental Approaches to Software Engineering*, pages 152–169. Springer, 2017. DOI: `10.1007/978-3-662-54494-5_9`.

[3] Bellairs, Richard. What is static analysis (static code analysis). *Perforce Software*, 2020.

[4] Bozó, I., Horpácsi, D., Horváth, Z., Kitlei, R., Köszegi, J., M., Tejfel., and Tóth, M. RefactorErl — source code analysis and refactoring in Erlang. In *Proceedings of the 12th Symposium on Programming Languages and Software Tools*, pages 138–148, Tallin, Estonia, October 2011.

[5] Chesneau, Benoit. hackney - HTTP client library in Erlang. `https://github.com/benoitc/hackney`. [Accessed: 25 August 2020].

[6] Ericsson AB. Homepage of the Erlang Programming Language. `http://www.erlang.org` [Accessed: 2019.08.07].

[7] Fritchie, Scott Lystig. A study of Erlang ETS table implementations and performance. In *Proceedings of the 2003 ACM SIGPLAN Workshop on Erlang*, ERLANG '03, page 43–55, New York, NY, USA, 2003. Association for Computing Machinery. DOI: `10.1145/940880.940887`.

[8] Github. `https://github.com/` [Accessed: 20 December 2020].

[9] Hinze, Ralf and Paterson, Ross. Finger trees: a simple general-purpose data structure. *Journal of functional programming*, 16(2):197–217, 2006. DOI: `10.1017/S0956796805005769`.

[10] Horák, Ales and Rambousek, Adam. The KYOTO database system. Technical report, Masarykova univerzita, 2010. `http://www.kyoto-project.eu/`.

[11] Horváth, Zoltán, Kozsik, László Lövei Tamás, Király, Roland, Tóth, Melinda, Kitlei, Róbert, Horpácsi, Dániel, and Bozó, István. Extended semantic queries on Erlang programs and comprehensive testing of RefactorErl. Technical report, Tech. Report 2010. Ericsson Hungary, 2010.

[12] Horváth, Zoltán, Lövei, László, Kozsik, Tamás, Kitlei, Róbert, Víg, Anikó Nagyné, Nagy, Tamás, Tóth, Melinda, and Király, Roland. Modeling semantic knowledge in Erlang for refactoring. In *Knowledge Engineering: Principles and Techniques, Proceedings of the International Conference on Knowledge Engineering, Principles and Techniques, KEPT 2009*, volume 54(2009) Sp. Issue of *Studia Universitatis Babeş-Bolyai, Series Informatica*, pages 7–16, Cluj-Napoca, Romania, Jul 2009.

[13] Logan, Martin, Merritt, Eric, and Carlsson, Richard. *Erlang and OTP in Action.* Manning Publications Co., 2010.

[14] Mattsson, Håkan, Nilsson, Hans, and Wikström, Claes. Mnesia - A distributed robust DBMS for telecommunications applications. In *International symposium on practical aspects of declarative languages*, pages 152–163. Springer, 1999. DOI: `10.1007/3-540-49201-1_11`.

[15] Ohst, Dirk, Welle, Michael, and Kelter, Udo. Differences between versions of UML diagrams. In *Proceedings of the 9th European software engineering conference held jointly with 11th ACM SIGSOFT international symposium on Foundations of software engineering*, pages 227–236, 2003. DOI: `10.1145/940071.940102`.

[16] Rakić, Gordana, Budimac, Zoran, and Savić, Milos. Language independent framework for static code analysis. In *Proceedings of the 6th Balkan Conference in Informatics*, BCI '13, page 236–243, New York, NY, USA, 2013. Association for Computing Machinery. DOI: `10.1145/2490257.2490273`.

[17] RefactorErl. Static source code analyser and refactoring tool for Erlang. `https://plc.inf.elte.hu/erlang`. [Accessed: 8 August 2020].

[18] Sink, Eric. *Version Control by Example*. PYOW Sports Marketing, 1st edition, 2011.

[19] Tóth, Melinda and Bozó, István. Static analysis of complex software systems implemented in Erlang. In *Central European Functional Programming School*, pages 440–498. Springer, 2011. DOI: `10.1007/978-3-642-32096-5_9`.

Acta Cybernetica **25** (2021) 223–232.

Using the Fisher Vector Approach for Cold Identification[*]

José Vicente Egas-López[a] and Gábor Gosztolya[b]

Abstract

In this paper, we present a computational paralinguistic method for assessing whether a person has an upper respiratory tract infection (i.e. cold) using their speech. Having a system that can accurately assess a cold can be helpful for predicting its propagation. For this purpose, we utilize Mel-frequency Cepstral Coefficients (MFCC) as audio-signal representations, extracted from the utterances, which allowed us to fit a generative Gaussian Mixture Model (GMM) that serves to produce an encoding based on the Fisher Vector (FV) approach. Here, we use the URTIC dataset provided by the organizers of the ComParE Challenge 2017 of the Interspeech Conference. The classification is done by a linear kernel Support Vector Machines (SVM). Owing to the high imbalance of classes on the training dataset, we opt for undersampling the majority class, that is, to reduce the number of samples to those of the minority class. We find that applying Power Normalization (PN) and Principal Component Analysis (PCA) on the Fisher Vector features is an effective strategy for the classification performance. We get a better performance than that of the Bag-of-Audio-Words approach reported in the paper of the challenge.

Keywords: computational paralinguistics, speech processing, machine learning, fisher vector

1 Introduction

Upper respiratory tract infection (URTI) is an infectious process for any of the components of the upper airway. E.g., the common cold, a sinus infection, amongst

[*]This study was supported by the Hungarian Artificial Intelligence National Laboratory, by the National Research, Development and Innovation Office of Hungary via contract NKFIH FK-124413, and by the grant NKFIH-1279-2/2020 of the Hungarian Ministry of Innovation and Technology. Gábor Gosztolya was also funded by the János Bolyai Scholarship of the Hungarian Academy of Sciences and by the Hungarian Ministry of Innovation and Technology New National Excellence Programme ÚNKP-20-5.

[a]Institute of Informatics, University of Szeged, Hungary, E-mail: egasj@inf.u-szeged.hu, ORCID: 0000-0002-5622-9192

[b]MTA-SZTE Research Group on Artificial Intelligence, Szeged, Hungary, E-mail: ggabor@inf.u-szeged.hu, ORCID: 0000-0002-2864-6466

DOI: 10.14232/actacyb.287868

others. Being able to automatically assess whether a subject has a cold may be relevant when trying to prevent the spread of it by predicting its patterns of propagation. The area of computational paralinguistics differs from Automatic Speech Recognition (ASR), which focuses on the actual *content* of the speech of an audio signal. Here, computational paralinguistics may provide the necessary tools for determining the *way* the speech is spoken. Various studies have offered promising results in this area: diagnosing neuro-degenerative diseases using the speech of the patients [5, 6, 7], the classification of crying sounds and heart beats [10], estimating the sincerity of apologies [9], determining the depression of a subject [4]. In this study, we focus on finding specific voice patterns latent in the speech of subjects having a *cold*.

Previous studies applied various approaches for classifying *cold* subjects using the same corpus. For example, Gosztolya et al. employed Deep Neural Networks for feature extraction for this purpose [8]. Huckvale and Beke utilized four types of voice features for studying changes in health [11]. Furthermore, Kaya et al. [14] introduced the application of a weighting scheme on instances of the corpus, making use of a Weighted Kernel Extreme Learning Machine in order to handle the imbalanced data that comprises the URTIC corpus. As any other computational paralinguistic task, assessing a cold from the speech is a challenging issue. Finding out the latent patterns that could characterize or represent a cold speech does not only depend on the feature extraction phase but in the data itself too. This may be attributed to different perspectives: limited amount of data, data imbalance, quality of the recordings.

In this study, we exploit the Upper Respiratory Tract Infection Corpus (URTIC that was the dataset of one of the Sub-Challenges in the ComParE Challenge from Interspeech 2017) [21]. In the feature extraction phase, we selected frame-level features. Namely, we utilize Mel-frequency Cepstral Coefficients (MFCC) as audio-signal representations, extracted from the utterances. This allowed us to fit a generative Gaussian Mixture Model (GMM) that can produce an encoding based on the Fisher Vector (FV) approach. That is, the computation of low-level patch descriptors together with their deviations from the GMM gives us an encoding (i.e. feature) called the Fisher Vector.

Unweighted Average Recall (UAR) scoring was used to measure the performance of the model since it is the de facto standard metric for these kinds of challenges [18]. To the best of our knowledge, this is the first study that focuses on making use of a FV representation in order to detect a cold.

Furthermore, we find that applying Power Normalization (PN) and Principal Component Analysis (PCA) on the Fisher Vector features is an effective strategy for the classification performance. In the next part of our study, we employ a late-fusion of the ComParE Bag-of-Audio-Words (BoAW) features with the Fisher Vector representations. Mentioned fusion technique contributes to the classification performance.

Table 1: Upper Respiratory Tract Infection Corpus (URTIC).

Class	**Train**	**Development**	**Test**	**Total**
Cold	970	1011	895	2876
Not-Cold	8535	8585	8656	25,776
Total	9505	9596	9551	28,652

2 Data

The entire dataset consists of 382 male speakers, 248 female speakers, with a mean age of 29.5 years; and a sampling rate of 44.1kHz downsampled to 16kHz. For the Sub-Challenge, the corpus was provided by the Institute of Safety Technology, University of Wuppertal, Germany. The following tasks were performed by the participants: they had to read short stories (e.g. the well-known story in the field of phonetics *The North Wind and the Sun*, to produce voice commands (such as numbers from 1 to 40), and to narrate spontaneous speech (i.e. tell something about their last weekend or their best vacation). Note that the number of tasks varied for each speaker. The recordings were split into 28,652 chunks of 3 to 10 seconds in length. Specifically, the division of the chunks was carried out in a speaker-independent manner, each partition (i.e. train, development, and test) having 210 speakers. The training and development sets are both comprised by 37 subjects having a cold and 173 subjects not having a cold. The reader may see more details in [22]. The number of samples and classes for each dataset is described in Table 1.

3 Methodology

Figure 1 shows the methodology employed in this study: (1) Frame-level features (MFCC) were extracted from the utterances; (2) A Gaussian Mixture Model (GMM) is trained utilizing the MFCC representations; (3) Fisher Vector features are extracted using the trained GMM; and (4) SVM performs the classification task.

3.1 Frame-level feature extraction

The features we employed were the well-known MFCCs with a dimension of 13, along with their first and second order derivatives, frame-length and frame-shift of 25 ms and 10 ms, respectively. We used the Kaldi Speech Recognition Toolkit [17] for this task.

3.2 Fisher Vector (FV)

The FV approach is an image representation that pools local image descriptors [19]. It was originally intended for image classification but here we exploit its ability to generate a complete representation of the samples which are later characterized by their deviation from a generative GMM. The samples can be thought of as local patch descriptors of an image. In our case, they are the frame-level features of an audio signal. FV is an improved version of the general case called the Fisher Kernel (FK) [12], which measures the similarity of two objects from a parametric generative model of the data. The FK will be explained more in detail in the next section. FV basically assigns a local descriptor to elements in a visual dictionary. This approach stores visual word occurrences and takes into account the difference between dictionary elements and pooled local features, it stores their statistics as well.

3.2.1 Fisher Kernel (FK)

It seeks to measure the similarity of two objects from a parametric generative model of the data (X) which is defined as the gradient of the log-likelihood of X [12]:

$$G_\lambda^X = \bigtriangledown_\lambda \log \upsilon_\lambda(X), \tag{1}$$

where $X = \{x_t, t = 1, \ldots, T\}$ is a sample of T observations $x_t \in \mathcal{X}$, υ represents a probability density function that models the generative process of the elements in $\mathcal{X}$ and $\lambda = [\lambda_1, \ldots, \lambda_M]' \in R^M$ stands for the parameter vector υ_λ [19]. Thus, such a gradient describes the way the parameter υ_λ should be changed in order to best fit the data X. A way to measure the similarity between two points X and Y by means of the FK can be expressed as follows [12]:

$$K_{FK}(X,Y) = G_\lambda^{X\prime} F_\lambda^{-1} G_\lambda^Y. \tag{2}$$

Eq. (3) shows how the Cholesky decomposition $F_\lambda^{-1} = L_\lambda' L_\lambda$ can be utilized to rewrite the Eq. (2) in terms of the dot product:

$$K_{FK}(X,Y) = \mathscr{G}_\lambda^{X\prime} \mathscr{G}_\lambda^Y, \tag{3}$$

where

$$\mathscr{G}_\lambda^X = L_\lambda G_\lambda^X = L_\lambda \bigtriangledown_\lambda \log \upsilon_\lambda(X). \tag{4}$$

Such a normalized gradient vector is the so-called *Fisher Vector* of X [19]. Both the FV $\mathscr{G}_\lambda^X$ and the gradient vector G_λ^X have the same dimension.

3.2.2 Fisher Vector for audio-signals

Let $X = \{X_t, t = 1 \ldots T\}$ be the set of D-dimensional local SIFT descriptors extracted from an image and let the assumption of independent samples hold, then Eq. (4) becomes:

$$\mathscr{G}_\lambda^X = \sum_{t=1}^{T} L_\lambda \bigtriangledown_\lambda \log \upsilon_\lambda(X_t). \tag{5}$$

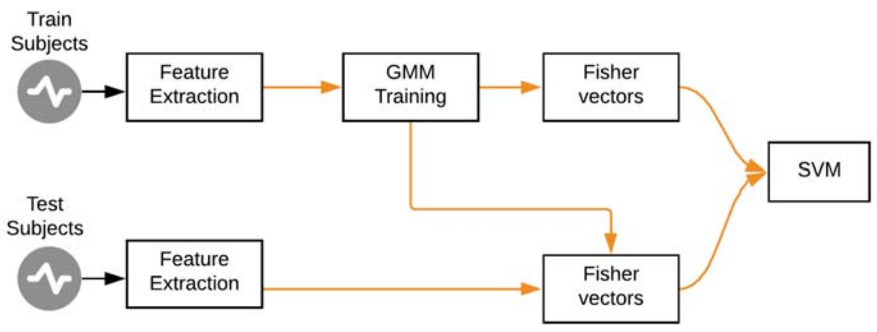


Figure 1: The methodology applied in this study.

The assumption of independence permits the FV to become a sum of normalized gradients statistics $L_\lambda \bigtriangledown_\lambda \log \upsilon_\lambda(x_t)$ calculated for each SIFT descriptor:

$$X_t \rightarrow \varphi_{FK}(X_t) = L_\lambda \bigtriangledown_\lambda \log \upsilon_\lambda(X_t), \tag{6}$$

which describes an operation that can be thought of as a higher dimensional space embedding of the local descriptors X_t.

Hence, the FV approach extracts low-level local patch descriptors from the audio-signals' spectrogram. Then, with the use of a GMM with diagonal covariances we can model the distribution of the extracted features. The log-likelihood gradients of the features modeled by the parameters of such GMM are encoded through the FV [19]. This type of encoding stores the mean and covariance *deviation* vectors of the components k that form the GMM together with the elements of the local feature descriptors. The image is represented by the concatenation of all the mean and the covariance vectors that gives a final vector of length $(2D + 1)N$, for N quantization cells and D dimensional descriptors [16, 19].

The FV approach can be compared with the traditional encoding method: BoV, and with a first order encoding method like Vector of Locally Aggregated Descriptors (VLAD) [1]. In practice, BoV and VLAD are outperformed by FV due to its second order encoding property of storing additional statistics between codewords and local feature descriptors [23].

The FV representation, regardless of the number of local features (i.e. SIFT), or in our case, frame-level features (MFCCs), extracts a *fixed-sized* feature representation from each image (i.e. from each MFCC representation). Here, we use FV features to encode MFCC features extracted from audio-signals of HC and PD subjects. FV allows us to give a complete representation of the sample set by encoding the count of occurrences and higher-order statistics associated with its distribution.

3.3 Classification

Support Vector Machines (SVM) is the classification algorithm used to assess the recordings, it is typically the standard choice for paralinguistics tasks. Moreover, this algorithm can achieve good performances when used with FV [19, 24]. As for the evaluation metric, Unweighted Average Recall (UAR) is the proper way to measure the performance of these kinds of tasks; principally because it is commonly used when there is the need to handle class imbalance situations. Furthermore, this metric has been utilized since the very first ComParE Challenge (see [20] for more details about the UAR evaluation metric).

4 Experimental Setup

The training dataset consists of 9505 utterances, where 8535 (89.8%) are labeled as *healthy* (not-cold) and the rest, 970 (10.2%), are labeled as *cold.* Likewise, the development dataset comprises 1011 *cold* and 8585 *not-cold* labels, which are 10.53% and 89.47%, respectively. Such a high class imbalance is more likely to diminish the performance of the SVM classifier. To overcome this, we used random undersampling which reduces the number of samples associated with all classes to that of the minority class, i.e. *cold.* We relied on imbalanced-learn [15], which is a Python package offering several resampling methods used in datasets that have a between-class imbalance. In our first experiments we reduced the dimensions of the features via Principal Component Analysis (PCA) [13], keeping a variance of 0.95. Chatfield et al. demonstrate that applying PCA before classification enhances the discrimination task with FV and reduces the memory consumption as well [3].

Moreover, the features (Fisher Vectors) were normalized with Power Normalization (PN) and *l2-Normalization.* Power Normalization was found to be helpful for FVs representations [19] as it reduced the impact of the features that become more sparse as the number of Gaussian components increases. In the following experiments, we applied these normalization techniques before reducing the dimensions using PCA. Likewise, we found that *l2-Norm.* helped to alleviate the effect of having different utterances with distinct amounts of background information projected into the extracted features, which attempts to improve the prediction performance.

The GMM used in our experiments to compute the FVs was set to operate with a varying number of components: G_c ranged from 2 to 128. The construction of the Fisher Vector representations was made with the help of a Python-wrapped version of the VLFeat library [25]. As stated before, the classification was done using a Support Vector Machines algorithm. We employed the libSVM implementation [2] with a linear kernel and, as suggested in [12], the C complexity parameter was set in the range $10^{-5}, \ldots, 10^{1}$. In order to search for the best complexity value (C) of the SVM, Stratified Group k-fold Cross Validation (CV) was applied over the training and development sets. This type of CV allowed us to avoid having the same speaker in more than one specific fold, while simultaneously preserving the percentage of samples for each class within each fold.

Table 2: UAR scores obtained when SVM classified the data using Fisher Vectors.

	GMM	**Performance (%)**	
Features	**size**	**Cross-Val**	**Test**
ComParE (BoAW-baseline)	-	64.54%	67.30%
Fisher Vectors	64	63.98%	66.12%
Fisher Vectors + PCA	64	64.72%	67.65%
Fisher Vectors + PN + PCA	64	64.92%	67.81%
ComParE + Fisher Vectors (+PN+PCA)	-	63.01%	70.17%

Finally, we performed *late-fusion* of the best configurations. Namely, the class-wise posterior estimates generated by the SVM algorithm could provide a simple way of classifier combination by taking the mean of two or more posterior vectors.

5 Results

As shown in Table 2, for the baseline we utilized the ComParE functionals (i.e. Bag-of-Audio-Words features) that were originally presented and described in [21]. According to the results outlined in Table 2, these representations achieved an UAR score of 67.3% on the test set. This score was slightly outperformed by two of our configurations: when PCA was applied (67.65%), and when PN was applied along with PCA (67.81%). Table 1 shows the results obtained when using Fisher Vectors with their complete number of features as a function of their reduced number of features. As can be seen, when the classifier learned the *raw* Fisher Vector features it achieved a UAR score of 63.98% in the CV. On the test set the performance was higher (66.12%). PCA proved to be useful here by contributing to a better classification performance in both CV and test phases (64.72% and 67.65%, respectively). However, we found that applying PN before PCA was effective as the CV and test UAR scores increased to 64.92% and 67.81%, respectively. Afterwards, we used the ComParE BoAW [22] feature set posterior probabilities and we combined them with those of the (power-normalized and reduced) Fisher Vectors, that is, we carried out a *late fusion.* The UAR score rose to 70.17% of UAR score on test set, which outperformed the BoAW baseline.

6 Conclusions

In this study, we presented the Fisher Vector approach as a method of classifying speech from subjects having a cold. Compared with studies done by other teams using the same dataset [11, 22], our performance is competitive. Moreover, our feature extraction approach seems to be simpler than that of the mentioned studies as we utilized one single type of feature representation for training a model. We found that SVM gave better results when the feature pre-processing step was applied before executing the training phase. Thus, we demonstrated how applying Power Normalization along with dimension reduction via Principal Component Analysis on the Fisher Vector features improved the classification performance. Combining Power Normalization with PCA gave a better UAR score on test set. These results are higher compared to those got using the Bag-of-Audio-Words approach described in [22]. We can therefore say that PCA with the SVM allowed us to carry out a better classification of the actual data while taking care of the memory consumption. PN helped to reduce the impact of the features that increase their sparsity as the number of Gaussian components increase. Furthermore, L2-normalization was applied before fitting the data. This helped to alleviate the effect of having different utterances with distinct amounts of background information projected into the extracted features, which attempts to improve the prediction performance. In a future study, we will try out the FV approach on bigger datasets and evaluate the performance of a time-delay neural network when it uses them as input features.

References

[1] Arandjelovic, Relja and Zisserman, Andrew. All about VLAD. In *Proceedings of CVPR*, pages 1578–1585, 2013. DOI: `10.1109/CVPR.2013.207`.

[2] Chang, Chih-Chung and Lin, Chih-Jeh. LIBSVM: A library for Support Vector Machines. *ACM Transactions on Intelligent Systems and Technology*, 2:1–27, 2011. DOI: `10.1145/1961189.1961199`.

[3] Chatfield, Ken, Lempitsky, Victor, Vedaldi, Andrea, and Zisserman, Andrew. The devil is in the details: An evaluation of recent feature encoding methods. In *British Machine Vision Conference*, volume 2, pages 76.1–76.12, 11 2011.

[4] Cummins, N., Epps, J., Sethu, V., and Krajewski, J. Variability compensation in small data: Oversampled extraction of i-vectors for the classification of depressed speech. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 970–974, 2014. DOI: `10.1109/ICASSP.2014.6853741`.

[5] Egas-López, José Vicente, Orozco-Arroyave, Juan Rafael, and Gosztolya, Gábor. Assessing Parkinson's Disease From Speech Using Fisher Vectors. *Proceedings of Interspeech*, pages 3063–3067, 2019. DOI: `10.21437/Interspeech.2019-2217`.

[6] Egas-López, José Vicente, Tóth, László, Hoffmann, Ildikó, Kálmán, János, Pákáski, Magdolna, and Gosztolya, Gábor. Assessing Alzheimer's Disease from Speech Using the i-vector Approach. In *Proceedings of SPECOM*, pages 289–298. Springer, 2019. DOI: `10.1007/978-3-030-26061-3_30`.

[7] Gosztolya, Gábor, Bagi, Anita, Szalóki, Szilvia, Szendi, István, and Hoffmann, Ildikó. Identifying schizophrenia based on temporal parameters in spontaneous

speech. In *Proceedings of Interspeech*, pages 3408–3412, Hyderabad, India, Sep 2018. DOI: `10.21437/Interspeech.2018-1079`.

[8] Gosztolya, Gábor, Busa-Fekete, Róbert, Grósz, Tamás, and Tóth, László. DNN-based feature extraction and classifier combination for child-directed speech, cold and snoring identification. In *Proceedings of Interspeech*, pages 3522–3526, Stockholm, Sweden, Aug 2017. DOI: `10.21437/Interspeech.2017-905`.

[9] Gosztolya, Gábor, Grósz, Tamás, Szaszák, György, and Tóth, László. Estimating the sincerity of apologies in speech by DNN rank learning and prosodic analysis. In *Proceedings of Interspeech*, pages 2026–2030, San Francisco, CA, USA, Sep 2016. DOI: `10.21437/Interspeech.2016-956`.

[10] Gosztolya, Gábor, Grósz, Tamás, and Tóth, László. General utterance-level feature extraction for classifying crying sounds, atypical & self-assessed affect and heart beats. In *Proceedings of Interspeech*, pages 531–535, Hyderabad, India, Sep 2018. DOI: `10.21437/Interspeech.2018-1076`.

[11] Huckvale, Mark and Beke, András. It sounds like you have a cold! Testing voice features for the Interspeech 2017 Computational Paralinguistics Cold Challenge. In *Proceedings of Interspeech*, pages 3447–3451. International Speech Communication Association (ISCA), 2017. DOI: `10.21437/Interspeech.2017-1261`.

[12] Jaakkola, Tommi S. and Haussler, David. Exploiting generative models in discriminative classifiers. In *Proceedings of NIPS*, pages 487–493, Denver, CO, USA, 1998.

[13] Jolliffe, Ian. T. *Principal Component Analysis.* Springer-Verlag, 1986. DOI: `10.1007/978-1-4757-1904-8`.

[14] Kaya, Heysem and Karpov, Alexey A. Introducing weighted kernel classifiers for handling imbalanced paralinguistic corpora: Snoring, addressee and cold. In *INTERSPEECH*, pages 3527–3531, 2017. DOI: `10.21437/Interspeech.2017-653`.

[15] Lemaître, Guillaume, Nogueira, Fernando, and Aridas, Christos K. Imbalanced-learn: A Python toolbox to tackle the curse of imbalanced datasets in machine learning. *The Journal of Machine Learning Research*, 18(1):559–563, 2017.

[16] Perronnin, F. and Dance, C. Fisher kernels on visual vocabularies for image categorization. In *Proceedings of CVPR*, 2007. DOI: `10.1109/CVPR.2007.383266`.

[17] Povey, Daniel, Ghoshal, Arnab, Boulianne, Gilles, Burget, Lukáš, Glembek, Ondrej, Goel, Nagendra, Hannemann, Mirko, Motlíček, Petr, Qian, Yanmin,

Schwarz, Petr, Silovský, Jan, Stemmer, Georg, and Vesel, Karel. The Kaldi speech recognition toolkit. *Proceedings of ASRU*, 01 2011.

[18] Rosenberg, Andrew. Classifying skewed data: Importance weighting to optimize average recall. In *Proceedings of Interspeech*, pages 2239–2242, 2012. DOI: `10.21437/Interspeech.2012-131`.

[19] Sánchez, Jorge, Perronnin, Florent, Mensink, Thomas, and Verbeek, Jakob. Image classification with the Fisher Vector: Theory and practice. *International Journal of Computer Vision*, 105:222–245, 2013. DOI: `10.1007/s11263-013-0636-x`.

[20] Schuller, Björn, Batliner, Anton, Steidl, Stefan, and Seppi, Dino. Recognising realistic emotions and affect in speech: State of the art and lessons learnt from the first challenge. *Speech Communication*, 53(9-10):1062–1087, 2011. DOI: `10.1016/j.specom.2011.01.011`.

[21] Schuller, Björn, Steidl, Stefan, Batliner, Anton, Bergelson, Elika, Krajewski, Jarek, Janott, Christoph, Amatuni, Andrei, Casillas, Marisa, Seidl, Amdanda, Soderstrom, Melanie, et al. The Interspeech 2017 computational paralinguistics challenge: Addressee, cold & snoring. In *Proceedings of Interspeech*, pages 3442–3446, 2017. DOI: `10.21437/Interspeech.2017-43`.

[22] Schuller, Björn, Steidl, Stefan, Batliner, Anton, Hantke, Simone, Bergelson, Elika, Krajewski, Jarek, Janott, Christoph, Amatuni, Andrei, Casillas, Marisa, Seidl, Amanda, Soderstrom, Melanie, Warlaumont, Anne S., Hidalgo, Guillermo, Schnieder, Sebastian, Heiser, Clemens, Hohenhorst, Winfried, Herzog, Michael, Schmitt, Maximilian, Qian, Kun, Zhang, Yue, Trigeorgis, George, Tzirakis, Panagiotis, and Zafeiriou, Stefanos. The INTERSPEECH 2017 computational paralinguistics challenge: Addressee, Cold & Snoring. In *Proceedings of Interspeech*, pages 3442–3446, Stockholm, Sweden, Aug 2017.

[23] Seeland, Marco, Rzanny, Michael, Alaqraa, Nedal, Wäldchen, Jana, and Mäder, Patrick. Plant species classification using flower images: A comparative study of local feature representations. *PLOS ONE*, 12(2):1–29, 02 2017. DOI: `10.1371/journal.pone.0170629`.

[24] Smith, David C and Kornelson, Keri A. A comparison of Fisher vectors and Gaussian supervectors for document versus non-document image classification. In *Applications of Digital Image Processing XXXVI*, volume 8856, page 88560N. International Society for Optics and Photonics, 2013. DOI: `10.1117/12.2023329`.

[25] Vedaldi, Andrea and Fulkerson, Brian. VLFeat: an open and portable library of Computer Vision algorithms. In *Proceedings of ACM Multimedia*, pages 1469–1472, 2010.

Acta Cybernetica **25** (2021) 233–255.

A Comparative Study on the Privacy Risks of Face Recognition Libraries*

István Fábián[ab] and Gábor György Gulyás[ac]

Abstract

The rapid development of machine learning and the decreasing costs of computational resources has led to a widespread usage of face recognition. While this technology offers numerous benefits, it also poses new risks. We consider risks related to the processing of face embeddings, which are floating point vectors representing the human face. Previously, we showed that even simple machine learning models are capable of inferring demographic attributes from embeddings, leading to the possibility of re-identification attacks. This paper proposes a new data protection evaluation framework for face recognition, and examines three popular Python libraries for face recognition (OpenCV, Dlib, InsightFace), comparing their face detection performance and inspecting how much risk each library's embeddings pose regarding the aforementioned data leakage. Our experiments were conducted on a balanced face image dataset of different sexes and races, allowing us to discover biases. Based on our results, Dlib has a significant FNR of 4.2% on the total dataset, and an eccentric 5.9% FNR on black people. Finally, our findings indicate that all three libraries could enable sex or race based discrimination in re-identification attacks.

Keywords: face recognition, machine learning, privacy

1 Introduction

With the trend of technology getting cheaper and the advance of smart technologies, security and surveillance cameras are getting more and more widespread recently.

*The research has been supported by the European Union, co-financed by the European Social Fund (EFOP-3.6.2-16-2017-00013, Thematic Fundamental Research Collaborations Grounding Innovation in Informatics and Infocommunications). Project no. FIEK_16-1-2016-0007 has been implemented with the support provided from the National Research, Development and Innovation Fund of Hungary, financed under the Centre for Higher Education and Industrial Cooperation - Research infrastructure development (FIEK_16) funding scheme.

[a]Department of Automation and Applied Informatics, Faculty of Electrical Engineering and Informatics, Budapest University of Technology and Economics, Hungary

[b]E-mail: `fabian@aut.bme.hu`, ORCID: `0000-0003-0293-0335`

[c]E-mail: `gabor.gulyas@aut.bme.hu`, ORCID: `0000-0003-0877-0088`

DOI: `10.14232/actacyb.289662`

According to recent news, Chongqing, a single Chinese city alone has more than 2.5 million surveillance cameras installed [19]. This problem set is not constrained to countries similar to China, as for example London also has more than 600,000 of such cameras [19]. These devices enable emerging artificial intelligence based face recognition technologies in the physical world at scale. This will certainly have a significant impact on the society as a whole, and on the personal level as well, as these advances enable surveillance at large extent as never seen before.

In this paper, we look at the case of the large-scale storing and processing of face imprints generated by face recognition technologies. This technology uses the photo or a video frame containing a person's face to extract an imprint from it. The imprint, or the embedding, describes the face based on its unique characteristics, thus it can be used for identification. When generated by deep learning techniques, the embedding is usually hard for a human to interpret, as usually it is a vector of real values. The length of this vector may vary depending on the used technique.

Identification (i.e. the recognition) works by comparing multiple embedding vectors to each other by calculating similarity between them (e.g. via the Euclidean or Manhattan distance). At the end, pairwise similarities of the embeddings indicate whether the two faces should be considered to be of the same person. It is presumed that the lower the distance, the higher the similarity, and the similarity of embeddings is proportional to the similarity of the faces. Usually if the distance is below a certain threshold, the embeddings are considered to belong to the same person. Or in other words, identification is effectively done by clustering embeddings.

In our research, we are concerned with the possible privacy risks related to utilizing face recognition embeddings. This paper extends our previous work, "On the Privacy Risks of Large-Scale Processing of Face Imprints" [11].

In our previous work we have evaluated a re-identification attack scheme through where we simulated the attacker precision in predicting demographics from embeddings (without executing any machine learning tasks). In our current work, we look in deeper details into these attacks.

We provide a thorough comparison of three popular face recognition Python libraries: OpenCV, Dlib, and InsightFace. We compare these libraries from two different perspectives on people of both sexes, four different races and multiple age groups. First, we consider the face detection performance of these libraries. Then, we consider embedding inference, where we examine how accurately we can train a machine learning model to infer demographic data from the embeddings generated by the libraries.

We also build on results from our previous work in "De-anonymizing Facial Recognition Embeddings"[12] where we showed that re-identification attacks by inferring demographic data from face embeddings are a valid threat (see Figure 1), which justifies the relevance of our current research.

We consider the following setup: cameras observe some areas (for example at a company, or in a public space) and extract facial embeddings of people passing by. Either the cameras themselves are capable of doing the extraction, or they transfer their footage to a capable server device that would do so. Depending on the use case (tracking, authentication, identification, etc.), either embeddings are stored in

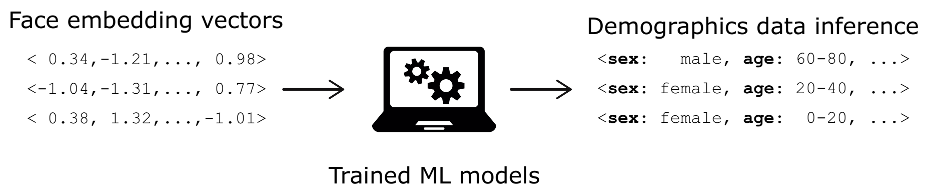

Face embedding vectors
< 0.34,-1.21,..., 0.98>
<-1.04,-1.31,..., 0.77>
< 0.38, 1.32,...,-1.01>
Trained ML models
Demographics data inference
<sex: male, age: 60-80, ...>
<sex: female, age: 20-40, ...>
<sex: female, age: 0-20, ...>

Figure 1: A possible privacy concern regarding face recognition is the inference of sensitive demographic data from face embeddings through inference of specific machine learning models.

a database to be used later on, or are compared in real-time to other embeddings that are already stored in the database.

The reason why the processing may be concerning is that embeddings are considered biometric data and unlike other biometric data such as fingerprints, facial images can be easily captured without a person's knowledge and consent, and also at a large scale [2]. Therefore, in this paper, we look at risks related to the processing of embeddings, more specifically we analyze the privacy risk of demographic-based person re-identification by using face imprints.

This paper is structured as follows. Section 2 summarizes relevant research related to this topic, including how face recognition works and what its privacy concerns are. Section 3 introduces a proposed new data protection evaluation framework for face recognition. Section 4 demonstrates a theoretical attack and evaluates its results. Section 5 compares three popular face recognition libraries and introduces the dataset on which they were tested. Finally, Section 6 concludes the paper with a summary of its main takeaways.

2 Related work

In this section we review facial recognition: its history, how it started, major breakthroughs, and how deep learning based state of the art face recognition systems work. Then, we introduce and discuss the most important features of the three Python libraries we used in our work. At the end of the section we also discuss ethical and privacy concerns related to the application of facial recognition.

2.1 About face recognition

Historically, the dawn of facial recognition began in the 1960s, when researchers began to use computers to recognize human faces [30]. The first trial was a man-machine approach, where human personnel had to manually mark facial landmarks on photographs (e.g. eyes, eyebrows, ears, nose, lips), and the coordinates of these landmarks were then transformed by a computer to undo the effects of variations in head rotation and tilt. Then for each person a list of these coordinates were stored, and in the recognition phase, the distances were calculated between the photograph and all the stored records, and the lowest distance was supposed to reveal the recognized person.

The next major milestone was reached in the 1980s and 1990s, when researchers came up with the eigenfaces approach [31]. The goal of this approach was to be able to represent faces as 1 dimensional vectors (instead of 3 dimensional RGB images), as a combination of predetermined "base" faces, called eigenfaces. The basic idea was to take a facial image dataset, align and center all faces, and create a data matrix by turning the images into vectors. This was followed by calculating the mean face (μ) by averaging the data matrix. The eigenfaces (e) were then constructed by determining the matrix's eigenvectors and reshaping them into images. Afterwards, each new face X could be represented as the mean face plus a linear combination of the eigenfaces: $X = \mu + w_1 \cdot e_1 + w_2 \cdot e_2 + ... + w_n \cdot e_n$, where w_i represents the coefficients of the eigenfaces. In the recognition phase, the similarities between different faces could be determined by calculating a distance (e.g. Euclidean distance) between the coefficients of the eigenfaces belonging to different individuals (where a lower distance meant closer similarity). The biggest advantage of this approach was that it no longer required human manual input, and it was completely automated so it worked even in real-time settings. However, a significant drawback was that it was very sensitive to lightning, scale and facial expression variations, so it could only work in highly controlled environments.

The next breakthrough, which is the current state of the art in face recognition, was made possible by the utilization of deep learning algorithms. These algorithms take the pixels of a photo (or frame) of a person as an input and firstly detect the face in the image. Various techniques can be used for face detection, such as Histogram of Oriented Gradients (HOG) [4][25], Haar-Cascades [32] or even a neural network. Once the face is detected, certain transformations are performed to make it frontal facing and centered, and finally a vector of floating point numbers is generated as an output. These vectors are supposed to describe the human face's unique features.

To create such vectors, a special training setup is needed. Most often, a Siamese network architecture [33] and a special loss function, such as triplet loss [26] is used, where during each iteration of the training three identical networks (hence the name "Siamese" networks) are fed three different face images, two of the same person (the "anchor" and the "positive" image) and one of a different person (the "negative image"). The goal of the training is to modify the weights of the network such that the output embeddings of the anchor and positive images will be close in vector space, while the negative image's embedding will be farther. The advantage of this training setup is that the network can learn to generalize and cluster the same faces together without having to see each possible human face during training.

Then, during recognition phase, these output vectors, also known as face embedding vectors, are compared according to a certain distance metric (e.g. the Euclidean or Manhattan distance) to determine whether two embeddings belong to the same face or not. The length of this vector may differ from implementation to implementation, for example some libraries might generate a 128 dimensional vector [26][21], whereas other libraries generate a 512 dimensional vector [8].

2.2 Face recognition libraries

The three libraries we used in our work are as follows.

The OpenCV library [1] implements a deep convolutional neural network based on the FaceNet [26] structure. Previous networks were trained on a set of known identities and used an intermediate bottleneck-layer to learn a generalized representation of faces for recognition. This setting was inefficient and problematic, because the bottleneck layer couldn't always generalize to new faces, and the representation size of faces were usually thousands of dimensions large. In contrast, FaceNet is an end-to-end solution that directly maps images of faces into the 128-dimensional embedding metric space without requiring a representational bottleneck-layer, using the triplet-based loss function described above. FaceNet was built on two different architectures, the Zeiler Fergus and the GoogLeNet style Inception models. While the Zeiler Fergus model has 140 million parameters, the Inception model has only 7.5 million, making its usage possible on lower computation capacity devices, such as mobile phones.

The Dlib library [21] is based on a ResNet-34 [15] structure deep convolutional neural network. In theory, by increasing the network depth, performance should improve as the model should be able to learn more features. In practice there are, however, obstacles to increasing the depth indefinitely. One obstacle is the problem of the vanishing/exploding gradients, which can be solved by normalized initialization and batch normalization. Another obstacle is that researchers found that adding more layers to a network could actually result in higher training error. The key idea of residual networks such as ResNet-34 is the addition of residual layers to deep convolutional nets. In these models, shortcut connections are added that skip certain layers, performing identity mapping between two non-neighboring layers, thereby not only solving the problem of higher training errors, but actually producing accuracy gains in very deep networks.

The InsightFace [8] library also utilizes a deep convolutional neural network which was based on multiple other networks (ResNet, MobilefaceNet[3], Inception-ResNet_v2 [29], DenseNet [17], etc.) and besides triplet (Euclidean/Angular) loss it also uses multiple loss functions including Additive Angular Margin Loss (ArcFace), which was created with the specific aim to obtain highly discriminative features for face recognition [7]. By maximizing face class separability (i.e. clustering faces belonging to the same person much more closely than other loss functions), this approach enables the network to be less sensitive towards pose and age variations.

The performance of FR libraries is usually tested by benchmarking them on various face image datasets, including the Labeled Faces in the Wild dataset [18], which is the most common benchmarking dataset. From this perspective, Dlib achieves 99.38%, OpenCV achieves 99.63% and InsightFace achieves the highest 99.83% accuracy on this dataset.

2.3 Ethical concerns and privacy risks

While FR technology offers a lot of benefits to humanity and it already has a lot of uses in our everyday lives (e.g. smartphones unlocking by recognizing their owner's face, automatic tagging of people on social networking sites, automated border control gates, finding a lost person, tracking someone etc.) this technology could also pose numerous threats to society.

One of the biggest concerns is that of discrimination. It could be caused not only by face recognition itself, but also by the underlying face detection technology. Some face detection algorithms (like the previously mentioned Haar-Cascades) work by detecting edges, lines and shapes in images. Under certain circumstances (e.g. poor lightning conditions), these techniques work better on light skinned individuals, and perform worse on darker skinned people. A good example of this was when Hewlett-Packard's motion-tracking webcams failed to detect a black person's face [27], but Google Photos also struggled with detecting black persons, mislabeling them for gorillas instead [34]

To analyze the level of discrimination, the Face Recognition Vendor Test conducted by the National Institute of Standards and Technology (NIST) examined the accuracy variations and potential biases across different demographic groups based on sex, age and race [14]. In their study, they examined the performance of 189 face recognition algorithms made by 99 different developers, on over 18 million photographs taken of more than 8 million people. Their report examined the variation between false positives and false negatives for the different demographics analyzed. Overall they found that false positives were much more common than false negatives, and the ratio of false positives was higher among West- and East-African, East-Asian, American-Indian and African-American groups. They also found the false positive ratio to be higher among women, and the youngest and oldest individuals. Considering some of the use cases (e.g. law enforcement usage to identify suspects) these high false positive rates could have a lot of negative consequences on people's lives, like in the case of 3 black men who were mistakenly identified and falsely arrested [16]. Knowing about the existence of these sex and race dependent face recognition performance variations, in our work we examined whether similar demographic biases are also present in the inference of sensitive details from the embeddings. Our results are discussed in Section 5.

Apart discrimination and bias issues, face recognition also poses privacy threats. According to the General Data Protection Regulation (GDPR), face embeddings are biometric data, as the GDPR defines biometric data as *"personal data resulting from specific technical processing relating to the physical, physiological or behavioural characteristics of a natural person, which allow or confirm the unique identification of that natural person, such as facial images or dactyloscopic data"* [10]. As such, processing face embeddings are forbidden by default, and their processing requires special conditions to be met or to have all concerned subject to consent. However, by the nature of video surveillance, consent can be very difficult to obtain; as in public spaces data subjects may not even be aware of being surveilled. Another problematic aspect of processing biometric data is that while it can be in fact used for identification, it should not be used for authentication. Unlike a password, a person's biometric traits are not replaceable and not revocable, which may lead to severe security risks (e.g. biometric data leakage in database hacks). For these reasons, face recognition should be used as a second factor authentication at most, which is not always the case in real world applications.

Privacy threats arise in different shapes and colors in different sectors. By governments in the public sector, there could be misguided use cases that could even threaten democracy as we know it (e.g. mass surveillance using FR in totalitarian regimes, law enforcement usages discriminating certain groups). Risks concerning individuals relate to using FR services on cloud providers that may not respect or protect their data carefully (e.g. Facebook automatic facial recognition on uploaded images posing interdependent privacy risks). In the private sector there may be irresponsible use cases where the nature of biometric face embeddings is not treated with enough caution (e.g. face image or face embedding database leaks, face spoofing attacks, leaking sensitive information via face embeddings, etc).

Due to the numerous privacy harms that could result from the irresponsible usage of facial recognition, in the following Section we introduce a novel data protection evaluation framework that can be used to examine the potential risks in a systematic way.

3 Facial Recognition Data Protection Impact Assessment Framework

In this Section we propose a detailed data protection evaluation framework for facial recognition. Such framework could be a helpful guide in conducting the Data Protection Impact Assessment (DPIA) for applications that utilize face recognition.

Under the GDPR, it could be a mandatory requirement to conduct a DPIA for any case where sensitive data might be published or leaked (e.g. biometric data such as face imprints) [10]. It necessitates the data processor to examine the privacy harms resulting from a potential attack, and to make certain technical and organizational measures so as to minimize the impact of such an attack. As part of the DPIA, the data handler has to evaluate all plausible settings and risk scenarios, so conducting the DPIA is non-linear, cyclic task.

Our proposed framework enables a systematic approach to conduct the DPIA according to GDPR guidelines. To the best of our knowledge, currently no such framework exists specifically for face recognition related data processing. The framework is seen in Figure 2.

The first stage represents the processed data by the data processor. In our case, the data includes face embeddings along with some extra information. This may include sensitive, directly or non-directly identifying personal information for individuals, depending on the concrete use case. (One example could be a camera system at an airport, that could record the embeddings of people entering a prayer room, posing the risk of sensitive information leakage.)

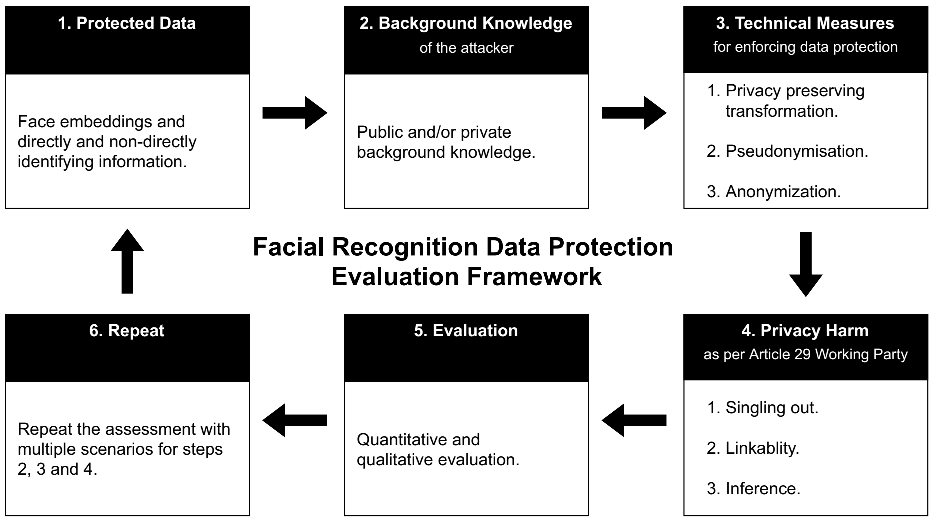

1. Protected Data
Face embeddings and directly and non-directly identifying information.
2. Background Knowledge
of the attacker
Public and/or private background knowledge.
3. Technical Measures
for enforcing data protection
1. Privacy preserving transformation.
2. Pseudonymisation.
3. Anonymization.
Facial Recognition Data Protection Evaluation Framework
4. Privacy Harm
as per Article 29 Working Party
1. Singling out.
2. Linkablity.
3. Inference.
5. Evaluation
Quantitative and qualitative evaluation.
6. Repeat
Repeat the assessment with multiple scenarios for steps 2, 3 and 4.

Figure 2: Our proposed framework for helping to carry out the Data Protection Impact Assessment of facial recognition, as required by the GDPR.

The second stage illustrates the potential background knowledge of the attacker. Depending on the nature of the attack, the attacker might have access to only public or both public and private auxiliary information that she could use for an attack. In case of an outside attack, the hacker could only use publicly available data to carry out a privacy attack (e.g. social media posts and photos, voter registration lists, etc.). However, in case of an inside attack, the malicious actor could have access to protected data that has high overlap with the published or leaked original dataset, thus presents higher risk (e.g. if the attacker is a system administrator).

The third stage details the technical measures that could be taken by the data processor to minimize the privacy harms resulting from attacks. The data processor could apply privacy preserving transformations (e.g. mapping, hashing, data perturbations), pseudonymization (e.g. cryptographic or hashing techniques) and/or anonymization (e.g. k-anonymity) in this step. The point of these measures is to narrow down the possibilities of a malicious party to minimize the impacts of the attack.

The fourth stage discusses the potential privacy attacks by the malicious party as per the GDPR. The Article 29 Working Party determined three different attack types [13]: singling out (the malicious actor successfully identifies an individual in the dataset), linkability (connecting two records of the same individual from different databases) and inference (finding out new information about individuals with high probability).

The fifth stage distinguishes two approaches for evaluating the impact of an attack: quantitative and qualitative approaches. Quantitative approaches take into account the success rate of an attacker, such as the percentage of individuals re-identified, true positive rate, false positive rate, recall and other similar metrics. On the other hand, qualitative approaches deal with the nature of the suffered privacy harm, such as the leakage of sensitive information like sexual, political, religious orientation, behavioral preferences or the revelation of someone's location. These could have moral or material impact on the degree of personal freedom of individuals.

Finally, the sixth stage emphasises the cyclic nature of the DPIA. Namely, the quantitative and qualitative evaluation of the attack must be completed for multiple different scenarios for the assumed background knowledge, technical measures taken, and privacy harm considered.

We believe that the above introduced general framework is a helpful starting point for preparing the DPIA and to analyze numerous different privacy threats. In our work, we considered inference based linkability as the privacy harm, where the attacker uses her background knowledge combined with demographic information inferred from the embeddings to carry out a re-identification attack. The following Section details our work regarding the attack and the estimated risk.

4 Attack and risk level estimation

Previously, we have shown that sex, race and age can be predicted with high accuracy from face embedding vectors [12], but researchers showed that even the original face image can be reconstructed from the embeddings [22], which means that certain types of data that can be determined by looking at a person's face, such as hair color, glasses, etc., are also stored in embeddings. Such traits can be referred to as soft biometric traits [5], which define some information about an individual, but are not distinctive enough to make them uniquely identifiable.

The problem is that personal attributes that are not personally identifiable information yet can be combined together or indirectly merged with external data sources in order to put back the names over de-identified data (i.e. where all directly identifying attributes are removed) [28]. We call such procedures re-identification attacks. Consider an example where a company publishes a database with information about its employees, de-identified by removing explicitly identifying fields (names, email, etc.) and replacing them with unique random IDs. While this database alone might be considered de-identified, but an attacker may link records from this company-related dataset to a medical dataset's corresponding records by using demographic data.

There are several ways how an attacker can be successful at re-identification by using face embeddings:

- By matching embeddings: e.g. the attacker has a photo, extracts an embedding and looks for a match in a database containing embeddings. As mentioned in Section 1, if the distance between the two embeddings is below a threshold then the embeddings belong to the same person with some probability.

- If direct search of embeddings is not possible, the attacker could reconstruct the face from the embeddings in the database [22], and run a visual search in a face database (e.g. photos on a social network).
- Knowing that embeddings contain demographic data about the data subject, the attacker can try reconstructing such data from the stored embedding itself (e.g. using a machine learning model trained for this task) and using that to do cross matching in another database.

As we know that demographic data predictions are feasible, we consider the third class of attacks, which is an inference based linkability attack as per our proposed framework. This is also motivated with the fact that the zip code, sexuality and date of birth combined together provide a unique identifier for 87% of the population based on US census data. [28] Referring back to our framework, if an attacker combines her background knowledge with accurately predicted demographic data from embeddings, and knows further pieces of background information such as place of work or residence, she will be able to look up the identity of the data subject by looking her or him up on social network sites (e.g. on LinkedIn).

Let us explain this concrete attack as follows (see Figure 3). Let us assume a company where the employees are monitored by FR-capable smart CCTVs that store the extracted face embeddings in a central database. If the attacker manages to get the database, she can perform the following attack. In the 1st step the attacker downloads a publicly available face images dataset. In the 2nd step, the attacker labels the downloaded face images with demographic attributes such as sex, race and age (if they are not already labeled by default) and runs FR on them to extract the face embeddings. Afterwards, she trains a machine learning model to classify embeddings into demographic categories according to the training labels. In the 3rd step the attacker deploys the machine learning model, and then in the 4th step she successfully infers the demographic attributes of the people whose embeddings are stored in the stolen database. In the final 5th step the attacker uses this extracted demographic data to re-identify the people on a social network site.

In this Section, we demonstrate this attack in multiple scenarios, based on the number of people in the database and the accuracy of the demographic data prediction algorithms. In Subsection 4.1 we explain how we generated the data for our experiments, and in Subsection 4.2 we describe the results of our experiment.

4.1 Data generation

To determine the feasibility and threat level of the attack, we ran simulations on the UCI Machine Learning Repository's Adult Dataset [9]. This dataset contains demographic information (including age, sex and race) for more than 30,000 records. These records are not of individual people, but of types of individuals, where the *'fnlwgt'* column describes the number of individuals represented by the given record.

As per our attacker model, our aim with the simulations was to examine what level of re-identification is theoretically possible in a database containing people's

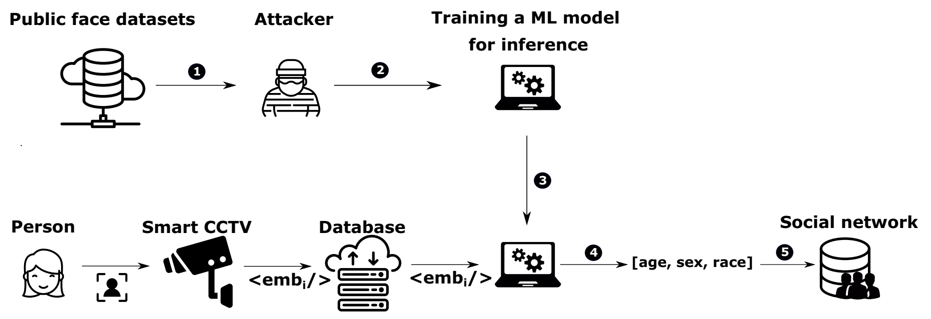


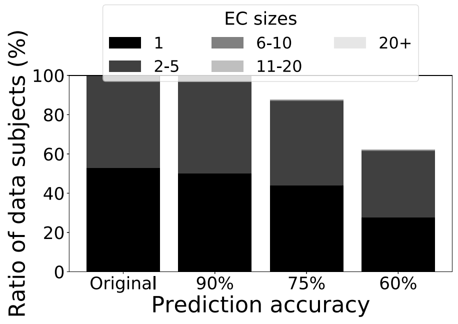
Figure 3: The considered attack when a malicious third party reconstructs demographic data from embeddings and re-identifies the embedding by looking up potential data subjects on social networking sites.

face embeddings. The database sizes were chosen to be reasonable assumptions for the number of employees of a small or medium sized company. To construct the smaller databases of size 10, 50, 100 and 300 for the simulation, we randomly sampled the required number of entries from [9] using the values in the *'fnlwgt'* column as weights, which indicate the number of people represented by a given entry.

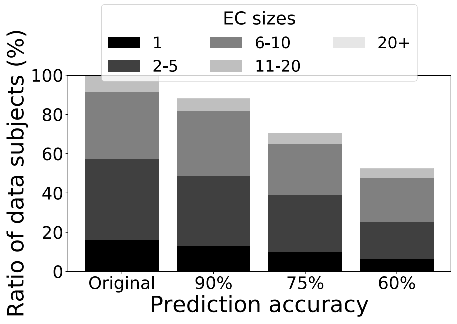
4.2 Evaluation

We ran the experiments by assuming the accuracy for predicting age, race and sex to vary between 60%, 75% and 90% and we assumed a machine learning model that can predict age in 10 year intervals. After creating the smaller databases, some of their rows were left untouched based on the prediction accuracy percentages (60%, 75% and 90%), while the remaining rows' age attributes were randomly permuted to simulate inaccurate predictions. This random permutation was then repeated with the same prediction accuracy percentage for the other two attributes, too (sex and ethnicity). This way we ended up with three derived databases for each smaller database, where all three attributes were simulated to be predicted with either 60%, 75% or 90% accuracy. As the last step, for each predicted database we counted what percentage of data subjects were correctly predicted to fall in an equivalence class of size 1, 2-5, 6-10, 11-20 and 20+ (where the smaller the equivalence class, the higher the risk of re-identification is). We then repeated this procedure 100 times and averaged out the results.

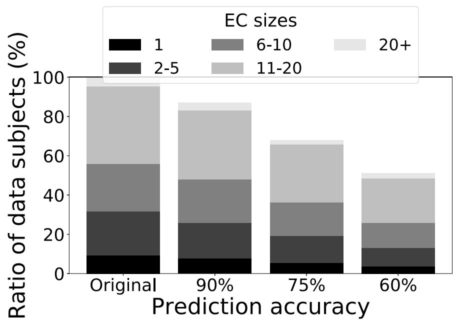
Figures 4(a)–4(d) show our findings. We can observe that there are many records in unique or small equivalence classes both in smaller ($|D| = 10$, $|D| = 50$) and larger ($|D| = 100$, $|D| = 300$) predicted databases, which poses privacy risks. The attacker is the most successful at re-identification in the case of the smallest database of 10 people, with the highest 90% prediction accuracy, when 50.1% of people fall in a unique equivalence class, and all the others fall in an equivalence class of size 2-5. If the accuracy is decreased to 60%, still 27.7% falls in a unique



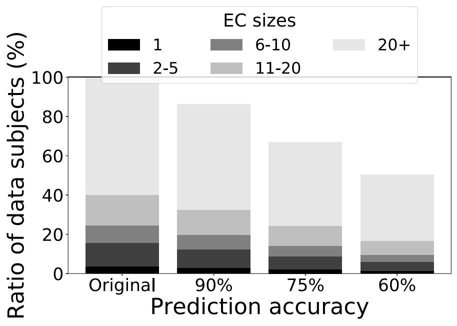
(a) $|D| = 10$



(b) $|D| = 50$



(c) $|D| = 100$



(d) $|D| = 300$

Figure 4: The ratio of equivalence classes (EC) in the predicted database (D) for various database sizes and prediction accuracies.

equivalence class, and 33.9% falls in an equivalence class of size 2-5 (see Figure 4(a)). Regarding the largest database of 300 people, 3.75% of individuals are in a unique equivalence class, and 11.79% are in an equivalence class of size 2-5. Even in the worst case scenario for the attacker, which is 60% accuracy for a database of 300, the rate of people in unique equivalence classes does not fall below 1.38%, nor does the rate of people in an equivalence class of size 2-5 fall below 4.64% (see Figure 4(d)). Also, it is worth noting that while the percentage of people re-identified may be lower in the case of large databases, the expected number of people re-identified may still be higher in these cases. So while an increase in database size and a decrease in prediction accuracy results in a decrease in re-identification probability, the risks are not diminished drastically.

In summary, as expected, the smaller the database size, the higher the re-identification risk is, because smaller sized databases have a higher chance of being reconstructed in such a way that people are correctly mapped to an equivalence class of size 1 or 2-5. Indeed, the higher the prediction accuracy, the higher the re-identification risk is, because the higher percentage of people are predicted to be in the correct equivalence class. As a result, due to the privacy risk presented, the actual achievable prediction accuracy must be examined, which is detailed in Subsection 5.3.

5 Comparison of the state-of-the-art face recognition libraries

5.1 Data generation

We compare three of the most popular open access FR libraries (OpenCV, Dlib and InsightFace) from a face detection, face recognition and face embedding inference point of view, i.e. how accurately can a machine learning model learn to predict demographic data from the embeddings generated by each library. First, we had to generate a dataset of face images. While there are many publicly available face image datasets, for our purposes we needed a dataset that contained photos of a wide diversity of people: people from both sexes, from four different races and from multiple age groups.

To generate our own dataset, we used the publicly available UTKFace dataset [35], which is a large-scale face dataset that met our requirements, because it contains over 20,000 face images with annotations of age, sex and race. Moreover, the images are labeled with file names formatted like *[age]_[gender]_[race]_[date&time]*, where *age* is an integer from 0 to 116, *sex* is 0 for males or 1 for females, and *race* is 0 for whites, 1 for blacks, 2 for asians, 3 for indians or 4 for other races.

While this dataset was a great starting point for our research, it was not perfect, because we needed a more balanced dataset. As a result, we only used 12192 photos from UTKFace, since there were only 1524 photos per each of the eight race-sex pairs that we worked with (males and females paired with whites, blacks, asians and indians). Of course, some classes (e.g.: white males) had more than 1524 photos, but due to our need for a balanced dataset, we had to choose the number of photos per class based on the least represented class. Even though this subset of UTKFace was balanced regarding sex and race, it still was not balanced regarding age. For example people aged between 20 and 40 were overrepresented, while people aged over 50 were underrepresented, etc. (see Figure 5 for more details regarding the age distribution of our dataset).

5.2 Face detection and recognition

To compare the three libraries, we ran their face recognition algorithms on our dataset. We then examined how many faces each library found out of the 12192, along with the number of false negatives (where a library mistakenly did not find a face in an image) and false positives (where a library mistakenly found multiple faces instead of just one). To gain a better understanding of the accuracy of each library on different races and sexes, we also examined the races and sexes where these false negatives or false positives occurred. Table 1 shows our findings.

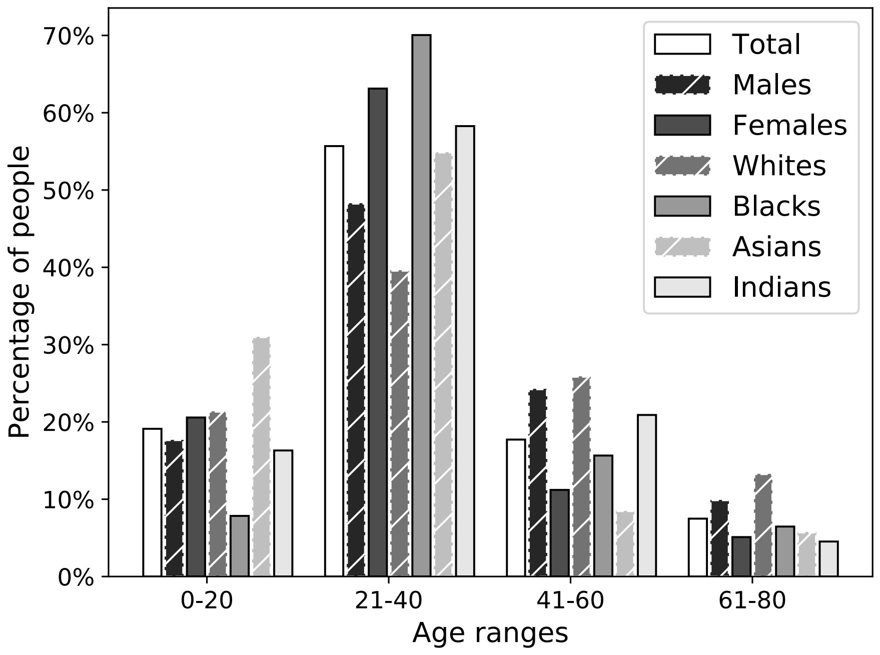


Figure 5: The age distribution of our dataset regarding both sexes and all four examined races

In conclusion, OpenCV and InsightFace performed mostly the same on both sexes and all four races examined, as both libraries had a negligible number of false negatives and false positives. The only difference is the runtime, where OpenCV was about 6 times faster. While Dlib produced zero false positives, it produced a significant false negative rate of 4.2%, which means it is a bit less reliable at detecting people, especially, but not exclusively, black males and females who had a false negative rate of 7.2% and 4.6%. Also, Dlib had the longest runtime, with 4110 seconds, it was about 12 times slower than OpenCV, and 2 times slower than InsightFace.

It is important to note that our tests were conducted on a dataset of cropped face images as opposed to regular face datasets (e.g. "faces in the wild" [18]). So while these results might indicate that there is almost no difference in the false positive rate of the libraries, but due to the nature of our dataset even a very small false positive rate is significant (e.g. in real world conditions OpenCV produces far more false alarms than Dlib [20]). Therefore it is future work to run these experiments on a dataset of non-cropped images "in the wild", too.

In summary, the choice of the right algorithm depends on the use case where facial recognition is applied. When having no false alerts is a significant issue, Dlib is the right choice. When having no false negatives is important, it is better

Table 1: The face detection performance of OpenCV, Dlib and InsightFace on different sexes and races

Lib	**Faces detected**	**Sex**	**Race**	**False pos.**	**Total false pos.**	**False neg.**	**Total false neg.**	**Run time [s]**
Open CV	12183	Male	White	2	3	1	4	322
			Black	1		3		
			Asian	0		0		
			Indian	0		0		
		Female	White	0	2	0	0	
			Black	0		0		
			Asian	1		0		
			Indian	1		0		
Dlib	11676	Male	White	0	0	48	282	4110
			Black	0		110		
			Asian	0		70		
			Indian	0		54		
		Female	White	0	0	51	234	
			Black	0		70		
			Asian	0		62		
			Indian	0		51		
Insight Face	12185	Male	White	0	2	1	4	1858
			Black	1		3		
			Asian	1		0		
			Indian	0		0		
		Female	White	1	1	0	0	
			Black	0		0		
			Asian	0		0		
			Indian	0		0		

to choose another library to avoid situations where some of the consumers could be negatively impacted, such as the previously mentioned incident of Hewlett-Packard's motion-tracking webcams not working on black people [27]. In other cases we may choose between the libraries by considering runtime or the accuracy of additional features. For instance, InsightFace does a great job in detecting facial landmark points, especial when the face is visible from the side profile [6].

5.3 Demographic attribute inference from embeddings

Lastly, we compared each library in terms of how accurately a machine learning model can predict demographic data (sex, race and age) from the face embeddings they produce. The training data was generated by running each library's FR algorithm on our dataset and collecting the face embedding vectors with their corresponding class labels into *pandas* [23] dataframes, where the labels were deduced from the image file names. Since not all faces were detected in all images by all libraries and since we wanted to train our models on balanced datasets, we had to discard some images in order to always use only as many images per each class as the least represented class permitted (i.e. the class with the lowest number of faces detected).

In total, we built three predictive models per each library, one for sex classification, one for race classification and one for age classification. We used Scikit-Learn's [24] *train_test_split* function to split our dataframes into a train and a test set, and then used Scikit-Learn's *RandomForestClassifier* module to train three random forest classifiers to predict the demographic attributes from the face embeddings. The reason we used random forests was that we wanted to show that even easy to use "off the shelf" ML models can work that do not require deep expertise in ML from an attacker. Random forests satisfy the latter criteria by having a small number of hyperparameters to tune. In the case of age prediction, expecting exact accuracy is not realistic (as it is also difficult for a human to guess the age that precisely), so instead we applied the predictions into ranges between 1-20, 21-40, 41-60 and 61-80 years. Figures 6, 7, 8 show our findings. To evaluate our results, we calculated the prediction F1 score, which is a descriptive metric that takes into consideration the true positive (TP), false positive (FP) and false negative (FN) rates as well: $F1 = \frac{TP}{(TP+\frac{1}{2}\cdot(FP+FN))}$.

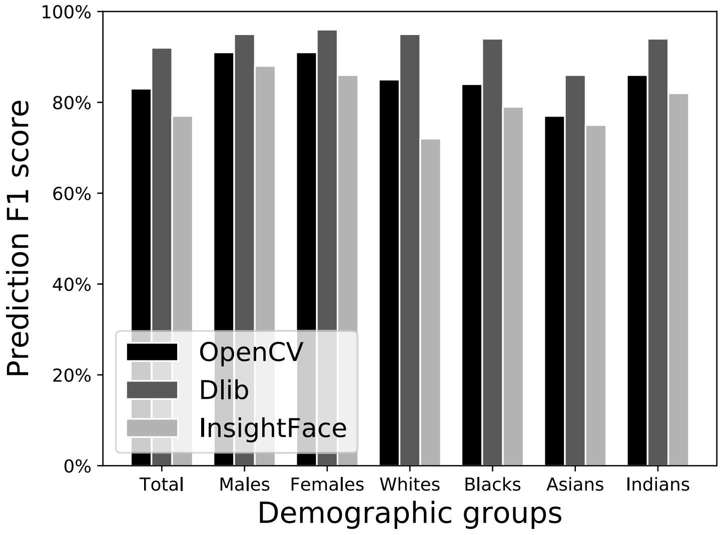


Figure 6: Prediction accuracies for different demographic groups using face embeddings generated by OpenCV, Dlib and InsightFace: Sex prediction

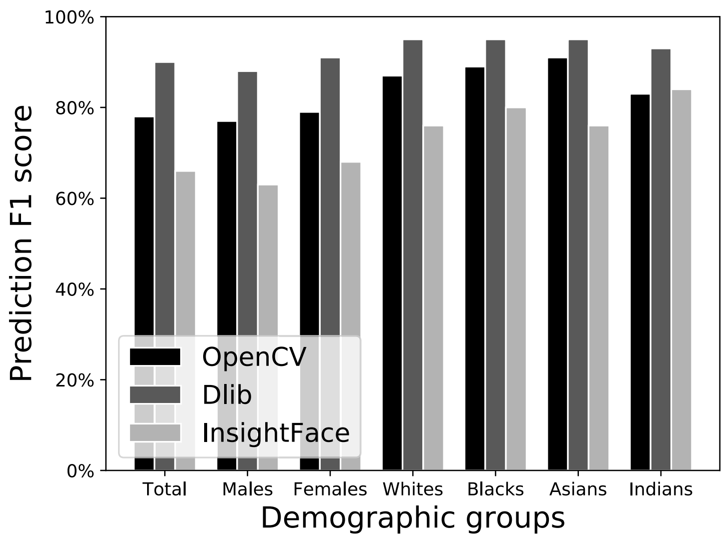


Figure 7: Prediction accuracies for different demographic groups using face embeddings generated by OpenCV, Dlib and InsightFace: Race prediction

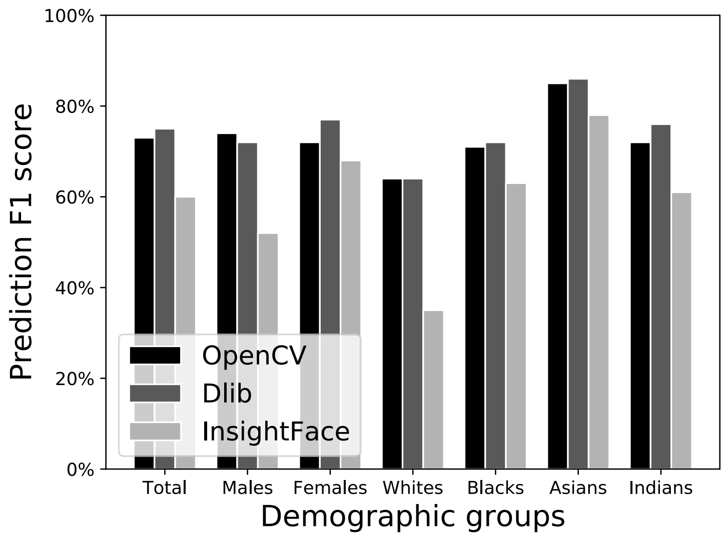


Figure 8: Prediction accuracies for different demographic groups using face embeddings generated by OpenCV, Dlib and InsightFace: Age prediction

In conclusion, our random forest models performed the best on the embeddings generated by Dlib, where the sex classifier achieved over 92%, the race classifier over 89%, and the age classifier over 75% prediction F1 score. The second best performance was achieved when the models were trained and tested on the embeddings generated by OpenCV, where the sex classifier achieved over 83%, the race classifier 78%, and the age classifier over 73% prediction F1 score. The random forest models performed the worst when trained and tested on the embeddings of InsightFace, in which case the sex classifier achieved only over 77%, the race classifier only over 66%, and the age classifier only 60% prediction F1 score.

To test for potential biases, we examined the prediction performance not only for the total population of our dataset, but also on the following smaller demographic groups: males, females, whites, blacks, asian, indians. The performance of the classifiers on these demographic subgroups were mostly uniform, with only a few outliers. While some of the reported differences are very slim, even these could have notable privacy implications as discussed later.

In the case of OpenCV embeddings, the sex classifier performed considerably worse in case of asians than any other race. The race classifier, however, performed the best for asians, and notably worse for indians. The age classifier's performance was significantly worse for white people, but significantly better for asian people. The race prediction performed slightly better for females, whereas the age prediction slightly better for males.

In the case of Dlib embeddings, the sex classifier also achieved a noticeably worse score on asians compared to all other demographic groups. While there was only a very slight difference, but the race classifier achieved the lowest score on the indian population. The age classifier performed the worst on white people, while it performed by far the best on asian people. Regarding sexes, both the race and age predictor performed notably better for females.

In case of the embeddings of InsightFace, the results were a bit different. The sex classifier performed worse than average on whites, and better than average on indians. The race classifier also achieved better than average score for indians, and the lowest score on asians and whites. In the case of age prediction, the most extreme outlier was the much lower score for white people, while the score of asians was also significantly higher than average. In this case, the sex predictor performed slightly better for males, however the race and age prediction was significantly better for females.

Based on these results it seems that there could be noteworthy differences in predicting demographic attributes for different sexes and races. While the impact of these differences may not be significant in all applications (e.g.: targeted advertising in retail), in other scenarios they could have a profound effect on people's lives (e.g.: mass surveillance, law enforcement profiling). Another important aspect to consider is how many people will be affected by the technology in each use case. For example applications in the public sector (e.g.: surveillance by governments) will impact far more people than typical use cases in the private sector (e.g. employee tracking), and in those cases even seemingly small differences of 0.5-1% can affect thousands or tens of thousands of people, which emphasizes the importance of treating facial recognition technology with great caution.

6 Conclusion and future work

In this paper we have reviewed the main principles behind facial recognition algorithms, introduced three popular Python libraries, and presented the potential discriminational and privacy risks in relation to the processing of face embeddings. We have discussed why face embeddings must be considered sensitive biometric data and we proposed a novel data protection evaluation framework for facial recognition, which could be a general starting point for conducting the DPIA required by the GDPR. We have also looked at various attacker models that could pose a threat to data subjects' privacy via inference based re-identification.

In particular, we analyzed the risks of re-identification by reverse-engineering demographic data (age, sexuality, race) from embeddings stored in a database. We found that the smaller the database and the higher the accuracy of prediction, the higher the re-identification risks are. In the case of a 10 person database and 90% accuracy, 50.1% of people are likely to be precisely re-identified, while this number decreased to 27.7% at 60% prediction accuracy. The risks are also not negligible even for larger databases, because for a database of 300 we showed that at 90% accuracy 3.75% of people are in a unique equivalence class, and 11.79% are in an equivalence class of size 2 to 5 and are likely to be de-anonymized. It must be noted that while the re-identification percentages decrease for larger databases, the absolute number of successful re-identification cases increase.

Afterwards, we compared the performance of three face recognition Python libraries (OpenCV, Dlib, InsightFace) on a custom face image dataset that we have generated. Our findings indicate that while all three libraries produce a negligible number of false positives, Dlib produces far more false negatives than the other two, especially for black people. Regarding run time, OpenCV is about 12 times faster and InsightFace is about 2 times faster than Dlib.

Finally, we extracted face embeddings from our custom dataset using all three libraries to then train random forest classifiers to predict sex, race and age from each library's embeddings. We then compared the prediction accuracies of our random forest models on the total dataset and also on demographic subgroups. Our findings indicate that those models perform the best that were trained and tested on the embeddings of Dlib, followed by the embeddings of OpenCV and finally InsightFace.

Based on our results, there can be differences in prediction accuracies between different sexes and races, and the impact of these biases always has to be evaluated in each application scenario (e.g. law enforcement profiling vs. retail profiling).

For future work, our aim is to gain better understanding and greater explainability of the inner workings of our models in order to discover why misclassifications happen and how demographic data is encoded in embeddings. Also, our focus is to design a procedure that could prevent demographic data leakage from the stored embeddings.

While one obvious approach could be to encrypt the embeddings before storage, they would necessarily have to be decrypted for the calculation of Euclidean or Manhattan distances, so it wouldn't permanently solve the leakage problem. Another plausible solution would be to use homomorphic encryption, which would allow operations to be performed on the embeddings in encrypted form, but due to its computational complexity and slow performance its usage might not be feasible in real-time applications.

Therefore, our research aims find a solution (e.g. adversarial search techniques) to modify the embeddings in such a way to notably lower the prediction accuracies by machine learning models for all demographic and sensitive attributes, without compromising the usability of the face embeddings (i.e.: without significantly changing their relative Euclidean distances). Our hope is that achieving this will allow a much more privacy friendly way to utilize face recognition and process face embeddings.

Acknowledgments

The authors would also like to thank Kenéz Csiktusnádi-Kiss for his work and support in this research.

Icons made by Pixel perfect, catkuro, Eucalyp, fjstudio, Freepik, Pause08, surang, xnimrodx, bimbimkha from `www.flaticon.com`.

References

[1] Bradski, G. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 25:120–125, 2000.

[2] Castelluccia, Claude and Le Métayer Inria, Daniel. Impact analysis of facial recognition. Working paper or preprint, URL: `https://hal.inria.fr/hal-02480647`, February 2020.

[3] Chen, Sheng, Liu, Yang, Gao, Xiang, and Han, Zhen. MobileFaceNets: Efficient CNNs for accurate real-time face verification on mobile devices. *ArXiv*, abs/1804.07573, April 2018.

[4] Dalal, N. and Triggs, B. Histograms of oriented gradients for human detection. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 886–893, 2005. DOI: `10.1109/CVPR.2005.177`.

[5] Dantcheva, Antitza, Elia, Petros, and Ross, Arun. What else does your biometric data reveal? A survey on soft biometrics. *IEEE Transactions on Information Forensics and Security*, 11, 2015. DOI: `10.1109/TIFS.2015.2480381`.

[6] Deng, Jiankang. Video face recognition demo of ArcFace, 2018. URL: `https://www.youtube.com/watch?v=y-D1tReryGA&ab_channel=JiankangDeng`.

[7] Deng, Jiankang, Guo, Jia, Xue, Niannan, and Zafeiriou, Stefanos. ArcFace: Additive angular margin loss for deep face recognition. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4685–4694, 2019. DOI: `10.1109/CVPR.2019.00482`.

[8] Deng, Jiankang, Guo, Jia, Yuxiang, Zhou, Yu, Jinke, Kotsia, Irene, and Zafeiriou, Stefanos. RetinaFace: Single-stage dense face localisation in the wild. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5203–5212, June 2020.

[9] Dua, Dheeru and Graff, Casey. UCI machine learning repository, 2019. URL: `http://archive.ics.uci.edu/ml`.

[10] European Parliament and of the Council. Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 april 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing directive 95/46/EC (General Data Protection Regulation), 2016. URL: `https://eur-lex.europa.eu/eli/reg/2016/679/oj`.

[11] Fábián, István and Gulyás, Gábor György. On the privacy risks of large-scale processing of face imprints. In *The 12th Conference of PhD Students in Computer Science*, 2020. `https://www.inf.u-szeged.hu/~cscs/proceedings.php`.

[12] Fábián, István and Gulyás, Gábor. De-anonymizing facial recognition embeddings. *Infocommunications Journal*, 12:50–56, 2020. DOI: `10.36244/ICJ.2020.2.7`.

[13] GDPR. Article 29 Data Protection Working Party, opinion 05/2014 on anonymisation techniques, 2014. URL: `https://ec.europa.eu/justice/article-29/documentation/opinion-recommendation/files/2014/wp216_en.pdf`.

[14] Grother, P., Ngan, M., Hanaoka, K., and National Institute of Standards and Technology (U.S.). *Face Recognition Vendor Test (FVRT): Part 3, Demographic Effects*. NIST interagency report. National Institute of Standards and Technology, 2019.

[15] He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. DOI: `10.1109/cvpr.2016.90`.

[16] Hill, Kashmir. Another arrest, and jail time, due to a bad facial recognition match. *The New York Times*, 2020. URL: `https://www.nytimes.com/2020/12/29/technology/facial-recognition-misidentify-jail.html`.

[17] Huang, Gao, Liu, Zhuang, Van Der Maaten, Laurens, and Weinberger, Kilian Q. Densely connected convolutional networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2261–2269, 2017. DOI: `10.1109/CVPR.2017.243`.

[18] Huang, Gary B., Ramesh, Manu, Berg, Tamara, and Learned-Miller, Erik. Labeled faces in the wild: A database for studying face recognition in unconstrained environments. Technical Report 07-49, University of Massachusetts, Amherst, October 2007.

[19] Keegan, Matthew. Big brother is watching: Chinese city with 2.6m cameras is world's most heavily surveilled. *The Guardian*, 2019. URL: `https://www.theguardian.com/cities/2019/dec/02/big-brother-is-watching-chinese-city-with-26m-cameras-is-worlds-most-heavily-surveilled`.

[20] King, Davis. dlib vs OpenCV face detection, 2014. URL: `https://www.youtube.com/watch?v=LsK0hzcEyHI`.

[21] King, Davis E. Dlib-ml: A machine learning toolkit. *Journal of Machine Learning Research*, 10(60):1755–1758, December 2009.

[22] Mai, Guangcan, Cao, Kai, Yuen, Pong C., and Jain, Anil K. On the reconstruction of face images from deep face templates. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41(5):1188–1202, May 2019. DOI: `10.1109/tpami.2018.2827389`.

[23] McKinney, Wes. Data structures for statistical computing in python. *Proceedings of the 9th Python in Science Conference*, 445:51–56, 2010. DOI: `10.25080/majora-92bf1922-00a`.

[24] Pedregosa, Fabian, Varoquaux, Gaël, Gramfort, Alexandre, Michel, Vincent, Thirion, Bertrand, Grisel, Olivier, Blondel, Mathieu, Prettenhofer, Peter, Weiss, Ron, Dubourg, Vincent, et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12(85):2825–2830, 2011.

[25] Ramirez Cerna, Lourdes, Camara-Chavez, Guillermo, and Menotti Gomes, David. Face detection: Histogram of oriented gradients and bag of feature method. In *Proceedings of the 2013 International Conference on Image Processing, Computer Vision, and Pattern Recognition (IPCV'13)*, 2013. `https://worldcomp-proceedings.com/proc/p2013/IPC.html`.

[26] Schroff, Florian, Kalenichenko, Dmitry, and Philbin, James. FaceNet: A unified embedding for face recognition and clustering. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, Jun 2015. DOI: `10.1109/cvpr.2015.7298682`.

[27] Simon, Mallory. HP looking into claim webcams can't see black people. *CNN*, 2009. URL: `https://edition.cnn.com/2009/TECH/12/22/hp.webcams/index.html`.

[28] Sweeney, Latanya. Simple demographics often identify people uniquely. *Carnegie Mellon University, Data Privacy*, 2000. DOI: `https://doi.org/10.1184/R1/6625769.v1`.

[29] Szegedy, Christian, Ioffe, Sergey, Vanhoucke, Vincent, and Alemi, Alexander A. Inception-v4, Inception-ResNet and the impact of residual connections on learning. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, AAAI'17, pages 4278–4284. AAAI Press, 2017.

[30] Thorat, S. B., Nayak, S. K., and Dandale, Jyoti P. Facial recognition technology: An analysis with scope in India. *ArXiv*, abs/1005.4263, 2010.

[31] Turk, M.A. and Pentland, A.P. Face recognition using eigenfaces. In *Proc. IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 586–591, 1991. DOI: `10.1109/CVPR.1991.139758`.

[32] Viola, P. and Jones, M. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1, 2001. DOI: `10.1109/CVPR.2001.990517`.

[33] Wu, Haoran, Xu, Zhiyong, Zhang, Jianlin, Yan, Wei, and Ma, Xiao. Face recognition based on convolution siamese networks. In *10th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)*, pages 1–5, 2017. DOI: `10.1109/CISP-BMEI.2017.8302003`.

[34] Zhang, Maggie. Google Photos tags two African-Americans as gorillas through facial recognition software. *Forbes*, 2015. URL: `https://www.forbes.com/sites/mzhang/2015/07/01/google-photos-tags-two-african-americans-as-gorillas-through-facial-recognition-software/`.

[35] Zhang Zhifei, Song, Yang and Qi, Hairong. Age progression/regression by conditional adversarial autoencoder. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2017. DOI: `10.1109/cvpr.2017.463`.

Acta Cybernetica **25** (2021) 257–269.

Speech De-identification with Deep Neural Networks[*]

Ádám Fodor[a], László Kopácsi[a], Zoltán Á. Milacski[b], and András Lőrincz[c]

Abstract

Cloud-based speech services are powerful practical tools but the privacy of the speakers raises important legal concerns when exposed to the Internet. We propose a deep neural network solution that removes personal characteristics from human speech by converting it to the voice of a Text-to-Speech (TTS) system before sending the utterance to the cloud. The network learns to transcode sequences of vocoder parameters, delta and delta-delta features of human speech to those of the TTS engine. We evaluated several TTS systems, vocoders and audio alignment techniques. We measured the performance of our method by (i) comparing the result of speech recognition on the de-identified utterances with the original texts, (ii) computing the Mel-Cepstral Distortion of the aligned TTS and the transcoded sequences, and (iii) questioning human participants in A-not-B, 2AFC and 6AFC (Alternative Forced-Choice) tasks. Our approach achieves the level required by diverse applications.

Keywords: speech processing, voice conversion, deep neural network, text-to-speech, speaker privacy

[*]The research has been supported by the European Union, the Ministry of Innovation and Technology NRDI Office within the framework of the Artificial Intelligence National Laboratory Program and by the National Research, Development and Innovation Fund of Hungary, financed under the Thematic Excellence Programme no. 2020-4.1.1.-TKP2020 (National Challenges Subprogramme) funding scheme through the "Application Domain Specific Highly Reliable IT Solutions" project and co-financed by the European Social Fund (EFOP-3.6.3-16-2017-00002, EFOP-3.6.3-VEKOP-16-2017-00001).

[a]Equal contributions. Department of Artificial Intelligence, Eötvös Loránd University, Budapest, Hungary, E-mail: {foauaai, kopacsi}@inf.elte.hu, ORCIDs: 0000-0001-7370-930X and 0000-0003-2387-2015

[b]Department of Artificial Intelligence, Eötvös Loránd University, Budapest, Hungary, E-mail: miztaai@inf.elte.hu, ORCID: 0000-0002-3135-2936

[c]Corresponding author, Department of Artificial Intelligence, Eötvös Loránd University, Budapest, Hungary, E-mail: lorincz@inf.elte.hu, ORCID: 0000-0002-1280-3447

DOI: 10.14232/actacyb.288282

1 Introduction

Cloud-based speech services have improved recently due to the large amount of voice data that is exploited by deep learning technology [1, 3], giving rise to superhuman performance in several tasks. Consequently, it seems reasonable to use such utilities in practice.

Unfortunately, many speech applications involve legal concerns regarding privacy. Several methods have been proposed to eliminate personal information from samples without spoiling the linguistic content before uploading. We should also mention, that in many cases the private information is carried by the linguistic content and not by the voice of the speaker. For example, when a doctor dictates medical records, the private content is the medical content and not the identity of the doctor. But in the case of diagnostic sessions with autistic people, it is the speaker whose identity should remain hidden. If an external ASR is used on the transformed speech of a patient, the identity will remain concealed, and the linguistic content can be generated safely.

Voice conversion (VC) operates by altering certain features of human speech [31]. Voice transformation (VT) converts the signal as if it was uttered by a target speaker [23]. De-identification is the process that intends to remove any personal information from the data that could be associated with identity. VC and VT may be applied to solve de-identification, but the papers in the literature suffer from several flaws: the VC algorithm in [22] is approximately invertible and relies on a good voice transformer, while VT [23, 29] requires data from pairs of speakers and is unable to anonymize the target speaker.

Our contributions are as follows. For de-identification, we propose to transform utterances to a generic voice of a Text-to-Speech (TTS) engine, by taking advantage of utterance-text sample pairs. We use an end-to-end trainable Deep Neural Network (DNN) to learn the many-to-one VT task. We suggest to learn the mapping at vocoder level. We show that the trained network gives rise to tolerable distortions at utterance level by conducting two experiments: comparing the outputs of Google's Automatic Speech Recognition (ASR) system for the original TTS output and the de-identified utterance and measuring the Mel-Cepstral Distortion (MCD) [19]. To confirm de-identification success, we further performed three kind of perceptual listening studies with human subjects (A-not-B test: distinguishing transformed utterances of different speakers, 2-Alternative Forced-Choice (2AFC) test: classifying utterances from female/male speakers, and 6-Alternative Forced-Choice (6AFC) test: estimating the number of speakers). Our proposal is irreversible and it requires only speech-transcript sample pairs for training, which are readily accessible in the literature. We argue that our method performs favorably compared to several baseline methods.

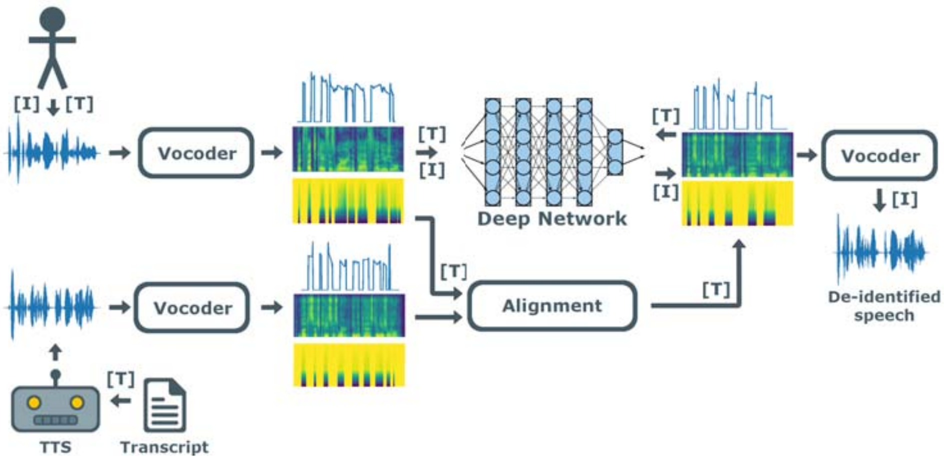


Figure 1: Schematic diagram of our proposed method. Training [T]: vocoded human voice is input to a Deep Neural Network (DNN) that is trained to approximate the aligned TTS output. Inference [I]: vocoded human voice is de-identified by the DNN and transformed back to utterances by the vocoder.

2 Related work

De-identification can be solved by either VC or VT methods. A subset of the literature focuses on classical algorithms instead of leveraging the potential of DNN architectures, and hence fail to produce state-of-the-art speech quality. The so-called transterpolation VC technique gave rise to significant improvements over diverse VC methods as reviewed in [16]. Another VC approach exploits the two-step procedure of an ASR system followed by a TTS [17]. However, due to the method of the conversion, the latter cannot take advantage of the superior performance of cloud-based ASR systems. A potential problem of VT methods is that the target speaker is not generic and hence it cannot be anonymized. Generic is meant here as monotonic and “robotic”, which does not contain any prosody. Neural TTS systems nowadays are realistic enough, that it may include unintended prosody, which makes the transformation harder. The problem can be resolved by converting to an average voice, however, we decided to go with a TTS system instead of generating an average voice. In addition, VT methods need speech corpora of the original speaker and the target speaker, too. To avoid the need for a parallel corpora an approach that used a pool of pre-trained transformations between a set of speakers was put forth in [22]. A transformation function was applied to the source and the target speakers based on speaker similarity and dissimilarity, respectively. By applying several sound distortion algorithms de-identification was achieved and the transformation could be reversed, offering several advantages at the cost of vulnerability.

In contrast, several recent works propose DNNs for many-to-one VC and VT.

For an overview, see [24]. A VC method using Mel-Cepstral inputs for deep autoencoders was introduced in [23]. Speaker-dependent Conditional Restricted Boltzmann Machine (CRBM) was applied using Mel-Frequency Cepstral Coefficients (MFCCs) and deltas for solving the VC task for each speaker pair in [29]. An autoencoder-based VT approach was proposed to reduce the required size of the data sets and to shorten conversion time in [32]. A VT method that generates a one-to-one speaker-dependent DNN using the weights of a speaker-independent DNN was suggested in [21]. Spectral envelope, fundamental frequency ($F0$), intensity trajectory and phone duration were converted in [30] subject to an ℓ_1 norm constraint during pre-training. Nevertheless, all of these methods restrict themselves to the case of VC and VT, without using transcript data. In this paper, we directly tackle de-identification and propose to use textual data as well besides the original speech for training, which are largely available online.

3 Proposed method

We describe the features set, the pre-processing steps and the DNN architectures in detail here. The de-identification pipeline can be seen in Fig. 1. To differentiate processes, training and inference are marked with "T" and "I", respectively.

3.1 Measures

In speech recognition, the standard measure is the word error rate (WER), defined as the edit distance between the true word sequence and the most probable word sequence emitted by the transcriber. However, it is not ideal for short sequences. In cases of 1-word sequences, where the transcriber recognizes the expected word, but mistakes 1-2 letters, the measure is 0. This is not representative enough, that is why the Levenshtein distance is used instead.

Levenshtein distance is a measure of the similarity between two strings, which we will refer to as the S source string and the T target string. The distance between two words is the minimum number of single-character edits (insertions, deletions or substitutions) required to transform S into T. This measure is also called character error rate (CER) or letter error rate (LER). Letter accuracy rate (LAR) will be used in this document later on:

$$LAR(S,T) = 1 - LER(S,T) \tag{1}$$

3.2 Feature extraction

In order to find the best feature set, we compared multiple vocoder systems in terms of speech synthesis quality (Ahocoder [7], MagPhase [8], PulseModel [6], WORLD [28] and STRAIGHT [18]) using the TIMIT [35] test set. We took each sample and extracted the vocoder parameters from them. We also experimented with converting the spectral parameters into a mel-cepstral representation. Following this, we re-synthesized the samples, and measured the mean Letter Accuracy Rate (LAR) values between the predicted transcript of Google Cloud Speech-to-Text system and the transcripts provided with the TIMIT corpus.

During our experiments, we observed that using mel-cepstral representation during encoding produced more favorable results.

The LAR of the test set was 97%. By applying vocoder systems the LAR was barely affected, in every case, the relative degradation was less than 2%. In subsequent sections, we used the python wrapper of WORLD vocoder called PyWorld, because of its low computational requirements, continuous support and easy usability. The extracted features are the estimation of the fundamental frequency ($F0$), spectral envelope and aperiodicity.

$F0$ is the fundamental frequency of the vibration of our vocal folds. We perceive it as pitch. The $F0$ contour is estimated with DIO [27]. To improve the noise robustness of DIO, we also applied StoneMask pitch refinement algorithm.

Let us introduce x_n the subsampled audio signal, and X_k the frequency spectrum [15], which is the discrete Fourier transform (DFT) of a signal defined for $k = 0, 1, ..., N - 1$.

We can compute the *magnitude spectrum* M_k, the *phase spectrum* Φ_k and the *power spectrum* P_k using the following equations:

$$M_k = |X_k| \tag{2}$$

$$\Phi_k = \arctan \left| \frac{Re(X_k)}{Im(X_k)} \right| \tag{3}$$

$$P_k = Re(X_k)^2 + Im(X_k)^2. \tag{4}$$

To convert the values of k into actual frequencies we can use the following formula:

$$f = \frac{k \cdot f_s}{N}, \tag{5}$$

where f_s is the sampling frequency and N is the number of samples.

The spectral envelope [15] is the contour of the magnitude spectrum, which is estimated with CheapTrick [25]. The shape of this curve approximates the frequency response of the vocal tract.

The aperiodicity is defined as the power ratio between the speech signal and the aperiodic component of the signal. It is extracted by D4C algorithm [26].

The cepstrum is the inverse discrete Fourier transform (IDFT) of the logarithm of the audio signal's P_k power spectrum:

$$C_n = IDFT(\log(P_k)), \tag{6}$$

where $k = 0, 1, ..., N - 1$. It gives us a more compact, low dimensional, decorrelated representation.

With mel-cepstral analysis [10, 33] we can warp the frequency scale and compress the frequency coefficients. With the following formula we can calculate the

M-th order mel-cepstral coefficients:

$$\log\left(X(e^{-i\omega})\right) = \sum_{m=0}^{M} \tilde{c}_m e^{-i\tilde{\omega}}, \tag{7}$$

where $X\left(e^{-i\omega}\right)$ is the discrete Fourier transform of x_n, and $\tilde{c}_m$ is the m-th order mel-cepstral coefficients.

$$\tilde{\omega} = \tan^{-1}\frac{\left(1-\alpha^2\right)\sin\omega}{(1+\alpha^2)\cos\omega - 2\alpha}. \tag{8}$$

is the phase response of an all-pass filter. It gives us the warped frequency scale. The $\alpha \in [-1, 1]$ is the all-pass constant, which gives the warping characteristic. With the right α value, which is chosen to be 0.58 based on Merlin [34] suggestion, the mel-scale becomes a good approximation to the human auditory frequency scale.

The following features are used as inputs and targets to several CNN and ConvLSTM architectures: Mel-Cepstral Coefficients (MCEP) and band aperiodicity (BAP) were calculated using Eq. (7) from the spectral envelope and aperiodicity, respectively. Linear interpolation of $\log F0$ was calculated from $F0$. We also applied a thresholded binary voiced/unvoiced (V/UV) mask. Dynamic features (delta and delta-delta) were determined using MCEP and BAP.

3.3 Data sets

We employed the following benchmark corpora in our voice conversion evaluations.

TIMIT [35] is used frequently for comparing different machine learning methods. This database is attractive for verification and parameter tuning of the algorithms since it is relatively small, but still has phonetically diverse samples. The training set has 462 speakers, 8 utterances/speaker. The validation set consists of 50 speakers, totally 400 utterances, and the test set contains 192 sentences from 24 speakers. Each utterance is approximately 3.5 seconds long on average. The speakers also represent 8 major dialect regions of the United States.

NTIMIT [9] is a multi-speaker speech database with phone bandwidth that is derived from TIMIT by adding noise to the samples.

3.4 Pre-processing

The target TTS voices are generated with the Festival Speech Synthesis System [4] using the transcripts of the datasets. The choice of Festival was motivated by comparing several TTS systems and supported generic voices. The TTS generated sound files were aligned to match with the corresponding sound files produced by the speaker. We used Dynamic Time Warping (DTW) for the alignments.

In case of the TIMIT [35] and NTIMIT [9] data sets, audio normalization was unnecessary. The train-dev-test speakers are carefully separated. We also augmented data by applying speed warping factors to enlarge the TIMIT dataset.

Vocoder features were extracted from both the original speakers' and the TTS voice. The interpolated $\log F0$, the V/UV mask vector and delta and delta-delta features are calculated. Multiple combinations of these features are tested as inputs for different network architectures. Z-score normalization was applied to all of the calculated features, resulting in zero mean and unit variance.

3.5 Modeling feature transformation

For feature transformation, various deep learning architectures were applied and compared: (1) we experimented with an architecture, which we refer to as Dense, having four 1,024 unit dense layers. (2) we used a Convolutional Neural Network (ConvNet) with two 1D convolutional layers of 512 units and kernel width 7 and stride 1, and two 1,024 unit dense layers. (3) tried a model, which we call C-BLSTM, having three batch normalized 256 unit 1D convolutional layers with kernel width 3, one 128 unit BLSTM layer and two 512 unit dense layers was also tested, where the first dense layer was batch normalized. Finally, two state-of-the-art architectures based on (4) Residual Networks (ResNet) [11] and (5) Wav2Letter [20] were also evaluated.

Within all networks, we used ReLU activation functions and dropout layers with probability between 0.2 and 0.3, in addition to adding a final dense output layer on top with linear activation.

4 Results

Here, we present our results. We note that before training on larger data sets, a sanity check was carried out by varying the size of TIMIT using the Dense, the ConvNet and the C-BLSTM architectures, confirming that augmentation improves the results.

4.1 Experimental setup

Festival 2.5 with "voice_cmu_us_rms_cg" was used to generate the target TTS utterances in the experiments. We applied PyWorld [13] and SPTK 3.9 [14] for feature extraction. We implemented the neural networks in Python using the Keras [5] deep learning framework backed by Tensorflow [2]. We used early stopping with patience 10 and Adam optimizer with its default parameters. The loss function is Mean Squared Error (MSE), however, if interpolated $\log F0$ with V/UV mask is used as input, it is not used in loss calculation. We trained the models on TIMIT [35] dataset.

Regardless of the model, the network took the whole sequence as input, and its target was the DTW transformed TTS utterances.

Our implementation is available on GitHub[1].

[1]`https://github.com/lkopi/deidentification`

4.2 Objective evaluation

To show that the trained network producing quality outputs at utterance level, we conducted two quantitative experiments.

The first experiment concerned ASR accuracy. We uploaded the de-identified network outputs to Google Cloud Speech-to-Text system and quantified the difference between the true and the predicted transcript.

Precision values are given in Table 1. First, we measured the TTS voices (without applying DTW). It reached 97% ASR accuracy. The DTW aligned TTS reached 93%. ASR accuracy, i.e., a 4% drop was found. The tested DNNs performed well on the TIMIT data set, producing well understandable speech after synthesization. Intriguingly, they mostly achieved 80-85% ASR accuracy, only, a relatively large drop compared to the 93% goal.

Cloud-based ASR services improve constantly. One can expect that the performance will increase over time. We found that the ResNet and Dense architectures marginally outperformed the others.

In the second experiment, we evaluated Mel-Cepstral Distortion (MCD) [19] between de-identified Dense network outputs and the aligned TTS signals. MCD is given by the following equation:

$$MCD[dB] = \frac{1}{N}\sum_{n=1}^{N}\frac{10}{\log 10}\sqrt{2\sum_{d=1}^{D}(c_{n,d}-\hat{c}_{n,d})^2}, \tag{9}$$

where N is the number of frames in the analysis, D is the number of coefficients, $c_{n,d}$ and $\hat{c}_{n,d}$ are the dth coefficient of the nth frame of the target and the predicted MCEP vector, respectively.

The final MCD values were obtained by averaging over all 1,680 test sample pairs of the TIMIT corpora using the Dense architecture. The mean and standard deviation values are presented in Fig. 2 for our method (last column) together with values available in the literature. Our method seemingly outperformed its baselines, however other methods listed in the figure were trained on different datasets, so

Table 1: Details of the objective evaluations on the TIMIT database. The average and the standard deviation of the Letter Accuracy Rate (LAR) measured with Google Cloud Speech-to-Text system is presented for the proposed architectures and input features. Notation: "il$F0$" means interpolated $\log F0$.

Method	Architecture	ASR (LAR) precision using the following features:		
		$\log F0$ + MCEP + BAP	il$F0$ + MCEP + BAP	il$F0$ + MCEP + BAP + deltas
Dense	5 dense	0.77 ±0.17	**0.85 ±0.15**	**0.84 ±0.16**
ConvNet	2 conv + 3 dense	0.76 ±0.17	0.82 ±0.17	0.82 ±0.15
C-BLSTM	3 conv + blstm + 3 dense	0.77 ±0.14	0.79 ±0.15	0.79 ±0.15
ResNet	4 residual + 3 dense	**0.79 ±0.15**	0.82 ±0.16	0.82 ±0.14
Wav2Letter	9 conv + 2 dense	0.76 ±0.16	0.79 ±0.17	0.77 ±0.16

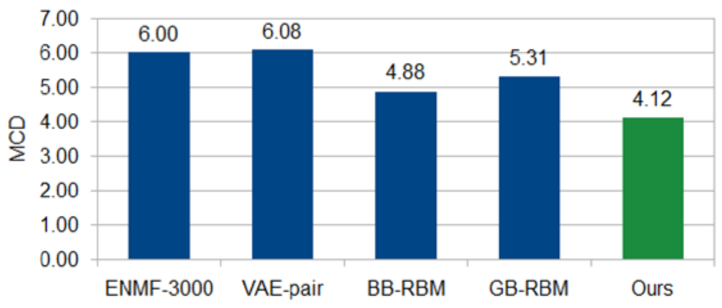


Figure 2: Mean Mel-Cepstral Distortion (MCD) values of various schemes: Exemplar-based Nonnegative Matrix Factorizations (ENMF) [12] using 3000 randomly selected source-target pair frames, VAE-pair [12], Bernoulli-Bernoulli RBM (BB-RBM) [29], Gaussian-Bernoulli RBM (GB-RBM) [29] and our proposed method.

a direct comparison is not possible. ENMF-3000 and VAE-pair were evaluated on the VCC2016 Speech Corpus, and the BB-RBM and GB-RBM were ran on ATR Japanese speech database. The figure only allows to give an impression about the performance of our method.

4.3 Subjective evaluation

To confirm that our Dense network can properly de-identify human speech, we conducted four qualitative experiments with human participants in an isolated environment. The synthesized outputs are intelligible and successfully de-identified. The collection of the subjective tests is available online[2].

In all four tests, the results were convincing, subjects performed like *random choice*, see Table 2. Participants were unable to sense any of the relevant aspects of the speakers. In both 6AFC tests, the task was to guess the number of speakers (between 1 and 6). In the first 6AFC test, none of the subjects inferred accurately. In the second 6AFC test, 4 subjects out of 22 predicted correctly, which matches random guessing within tolerance.

5 Summary

We presented a deep neural network based speech de-identification method that can map vocoder features of human speech to those of a generic TTS engine with little or minimal loss of sound quality using the TIMIT data set. The novelty of our scheme is that de-identification is based on speech-text sample pairs, which are widely available in the speech processing community. In the resulting signal,

[2]`https://people.inf.elte.hu/foauaai/deidentification`

Table 2: Results of the perceptual listening experiments. We report the average and the standard deviation of the identification accuracy.

Task	# of subj.	# of samp.	Accuracy mean ±std	Random choice
A-not-B	22	20	0.56 ±0.15	0.5
Female/Male (2AFC)		15	0.51 ±0.15	0.5
# of Speakers (6AFC)		6	0	0.16
		6	0.18	0.16

the identity of the speaker is concealed, as confirmed by our perceptual listening experiments.

A limitation of our technique is that the dynamics of the original speaker are inherited due to the application of DTW. We hypothesize that this problem may be alleviated by applying DTW in the loss function of the deep network. We leave such studies to future work.

Our technique enables privacy-aware speech recognition for adults. The proposed method is lightweight and can be used for collecting de-identified databases when the privacy of the user is important, for example in cloud-based speech services or in medical records. The fact that our method requires only speech-transcript sample pairs is a very promising aspect for deep learning, which requires large and high quality databases.

References

[1] Aaron van den, Oord, Dieleman, Sander, Zen, Heiga, Simonyan, Karen, Vinyals, Oriol, Graves, Alex, Kalchbrenner, Nal, Senior, Andrew, and Kavukcuoglu, Koray. Wavenet: A generative model for raw audio. *arXiv:1609.03499*, 2016.

[2] Abadi, Martín, Agarwal, Ashish, et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from `tensorflow.org`.

[3] Amodei, Dario, Ananthanarayanan, Sundaram, et al. Deep Speech 2: End-to-end speech recognition in English and Mandarin. In *Proceedings of the 33rd International Conference on Machine Learning*, pages 173–182, 2016.

[4] Black, Alan. The Festival Speech Synthesis System: System Documentation (1.1.1). Technical Report HCRC/TR-83, Human Communication Research Center, 1997.

[5] Chollet, Francois et al. Keras. `https://keras.io`, 2015.

[6] Degottex, Gilles, Lanchantin, Pierre, and Gales, Mark. A log domain pulse model for parametric speech synthesis. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 26(1):57–70, 2018. DOI: `10.1109/taslp.2017.2761546`.

[7] Erro, Daniel, Sainz, Inaki, Navas, Eva, and Hernaez, Inma. Harmonics plus noise model based vocoder for statistical parametric speech synthesis. *IEEE Journal of Selected Topics in Signal Processing*, 8(2):184–194, 2014. DOI: `10.1109/jstsp.2013.2283471`.

[8] Espic, Felipe, Botinhao, Cassia Valentini, and King, Simon. Direct modelling of magnitude and phase spectra for statistical parametric speech synthesis. *Proc. Interspeech*, 2017. DOI: `10.21437/interspeech.2017-1647`.

[9] Fisher, William M., Doddington, George R., Goudie-Marshall, Kathleen M., Jankowski, Charles, Kalyanswamy, Ashok, Basson, Sara, and Spitz, Judith. NTIMIT: A phonetically balanced, continuous speech, telephone bandwidth speech database. In *Proc. IEEE ICASSP*, pages 109–112, 1990. DOI: `10.1109/icassp.1990.115550`.

[10] Fukada, Toshiaki, Tokuda, Keiichi, Kobayashi, Takao, and Imai, Satoshi. An adaptive algorithm for mel-cepstral analysis of speech. In *Acoustics, Speech, and Signal Processing, 1992. ICASSP-92., 1992 IEEE International Conference on*, volume 1, pages 137–140. IEEE, 1992. DOI: `10.1109/icassp.1992.225953`.

[11] He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. Deep residual learning for image recognition. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[12] Hsu, Chin-Cheng, Hwang, Hsin-Te, Wu, Yi-Chiao, Tsao, Yu, and Wang, Hsin-Min. Voice conversion from non-parallel corpora using variational auto-encoder. In *APSIPA, Asia-Pacific*, pages 1–6. IEEE, 2016. DOI: `10.1109/apsipa.2016.7820786`.

[13] Hsu, Jeremy et al. PyWorldVocoder: A Python wrapper for World Vocoder. `https://github.com/JeremyCCHsu/` Python-Wrapper-for-World-Vocoder, 2016.

[14] Imai, Satoshi, Kobayashi, Takao, et al. Speech signal processing toolkit (SPTK), 2009. `http://sp-tk.sourceforge.net`.

[15] Iser, Bernd, Minker, Wolfgang, and Schmidt, Gerhard. Broadband spectral envelope estimation. *Bandwidth Extension of Speech Signals. Lecture Notes in Electrical Engineering*, 13:67–95, 2008. DOI: `10.1007/978-0-387-68899-2_5`.

[16] Jin, Qin, Toth, Arthur R., Schultz, Tanja, and Black, Alan W. Speaker de-identification via voice transformation. *IEEE Workshop on Automatic Speech Recognition & Understanding*, pages 529–533, 2009. DOI: 10.1109/ASRU.2009.5373356.

[17] Justin, Tadej, Struc, Vitomir, Dobrisek, Simon, Vesnicer, Bostjan, Ipsic, Ivo, and Mihelic, France. Speaker de-identification using diphone recognition and speech synthesis. *11th IEEE International Conference and Workshops on Automatic Face and Gesture Recognition (FG)*, pages 1–7, 2015. DOI: 10.1109/FG.2015.7285021.

[18] Kawahara, Hideki. STRAIGHT, exploitation of the other aspect of VOCODER: Perceptually isomorphic decomposition of speech sounds. *Acoustical Science and Technology*, 27(6):349–353, 2006. DOI: 10.1250/ast.27.349.

[19] Kominek, John, Schultz, Tanja, and Black, Alan W. Synthesizer voice quality of new languages calibrated with mean mel cepstral distortion. *First International Workshop on Spoken Languages Technologies for Under-Resourced Languages (SLTU-2008)*, pages 63–68, 2008.

[20] Liptchinsky, Vitaliy, Synnaeve, Gabriel, and Collobert, Ronan. Letter-based speech recognition with Gated ConvNets. *CoRR*, abs/1712.09444, 2017.

[21] Liu, Li-Juan, Chen, Ling-Hui, Ling, Zhen-Hua, and Dai, Li-Rong. Spectral conversion using deep neural networks trained with multi-source speakers. *Proc. IEEE ICASSP*, pages 4849–4853, 2015. DOI: 10.1109/ICASSP.2015.7178892.

[22] Magariños, Carmen, Lopez-Otero, Lopez-Otero, Paula, Docio-Fernandez, Laura, Rodriguez-Banga, Eduardo, Erro, Daniel, and Garcia-Mateo, Carmen. Reversible speaker de-identification using pre-trained transformation functions. *Computer Speech & Language*, 46:36–52, 2017. DOI: 10.1016/j.csl.2017.05.001.

[23] Mohammadi, Seyed Hamidreza and Kain, Alexander. Voice conversion using deep neural networks with speaker-independent pre-training. *Proc. IEEE SLT Workshop*, pages 19–23, 2014. DOI: 10.1109/SLT.2014.7078543.

[24] Mohammadi, Seyed Hamidreza and Kain, Alexander. An overview of voice conversion systems. *Speech Communication*, 88:65–82, 2017. DOI: 10.1016/j.specom.2017.01.008.

[25] Morise, Masanori. CheapTrick, a spectral envelope estimator for high-quality speech synthesis. *Speech Communication*, 67:1–7, 2015. DOI: 10.1016/j.specom.2014.09.003.

[26] Morise, Masanori. D4C, a band-aperiodicity estimator for high-quality speech synthesis. *Speech Communication*, 84:57–65, 2016. DOI: `10.1016/j.specom.2016.09.001`.

[27] Morise, Masanori, Kawahara, Hideki, and Nishiura, Takanobu. Rapid F0 estimation for high-SNR speech based on fundamental component extraction. *IEICE TRANSACTIONS on Information and Systems*, 93:109–117, 2010.

[28] Morise, Masanori, Yokomori, Fumiya, and Ozawa, Kenji. WORLD: A vocoder-based high-quality speech synthesis system for real-time applications. *IEICE Trans. Info. Sys.*, 99(7):1877–1884, 2016. DOI: `10.1587/transinf.2015edp7457`.

[29] Nakashika, Toru, Takiguchi, Tetsuya, and Ariki, Yasuo. Voice conversion based on speaker-dependent restricted Boltzmann machines. *IEICE Transactions on Information and Systems*, E97.D(6):1403–1410, 2014. DOI: `10.1587/transinf.E97.D.1403`.

[30] Nguyen, Hy Quy, Lee, Siu Wa, Tian, Xiaohai, Dong, Minghui, and Chng, Eng Siong. High quality voice conversion using prosodic and high-resolution spectral features. *Multimedia Tools and Applications*, 75(9):5265–5285, 2016. DOI: `10.1007/s11042-015-3039-x`.

[31] Qian, Jianwei, Du, Haohua, Hou, Jiahui, Chen, Linlin, Jung, Taeho, Li, Xiang-Yang, Wang, Yu, and Deng, Yanbo. VoiceMask: Anonymize and sanitize voice input on mobile devices. *arXiv:1711.11460*, 2017.

[32] Sekii, Yusuke, Orihara, Ryohei, Kojima, Keisuke, Sei, Yuichi, Tahara, Yasuyuki, and Ohsuga, Akihiko. Fast many-to-one voice conversion using autoencoders. *Proceedings of the 9th International Conference on Agents and Artificial Intelligence*, pages 164–174, 2017. DOI: `10.5220/0006193301640174`.

[33] Tokuda, Keiichi, Kobayashi, Takao, Masuko, Takashi, and Imai, Satoshi. Mel-generalized cepstral analysis — a unified approach to speech spectral estimation. In *Third International Conference on Spoken Language Processing*, 1994.

[34] Wu, Zhizheng, Watts, Oliver, and King, Simon. Merlin: An open source neural network speech synthesis system. In *9th ISCA Speech Synthesis Workshop*. ISCA, 2016. DOI: `10.21437/ssw.2016-33`.

[35] Zue, Victor, Seneff, Stephanie, and Glass, James. Speech database development at MIT: TIMIT and beyond. *Speech Communication*, 9:351–356, 1990. DOI: `10.1016/0167-6393(90)90010-7`.

Acta Cybernetica **25** (2021) 271–284.

Toolset for Supporting the Research of Lattice Based Number Expansions

Péter Hudoba[ab] and Attila Kovács[ac]

Abstract

The world of generalized number systems contains many challenging areas. Computer experiments often support the theoretical research. In this paper we introduce a toolset that helps to analyze some properties of lattice based number expansions. The toolset is able to (1) analyze the expansions, (2) decide the number system property, (3) classify and visualize the periodic points.

The toolset is implemented in Python, published alongside with a database that stores plenty of special expansions, and is able to store the custom properties like signature, operator eigenvalues, etc. Researchers can connect to the server and request/upload data, or perform experiments on them.

We present an introductory usage of the toolset and detail some results that has been observed by the toolset. The toolset can be downloaded from `http://numsys.info` domain.

1 Introduction

The generalization of positional number representations to a wide range of digit sets or to higher dimensions is a fascinating story. Grünwald (1885) investigated negative-based, Kempner (1936), Knuth (1960), Khmelnik (1964), Penney (1965) complex-based systems. From the 70's Kátai, B. Kovács, Környei, Pethő (the "Hungarian school") and Gilbert examined systematically the radix extensions in *algebraic number fields*. In the 90's the *topological aspects* of radix representations were studied by Bandt, Indlekofer, Járai, Kátai, Lagarias, Wang, Vince, and later by Akiyama, Thuswaldner and others. The canonical number representation was generalized to *arbitrary polynomial systems* by Pethő (1989), and investigated later extensively by many authors (incl. Akiyama, Brunotte, Kovács, Pethő, Rao, Scheicher, Thuswaldner). The number system concept in *general lattices* was investigated first by Vince (1993). The *algorithmic aspects* of canonical (polynomial) systems was initiated by Brunotte (2001) and for general lattices by the second

[a]Eötvös Loránd University, Budapest, Hungary
[b]E-mail: `peter.hudoba@inf.elte.hu`, ORCID: `0000-0001-5810-4193`
[c]E-mail: `attila@inf.elte.hu`, ORCID: `0000-0002-1858-7618`

DOI: `10.14232/actacyb.289524`

author (2000). Recently, a special type of radix systems (SRS) studied in length by Thuswaldner and his co-workers (the "Austrian school").

2 Preliminaries

Let Λ be a lattice in $\mathbb{R}^n$ and let $M : \Lambda \to \Lambda$ be a linear operator such that $\det(M) \neq 0$. Let furthermore $0 \in D \subseteq \Lambda$ be a finite subset. Lattices can be seen as finitely generated free Abelian groups and have many significant applications in pure mathematics (Lie algebras, number theory and group theory), in applied mathematics (coding theory, cryptography) because of conjectured computational hardness of several lattice problems, and are used in various ways in the physical sciences. We note that the number system research in general lattices comprises also the orders.

Definition 1. *The triple* (Λ, M, D) *is called a* number system *(GNS) if every element* x *of* Λ *has a unique, finite representation of the form*

$$x = \sum_{i=0}^{L} M^i d_i \ ,$$

where $d_i \in D$ *and* $L \in \mathbb{Z}$ *(*$L+1$ *is the length of the expansion).*

Here M is called the *base* and D is the *digit set.* It is easy to see that similarity preserves the number system property, i.e., if M_1 and M_2 are similar via the matrix Q then (Λ, M_1, D) is a number system if and only if $(Q\Lambda, M_2, QD)$ is a number system at the same time. If we change the basis in Λ a similar integer matrix can be obtained, hence, there is no loss of generality in assuming that M is integral acting on the lattice $\mathbb{Z}^n$. If two elements of Λ are in the same coset of the factor group $\Lambda/M\Lambda$ then they are said to be *congruent* modulo M. The following theorem gives a necessary condition for the number system property.

Theorem 1. *If* (Λ, M, D) *is a number system, then (1)* D *must be a full residue system modulo* M*, (2)* M *must be expansive, and (3)* $\det(I_n - M) \neq \pm 1$*. (unit condition). If a system fulfils the first two conditions then it is called a radix system.*

We note that the theorem in this form was stated first in [9] but it was well-known and used much earlier by Kátai and Vince. The full residue system property can be decided easily using Smith normal form [8]. Algorithms, that calculate the eigenvalues of M exactly in a finite number of steps exist only for a few special classes of matrices. For general matrices, iterative algorithms produce approximate solutions. In polynomial systems, where M is the companion of a monic integer polynomial f, deciding the Schur or Hurwitz stability of f is computationally equivalent with the expansivity check. Verification of the unit condition is trivial.

Write $\varphi : \Lambda \to \Lambda$, $x \overset{\varphi}{\mapsto} M^{-1}(x-d)$ for the unique $d \in D$ satisfying $x \equiv d$ (mod M). Since M^{-1} is contractive and D is finite, there exists a norm $\|.\|$ on Λ and a constant C such that the orbit of every $x \in \Lambda$ eventually enters the finite set $S = \{x \in \Lambda \mid \|x\| < C\}$ after repeated application of φ. This means that the sequence $x, \varphi(x), \varphi^2(x), \ldots$ is eventually periodic for all $x \in \Lambda$. Clearly, (Λ, M, D) is a number system iff for every $x \in \Lambda$ the orbit of x eventually reaches 0. A point p is called *periodic* if $\varphi^k(p) = p$ for some $k > 0$. The orbit of a periodic point p is a *cycle*. The set of all periodic points is denoted by $\mathcal{P}$. The *signature* $[l_1, l_2, \ldots, l_\omega]$ of a radix system is a finite sequence of non-negative integers in which the periodic structure $\mathcal{P}$ consists of $\#l_i$ cycles with period length i $(1 \le i \le \omega)$. Clearly, the signature of a number system is $Sig = [1]$.

The following problem classes are in the mainstream of the research.

- For a given (Λ, M, D) the *decision problem* asks if the triple forms a number system or not.
- For a given (Λ, M, D) the *classification problem* means finding all cycles (*witnesses*).
- The *parametrization problem* means finding parametrized families of number systems.
- The *construction problem* aims at constructing a digit set D to M for which (Λ, M, D) is a number system. In general, construct a digit set D to M such that (Λ, M, D) satisfies a given signature.

We note that the algorithmic complexity of the decision and classification problems are still unknown.

The *fundamental domain* or set of "fractions" in (Λ, M, D) can be defined as

$$H = \left\{ \sum_{i=1}^{\infty} M^{-i} d_i : d_i \in D \right\} \subseteq \mathbb{R}^n .$$

Theorem 2. *(a) H is compact. (b) $x \in \mathcal{P} \Leftrightarrow x \in -H$.*

The compactness was proved by many authors (see e.g. Vince [15]). The $\Rightarrow$ part of (b) was proved in [9]. The other direction is obvious as well, otherwise 0 would have at least two different representations.

The theorem means that in order to determine the periodic points it is enough to localize the lattice points in $-H$. There are two different approaches to overcome this problem: the IFS-method (see [8, 10]), and the covering method (see [8, 4]), which was optimized by the authors [6]. The idea of the latter is that we can put the compact set $-H$ into a box B in which the integer elements are easily enumerable. Then, we can compute the pairs $(x, \varphi(x))$ for all $x \in B$, and finally, we determine the cycles applying one of the cycle finding methods.

There are other algorithms for solving the decision/classification problems. Based on the method of Vince [15], Brunotte [2] described a digit-propagation algorithm for polynomial systems with canonical digits. Later, his method was generalized for arbitrary operators and digit sets [4]. The shortcoming of this method is the sequential nature of the digit propagation, however, there is an algorithmic attempt to overcome this disadvantage [14].

Let $f(x) = a_0 + a_1x + a_2x^2 + \cdots + a_{n-1}x^{n-1} + x^n$ be an integer (monic) polynomial. Let us denote the factor ring $\mathbb{Z}[x]/(f)$ by Λ_f. Then Λ_f is a lattice and all the problems regarding number expansions in Λ_f can be formulated in $\mathbb{Z}^n$, where M is the companion of f. If f is irreducible then Λ_f is isomorphic to $\mathbb{Z}[\theta]$ where $f(\theta) = 0$ in an appropriate extension of $\mathbb{Q}$. Hence, if the digit set D is restricted to be a set of non-negative numbers $D = \{0, 1, \ldots \mid a_0 \mid -1\}$, we get a straightforward generalization of the traditional number systems in $\mathbb{Z}$. In this case the digit set is called *canonical.* If the radix system (Λ_f, θ, D) satisfies the unique representation property of Definition 1 with some canonical digit set D then it is called a *canonical number system* (CNS). The notion of canonical digit sets can be extended to form a *j-canonical* set $D_j = \{0, e_j, \ldots, (\mid a_0 \mid -1)e_j\} \subset \mathbb{Z}^n$ (e_j is the jth unit vector) [8]. There exists a canonical number system in $\mathcal{O}_K$ – the ring of integers of the algebraic number field K – if and only if there is a power integral basis in $\mathcal{O}_K$ [12]. We note that canonical digit sets may or may not exist in different lattices and canonicity depends on the chosen basis. The *symmetric digit set* is formed by those integer multiples of e_j which are closest to the origin, and was introduced by Kátai [7]. The *adjoint digit set* consists of those lattice points which are in $\det(M)\big[-\frac{1}{2}, \frac{1}{2}\big)$. The *dense digit set* — in which each digit has the smallest norm in its residue class — were introduced and used extensively by the second author. We note that the adjoint set is a dense one in a special basis.

3 The toolset

In order to be able to support the theoretical research we built a Python based toolset that aid at the investigations and experiments. The toolset implements some basic functionalities for number expansion research. It offers multiple ways to solve the decision or classification problems from a simple brute force to probabilistic solutions. In this section we give a short outline of the functionality of the toolset.

3.1 Construction

The toolset contains multiple classes with different purposes. The main class, named RadixSystem, implements a Radix System that we can create with a base and a digit set. The base matrix can be set directly, but there is a function to convert a polynomial to a matrix (creating its companion) as well. The digit set can be passed directly with a list, exactly determining the digits, or with a generator, that generates specific types of digit sets (*RadixSystemSymmetricDigits*, *RadixSystemCanonicalDigits*, *RadixSystemShiftedCanonicalDigits*, *RadixSystemAdjointDigits*, *RadixSystemDenseDigits*).

An example for creating the following radix systems can be seen in the Listing 1:

$$\left(\mathbb{Z}^2, \begin{bmatrix} 0 & -7 \\ 1 & -7 \end{bmatrix}, \left\{ \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \ldots, \begin{bmatrix} 6 \\ 0 \end{bmatrix} \right\}\right) \text{ and } \left(\mathbb{Z}^2, \begin{bmatrix} 0 & -7 \\ 1 & -7 \end{bmatrix}, \left\{ \begin{bmatrix} -3 \\ 0 \end{bmatrix}, \ldots, \begin{bmatrix} 3 \\ 0 \end{bmatrix} \right\}\right)$$

Listing 1: Example of construction with the toolset

```
rs = RadixSystem([[0,-7],[1,-7]],
    [[0, 0], [1, 0], [2, 0], [3, 0], [4, 0], [5, 0], [6, 0]])

#Using digit set generator
rs = RadixSystem([[0,-7],[1,-7]], RadixSystemCanonicalDigits())

#Creates the same system with symmetric digit set
rs = RadixSystem([[0,-7],[1,-7]], RadixSystemSymmetricDigits())
```

3.1.1 Necessary conditions

A radix system object can be created by the toolset, but if the necessary conditions for the computations do not hold then the toolset will throw the appropriate exception. The necessary conditions (see Section 2) mean that the base operator has to be regular ($\det(M) \neq 0$) (otherwise the system throws a *RadixSystemRegularityException* exception), the digit set must be a full residue system modulo the base (otherwise the system throws the *RadixSystemFullResidueSystemException*) and the base operator has to be expansive.

The RadixSytem class does not check the unit condition (third necessary condition of Theorem 1) — because it is not a condition for a Radix System, only for a Number Nystem — the class has a function, named *checkUnitCondition*, that returns true if the radix system fulfils that condition.

3.2 Expansions

If the user wants to find the expansion of a lattice point in a specific system she can apply the φ function which is implemented in *phiFunction*. Applying the Smith Normal Form, the system performs the computation efficiently. Applying the *phiFunction* iteratively the system computes the orbit of a point containing the periodic parts (*getOrbitFrom*). Recall that if the orbit of a lattice point ends at zero then it has a finite expansion (*hasFiniteExpansion*).

Listing 2: Example of using φ based functions

```
rs = RadixSystem([[0,-7],[1,-7]], RadixSystemSymmetricDigits())
print(rs.phiFunction([2,3]))
print(rs.getOrbitFrom([6,3]))

# Result:
# [3, 0]
# [[6, 3], [-4, -1], [6, 1], [-6, -1], [6, 1]]
```

3.3 Covers

Based on Theorem 2 ($x \in \mathcal{P} \Leftrightarrow x \in -H$) the system determines a Box that contains all the periodic points (*getCoverBox*). The volume of this Box can also be calculated (*getCoverBoxVolume*) and a Python generator can be obtained to iterate through all of the points within the Box (*getPointsInBox*). It is a simple brute force algorithm for GNS decision (detailed further in Subsection 3.5).

There is a heavyweight algorithm *getCycles* that calculates all of the orbits from all of the points of the Box and returns all of the cycles. Clearly, if the *getCycles* returns only the zero point then the radix system is a number system.

3.4 Drawing tools

The toolset has a class for drawing different aspects of number expansions. The user can analyze the expansions by the expansion graph. By default, it shows the trajectory from all of the points inside the Box. In Figure 1 we can see a radix system that does not fulfil the number system property (since it has a non-trivial cycle $[-1, 1]$). In Figure 2 we can see a plot of some fraction sets.

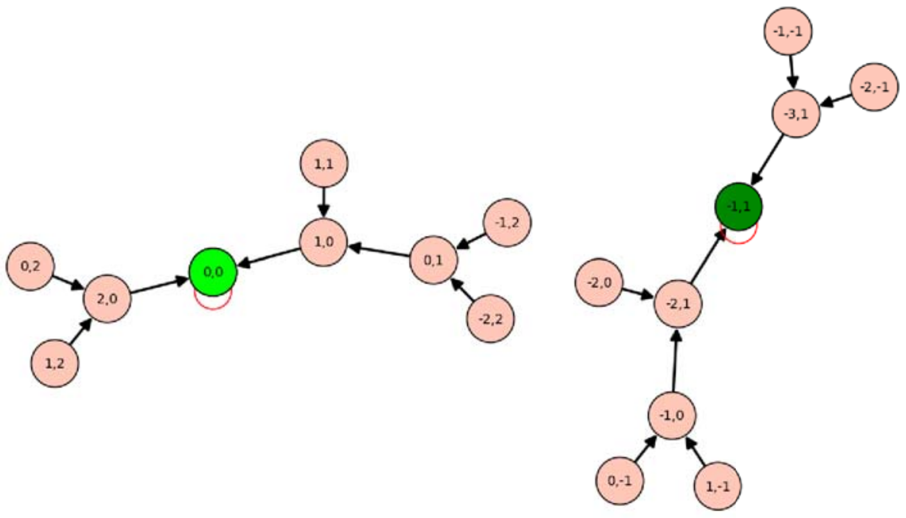


Figure 1: An expansion graph.

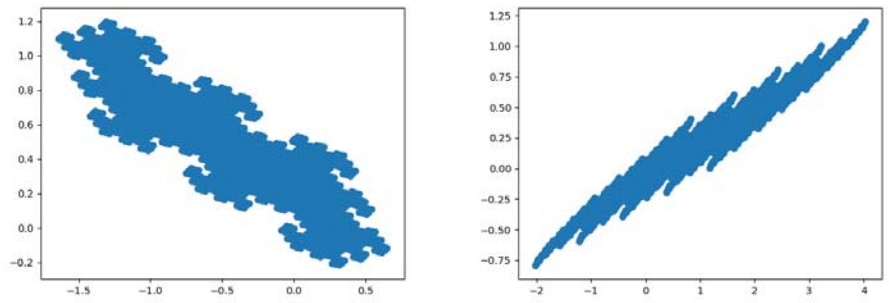


Figure 2: A fraction set of $[[0,-3],[1,2]]$ and $[[0,-5],[1,-4]]$ with canonical digit sets.

3.5 Decision techniques

In this subsection we discuss some decision methods that can be used by the toolset.

3.5.1 Naive decision

The naive decision method checks the orbits from all points of the cover Box and if there is only one periodic point (should be the zero) then returns true.

Listing 3: Example of how to call a naive decide method of the toolset

```
rs = RadixSystem([[0,-7],[1,-7]], RadixSystemCanonicalDigits())
print(rs.decideGNS())
```

3.5.2 Volume optimization

The naive method iterates through all of the points within the bounding Box. However, since integer similarity transformations preserve the number system property, we can try to change the basis where the bounding box is smaller. In [4] the authors suggested a simulated annealing genetic algorithm that finds a similarity transformation minimizing the size of the cover box. In Figure 3, we can see an example how the algorithm decreases the volume of the possible space of periodic points. For higher dimensions, the speedup is much higher.

Listing 4: Example of how to call start a cover box volume optimization

```
rs = RadixSystem([[0,-7],[1,-7]], RadixSystemCanonicalDigits())
volumeOptimized = rs.optimize()
print(volumeOptimized.decideGNS())
```

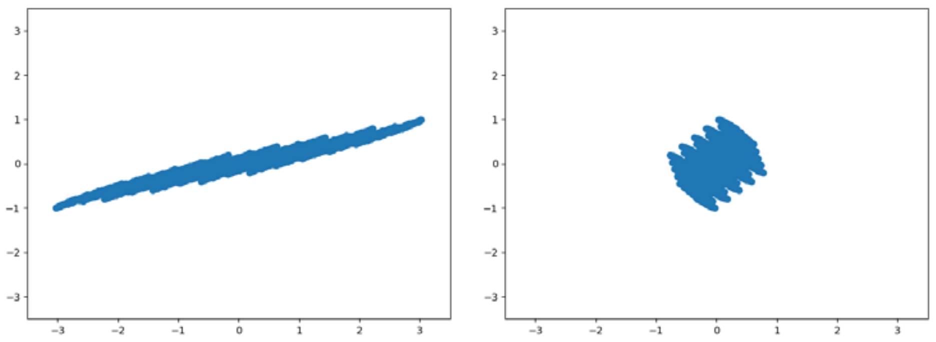

(a) Before optimization (b) After optimization

Figure 3: A fraction set of the operator [[0, -5], [1, -4]] applying the symmetric digit set.

3.5.3 Two-step optimization

In [6] a method was suggested as an extension of the volume optimization. Besides optimizing the volume the authors showed a method of minimizing the number of multiplications in the function φ as well. The amount of multiplications is affected by the Smith Normal Form computation and the inverse computation of the base. If we have the two transformations we can iterate through all of the points of the volume optimized space, transform into the φ optimized space and find there the orbit.

Listing 5: Example of how to use the two-step optimization

```
rs = RadixSystem([[0,-7],[1,-7]], RadixSystemCanonicalDigits())

optimizedVol, optimizeVolT = rs.optimize(returnTransformationAlso=True)

optimizedPhi, optimizePhiT = rs.optimize(
    targetFunction = lambda actVal, T:
          phiOptimizeTargetFunction(actVal, T,optimizeVolT.inverse()),
    returnTransformationAlso=True)

transformMatrix = optimizePhiT * optimizeVolT.inverse()
print(optimizedPhi.decideGNS(startPointSource=optimizedVol,
                             pointTransform=transformMatrix))
```

3.5.4 Length-n cycle

Based on our experiments, we observed that there are significantly larger number of cases when the cycles are short. Therefore we try to find cycles directly based on some digit combinations.

Considering the periodic points with length one, there is a $d \in D$ for $x \in \mathcal{P}$ where $M^{-1}(x-d) = x$ holds. We can reformulate this statement as $x - d = Mx \Rightarrow x - Mx = d \Rightarrow x = (I - M)^{-1}d$. In the algorithm we just simply iterate through all of the digits and check whether the result is a lattice point. If so, we have found a loop. If the digit set is small, this algorithm is really fast.

We can find length two cycles with the same technique. If there is an $x \in \mathcal{P}$ length-two periodic point, then there are $d_1, d_2 \in D$, where $M^{-1}(M^{-1}(x - d_1) - d_2) = x \Rightarrow M^{-1}(x - d_1) = Mx + d_2 \Rightarrow x = M^2x + Md_2 + d_1 \Rightarrow x = (I - M^2)^{-1}(Md_2 + d_1)$. If is there are d_1 and d_2 digits where x is a lattice point then we have found a cycle.

In general, for an length-n periodic point $\varphi^n(x) = x$, hence there are $d_1, ..., d_n \in D$ digits, where $x = (I - M^n)^{-1}(\sum_{i=1}^{n} M^{i-1}d_i)$ is a lattice point.

The weakness of this algorithm is its exponential complexity, i.e., if the digit set is big or there is not any short cycle in the system, then the algorithm finds no cycles (even if it exists).

Listing 6: Example of how to find the directly with fixed lengths cycles

```
rs = RadixSystem([[0,-7],[1,-7]], RadixSystemCanonicalDigits())
print(rs.findNLengthCycle(1))
print(rs.findNLengthCycle(2))

# Result:
# [[0, 0], [0, 0]]
# [[[12, 2], [-12, -2], [12, 2]], [[6, 1], [-6, -1], [6, 1]], [[-18, -3],
    [18, 3], [-18, -3]]]
```

3.5.5 Randomized method

For a given cycle we call all of the lattice points that lead to that cycle by iteration of φ as the **basin of the cycle**. Our experiments showed that at general radix systems most of the orbits lead to a non-zero periodic point. Hence, we can choose random lattice points uniformly and check the orbit to find witnesses (disproving the GNS property).

Listing 7: Example of how to run random test to find non-zero cycle

```
rs = RadixSystem([[0,-7],[1,-7]], RadixSystemSymmetricDigits())
print(rs.probGNSTest(numberOfTrials=100))

# Result:
# [-6, -1]
```

3.5.6 Smart decide

The smart decide algorithm estimates the runtime of the different methods and suggests the best algorithm to solve the decision problem. As we can see in Figure 4 the naive decision time increases faster than the smart decision function.

The algorithm has several steps:

1. If the cover Box is "small", simply brute force the space with the naive method; END.

2. Otherwise search length-n periodic points for small n directly; If it finds a non-trivial witness, return, otherwise continue with the next step.

3. Calculate the orbit of the "close-to-zero" lattice points (maximum the absolute value of $\det(M)$); if it finds a non-trivial witness, return, otherwise continue with the next step (the closeness is in the sense of the infinity norm)

4. Calculate orbits from uniformly chosen random lattice points; if it finds a non-trivial witness, return, otherwise continue with the next step.

5. Volume and φ optimization.

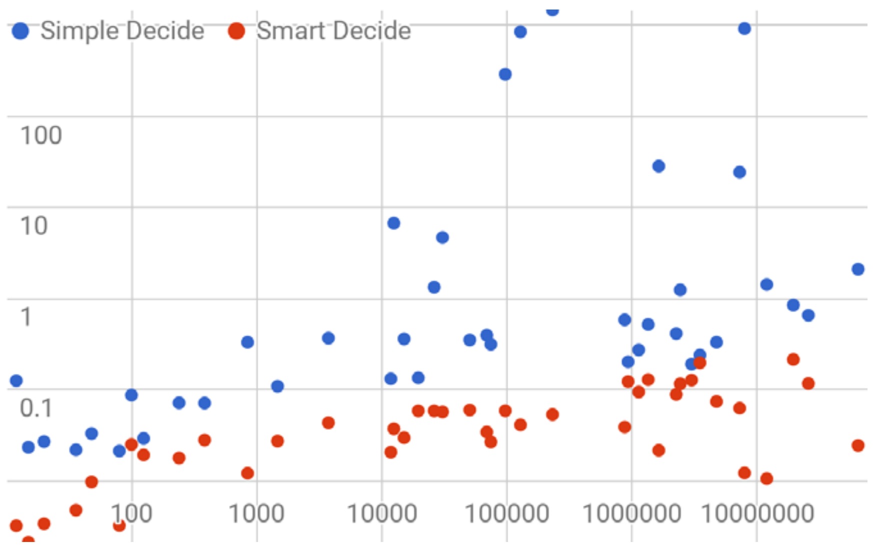


Figure 4: Runtime comparison of the simple decide method and the smart one by the size of the cover Box

3.6 Validation

In order to validate the correct functionality of the toolset we initiated multiple levels of testing.

Let $p(x) = c_0 + c_1x + c_2x^2 + \ldots + c_{n-1}x^{n-1} + x^n$ be the characteristic polynomial of the operator M. The following theorems were applied for validation:

- If the strictly dominant condition
$$\sum_{i=1}^{k} |c_i| < |c_0|$$
holds then then M is expansive. This is an immediate consequence of Rouché's theorem.

- If there is a norm for which $\|M^{-1}\| < 1/2$ then M can serve as a basis for a number system with dense digits [5].

- If $1 \le c_{n-1} \le \cdots \le c_1 \le c_0$ then the companion of $p(x)$ serve a basis for a number system with canonical digits [11].

- If the strictly strong dominant condition
$$2\sum_{i=1}^{k} |c_i| < |c_0|$$
holds then the companion of $p(x)$ can serve as a basis for a number system with symmetric digits [5].

We tested the toolset with known special cases of number systems, e.g. in [1] the 2nd theorem states that if

$$c_2, .., c_{n-1}, \sum_{i=1}^{n} c_i \ge 0$$

and the strictly strong dominant condition holds then the companion of $p(x)$ is a number system with canonical digits. We used listing of GNS examples in multiple articles [13, 3], and we sampled polynomials and operators randomly for validation as well.

4 Database

The research area has plenty of unsolved problems. Most of the problems have solutions for a specific forms of radix systems. To state and validate various conjectures it is necessary to collect and filter the partial results and sample candidates. Therefore we implemented a server-side application which is able to store various data on number expansions. At present the database contains more than 10 000 items. The uploaded data are about companions of expansive polynomials with constant terms ±2,±3,±5, ±7 together with their number system status and witnesses, etc. We used canonical and symmetric digit sets as well, and we calculated many combinations of product systems.

The data server allows to read data from the server publicly via a JSON API and the registered users with own API token can send new items/properties to the database. The items can be filtered by any custom property.

The server already stores plenty of properties, like eigenvalues, eigenvectors, periodic points and orbits, classification details, etc.

Listing 8: Example of how to request candidates from the public database

```
result = callServer('http://numsys.info/radix-system/list',{
    '.volume':'<1000',
    '.dimension':'3',
    'size':5
})

for r in result:
    rs = RadixSystem(r['base'],r['digits'])
    print(r['base'],r['digits'],rs.smartDecide())

# Result:
# [[0, 0, -2], [1, 0, -2], [0, 1, -2]] [[0, 0, 0], [1, 0, 0]] True
# [[0, 0, -2], [1, 0, 0], [0, 1, -1]] [[0, 0, 0], [1, 0, 0]] False
# [[0, 0, -2], [1, 0, -1], [0, 1, -1]] [[0, 0, 0], [1, 0, 0]] True
# [[0, 0, -2], [1, 0, 1], [0, 1, 0]] [[0, 0, 0], [1, 0, 0]] True
# [[0, 0, -2], [1, 0, 0], [0, 1, 0]] [[0, 0, 0], [1, 0, 0]] True
```

5 Experimental observations

The database helps the researchers filtering out some special data. Analysing the uploaded data we have some observations.

- In general, the non-zero basins are significantly large on average. Based on this observation the randomized method is a viable alternative for checking the GNS property (more detail in Section 3.5.5).
- In general, the cycle lengths are relatively short. Therefore we suggest the length-n cycle method for the decision, if possible (detailed in Section 3.5.4.)
- There is always at least one lattice point that leads to a non-zero periodic point in the $|\det(M)|$ neighbourhood of the origin (in infinity norm).
- More than 700 samples in the database suggested the following theorem:

Theorem 3. *Suppose that the system* (Λ, M, D) *is GNS. Then* (Λ, M^n, D_n) *is GNS for all* $n \in \mathbb{N}$*, where*

$$D_n = \{d_0 + Md_1 + M^2d_2 + \ldots + M^{n-1}d_{n-1} \ : \ d_i \in D\}$$

taking all possible combinations for the digits d_i *above.*

Proof. Let $n > 1$ be fixed. Since (Λ, M, D) is GNS therefore all $x \in \Lambda$ can be written uniquely in the form

$$x = d_0 + Md_1 + \cdots + M^k d_k \,, \tag{1}$$

where $d_i \in D$. Equation (1) can be rewritten as

$$x = (d_0 + \cdots + M^{n-1}d_{n-1}) + M^n(d_n + \cdots + M^{n-1}d_{2n-1}) + \cdots + M^k d_k \,.$$

Since the coefficients of each M^{nj} are digits from D_n for all $j \geq 0$ therefore the system (Λ, M^n, D_n) is GNS as well. $\square$

6 Conclusion and further work

The paper introduced a toolset for supporting lattice-based number expansion computations. The toolset was implemented in Python. Besides, the authors built a database storing different radix system parameters and offers the researchers to upload and search in this database. In the future we plan to improve, extend and distribute the toolset and try to find a mathematical proof for some of our observations.

References

[1] Akiyama, S. and Rao, H. New criteria for canonical number systems. *Acta Arithmetica*, 111(1):5–25, 2004. DOI: 10.4064/aa111-1-2.

[2] Brunotte, H. On trinomial bases of radix representations of algebraic integers. *Acta Scientiarum Mathematicarum*, 67(3–4):521–527, 2001.

[3] Burcsi, P. and Kovács, A. Exhaustive search methods for CNS polynomials. *Monatshefte für Mathematik*, 155(3-4):421, 2008. DOI: 10.1007/s00605-008-0005-y.

[4] Burcsi, P., Kovács, A., and Papp-Varga, Zs. Decision and classification algorithms for generalized number systems. *Ann. Univ. Sci. Budapest. Sect. Comput*, 28:141–156, 2008.

[5] Germán, L. and Kovács, A. On number system constructions. *Acta Mathematica Hungarica*, 115(1-2):155–167, 2007. DOI: 10.1007/s10474-007-5224-5.

[6] Hudoba, P. and Kovács, A. Some improvements on number expansion computations. *Numeration 2016*, page 65, 2017.

[7] Kátai, I. *Generalized number systems and fractal geometry.* Janus Pannonius Tudományegyetem, Pécs, 1995.

[8] Kovács, A. On the computation of attractors for invertible expanding linear operators in z (kappa). *Publicationes Mathematicae Debrecen*, 56(1-2):97–120, 2000.

[9] Kovács, A. *Number Systems in Lattices.* PhD thesis, Eötvös Loránd University, Budapest, Hungary, 2001.

[10] Kovács, A. Number expansions in lattices. *Mathematical and Computer Modelling*, 38(7-9):909–915, 2003. DOI: `10.1016/S0895-7177(03)90076-8`.

[11] Kovács, B. Canonical number systems in algebraic number fields. *Acta Mathematica Academiae Scientiarum Hungarica*, 37(4):405–407, 1981. DOI: `10.1007/BF01895142`.

[12] Kovács, B. Integral domains with canonical number systems. *Publ. Math. Debrecen*, 36:153–156, 1989.

[13] Pethő, A. On a polynomial transformation and its application to the construction of a public key cryptosystem. *Computational Number Theory*, pages 31–43, 1991. DOI: `10.1515/9783110865950.31`.

[14] Tátrai, A. Parallel implementations of brunotte's algorithm. *Journal of Parallel and Distributed Computing*, 71(4):565–572, 2011. DOI: `10.1016/j.jpdc.2010.12.010`.

[15] Vince, A. Replicating tessellations. *SIAM Journal on Discrete Mathematics*, 6(3):501–521, 1993. DOI: `10.1137/0406040`.

Acta Cybernetica **25** (2021) 285–306.

Taxonomy for The Trade-off Problem in Distributed Telemedicine Systems*

Zoltán Richárd Jánki[ab] and Vilmos Bilicki[ac]

Abstract

Web systems are facing a great challenge because of the increasing amounts of data and demand for features. By meeting these requirements, distributed systems have gained ground, but they bring their own problems as well. These issues are present in telemedicine. Since telemedicine is a wide field, various phenomena have different effects on the data. Availability and consistency play important roles in telemedicine, but since the CAP and PACELC theorems describe the trade-off problem, no one can guarantee both capabilities simultaneously. Our study seeks to get an in-depth view of the problem by considering real world telemedicine use-cases and we present an easily tuneable system with a taxonomy that assists the design of telemedicine systems. Model checking verifies the correctness of our model and data quality measurements. During the evaluation, we found interesting states and the consequence of this is called *hypothetical-zero-latency.*

Keywords: taxonomy, data quality, cache, trade-off, telemedicine, distributed system

1 Introduction

Telemedicine is one of the areas of healthcare that is developing quickly and it is finding a place in modern medicine. The number of electronic healthcare records (EHR) is not only growing rapidly, but it raises several Information Technology (IT) issues as well. In the past decades, several theoretical and practical IT solutions have eased the continuously arising problems, like standardizations, systems, tools and cloud solutions. Naturally, new solutions should address new issues, so it is a neverending story [8].

Installing standardization can markedly influence the behaviour of a system. In Telemedicine, the well-known Health Level Seven's (HL7) Fast Healthcare Interoperability Resources (FHIR) specification [13] is a widely accepted and used

*This research was supported by the EU-funded Hungarian grant EFOP-3.6.1-16-2016-00008 and it was also supported by the 2018-1.1.1-MKI-2018-00249 project and the Dericom Ltd.

[a]Department of Software Engineering, University of Szeged, Hungary

[b]E-mail: `jankiz@inf.u-szeged.hu`, ORCID: `0000-0003-1829-5663`

[c]E-mail: `bilickiv@inf.u-szeged.hu`, ORCID: `0000-0002-7793-2661`

DOI: `10.14232/actacyb.290352`

standard that was elaborated for exchanging healthcare information electronically. It provides a loose data model for developers that describes the different entities of healthcare well. In telemedicine, not just the data model can be standardized, but also the communication among the services.

As the size of the databases - containing EHRs - is growing quickly, telemedicine systems are continuously developing. Moreover, there is a significant number of computing tasks in healthcare that require a variety of resources. Most of the interconnected telemedicine applications are Web-based and in many cases, the backbone is a distributed system. In most of telemedicine use-cases, data paths between endpoints are complex, so simple client-server architectures are very uncommon today. A complex data path contains plenty of servers, caches, computational units that make aggregations on data and serve readily available services. So, system logic and data storages are scattered and systems consist of most than just a thin client and a monolithic server. Recent mobile end devices have unused resources, but computing tasks are resource-intensive processes. However, there are privacy concerns regarding data storages. In many cases, regulations do not allow us to keep patient data in remote data centers. Thus, in telehealth, fog computing is becoming more and more popular. In fog computing, computation tasks are outsourced to edge devices in order to keep data as close to the source as possible. Kraemer et al. introduced use-cases in [17], and how complex data paths can be created.

Sometimes applying fog computing is not necessary or not feasible, but closeness of data is essential because of huge communicational distances. Long data paths can lead to noticeable delays. In order to minimize latency between clients and servers, caches can be placed on data paths. Content Delivery Networks (CDN) form the so-called transparent backbone of Internet in charge of content delivery. As they effectively shorten physical distances, latencies are reduced. CDN stores a cached version of content at different geographical locations in order to make it available for many different locations far away from each other. CDNs are not only used in industry sectors, but also in telemedecine [10].

Besides the advantages of distributed systems, there are some disadvantages as well. Eric Brewer states that there are no distributed systems that can guarantee at most two of the three desirable properties: consistency (C), availability (A) and partition-tolerance (P) [6]. It is hard to find the right balance among the properties mentioned in the CAP theorem. In our recent paper [15], we presented a system model that provides an approach to resolving the consistency and availability trade-off problem of distributed systems. We examined the data path of one telemedicine use-case and checked all the possible states in order to ascertain where we can use caches and how we must configure them in order to guarantee a strong consistency level. This paper completes our previous work with a taxonomy that classifies telemedicine use-cases by considering the offline status where real-world examples and data paths are attached to the groups. Based on the strength of measured consistency, we also calculated the quality of data and the model checking produced an interesting phenomenon of distributed systems.

2 Related work

There are big challenges in many countries on account of the aging population and the rise in chronic diseases while trying to reduce costs, but maintain high-quality care for patients. Fortunately, telemedicine can reduce the burden on nurses and practitioners. The number and variety of telemedicine applications is continuously increasing and finding uses. In 2020, in Hungary, the legislative options of teleconsultation was initiated in healthcare [14].

One of the most important requirements is integrability when designing a telemedicine system. Standardized systems can readily exchange healthcare data among themselves. HL7's FHIR [13] is one of the most well-known standards that improves system integrability. Although FHIR was designed for relational database systems, it can be adapted to NoSQL database systems as well.

Choosing the most appropriate database system for a project is a big challenge. We have to take into account the fact that cloud solutions are widespread, and they are used not just for common data storage and computing, but also in telemedicine [25]. Clouds have enhanced the use of distributed systems, but they may introduce several problems in spite of increasing data and transactional throughput and placing data near clients. Eric Brewer's CAP theorem clearly describes the limitations of such a system, but it does not constrain the capabilities of a system. Daniel J. Abadi introduced the so-called PACELC theorem [1], which is an extension of CAP. PACELC states that in the case of network partitioning (P), a trade-off has to be made between availability (A) and consistency (C), but else (E), when the system is running normally in the absence of partitions, another trade-off has to be made between latency (L) and consistency (C). Since telemedicine is diverse, it is not trivial to find the proper balance between the capabilities when designing a system. Thus, an appropriate taxonomy can help designers to develop a telemedicine system that most effectively meets all functional and non-functional requirements.

Peter Bailis et al. presented the Probabilistically Bounded Staleness (PBS) method [2] that shows how much time has to elapse for eventual consistency in quorum-replicated data stores like Apache Cassandra. In their study, t-visibility and k-staleness metrics describe the trade-off between availability and consistency, and the WARS model represents latency. Their results were obtained by Monte Carlo simulations and good approximations can be achieved. Although these metrics describe the problem very well, the results could be more accurate if a formal system model was developed and the entire graph space was analyzed.

Furthermore, Microsoft designed Azure Cosmos DB as a tuneable database system with 5 consistency levels starting from strong to eventual [22]. They elaborated a system specification using the Temporal Logic of Actions (TLA) and its TLA+ formal language, and evaluated their model using TLA Checker (TLC) [18]. Amazon also created TLA+ specs about their systems [23]. TLA+ and TLC together form a valuable toolkit because instead of making approximations, they construct a state graph from the possible states that the checked system can go into and make a graph traversal. We used the same toolkit for finding the proper trade-off between availability and consistency in our telemedicine systems. Also, the whole state space is available after the execution of TLC, so every possible state can be analyzed separately. However, TLA+ is not the only formal language that can model a system in action. Maude [4][21] is another specification language and tool for modelling distributed systems. Lots of tools were implemented that can work with Maude and can be used for specific models. Since TLA+ and its toolbox is always kept up-to-date and contains everything in one place, we decided to use TLA+.

These studies focus on in-system behaviour, even though there are also external factors that can significantly influence the availability and consistency of distributed systems. Quality of Service (QoS) gives the overall performance of a service. Phumzile Malindi [20] collected the demanded requirements of network parameters after taking into account different telemedicine areas. These parameters were throughput, delay, jitter and context. It was shown via simulations how a network should be configured and which data compression guarantees better quality. These parameters can help us to perform a more accurate and realistic state graph analysis in a system.

Lastly, many studies investigated how latency affects different telemedicine areas [3][16], but only a few of them were concerned with consistency in telehealth. Although Nekane Larburu et al. used delay and consistency parameters for Quality of Data (QoD) measurements in a Clinical Decision Support System (CDSS) [19], they did not use metrics to measure the consistency of the MobiGuide system. We made metric-based evaluations in a telemedicine system that shows how latency affects both consistency and data quality.

3 Our results

In our paper, we explain the following results in distributed telemedicine systems.

- Here, a new methodology for modelling information critical heterogenous systems: we stated formal definitions of processes in complex distributed healthcare systems. Based on system model evaluations, we elaborate a taxonomy that makes suggestions for particular telemedicine use-cases on how the data path should be constructed.

- The verification of information critical systems and metrics: the implemented system model is verified via a model checker that builts up a state graph containing possible states of the system. We evaluate the whole state graphs by comparing consistency among different states.

- New metrics for reliability of data: we present a distance-based metric for the data quality measurement of numeric data portions in telemedicine systems. Moreover, after evaluating state graphs of our system models, we will visualize the trend of data quality during graph traversal.

4 Challenges

Creating telemedicine system specs raised a number of questions that we listed as challenges. In this section, we will focus on the affected areas listed in Table 1.

Table 1: List of challenges in distributed telemedicine systems.

No.	**Challenge**
1	Consistency measurement
2	Trade-off between availability and consistency
3	Offline states
4	QoS
5	QoD

4.1 Consistency measurement

Performing a consistency measurement is not a simple problem. First of all, one needs to find proper metrics that describe consistency well. Furthermore, consistency can be measured via simulations and model checking. Eventually, consistency may vary due to the behaviour of external components. Our system specs and verifications are fine for using metrics of the PBS model with model checking and also for taking into account external factors. Here, k-staleness is the parameter for finely tuning the consistency of any part of the data path. A lower k value guarantees stronger consistency. In this context, consistency can be defined as in Definition 1.

Definition 1. *Any read on a data item x returns a value corresponding to the result of at most a k-version older write on x.*

$$(x_n \in X) \wedge (x_{n-k} \rightarrow x_n) \Rightarrow (x_{n-k} \in X)$$

4.2 A trade-off between availability and consistency

Firstly, it is difficult to choose the best database system for a project and it is almost impossible to find one that is universally acceptable. Since there are many different areas in telemedicine, which database system is the best choice depends on the context. Recently, most of the telemedicine systems use some kind of cloud solution and they are based on distributed systems, hence as the CAP and PACELC theorems state, a trade-off between availability and consistency has to be made. It is not clear which outcome of trade-off is the best, because it is always context dependent, especially in telemedicine. One solution is the so-called polyglot persistence where we use multiple data storage technologies and they are varied according to needs across an application. Categorizations for database systems using the PACELC theorem [9] have been published. For this purpose, we created a Software Development Kit (SDK) in which database systems are easily interchangeable across an application. Furthermore, we modelled and verified a system in which trade-off is easily tuneable with a k-staleness parameter.

4.3 Offline states

Secondly, any part of a system can go in an offline state. Offline state in telemedicine means that the client application or the server-side is not available at a given time. It is also possible that both of them are out of operation. The offline state and availability are closely related, since offline status can lower availability. However, it can be related to partitioning depending on the telemedicine use-case. In case of a system that is designed for general practitioners for daily usage, partitioning cannot cause problems because only one entity writes data in the system. On the other hand, in a remote otolaryngological system, when a general practitioner and more than one specialist can make diagnosis, partitioning can cause problems. In case of partitioning, consistency can be eventually guaranteed, but the system operates continuously. In order to keep availability at a high level, we recommend placing content delivery networks (CDN) and caches on the data path. Although caches assist availability, consistency must be abandoned. In our system model, we placed caches on the data path and configured it with a staleness parameter in order to get a high consistency level.

4.4 Quality of Service

QoS is a challenge in telemedicine systems that affects mainly realtime services. Teleconsultation and telesurgery are the two most common areas of realtime telemedicine. However, there are services in these categories, but they also require realtime communication. There is a close connection between QoS parameters and networking, and network configuration can improve QoS. These configurations can be implemented in the patient's home and in the clinic as well.

4.5 Quality of Data

The QoD measurement is another context dependent concept. Its application greatly depends on the type of dataset and the goal of data usage. In telemedicine systems, the most rapidly changing data portions are numeric data sets. QoD calculations usually contain an aggregation of distance function results that describe inconsistencies between the real-world phenomena and the data obtained from resources. Latency can be found everywhere in a system as a variable and it strongly affects data quality. Hinrichs' Definition 2 describes what QoD means in our context, in which x_{db} represents data stored in a database and x_{real} stands for real-world data at a given t point [19]. We took measurements on the data quality and learned how data can be corrupted due to latency and multiple data-process instances.

Definition 2.

$$Q(x) := \begin{cases} \dfrac{1}{d(x_{db}, x_{real}) + 1}, & \textit{if } x_{db} \neq x_{real} \\ 1, & \textit{otherwise} \end{cases}$$

5 Telemedicine use-cases

We examined data paths in active telemedicine projects to see whether the offline status is allowed or not. Offline capability is important because it can efficiently improve availability even when consistency worses. However, in our recent paper [15], we showed that consistency level can also be increased if we set constraints on the maximal staleness of the data. There, we introduced our modelling approach and constructed the first formal definitions of the core processes in a distributed telemedicine system. It was also shown how the metrics are used during the model checking, and how the results can be evaluated. In that study, we proposed what benefits and drawbacks can arise from using lower k parameter, but in a smaller state graph. In Table 2, we list some real telemedicine scenarios with their availability and consistency requirements. Mainly, these use-cases determine the development of our taxonomy. Figure 1 shows a schematic illustration of these telemedicine scenarios.

Table 2: Real telemedicine use-cases with a trade-off

Telemedicine use-case	**Availability**	**Consistency**	**Offline status enabled**
Teleconsultation with remote monitoring		X	
Da Vinci surgery		X	
Remote diagnostic in otolaryngology	X		X
Remote monitoring in ICU		X	X
Remote monitoring in CAPD	X		X
Remote monitoring in spirometry	X		X

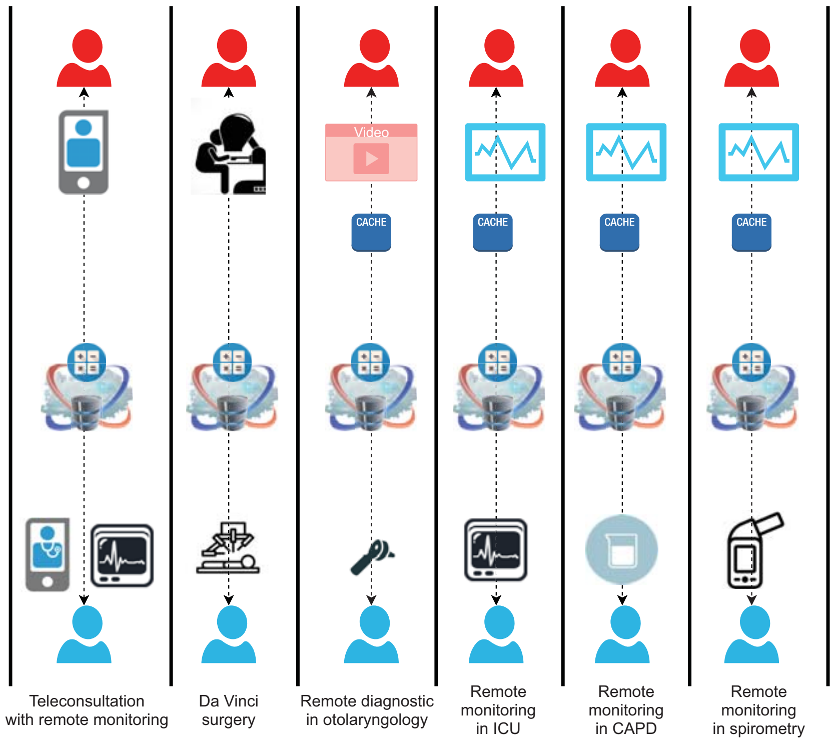

Video
CACHE
CACHE
CACHE
CACHE
Teleconsultation with remote monitoring
Da Vinci surgery
Remote diagnostic in otolaryngology
Remote monitoring in ICU
Remote monitoring in CAPD
Remote monitoring in spirometry

Figure 1: Data paths in real telemedicine use-cases

5.1 Teleconsultation with remote monitoring

In this use-case, a video conference is set up between a patient and doctor while the remote monitoring of vital signs is being performed. Both video chat and monitoring are carried out in realtime, and due to this the need for consistency overrides availability. Since consistency is crucial here, the system does not include caches. Sometimes raw data can be aggregated (e.g.: ECG signal processing) and this can cause inconsistencies for a short time irrespective of the consistency configurations.

5.2 Da Vinci surgery

Da Vinci surgery is one of the most well-known telesurgery methods that is used in several surgical procedures. The operation is carried out by a specialist, who controls a robot remotely while the patient's vital signs are monitored remotely. As a surgical procedure also occurs in realtime, consistency is preferred over availability. Caching is disabled in order not to lose a high consistency level. Some aggregations may also occur in the cloud, and this can lead to inconsistent states for a certain period. QoS is very important in telesurgery, because high latency can make a surgical procedure unstable. A stable network connection, minimized jitter and delay are all necessary for this.

5.3 Remote diagnostic in otolaryngology

One of our telemedicine projects that was supported by the European Union (EU) is the development of a remote diagnostic system in otolaryngology [5]. Patients visit their general practitioner, who does not hospitalize patients, but takes photos and records a video of body areas of patients. After uploading images and videos, a referral is forwarded to a medical specialist who makes a diagnostic report based on the received frames. There are no realtime communications between two doctors, and no recent updates can be found in the data path, so availability has a higher priority than consistency. Eventual consistency is sufficient in this system, so we can further improve availability with caches. Also, there is no possibility of staleness in the data.

5.4 Remote monitoring in ICU

The Intensive Care Unit (ICU) is a department of a hospital that provides intensive care medicine to patients with life-threatening illnesses. Here, continuous remote monitoring is essential and often decisions have to be made in seconds. Thus consistency is more important than availability. However, availability also plays an important role, because in an offline state, the last visible record may be decisive. For this purpose, caches can be placed on data paths, but they have to be configured with low k parameter values in order to get up-to-date data.

5.5 Remote monitoring in CAPD

Continuous Ambulatory Peritoneal Dialysis (CAPD) is another EU-financed telemedicine project [5] and its system is maintained by us. This service monitors the patient's dialysis fluid intake and outcome. In addition, blood pressure and weight are measured, but the time elapsed between measurements may be several hours. In this case, eventual consistency is also acceptable, so availability has the highest priority. Since there are no rapid changes (rapid requests) in the data sets, inconsistency can never really occur, so caches can be configured with the higher k parameter values. The decision support system generates alerts if the patient's measured values are over or under a given threshold. Due to the offline state, a possible alert may be missed.

5.6 Remote monitoring in spirometry

Next, spirometry remote monitoring system is a quite similar EU-financed telemedicine project [5]. Patients have medical tests at home and this gives data on the volume of air being inhaled or exhaled as a function of time. Raw data leaves it using a spirometer and it is sent to a mobile device via a Bluetooth connection. Tests are repeated 3 times in one measurement. The mobile device sends raw data to the cloud that makes the following aggregations: FEV1, FVC, PEF, MMEF2575, FEV1/FVC. A pulmonologist is interested in the best aggregated values from 3 tests. A pulmonological evaluation may take place a few hours later, so eventual consistency is appropriate. However, QoD can present interesting phenomena, since tests are performed in seconds, hence raw data is periodically transmitted to the cloud frequently. Here, cloud computing works as a trigger, so due to the distribution, each event starts a new server instance. The events may be processed simultaneously, so the cloud cannot guarantee any ordering of events [11]. This occurrence may lead to inconsistencies that need to be resolved.

6 Our modelling approach

6.1 System model

In this section, we introduce our system spec that is suitable for measuring consistency under different circumstances and it is also capable for finding rare or near-impossible phenomena in a system. Definition 3 shows formal definitions of basic operations and data sets that we have evaluated after executions. *ClientData* is for the data set that the client started to upload to the database. *DBData* represents the actual status of database, while *ProcData* contains data obtained after aggregation. We used a tuple data structure in order to make database instances comparable. Definitions 4, 5 and 6 formally describes processes of our measuring system. Client writes (CW) raw data received from devices or sensors. CW has two possible outcomes, namely a write action if the threshold of the allowed maximum number of operations (*MaxNumOp*) is not exceeded and the system terminates. This is a limitation in the system model used to examine a finite set of possible states of system.

Definition 3.

$$ClientData := (\{x_m, op_m\}, \dots, \{x_1, op_1\}) \tag{1}$$
$$DBData := (\{y_n, op_n\}, \dots, \{y_1, op_1\}) \tag{2}$$
$$ProcData := (\{z_l, op_l\}, \dots, \{z_1, op_1\}) \tag{3}$$

$$ClientWrite(x) := \{x_i, op_i\} \circ ClientData \tag{4}$$
$$DBWrite(y) := \{y_j, op_j\} \circ DBData \tag{5}$$
$$ProcWrite(z) := \{z_k, op_k\} \circ ProcData \tag{6}$$

Definition 4.

$$CW(x) := \begin{cases} ClientWrite(x) & \textit{if } (numOp < MaxNumOp) \\ Termination, & \textit{otherwise} \end{cases}$$

Definition 5.

$$DBW(x) := \begin{cases} DBWrite(x) & \textit{if } (\textbf{len } ClientData > \textbf{len } DBData) \\ Waiting, & \textit{otherwise} \end{cases}$$

Definition 6.

$$DBPROC(x) := \begin{cases} ProcWrite(x) & \textit{if } (\textbf{len } DBData > \textbf{len } ProcData) \\ Waiting, & \textit{otherwise} \end{cases}$$

DBW and *DBPROC* processes work like triggers that are waiting for changes in data sets, thus unnecessary process execution cannot take place. *DBW* describes the process of persistence and *DBPROC* is responsible for data aggregation. After the mathematical definitions, we included TLA+ spec parts as well that can be seen in figures 2, 3 and 4. Since our model describes a distributed telemedicine system, we can have database replicas and multiple computational units. Processes are defined for group of instances, but in order to identify which instance is working currently, different identifiers are assigned to the replicas and server instances. Since only 1 client is present, it is identifed with the id 10 ($pc[10]$). In the *DBW* and *DBPROC* definitions, $pc[self]$ means that the model checker must substitute the proper identifier of the instance for the running process. The client just simply pushes the data until the number of write operations does not reach the threshold. Otherwise, the simulation terminates. If a new data is inserted to the database, it is stored in a list (*ClientRawData*), and operation identifier is assigned to this element. *DBW* and *DBPROC* processes check whether new data has arrived, and if so, makes the persistence and the aggregation.

6.2 The consistency measurement technique

After recalling the CAP and PACELC theorems, the measuring consistency may be the trickiest one of the measurement of 3 desirable capabilities. In order to make a property observable, we have to find proper metrics that describe it. Peter Bailis et al. introduced the PBS method for availability and consistency measurements. It is based-on k and t parameters that denote staleness and visibility values. Using these parameters, they made approximations of the availability and consistency of a quorum-based database system. The Azure Cosmos DB TLA+ system specification told us consistency can be measured not just under a simulation, but also with

```
CW ≜ ∧ pc[10] = "CW"
     ∧ IF (numOp < MaxNumOp)
          THEN ∧ numOp' = numOp + 1
               ∧ finalData' = finalData + 1
               ∧ ClientRawData' = ⟨[d ↦ finalData', op ↦ numOp']⟩ ∘ ClientRawData
               ∧ pc' = [pc EXCEPT ![10] = "CW"]
          ELSE ∧ pc' = [pc EXCEPT ![10] = "Done"]
               ∧ UNCHANGED ⟨finalData, ClientRawData, numOp⟩
     ∧ UNCHANGED ⟨readData, dbLat, calcLat, DbRawData, DbProcData,
                  lenClientRawData, lenDbRawData, latRead, latWrite,
                  latProc⟩
```

Figure 2: The *CW* definition in TLA+

```
DB_W(self) ≜ ∧ pc[self] = "DB_W"
             ∧ IF (Len(ClientRawData) > lenClientRawData)
                  THEN ∧ lenClientRawData' = Len(ClientRawData)
                       ∧ pc' = [pc EXCEPT ![self] = "DB_W_LAT"]
                  ELSE ∧ pc' = [pc EXCEPT ![self] = "DB_W"]
                       ∧ UNCHANGED lenClientRawData
             ∧ UNCHANGED ⟨finalData, readData, dbLat, calcLat,
                          ClientRawData, DbRawData, DbProcData,
                          lenDbRawData, numOp, latRead, latWrite, latProc⟩
```

Figure 3: The *DBW* definition in TLA+

```
DB_PROC(self) ≜ ∧ pc[self] = "DB_PROC"
                ∧ IF (Len(DbRawData) > lenDbRawData)
                     THEN ∧ lenDbRawData' = Len(DbRawData)
                          ∧ pc' = [pc EXCEPT ![self] = "DB_PROC_LAT"]
                     ELSE ∧ pc' = [pc EXCEPT ![self] = "DB_PROC"]
                          ∧ UNCHANGED lenDbRawData
                ∧ UNCHANGED ⟨finalData, readData, dbLat, calcLat,
                             ClientRawData, DbRawData, DbProcData,
                             lenClientRawData, numOp, latRead, latWrite,
                             latProc⟩
```

Figure 4: The *DBPROC* definition in TLA+

logical modelling. Based on these techniques, we elaborated a system specification that combines logical modelling with the k-staleness metric. In our short paper [15], we evaluated our model and examined how latency affects consistency. We showed that we can have inconsistent states if latency exists, but with the k-staleness parameter, we can configure caches on the data path and improve both availability and consistency.

Firstly, we reworked our spec and checked how the consistency level decreases when we have multiple server instances. Secondly, we extended our processes by following the above-mentioned telemedicine use-cases and we were able to measure data quality.

6.3 The QoD measurement technique

After investigating several QoD studies, we opted for a basic distance function - shown in Equation 7 - for data quality measurements [12], and which is suitable for numeric data sets.

$$d(w_{db}, w_{real}) := |w_{db} - w_{real}| \tag{7}$$

We substituted the distance function into Hinrichs correctness metric formula - stated in Equation 8 - and calculated it for each value pairs. Here, w_{db} stands for the value stored in the database and w_{real} denotes the real-world value. This metric is suitable for observing the correct order and it is vital for consistency.

$$Q_{corr.}(w_{db}, w_{real}) := \frac{1}{d(w_{db}, w_{real}) + 1} \tag{8}$$

7 The simulation environment

Figure 5 shows our system spec. We modelled a client that collects data from a Bluetooth device used in telemedicine systems. The raw data obtained is uploaded to the cloud and it remains there. Computational processes are triggered when data is stored and it is aggregated as we did in spirometry with FEV1, PEF and other parameters. After the evaluation is over, the computed data is stored in a database that is needed by a doctor. This environment covers all of our scenarios listed in Section 5.

Each instance was defined as a process in our TLA+ spec, so we created 3 process definitions like the client, database and computational unit, and various latency values were attached to the database and computational operations. Sensor and cache were not modelled as processes because they do not contain logic in this specification.

In order to evaluate the state graph, the model checker must be terminated somewhere. We limited the maximum number of client write operations (*MaxNumOp*) to 10 because model checking produces millions of states and it is rapidly growing when the number of operations are increasing. Latency values present the number of states in the graph that a process must wait after starting a request and before getting a response in case of a client or passing back data in case of a

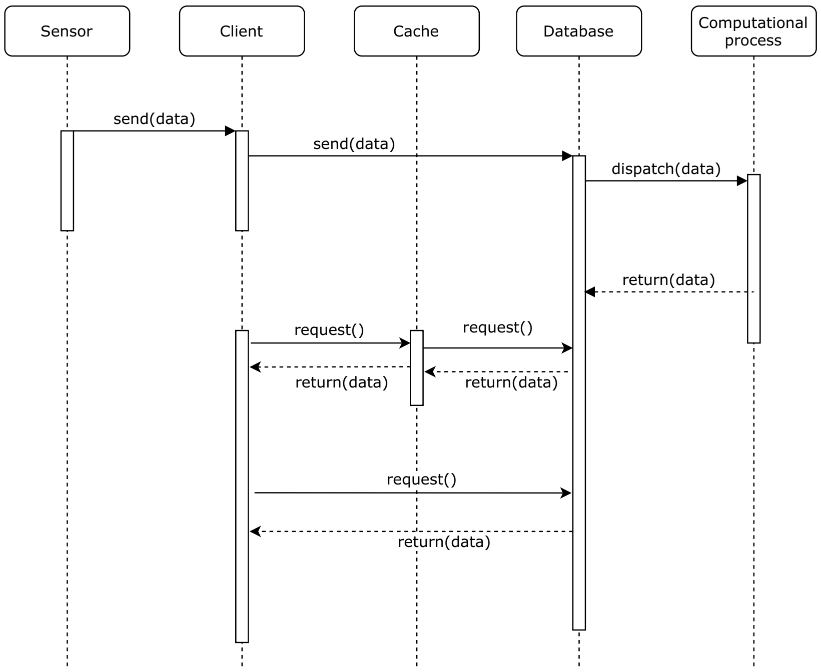


Figure 5: The simulation environment

database or a computational unit. The model checker produce a state graph about what possible states a system has and what values the variables contain at a given state. Latency values were chosen from the $[0-3]$ interval for the client and the servers and $[0-1]$ interval for the cache. Thus, when a client has a latency value 2, after starting the request, it will wait until 2 new states are not present in the state graph that was produced by itself. As it is presented in [24], there are huge differences among different computer actions. During our simulation only Central Processing Unit (CPU) and Random Access Memory (RAM) are used, and the RAM access takes the most of the time in the calculation of a new state in the graph. So, a new state can be generated within 100 nanoseconds. The significant amount of latency is occurred by the network. It is also known that a network connection is almost 10 000 000 times slower than accessing the RAM [7], so increasing the latency by 1 means approximately 100 milliseconds delay in our simulation environment. A delay between 0 and 300 milliseconds can be valid for a client and a server, but data can be retrieved faster from a cache (e.g. a CDN).

The TLC Model Checker produced more than 20 million states and our dot graph file was about 10 GB. There is no available tool that can visualize such huge dot files, but we implemented a transformation script that created a JSON structured file from a raw dot file with an 50% reduction in the file size. Since the original dot files that were produced by TLC contain all the variables and their values for every states, a small modification of the model or a tiny extension of the examined variable intervals can lead to huge file size growth. Our compression script not just transformed the dot format to JSON, but also removed those variables and their values that were unnecessary for the analysis. Also, the lines about transition information were removed because they do not carry information about the variables. The generated dot state identifiers were changed to numbers starting from 0 and incremented by 1 at each new state. Both consistency and data quality evaluations were carried out on the same state graphs with the Python Pandas library.

8 Evaluations

8.1 Consistency results

Earlier we observed that the k-staleness parameter works well in caches, so both availability and consistency can be improved using this technique. Our model checking procedure constructed state graphs that showed how latency affects the consistency level and data set. In order to increase the availability, we tested our environment using cache and the data retrieved from cache was compared to the data that is present in the database. Here, the cache was configured with k-staleness parameter. We modelled multiple database and computational unit instances in order to have a realistic distributed system model. Table 3 shows how latency affects the consistency while varying k values. Here, latency steps are taken in 100 milliseconds as described in Section 7.

Table 3: Consistency evaluation in client-server interaction with given k-staleness parameter values

Latency	$k = 0$	$k = 1$	$k = 2$	$k = 3$
0 unit	92.84%	83.86%	80.47%	78.89%
1 unit	86.67%	79.69%	75.12%	72.46%

We found that if k parameter value is 0, consistency can almost get 100%, but latency can cause drastic decline. Increasing k by one makes approximately 5% reduction in consistency level.

8.2 Data quality results

Besides consistency measurements, we tested data quality on the whole state graph. We learned that through graph traversal the data quality starts to decline. During our evaluations, we found how data quality changes in client-server and server-server interactions.

In the case of client-server communication, we can guarantee a data quality above 90% if we add a restriction on data staleness with $k = 0$ value. Of course, increasing latency reduces QoD, but a cache with lower k value can gain not only the availability of such a system, but also the quality (even with 10%). Lower the k value, higher consistency and data quality can be guaranteed as shown in Table 4. Latency steps are taken in 100 milliseconds.

Table 4: QoD evaluation in client-server interaction with given k-staleness parameter values

Latency	$k = 0$	$k = 1$	$k = 2$	$k = 3$
0 unit	95.47%	89.47%	87.52%	84.63%
1 unit	91.58%	87.13%	84.14%	81.24%

In the case of server-server interaction, we realized that latency can destroy, and also improve data quality in server-server interactions. As the matrix in Figure 6 shows, travelling horizontally and increasing the latency of computational processes, QoD may be reduced, but going vertically down and enlarging the latency of persistence, we can see a change for the better. And, the highest data quality level can be achieved in the $3 - 3$ units position. If we assume that in case of telesurgery, an aggregated heart rate value is provided in every 5 seconds, and every fifth calculation result differs by 1 from the correct one, the QoD value is only $90,08\%$ at the end of a one-hour long term. If the difference in every fifth aggregation is 2, the QoD is only $86,78\%$. Although these are small differences in aggregations, and they do not have significant effects from the patient's point of view, the QoD values are much lower than the ones that our simulations produced. Of course, data quality can be further improved if we add more latency units to the model.

DB\Comp	0 *unit*	1 *unit*	2 *units*	3 *units*
0 *unit*	98.96%	99.16%	99.16%	99.14%
1 *unit*	99.29%	99.44%	99.45%	99.46%
2 *units*	99.42%	99.55%	99.56%	99.58%
3 *units*	99.52%	99.64%	99.65%	99.67%

Figure 6: Consistency changes with a latency increase

While evaluating data quality values, we realized some interesting states in the graph.

If we assume that - in a perfect world - there is no latency anywhere, we still cannot guarantee in a distributed system that the data quality will be 100%.

If no latency exists in a world, then raw data comes from client quasi simultaneously. Since there is only one client who uploads data, there is a known correct order of data, but they arrive at the same time as the computation process. Thus, in distributed systems perhaps more than one process will be started at the same time, and there is no guarantee about which raw data part will be skipped by them. We called this phenomenon *hypothetical-zero-latency.*

Based on these results, we constructed a taxonomy for distributed telemedicine systems.

9 Taxonomy for the trade-off problem

During our previous studies in telemedicine, we encountered various classifications of telemedicine services. To our knowledge, there are no categorization in telemedicine that can be applied for measuring availability and consistency in distributed telemedicine systems.

We elaborated a taxonomy that focuses on this trade-off problem and classifies telemedicine use-cases by considering the requirements for availability and consistency. Taxonomy breaks down telemedicine areas starting with offline capability of systems and the degree of staleness in data. Based on these categories, we can make evaluations in different use-cases and make suggestions for system developers on how to solve the trade-off problem stated in the CAP and PACELC theorems. Our classification scheme is presented in Table 5.

Table 5: Trade-off taxonomy

Use-case category	**Possibility of staleness**	**Recommendation for system selection**
Non-offline telemedicine	No	PC/EL without cache
	Yes	PC/EC without cache
Semi-offline telemedicine	No	PC/EL with cache higher k-staleness parameter
	Yes	PC/EL with cache lower k-staleness parameter
Offline telemedicine	No	PA/EL with cache higher k-staleness parameter
	Yes	PA/EL with cache higher k-staleness parameter

If the offline status is not enabled but staleness of data occurs (however, this is rare), we recommend using a PC/EL database system and avoid using caches. Consistency is really important in such situation and teleconsultation use-cases are mainly in this category, so availability plays an important role. As regards non-offline telemedicine, if staleness is likely to occur, PC/EC systems are suggested, because strong consistency is required. Telesurgery use-cases cover this class. Going forward, for the semi-offline category, PC/EL systems are the recommended ones with parameterized caches. Such telemedicine services can go into an offline state and they are occasionally realtime. An offline state is not a problem, if a high level of availability is guaranteed. Lastly, in offline telemedicine longer offline periods are permitted, but caches are highly recommended in order to prevent a reduction in availability, so PA/EL systems are recommended here. Figures 7, 8, 9, 10, 11, 12 show the formal definitions of the system that applies this classification. If the data is available, it is stored with the given system configurations. Hence, a polyglot persistence can be performed, and telemedicine applications can be served with the most optimal settings. *NO_NS* and *NO_S* denotes the non-offline telemedicine cases with and without possible data staleness. In these classes caches are disabled on the data path because consistency is preferred. *SO_NS* and *SO_S* are the semi-offline cases when caching is allowed and k-staleness parameters are configured as the taxonomy states because high availability is important, but a high consistency level is also needed and it can be guaranteed with a lower k parameter value. Lastly, *O_NS* and *O_S* classes permit an offline status with relatively high k-staleness parameter values because availability is preferred to consistency.

```
$NO\_NS(self) \triangleq$ $\land\ pc[self] = \text{"NO\_NS"}$
                  $\land$ IF $(lat\_db[self] < db\_latency)$
                        THEN $\land\ lat\_db' = [lat\_db$ EXCEPT $![self] = lat\_db[self] + 1]$
                             $\land\ pc' = [pc$ EXCEPT $![self] = \text{"NO\_NS"}]$
                             $\land$ UNCHANGED $\langle db\_type, dbData, Cache \rangle$
                        ELSE $\land\ db\_type' = \text{"PC/EL"}$
                             $\land\ Cache' =$ FALSE
                             $\land\ dbData' = \langle [d \mapsto (Head(clientData).d),\ type \mapsto db\_type'] \rangle \circ dbData$
                             $\land\ pc' = [pc$ EXCEPT $![self] = \text{"DB"}]$
                             $\land$ UNCHANGED $lat\_db$
                  $\land$ UNCHANGED $\langle db\_latency, proc\_latency, clientData, procData,$
                            $readData, cachedData, K, num\_op, data, lat\_proc,$
                            $d \rangle$
```

Figure 7: Formal definition of the process from the *NO_NS* class

10 Conclusions

In our research, we found that the trade-off problem - presented in the CAP and PACELC theorems - can have significant effects in different telemedicine use-cases. Related works described the consequences of having an inappropriate balance between availability and consistency. Our experiences in real-world telemedicine scenarios helped us to demonstrate how our system can be easily tuned and adapted under different circumstances. We introduced a new methodology for modelling information critical heterogenous systems, and verified these systems and metrics by constructing state graphs and evaluating them via graph traversal. Moreover,

```
NO_S(self) ≜ ∧ pc[self] = "NO_S"
             ∧ IF (lat_db[self] < db_latency)
                  THEN ∧ lat_db' = [lat_db EXCEPT ![self] = lat_db[self] + 1]
                       ∧ pc' = [pc EXCEPT ![self] = "NO_S"]
                       ∧ UNCHANGED ⟨db_type, dbData, Cache⟩
                  ELSE ∧ db_type' = "PC/EC"
                       ∧ Cache' = FALSE
                       ∧ dbData' = ⟨[d ↦ (Head(clientData).d), type ↦ db_type']⟩ ∘ dbData
                       ∧ pc' = [pc EXCEPT ![self] = "DB"]
                       ∧ UNCHANGED lat_db
             ∧ UNCHANGED ⟨db_latency, proc_latency, clientData, procData,
                          readData, cachedData, K, num_op, data, lat_proc,
                          d⟩
```

Figure 8: Formal definition of the process from the *NO_S* class

```
SO_NS(self) ≜ ∧ pc[self] = "SO_NS"
              ∧ IF (lat_db[self] < db_latency)
                   THEN ∧ lat_db' = [lat_db EXCEPT ![self] = lat_db[self] + 1]
                        ∧ pc' = [pc EXCEPT ![self] = "SO_NS"]
                        ∧ UNCHANGED ⟨db_type, dbData, K, Cache⟩
                   ELSE ∧ db_type' = "PC/EL"
                        ∧ Cache' = TRUE
                        ∧ K' = 3
                        ∧ dbData' = ⟨[d ↦ (Head(clientData).d), type ↦ db_type']⟩ ∘ dbData
                        ∧ pc' = [pc EXCEPT ![self] = "DB"]
                        ∧ UNCHANGED lat_db
              ∧ UNCHANGED ⟨db_latency, proc_latency, clientData, procData,
                           readData, cachedData, num_op, data, lat_proc, d⟩
```

Figure 9: Formal definition of the process from the *SO_NS* class

```
SO_S(self) ≜ ∧ pc[self] = "SO_S"
             ∧ IF (lat_db[self] < db_latency)
                  THEN ∧ lat_db' = [lat_db EXCEPT ![self] = lat_db[self] + 1]
                       ∧ pc' = [pc EXCEPT ![self] = "SO_S"]
                       ∧ UNCHANGED ⟨db_type, dbData, K, Cache⟩
                  ELSE ∧ db_type' = "PC/EL"
                       ∧ Cache' = TRUE
                       ∧ K' = 0
                       ∧ dbData' = ⟨[d ↦ (Head(clientData).d), type ↦ db_type']⟩ ∘ dbData
                       ∧ pc' = [pc EXCEPT ![self] = "DB"]
                       ∧ UNCHANGED lat_db
             ∧ UNCHANGED ⟨db_latency, proc_latency, clientData, procData,
                          readData, cachedData, num_op, data, lat_proc, d⟩
```

Figure 10: Formal definition of the process from the *SO_S* class

we presented a distance-based metric for the data quality measurement of numeric data portions in telemedicine systems. Since the mentioned use-cases are real tele-

```
$O\_NS(self) \triangleq$ $\land\ pc[self] = \text{“O\_NS”}$
      $\land$ IF $(lat\_db[self] < db\_latency)$
            THEN $\land\ lat\_db' = [lat\_db$ EXCEPT $![self] = lat\_db[self] + 1]$
                 $\land\ pc' = [pc$ EXCEPT $![self] = \text{“O\_NS”}]$
                 $\land$ UNCHANGED $\langle db\_type, dbData, K, Cache\rangle$
            ELSE $\land\ db\_type' = \text{“PA/EL”}$
                 $\land\ Cache' =$ TRUE
                 $\land\ K' = 5$
                 $\land\ dbData' = \langle[d \mapsto (Head(clientData).d), type \mapsto db\_type']\rangle \circ dbData$
                 $\land\ pc' = [pc$ EXCEPT $![self] = \text{“DB”}]$
                 $\land$ UNCHANGED $lat\_db$
      $\land$ UNCHANGED $\langle db\_latency, proc\_latency, clientData, procData,$
                 $readData, cachedData, num\_op, data, lat\_proc, d\rangle$
```

Figure 11: Formal definition of the process from the *O_NS* class

```
$O\_S(self) \triangleq$ $\land\ pc[self] = \text{“O\_S”}$
      $\land$ IF $(lat\_db[self] < db\_latency)$
            THEN $\land\ lat\_db' = [lat\_db$ EXCEPT $![self] = lat\_db[self] + 1]$
                 $\land\ pc' = [pc$ EXCEPT $![self] = \text{“O\_S”}]$
                 $\land$ UNCHANGED $\langle db\_type, dbData, K, Cache\rangle$
            ELSE $\land\ db\_type' = \text{“PA/EL”}$
                 $\land\ Cache' =$ TRUE
                 $\land\ K' = 3$
                 $\land\ dbData' = \langle[d \mapsto (Head(clientData).d), type \mapsto db\_type']\rangle \circ dbData$
                 $\land\ pc' = [pc$ EXCEPT $![self] = \text{“DB”}]$
                 $\land$ UNCHANGED $lat\_db$
      $\land$ UNCHANGED $\langle db\_latency, proc\_latency, clientData, procData,$
                 $readData, cachedData, num\_op, data, lat\_proc, d\rangle$
```

Figure 12: Formal definition of the process from the *O_S* class

medicine systems as well, it is planned to make measurements during their project pilots using event-based thechniques for tracking the applications and the servers. System modelling and data quality measurements helped us to elaborate a taxonomy for distributed telemedicine systems based on the trade-off problem and explore *hypothetical-zero-latency* phenomenon. In the future, we plan to extend our current taxonomy, introduce more use-cases that can be categorized and examine *hypothetical-zero-latency cases* in greater detail.

References

[1] Abadi, Daniel. Consistency tradeoffs in modern distributed database system design: CAP is only part of the story. *IEEE Computer*, 45:37–42, 2012. DOI: `10.1109/MC.2012.33`.

[2] Bailis, Peter, Venkataraman, Shivaram, Franklin, Michael, Hellerstein, Joseph,

and Stoica, Ion. Probabilistically bounded staleness for practical partial quorums. *Proceedings of the VLDB Endowment*, 5, 2012. DOI: `10.14778/2212351.2212359`.

[3] Bhandari, Sabin, Sharma, Shree Krishna, and Wang, Xianbin. Latency minimization in wireless IoT using prioritized channel access and data aggregation. In *GLOBECOM 2017 — 2017 IEEE Global Communications Conference*, 2017. DOI: `10.1109/GLOCOM.2017.8255038`.

[4] Bobba, Rakesh, Grov, Jon, Gupta, Indranil, Liu, Si, Meseguer, Jose, Ölveczky, Peter, and Skeirik, Stephen. *Survivability: Design, Formal Modeling, and Validation of Cloud Storage Systems Using Maude.* In *Assured Cloud Computing*, pages 10–48. Wiley, 2018. DOI: `10.1002/9781119428497.ch2`.

[5] Boromisza, Piroska. A XVI. egészségügyi infokommunikációs konferenciáról jelentjük. *IME - Interdiszciplináris Magyar Egészségügy*, 17:55–57, 2018.

[6] Brewer, Eric. CAP twelve years later: How the "rules" have changed. *Computer*, 45:23–29, 2012. DOI: `10.1109/MC.2012.37`.

[7] F. Silva, Tiago. The good, the bad and the ugly in software development. `https://tiagodev.wordpress.com/tag/event-loop/`. Accessed: 2021-03-15.

[8] Gamal, Aya, Barakat, Sherif, and Rezk, Amira. Standardized electronic health record data modeling and persistence: A comparative review. *Journal of Biomedical Informatics*, 114:103670, 2021. DOI: `10.1016/j.jbi.2020.103670`.

[9] Gessert, Felix, Wingerath, Wolfram, Friedrich, Steffen, and Ritter, Norbert. NoSQL database systems: A survey and decision guidance. *Computer Science — Research and Development*, 32:353–365, 2017. DOI: `10.1007/s00450-016-0334-3`.

[10] GlobalDots. `https://www.globaldots.com/content-delivery-network-explained`. Accessed: 2020-09-29.

[11] Google. Extend cloud firestore with cloud functions. `https://firebase.google.com/docs/firestore/extend-with-functions`. Accessed: 2020-09-29.

[12] Heinrich, Bernd, Kaiser, Marcus, and Klier, Mathias. How to measure data quality? — A metric based approach. In *International Conference on Information Systems*, 2007.

[13] HL7. FHIR overview. `https://www.hl7.org/fhir/overview.html`. Accessed: 2020-09-25.

[14] Hungarian Government. Magyar közlöny, 2020. 157/2020. (IV. 29.), Accessed: 2020-09-25.

[15] Jánki, Zoltán Richárd and Bilicki, Vilmos. Crosslayer cache for telemedicine. In *The 12th Conference of PhD Students in Computer Science*, pages 159–163, 2020.

[16] Kibsgaard, Martin and Kraus, Martin. Measuring the latency of an augmented reality system for robot-assisted minimally invasive surgery. In *Proceedings of the 12th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications — GRAPP, (VISIGRAPP 2017)*, pages 321–326. INSTICC, SciTePress, 2017. DOI: `10.5220/0006274203210326`.

[17] Kraemer, Frank, Bräten, Anders, Tamkittikhun, Nattachart, and Palma, David. Fog computing in healthcare — A review and discussion. *IEEE Access*, 5:9206–9222, 2017. DOI: `10.1109/ACCESS.2017.2704100`.

[18] Lamport, Leslie, Matthews, John, Tuttle, Mark, and Yu, Yuan. Specifying and verifying systems with TLA+. In *Proceedings of the 10th Workshop on ACM SIGOPS European Workshop*, EW 10, pages 45–48, New York, NY, USA, 2002. Association for Computing Machinery. DOI: `10.1145/1133373.1133382`.

[19] Larburu, Nekane, Bults, Richard, and Hermens, Hermie. Quality-of-data management for telemedicine systems. *Procedia Computer Science*, 63:451–458, 2015. DOI: `10.1016/j.procs.2015.08.367`.

[20] Malindi, Phumzile. QoS in telemedicine. In *Telemedicine Techniques and Applications*, 2011. DOI: `10.5772/20240`.

[21] Maude. Maude overview. `http://maude.cs.illinois.edu/w/index.php/Maude_Overview`. Accessed: 2020-09-28.

[22] Microsoft. What are consistency levels in Azure Cosmos DB? `https://docs.microsoft.com/en-gb/azure/cosmos-db/consistency-levels`, 2020. Accessed: 2020-09-25.

[23] Newcombe, Chris, Rath, Tim, Zhang, Fan, Munteanu, Bogdan, Brooker, Marc, and Deardeuff, Michael. Use of formal methods at Amazon Web Services. `https://lamport.azurewebsites.net/tla/formal-methods-amazon.pdf`, 2014. Accessed: 2020-09-25.

[24] Poli, John. Compute performance — Distance of data as a measure of latency. `https://formulusblack.com/blog/compute-performance-distance-of-data-as-a-measure-of-latency/`. Accessed: 2021-03-15.

[25] Saini, Anjali and Yadav, P.K. Distributed system and its role in healthcare system. *International Journal of Computer Science and Mobile Computing*, 4:302–308, 2015.

Acta Cybernetica **25** (2021) 307–318.

Evaluating the Performance of a Novel JWT Revocation Strategy[*]

László Viktor Jánoky[ab], Péter Ekler[ac], and János Levendovszky[de]

Abstract

JSON Web Tokens (JWT) provide a scalable, distributed way of user access control for modern web-based systems. The main advantage of the scheme is that the tokens are valid by themselves - through the use of digital signing - also imply its greatest weakness. Once issued, there is no trivial way to revoke a JWT token. In our work, we present a novel approach for this revocation problem, overcoming some of the problems of currently used solutions. To compare our solution to the established solutions, we also introduce the mathematical framework of comparison, which we ultimately test using real-world measurements.

Keywords: JWT, JSON Web Tokens, user access control

1 Introduction

In the field of web application security, JSON Web Tokens play an increasingly significant role. They are very well suited for application in distributed systems, as their validation can be done by the consuming service, without the need for central access to a trusted source. This property, however, also means that there is no easy way to revoke a token once it has been issued.

This paper overviews the current revocation strategies and introduces a mathematical framework for comparison to help system designers find the optimal solution. The mathematical framework is then validated with measurements on a real

[*]The research reported in this paper was supported by the BME Artificial Intelligence TKP2020 IE grant of NKFIH Hungary (BME IE-MI-SC TKP2020). Project no. FIEK_16-1-2016-0007 has been implemented with the support provided from the National Research, Development and Innovation Fund of Hungary, financed under the Centre for Higher Education and Industrial Cooperation – Research infrastructure development (FIEK_16) funding scheme.

[a]Department of Automation and Applied Informatics, Budapest University of Technology and Economics, Hungary

[b]E-mail: `janoky.laszlo@aut.bme.hu`, ORCID: `0000-0002-9070-9026`

[c]E-mail: `peter.ekler@aut.bme.hu`, ORCID: `0000-0002-2396-3606`

[d]Department of Networked Systems and Services, Budapest University of Technology and Economics, Hungary

[e]E-mail: `levendov@hit.bme.hu`, ORCID: `0000-0003-1406-442X`

DOI: `10.14232/actacyb.289455`

application. We also further elaborate on our novel solution, first introduced in [7]. After its brief introduction, our method is compared with the other strategies using the common mathematical framework.

The paper is structured as follows: Section 1 provides an overview of the currently used revocation schemes and their main characteristics. In Section 2, we present a detailed description of our new approach. Section 3 deals with the formal description of the performance characteristics of the different strategies, including our solution. In Section 4, we verify our cost model by measuring different revocation schemes in a real application. Finally, Section 5 wraps the discussion by providing an overview of the work done.

1.1 Literature review

The main source of literature regarding JSON Web Tokens is the *Request for Comments* (RFC) documents of the Internet Society (ISOC). For example, RFC 7519 [9] describes the basics of JSON Web Tokens (JWT) as "URL safe means of representing claims to be transferred between two parties".

While this definition is correct, JWTs are increasingly used in web applications as part of different authentication and authorization schemes, such as bearer tokens [6] in the OAuth2 framework [5] [10] or OpenID Connect [13]. The introspection of these tokens are described in RFC 7662 [12], which briefly touches the question of revocation without providing details on its implementation.

As the problem is more of a practical than theoretical kind, current industry approaches for JWT revocation can be found in technical documentation of different authentication solutions [11] [2], or technical blog posts such as [4] instead of more traditional scientific papers.

1.2 Overview of JSON Web Token Revocation methods

A JWT used to determine access for a protected resource is called an access token in these schemes. The token is usually digitally signed or otherwise cryptographically secured [8]. In both cases, we simply refer to the signing key or the public key as a secret.

In most scenarios, the access tokens are issued along with a second, more traditional, server-side token called a refresh token. This second token makes it possible for the client to acquire a new access token in the future.

When a client logs out from the system, the refresh token is destroyed, and existing JWT tokens are revoked. This revocation is not a trivial task, as the validity of a JWT is determined by the cryptographic assurance, which cannot be easily revoked.

Short-lived tokens: Each generated JWT token has a finite, usually very short lifespan. In this scheme, a token is never directly revoked, but the means of acquiring new tokens are made unavailable (i.e. the refresh token is destroyed). Hence when the short lifespan runs out, no further access is possible to the system.

Blacklist: In the case of a blacklist, revoked access tokens are placed in a shared location (typically a database), where each consuming service can check for invalidated tokens. The big downside of this approach is that it requires data access for each request served - even for ones with valid tokens; thus, the token's validity can no longer be determined in itself.

Secret change: A rarely used solution for invalidation is the changing of the cryptographic secret used to issue and check the validity of tokens. Changing this secret leads to all tokens being revoked, but still logged in users can apply for new ones using their refresh token.

2 The novel revocation algorithm

Our novel revocation strategy is based on the extension of the third option, which is based on changing the secret. In this section, we give a quick overview of this approach.

2.1 Basic principle

When a JWT secret is changed, all the tokens issued with it become invalid. If a user logs out, every other user in the system must request a new token. In practice, this method scales very poorly with the increased number of clients as the number of revocation events, and thus new token acquisitions increase exponentially.

The basic idea of our method is to control the number of revocation events by arranging the users in groups (for example, by hashing their usernames) and assigning a different secret for each group. If a token is revoked in a group, only tokens signed with the group secret will be revoked, instead of all the tokens. As logouts are typically infrequent events, one can use statistical methods as described in [7] to calculate an optimal group size, which minimizes the number of unnecessary revocations while maintaining a manageable number of secrets.

Being able to control the group sizes, we can optimize the performance of the strategy for our specific system. Using a larger number of client groups means less token revocations (network and computation overhead) in exchange for more memory usage.

A typical system using our strategy consists of 3 types of components; a *secured service* providing services to *clients* and an *auth server* storing authentication information and handling token generation. The process described in the following is demonstrated in more detail in Figure 1.

1. Initial token acquisition is very similar to other methods; the client authenticates themselves e.g., by giving a username/password combination.

2. After successful authentication, they are provided with an access and a refresh token. The access token is a JWT, used to consume the secured service, while the refresh token is used to acquire new access tokens.

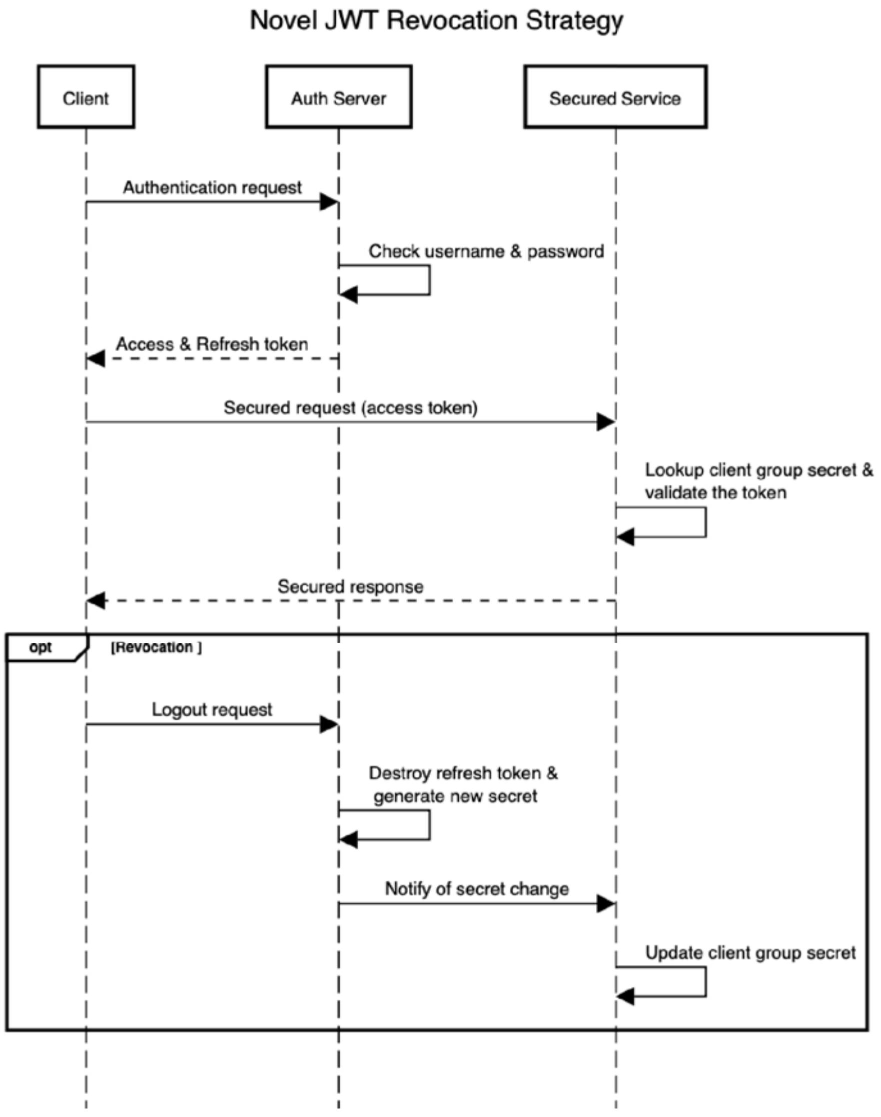

Novel JWT Revocation Strategy
Client
Auth Server
Secured Service
Authentication request
Check username & password
Access & Refresh token
Secured request (access token)
Lookup client group secret & validate the token
Secured response
opt
[Revocation]
Logout request
Destroy refresh token & generate new secret
Notify of secret change
Update client group secret

Figure 1: Detailed call structure of our novel strategy

3. Clients use the access token to consume the secured resources. Upon such a request, the secured service looks up the secret used for the given client group and validates the token against it.

4. When a client wants to log out, they signal the auth server, which destroys the refresh token, changes the group secret, and notifies the connected services.

5. Upon receiving an access token signed with the older secret, the secured service refuses the request. If the initiating client still has a valid refresh token, they can acquire a new access token with the new secret and continue using the system.

2.2 Propagating secret change events

With our proposed method, revocation is instantaneous, and the basic premise of JWT tokens remain intact, that the tokens are valid by themselves.

This solution requires the existence of a channel for propagating the information about the change of the JWT secret. The channel must be available between the token issuer and each service consuming the tokens.

For cases where such a channel is unavailable, the secret change events can be propagated by the tokens themselves, albeit with lower security guarantees. In this approach, the JWT Secret is generated as a rolling code by a pseudo-random number generator [3], each service keeping track of the currently active value. When a token is revoked and the group's secret is changed, the new tokens are issued with the new secret. When a service, still using the old code, receives a token signed with the new secret (the next value from the rolling code), it will also update the secret accordingly.

This method provides eventual consistency for the system in the long run, without the need for a dedicated channel for JWT secret change event prorogation. As a trade-off, the instantaneous revocation is lost, a token is only revoked after the code is rolled (another token, using the new secret is received).

3 Performance evaluation framework

To predict and compare the performance of different strategies, a common mathematical framework must be set. In this section, we will describe the basic operations associated with the handling of tokens, and then determine the cost function of each revocation strategy based on these costs.

3.1 Basic operations with tokens

Regarding the token operations, we determined a set of basic operations, which make up each approach. The costs of these operations can be parametrized for the actual system, which can be based on measurements or estimations. The following main costs were identified:

- C_i **(Issue cost):** the cost of issuing a new token.
- C_v **(Validation cost):** the cost of checking the validity of a token.
- C_c **(Communication cost):** the cost of system modules communicating with each other.

- C_d **(Data access cost):** the cost of accessing data stored in a persistent storage.

In order to predict the performance of a strategy in a system, it is not enough to know the cost of these atomic operations; one must also calculate how many times they will occur. The main factor in determining the total performance cost a system must endure is determined by the clients consuming its service. By knowing their numbers and characterizing the client sessions, predictions can be made about their impact on the system [1]. In order to calculate the former, the following properties must be known about the clients. These can also be acquired by measurements or estimations based on known properties.

- N**:** the number of clients in the system.
- $f_i(t)$**:** probability distribution of client session lengths.
- r**:** average number of protected resource access / client / second.

For strategies based on changing the secret, the average time between token revocations - and hence secret changes - can be calculated for a group of clients based on these metrics. It is basically the expected value of the time until the first token revocation in the group, and denoted as T_{rvk}.

Knowing both the cost and occurrences of typical operations in the system, a cost function can be constructed which describes the average cost of a revocation strategy.

3.2 Cost functions of the different strategies

Short-lived tokens

The short-lived approach has one parameter, T_{life}, which denotes the lifetime of a token. As for maximizing the performance of this approach, this T_{life} should be chosen as the longest tolerable time for token revocation after logout. The longer it is set, the less secure the system is.

The overall cost function consists of two parts, the cost of validating the tokens of the incoming requests and the cost of issuing new tokens to replace the expiring ones.

$$C = (N * r * C_v) + (N * \frac{1}{T_{life}} * C_i)$$

Blacklist

In the case of a blacklist, the main cost factor comes from the necessity to check whether a token is on the blacklist for each request, which makes this approach the worst in terms of scaling when increasing the number of requests.

After accounting for this factor, there are no other costs associated with this method. Therefore the cost function can be defined as the following.

$$C = N * r * C_v * C_d$$

Secret change

In case of a secret change, the baseload of authorizing incoming request is still present, but it is accompanied by the load of new token generation for each client in case of each revocation.

$$C = (N * r * C_v) + (N * \frac{1}{T_{rvk}} * C_i)$$

Notice that the formula is very similar to the short-lived cost function. This is not a coincidence; in both cases, the number of token revocations depends heavily on the average lifespan of a token. In the first case, it is purely determined by the age of the token itself, while in the second case, every client logout event triggers it.

Our novel strategy

As previously stated, our method works by dividing the clients into different groups, let K denote the number of these groups. As our method builds on the secret change event, the cost function is similar too, but because of the separation separation, the T_{rvk} calculated for the whole client population size must be recalculated to the number of $\frac{N}{K}$ clients. This value is denoted by T'_{rvk}. As K increases, T'_{rvk} monotonously increasingly approaches the mean value of $f_i(t)$.

$$C = (N * r * C_v) + (K * \frac{1}{T'_{rvk}})(\frac{N}{K} * C_i + C_c)$$

4 Cost model measurement and validation

To validate our model, a test system was implemented, and different measurements were run. This section describes the measurement setup and the results achieved.

4.1 Setup

The test setup is a set of applications written to simulate the working of a real system. The following components were implemented.

- An **Auth server**, which implements the different revocation schemes, provides tokens and stores users.
- **Foo service**, which is an example of a service providing protected resources.

- **Bar service**, which is another example of a protected service.
- **Test client**, which emulates the behavior of one or more clients. This component gathers and aggregates the data during the measurement.

The source code can be found at the following link: `https://github.com/jlaci/JWT-Revocation-Test`. The components were written in Java 11 (Zulu Open JDK 11.0.5), using the Spring Framework (managed by Spring Boot 2.3.1), the backing database was a MySQL 8. The hardware running the test was a 2019 MacBook Pro 16"(Intel Core i9-9880H, 16GB DDR4 at 2667 MHz, 1TB SSD).

The system can be run with different profiles, each corresponding to the various revocation schemes. The length of the measurement, the number of clients, and their characteristics can be configured, while cost values are based on the actual system implementation.

These system costs were measured before running the simulation by executing the associated operation 500 times and averaging the results. After that, each revocation scheme was measured with the same characteristics, and the results were aggregated.

4.2 Results

First, the actual values are determined for each cost. Table 1 lists the measured values of the different cost model terms.

Table 1: Measured values of each costs

Cost	Value(ms)
C_i (Issue cost)	28.13
C_v (Validation cost)	0.17
C_d (Communication cost)	0.51
C_c (Data access cost)	4.276

Comparison of the different schemes in a typical system

The first simulation was set up to represent a typical system employing the different JWT revocation schemes. This run aims to validate the outlines of our mathematical model and give an overview of how each strategy would fare in a close-to-real application. The simulation parameter values were chosen based on the author's experience with typical systems to represent a realistic application without favoring any strategy.

This simulation was run for 300 seconds with 100 clients. Client session characteristics were chosen to follow a uniform distribution with a mean value of 60 seconds and a maximum of 120 seconds. With 100 clients, this means T_{rvk} was calculated as 1.2 seconds. Each client, on average, executed 0.5 authenticated requests per second. Our novel algorithm used 20 client groups ($K = 20$).

Table 2 show the raw numerical results for the predicted and actual costs of each revocation strategy given the same system parameters.

Table 2: Predicted and measured cost

Strategy	Predicted	Measured
Short-Lived	0.10	0.22
Blacklist	0.22	0.41
Secret Change	2.35	3.17
Novel	0.13	0.45

Figure 2 visualizes the predicted and measured values for better understanding. As we can see from the results, the actual, measured costs are always higher than our predicted ones. This difference can be attributed to factors that were not modeled in our framework, e.g., technical overheads such as serialization and deserialization, logging, and generic costs of serving requests in a thread-pool based model. These costs are not primarily associated with the JWT revocation strategy itself; including them would make the model impractically complex.

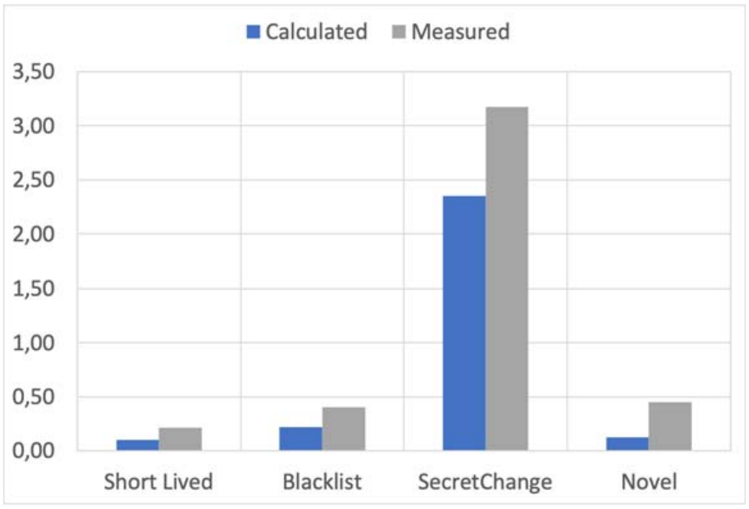


Figure 2: The comparison of results

Our expectations regarding the scale and relative performance of each strategy were mostly met. The largest error between measurements and prediction occurred with our novel approach, most likely due to implementation constraints, such as the library we used for encryption and decryption not being optimized for frequent secret changes. Nevertheless, even with this less accurate measurement, we can see the following general tendencies. The short-lived method provides the best performance while offering the worst security. Blacklist and our novel approach have roughly the same performance for $r = 0.5$ request / client / second with the same level of security, while secret change has the worse performance with slightly worse security.

Comparing blacklist and novel approach for larger load

Based on the previous data, the blacklist and our solution would appear to be the same in terms of performance for relatively low request numbers. This changes, however, when increasing the number of requests per second made by the clients. As seen in Figure 3 with the increased number of requests, our solution scales much better than the blacklist approach while providing the same level of security.

Due to processing power constraints of the test setup, we could not measure beyond this point $r = 2.5$ with a reasonable number of clients. If we set the target r higher than this, the actual measured rate of requests were not increased accordingly, due to resource contention between the simulation threads. Based on the calculations and previous measurements, however, we are confident that the trend would continue, and our solution would prove to be better than any other strategy (blacklist and secret change) with the same security level.

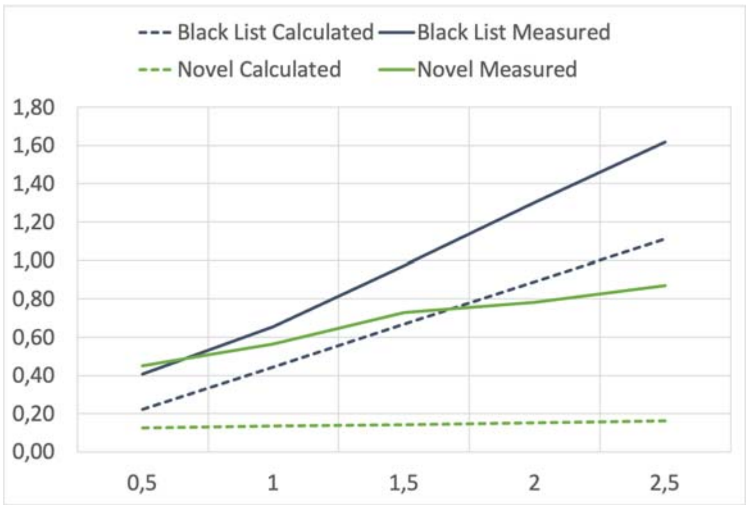


Figure 3: The comparison black list and novel

5 Conclusions

In this paper, we described the main approaches of JWT revocation and introduced our novel solution. We provided a tool-set for choosing the best revocation strategy based on the different characteristics of the system. The basis of this tool-set is the measurement of client population characteristics and costs associated with common operations when dealing with JWTs.

We proved the validity of our cost model by implementing a system using the described methods and measuring its performance for different revocation strategies. The measured results supported that our novel revocation method is competitive with the current solutions while offering advantages in usability, such as instantaneous revocation. We showed that our solution scales better for an increased number of requests than any other strategy with the same security guarantees (instantaneous revocation).

With the proven mathematical framework at hand, one can find the optimal revocation strategy for any system by choosing the minimal cost function. In cases where the function has additional variable parameters, traditional approaches like linear search can be used to find the optimal solutions.

Ultimately, we hope that our work will aid the capacity planning and system design of distributed systems, as the JWT based authentication schemes have the highest potential in this area. We hope that our novel solution will help with increasing the performance and efficiency of these systems, therefore conserving energy and costs associated with providing a large-scale distributed service.

References

[1] Arlitt, Martin. Characterizing web user sessions. *ACM SIGMETRICS Performance Evaluation Review*, 28(2):50–63, 2000. DOI: `10.1145/362883.362920`.

[2] Auth0 Inc. Revoke tokens. `https://auth0.com/docs/`.

[3] Blum, Manuel and Micali, Silvio. How to generate cryptographically strong sequences of pseudorandom bits. *SIAM journal on Computing*, 13(4):850–864, 1984. DOI: `10.1137/0213053`.

[4] dWTV. Learn how to revoke JSON Web Tokens, July 2017. `https://developer.ibm.com/tv/learn-how-to-revoke-json-web-tokens/`.

[5] Hardt, Dick. The OAuth 2.0 authorization framework. Technical report, RFC 6749, October, 2012.

[6] Hardt, Dick and Jones, Michael. The OAuth 2.0 authorization framework: Bearer token usage. `https://tools.ietf.org/html/rfc6750`.

[7] Jánoky, László Viktor, Levendovszky, János, and Ekler, Péter. An analysis on the revoking mechanisms for JSON Web Tokens. *International*

Journal of Distributed Sensor Networks, 14(9), September 2018. DOI: `10.1177/1550147718801535`.

[8] Jones, M., Bradley, J., and Sakimura, N. JSON Web Signature (JWS). RFC 7515, RFC Editor, May 2015. `http://www.rfc-editor.org/rfc/rfc7515.txt`.

[9] Jones, M., Bradley, J., and Sakimura, N. JSON Web Token (JWT). RFC 7519, RFC Editor, May 2015. `http://www.rfc-editor.org/rfc/rfc7519.txt`.

[10] Okta Inc. Self-encoded access tokens. `https://www.oauth.com/oauth2-servers/access-tokens/self-encoded-access-tokens/`.

[11] Pontarelli, Brian. Revoking JWTs & JWT expiration. `https://fusionauth.io/learn/expert-advice/tokens/revoking-jwts/`.

[12] Richer, Justin. OAuth 2.0 token introspection. `https://tools.ietf.org/html/rfc7662`.

[13] Sakimura, Natsuhiko, Bradley, John, Jones, Mike, De Medeiros, Breno, and Mortimore, Chuck. OpenID connect core 1.0. *The OpenID Foundation*, page S3, 2014.

Acta Cybernetica **25** (2021) 319–349.

Lamred: Location-Aware and Privacy Preserving Multi-Layer Resource Discovery for IoT[*]

Mohammed B. M. Kamel[abcd], Peter Ligeti[ae], and Christoph Reich[bf]

Abstract

The resources in the Internet of Things (IoT) network are geographically distributed among different parts of the network. Considering huge number of IoT resources, the task of discovering them is challenging. While registering them in a centralized server such as a cloud data center is one possible solution, but due to billions of IoT resources and their limited computation power, the centralized approach leads to some efficiency and security issues. In this paper we proposed a location-aware and privacy preserving multi-layer model of resource discovery (Lamred) in IoT. It allows a resource to be registered publicly or privately, and to be discovered with different locality levels in a decentralized scheme in the IoT network. Lamred is based on structured peer-to-peer (P2P) scheme and follows the general system trend of fog/edge computing. Our model proposes Region-based Distributed Hash Table (RDHT) to create a P2P scheme of communication among fog nodes. The resources are registered in Lamred based on their locations which results in a low added overhead in the registration and discovery processes. Lamred generates a single overlay and it can be generated without specific organizing entity or location based devices. Lamred guarantees some important security properties and it showed a lower latency comparing to the centralized and decentralized resource discovery models.

Keywords: resource discovery, DHT, IoT

[*]This research has been partially supported by Application Domain Specific Highly Reliable IT Solutions project which has been implemented with the support provided from the National Research, Development and Innovation Fund of Hungary, financed under the Thematic Excellence Programme TKP2020-NKA-06 (National Challenges Subprogramme) funding scheme, by ÚNKP-20-3 New National Excellence Program of the Ministry for Innovation and Technology from the source of National Research, Development and Innovation Fund, by SH program and by the Ministry of Science, Research and the Arts Baden-Württemberg Germany

[a]Eotvos Lorand University, Budapest, Hungary
[b]Hochschule Furtwangen University, Furtwangen, Germany
[c]University of Kufa, Najaf, Iraq
[d]E-mail: `mkamel@inf.elte.hu`, `mkamel@hs-furtwangen.de`, ORCID: 0000-0003-1619-2927
[e]E-mail: `turul@cs.elte.hu`, ORCID: 0000-0002-3998-0515
[f]E-mail: `christoph.reich@hs-furtwangen.de`, ORCID: 0000-0001-9831-2181

DOI: 10.14232/actacyb.289938

1 Introduction

The Internet of Things (IoT) network consists of billions of resources distributed in different parts of the network. The huge number of resources and their different levels of accessibility (e.g. private resources, local resources and public resources) make the task of registering and discovering them a challenging task. Adopting a centralized scheme such as relying on a cloud service helps organizing the resources in an entity that has a high computation capability and can be used to discover those registered resources. But, in systems that rely only on a centralized entity a significant amount of traffic has to be used for the registration and discovery processes which might affect the overall efficiency of the system. Comparing to cloud computing infrastructure that send the traffic to a centralized cloud data center, the fog/edge nodes in the fog and edge computing infrastructures try to distribute the data among nodes and keep it as close as possible to the origin source of data. Hence, fog computing extends the cloud computing to the edge of the network, close to the point of origin of the data [3]. Processing the data locally during the registration and discovery of resources helps to achieve scalability, at the same time mitigates the potential privacy and security risks against single point of attack and failure. However, there should be a unique decentralized scheme that defines and arranges the relationship between the fog/edge nodes and their responsibilities.

Distributed Hash Table (DHT) creates an overlay by assigning a seemingly unique identifiers to the participating nodes. The generated overlay can be used to organize the distributed nodes in the decentralized resource registration and discovery processes [15]. The identifiers in DHT are generated by feeding some of the parameters of the peer nodes (e.g. IP addresses) to a hash function, and the output is used as the identifiers of the nodes. Depending on the identifier, each node is resided in a specific location in the overlay with a predefined responsibilities. Due to the random-looking behaviour of the hash functions, the output of the relatively close parameters in the input range might not be close in the hash space. While this property is required to ensure the random and uniform distribution of nodes and the stored data in the overlay, but adopting the original DHT technique in fog and edge computing infrastructures might results that two adjacent nodes reside in two far locations in the overlay. As a result, while adopting DHT in resource discovery [15] removes the centralized entity, but might map the geographically close nodes to distant nodes in the resulted space. If the nodes in the resource discovery models are distributed without considering their physical locations, an efficiency issue might be raised. This is due to the reason that the logical path of nodes on the underlying network could vary from the logical based path in the overlay network that is organizing the distributed nodes. Thus the lookup latency can be high, which in this case leads to operational inefficiency in applications running over it [18]. During organizing nodes in the resource discovery model, the locations of nodes have to be taken into consideration. Afterward, a resource is registered based on its location in a close node in the distributed system which reduces the required time to register and reach that specific node.

Therefore, while adopting DHT as a structured Peer-to-Peer (P2P) scheme to organizing fog and edge nodes in IoT has some advantages such as scalability and functionality without involving any centralized entity, but DHT might cause the data to be stored in a far node. In this paper we proposed a location-aware and privacy preserving multi-layer model of resource discovery (Lamred) in IoT. Lamred aims to keep the data as close as possible to the origin of the data by taking into consideration the locations of both resources and IoT gateways and utilizing a single DHT overlay. It can be implemented without specific location based devices, and add no extra local overhead comparing to traditional DHT overlays. Here are the main contributions of this paper:

- Propose Lamred, a new DHT based model as a P2P overlay for resource discovery in the IoT Network.
- Propose a Region based Distributed Hash Table (RDHT) for location aware resource registration and discovery. Lamred keeps the resources as close as possible to the clients, hence reducing the required time during the registration and discovery processes.
- Propose a private tag generation method in Lamred for private resource registration and discovery.
- Use cryptographic primitives to protect the private resources in the system and ensure the required anonymity and privacy in Lamred.

The rest of this paper is organized as follows. The next section defines some of the preliminaries. Section 3 summarizes the efforts in current research field of resource discovery. Section 4 describes Lamred, the proposed model of resource discovery, and introduces its different components. In Section 5 we evaluate the model, proof the required security properties and discuss the performance of Lamred. Finally, we conclude our work in Section 6.

2 Preliminaries

2.1 Cryptographic Primitives

Definition 1 (collision-resistant one-way hash function)**.** *A function H(.) that maps an arbitrary length input M into a fixed-length digest d is called collision-resistant one-way hash function it satisfies the following properties:*

- *Given M, it is easy to compute $H(M)$.*
- *Given d, it is hard to find any M s.t. $d = H(M)$.*
- *Given $d = H(M)$ and M, it is hard to find M' s.t. $M' \neq M$ and $H(M) = H(M')$.*

- *It is hard to find two distinct messages M' and M'' s.t. $M' \neq M''$ and $H(M') = H(M'')$.*

If the solution can be computed in the polynomial time, therefore it is considered *easy* to compute. On the other hand, if there is no solution known to solve the problem in polynomial time, it is considered a *hard* problem [7].

Definition 2 (Probabilistic Polynomial Time)**.** *An algorithm $\mathcal{A}$ is a PPT (Probabilistic Polynomial Time) if its probabilistic and $\exists c \in \mathbb{N}$ such that $\forall x, \mathcal{A}(x)$ halts in $|x|^c$ steps.*

2.2 Distributed Hash Table

DHT is a distributed system that creates a structured P2P overlay in a network. The participating nodes in the network can join and leave the DHT at any specific time. Upon joining a new node, a new identifier is assigned to it and depending on the assigned identifier, it will be responsible of storing a set of data in the network. Using multiple replicas helps the DHT to be fault tolerant and improves the availability of data in the network. Using identifiers instead of other types of addressing (e.g. IPs) helps to balance and manage the data storage among participating nodes without any centralized entity. In addition to load balancing, it solves the scalability by providing the service of generating the identifiers by the participating nodes themselves. There are several protocols to implement DHT such as Chord [29], Kademila [20], Pastry [28], and Tapestry [33].

DHT uses a large address space of integer numbers. The size of the address space depends on the fixed output size of the function that is used to generate the identifiers. The size of the key space is he same as the address space, i.e. the same function is used to generate identifiers for nodes and keys for the stored data. To achieve the random function of identifier generation and uniform distribution of data among all participating nodes a collision-resistant one-way hash function (definition 1) is used in DHT.

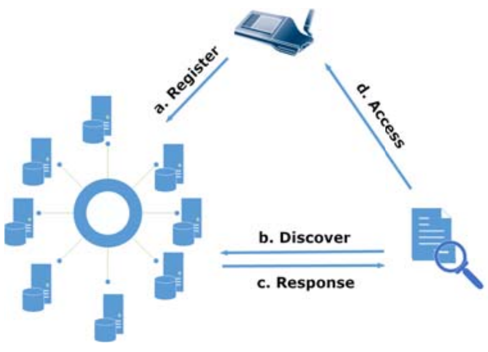
Similar to hash tables [19], the data in DHT is stored in key/value pairs. The value parameter includes any stored information about the data (e.g. the address of the data) and can be retrieved from the DHT based on its associated key. The key parameter of the key/value pair is generated by feeding specific information (e.g. the attribute of the stored data such as its name, its type, etc.) to the collision-resistant one-way hash function which produces a uniformly distributed randomized hash value. The generated hash value which represents the key parameter in the key/value pair is used to determine the responsible node in the network of storing this specific pair. To achieve the distributed indexing, DHT defines a specific portion in the key space that each particular node is responsible for. DHT has two implementation interfaces for storing and lookup: *Put* and *Get*. The *Put* interface takes the key/value pair and stores this pair in the DHT. The *Get* interface takes a single parameter key and lookup in the DHT to retrieve the identifier of a node that is responsible to store the corresponding value to the given key. In the DHT the store (i.e. put interface) and lookup (i.e. get interface) operations are guaranteed to be done with an upper bound of $O(log(n))$, in which n is the number of nodes in the DHT. This feature guarantees that any participating node in DHT can store a pair of key/value or lookup based on a given key by routing through of maximum $log(n)$ nodes.

2.3 Resource Discovery

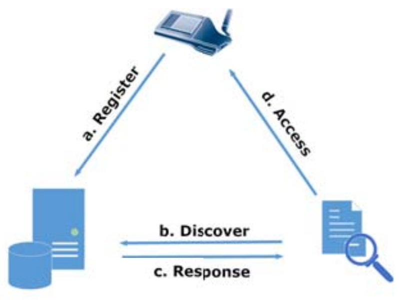
Resources in IoT can be IoT data or IoT devices. These resources are registered in the network and can be discovered by the clients. The process of discovery is to get the access address (e.g. URI or IP and port addresses) of IoT data, IoT devices or a combination of them as a result of discovering (i.e. querying) the resources in the network. The search techniques can be functional (event-based, location-based, time-related, content-based, spatio temporal-based, context-based, real-time and user interactive searching) or implementational (text-based, metadata-based or ontology-based approach) [25].

The resources can be registered in different parts of the network distributed among many nodes (Figure 1a) or in a centralised trusted entity (Figure 1b). The resource discovery [9] is a mechanism to return the access address of a resource based on the information provided during the lookup operation. The resource access address can be its IP and port addresses, its URI or other metadata and further links about the resource. The discovery process starts by issuing a query including the attributes of the required resources to be discovered. An attribute of a resource can be any information that describes it, such as its location, its type, etc. The query is issued by a client and sent to the discovery system. The query that is received by the discovery system is then processed and divided into sub-queries. As instance, the query can be divided based on the attributes of the required resources, and a sub-query is issued in the system for each attribute. The discovery system then finds and communicates with the responsible nodes to get the required information about the resources. After getting the list of resources, they are ranked based on some scoring methods and the final result is sent back to the requested client.

The data that are generated by the discovered resources in the system can be collected in either *request/response* or *publish/subscribe* patterns. In request/response, the data from the discovered resource is returned back to the clients based on their requests. As instance, Constrained Application Protocol (CoAP) [4] is a document transfer protocol that works based on a request/response approach on a client-server architecture. In the publish/subscribe, the discovered resource publishes its data to the clients that are already subscribed. The process can be done through a publish/subscribe server that is the middleware between the subscribed clients and the published resources. MQ Telemetry Transport (MQTT) [11] is a protocol that is based on publish/subscribe approach and facilitates one to many communications through a common node (i.e. broker). Resources publish the messages by sending them to the broker and on the other hand clients subscribe for a specific message in the broker.



(a) Decentralised Scheme



(b) Centralized Scheme

Figure 1: Discovery and Access Mechanism

3 Related Work

Some researchers adopt the use of a centralised entity as part of their proposed models that manages some parts or all parts of the system. Cheshire and Krochmal [5] proposed a Domain Name System (DNS) based discovery for the IoT network. It defines a model on how the users register their resources and discover the resources based on the DNS protocol. The proposed model does not modify the underlying DNS protocol messages and codes and as a result is simple to implement. In this model, a centralized authority stores the registered resources and there is no additional security consideration to the original DNS protocol itself. Authors in [12] have proposed a large scale resource discovery to discover the devices and sensors in the IoT network by building a scalable architecture called Digcovery. The framework enables the users to register their resources into a shared infrastructure and to access/discover accessible resources by a mobile phone. Their proposed work is focuses on the discoverability of devices based on context-awareness and geo-location. Digcovery allows high scalability for the discovery based on a flexible architecture. However, it relies on a centralized point called *digcoverycore* for management and discovery.

Datta and Bonnet [8] proposed a resource discovery framework for IoT. The proposed framework includes a centralized registry that registers and indexes the attributes of resources. The attributes of the resources are used as the parameters during the discovery process through the search engine, that returns the access addresses of the discovered resources. The authors in [13] proposed a discovery model for IoT that performs the discovery based on various constraint parameters such as input/output (IO), precondition/effect and quality of experience (QoE). In the proposed model, a centralized directory server is used to register and discover the services in the IoT network. The discovery is done using semantic service description method OWL-S^{iot} that describes both the IoT services and discovery requests. Using the centralized scheme helps organizing the resources in an entity that has a high computation capability, however, this centralized entity might turn into a single point of failure, which, if fails, the overall system will stop. This profoundly affect the availability and reliability of the system. Additionally, the centralized entity could turn into a bottleneck for the system affecting the overall system performance.

Several researches utilized the P2P scheme in the IoT network as a method for distributed resource discovery. The model in [15] removes the centralized entity by managing the fog nodes in a P2P scheme. It divides the resources into public resources that are discoverable by all clients and private resources that can be discovered by a subset of clients. In addition it provides other features such as multi-attribute discovery. However, the main drawback of this model is that by not considering the physical location of the resources and fog nodes it fails to keep the registration process low by using only the fog nodes that are in the same region of the registered resource. A particular emphasis on the links and nodes locality presents a Mesh-DHT in [30] that implemented in IEEE 802.11 wireless mesh network (WMN). The authors employed the Mesh-DHT for building a scalable DHT in WMNs. This approach enables an entirely distributed organization of information by building a stable, location-knowledgeable overly network. Because nodes primarily talk to physically nearby nodes, it allows minimizing the overhead in DHT communication of WMNs, therefore requiring fewer transmissions. However, the model can not reflect the locality of mesh routers in the overlay construction and therefore does not able to represent the locality of keys. Wirtz et al. [31] have been proposed an improved version of the DHT based service registration and discovery. Their work is based on their previous model (Mesh-DHT) and focuses to address its main drawback. Their proposed model partitions the global DHT overlay into different scopes with different degrees of locality that are hierarchically organized in levels. By choosing an appropriate level, a lookup can be restricted to only consider the locally available information. The put(key, value) and get(key) in DHT are extended to put(key, value, level) and get(key, level), respectively.

The authors in [21] proposed a single-gateway based hierarchical DHT solution (SG-HDHT) for an efficient resource discovery in Grids (i.e. Virtual Organizations (VO)). The model forms a tree of structured overlay and consists of a two-level hierarchical overlay network. It defines a global DHT and number of second level DHTs, a DHT overlay for each VO. Only one peer (called a gateway or super peer) in a DHT overlay of a VO is attached to the global level DHT of the hierarchy. The proposed resource discovery in this model deals with two different classes of peers: super peers and simple nodes. The lookup is directed to the super peer of the VO and then through the global DHT to the superpeer of the requested resource.

The authors in [1] proposed a wireless communication and computation framework that sustains the scalability for a massive increase of IoT devices. The researchers adopt the fog computing paradigm and therefore their proposed model enlarges the cloud-based solution by providing computing services close to the source of data generation. WMN nodes are used as the fog nodes in the proposed model. The authors have employed Chord [29], to generate a DHT based P2P overlay of fog nodes for resource discovery. This proposed model specifically targets the underutilized processing power of devices for computing purposes. The discovery in this model is done by involving the fog nodes as brokers for the discovery of the required resources. Pahl and Liebald [23] introduced a distributed modular directory of service properties and a query federation mechanism based on virtual state layer (VSL) [24] that allows mapping complex semantic queries on the simple search. The presented modularaization adds little latency which makes it suitable for time-critical operations. The proposed model supports multi attribute discovery and allows adding new attributes in the system at runtime that fits the nature of the dynamically varying IoT.

Authors in [6] proposed an architecture consists of two discovery levels, local and global service discovery. It uses the P2P scheme for resource discovery, and IoT gateways are the peers in the P2P overlay. This architecture uses two layers: the Distributed Location Service (DLS) and the Distributed Geographic Table (DGT) [26]. The DLS is a DHT based architecture that works as a name resolution service by providing any required information to access any resource in the network, depending on its URL. The DGT builds a layer to distribute the information depending on the location of nodes, which can be used to discover the resources based on their geographic location information. The model successfully manages the registration and discovery based on the locations of the resources. Although DGT keeps the location data of the IoT gateways, but since DGT and DLS are loosely coupled then in order to discover the resources in a given location the system has to retrieve the IoT gateways data from DGT overlay and then lookup the DLS overlay for the required resources. In addition to keep the data as close as possible to the registered resources, Lamred aims to utilize a single DHT overlay and add no extra local overhead and low global overhead comparing to traditional DHT overlays. Furthermore, it aims to allow the participating nodes to join Lamred without using specific location based devices.

4 Location-Aware and Privacy Preserving Multi-Layer Resource Discovery (Lamred)

The resource discovery in fog/edge computing has some requirements that have to be addressed. Due to the distributed nature of the IoT gateways and the limited computing power of the IoT resources, the resource discovery model has to depend only on low computation processes and does not involve any centralized entity. Lamred allows four levels of discovery: *local discovery* that is limited to a single IoT gateway, *intra-regional discovery* that is limited to a local region (i.e. sub-region of a region set), *regional discovery* that is done in a specific geographical area and *public discovery* (i.e. location independent) that is done among all publicly registered resources, regardless of their locations. In addition, Lamred distinguishes between two types of resources, *public resources* that can be discovered by any client in the system (e.g. a public temperature sensor or a resource offering a public service) and *private resources* that can be discovered by a predefined subset of clients (e.g. private resources in a smart home or a local printer in an organization).

There are three main disjoint sets in Lamred: set of clients ($\mathcal{C}$), set of objects ($\mathcal{O}$) and set of gateways ($\mathcal{W}$). The finite set $\mathcal{C}$ consists of the IoT clients in the network. An object $o \in \mathcal{O}$ is any device in the IoT network with proper computational power that handles a resource u. Subsets of $\mathcal{C}$ and $\mathcal{O}$ are connected to different IoT gateways in $\mathcal{W}$. A gateway w's responsibility may vary from handling a few nodes (e.g. smart home) to hundreds of nodes (e.g. environmental monitoring). The proposed model creates a region-based DHT (RDHT) [17] overlay that provides a structured P2P method of addressing and discovery of the peers. The members of $\mathcal{W}$ (i.e. IoT gateways) represent the peers in the P2P overlay. Let $H(.)$ be a collision resistant one-way hash function with d bits message digest, $Enc_k(m)$ be an encryption of the message m using symmetric key k and $Sign_w(m)$ be a digital signature for message m generated by $w \in \mathcal{W}$ gateway.

4.1 Lamred Properties

The Lamred has been designed to address the requirements for resource registration and discovery in the IoT network. Table 1 shows a comparison of some of the supported properties in the different resource discovery models. In general, Lamred has the following properties:

- **Location Aware:** Lamred utilizes RDHT [17] that creates an overlay of IoT gateways divided logically into multiple region sets and local regions (i.e. sub-regions) in DHT overlay based on their physical locations. It generates a single overlay that can be generated without specific organizing entity or location based devices.
- **Multi-attributes:** Each resource has number of attributes. These attributes can be its location, its type, its provided service and so on. To discover a resource or a set of resources, in addition to their exact identifiers more than one attribute might be needed to get the precise result of the required resources. Lamred supports the multi-attributes discovery and the clients are able to discover the resources based on multiple attributes. In addition to the predefined set of attributes, participants in Lamred are able to create new attributes in real time.
- **Scalability:** The scalability describes the ability of the a decentralized resource discovery to adjust the registration and discovery process as the system grows in term of number of nodes. Lamred distributes the responsibility among many nodes that can continue working efficiently as the number of nodes grows.
- **Management:** Lamred provides defined interfaces for the authorized IoT entities to be able to add, remove, update and discover resources in the network.
- **Discoverability:** Because of the use of RDHT overlay, Distributed Address Table (DAT) [16] can be integrated as a part of Lamred (in a specific region

Table 1: Supported properties in Resource Discovery Models

Features	Decentralized	Location aware overlay	Multi attributes	Security considerations
Jara et. al. [12]	✗	-	✓	✗
Cheshire et. al. [5]	✗	-	✓	✓
Datta and Bonnet [8]	✗	-	✓	✗
Jia et. al. [13]	✗	-	✓	✗
Cirani et. al.[6]	✓	✓	✗	✗
Wirtz et. al. [30]	✓	✓	✗	✗
Wirtz et. al. [31]	✓	✓	✗	✗
Mokadem et. al. [21]	✓	✗	✗	✗
Shabir et. al. [1]	✓	✗	✓	✗
Kamel et. al. [15]	✓	✗	✓	✓
Pahl et. al. [23]	✓	✗	✓	✓
Lamred	✓	✓	✓	✓

in RDHT) to allow discoverability of all resources in the network including as instance those behind the Network Address Translator (NAT).

- **Responsibility Definition:** Each node in RDHT overlay of Lamred is aware of the range of its responsibility to registering a subset of resources. This results that the clients that need to discover a resource in the system being aware of the specific IoT gateway in Lamred that is responsible to store the required information to access that specific resource, and issue a discovery request to that specific node.
- **Discoverability Range:** Lamred uses a private/public architecture and is able to keep some of the resources private and only discoverable by the authorized clients in the IoT network.

4.2 Security model

An object $o \in \mathcal{O}$ that needs to register its private resource in Lamred has a predefined set $\mathcal{F}_o \subset \mathcal{C}$ of friend clients. The members of $\mathcal{F}_o$ are able to discover the privately registered resource of object o. We assume that from the viewpoint of any object $o \in \mathcal{O}$ in Lamred, the set of friends $\mathcal{F}_o$ are *honest* nodes. The rest of the clients $\mathcal{R}_o = \{r \in \mathcal{C} \setminus \mathcal{F}_o\}$ can be assumed to be *malicious.* In the case of the finite set of IoT gateways $\mathcal{W}$ in Lamred, we have to assume that there is no cut containing malicious nodes only in the communication graph composed of the clients, objects and gateways (otherwise, the malicious nodes together could make the communication impossible). More precisely, we assume that for a given resource u of an object o for every friend $f \in \mathcal{F}_o$ there exist a path $(w_1, w_2, \ldots, w_k)$ in the communication graph such that the object o is connected to w_1, the friend client f is connected to w_k and all of $w_1, \ldots, w_k$ are *semi-honest.* The semi-honest entities are assumed to follow the protocol properly, but they might store the received data locally in an attempt to get more information from the stored data.

Beside these properties, the nature of the communication model also regulates the applicable security. In Lamred only the IoT gateways assumed to be able to use public key cryptography, while the objects handling the IoT resources can encrypt and decrypt messages using symmetric keys only because of their limited computation power. In case of IoT gateways, we suppose that every $w \in \mathcal{W}$ can generate a digital signature $Sign_w(p)$ of any transmitted packet p. The proposed construction is supposed to achieve security requirements in the computational sense, i.e. we assume PPT adversaries (definition 2) with negligible success probabilities when attempt to attack the scheme. A function has a negligible success probability if the success occurs with a probability smaller than any polynomial fraction if the size of the input exceeds a given bound [2]. The private resources are registered and discovered by an added private tuple (see Section 4.5). The goal of the security model in the privately registered resources of Lamred is to allow friend clients to securely and anonymously discover the private resources, which comes from the following security properties:

- **Resource anonymity**: Every PPT adversary can learn any connection between a given private tuple and a given private resource with negligible probability only.
- **Resource privacy**: Every PPT adversary can learn the address of a resource from a given private tuple with negligible probability only.
- **Unforgeability**: Every PPT adversary is able to generate, remove or update a valid private tuple of a resource on behalf of a given honest object with negligible probability only.

4.3 Location Regions in Lamred

Lamred consists of maximum 2^g regions with maximum of 2^d IoT gateways in each region. The regions are grouped in $2^{g/2}$ sets, each set with one representative region and $2^{g/2} - 1$ local regions. Consequently, an identifier of a node in Lamred consists of three concatenated parts: region set id, local region id and local node id and is $(g + d)$-bit long. During the creation of identifiers in Lamred, two hash functions are used. The first hash function generates g-bit output digest based on a given information of the region set and the local region, while the second hash function generates d-bit output digest based on the given node information. g and d parameters can have same value and the same hash function can be used to generate the different parts of the identifiers. There are two specific generic regions in Lamred, namely *private* and *public* regions. Each node joins private and public regions regardless of its physical location. In addition to that, a new node joins a local region in the Lamred based on its location. Figure 2 illustrates the regions in RDHT overlay of Lamred.

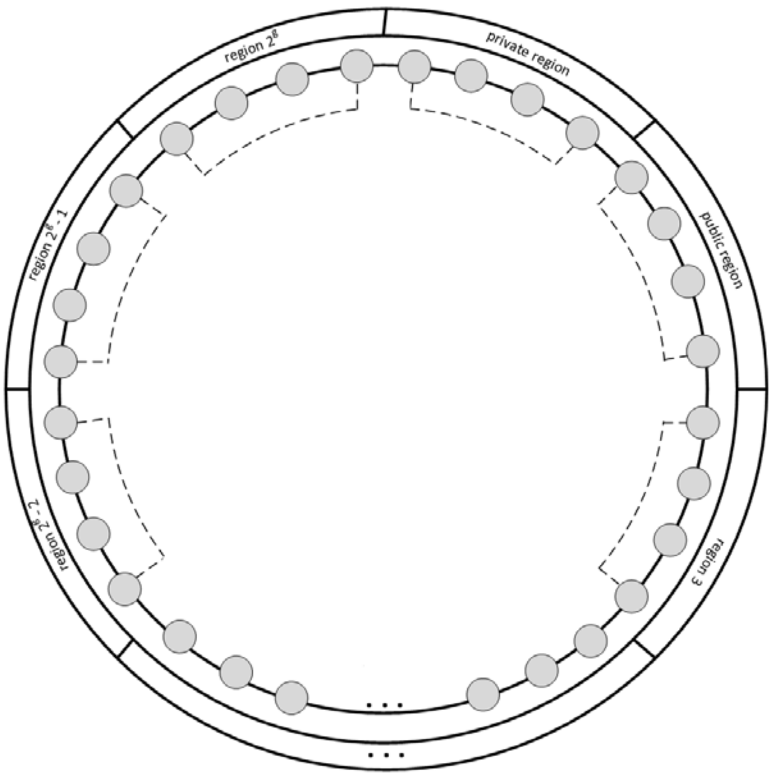


Figure 2: Regions in RDHT

The regions are generated by feeding the information of the locations to the hash function that outputs a g-bit digest. The given input information of the locations can be represented by human readable names of regions or a specific prefix of latitude/longitude data. Each region set has a representative region, in which the nodes in the that region represent other nodes in the region set. To create a region id, the information of the representative region of that region set is fed to the hash function and the first left $g/2$ bits represents the first $g/2$ bits of the generated region id. The local region information is then fed to the hash function and the last $g/2$ bits is taken that will represent the second $g/2$ bits of the generated region id. The representative region itself, will have the last $g/2$ bits all set to zero. As a result, all the regions in each of the $g/2$ region sets in RDHT share the same $g/2$ prefix bits. Because of the Avalanche effect property [32] of the hash function algorithms, each subset of a generated digest by the hash function should be affected equally as any other subset of the digest. Therefore, generating the region id by taking $g/2$ bits from the g bits digest of representative region and $g/2$ bits from the g bits digest of the local region should not affect the randomness of the generated identifier. The remaining d-bit of the identifier of a node is generated by hashing the information of the IoT gateway (e.g. its IP address). Figure 3 shows the generation of the identifier of a node in Lamred.

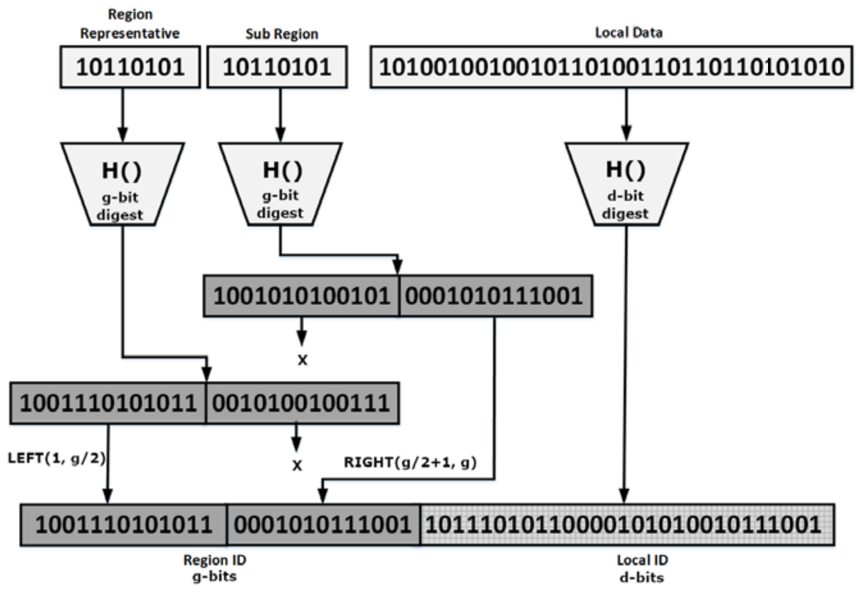


Figure 3: Identifier generation in Lamred

A new IoT gateway $w \in \mathcal{W}_{rg} \subset \mathcal{W}$ joins Lamred by registering in its local region, along with other members of $\mathcal{W}_{rg}$ in the same local region. This is done by hashing the location information of the representative region and its local region to get the first g bits of the identifier of node w and then hashing its unique information (e.g. IP address of w) resulted in the rest d bits of the identifier of node w. In addition to the local region, the newly joined node w can join private and public regions as well. This is done by first hashing its unique information (e.g. IP address of w) to be able to generate two identifiers that are used in private and public regions. The generated identifier in the private region starts with g zeros followed by the d bits output of the hash function used by w. The generated identifier in the public region starts with $g - 1$ zeros followed by a single bit 1 and the d bits output of the hash function used by w. The joining process is done through an *introducer node* that is already a member of Lamred. The joining process of a newly joined node w starts by sending a look up request through the *introducer node* for its own identifier (i.e. the newly generated identifiers of node w) in both private and public regions, as well as in its own local region.

Similar to Kademlia [20] there are two general α and k parameters in Lamred that determine the parallelism and system wide replication, respectively. Each node in Lamred, has d lists of the k-buckets [20] that includes the access addresses to the nodes in the same region. In addition, each node in any region of a region set should keep $g/2$ lists of the k-buckets that includes access addresses to all representative regions in all region sets in RDHT, including the public and private regions. The nodes in the representative region of any subset, keep $g/2$ lists of the k-buckets that includes access addresses to all local regions in the region set. As a result, the nodes in the representative regions have $d + g$ lists and all other nodes in any region in RDHT have $d + g/2$ lists. Each of those lists has maximum of k entries. Considering the size of the DNS domain cache in Raspberry Pi that is defaulting to 10,000 entries[1], storing the access addresses of maximum k IoT gateways in maximum $d + g$ lists does not require any additional storage consideration in the Lamred peers. Figure 4 shows an example of a RDHT overlay of Lamred with 3 region sets, in addition to the public and private regions. In this tiny example the hash function of both region and local identifiers generates a 4-bit digest, therefore, the identifier of each node consists of 8-bits that includes 4-bits region identifier and 4-bits local identifier. As instance, initiator peer with identifier 10001110 wants to get the access address of a resource that is stored in the destination peer 11101111. Since the destination peer is in region id (1110), the representative region that this peer belongs to is (1100). Therefore, the initiator peer sends the request to the node (11001110) in the representative region which is then directed to the specific region and finally to the destination peer in the required region.

4.4 Public Resource Registration and Discovery

A resource u in the network has its specific access address and a set of attributes that describe its properties (e.g. its type, its provided service, etc.). When an object $o \in \mathcal{O}$ wants to register its resource u in the network, it has to add the required information in Lamred through a member of $\mathcal{W}$ that is directly connected to. This set of information includes the tag that is generated by hashing the attribute type of the resource, the value of the added attribute, the ownership information and the access address to resource u as illustrated in Figure 5. After adding the tuple $(tag, value, ownership, data)$ of resource u to Lamred, it can be discovered by all clients in the network. There are two options for registering a resource in the network. The first option which is the default choice for a resource is to register it in the local region, i.e. in the same region that it belongs to. Since registering it locally ensures that the tuple will be stored in a node in the same geographical region, it requires less time and overhead for registration. The resources that do not depend on a specific location or provide services that are location-independent, can register themselves in the public region as well. In addition to that, the directly connected IoT gateway (i.e. the IoT gateway w that the object o is connected to) keeps a copy of the registered resource locally in the cache for a specific time depending on the caching expiry parameters.

The overall workflow of resource registration and discovery is shown in Figure 6. An object $o \in \mathcal{O}$ registers its resource u in the network as tuples of $(tag, value, ownership, data)$. The set of the attributes that describe resource u are fed to the hash function to generate the *tag* parameter. The *value* in the added

[1] `https://docs.pi-hole.net/ftldns/dns-cache/`

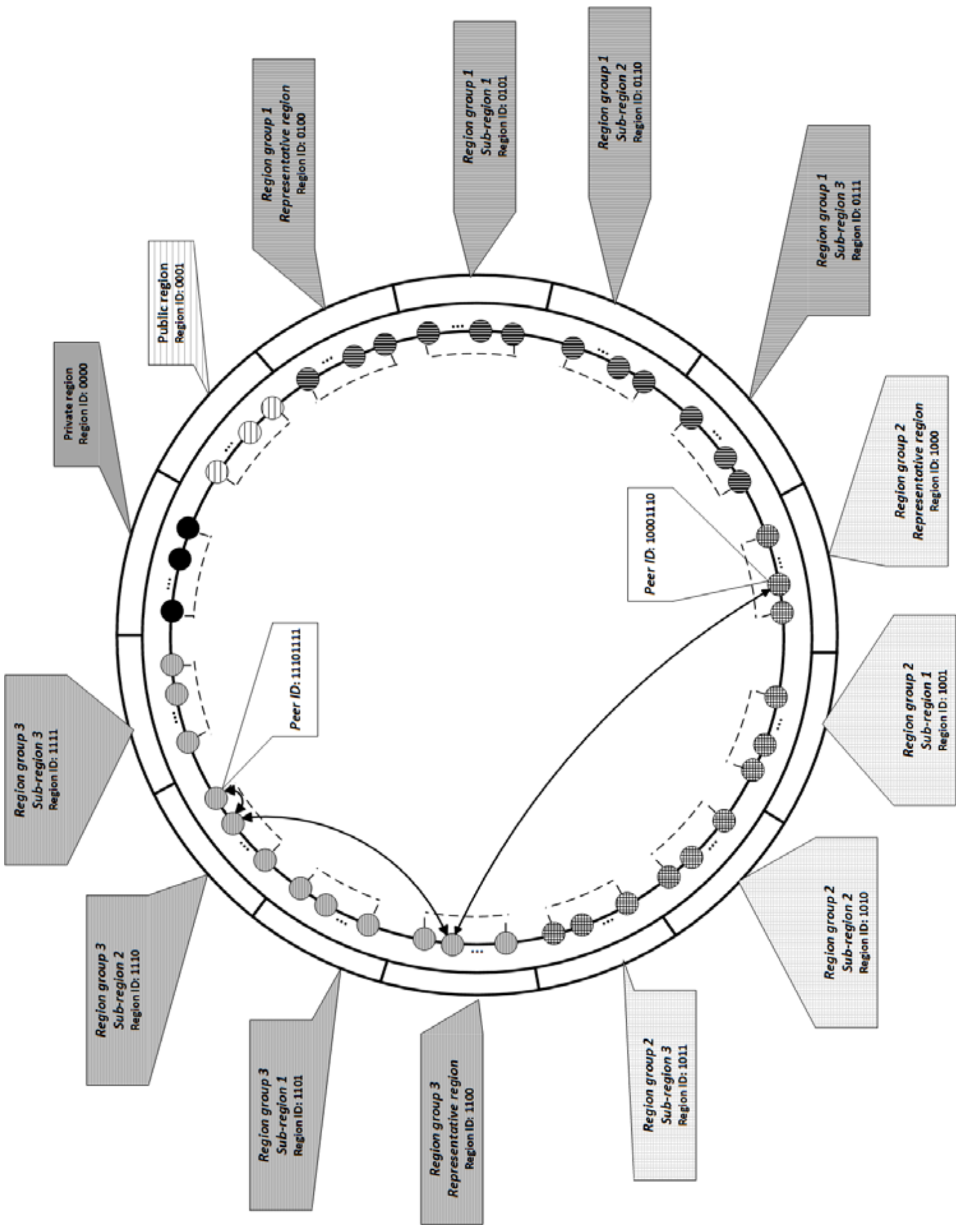


Figure 4: An example of Lamred implementation with 4 bits per region id and local id

tuple indicates the actual value of each of the attributes of the registered resource. During resource registration, the object o generates a random number r and adds

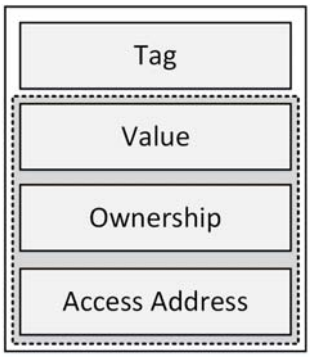


Figure 5: The public tuple structure in Lamred

its hashed value as a later proof of *ownership* of the generated tuple. Revealing the pre-image of the hash value (i.e. r) guarantees the ability of proving the ownership of the tuple that is used during updating or removing it from the Lamred. The access address (i.e. *data* in the tuple) parameter of the resource u might consist of its address, URI or other metadata about the resource u. The tuple is then stored in RDHT based on the its tags with a predefined number of replicas. The actual number depends on the *replication factor* rp. Choose an appropriate rp parameter depends on the nature of the network. As a general rule for choosing the appropriate rp value, the probability of existence of a subset of offline nodes in Lamred $\mathit{Offline} \subset \mathcal{W}$ with cardinality greater than the number of replicas has to be negligible ϵ. This is shown in equation 1. In addition, the existence of replicas increases the system performance by reducing the access load on any specific node in Lamred.

$$P(\|\mathit{Offline}\| \geq rp) < \epsilon \tag{1}$$

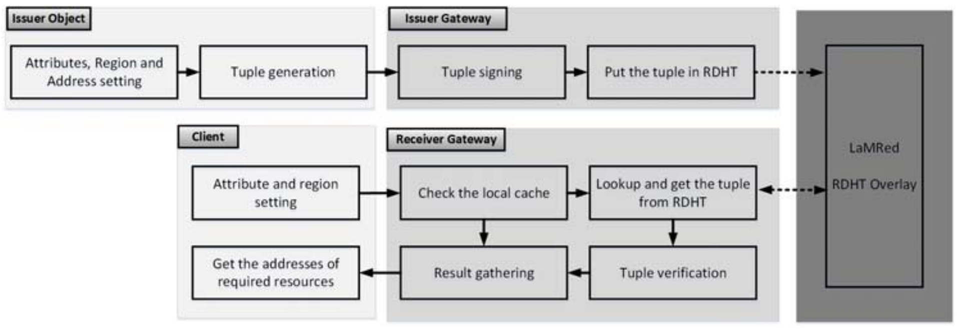


Figure 6: Overall workflow of resource registration and discovery in Lamred

Let $\mathcal{W}_{rg} \subset \mathcal{W}$ be a subset of IoT gateways in a specific region that the resource u has been registered in. In addition to storing the tuples locally in the directly connected IoT gateway $w \in \mathcal{W}_{rg} \subset \mathcal{W}$ and depending on the replication factor, the *close nodes* in the same local region $\mathcal{W}_{rg}$ to *tag* parameter are responsible for storing $(tag, value, ownership, data)$ tuple. If a node with identifiers id_w and a tuple with tag tag_v are close or equal, then we denote it with $id_w \approx tag_v$. The model does not depend on any specific *distance function (dst)* to compute the closeness. It can be any particular distance function. Metrics such as bitwise exclusive or (xor) [20] can be used to compute *dst* value.

Let $\mathcal{I}_d$ be a set of all possible sequences of d-bit binary digit (i.e. identifiers) in region rg and each peer $w \in \mathcal{W}_{rg} \subset \mathcal{W}$ has an identifier $id_w \in \mathcal{I}_d$ and each resource u has a $tag_u \in \mathcal{I}_d$. Let define the following set of peers

$$M(u) = \{w : tag_u \approx id_w, \nexists w' \mid dst(id_{w'}, tag_u) < dst(id_w, tag_u)\} \tag{2}$$

The set $M(u)$ links each resource u depending on its added attribute tag_u to a node $w \in \mathcal{W}_{rg}$ that its identifier $id_w \in \mathcal{I}_d$ is close or equal to tag_u. The cardinality of $M(u)$ depends on the replication factor rp parameter. The procedure of registering a public resource u in the network consists of three steps:

- `Tuple Definition and Generation`: The object o and based on the attributes that describe the resource u generates the tags, i.e. hash value of the attributes. Each of the tags is put along with their values, the ownership parameter that is the hash value of a randomly generated number r and the access address to the resource u and send the generated tuples to directly connected w. In addition to that, the object o determines whether u has to be stored in the same region or in the public region.
- `Tuple Signing`: In this step the appropriate set of tuples of the resource u are signed by $w \in \mathcal{W}$.
- `Resource Registration`: The gateway w registers the resource u by storing the tuples at the corresponding nodes in Lamred.

The public resources that are registered without any restrictions in Lamred can be discovered by all clients in the network based on their attributes and the registered regions. Lamred allows discovery of the registered resources based on one or more attributes. A client $c \in \mathcal{C}$ lookup for a resource by sending a lookup request with the required set of attributes, their values and the required region to the node w in Lamred that is directly connected to. The node w after receiving a discovery request from a client c generates the appropriate tags for the discovery process based on the received attributes. The discovery process contains three main steps as follows.

- `Query Generation`: In the first step, the node w generates the set of tags based on the received attributes from a client c. This is done by hashing each of the requested attributes in the client's request. In addition to that, the region id is also added to the generated tag.

- `Lookup`: The second step starts by issuing the lookup request by w in Lamred. The result $\mathcal{R}_i$ of each of the lookup operations is a set of data parameters that indicates the resulted resources based on the given attribute i and its required value.
- `Result Gathering`: After receiving the results and verifying them based on their attached digital signatures, the intersected members of sets $\mathcal{R}_0 \cap \mathcal{R}_1 \cap \cdot\cdot \cap \mathcal{R}_n$ will be returned as a result to the requested client c. In this step and prior to returning the result to client c, some scoring methods can be applied.

The tuples of the registered resources are remained in Lamred based on the caching expiry parameter. In addition to that, an object is able to update the data or remove its registered resource from Lamred by issuing a request including the pre-image of the ownership field in the added tuple. The request is signed by the directly connected node in Lamred, w and is sent to the corresponding node in Lamred. After checking the ownership of the tuple (i.e. $H(r) = ownership$), the requested tuple is updated by a new tuple or removed from Lamred based on the received request.

4.5 Private Resource Registration and Discovery

Every object o in the IoT network is able to keep a resource private and discoverable only by a predefined set $\mathcal{F}_o$ by generating a private tuple as illustrated in Figure 7. An object o has a set of its friends $\mathcal{F}_o = \{f_1,\ldots,f_n\} \subset C$ that can be communicated with in a secure and trusted way. The members of a friend set $\mathcal{F}_o$ of an object o are connected through members of $\mathcal{W}$ but they are not part of RDHT overlay itself. Each private resource has a private identifier id_u that is chosen uniformly at random from a given range, e.g. from bit strings of length 512. The identifier id_u is known only by the members of $\mathcal{F}_o$. Additionally, each $c \in \mathcal{C}$ has also a private identifier id_c that is chosen uniformly at random from a given range. The object o stores the private identifiers of each $f \in \mathcal{F}_o \subset \mathcal{C}$ locally. In addition, for every object o and for each $f \in \mathcal{F}_o$, an initial value (IV_{of}) and a common secret key (k_{of}) are generated and shared between them on a secure channel. The key k_{of} is used to encrypt the indirect transmitted data between them. These keys are stored at each node locally at the setup phase and its future distribution scheme is out of the scope of this paper.

If an object o registers its resource u privately, only the members of $\mathcal{F}_o$ can discover and access this specific resource of o. To do so, an object o has to generate a $privateTag_{uf}$ for a resource u and every friend client $f \in \mathcal{F}_o$ using equation 3. The access address of the private resource is then encrypted using the shared key k_{of}. A random number r is also generated and its hashed value is added as a later proof of *ownership* of the generated private tuple. A private resource can be registered privately in the local region or in the private region. While

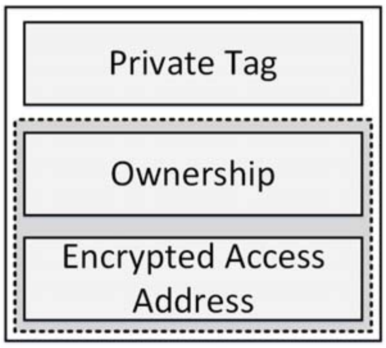


Figure 7: The private tuple structure in Lamred

registering a private resource in the private region does not guarantee the low latency process, but it hides the actual region that the private resource belongs to. On the other hand, registering a private resource in the local region ensures a low latency process, but reveals its local region. The decision of whether the private resource should be registered in the same local region or in the generic private region is made by the object o that handles the private resource. After generating the tuples as $(privateTag_{uf}, ownership, encryptedaccessAddress)$, the corresponding node in Lamred receives the resulted tuples (a single tuple for each $f \in \mathcal{F}_o$) from the directly connected object and put them in the same local region or the private region of RDHT. After registration, the members of $\mathcal{F}_o$ can discover the registered private resource by computing its private tag. These private tags are not permanent and used only once. The $privateTag_{uf}(new)$ parameter can be calculated using $privateTag_{uf}(old)$, id_u and id_f values. At any given time, the current private tag of a resource is computed as 3:

$$privateTag_{uf}(new) = H(privateTag_{uf}(old) \oplus id_u \oplus id_f) \tag{3}$$

where $privateTag_{uf}(old)$ is the previous private tag of the resource u (i.e. the output of the previous hash) and the initial value is $privateTag_{uf}(old) = H(IV_{of} \oplus id_u \oplus id_f)$. The one-time private tag ensures that the IoT gateway $w \in \mathcal{W}$ that $id_w \approx privateTag_{uf}$ is not the same during the life cycle of the resource. Similar to publicly registered resources, the private tuple of a privately registered resource can be updated or removed from Lamred by issuing a request and revealing the pre-image of the ownership field in the added private tuple. Although the discovery process of a private resource in the network resembles the public discovery, but there are two differences. First is that in order to be able to discover a resource u that is handled by o, a client has to be able to compute its private tag, i.e. being a valid member of $\mathcal{F}_o$. Secondly, after receiving the discovery result, the returned access data is confidential and can be read only by knowing the secret key k corresponding to this specific node.

5 Evaluation

5.1 Security Analysis

Theorem 1. *If $H(.)$ is a one-way hash function then the system satisfies resource anonymity.*

Proof. Suppose that the private tuple

$$P = (P_1|P_2|P_3) = (H(privateTag_{uf}(old) \oplus id_u \oplus id_f)|H(r)|Enc_{k_{of}}(accessAddress))$$

is stored in Lamred by object o to register the resource u that can be discovered only by its friend client $f \in \mathcal{F}_o$. Note that, only P_1 includes some information related to the private resource u (i.e. id_u), hence we can deal with this part of the private tuple only, hence the goal of the adversary is to compute id_u from the tuple. Let assume that the adversary knows $privateTag_{uf}(old)$ also (e.g. from previous communications). First suppose that a client $m \in \mathcal{C} \setminus \mathcal{F}_o$ wants to learn some information. Additionally, we can suppose that $m \in \mathcal{F}_f$, i.e. m knows id_f. If m could find a pre-image of $H(privateTag_{uf}(old) \oplus id_u \oplus id_f)$, and if she could remove $privateTag_{uf}(old) \oplus id_f$ then she can compute id_u. However, since $H()$ is a one-way function, m can find any x with $H(x) = P_1$ with negligible probability only. The remaining nodes in Lamred outside $\mathcal{F}_f$ are in a much hopeless situation, since even if they are assumed to find a pre-image of the hash, after that they have to remove $privateTag_{uf}(old)$ and id_f and the later is chosen randomly arising unconditional resource anonymity in this case. This completes the proof. □

Theorem 2. *If Enc is a computationally secure encryption then the system satisfies resource privacy.*

Proof. Suppose that the private tuple

$$P = (P_1|P_2|P_3) = (H(privateTag_{uf}(old) \oplus id_u \oplus id_f)|H(r)|Enc_{k_{of}}(accessAddress))$$

is stored in Lamred by object o to register the resource u that can be discovered only by its friend client $f \in \mathcal{F}_o$ and a malicious node $m \in \mathcal{W} \cup (\mathcal{C} \setminus \mathcal{F}_o)$ wants to discover and learn the access address of the registered private resource. Note that, only P_3 depends on the access address of the private resource, hence we can deal with this part of the private tuple only. This last part of the tuple is the $accessAddress$ encrypted with a computationally secure encryption. Therefore, without the knowledge of the symmetric key k_{of} the address $accessAddress$ can be computed with negligible probability only. This completes the proof. □

Theorem 3. *If $H(.)$ is a collision-resistant one-way hash function and Enc is a computationally secure encryption, then the system satisfies unforgeability.*

Proof. Suppose that the object o registers the resource u and the actual private tuple

$$P = (P_1|P_2|P_3) = (H(privateTag_{uf}(old) \oplus id_u \oplus id_f)|H(r)|Enc_{k_{of}}(accessAddress))$$

is stored in Lamred. Let $m \in \mathcal{W} \cup (\mathcal{C} \setminus \mathcal{F}_o)$ be a malicious node and first suppose that m wants to remove or update this tuple. Then m has to compute the pre-image of P_2, which is possible with negligible probability only since $H(.)$ is one-way.

Next, suppose that m wants to generate a new valid private tuple. To achieve this, the malicious node m has to first compute the new private tag (i.e. P_1) and then replace the part containing information related to $accessAddress$ (i.e. P_3). We will show that neither part of the tuple can be computed with non-negligible probability. First suppose that m wants to compute $P_1' = H(P_1 \oplus id_u \oplus id_f)$, furthermore assume that $m \in \mathcal{F}_f$ (i.e. m knows id_f) and $privateTag_{uf}(old)$ is also known by m. Then the only remaining part necessary for P_1' is id_u and only P_1 and $privateTag_{uf}(old)$ depends on this identifier. In both cases m has to compute the pre-image of the hash function $H(.)$ which can be done with negligible probability only, since $H(.)$ is a one-way function. Finally, suppose that m wants to compute $P_3' = Enc_{k_{of}}(accessAddress')$ for a fake address $accessAddress'$. Such fake address can be found with negligible probability since Enc is a computationally secure encryption. This completes the proof. □

5.2 Performance Analysis

In addition to proving the security properties in Lamred, the main concern is to keep the data of the registered public and private resources in the system as close as possible to the point of origin to prevent the high latency of long distances. In order to study the performance of Lamred and validate its feasibility and reliability, several issues such as region sizes, required preparation time for registration and discovery in constrained IoT devices, local and global discovery, and the affect of local cache size and churn on Lamred have been investigated. The network latency has been taken into consideration for measuring the performance of Lamred. Table 2 shows the assumed random parameters of real-time latency[2] for each of the different network links in the system.

Table 2: Network parameters

type	parameter
local connection latency	2 ms
sub-regional latency (local region)	3 - 8 ms
intra-regional latency (region set)	10 - 30 ms
long distance latency	80 - 120 ms

[2]`https://wondernetwork.com/pings`

We should call the reader's attention to the fact that the IoT gateways are the peers in RDHT and not the IoT clients or IoT resources. The members of $\mathcal{C}$ and $\mathcal{O}$ (i.e. clients and objects that handle IoT resources) are not part of RDHT itself and are connected through the peers in RDHT. The Kademlia implementation[3] of PeerSim simulator[22] has been used for the performance experiments. The implementation has been modified to fit our proposed model. In the implementation and as with uTorrent[4], the popular implementation of Kademlia, system wide replication is set to 8 and the lookup parallelism is set to 4. The results of researches [14][27] that focus on studying these two factors and other parameters in Kademlia [20] implementation to improve the lookup latency in DHT based implementation can be applied on Lamred. The system performance has been tested using a simulated Lamred network with 400 million to 2 billion IoT gateways. The IoT gateways are distributed and grouped in 200 region sets with 200 regions per region set (i.e. overall 40,000 regions with 10,000 to 50,000 IoT gateways per region). Table 3 shows the resource discovery latency in a local region. Figure 8 shows the resource discovery in Lamred with different region size and discovery scope. The sub-regional discovery is done within the same region, intra-regional discovery is done between two regions that are within the same region set and regional discovery (i.e. long distance discovery) is done between two regions that are in two different region sets. Due to huge number of IoT gateways in RDHT, the intra-regioal and regional discovery have been simulated in different stages. Without loss of generality, we assumed that no churn occurred and no cache has been used in Lamred during these tests.

Table 3: Resource discovery in a region in Lamred

region size	discovery latency
10,000	45.27 ms
20,000	47.14 ms
30,000	48.1 ms
40,000	48.9 ms
50,000	49.7 ms

To evaluate the efficiency of the Lamred for handling issues of robustness, availability, and replication, we performed a set of experiments where we introduced churn in the network. Over 100 to 2000 milliseconds intervals and for a period of 120 seconds, we randomly either killed an existing IoT gateway or started a new one in a region of 10,000 nodes. During the evaluation, resource discovery rate of 100 requests per second have been issued. As shown in the presented result in figure 9, there is 11 ms delay comparing to the network without churn in the discovery time in Lamred when the churn rate is 0.1 second (i.e. every 100 millisecond either an IoT gateway leaves or joins Lamred) and less than one millisecond delay when

[3]`http://peersim.sourceforge.net/`
[4]`https://www.utorrent.com/`

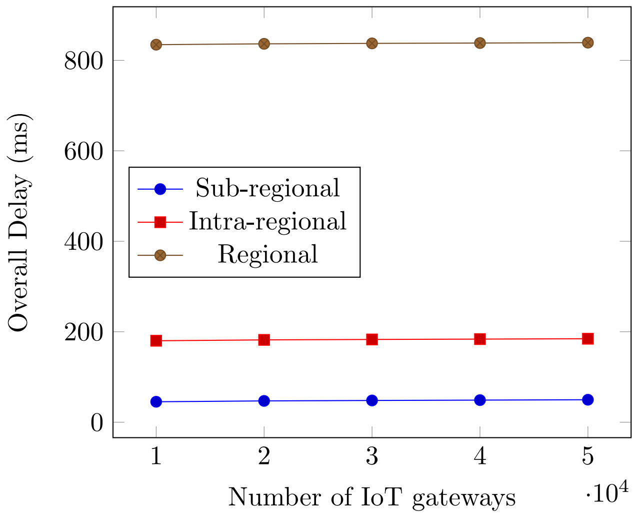


Figure 8: Regional Resource Registration Delay

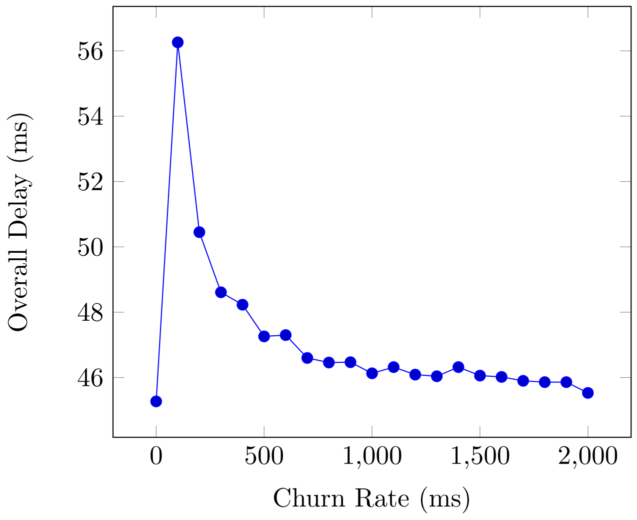


Figure 9: Churn affect on Lamred

the churn rate is higher than 1.6 second between each occurrence.

Figure 10 shows the affect of cache on Lamred. During the evaluation in a region of 10,000 IoT gateways and 1000 requests per second, the probability of discovering a resource that has been already resided in the local cache has been set to 0.05 - 0.25. The analysis showed that the local cache in Lamred nodes that includes the

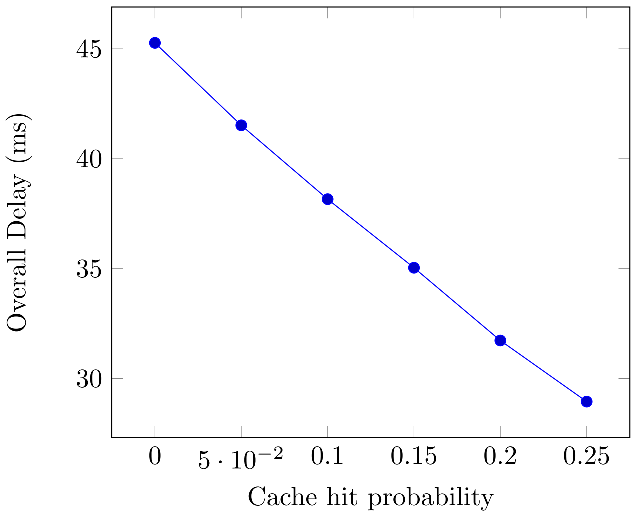


Figure 10: Cache affect on Lamred

locally registered and frequently discovered resources, improves the overall delay linearly.

As part of the evaluation, Lamred has been compared with the centralized service discovery (CSD) [13] and the decentralized resource discovery (DRD) [15] models. Since the DRD model [15] uses the IoT gateways without considering their locations, during analysis and comparison we simulated this model by creating a region set and assuming that the resources are registered in the regions without considering the locations of the resources. The direct matching scheme that has the minimum response time in the CSD model [13] has been used. There are 1000 IoT objects that have been distributed uniformly at random among a region in Lamred with 5,000 - 10,000 IoT gateways. As it appears from figure 11, although the resource discovery in centralized discovery is fixed, but the lookup process of discovering a resource in the network of a centralized scheme is higher than the proposed model. At the same time, as it is notable, when in the proposed model the number of gateways in the system increases the delay of the lookup process increases logarithmically. The reason is the use of the DHT overlay for discovery in which the lookup time among n peers is $log(n)$.

Lamred shows that it has a low latency that makes it suitable for IoT network with large number of IoT resources. The latency in a region of Lamred has been compared with the latency in some of the recent works and the result is listed in Table 4.

The private resource registration and discovery follows a different approach than other regions, as discussed in Section 4.5. In this case the object has to generate the private tag of the resource to be used in the private region of Lamred and on the other hand, the client application has to calculate the private tag in order to

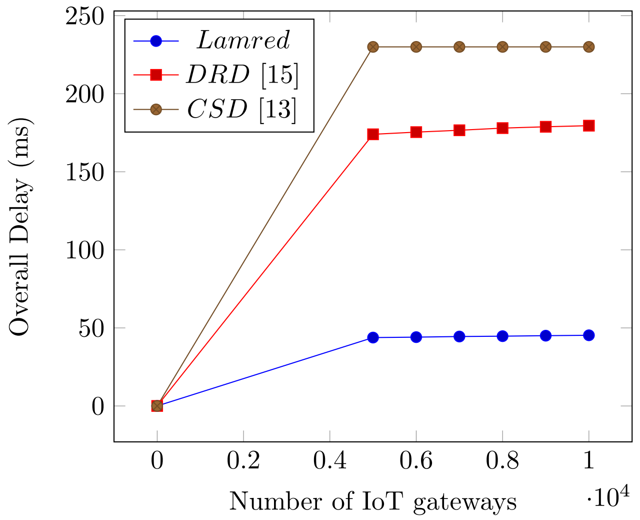


Figure 11: Resource discovery delay in different models

Table 4: Latency in resource discovery Models

Model	Properties	Latency
Datta and Bonnet [8]	Search Engine Based Resource Discovery	450-600 ms
Jia et. al. [13]	Centralized Resource Discovery	230 ms
Kamel et. al. [15]	Decentralized Resource Discovery	150 ms
Pahl et. al. [23]	4 predicates/ search providers	80 ms
Lamred	A region with 10,000 IoT gateways	45 ms

be able to discover the private resource and get the encrypted access address of it. The private tag generation in equation 3 has been tested on an MCU with single-core 32-bit 80 MHz microcontroller. The SHA256 [10] is used as hashing algorithm for tag generation and AES-128-CBC is used as encryption algorithm. For analysis and implementation of SHA256 and AES algorithms on the MCU, the Crypto library[5] for the ESP8266 IoT devices has been used. During the test, each of the cryptographic operations has been repeated 20 times, and their mean value is registered. The average time required to perform the encryption, decryption, hashing and the private tag generation and discovery are shown in Tables 5 and 6.

[5]`https://github.com/intrbiz/arduino-crypto`

Table 5: Required operation time by the microcontroller for private resource registration

Operation	Required time
XOR operation	8 ms
RNG	5 ms
SHA256 (Private Tag and RN)	2 * 227 ms
AES-128-CBC encryption	288 ms
Tag generation (Registration)	**805 ms**

Table 6: Required operation time by the microcontroller for private resource discovery

Operation	Required time
XOR operation	8 ms
SHA256 (Private Tag)	227 ms
AES-128-CBC decryption	348 ms
Tag generation (Discovery)	**583 ms**

5.3 Complexity Analysis

Suppose that Lamred consists of 2^g regions, divided into $N_{RegionSet}$ region sets. Let's suppose that the cardinality of IoT gateways in the private and public regions are N_P, and in the local regions $R1$, $R2$ and $R3$ of RDHT overlay are N_{R1}, N_{R2} and N_{R3}, respectively. Suppose that both $R1$ and $R2$ are in the same region set that includes N_{RS1} local regions, and $R3$ is in a different region set. We discuss the complexity of the proposed model in five cases:

- Sub-regional registering or discovering a resource in the same local region (RD_{local})
- Location independent registering or discovering a resource in the private/ public regions (RD_{pp})
- Discovering a resource that is stored in the cache of an IoT gateway (D_{cache})
- Intra-regional discovering a resource in the same region set (RD_{RS})
- Regional discovering of a resource in a different region set than the client region ($D_{regional}$)

Registering or discovering a resource in the same region $R1$ that the object and client belong to is done by the corresponding peer $w \in \mathcal{W}_{R1}$ and is equal to $RD_{local} = O(log(N_{R1}))$. Registering or discovering a resource in the private or public regions regardless of its location is done by first reaching a node in the target private or public regions and then finding the exact node in these regions responsible for storing the access address of the required resource. Since each node in Lamred has the access addresses of nodes in public and private regions, it takes $O(1)$ to reach each of these two regions and then perform a lookup request for the exact required node. Therefore, registering or discovering a resource in the private or public regions depends on the number of nodes in each of these regions and is equal to $RD_{PP} = O(log(N_P))$.

Each IoT gateway in Lamred keeps a copy of the registered or previously discovered resources locally for a specific time depending on the caching expiry parameters. If a client requests to discover a resource that resides in the cache (which happens for the frequently discovered resources), then the result is returned directly to the client and is equal to $D_{cache} = O(1)$. If the client and the discovered resource are in regions $R1$ and $R2$ that are in the same region set including overall N_{RS1} local regions, the discovery access time takes $RD_{RS} = O(log(N_{RS1}N_{R2}))$ and is done in two stages. Firstly, it takes $O(log(N_{RS1}))$ to reach the target region (i.e. $R2$) and then it takes $O(log(N_{R2}))$ to discover the required resource by reaching the specific responsible node in target region $R2$. Discovering a resource in region $R2$ by a client belongs to region $R3$ that is in a different region set rakes $D_{regional} = O(log(N_{RegionSet}N_{RS1}N_{R2}))$ and is done in three stages. Firstly, accessing the representative region of the region set that the target region $R2$ belongs to takes $O(log(N_{RegionSet}))$ based on the number of available region sets in Lamred. Then, reaching the region $R2$ takes $O(log(N_{RS1}))$. Finally, it takes $O(log(N_{R2}))$ to perform a lookup and discover the required resource by reaching the specific responsible node in target region $R2$.

6 Conclusion

In this paper a location aware and privacy preserving multi layer model of resource discovery (Lamred) in IoT has been proposed. It adopts the peer to peer (P2P) scheme by utilizing Regional Distributed Hash Table (RDHT), a proposed version of DHT. Lamred ensures that there is no single point of failure in the system and the network can be easily scaled without any need of a reorganizing and synchronizing authority. The RDHT overlay is generated by taking into consideration the physical location of IoT gateways in the system. Resources are not part of RDHT overlay, but they can be registered locally, globally or privately different regions in RDHT through an IoT gateway. On the other hand, clients can discover the resources based on one or more attributes of the required resources. During the discovery phase, the client can choose a specific local region or the public region for the discovery of the resources. The private resources that are registered privately either in the local region or in the private region can only be discovered by a predefined set of clients in Lamred. During the evaluation, Lamred showed a lower latency comparing to the centralized and location-independent decentralized resource discovery models. In addition, the required security properties of the registered resources in Lamred, namely resource anonymity, privacy, and unforgeability have been proved.

Some open problems remain related to the proposed model. On one hand, while the current model supports registering private resources in Lamred, but it offers a two-level binary policy and can not define the set of attributes of clients that are able to discover privately registered resources. This problem has to be addressed in future works. On the other hand, in Lamred a separate lookup in the DHT overlay for each of the attributes is issued which will add a significant overhead to the system, there should be a future study to improve the efficiency of the discovery process.

References

[1] Ali, Shabir, Banerjea, Shashwati, Pandey, Mayank, and Tyagi, Neeraj. Wireless Fog-Mesh: A communication and computation infrastructure for IoT based smart environments. In *Mobile, Secure, and Programmable Networking*, pages 322–338, 2019. DOI: `10.1007/978-3-030-03101-5_27`.

[2] Bhatnagar, Nirdosh. *Mathematical Principles of the Internet.* CRC Press, 2019. ISBN: 9781138505483.

[3] Bonomi, Flavio, Milito, Rodolfo, Zhu, Jiang, and Addepalli, Sateesh. Fog computing and its role in the Internet of Things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pages 13–16. ACM, 2012. DOI: `10.1145/2342509.2342513`.

[4] Bormann, Carsten, Castellani, Angelo P, and Shelby, Zach. CoAP: An application protocol for billions of tiny internet nodes. *IEEE Internet Computing*, 16(2):62–67, 2012. DOI: `10.1109/MIC.2012.29`.

[5] Cheshire, Stuart and Krochmal, Marc. DNS-based service discovery. RFC 6763, RFC Editor, 2013.

[6] Cirani, Simone, Davoli, Luca, Ferrari, Gianluigi, Léone, Rémy, Medagliani, Paolo, Picone, Marco, and Veltri, Luca. A scalable and self-configuring architecture for service discovery in the Internet of Things. *IEEE Internet of Things Journal*, 1(5):508–521, 2014. DOI: `10.1109/JIOT.2014.2358296`.

[7] Damgård, Ivan Bjerre. A design principle for hash functions. In *Conference on the Theory and Application of Cryptology*, pages 416–427. Springer, 1989. DOI: `10.1007/0-387-34805-0_39`.

[8] Datta, Soumya Kanti and Bonnet, Christian. Search engine based resource discovery framework for Internet of Things. In *2015 IEEE 4th Global Conference on Consumer Electronics (GCCE)*, pages 83–85. IEEE, 2015. DOI: `10.1109/GCCE.2015.7398707`.

[9] Datta, Soumya Kanti, Da Costa, Rui Pedro Ferreira, and Bonnet, Christian. Resource discovery in Internet of Things: Current trends and future standardization aspects. In *2nd World Forum on Internet of Things (WF-IoT)*, pages 542–547. IEEE, 2015. DOI: `10.1109/WF-IoT.2015.7389112`.

[10] Eastlake, Don and Hansen, Tony. US secure hash algorithms (SHA and HMAC-SHA). RFC 4634, RFC Editor, 2006.

[11] Hunkeler, Urs, Truong, Hong Linh, and Stanford-Clark, Andy. MQTT-S—A publish/subscribe protocol for Wireless Sensor Networks. In *2008 3rd International Conference on Communication Systems Software and Middleware and Workshops (COMSWARE'08)*, pages 791–798. IEEE, 2008. DOI: `10.1109/COMSWA.2008.4554519`.

[12] Jara, Antonio J, Lopez, Pablo, Fernandez, David, Castillo, Jose F, Zamora, Miguel A, and Skarmeta, Antonio F. Mobile digcovery: A global service discovery for the Internet of Things. In *2013 27th International Conference on Advanced Information Networking and Applications Workshops*, pages 1325–1330. IEEE, 2013. DOI: `10.1109/WAINA.2013.261`.

[13] Jia, Bing, Li, Wuyungerile, and Zhou, Tao. A centralized service discovery algorithm via multi-stage semantic service matching in Internet of Things. In *2017 IEEE International Conference on Computational Science and Engineering (CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC)*, volume 1, pages 422–427. IEEE, 2017. DOI: `10.1109/CSE-EUC.2017.82`.

[14] Jimenez, Raul, Osmani, Flutra, and Knutsson, Björn. Sub-second lookups on a large-scale Kademlia-based overlay. In *2011 IEEE International Conference on Peer-to-Peer Computing*, pages 82–91. IEEE, 2011. DOI: `10.1109/P2P.2011.6038665`.

[15] Kamel, Mohammed B. M., Crispo, Bruno, and Ligeti, Peter. A decentralized and scalable model for resource discovery in IoT network. In *2019 International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, pages 1–4. IEEE, 2019. DOI: `10.1109/WiMOB.2019.8923352`.

[16] Kamel, Mohammed B. M., Ligeti, Peter, Nagy, Adam, and Reich, Christoph. Distributed Address Table (DAT): A decentralized model for end-to-end communication in IoT. To appear in Journal of P2P Networking and Applications, 2021.

[17] Kamel, Mohammed B. M., Ligeti, Peter, and Reich, Christoph. Region-based distributed hash table for fog computing infrastructure. In *13th Joint Conference on Mathematics and Informatics*, pages 82–83, 2020.

[18] Lua, Eng Keong, Crowcroft, Jon, Pias, Marcelo, Sharma, Ravi, and Lim, Steven. A survey and comparison of peer-to-peer overlay network schemes. *IEEE Communications Surveys & Tutorials*, 7(2):72–93, 2005. DOI: `10.1109/COMST.2005.1610546`.

[19] Maurer, Ward Douglas and Lewis, Theodore Gyle. Hash table methods. *ACM Computing Surveys (CSUR)*, 7(1):5–19, 1975. DOI: `10.1145/356643.356645`.

[20] Maymounkov, Petar and Mazieres, David. Kademlia: A peer-to-peer information system based on the xor metric. In *International Workshop on Peer-to-Peer Systems*, pages 53–65. Springer, 2002. DOI: `10.1007/3-540-45748-8_5`.

[21] Mokadem, Riad, Hameurlain, Abdelkader, and Tjoa, A Min. Resource discovery service while minimizing maintenance overhead in hierarchical DHT systems. *International Journal of Adaptive, Resilient and Autonomic Systems (IJARAS)*, 3(2):1–17, 2012. DOI: `10.4018/jaras.2012040101`.

[22] Montresor, Alberto and Jelasity, Márk. PeerSim: A scalable P2P simulator. In *Proc. of the 9th Int. Conference on Peer-to-Peer (P2P'09)*, pages 99–100, Seattle, WA, September 2009. DOI: `10.1109/P2P.2009.5284506`.

[23] Pahl, M. and Liebald, S. A modular distributed IoT Service Discovery. In *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pages 448–454, 2019.

[24] Pahl, Marc-Oliver. *Distributed smart space orchestration*. PhD thesis, Technische Universität München, 2014. `https://mediatum.ub.tum.de/1196145`.

[25] Pattar, Santosh, Buyya, Rajkumar, Venugopal, KR, Iyengar, SS, and Patnaik, LM. Searching for the IoT resources: Fundamentals, requirements, comprehensive review, and future directions. *IEEE Communications Surveys & Tutorials*, 20(3):2101–2132, 2018. DOI: `10.1109/COMST.2018.2825231`.

[26] Picone, Marco, Amoretti, Michele, and Zanichelli, Francesco. Geokad: A P2P distributed localization protocol. In *2010 8th IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*, pages 800–803. IEEE, 2010. DOI: `10.1109/PERCOMW.2010.5470545`.

[27] Roos, Stefanie, Salah, Hani, and Strufe, Thorsten. On the routing of Kademlia-type systems. In *Advances in Computer Communications and Networks*. River Publishers, 2017.

[28] Rowstron, Antony and Druschel, Peter. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing*, pages 329–350. Springer, 2001. DOI: `10.1007/3-540-45518-3_18`.

[29] Stoica, Ion, Morris, Robert, Karger, David, Kaashoek, M Frans, and Balakrishnan, Hari. Chord: A scalable peer-to-peer lookup service for internet applications. *ACM SIGCOMM Computer Communication Review*, 31(4):149–160, 2001. DOI: `10.1145/964723.383071`.

[30] Wirtz, H., Heer, T., Hummen, R., and Wehrle, K. Mesh-DHT: A locality-based distributed look-up structure for Wireless Mesh Networks. In *2012 IEEE International Conference on Communications (ICC)*, pages 653–658, June 2012. DOI: `10.1109/ICC.2012.6364336`.

[31] Wirtz, Hanno, Heer, Tobias, Serror, Martin, and Wehrle, Klaus. DHT-based localized service discovery in wireless mesh networks. In *2012 IEEE 9th International Conference on Mobile Ad-Hoc and Sensor Systems (MASS 2012)*, pages 19–28. IEEE, 2012. DOI: `10.1109/MASS.2012.6502498`.

[32] Yang, Yijun, Zhang, Xiaomei, Yu, Jianping, Zhang, Peng, et al. Research on the hash function structures and its application. *Wireless Personal Communications*, 94(4):2969–2985, 2017. DOI: `10.1007/s11277-016-3760-4`.

[33] Zhao, Ben Y, Huang, Ling, Stribling, Jeremy, Rhea, Sean C, Joseph, Anthony D, and Kubiatowicz, John D. Tapestry: A resilient global-scale overlay for service deployment. *IEEE Journal on selected areas in communications*, 22(1):41–53, 2004. DOI: `10.1109/JSAC.2003.818784`.

Acta Cybernetica **25** (2021) 351–367.

Distance-Based Skeletonization on the BCC Grid[*]

Gábor Karai[ab] and Péter Kardos[ac]

Abstract

Strand proposed a distance-based thinning algorithm for computing surface skeletons on the body-centered cubic (BCC) grid. In this paper, we present two modified versions of this algorithm that are faster than the original one, and less sensitive to the visiting order of points in the sequential thinning phase. In addition, a novel algorithm capable of producing curve skeletons is also reported.

Keywords: BCC grid, distance transform, topology preservation, thinning

1 Introduction

Skeletons are region-based shape descriptors which summarize the general form of (segmented) digital binary objects [17]. In 3D, *surface skeletons* (that may contain 2D thin surface patches) are to represent general objects, and *curve skeletons* (that are stick-like 1D descriptors) are concise representations of tubular and tree-like 3D objects. Distance-based skeletonization techniques focus on the detection of ridges in the *distance map* by using a proper distance and a fast algorithm for distance transform [2, 3]. Another strategy for skeletonization, called as *thinning*, is an iterative object reduction in a topology preserving way [11]. Distance-based methods are often combined with thinning. These approaches can be further classified as follows [5]:

- *Anchor-based thinning*: First, the set of *centres of maximal balls (CMB's)* is detected as safe skeletal (anchor) points. Then thinning is applied to connect CMB's and determining the set of remaining skeletal points.
- *Distance-ordered thinning*: Distance mapping is followed by the thinning process. In the k-th iteration, only object points with distance value k are taken into consideration for possible deletion.

[*]This research was supported by the project "Integrated program for training new generation of scientists in the fields of computer science", no EFOP-3.6.3-VEKOP-16-2017-00002. The project has been supported by the European Union and co-funded by the European Social Fund.

[a]Institute of Informatics, University of Szeged, Hungary
[b]E-mail: karai@inf.u-szeged.hu, ORCID: 0000-0001-9609-8628
[c]E-mail: pkardos@inf.u-szeged.hu, ORCID: 0000-0001-8857-4102

DOI: 10.14232/actacyb.290076

- *Hybrid approach*: It combines anchored and distance-ordered thinning.

Most 3D skeletonization algorithms work on digital pictures sampled on the cubic grid. An alternative structure, the *body-centered cubic* (BCC) grid tessellates the space into truncated octahedra, which results in a less ambigous connectivity structure compared to the cubic grid [21]. The importance of BCC grid shows an upward tendency in the analysis of 3D digital images [1, 13, 14, 15, 24].

The first skeletonization algorithm for binary objects sampled on the BCC grid was proposed by Strand [19]. His algorithm considers a fixed distance, and is only capable of producing surface skeletons. According to our best knowledge, there is not any other similar result in the later literature. In this paper, we present two modified versions of Strand's method that work for arbitrary distances, and a further one capable of producing curve skeletons.

The rest of this work is organized as follows. Section 2 reviews the basic notions and results of 3D digital topology. In Section 3, we discuss the original algorithm of Strand working on the BCC grid. In Section 4, we propose the modified versions of that algorithm for computing surface skeletons, while in Section 5, we also present a variant to produce curve skeletons. Some experimental results are shown in Section 6, and we round off this work with some concluding remarks.

2 Definitions and Notions

In this section, we review the basic concepts of digital topology and distance transform.

Let us denote by $\mathbb{B}$ the BCC grid, whose elements are called *points*. The BCC grid is defined as the following subset of $\mathbb{Z}^3$:

$$\mathbb{B} = \{(x, y, z) \in \mathbb{Z}^3 \mid x \equiv y \equiv z \pmod 2\}. \tag{1}$$

We make a distinction among the following three types of neighborhood of a point $p = (p_x, p_y, p_z) \in \mathbb{B}$ (see Figure 1):

$$N_6(p) = \{(q_x, q_y, q_z) \in \mathbb{B} \mid |p_x - q_x| + |p_y - q_y| + |p_z - q_z| = 2\},$$

$$N_8(p) = \{(q_x, q_y, q_z) \in \mathbb{B} \mid |p_x - q_x| + |p_y - q_y| + |p_z - q_z| = 3\},$$

$$N_{14}(p) = N_6(p) \cup N_8(p).$$

Two points $p, q \in \mathbb{B}$ are *i-adjacent* if $q \in N_i(p)$ $(i = 6, 8, 14)$. Furthermore, let $N_i^*(p) = N_i(p) \setminus \{p\}$.

The *lexicographical order* relation "$\prec$" between two distinct points $p = (p_x, p_y, p_z)$ and $q = (q_x, q_y, q_z)$ in $\mathbb{B}$ is defined as follows:

$$p \prec q \quad \Leftrightarrow \quad (p_z < q_z) \vee (p_z = q_z \wedge p_y < q_y) \vee (p_z = q_z \wedge p_y = q_y \wedge p_x < q_x).$$

The sequence of distinct points $\langle x_0, x_1, \ldots, x_s\rangle$ in a non-empty set of points $X \subset \mathbb{B}$ is called an *i-path* of length s from x_0 to x_s in X if x_k is i-adjacent to x_{k-1}

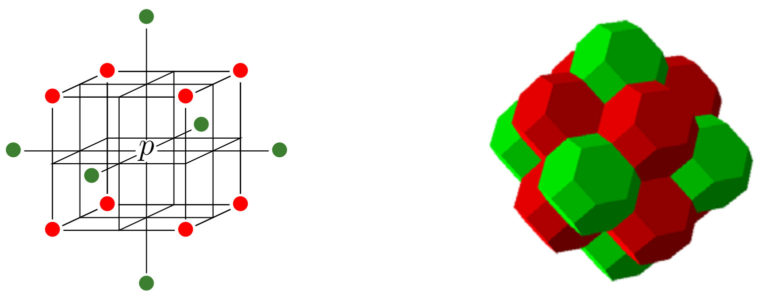


Figure 1: Elements of $N_{14}(p)$ in $\mathbb{B}$ (left), voxel representation of these points (right). The set $N_6(p)$ contains the central point p and the six points (voxels) marked green. The set $N_8(p)$ contains p and the eight points (voxels) marked red.

$(i \in \{6, 8, 14\}, k = 1, \ldots, s)$. Note that a single point is an *i-path* of length 0. Two points $p, q \in \mathbb{B}$ are said to be *i-connected* in $X \subseteq \mathbb{B}$ if there is an i-path from p to q in X. The set X is *i-connected* in the set of points $Y \supseteq X$ if any two points in X are i-connected in Y.

Following the concept of digital pictures in [9], we define the $(14, 14)$ *binary digital picture* as a quadruple $\mathcal{P} = (\mathbb{B}, 14, 14, B)$. Each point in $B \subseteq \mathbb{B}$ is called a *black point* and has a value of 1 assigned to it. Picture $\mathcal{P}$ is finite if it contains finitely many black points. Each point in $\mathbb{B} \setminus B$ is called a *white point* and has a value of 0 assigned to it. 14-adjacency is associated with both black and white points. A *black component* or an *object* is a maximal 14-connected set of points in B, while a *white component* is a maximal 14-connected set of points in $\mathbb{B} \setminus B$. A black point is called a *border point* in a picture if it is 14-adjacent to at least one white point. Furthermore, a border point p is called a *strong border point*, if $N_8(p)$ contains at least one white point, or else p is called a *weak border point*. In a finite picture, there is a unique white component that is called the *background*. A finite white component is called a *cavity*.

A *reduction* transforms a binary picture only by changing some black points to white ones (which is referred to as the deletion of 1's). A 3D reduction does *not* preserve topology [8] if

- any object in the input picture is split (into several objects) or is completely deleted,
- any cavity in the input picture is merged with the background or another cavity,
- a cavity is created where there was none in the input picture, or
- a hole (that e.g. doughnuts have) is eliminated or created.

A *simple point* is a point whose deletion is a topology preserving reduction [9]. Sequential thinning algorithms traverse the border points of a picture, and focus on the actually visited single point for possible deletion, hence for such algorithms, the deletion of only simple points ensures topology preservation. The following result states that simplicity is a local property in $\mathbb{B}$ which can be verified by investigating the 14-neighborhood of points:

Theorem 1. [23] *Let p be a black point in a $(\mathbb{B}, 14, 14, B)$ picture. Then p is a simple point if and only if the following conditions hold:*

1. *Point p is 14-adjacent to just one 14-component of $N^*_{14}(p) \cap B$.*
2. *Point p is 14-adjacent to just one 14-component of $N_{14}(p) \setminus B$.*

Figure 2 shows examples for simple and non-simple points.

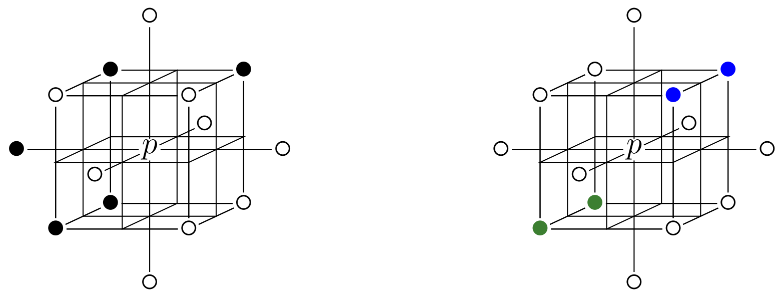


Figure 2: Example for a simple (left) and a non-simple point p (right) on the BCC grid. The distinct black 14-components are labeled with different colors.

Let Γ be the set of white points in a picture. The *distance transform* is a function that assigns to each point p the distance between p and the closest border point $q \in \Gamma$ to it. The most frequently used distance functions of two points p and q are the Euclidean distance, $d_e(p,q) = \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2 + (p_z - q_z)^2}$ and the neighborhood distances, where $d_i(p,q)$ denotes the length of the shortest i-path between p and q [12]. In $\mathbb{B}$, neighborhood distances $d_8(p,q)$ and $d_{14}(p,q)$ are taken into consideration. These distances, however, do not give a good approximations to the Euclidean distance. As a better solution for this purpose, the lengths of the moves from a point to its neighbors can be weighted according to some criteria, and the path of the minimal sum of weights called as *Chamfer distance*s can be used [4, 12]. Let $\langle a, b \rangle$ denote the general *Chamfer mask* for weighted distance transform on the BCC grid, where a and b are the weights assigned to all points in $N^*_8(p)$ and $N^*_6(p)$, respectively. Some Chamfer masks are presented in [7, 22].

Let DT be a distance map of a $(14, 14)$ picture. Point $p \in B$ is called a *local maximum* if for any point $q \in N^*_{14}(p)$, $DT(p) \geq DT(q)$. An *inscribed ball* is a sphere that is contained within the object and tangent to at least one of the object's faces, and it's called *maximal* if it is not covered by any other inscribed ball. A *center of maximal ball (CMB)* is labeled with the radius of that ball [5]. Point $p \in B$ is a CMB if for any point $q \in N^*_{14}(p)$, $DT(q) < DT(p) + w$, where w is the weight that corresponds to q on the Chamfer mask when placed on p. In case of d_8 distance, only N_8^* is considered for detection of local maxima and CMB's.

3 Strand's algorithm

The method proposed by Strand uses d_{14} distance and sequential thinning, and it can produce only surface skeletons by preserving non-simple points and surface edge points. A point $p \in B$ is called a *surface edge point*, if the following two conditions hold:

- there are two points $q, r \in N_{14}^*(p) \cap B$ such that $N_{14}(q) \cap N_{14}(r) = \{p\}$ and
- there is no point $s \in N_{14}^*(p) \cap B$ such that $N_{14}(p) \cap N_{14}(s) \subseteq B$.

The thinning part consists of two phases. Forward thinning reduces the input object to a 3–4 voxel thick object, which is peeled further during backward thinning. Note that surface edge point condition is applied only in the last phase. The unusual in this process is that during the backward thinning phase, object points are visited in descending order of their distance value. The sequential thinning suffers from the disadvantage of being sensitive to the visiting order of border points with the same distance value. As a result the final skeleton usually has some "false" skeleton points, which causes insignificant segments. This situation is illustrated in Figure 3 and an example for its occurrence can be found in Figure 4.

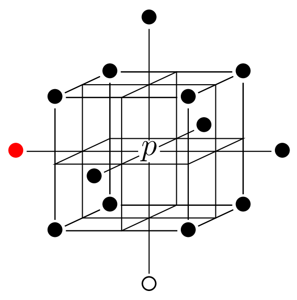


Figure 3: Example for unfavorable visiting order during sequential thinning. Weak border point p becomes non-simple after deletion of its neighbor marked red.

Theorem 2. *The runtime complexity of Strand's algorithm (see Algorithm 1) is $O(|I|^{4/3})$, where $|I|$ denotes the number of points in the image.*

Proof. The first phase of the algorithm is linear since computing the d_{14} distance transform requires two raster scans [22] and the sequential detection of CMB's takes one extra scan. Forward thinning is linear too, because the object point's local neighborhood is examined exactly once. In worst case, weak border points on the original object may appear as surface or line endpoints, which means the

Algorithm 1 Strand's skeletonization algorithm.

```
Input: picture $(\mathbb{B}, 14, 14, X)$ with the initial objects in it
Output: picture $(\mathbb{B}, 14, 14, X)$ containing the surface skeleton
    // Distance mapping and identifying CMB's
 1: $DT \leftarrow computeDT(X)$
 2: $H \leftarrow \{p \in X \mid p$ is a $CMB\}$
    // Forward thinning
 3: for $k \leftarrow 1$ to $max(DT)$ do
 4:   for each $p \in X$ do
 5:     if $DT(p) = k$ and $p$ is simple and $N_{14}(p) \cap H = \emptyset$ then
 6:       $X \leftarrow X \setminus \{p\}$
 7:     end if
 8:   end for
 9: end for
    // Backward thinning
10: repeat
11:   $changed1 \leftarrow false$
12:   for $k \leftarrow max(DT)$ downto 1 do
13:     repeat
14:       $changed2 \leftarrow false$
15:       $D \leftarrow \{p \in X \setminus H \mid DT(p) = k$ and $p$ is simple$\}$
16:       for each $p \in D$ do
17:         if $p$ is simple and $p$ is not a surface edge point then
18:           $X \leftarrow X \setminus \{p\}$
19:           $changed1 \leftarrow true$
20:           $changed2 \leftarrow true$
21:         end if
22:       end for
23:     until $changed2 = false$
24:   end for
25: until $changed1 = false$
```

algorithm may classify black points incorrectly during the first iteration. As a consequence, the length of these segments is equal to half of the object's thickness, which we refer to as T_{obj}.

During the *for* loop from Step 12 to 24, all remaining object points are visited, but it can peel only a one-unit layer from each segment because of the descending visiting order of the object point's distance value, since only simple points can be deletable. Hence, each object point will be visited at most T_{obj} times until the algorithm terminates, so the runtime complexity is $O(T_{obj} \cdot |I|)$. T_{obj} is maximal, if the input image has a cubic shape, which means the image size is the same for all dimensional axes. In this case, the original object is $\sqrt[3]{|I|}\,/\,2$ thick, if all points in the image are black. By inserting it into the previous formula, we get

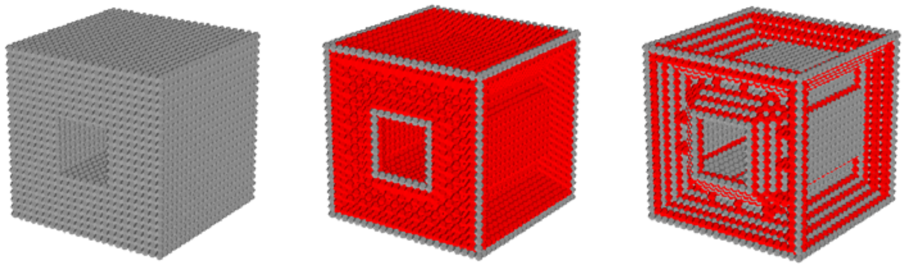


Figure 4: Result of Strand's algorithm on a holey cube. In the middle and right figure anchor voxels are gray and further skeleton voxels are red. Numbers in parentheses show the number of black points.

$O(\sqrt[3]{|I|}\,/\,2 \cdot |I|) = O(|I|^{4/3})$, which is not just the third phase's, but the whole algorithm's runtime complexity. $\square$

4 Our two modified algorithms to produce surface skeletons

To construct linear time algorithms, we merge the thinning phases in Algorithm 1 and simplify the organization of the thinning iterations. Our deletion rule also preserves non-simple points and surface edge points. We introduce a new parameter N which gives an upper limit to the visiting number of all black points during the thinning phase (i.e. each iteration will be repeated at most N times). Note that setting N to ∞ means that the visiting number is not limited. This parameter is applied in both of our improved algorithms. The motivation is to reduce the sensitivity to the visiting order of border points. The sequential thinning part consists of two substeps in our modified algorithms. First, we collect the deletable points from the actual image into set D. Then we individually delete each point in this set, if they are still simple when visited. In order to lessen the occurrence of false skeleton points, elements of D are visited in lexicographical order.

Our first improved variant, called *AB-N-visits*, is an anchor-based thinning method that considers local maxima instead of CMB's to be anchor points because many CMB's are proved to be "false" skeleton points on weighted distance maps [5]. In this approach, we consider only strong border points to make sure the strong and weak border points will be visited in different iterations. At the end of each iteration, all visited but non-deleted points are insterted in the set S of skeleton points (see Step 17 of Algorithm 2). This operation guarantees that no border points will be examined during the further iterations which already have been visited.

Our second improved variant, called *DO-N-visits*, is a distance-ordered thinning method that omits the detection of anchor points, i.e. the set H is not used in this version. The border points are visited in ascending order of their distance value during the thinning phase.

Algorithm 2 Anchor-based variant *AB-N-visits* to extract surface skeleton.

```
Input: picture $(\mathbb{B}, 14, 14, X)$ with the initial objects in it, visiting limit $N$
Output: picture $(\mathbb{B}, 14, 14, S)$ containing the surface skeleton
 1: $DT \leftarrow computeDT(X)$
 2: $H \leftarrow \{p \in X \mid p$ is a local maximum$\}$
 3: $S \leftarrow \emptyset$
    // Thinning
 4: repeat
 5:   $L \leftarrow \{p \in X \setminus (S \cup H) \mid p$ is a strong border point$\}$
 6:   $D \leftarrow \{p \in L \mid p$ is simple for $X$ and $p$ is not a surface edge point$\}$
 7:   $t \leftarrow 0$
 8:   repeat
 9:     $t \leftarrow t + 1$
10:     for each $p \in D$ in lexicographical order do
11:       if $p$ is simple then
12:         $X \leftarrow X \setminus \{p\}$
13:         $L \leftarrow L \setminus \{p\}$
14:       end if
15:     end for
16:   until $t = N$ or no points are deleted
17:   $S \leftarrow S \cup L$
18: until $D = \emptyset$
```

Theorem 3. *The runtime complexity of Algorithm 2 and Algorithm 3 is linear, if $N \in \mathbb{N}$.*

Proof. As we mentioned in Theorem 2, d_{14} distance mapping has linear computational cost. Here we note that d_8 and any $\langle a, b\rangle$ Chamfer distance have the same property [7]. Moreover, there are also linear time adaptations of the Euclidean distance transform on the BCC grid [20].

Following the indication in Theorem 2, $|I|$ denotes the number of points in the image. During the thinning phase, each object point is visited in exactly one iteration. In Algorithm 2, this is ensured by collecting all previously investigated but not deleted border points. In Algorithm 3, the distance-ordered strategy guarantees this property. It is also obvious that each border point is visited up to N times during one thinning iteration. As a consequence, all object points are visited maximum N times during the thinning phase. Hence, the computational cost is $O(N \cdot |I|)$, which means linear time complexity if N is a positive integer since it is a fixed parameter. □

Algorithm 3 Distance-ordered variant *DO-N-visits* to extract surface skeleton.

Input: picture $(\mathbb{B}, 14, 14, X)$ with the initial objects in it, visiting limit N
Output: picture $(\mathbb{B}, 14, 14, X)$ containing the surface skeleton

```
1:  $DT \leftarrow computeDT(X)$
    // Thinning
2:  for $k \leftarrow 1$ to $max(DT)$ do
3:    $D \leftarrow \{p \in X \mid DT(p) = k$ and p is simple and $p$ is not a surface edge point$\}$
4:    $t \leftarrow 0$
5:    repeat
6:      $t \leftarrow t + 1$
7:      for each $p \in D$ in lexicographical order do
8:        if $p$ is simple then
9:          $X \leftarrow X \setminus \{p\}$
10:       end if
11:     end for
12:   until $t = N$ or no points are deleted
13: end for
```

The extracted surface skeletons of the holey cube by our improved algorithms are shown in Figure 5. It is easy to see that these surface skeletons contain less insignificant skeleton points compared to the result in Figure 4.

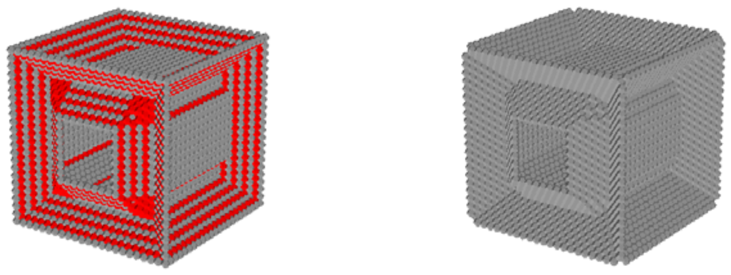


Figure 5: Result of our improved algorithms on a holey cube shown in Figure 4 on d_{14} distance map with $N = 2$. In the left figure anchor voxels are gray and further skeleton voxels are red. Numbers in parentheses indicate the number of skeleton points.

5 Modified variant to produce curve skeletons

By modifying our distance-ordered variant, we can also extract the curve skeleton directly from the original object. For this purpose, instead of surface edge points we retain either of two types of curve endpoints. Point $p \in B$ is a *C1 endpoint*, if $|N^*_{14}(p) \cap B| = 1$, i.e., p has only one black neighbor in $N^*_{14}(p)$. Let u be this black neighbor. Point p is a *C2 endpoint*, if p is a *C1* endpoint and u is a line point. Point u is a *line point* if there are exactly 2 black points in its 14-neighborhood and they are not 14-adjacent to each other. We consider a border point deletable if it is simple but not an endpoint. The corresponding algorithm is called *DO-CS*. We reorganised the thinning iterations: the non-deleted black points must be visited again during the further iterations in order to shrink the surface patches until line branches are only left. Parameter N is not used since the sufficient number of iterations repetition depends on the structure of the objects in the input picture. This algorithm terminates only if there are no more deletable points. Therefore, it is impossible to set a constant upper limit k to the visiting number of object points. As a result, the extraction of the curve skeleton has nonlinear time complexity.

It is important to note that anchor point condition should not be used because the detected ridge points belong to the surface skeleton, so surface segments will probably be also generated.

According to our experiments, visiting elements of D in lexicographical order has a relevant effect only on d_8 and d_{14} distance maps. The reason behind this phenomenon is the fact that on Chamfer or Euclidean distance maps the number of considered border points in a thinning iteration are less than in d_{14} or d_8 distance map, so there is a lower chance of unfavorable visiting order. Furthermore, there are much more configurations for surface edge points than for line endpoints. Hence, the curve endpoint criteria are less sensitive to the visiting order, especially the $C2$ condition.

Extracted surface skeletons of the holey cube are shown in Figure 6. Notice that d_8 skeleton is much more jagged due to the unlucky visiting order of border points.

6 Results

Our algorithms were tested on numerous objects of different shapes. Figures 7–10 show the surface skeletons and 11–13 show the curve skeletons. To make the difference between the applied parameters more visible, some of the following figures consist of fusioned images, where red, green and gray voxels belong to the first, second and both skeletons extracted with the indicated parameters, respectively. In the case of Strand's method the gray and red points mean the anchor points and further skeletal points, respectively. In case of any $\langle a, b\rangle$ weighted distances, local maxima are considered as anchor points instead of CMB's even for Strand's method. Numbers in parentheses show the number of object or skeleton points.

Algorithm 4 Variant *DO-CS* to extract the curve skeleton.

```
Input: picture $(\mathbb{B}, 14, 14, X)$ with the initial objects in it, line endpoint criterion
    $Ci$ $(i \in \{1, 2\})$
Output: picture $(\mathbb{B}, 14, 14, X)$ containing the curve skeleton
 1: $DT \leftarrow computeDT(X)$
    // Thinning
 2: for $k \leftarrow 1$ to $max(DT)$ do
 3:   repeat
 4:     $D \leftarrow \{p \in X \mid DT(p) \leq k$ and $p$ is simple and $p$ is not a $Ci$ endpoint$\}$
 5:     repeat
 6:       for each $p \in D$ do
 7:         if $p$ is simple then
 8:           $X \leftarrow X \setminus \{p\}$
 9:         end if
10:       end for
11:     until no points are deleted
12:   until $D = \emptyset$
13: end for
```

Figure 6: Produced curve skeletons of a holey cube shown in Figure 4 with different distances. The resulting curve skeletons with C1- and C2-endpoint condition coincide with each other because no C2-endpoint was detected. Numbers in parentheses indicate the number of skeleton points.

As we discussed in Section 3, Strand's algorithm leaves many insignificant skeleton segments in case of d_{14} distance. It's easy to see that Strand's method remarkably overshrinks the input object, when weighted distance is used. Also note that the more accurate approximation of the Euclidean distance is used, the less anchor points are detected. In the case of $N = 1$, algorithm *DO-N-visits* leaves one side of the letter A almost unchanged due to the unfavorable visiting order of border points. This phenomenon is successfully handled with setting N to 2. However, the modified versions may overshrink the object, if N is too large, especially in d_8 or d_{14} distance maps (see Figure 8). Hence, options $N \in \{2, 3\}$ are usually sufficient.

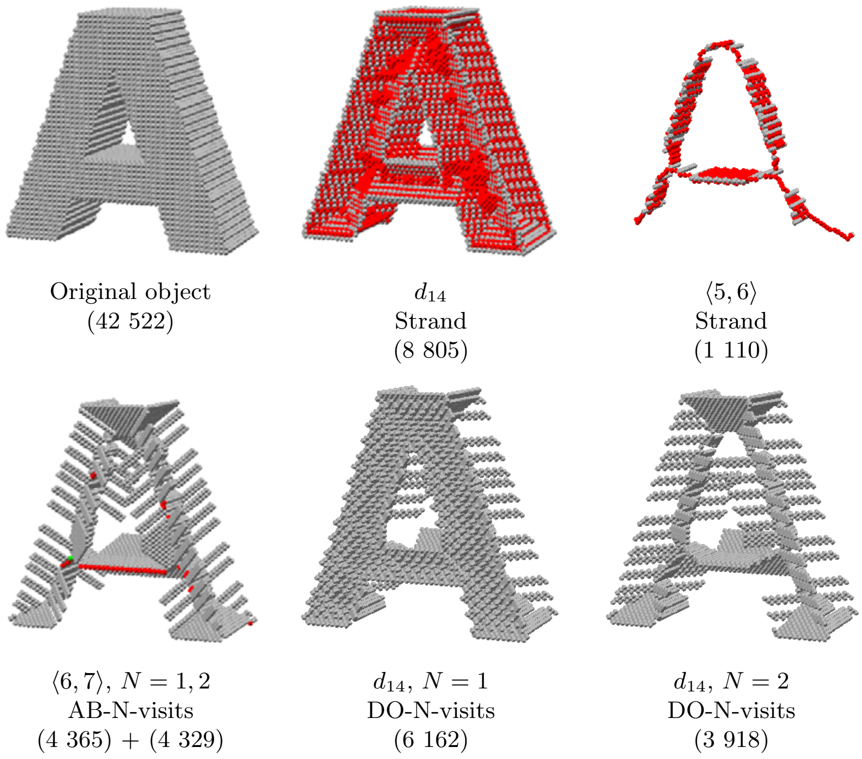


Figure 7: Produced surface skeletons of the letter A with various parameters.

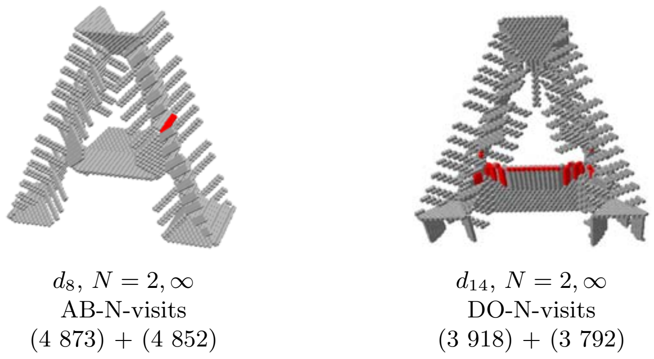


Figure 8: Overshrinked surface skeletons of the letter A. For $N = \infty$, the highest repetition number, i.e. the highest value of t in Algorithm 2 and Algorithm 3 was 14 and 12, respectively.

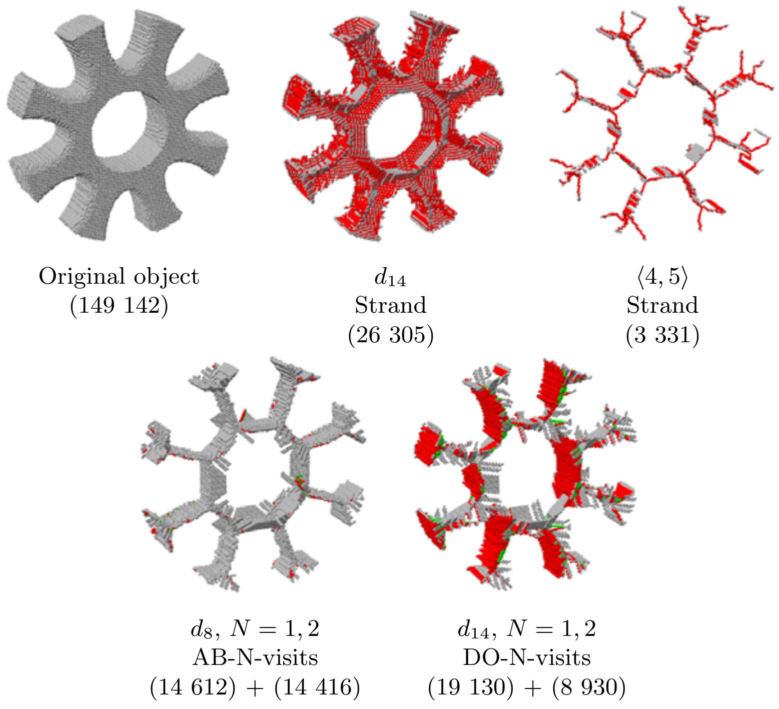


Figure 9: Produced surface skeletons of a gear with various parameters.

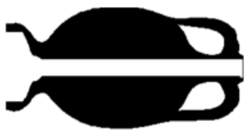
We can also observe that the medial surface is strongly jagged due to the nature of sequential thinning.

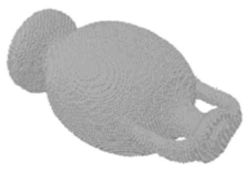
A similar phenomenon can be observed on the gear and the amphora. By setting N to 2 we get a much cleaner skeleton. Note that the number and distribution of skeletal points also depend on the thinning strategy.

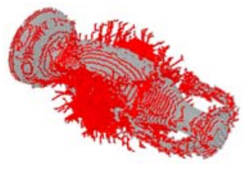
It can be well observed that significantly less false line segments were produced on the curve skeletons with condition $C2$ compared to $C1$.

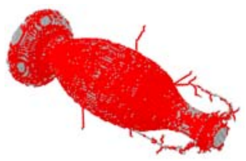
7 Conclusions

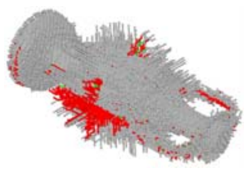
The proposed algorithms *AB-N-visits* and *DO-N-visits* have linear runtime complexity and are less sensitive to the visiting order of border points compared to Strand's method. According to our experiments, it is sufficient to set N to at most 3 to produce "clear" surface skeletons. All examined algorithms preserve topology due to the fact that only a single simple point is deleted at a time. All the proposed methods can be extended to arbitrary distances including any $\langle a, b\rangle$ weighted distance or even the Euclidean distance. The optimal parameters (e.g. distance, thinning strategy) depend on the shapes of the objects to be represented. We note that, though numerous authors have proposed methods to evaluate the performance


Cross-sectional image


Original object
(566 765)


d_{14}
Strand
(51 901)


$\langle 3,4\rangle$
Strand
(25 892)


d_{14}, $N = 1,2$
AB-N-visits
(53 522) + (51 493)

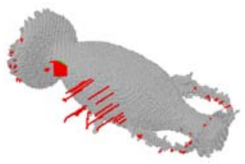
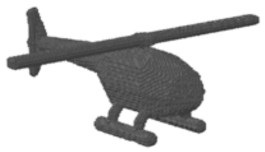
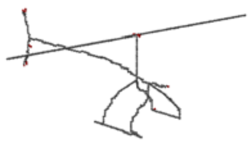

d_e, $N = 1,2$
DO-N-visits
(29 749) + (29 246)

Figure 10: Produced surface skeletons of an amphora with various parameters.


Original object
(94 031)


d_8
C1 + C2
(539) + (524)

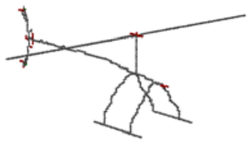
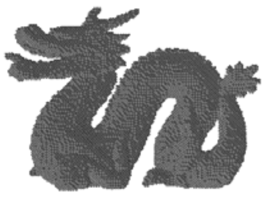
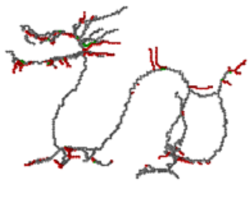

$\langle 3,4\rangle$
C1 + C2
(576) + (537)

Figure 11: Extracted curve skeletons of a helicopter with various parameters.


Original object
(105 933)


$\langle 4,5\rangle$
C1 + C2
(1 131) + (861)

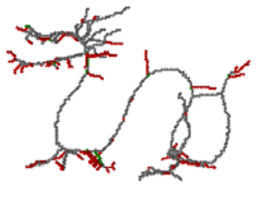
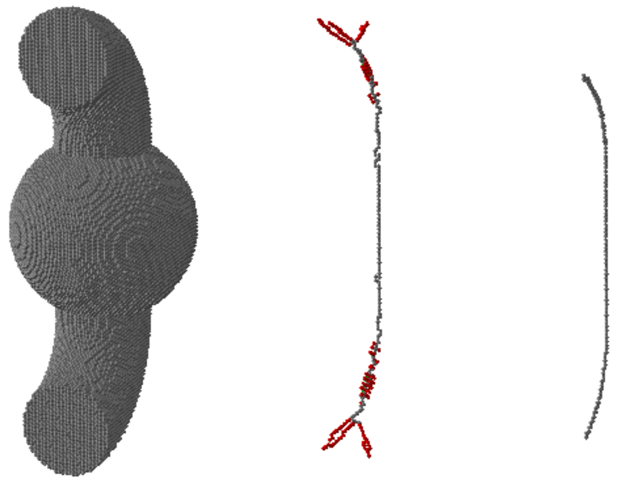

d_e
C1 + C2
(1 153) + (818)

Figure 12: Extracted curve skeletons of a dragon with various parameters.



Original object (212 639) | d_{14} C1 + C2 (283) + (145) | d_e C1 + C2 (121) + (121)

Figure 13: Extracted curve skeletons of an object with various parameters. In the right figure, the C1- and C2-skeletons are coincide with each other since no C2-endpoint was detected.

of skeletonization algorithms (see for example [6, 10, 18, 25]), due to the lack of definition of the "true skeleton" for a discrete object, a widely accepted approach evaluating the goodness of skeletonization algorithms is yet absent [16].

Future research should be devoted to constructing similar distance-based algorithms combined with parallel thinning strategies and adapting the presented results to the face-centered cubic grid [21].

References

[1] Ankudinov, V. and Galenko, P. K. Growth of different faces in a body centered cubic lattice: A case of the phase-field-crystal modeling. *Journal of Crystal Growth*, 539:125608, 2020. DOI: `10.1016/j.jcrysgro.2020.125608`.

[2] Arcelli, C. and Sanniti di Baja, G. Finding local maxima in a pseudo-Euclidian distance transform. *Computer Vision, Graphics, and Image Processing*, 43(3):361–367, 1988. DOI: `10.1016/0734-189X(88)90089-8`.

[3] Arcelli, C. and Sanniti di Baja, G. Ridge points in Euclidean distance maps. *Pattern Recognition Letters*, 13(4):237–243, 1992. DOI: `10.1016/0167-8655(92)90074-A`.

[4] Borgefors, G. Distance transformations in arbitrary dimensions. *Computer Vision, Graphics, and Image Processing*, 27(3):321–345, 1984. DOI: `10.1016/0734-189X(84)90035-5`.

[5] Borgefors, G., Nyström, I., and Sanniti di Baja, G. *Discrete Skeletons from Distance Transforms in 2D and 3D.* In *Medial Representations: Mathematics, Algorithms and Applications*, pages 155–190. Springer Netherlands, 2008. DOI: `10.1007/978-1-4020-8658-8_5`.

[6] Cornea, N. D., Silver, D., and Min, P. Curve-Skeleton Properties, Applications, and Algorithms. *IEEE Transactions on Visualization and Computer Graphics*, 13(3):530–548, 2007. DOI: `10.1109/TVCG.2007.1002`.

[7] Fouard, C., Strand, R., and Borgefors, G. Weighted distance transforms generalized to modules and their computation on point lattices. *Pattern Recognition*, 40(9):2453–2474, 2007. DOI: `10.1016/j.patcog.2007.01.001`.

[8] Kong, T.Y. On topology preservation in 2-d and 3-d thinning. *International Journal of Pattern Recognition and Artificial Intelligence*, 09(05):813–844, 1995. DOI: `10.1142/S0218001495000341`.

[9] Kong, T.Y. and Rosenfeld, A. Digital topology: Introduction and survey. *Computer Vision, Graphics, and Image Processing*, 48(3):357–393, 1989. DOI: `10.1016/0734-189X(89)90147-3`.

[10] Kruszynski, K. J., Liere van, R., and Kaandorp, J. A. Quantifying differences in skeletonization algorithms: a case study. In Villanueva, J.J., editor, *Visualization, Imaging, and Image Processing (Proceedings 5th IAESTED International Conference, VIIP'05, Benidorm, Spain, September 7–9, 2005)*, Canada, 2005. ACTA Press.

[11] Lam, L., Lee, S.-W., and Suen, C. Y. Thinning methodologies - a comprehensive survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(9):869–885, 1992. DOI: `10.1109/34.161346`.

[12] Marchand-Maillet, S. and Sharaiha, Y. M. *Binary Digital Image Processing: A Discrete Approach.* Academic Press, 2000. DOI: `10.1016/B978-0-12-470505-0.X5000-X`.

[13] Mujahed, H. and Nagy, B. Wiener Index on Lines of Unit Cells of the Body-Centered Cubic Grid. In Benediktsson, Jón Atli, Chanussot, Jocelyn, Najman, Laurent, and Talbot, Hugues, editors, *Mathematical Morphology and Its Applications to Signal and Image Processing*, pages 597–606, Cham, 2015. Springer International Publishing. DOI: `10.1007/978-3-319-18720-4_50`.

[14] Čomić, L. and Magillo, P. Repairing 3D binary images using the BCC grid with a 4-valued combinatorial coordinate system. *Information Sciences*, 499:47–61, 2019. DOI: `10.1016/j.ins.2018.02.049`.

[15] Čomić, L. and Nagy, B. A combinatorial coordinate system for the body-centered cubic grid. *Graphical Models*, 87:11–22, 2016. DOI: `10.1016/j.gmod.2016.08.001`.

[16] Saha, P. K., Borgefors, G., and Sanniti di Baja, G. A survey on skeletonization algorithms and their applications. *Pattern Recognition Letters*, 76(1):3–12, 2016. DOI: `10.1016/j.patrec.2015.04.006`, Special Issue on Skeletonization and its Application.

[17] Saha, P. K., Borgefors, G., and Sanniti di Baja, G. *Skeletonization: Theory, Methods and Applications*. Academic Press, 2017.

[18] Saha, P. K., Chaudhuri, B. B., and Majumder, D. D. A new shape preserving parallel thinning algorithm for 3d digital images. *Pattern Recognition*, 30(12):1939–1955, 1997. DOI: `10.1016/S0031-3203(97)00016-2`.

[19] Strand, R. Surface skeletons in grids with non-cubic voxels. In *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.*, pages 548–551, Cambridge, 2004. DOI: `10.1109/ICPR.2004.1334195`.

[20] Strand, R. The Euclidean Distance Transform Applied to the FCC and BCC Grids. In Marques, J. S., de la Blanca, N. P., and Pina, P., editors, *Pattern Recognition and Image Analysis*, pages 243–250, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. DOI: `10.1007/11492429_30`.

[21] Strand, R. The face-centered cubic grid and the body-centered cubic grid: a literature survey. Technical Report 35, Centre for Image Analysis, Uppsala University, Uppsala, Sweden, 2005.

[22] Strand, R. and Borgefors, G. Distance transforms for three-dimensional grids with non-cubic voxels. *Computer Vision and Image Understanding*, 100(3):294–311, 2005. DOI: `10.1016/j.cviu.2005.04.006`.

[23] Strand, R. and Brunner, D. Simple Points on the Body-Centered Cubic Grid. Technical Report 42, Centre for Image Analysis, Uppsala University, Uppsala, Sweden, 2006.

[24] Strand, R., Nagy, B., and Borgefors, G. Digital distance functions on three-dimensional grids. *Theoretical Computer Science*, 412(15):1350–1363, 2011. DOI: `10.1016/j.tcs.2010.10.027`.

[25] Tagliasacchi, A., Delame, T., Spagnuolo, M., Amenta, N., and Telea, A. 3D Skeletons: A State-of-the-Art Report. *Computer Graphics Forum*, 35(2):573–597, 2016. DOI: `10.1111/cgf.12865`.

Acta Cybernetica **25** (2021) 369–400.

Rootkit Detection on Embedded IoT Devices*

Roland Nagy[ab], Krisztián Németh[ac], Dorottya Papp[ad], and Levente Buttyán[ae]

Abstract

IoT systems are subject to cyber attacks, including infecting embedded IoT devices with rootkits. Rootkits are malicious software that typically run with elevated privileges, which makes their detection challenging. In this paper, we address this challenge: we propose a rootkit detection approach for embedded IoT devices that takes advantage of a trusted execution environment (TEE), which is often supported on popular IoT platforms, such as ARM based embedded boards. The TEE provides an isolated environment for our rootkit detection algorithms, and prevents the rootkit from interfering with their execution even if the rootkit has root privileges on the untrusted part of the IoT device. Our rootkit detection algorithms identify modifications made by the rootkit to the code of the operating system kernel, to system programs, and to data influencing the control flow (e.g., hooking system calls), as well as inconsistencies created by the rootkit in certain kernel data structures (e.g., those responsible to handle process related information). We also propose algorithms to detect rootkit components in the persistent storage of the device. Besides describing our approach and algorithms in details, we also report on a prototype implementation and on the evaluation of our design and implementation, which is based on testing our prototype with rootkits that we developed for this purpose.

Keywords: embedded systems, Internet of Things, security, malware

1 Introduction

The Internet has grown beyond a network of laptops, PCs, and large servers: it also connects millions of small embedded devices. This new trend is called the

*The presented work was carried out within the SETIT Project (2018-1.2.1-NKP-2018-00004), which has been implemented with the support provided from the National Research, Development and Innovation Fund of Hungary, financed under the 2018-1.2.1-NKP funding scheme.

[a]CrySyS Lab, Department of Networked Systems and Services, Budapest University of Technology and Economics, Hungary

[b]ORCID: `0000-0003-2305-3271`

[c]ORCID: `0000-0001-6677-6862`

[d]ORCID: `0000-0002-9976-614X`

[e]ORCID: `0000-0003-4233-2559`

DOI: `10.14232/actacyb.288834`

Internet of Things, or IoT in short, and it enables many new and exciting applications such as smart homes, intelligent transportation systems, smart factories, and personalized healthcare. At the same time, IoT also comes with a number of risks related to information security. The lack of security, however, cannot be tolerated in certain applications of IoT, including those in the domains of healthcare, transportation, and industrial automation. In such applications, security failures may lead to substantial monetary loss, physical damage of expensive equipment, or even loss of human life. Therefore, one of the biggest challenges today, which hinders the application of IoT technologies in many cases, is the lack of security guarantees.

Unfortunately, IoT systems are notoriously insecure. One of the reasons is that they are built from cheap embedded devices that are easy to compromise by exploiting weaknesses in the way they are operated and vulnerabilities of the software components running on them. A consequence of this is that malware for IoT devices has appeared [1, 16] and gaining momentum [25]. Malware designed for IoT devices is similar to malware designed for other types of computers: it compromises the integrity of the device by installing unwanted, and potentially harmful, software components on it. These software components can then be used to cause other types of compromise such as allowing the attacker to access the device remotely by installing a backdoor, tampering with messages sent by the device to other devices, or making data stored on the device unavailable by deleting or encrypting them.

Sophisticated malware tries to maintain its presence on infected devices while remaining invisible for the operators of those devices. This sort of malware is called *rootkit* [5]. Typically, rootkits run with elevated (root) privileges and they modify system commands and/or code and various data structures in the operating system (OS) kernel such that their files and running processes do not appear in the output of various system tools used to monitor the operation of the devices. Detecting such a rootkit is challenging, mainly because any detection program running at the same or lower privilege levels than the rootkit may also be compromised or may be misled by the tricks used by the rootkit to hide itself.

In this work, we aim at rootkit detection on embedded IoT devices, and we address the above challenge by running our rootkit detection mechanisms in a Trusted Execution Environment (TEE), which is isolated from the main OS of the device, and hence the rootkit – even running with root privileges on the main OS – cannot interfere with its operation. Such a TEE is supported on many embedded platforms, including the popular ARM platform that supports the establishment of TEEs by its TrustZone[1] technology. A TustZone enabled ARM processor can execute in two modes: in untrusted mode (called Normal World), it runs a common OS (e.g., Linux) and applications on top of it (often referred to as the Rich Execution Environment, or REE for short), whereas in trusted mode (called Secure World), it runs the trusted OS and the trusted applications of the TEE. Isolation

[1]`https://developer.arm.com/ip-products/security-ip/trustzone`, Last visited: Sep 20, 2020

between these two modes are ensured by hardware based protection mechanisms. As a result, software components running in the Normal World cannot access some of the resources (including a part of the system memory) of the device, whereas trusted software components of the TEE running in the Secure World have unlimited access to all resources. Thus, the TEE provides two advantages for rootkit detection: it can protect the integrity of some trusted rootkit detection code by keeping it inaccessible for potentially malicious software running in the REE, and it can provide a safe execution environment for that rootkit detection code where it can access and inspect all system resources.

However, as the same processor is shared between the trusted and the untrusted mode, the execution of untrusted software is suspended when the processor switches to trusted mode and starts executing trusted software. This means that our trusted rootkit detection code cannot observe the behavior of untrusted software components during their execution, but it can only inspect their memory images reflecting their state at the time of their suspension. In other words, our rootkit detection approach is based on analyzing memory snapshots of the untrusted system (OS and applications), and it consists of identifying the anomalies caused by the rootkit in the state of the kernel data structures representing processes, as well as computing the hash values of the memory images of running processes and comparing them to known good hash values stored safely in the TEE. In addition, as the rootkit may delete all its components from the memory before our rootkit detection code is invoked and save itself to persistent storage for later execution, we also scan and hash files stored on the flash disk of the device from within the TEE, and compare the computed hash values to known good hashes stored safely in the TEE.

The rest of the paper is organized as follows: In Section 2, we introduce the main features of rootkits and explain why they are difficult to detect. In Section 3, we give an overview of our rootkit detection approach, which is based on checking the system memory and the persistent storage from the TEE. These two components of our approach are described in more details in Sections 4 and 5, respectively. We implemented our approach using OP-TEE[2], an open source TEE implementation, and Section 6 contains the details on this implementation effort. In Section 7, we report on the evaluation of our rootkit detection mechanisms, which we performed with custom rootkits that we developed for this specific purpose. Finally, we discuss some limitations of the proposed rootkit detection approach and possible future extensions in Section 8, and conclude the paper in Section 9.

2 Rootkits

The term *rootkit* [21] refers to malware or modules of pieces of malware whose primary goal is to maintain stealth on infected devices while allowing continued access to its resources. Remaining hidden is in the best interests of the attackers: if the operators detect the infection, they will try to eliminate the responsible pieces

[2]`https://www.op-tee.org`, Last visited: Sep 20, 2020

of malware by reinstalling the system, and they will also try to patch the exploited vulnerability rendering the system unavailable for the attackers.

Rootkits employ a wide variety of techniques to remain hidden on infected devices [7, 21, 22]. *User-mode* techniques target the user space of devices and include techniques such as the manipulation of log files, modification of disk-resident system files (e.g., `ls`, `top`) or hooking libraries used by executables. One well-known user space hooking technique is the abuse of the `LD_PRELOAD` environment variable to load malicious libraries before benign ones, therefore overriding their provided functionalities. This technique is used by rootkits such as Azazel[3], Jynx2 [8], BEURK[4], vlany[5] and bedevil[6]. Other rootkits, such as HORSEPILL [19], abuse valid kernel features offered to user space programs to maintain stealth presence on the device.

Kernel-mode techniques, on the other hand, target the internal data structures and functionalities in the operating system's kernel. In the past, modifying the kernel memory image was a widely used technique to remain hidden on Linux systems. However, recent Linux kernel versions restrict access to `/dev/mem` and `/dev/kmem`[7], mitigating this attack. Other techniques in this category are kernel-space hooking [28] and direct kernel object manipulation. The former includes the use of malicious device drivers and the modification of the system call and interrupt descriptor table. Direct kernel object manipulation tampers with the integrity of the kernel by targeting dynamic kernel data structures. This technique can be used, for example, to hide processes from the system administrator. There exist a number of rootkits employing these techniques, including Adore-NG[8], LilyOfTheValley[9], the work presented in [14], OSOM [11], SucKIT [9] and Suterusu[10].

Many proof-of-concept rootkits compromise the virtualization layer, the BIOS and/or the firmware of hardware components [10, 15, 17, 18]. Such rootkits are called *OS-independent* rootkits and their advantages are manifold. Rootkits in the lowest-level components can survive reboots and re-installations, and they leave no traces on the disk. Their detection is particularly challenging because they do not make visible changes to the operating system.

In response to the rising threat of rootkits, a number of detection methods have been proposed [24]. *Signature-based* methods scan the files on the disk for byte sequences and use a signature database to detect known rootkits. Tools that implement this method include chkrootkit[11] and Rootkit Hunter[12]. The main lim-

[3]`https://github.com/chokepoint/azazel`, Last visited: Sep 16, 2020
[4]`https://github.com/unix-thrust/beurk`, Last visited: Sep 16, 2020
[5]`https://github.com/mempodippy/vlany`, Last visited: Sep 16, 2020
[6]`https://github.com/wiperpaul/bdvl`, Last visited: Sep 16, 2020
[7]`https://lore.kernel.org/lkml/18778.1508769258@warthog.procyon.org.uk/`, Last visited: Sep 16, 2020
[8]`https://github.com/yaoyumeng/adore-ng`, Last visited: Sep 16, 2020
[9]`https://github.com/En14c/LilyOfTheValley`, Last visited: Sep 16, 2020
[10]`https://poppopret.org/2013/01/07/suterusu-rootkit-inline-kernel-function-hooking-on-x86-and-arm/, Last visited: Sep 16, 2020`
[11]`http://www.chkrootkit.org/`, Last visited: Sep 17, 2020
[12]`http://rkhunter.sourceforge.net/`, Last visited: Sep 17, 2020

itation of signature-based methods is similar to those of signature-based intrusion detection systems and virus scanners, namely, that they cannot detect very recent or sufficiently modified old rootkits.

Behavior-based detection methods detect deviations from “normal” patterns of system behavior [12], e.g. timing discrepancies and irregularities in resources such as the Translation Lookaside Buffer (TLB). Such methods, however, require *a priori* measurements of the analyzed system in a controlled environment. Differences between the real and the controlled environment can decrease the accuracy of such approaches. The baseline measurements must also be stored securely on the device, otherwise, malware can influence the detection method.

Cross-view-based methods assume that there is no perfect rootkit which can perfectly emulate all aspects of the system. In order to detect compromises, they enumerate system parameters in at least two different ways and compare the results. However, the kernel consists of many dynamically changing structures: a change in an enumerated structure during cross-checking can lead to false alarms. To overcome this challenge, Carbonite[13] preempts scheduling, thereby prohibiting new processes to spawn. However, such an approach has an impact on performance.

Integrity-based detection methods compare a snapshot with a trusted baseline. In the case of files, the trusted baseline can be the hash value of a file computed in a controlled environment, which can be checked with tools such as Samhain[14]. Kernel data structures can also be monitored for malicious changes, e.g. system call re-maps can be detected using StMichael[15], running processes can be listed with KSTAT[16], and Gibraltar [3] uses a set of automatically derived data structure invariants for monitoring purposes. The main challenge of integrity-based detection is the secure storage of the baseline: if the kernel is assumed to be compromised, then no file or memory on the system is adequate for storage. Rootkits in the kernel can modify the return value of system calls necessary for reading files and may compromise the Virtual Memory Management unit of the operating system to access and/or tamper with data stored in memory.

Reliable detection of rootkits requires that the detector runs with higher privileges than the rootkit itself; as a result, detectors are placed in ever lower levels in the devices' architecture or even into a separate hardware. Paladin [2] is an example of the former approach. It defines protected zones for memory and files, and performs integrity checks from the hypervisor. Copilot [20], on the other hand, is an example of the latter: it is a coprocessor-based kernel integrity monitor implemented as a PCI card which connects the monitored system to the remote detector.

Our proposed method incorporates ideas from cross-view-based and integrity-based detection methods, including integrity checks on files similarly to other file integrity checkers [27], and anomaly detection in kernel data structures similarly to Strider GhostBuster [26]. We provide a more in-depth discussion of our detection methods in Section 3. Unlike other approaches, however, our proposed method

[13]`https://securiteam.com/tools/5jp0m1f40e/`, Last visited: Sep 21, 2020
[14]`https://www.la-samhna.de/samhain/`, Last visited: Sep 17, 2020
[15]`https://sourceforge.net/projects/stjude/`, Last visited: Sep 21, 2020
[16]`http://www.s0ftpj.org/docs/lkm.htm`, Last visited: Sep 17, 2020

does not need additional hardware: we leverage a TEE to prevent the rootkit from interfering with our rootkit detection process. We also use the TEE to safely store baseline values (e.g., file hashes and hash values of memory images).

3 Overview of our approach

As mentioned previously, the basic idea of our rootkit detection approach is to leverage the TEE (Trusted Execution Environment) for running functions aiming at detecting integrity violations and inconsistencies in the state of the untrusted system components of the REE (Rich Execution Environment) that may have been caused by a rootkit. The primary targets for checking integrity and consistency are the kernel code and the kernel data structures (in particular, data structures representing processes, as well as data structures holding function pointers) of the untrusted OS, as rootkits typically modify those to achieve their goals. The kernel code and data structures can be accessed from the TEE by reading the memory snapshot of the REE that is left behind when execution is transferred to our rootkit detection function in the TEE. In addition, besides checking code and data structures in the memory, we must be prepared for malware that tries to remove its traces from memory before the invocation of our rootkit detection function. Because it cannot remain in memory, it may try to hide itself in persistent storage in such a way that it can execute again later when the memory has been checked. Hence, our rootkit detection approach also includes checking the persistent storage for signs of malware.

To achieve our goals, we deploy two software components: a Trusted Application (TA) running in the TEE and a client application (CA) running as a user space process on top of the untrusted OS in the REE. Our CA should be started when the system is booted and then it should run continuously. The main role of the CA is to invoke the TA periodically and to pass certain data to the TA collected from the REE (e.g., the list of running processes as seen by the `ps` program when executed on the untrusted OS). The TA performs rootkit detection by executing different integrity and consistency verification functions as described below. In order to ensure that the TA is indeed invoked periodically, a watchdog timer can be started during the boot process that can only be reset by the TA; therefore, if the TA is not invoked, the timer expires and the device reboots itself. When the TA finishes its execution, control is returned to the CA, which runs concurrently with other applications and services in the REE.

In the sequel, we assume that the OS running in the REE is Linux. Some of our rootkit detection functions described below are specific to Linux, because rootkits often operate at low level in the system architecture and exploit specific features or mechanisms of the OS kernel. Yet, the principles even behind these Linux specific functions are sufficiently general to be applied for other operating systems as well. Moreover, some of the detection functions we present are agnostic to the OS used.

When invoked, our rootkit detection TA performs the following verification steps aiming at detecting inconsistencies in the data held by certain kernel data structures or modifications of kernel code:

- **Looking for hooks in the Virtual File System (VFS):** Rootkits often target the so called *operation structures* of the VFS and replace (hook) function pointers there such that file operations are handled by attacker code instead of legitimate kernel code. For instance, the rootkit may hook the `lookup` function in the operation structure of the inode of the `/proc` directory, and ensure that certain process IDs are removed from its output, and hence, become invisible to certain system utilities. Thus, in each operation structure of the VFS, we check if function pointers point inside the address space where the kernel code segment is located. Any function pointer pointing outside that address space is considered to be hooked. More details on detecting hooks of the VFS file operations are provided in Subsection 4.1.
- **Detecting hidden tasks:** Another way of hiding certain processes is to manipulate kernel data structures (a.k.a. Direct Kernel Object Manipulation or DKOM for short) representing them. At the kernel level, processes (threads) are represented by tasks, and there are different data structures, such as the task list, the task tree, and so called PID namespaces that hold information about existing tasks. In addition, tasks also appear in queues used by the kernel for scheduling them. Rootkits rarely modify these data structures in a consistent manner. For instance, in order to hide itself, a rootkit may remove its task structure from the process list or process tree, while it must keep itself in the run queue to be scheduled and have the chance to be executed on the CPU. Therefore, we check all those data structures that hold information about existing tasks and we compare the list of tasks obtained from them to each other and to the process list received from the CA running in the REE. Any inconsistency among these lists is interpreted as an integrity violation of the system. More details on detecting hidden tasks are provided in Subsection 4.2.
- **Integrity checks:** Besides manipulating task related kernel data structures, rootkits can also modify other important data structures in the kernel, as well as the code of running processes. For instance, a common rootkit technique, called *system call table hooking*, is to replace function pointers in the system call table such that when certain system calls are invoked, attacker code is executed before control is given to the legitimate function that handles those system calls. Another technique, called *inline hooking*, has similar effects, but in this case, the system call handling functions themselves are modified by inserting a jump instruction at the beginning of the function that points to some attacker code. Similarly, the code of any processes in the memory may be modified by the rootkit including the kernel code segment, system programs, and user space applications. For this reason, we perform integrity verification of the system call table, the kernel code segment (which includes the functions that handle system calls), system programs currently executing, and the code segment of our CA running in the untrusted execution

environment. This integrity verification is based on accessing these pieces of data structures and code in the REE memory from our TA, computing the hash values of their memory image, and comparing the computed values to known reference values stored securely within the TEE. These reference values are computed and saved in secure storage provided by the TEE after system installation when the system runs for the first time. More details on the integrity checks we perform are provided in Subsection 4.3.

- **Looking for persistent rootkit components:** Finally, as rootkits may remove their components from memory before our TA is invoked and our consistency and integrity verification is executed, and hide themselves on persistent storage, we must also look for these persistent components. For this, our TA accesses the persistent storage of the device, recursively hashes all files in a pre-selected set of folders, and then compares the computed hash values to known reference values stored securely in the TEE. These reference values are computed and saved in secure storage provided by the TEE after system installation when the system runs for the first time. The folders are pre-selected in such a way that they contain all the binaries and scripts that could legitimately execute on the device. This requires a certain organization of the files in the file system, notably to separate files (including programs) that should not change from those that has variable content (e.g., files used mainly for data storage). However, this is not a serious limitation of our approach, because this kind of separation is also useful for many other reasons related to the maintenance and troubleshooting of the device.

  An issue that we have to consider is that while our TA is waiting for I/O operations (i.e., reading file contents) to complete, control may be given back to the REE. When this happens, pre-scheduled jobs may be executed by the job scheduler (e.g., `cron`). Hence, in theory, a rootkit can hide its persistent component in a program file A and schedule the execution of A before removing itself from memory. Then, program A (and hence the rootkit) could be executed by the job scheduler during the file hashing operation performed by our TA, and when executing, the rootkit can move itself from program file A into program file B. If this move operation happens after file B has been hashed already and before file A being hashed, then the computed hash values would be good, and we would not detect the rootkit, which can then be reinstalled when file B is executed. As I/O operations are usually slow, our TA is mostly waiting during the file hashing, which means that control is mainly at the untrusted execution environment, and hence, chances of the above described scenario happening are not negligible. To cope with this issue, our CA disables all file access operations (including execution of programs) before invoking our TA and re-enables them only after the TA completes its job. More details on this are provided in Section 5.

Figure 1 gives a high level overview of our rootkit detection components (i.e., the CA and the TA), their interaction, and the operations they perform. As it can

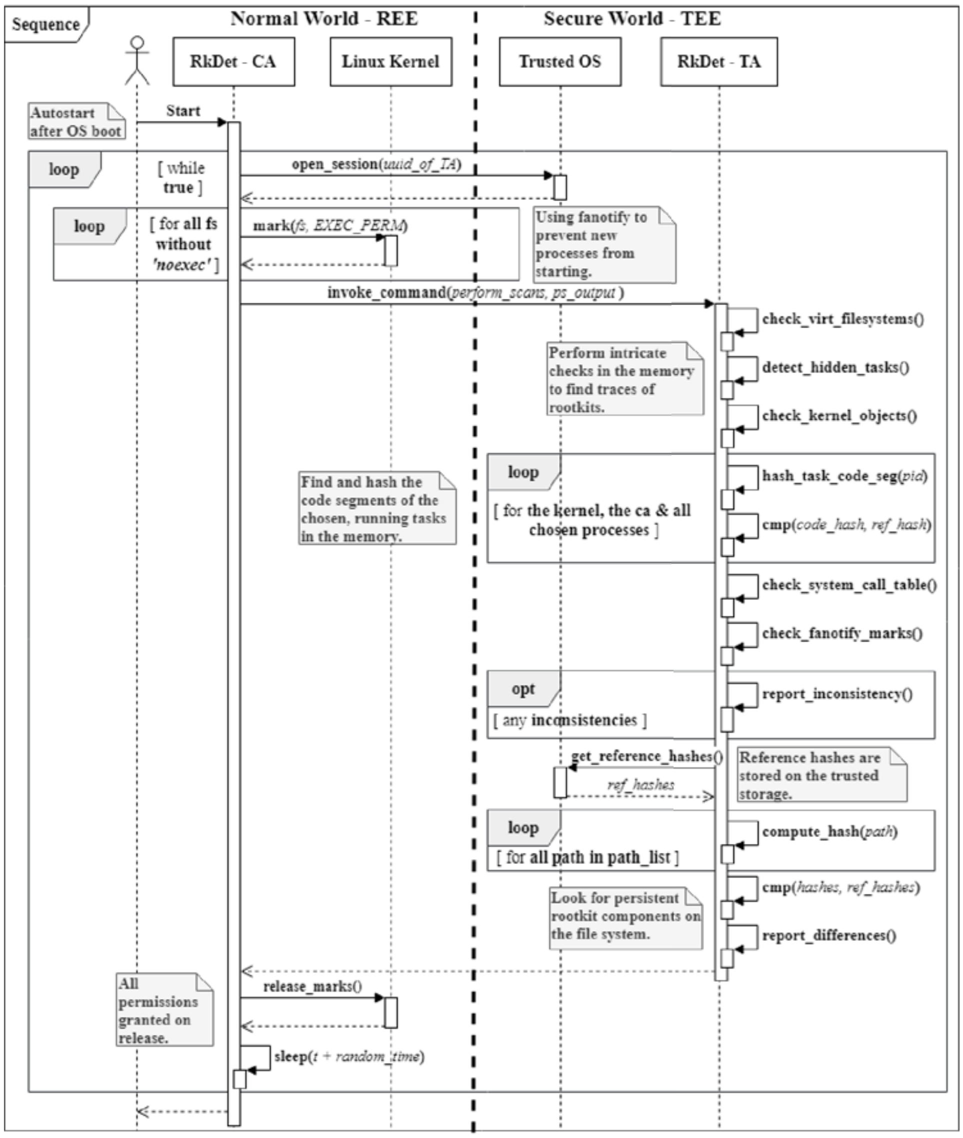

Sequence
Normal World - REE
Secure World - TEE
RkDet - CA
Linux Kernel
Trusted OS
RkDet - TA
Autostart after OS boot
Start
loop
[while true]
open_session(uuid_of_TA)
loop
[for all fs without 'noexec']
mark(fs, EXEC_PERM)
Using fanotify to prevent new processes from starting.
invoke_command(perform_scans, ps_output)
check_virt_filesystems()
Perform intricate checks in the memory to find traces of rootkits.
detect_hidden_tasks()
check_kernel_objects()
loop
[for the kernel, the ca & all chosen processes]
Find and hash the code segments of the chosen, running tasks in the memory.
hash_task_code_seg(pid)
cmp(code_hash, ref_hash)
check_system_call_table()
check_fanotify_marks()
opt
[any inconsistencies]
report_inconsistency()
get_reference_hashes()
ref_hashes
Reference hashes are stored on the trusted storage.
loop
[for all path in path_list]
compute_hash(path)
cmp(hashes, ref_hashes)
Look for persistent rootkit components on the file system.
report_differences()
All permissions granted on release.
release_marks()
sleep(t + random_time)

Figure 1: High level overview of our rootkit detection components, their interactions, and the operations they perform.

be seen, the CA is started at boot time, it continuously runs, and it invokes the TA at random time intervals. Before invoking the TA, the CA disables execution type access to files, such that new programs cannot be started during the checks of the TA. Then the TA performs the above described consistency and integrity checks on the kernel data structures and the code segments of the kernel, running system programs, and our CA. If integrity violations or inconsistencies are found, then they are reported to the operator of the device via some remote attestation protocol, but this is out of the scope of this paper. If the verification of the memory is successful, then the TA proceeds with hashing the files in the pre-selected folders in the persistent storage, and comparing the computed hash values to the stored reference values. Again, if an integrity violation is found, then it is reported. Otherwise, control is returned to the CA, which re-enables file access, and sleeps until the next round of all these operations.

4 Detection of rootkit components in the kernel memory

In this section, we discuss the checks we implemented in order to detect the traces of rootkit infections in the Linux kernel memory. These checks focus on the integrity of and consistency between kernel objects and target common techniques applied by rootkit. In the following subsections, we briefly introduce certain kernel objects, describe attacks against them, and present our methods to determine whether the system is infected by a rootkits. In Subsection 4.1, we present the Linux kernel's Virtual File System (VFS) and our technique to detect rootkit attacks against its structures. In Subsection 4.2, we discuss direct kernel object manipulation (DKOM) in details and our techniques to counter this attack. Finally, in Subsection 4.3, we describe our proposed checks to examine the integrity of the analyzed system.

4.1 Detecting hooks in the Virtual File System

The Virtual File System (VFS) [4] is an API in the Linux kernel which hides the differences of the various file system drivers. It uses 4 main data structures to abstract away the details of different file system implementations, shown in Figure 2. The *superblock* structure represents a mounted partition and stores metadata about the partition itself, which is usually present in the first block of the underlying physical device. Superblocks are chained together into a doubly linked circular list, which is accessible from the data segment of the kernel binary. Each superblock maintains a circular, doubly linked list of the *inodes* stored on the disk. An inode is the physical representation of a file or a directory stored on the device. An inode can be used by one or more directory entries, or *dentries* for short. For example, if we create a new file and a hard link pointing to it, then we will have one inode and two dentires referencing the inode.

Open files are represented by so-called *file* structures in the context of a process. Tampering with these is impractical from the rootkit authors point of view. For example, to modify the way a process reads a file, the hook must be scheduled between the open and the first read call, and this must be performed every time a process opens the file. There are easier and more stable solutions to implement

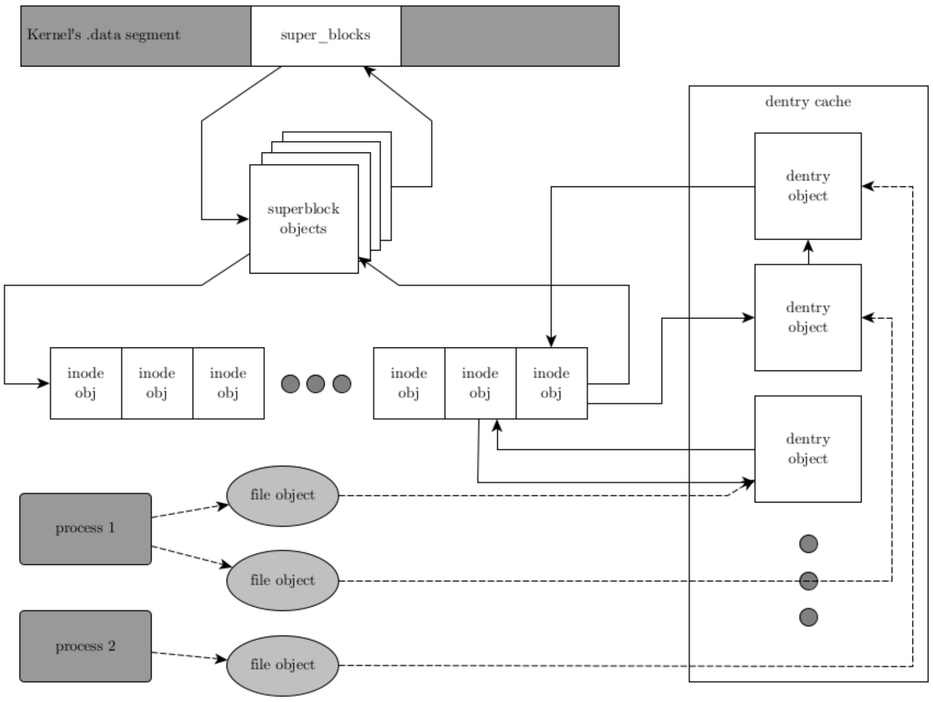


Figure 2: Relationship of VFS objects. White objects are checked, gray ones are ignored by our checks.

this behavior, so we consider files structures out of the scope of our integrity and consistency checks.

At code level, VFS uses so called *operation structures* which contain function pointers. Each function pointer implements a functionality used by the layers above VFS (e.g. `lookup` to retrieve the contained dentries in the inode of a directory or `get_inode_usage` to find out how many inodes are used on a partition), where the implementation is provided by the underlying file system drivers. These operation structures appear in all 4 of the previously mentioned structures.

Rootkits often target the operation structures: by hooking certain function pointers, they can alter the results returned by these functions. For example, an attacker can create a Linux kernel module, find the inode corresponding to the `/proc` directory, and hook the `lookup` member of its `inode_operations`. With a properly designed replacement, attackers can hide their presence by excluding certain process IDs whenever process information is retrieved from `/proc`.

We perform integrity checks on 7 different operation structures. They cover all the operation structures used by superblocks, inodes, and dentries. For superblocks, we analyze `super_operations`, which implement general file system operations (e.g., allocating inodes); `dquot_operations`, which handle quota objects on disk; `quotactl_ops`, which manage quotas on the file system; and `export_operations`, which are operations for the nfs daemon to communicate with file systems. For inodes, we analyze `inode_operations` and `file_operations`. The former provides operations to manage the inode, including `rename`, `unlink`, etc. The latter one contains the file operations assigned to a file structure when a process opens a dentry pointing to this inode. This structure contains function pointers like `read`, `write`, `flush` and many more. For dentries, we check `dentry_operations`, which hold directory entry related operations. For example, `delete` removes the dentry, but the inode remains intact, just in case other dentries use the inode as well.

For each operation structure, we examine the function pointers, and consider one to be hooked if it points outside of the address space where the code segment of the kernel is located. We perform the check recursively, i.e., we inspect every inode of a superblock and every dentry of an inode. Unfortunately, inspecting all superblocks is time consuming, therefore, we only focus on the `/proc` file system as hooking its inode operations is a common technique to hide rootkit processes.

4.2 Detecting hidden tasks

Processes and threads are represented by tasks in the Linux kernel. A task is approximately equivalent to a thread: single-threaded processes consist of a single task, while multi-threaded ones are made up of several tasks sharing the same address space. Each task is represented with the so-called `task_struct` structure. In Linux 5.1, there are three data structures which contain all of the existing tasks.

Task list: All task objects are linked together into a doubly linked circular list. In earlier versions of the kernel, this list was used to populate the `/proc` file system.

Task tree: Tasks are also organized into a tree via the relation of creation. When a task creates other tasks, they become its children and they refer to their creator as their parent. The root of this tree is the so called `init_task`, the kernel task which starts the init process (the first process in the user space).

Pid namespace, IDR and the `struct pid`: Pid stands for process identifier and IDR is the rewritten version of the old ID allocation API. Linux provides pid namespaces as an isolation feature. By default, there is only one such namespace, the initial pid namespace. Each namespace maintains a radix tree[17], containing pointers to pid structures[18] (`struct pid *`). Pid structures contain lists of pointers for the tasks with an equal ID, thread group ID, process group leader or session ID. The Pid namespace is responsible for keeping track of taken pids and for fast access to tasks via their pids[19]. In recent kernel versions, this mechanism populates the `/proc` directory.

[17]`https://lwn.net/Articles/175432/`, Last visited: Mar 19, 2021
[18]`https://lwn.net/Articles/195627/`, Last visited: Mar 19, 2021
[19]`https://lore.kernel.org/patchwork/patch/834401/`, Last visited: Mar 19, 2021

Direct kernel object manipulation (DKOM) is a rootkit technique which modifies the data structures traversed by the kernel when asked to retrieve information about tasks. For example, rootkits can use DKOM to hide a process by removing it from the data structure queried for process information. Many rootkits target the task list to hide a backdoor on older systems; the same can be achieved on modern Linux by removing a pid structure from the IDR of the initial pid namespace. DKOM is rarely applied thoroughly, rootkits usually remove processes only from the necessary data structures, thus leaving the kernel memory in an inconsistent state. For example, the Reptile rootkit[20] hides the kernel module implementing its functionality by removing the module from the list used by the kernel to populate `/proc/modules`. However, it is still visible under `/sys/module`. Another example is the deadlands rootkit[21]: it is capable of removing a process from the list of tasks and a hash table (used by older kernel versions for fast access of tasks by their pids), but the hidden task can still be found in the task tree.

Our analysis attempts to detect these inconsistencies. We assume that a trivial goal of an attacker is to hide a process from the output of the `ps` utility. First, in the REE, we execute `ps` and pass its output to our TA running in the TEE in order to compare it to pid lists we can extract from the REE kernel memory. The pid of `ps` itself must be excluded from this list because it is not running when we perform our check. Our TA, after checking `/proc`, iterates through the task list, saves the pid of every task, and then looks for pids present in the list, but missing from the output of `ps`. If no hidden process is found, the TA performs a depth-first search on the task tree and compares the pid list created in this way with the sample from `ps`. If no inconsistency is found, the TA performs the same check for the IDR. If it does not find any hidden processes, it tries to determine if there is any process present in any of the mentioned data structures, but missing from any of the others. This is done by taking the union of the three lists and comparing each list to the union.

The most important feature of the kernel is the capability to schedule processes. A task is considered to be *runnable*, if it is not waiting for anything (usually I/O) and is not stopped. Runnable tasks ready to be executed are collected in separate data structures; these are called run queues. Run queues are per-CPU data structures, i.e., each CPU has its own run queue, and every run queue wraps subrunqueues implementing the data structures used by the scheduling algorithms. We assume that removing a task from a run queue would make it unschedulable permanently, which is why we include run queues in our consistency check. Our TA collects the pids of all of the tasks found in the data structures of the schedulers and then compares them to the union of pids created earlier. If it finds a task in a run queue which is not present in the union, then the kernel is considered to be compromised.

[20]`https://github.com/f0rb1dd3n/Reptile`, Last visited: Sep 18, 2020
[21]`https://github.com/majdi/deadlands`, Last visited: Sep 18, 2020

4.3 Integrity checks

So far, we focused on revealing hidden processes by verifying the integrity of VFS components and by performing consistency checks on task-related data structures. In this subsection, we leave the concept of hidden processes and instead target common rootkit techniques; some of the checks presented here ensure the integrity of the system itself, while others defend our solution from a possible attacker.

First, we must ensure that our CA can be trusted. To this end, the CA is compiled as a static executable (i.e., its binary contains the code for all the used library functions as well). This allows us to protect it against LD_PRELOAD hooks, a common rootkit technique applied in the user space. The CA passes its own pid as a parameter to our TA, such that the TA can look for the corresponding task in the kernel memory. Each task has a pointer to a memory map structure, which stores information about the task's memory mappings. From this structure, we can determine the start and the end of the task's code segment, and we can use it to translate the virtual addresses of the process to physical addresses. Via physical addresses, we can access the contents of the memory pages storing the code of the task, and we can compute its hash to check whether the code of the CA has changed. The address translation depends on the system's configuration; in our case, the kernel uses 4 Kb pages and 4 layers of translation tables[22].

Next, we check the system call table. System calls are the interface between user space and kernel space. Whenever a user space process needs to perform an operation that is the kernel's responsibility, it invokes the appropriate system call. The system call table is an array of function pointers indexed by the system call number. On the ARM64 architecture, the system call table resides in the `.rodata` section of the kernel binary, which is marked read-only at boot time. If an attacker can remove the write protection, it is possible to overwrite pointers in the array and alter the kernel's control flow to execute a different implementation of the system call. This is the most popular target of kernel space rootkits. [6]

We were able to remove the write protection using the `update_mapping_prot` function[23], changing the last parameter from `PAGE_KERNEL_RO` to `PAGE_KERNEL`. Re-enabling the write protection can be done with the same function.

If kernel space randomization is disabled, we are able to retrieve the location of the system call table after the kernel is compiled. With this information and the number of entries in the table, we can easily determine the memory area to check. We do this by creating a hash and comparing it against the one computed on the intact system call table.

Another common technique is *inline hooking*. In this case, the attacker chooses a function to hook and replaces its first few bytes with an unconditional jump instruction and a pointer to the implementation he wishes to execute instead of the original one. The previously mentioned write protection is applied to the kernel's text segment as well, but similarly to the system call table, it can be removed by

[22]`https://www.kernel.org/doc/html/latest/arm64/memory.html`, Last visited: Mar 19, 2021

[23]`https://elixir.bootlin.com/linux/v5.1/source/arch/arm64/mm/mmu.c\#L525`, Last visited Sep 18, 2020

calling `update_mapping_prot` on the `text` segment[24]. We detect inline hooking, by hashing the kernel's entire text segment and comparing the hash value to a reference value. The location and size of the kernel's text segment is determined after the kernel is compiled. Note that this solution does not support self modifying kernel code and kernel address space layout randomization. The very same approach is used to verify the integrity of the text segment of the system processes and our CA running in the REE.

The Linux kernel provides APIs for monitoring file system operations (e.g., fanotify and inotify, which both use the same underlying kernel mechanism, fsnotify). We use fanotify in our CA to ensure that no executable is started while our TA performs our checks. We do this by placing marks at every mount point to make sure the kernel does not allow execution of any files until the CA approves it. While our checks are performed, the CA denies all execution requests. When the checks are completed, the marks are removed and every execution request is approved.

Since we use another kernel functionality and we assume that the kernel is compromised, our TA needs to check if the fanotify marks placed by our CA are intact. Therefore, the CA passes the file descriptor returned by fanotify and the pid of the CA to the TA. The TA then locates the fsnotify group among the open files using the received pid and the file descriptor. This group stores a list to the marks placed. Using the list of marks, the TA checks if all file system partitions accessible from user-space are indeed marked; if this is not the case, then an attack is detected. In order to ensure that the TA is aware of all available partitions, the CA discovers the partitions and saves their in-memory locations to the TA during the init process, right after secure boot, when the system is unlikely to be already compromised. In addition, we extend the VFS checks described in Subsection 4.1 such that the TA also checks if there exist partitions not saved previously. This is needed, because without checking this, the rootkit could create a new partition, which would not be among the saved ones, and hence, the TA would not check if there is a mark on it. By checking that all existing partitions have been saved before, we detect this potential attack too.

5 Detection of persistent rootkit components

Rootkits may try to evade our detection mechanisms discussed in Section 4 by ceasing all malicious activities, restoring kernel objects to their clean states and hiding on the file system using persistence mechanisms. Interested readers may find an in-depth discussion of the topic in [23], we only discuss a few techniques here. The most basic way of achieving persistence is using functions provided by the host OS for task automation and scheduling. On Linux distributions, this includes initialization scripts that execute commands automatically upon system boot or launching the shell, and task schedulers, e.g. `cron`, `anacron` and `at`.

[24]`https://elixir.bootlin.com/linux/v5.1/source/arch/arm64/mm/mmu.c\#L445`, Last visited: Mar 19, 2021

Existing software binaries can be easily modified or replaced on the disk by malware, resulting in the malware being executed when the infected binaries are executed. Libraries can be placed and loaded through the dynamic linker, resulting in the backdoored processes executing malicious code without modifying the program binary itself. Kernel modules that load during the boot process are also possible targets for achieving persistence. These persistence mechanisms leave traces which can be revealed by scanning the file system. We note that there are other mechanisms as well, some of which are not detectable from the file system (e.g., modified boot drivers and firmware). Their detection requires different protective measures, like a secure boot implementation.

We detect rootkits trying to evade our memory checks via persistence mechanisms by scanning the file system for integrity violations. We describe the known uninfected state in Subsection 5.1 and present the scanning process in Subsection 5.3. Our detection method assumes a specific structuring of the file system, detailed in Subsection 5.2. The baseline state to compare hashes against is stored in the TEE's trusted storage, the details are discussed in Subsection 5.4.

5.1 Design decisions and hashed file system entries

Our approach for detecting the persistent parts of rootkits on the file system is to recursively compute hash values for selected directories. To compile such a list, we originally experimented with parsing the crontab files of the system. Our goal was to extract all file references from the scheduled shell commands and use these results together with scanning a selected set of important directories recursively to reduce running time. We used a static analysis approach based on the source code of the bash command parser and `cron`'s crontab entry parser. It is possible to extract the scheduled commands from the crontab files, however, one must pay attention to the specified environment variables and keep track of which user's permissions and `HOME` setting will be applied during execution. We built an abstract syntax tree for each command in the crontab file, extracted the possible file paths from the command's parameters (including the invoked command's own file), and expanded all valid paths into absolute paths using the `PATH` and `HOME` variables.

However, this approach has serious issues which may result in the target list being incomplete. For example, if the crontab file includes a program which dynamically loads and executes another binary, our static analysis would miss that binary. What is more, our static analysis would need to be able to handle paths which are constructed dynamically (e.g., using loops). In order to overcome this challenge, either a "cron policy" is needed which limits how the users can define scheduled commands or the analysis should be extended with dynamic tools which run the commands in a controlled environment and extract the directories and files accessed during execution.

Another automated approach to constructing the list of file system entries to hash would be to identify all executable files on the file system. However, we cannot identify such files using the `x` permission flag, because file permissions are easy to change. Recognizing ELF binaries by reading the first few bytes for the magic value is doable but the device may contain other types of executable files (e.g., script files). Differentiating between script files for different interpreters, such as Python and bash, and simple text files is challenging without executing them.

As our method is designed for embedded IoT devices, we assume a small local file system. Thanks to the small file system, we can expect to perform a thorough scan in a reasonably short amount of time. Therefore, we compiled a general list of file entries to hash which consists of many of the well-known top-level Linux directories, like `/bin`, `/lib` and `/etc`. There are directories with highly volatile contents, such as `/tmp, /var` and `/dev`, which we do not include in our scanning process. We do not rely on the common and default locations and naming schemes of the persistent components of known rootkits. We also do not rely on any malware signature databases, only on our own reference hashes. This way, we could potentially detect previously undiscovered rootkits, and are prepared for ones that randomize the names and locations of their components.

5.2 Recommended file system structuring

We now discuss a few basic structuring practices regarding the file systems mounted on the device which serve to greatly reduce the false positive rate of the file system scans. Our detection of persistent rootkit components on the disk was designed with the assumption that the device adheres to these practices.

As we use hashing to detect inconsistencies, it is necessary to separate the mounts containing static, unchanging files like program binaries and linked libraries from the ones containing dynamic, often-changing ones, such as log files. It is recommended to add the `noexec` flag to every mount apart from the root file system, which should be the one containing all programs and scripts executed during normal operation.

For files that change during runtime and cannot be moved from the root file system, we provide the option to be excluded from the hashing process. We recommend to use this option only for non-executable files for the following reason. Even if rootkits infect these files to achieve persistence, they must also stop their execution to evade our checks. Therefore, they need another component to execute the files in which they hide to re-infect the system. If the file in which rootkits hide is never executed, the rootkit cannot re-infect the system using this evasion method.

5.3 The hashing process

The hashing process is performed entirely in the Trusted Execution Environment (TEE). We take a predefined list of directories and/or file paths to be hashed and create a hashing context for each entry. In the case of directories, we recursively read the contents of all files from all subdirectories, and load them into the hashing context. We produce a single SHA-256 digest for every individual entry in the target list, as shown in Figure 3. The list of paths to be hashed is shuffled before each new scan to provide a degree of unpredictability to the hashing order; this serves

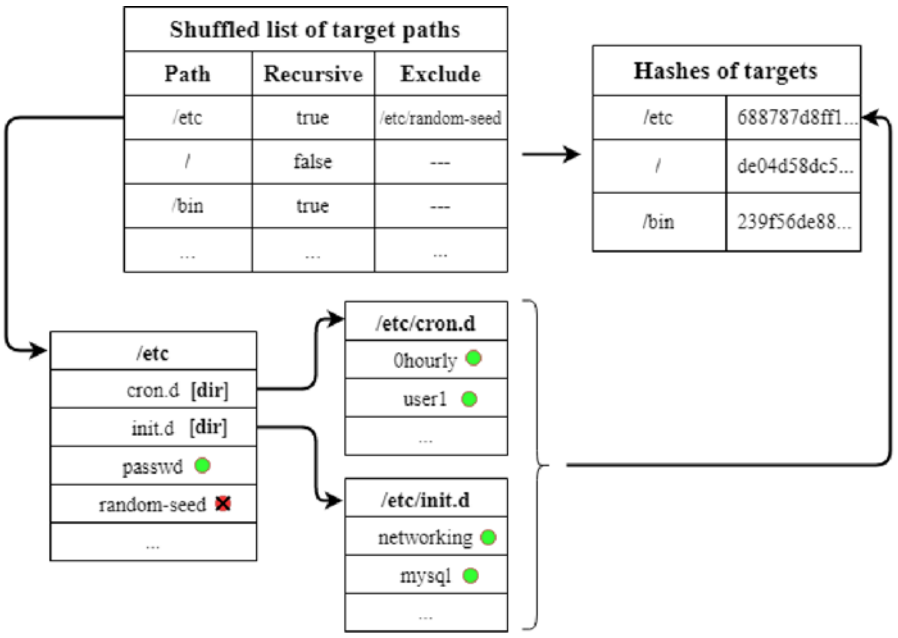


Figure 3: Each entry in the list of targets consists of an absolute path, a flag to indicate whether to traverse the subdirectories recursively or not, and a list of files to exclude from the hashing. For each target entry, a single SHA-256 digest is generated. If the recursive flag is set to false, only the files in the top level directory are calculated into the hash. If it is set to true, all subdirectories are recursively parsed until the maximum file path length of Linux, and all files not explicitly excluded are calculated into the hash.

as a secondary hardening technique against timing window attacks, supplementing the use of the fanotify API (see end of Subsection 4.3). This does not affect the digests themselves, since the order in which we read files inside directories remains unchanged.

The reference hashes to which we compare the computed ones are initialized on first launch, based on the state of the file system at that moment, and are placed into the TEE's trusted storage to protect them from tampering. Ideally, this first launch should occur before putting the target system into active use, but after the environment has been configured and all files necessary for operation have been written to the disk. The stored reference hashes become outdated after a system update or reconfiguration, and have to be replaced. In this paper, we do not discuss how this update can be solved, but an automated solution for this problem is part of our future plans.

The reference hashes are loaded from the trusted storage and compared to the computed hashes. Targets with mismatching hashes are noted and should be examined further by the operators. It is recommended to keep a full reference hash set based on the initial state of the storage in order to find the exact cause of the mismatch. Pin-pointing mismatching locations would require keeping a 32-byte hash on the trusted storage for every entity present on the file system. This approach, however, requires much storage space, which is a drawback in the case of IoT devices, as they are often equipped with only a small flash memory for storage. As a result, a trade-off must be made between the amount of storage available and the precision with which mismatching locations can be identified.

5.4 Using the TEE's trusted storage

In this subsection, we briefly discuss a few important features of the TEE's trusted storage API that are necessary for protecting the authenticity and integrity of our reference hashes. It is important to mention that according to the specification of the trusted storage API [13], a single, designated Trusted Storage Space is provided for each trusted application. As a result, our stored reference hashes are accessible only by authorized TAs running on the same device in the same TEE as when the data was created.

The inner workings of the trusted storage are highly dependent on the TEE implementation, and it may not be entirely separated from the REE file system. Consequently, file modifications from a root privileged user account could be a valid concern. The trusted storage is expected to provide confidentiality and authenticity through the use of authenticated encryption. This protects our reference hashes against targeted modification and replay attacks. For additional protection and separation from the REE file system, the reference files can be stored on a Replay Protected Memory Block (RPMB) partition[25].

The attacker could attempt to delete the reference hashes from the trusted storage, because if this file cannot be found, we assume that our detection method is in the initialization phase and file hashes are computed. According to the specification, the trusted storage is protected from such attacks because a stored object should not be accessible from outside the TA that created it. How this functionality is achieved in practice is, again, highly dependent on the TEE implementation. In our chosen implementation (see next section), the TEE recognizes the data corruption and generates an alert.

6 Implementation details

We use the Open Portable Trusted Execution Environment[26] (OP-TEE) as the trusted execution environment. It was initially developed by ST-Ericsson and currently owned and maintained by Linaro. Our implementation uses version 3.6.

Besides the TEE itself, which is essentially a minimal OS running in the Secure World, OP-TEE consists of Normal World components as well: Linaro has its

[25]`https://lwn.net/Articles/682276/`, Last visited: Sep 18, 2020
[26]`https://www.op-tee.org`, Last visited: Sep 20, 2020

own fork of the Linux kernel, which includes an OP-TEE driver. This driver is responsible for shared memory allocation between the two worlds and provides the RPC functionality through which the CA and TA can communicate. The driver exposes its functionality to the user space via a block device. However, in order to increase usability, the driver also has a counterpart in the user space, a daemon called `tee-supplicant`. OP-TEE-related library calls are handled by this daemon which communicates with the driver using ioctl. This is a bidirectional channel, meaning that the driver is also capable of requesting operations from the daemon.

There is also a method for extending the functionality of the OP-TEE kernel: pseudo-trusted applications (PTAs). These applications must be compiled into the OP-TEE OS and they are capable of exposing core functionality to TAs or CAs. We used this feature to implement functionality necessary for accessing non-secure memory, which is otherwise forbidden for TAs. We discuss how we read REE memory and files via PTAs from the TEE in Subsections 6.1 and 6.2, respectively, and describe the checks we implemented in order to protect the REE components of OP-TEE in Subsection 6.3.

6.1 Normal World memory API

To be able to read the memory of the REE kernel, we had to instruct the memory management unit of the OP-TEE OS to map the physical memory range where the Linux kernel resides. We determined this range from the boot log and modified `core_mmu.c` to register it as non-secure RAM. This makes it accessible in the context of the TEE, but TAs cannot read it due to address translation issues. The Normal and Secure Worlds are using two distinct address spaces, which means that if we take the address of a Linux kernel object, OP-TEE will not be able to process it. We solved this issue by translating Normal World addresses to physical address, and back to Secure World virtual addresses. Fortunately, OP-TEE provides us with tools to accomplish the latter task, so we only needed to implement the former one. The Linux kernel uses a macro called `__virt_to_phys_nodebug`, which can be unfolded into three other macros, which in turn are also unfolded into other macros. All in all, we implemented 30 macros based on `__virt_to_phys_nodebug` to be able to translate Normal World virtual addresses to physical addresses. The Normal World virtual addresses are obtained from the `System.map` file of the compiled kernel, except for runqueues. Since they are not intended to be used by any kernel module, we had to implement a patch to dump these addresses into the boot log, and then we hard coded them into our TA.

In order to access the physical memory in which the Linux kernel resides, we implemented a PTA. The PTA implements the following interface:

read_mem This function expects two parameters, a memory region to copy data into and a physical address to copy data from. It translates the address to a Secure World virtual address, and after performing the necessary checks, it populates the buffer with the requested amount of data.

hash_mem This function is used to create a hash from non-contiguous REE memory ranges. It expects an array of `phys_mem_range` structures (containing a pointer to a memory region and a size), and a buffer where to store the created hash. It initializes a hash context, iterates through the array of physical memory ranges, translates the addresses back to virtual addresses and feeds the specified regions to the hash function. When it is done, it copies the produced hash into the output buffer. We chose the SHA-256 hash function and OP-TEE OS is configured to use the libtomcrypt library[27] by default.

6.2 Normal World file API

OP-TEE does not provide access to the REE file system by default. However, we noticed that OP-TEE's trusted storage stores encrypted data on that file system, hence we suspected that there must be a way to access the REE file system from OP-TEE. After digging the source code, we discovered that the `tee-supplicant` daemon can be instructed via RPC calls to perform REE file operations. However, the set of RPC functions available to perform file operations were not written for general purposes, but they were designed specifically for the trusted storage. As a result, to be able to pass arbitrary filenames as parameters, we had to modify the `open` and `opendir` functions. Function `readdir` also had to be modified due to a bug we discovered. Our pull request with the fix is merged already[28], but it is only included in the 3.10 release. We modified the previously mentioned functions by making copies of them in our PTA and applying the necessary changes to the copies. In addition, as the root of the trusted storage is `/data/tee/` and filenames are prefixed with this string in `tee-supplicant`, the PTA expects absolute filenames and prefixes them with `../..` before passing them to `open` and `opendir`. The PTA implements the following interface:

hash_file This function expects a filename as a string and a buffer to store the computed hash in. It opens the requested file (if exists), reads its content by 4096 bytes and passes these blocks to a hash function. When the end of the file is reached, it finalizes the hash and copies it into the output buffer.

hash_dir This function takes four parameters: a directory name, an output buffer, an integer to indicate if we want to hash recursively (0 for false, 1 for true) and a pointer to a null-terminated array of strings. It creates a hash context, opens the specified directory (if exists) and reads its content. For every entry, we check the blacklist first (to avoid hashing files with changing content, e.g. `/etc/random-seed`). If the entry is not on the blacklist, we try to determine if it is a regular file or a directory. Since we have no `stat`-like primitive, we do this by invoking `opendir`. If `opendir` fails, we have a file, otherwise, a directory. For files, we do the same as above: read the file by blocks and

[27]`https://optee.readthedocs.io/en/latest/architecture/crypto.html`, Last visited: Sep 18, 2020

[28]`https://github.com/OP-TEE/optee\_os/pull/3962`, Last visited: Sep 18, 2020

feed every block to the hash function. If the entry is a directory and we hash recursively, then the function calls itself recursively on the entry. Otherwise, the entry is skipped. Finally, the hash is copied to the output buffer. We use the SHA-256 hash function from the libtomcrypt library.

In order to ensure the proper functioning of the above described PTA, we had to apply two patches to the `tee-supplicant` daemon. First, we had to give it root privileges, otherwise it cannot read certain files. Second, `tee-supplicant`'s `readdir` handles certain directories improperly: if a directory only stores hidden files, it is considered to be empty. From the aspect of our persistence checks, this behavior can be fatal, so we had to patch the appropriate function to only skip `.`, `..` and `.nfs*` (these files are created by an NFS server, when an open file is deleted).

6.3 Verification of OP-TEE specific components

As we do not trust software components in the REE, but our implementation relies on using OP-TEE's Normal World components, we needed to ensure the integrity of them. For OP-TEE's Linux driver, we did not need to implement any additional checks, because it is compiled as a part of the kernel, so its integrity is verified when our TA checks the integrity of the kernel code. The daemon `tee-supplicant`, however, had to be slightly modified before we applied the same technique as we used for checking the integrity of our CA. First, we built it as a static binary. Next, we had to ensure that the pages containing the code of the daemon are present in the memory, and not swapped out. We implemented a Linux driver to create an entry under `/proc`. If pids are written to it, it looks for the corresponding task, and if it is a thread of `tee-supplicant`, it generates a page fault for every page of its code segment. With these preparations, we can apply the same check on `tee-supplicant` as we did on the CA, except that we compute the hash of every task, if it has the proper name (`tee-supplicant`). This check is executed right after we check the code of our CA. We could extend this kind of integrity check to other tasks as well, as described in Subsection 4.3, however, our implementation currently verifies only the `tee-supplicant` daemon and the CA.

7 Evaluation

We evaluated our implemented method on different types of rootkits. Our results in detecting kernel space rootkits are presented in Subsection 7.1. In Subsection 7.2, we present the results of detecting a custom rootkit's persistent components. The performance of our implementation is presented in Subsection 7.3.

7.1 Detecting different rootkits in kernel memory

In order to test our rootkit detection method, we wrote multiple kernel space rootkits. We also wanted to test it against real kernel space rootkits found in the wild, but we were unable to find any that would work in our environment (the ones we found on github were written for older kernels or different architectures). However, we did manage to make user space rootkits work in our environment. In this subsection, we will describe these rootkits and the way our method was capable of detecting them.

Syscall hook: This test rootkit was implemented as a Linux kernel module. When it is loaded, it disables the write protection of the kernel's `.rodata` segment and replaces one of the function pointers in the system call table. Finally, it re-enables the write protection. On unload, it restores the modified pointer. We are capable of detecting the corruption of the system call table by comparing the freshly computed hash to one created from the intact table.

Inline hook: This rootkit is similar to the previous one: in this case, we remove the write protection of the text segment of the kernel and tamper with the first few bytes of a chosen function, then restore it on unload. Again, this modification will change the hash of the text segment, and comparing it with the original one will reveal the presence of the rootkit.

DKOM I: This rootkit creates a backdoor process and attempts to hide it via removing it from the initial pid namespace's IDR. For a regular `ps`, the backdoor process is invisible, however it is possible to connect to it. Our solution detects the backdoor because it was not removed from the list of all the tasks, but it was missing from the output of `ps`.

DKOM II: In this test, we extended the functionality of the previous rootkit: we remove the backdoor process from the task list and the task tree as well, thus, our solution's only chance is to find it in any of the run queues. At this point, detection success depends on the rootkit's payload. In our case, it was a bind shell, which spends little time in run queues; it mostly waits, therefore, it is in wait queues. As currently our implemnentation does not check wait queues (the issue of wait queues will be explained in Section 8), we did not detect this rootkit. However, if the payload had been a computationally intensive task, such as a cryptocurrency miner, it would probably have been detected.

A HORSEPILL variant: HORSEPILL [19] is a ramdisk-based rootkit, which exploits namespaces. It infects `klibc`, a minimal library used in the early user space. It hides a process by creating a new pid namespace and executes `systemd` in this namespace. Normally, it would not be possible to see kernel threads with this setup, but HORSEPILL has a workaround to fake these in the freshly created namespace. Unfortunately, HORSEPILL is not compatible with our test environment, as we use a different init system, we do not use klibc, and we do not assemble ramdisks on the system like personal computers usually do. However, we were able to port the idea behind HORSEPILL to work in our environment. In our case, the ramdisk is assembled by Buildroot[29], and starting the init process happens as follows: First, the Linux

[29]`https://buildroot.org/`, Last visited: Sep 18, 2020

kernel calls `/init`, a minimal shell script, and sets the 3 default file descriptors to `/dev/console`. Then, it invokes `/sbin/init`, which is a symlink to busybox[30]. We managed to start our HORSEPILL variant by replacing the symbolic link to another binary. This binary clones two new threads and executes the original init and our backdoor in them. Busybox is executed in a new pid namespace to hide the backdoor from the rest of the system. Our implementation lacks HORSEPILL's feature of faking kernel threads, therefore, they appear as hidden processes to our detection solution, much like the backdoor process. Originally, HORSEPILL fakes kernel threads by collecting their names, creating new processes in the namespace of init, and renaming them to the names of the kernel threads. The output of our detection mechanism would be the same in this case: we would find the original ones and the backdoor.

BEURK: BEURK is a userspace rootkit, the name stands for Beurk Experimental Unix RootKit. It exploits the linker's capability to load a library into the address space of a process before everything else, thus, it can hook certain library calls. It creates a backdoor when the `accept` function is called and certain conditions are met (local port matches its configuration, remote port is in range specified in its configuration). The created process is hidden from every other process, except its children. This is achieved by hooking the `readdir` and `readdir64` functions, which are wrappers around the `getdents` and `getdents64` system calls, respectively. When they are called on `/proc`, if the next entry is the pid of the backdoor, then it is skipped. We were able to detect the presence of the BEURK backdoor because it is missing from the output of `ps`, but present in the task list.

7.2 Detecting persistent components

In order to illustrate how our solution can detect the persistent components of rootkits, we created a proof of concept demonstration based on a potential vulnerability in an earlier version of our rootkit detection approach. Before we decided to use fanotify to prevent executables from starting during our checking process executed by our TA, we experimented with a technique that a sophisticated rootkit could use to avoid detection and stay persistent even after both the kernel memory and the file system checks have finished. The concept is the following:

1. Let's assume that the system has been compromised and the rootkit process is present in the memory. Let's also assume that the rootkit is able to predict when the next scan will happen and has an accurate idea of how long each scanning phase takes to finish.

2. Right before the scan is about to launch, the rootkit places a binary on the file system (which it has been storing on the heap), which, when executed,

[30]`https://busybox.net/`, Last visited: Sep 18, 2020

would load the rootkit back into memory and reinfect the system. It also runs `chmod` to make the file executable.

3. The rootkit then creates a crontab entry that is scheduled to execute the binary in the time window between the memory and the file system scan.

4. The rootkit process terminates at this point in order to avoid detection. Since it has no process loaded into memory and made no modifications to the kernel, the first phase of the scan finishes without finding any inconsistencies.

5. The scheduled task executes, the rootkit is loaded into memory. It immediately deletes the previously placed binary from the file system, and restores the modified crontab file.

6. Since every modification on the file system was reverted, the second phase of the scan finds no mismatching hashes.

This vulnerability was deemed very hard to exploit in practice due to the limited time window available for the attack. It is important to mention that the versions of cron based on Vixie-cron [31] were not designed to execute tasks in such an accurate manner either, it only "wakes up" at each minute. The task's execution could be delayed by an arbitrary number of seconds by using the sleep instruction in the scheduled task. Accurate timing would be hard to achieve reliably, considering that the interval between the scans is randomized, but we still considered it an architectural weakness, for which we needed a proper solution, not just hardening and mitigations.

With the current architecture, this vulnerability does not exist because we place the `FAN_OPEN_EXEC_PERM` fanotify mark on every file system. Fanotify would prevent the malicious crontab entry from executing the rootkit binary (more accurately, it would prevent the cron daemon from creating the bash session to run the shell command executing the binary), and the file system scan would detect the mismatches in the directory containing the rootkit binary.

As a demonstration, we prepared a small, statically linked C program which stores its own binary's bytes and is able to place and delete the binary on the file system at will. To simulate the perfectly timed scenario, instead of using cron, we used a small bash script to execute the binary on time, and disabled the memory checks.

The results were the following: The script tried to relaunch the program after it placed its binary in `/bin` and terminated, but fanotify prevented the relaunch. The script was waiting for permission to execute the binary, but the request was put on hold while the file system scan was running. The file system scan then detected the mismatch in `/bin`. Without the fanotify marks in place, the program would have been able to launch and delete the binary, and avoid detection.

[31] `https://directory.fsf.org/wiki/Vixie-cron`, Last visited: Sep 18, 2020

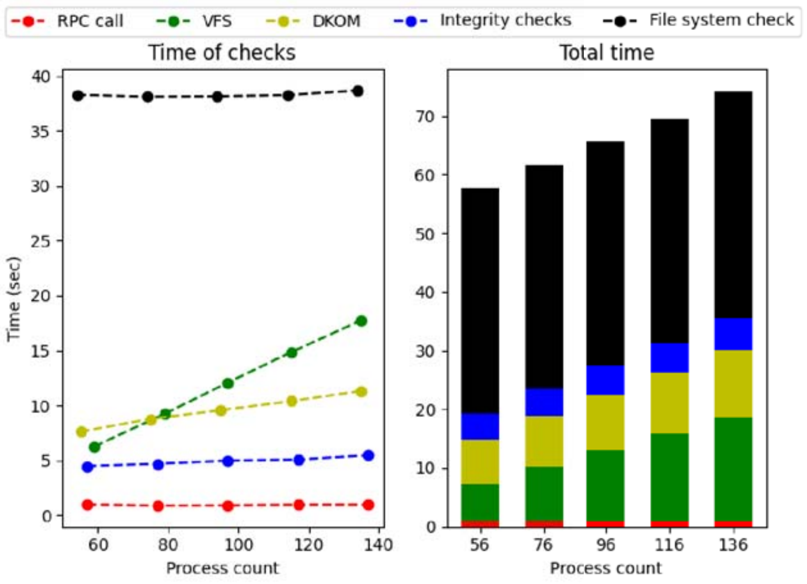


Figure 4: Execution time in seconds depending on the number of processes

7.3 Performance measurements

Performance impact is always a concern with anti-malware solutions. In this subsection, we present the performance characteristics of each part of our implementation and review their impact on the system. Our implementation runs in a virtualized environment with 1057 MBs of RAM and 2 Cortex-A57 cores.

Since most of our checks are process-related, we measured the execution speed depending on the number of currently running processes: we spawned new background processes with the `less` command, which did not consume relevant amount of CPU time slices, but increased the size of the data structures we needed to check. Figure 4 shows the execution time of the checks separately and in total as well.

RPC call stands for the communication between the CA and TA: all necessary input is collected, the TA is invoked and the TA performs normalization of its input. As expected, the number of running processes is irrelevant in this case.

Checking the integrity of the VFS does not scale well. This is due to the nature of the `/proc` file system. For every process, approximately 300 new files are created by the kernel. These are regular files and symbolic links, resulting in an increasing number of inodes. Our checks scale linearly with respect to the number of inodes.

The DKOM check achieves better performance with respect to the growth in the process count: for n processes, it traverses a list made of n elements and two trees with n nodes each. It also sorts arrays of n pids and performs binary searches to find differences between the collected lists of pids.

Integrity checks scale very well, the hash check of the kernel's `text` segment and the system call table can be performed with constant time complexity, and interaction with task-related data structures is necessary only when it is looking for the task of the CA (for the purpose of hashing its `text` segment and verifying the integrity of the `fanotify` marks) and the tasks of `tee-supplicant`. However, extending this integrity check to other tasks would probably have an impact.

Finally, our solution performs the file system checks. These checks have no relation to the process count, they depend on the number of files included in the check and the overall size of those files. This part is by far the most time consuming, since it requires RPC calls and world switches to read the bytes of the files.

While our scanning process is being executed in the TEE, it uses only one of the cores. As a result, the REE can execute on the other core. Processes running in the REE are not halted while we perform our checks, but significantly less resources are available to them until the checks are completed. Until our TA reaches the file system checks, the core it uses can only execute REE code when an interrupt occurs that has to be handled in the REE. During the file system checks, however, our implementation needs to wait for a lot of I/O operations, and while waiting, control is given back to the REE, where the Linux kernel's scheduler can execute other tasks on the first core as well.

8 Discussion and future work

In this section, we present some known weaknesses of our approach and discuss ideas to overcome them.

8.1 Known limitations and possible solutions

The first limitations comes from OP-TEE and the way it handles interrupts. Interrupts are divided into two groups, foreign and native interrupts. The former one needs to be handled by the Normal World, and the latter one by the Secure World. If an interrupt rises and the CPU which should handle it is not in the appropriate world, the machine switches to the correct world, and the interrupt handler is executed. However, OP-TEE does not have its own scheduler, but it uses the Linux kernel's scheduler. When a CPU is executing code in the Secure World and a foreign interrupt occurs, the execution of the TA does not continue immediately after the handler exits. It only resumes execution when the scheduler gives the CPU to the thread associated with the TA.

In addition, on a system with multiple cores, it is possible for one core to execute in the Normal World and another in the Secure World. This behavior can make our inconsistency checks unreliable: it is possible for a new thread to start during our checks, making them fail despite the lack of any hidden processes. This issue might be resolved if we can disable other cores during our checks, and disable interrupts as well to ensure the uninterrupted execution of our checks. Disabling a core is possible from the REE, however, it has a negative impact on the performance of the system. Disabling interrupts is a bit more complicated, although, PTAs can do it while they are running. It might be possible to disable and re-enable interrupts for other Secure World threads as well, or the check itself can be implemented in a PTA in order to use this feature.

Another, less manageable issue is the way the Linux kernel handles waiting tasks. This is implemented via wait queues, which store the tasks waiting for the same event. When the event occurs, it is possible to wake up all the tasks or just one of them. The issue lies with how Linux creates wait queues: some exists as global variables, but most of them are created on stacks or as members of other structures. So far, we have not been able to find a way to enumerate all the currently existing wait queues in memory, thus, we are unable to check all of them, only those implemented as global variables.

Our current method is unable to handle self modifying features in the Linux kernel and Address Space Layout Randomization in the kernel. These features would break the integrity check of the kernel's `text` segment.

A source of another limitation is our decision of not checking the entire VFS layer, only the objects accessible from the superblock of `/proc`. As a result, the kernel-level integrity of other file systems are not checked, but we rely only on file hashing. A possible attack against the persistence check might be to hook the `read` function in the default file operations of an inode to show the original content of the file for every entity except for the one executing it.

Finally, a way to bypass all of our checks would be for a malware to uninstall itself before the checks are performed. When the checks are completed, the malware could be reinstalled by exploiting the same vulnerability it originally exploited to infect the system. Note, however, that this can work against any rootkit detection approach, because there remains nothing malicious to detect in the system.

8.2 Future work

There are multiple features with which our method could be extended. First, our current implementation does not support updates. REE package updates would modify or add new files to the file system, so they would likely break the file system check for persistent rootkit components, and a kernel update would certainly break the integrity checks and probably the VFS and task-related ones too. In the future, we would like to address the former issue by recomputing reference hashes in a secure way. The latter issue may require re-implementing certain checks, making its automation challenging.

We discussed multiple checks for Linux kernel modules. They are the most convenient way to execute code in kernel space [7], therefore, rootkits often use them and try to hide their modules. Consequently, we disabled module support in the Linux kernel configuration, making rootkit installation more challenging. However, without module support, all necessary drivers must be compiled into the kernel, which is a functional restriction. In addition, there are other kernel resources worthy of being checked as well, such as components of the network stack.

We would also like to make our solution compatible with other kernel security features. We reviewed a long list of possible configurations and our solution is incompatible with only two of them: structure randomization and Kernel Address Space Layout Randomization. In case of structure randomization, the members of selected kernel structures are randomized at compile time. We could make our method compatible with this feature by compiling a kernel module with access functions, for example, to get the next task in the task list, and use these functions from a library in our TA. Kernel Address Space Layout Randomization is a technique which places the kernel's text segment at a random location at boot time. This feature interferes with several of our checks and we do not have a solution so far that can support this feature.

9 Conclusions

In this paper, we addressed the problem of detecting rootkits on embedded IoT devices. Rootkits are malicious software that typically run with elevated privileges, which makes their detection challenging. Our solution is based on identifying signs of a rootkit infection (i.e., modifications to the code of system programs and the operating system kernel, as well as inconsistencies in certain kernel data structures) using a trusted application that is running in an isolated trusted execution environment. Fortunately, such trusted execution environments are supported on many embedded platforms used in IoT applications, and their protection measures ensure that malicious code cannot interfere with our detection mechanisms even when running with root privileges. We described in detail how we check both the memory of the untrusted execution environment and the persistent storage from our trusted application, looking for integrity violations and inconsistencies. We also reported on a prototype implementation of our approach, including some specific implementation level issues that we had to solve to make our prototype working in practice. Finally, we evaluated our design and implementation by testing the prototype with rootkits that we developed for this purpose.

Our approach has some limitations that we discussed in the paper. In summary, we can detect modifications of the kernel code and system programs, as well as hooking attacks in the memory, and we can also detect the presence of rootkit components in the persistent storage of the IoT device. Detection of manipulations of process related kernel data structures is not complete, as we were not able to analyze certain data structures (e.g., wait queues). In addition, at the time of this writing, we do not support multi-core processors, address space layout randomization, and self-modifying code in the kernel. Some of these limitations can be addressed (e.g., the kernel can be statically compiled with all the drivers included), while others require more work in the future. Despite all these limitations, we believe that our work demonstrates that it is possible to protect even small embedded devices used in IoT applications from sophisticated and powerful software based attacks, and that IoT is not necessarily as insecure as it is commonly perceived.

References

[1] Antonakakis, M., April, T., Bailey, M., Bernhard, M., Bursztein, E., Cochran, J., Durumeric, Z., Halderman, J. A., Invernizzi, L., Kallitsis, M., Kumar, D., Lever, C., Ma, Z., Mason, J., Menscher, D., Seaman, C., Sullivan, N., Thomas, K., and Zhou, Y. Understanding the Mirai botnet. In *USENIX Security Symposium*, August 2017. `https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/antonakakis`, Last visited: Mar 19, 2021.

[2] Baliga, A, Chen, X, and Iftode, L. Paladin: Automated detection and containment of rootkit attacks. Technical report, Rutgers University Department of Computer Science, 2006. `https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.74.9742&rep=rep1&type=pdf`, Last visited: Mar 19, 2021.

[3] Baliga, Arati, Ganapathy, Vinod, and Iftode, Liviu. Detecting kernel-level rootkits using data structure invariants. *IEEE Trans. Dependable Secur. Comput.*, 8(5):670–684, September 2011. DOI: `10.1109/TDSC.2010.38`.

[4] Bharadwaj, R. *Mastering Linux Kernel Development*. Packt Publishing, 2017. ISBN: 9781785883057.

[5] Blunden, William. *The Rootkit Arsenal: Escape and Evasion in the Dark Corners of the System*. Jones and Bartlett Learning, 2012. ISBN: 9781449626372.

[6] Bravo, P. and García, D. Rootkits survey: A concealment story. Technical report, University of Oviedo, 2011. `https://pablo-bravo.com/files/survey.pdf`, Last visited: Sep 21, 2020.

[7] Bunten, A. Unix and Linux based rootkits techniques and countermeasures. In *16th Annual First Conference on Computer Security Incident Handling*. Budapest, Hungary, 2004. `https://www.first.org/resources/papers/conference2004/c17.pdf`, Last visited: Mar 18, 2021.

[8] Carbone, R. Malware memory analysis of the Jynx2 Linux rootkit. Technical report, Defence Research and Development Canada, 2014. `https://apps.dtic.mil/dtic/tr/fulltext/u2/1004190.pdf`, Last visited: Mar 19, 2021.

[9] Devik, S. Linux on-the-fly kernel patching without LKM. *Phrack Magazine*, 2001. `http://phrack.org/issues/58/7.html`, Last visited: Sep 21, 2020.

[10] Embleton, Sh., Sparks, Sh., and Zou, C. SMM rootkit: a new breed of OS independent malware. *Security and Communication Networks*, 6(12):1590–1605, 2013. DOI: `10.1002/sec.166`.

[11] Espensen, M. and Hoej, N. Rootkits: Out of sight, out of mind. Technical report, University of Copenhagen, 2014. `https://github.com/NinnOgTonic/Out-of-Sight-Out-of-Mind-Rootkit`, Last visited: Sep 16, 2020.

[12] Garfinkel, T., Adams, K., Warfield, A., and Franklin, J. Compatibility is not transparency: VMM detection myths and realities. In *Proceedings of the 11th USENIX Workshop on Hot Topics in Operating Systems*, HOTOS'07, USA, 2007. USENIX Association. `https://www.usenix.org/legacy/events/hotos07/tech/full_papers/garfinkel/garfinkel.pdf`, Last visited: Mar 19, 2021.

[13] GlobalPlatform Technology. TEE system architecture – version 1.2. Technical report, 2018. `http://globalplatform.org/specs-library/?filter-committee=tee`, Last visited: Sep 19, 2020.

[14] Gu, J., Xian, M., Chen, T., and Du, R. A Linux rootkit improvement based on inline hook. In *Proceedings of the 2nd International Conference on Advances in Mechanical Engineering and Industrial Informatics*, pages 793–798. Atlantis Press, 2016. DOI: `10.2991/ameii-16.2016.155`.

[15] Heasman, J. Implementing and detecting an ACPI BIOS rootkit. *Black Hat Europe*, 2006. `https://www.blackhat.com/presentations/bh-europe-06/bh-eu-06-Heasman.pdf`, Last visited: Mar 18, 2021.

[16] Herwig, S., Harvey, K., Hughey, G., Roberts, R., and Levin, D. Measurement and analysis of Hajime, a peer-to-peer IoT botnet. In *Network and Distributed Systems Security (NDSS) Symposium*, 2019. DOI: `10.14722/ndss.2019.23488`.

[17] Hudson, T., Kovah, X., and Kallenberg, C. Thunderstrike 2: Sith Strike. *Black Hat USA Briefings*, 2015. `https://www.blackhat.com/docs/us-15/materials/us-15-Hudson-Thunderstrike-2-Sith-Strike.pdf`, Last visited: Mar 19, 2021.

[18] Hudson, T. and Rudolph, L. Thunderstrike: EFI firmware bootkits for Apple MacBooks. In *Proceedings of the 8th ACM International Systems and Storage Conference*, pages 1–10, 2015. DOI: `10.1145/2757667.2757673`.

[19] Leibowitz, M. Horse Pill: A new kind of Linux rootkit. In *Black Hat USA*, 2016. `https://www.blackhat.com/docs/us-16/materials/us-16-Leibowitz-Horse-Pill-A-New-Type-Of-Linux-Rootkit.pdf`, Last visited: Mar 19, 2021.

[20] Petroni, Nick L., Fraser, Timothy, Molina, Jesus, and Arbaugh, William A. Copilot - a coprocessor-based kernel runtime integrity monitor. In *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13*, SSYM'04, page 13, USA, 2004. USENIX Association. `https://www.usenix.org/legacy/publications/library/`

proceedings/sec04/tech/full_papers/petroni/petroni.pdf, Last visited: Mar 19, 2021.

[21] Rudd, E. M., Rozsa, A., Günther, M., and Boult, T. E. A survey of stealth malware attacks, mitigation measures, and steps toward autonomous open world solutions. *IEEE Communications Surveys Tutorials*, 19(2):1145–1172, 2017. DOI: 10.1109/COMST.2016.2636078.

[22] Shah, A. and Giffin, J. Analysis of rootkits: Attack approaches and detection mechanisms. Technical report, Georgia Institute of Technology, 2008. http://beefchunk.com/documentation/security/RootkitsReport.pdf, Last visited: Mar 19, 2021.

[23] The MITRE Corporation. MITRE ATT&CK Tactics: TA0003 - Persistence, 2019. https://attack.mitre.org/tactics/TA0003/, Last visited: Mar 19, 2021.

[24] Todd, A., Benson, J., Peterson, G., Franz, T., Stevens, M., and Raines, R. Analysis of tools for detecting rootkits and hidden processes. In Craiger, Philip and Shenoi, Sujeet, editors, *Advances in Digital Forensics III*, pages 89–105, New York, NY, 2007. Springer New York. DOI: 10.1007/978-0-387-73742-3_6.

[25] Vervier, P.-A. and Shen, Y. Before toasters rise up: A view into the emerging IoT threat landscape. In *IoT Security Foundation Conference*, 2018. DOI: 10.1007/978-3-030-00470-5_26.

[26] Wang, Y., Beck, D., Vo, B., Roussev, R., and Verbowski, C. Detecting stealth software with strider ghostbuster. In *2005 International Conference on Dependable Systems and Networks (DSN'05)*, pages 368–377, 2005. DOI: 10.1109/DSN.2005.39.

[27] Wichmann, Rainer. File integrity checkers: A comparison of several host/file integrity monitoring programs, 2009. https://www.la-samhna.de/library/scanners.html, Last visited: Mar 3, 2021.

[28] Zhihong Tian, Bailing Wang, Zixi Zhou, and Hongli Zhang. The research on rootkit for information system classified protection. In *2011 International Conference on Computer Science and Service System (CSSS)*, pages 890–893, 2011. DOI: 10.1109/CSSS.2011.5974667.

Acta Cybernetica **25** (2021) 401–419.

Zero Initialized Active Learning with Spectral Clustering using Hungarian Method[*]

David Papp[a]

Abstract

Active learning tries to reduce the labeling cost by allowing the learning system to iteratively select the data from which it learns. In special case of active learning, the process starts from zero initialized scenario, where the labeled training dataset is empty, and therefore only unsupervised methods can be performed. In this paper a novel query strategy framework is presented for this problem, called Clustering Based Balanced Sampling Framework (CBBSF), which aims to uniformly select the initial labeled training dataset. The proposed Spectral Clustering Based Sampling (SCBS) query strategy realizes the CBBSF framework, and therefore it is applicable in the special zero initialized situation. This selection approach uses ClusterGAN (Clustering using Generative Adversarial Networks) integrated in the spectral clustering algorithm and then it selects an unlabeled instance depending on the class membership probabilities. In order to derive class membership probablities from the clustering information SCBS uses the Hungarian algorithm. Experimental evaluation was conducted on balanced and imbalanced MNIST datasets, and the results showed that SCBS outperforms the state-of-the-art zero initialized active learning query strategies in terms of accuracy.

Keywords: active learning, zero initialization, query strategy, clustering, spectral clustering, hungarian method

1 Introduction

The main goal of classification applications is to make predictions with high accuracy. A crucial part of this process is the model creation, which is based on the labeled training data (where the labels are the ground truth categories); hence the gathering of labeled data is also an important component of supervised machine learning. One can collect large amount of inexpensive unlabeled data through

[*]The research was supported by the ÚNKP-19-3 New National Excellence Program of the Ministry of Human Capacities. The research has been supported by the European Union, cofinanced by the European Social Fund (EFOP-3.6.2-16-2017-00013, Thematic Fundamental Research Collaborations Grounding Innovation in Informatics and Info-communications).

[a]Dept. of Telecommunications and Media Informatics, Budapest University of Technology and Economics, Budapest, Hungary, E-mail: `pappd@tmit.bme.hu`, ORCID: `0000-0002-8814-2745`

DOI: `10.14232/actacyb.288006`

real-world applications [16], however labels for this data can be expensive [23], time-consuming or difficult to obtain. For example accurate labeling of speech utterances requies trained linguists [31], pose labelling in videos is extremely time consuming [24], annotating gene and disease mentions for biomedical information extraction usually requires PhD-level biologists [4]. Consequently, in these cases it is recommended to limit the number of labeled data that used for training, while attempting to achieve high accuracy.

Let $U = \{u_i\}, i = 1...m$ denote the total amount of (unlabeled) data available for training; the goal is to select only a subset of this data and assign labels to them, thereby creating the $L = \{l_j\}, j = 1...n$ labeled dataset. The easiest technique is to randomly select L, this method is called passive learning, or random sampling; although the resulting labeled training dataset has a large variance due to the randomness. A more sophisticated approach would be to consider the informativeness of the unlabeled data and then select the most informative ones. This approach is called active learning [20], where the learning system is allowed to iteratively select unlabeled instances and ask for their label. The key idea is that carefully picked, informative data allow the learning algorithm to perform better with less training. A decisive part of an active learning system is how it estimates the informativeness of unlabeled instances; the procedure employed for this purpose is called query strategy.

Usually, active learning query strategies assume that the selection process already started and train a classification model based on L. In special zero initialized situation, the procedure starts with empty L, and therefore only unsupervised techniques (e.g. clustering) can be used. It is often observed, especially for imbalanced or multi-class data sets, that the active learning process does not select the same number of items from each category during the query iterations. This happens because traditional query strategies do not take sample distribution into account in the resulting labeled training dataset. However, the underrepresented classes contain small number of samples, and therefore some attributes are available to the learning system with only an incomplete set of values, thus they lead to sub-optimal models. In zero initialized active learning this is a critical problem, since the process starts with empty L, so in some cases, underrepresented categories contain no samples at all, consequently, the affected attributes are entirely missing. In other words, it is important to query several training items into each category at the start of a zero initialized active learning process.

The subject of this paper is the so-called pool-based unsupervised active learning (UAL) [21], where an instance can be selected from a pool of unlabeled instances (U), while there is not enough labeled data (L) to learn. The learning setup is a multiclass classification problem with k classes, although, the selection and the predictions are based on an unsupervised solution instead of a supervised machine learning method. This paper is concerned with the beginning of the unsupervised active learning, where the number of the labeled data not only a few but zero; i.e. zero initialized unsupervised active learning. Active learner starting from the initial training set selected by appropriate methods can reach higher accuracy faster than that starting from randomly generated initial training set [10]; and therefore, the primary objective was to select a balanced initial training set (so the goal was to get almost the same number of instances of each class).

The main contributions of this paper are (i) the Clustering Based Balanced Sampling Framework (CBBSF) for zero initialized active learning, and (ii) the Spectral Clustering Based Sampling (SCBS) query strategy that realizes CBBSF. SCBS utilizes ClusterGAN (Clustering using Generative Adversarial Networks, [15]) integrated in spectral clustering [26] process to form the clusters in zero initialized environment. After that Hungarian method [12] is employed to connect class membership probabilites to cluster membership probabilities. The rest of this paper is structured in the following way: the next section contains the relevant related work in the literature, Section 3 delineates the proposed CBBSF framework, then Section 4 presents the SCBS selection strategy, and after that the experimental evaluation is presented, finally the conclusions are summarized in the last section.

2 Related work

There are some traditional query strategy frameworks in the literature, e.g. uncertainty sampling [6], query-by-committee (QBC) [25], expected model change [2], expected error reduction [14], or density-weighted method [1]. On the other hand, there are recently proposed query strategies, like uncertainty sampling with diversity maximization [29], Balanced Active Learning (BAL) method [17], extended margin and soft balanced strategy [18], Prototype Based Active Learning (PBAC) algorithm [3] and the hybrid, Expected Difference Change (EDC) [19]. However, these approaches expect the L to be not empty, because all of them applies some kind of supervised machine learning algorithm (e.g. decision tree, random forest [22]), where L is used as training data. Hence, they are not suitable for the special zero initialized active learning (where L is empty), moreover, in this situation most of them are even unable to be executed. The field of active zero-shot learning [28] [27] [7] is partially related to this subject, where the goal is to find a small number of informative seen classes to facilitate unseen class predictions. The setting of active zero-shot learning task contains seen and unseen categories, however in this paper a different (zero initialized) starting environment is examined, where only unseen classes are available.

Unsupervised learning techniques have been successfully used to select the intial training set for active learning. One method is called centroid based selection [11] [9], where unlabeled instances closest to the cluster centroids are selected as starting dataset. In the work [11] the selection happened in one step, while the proposed approach in this paper introduces information gain between the selection of two consequtive items, and therefore it is mandatory to select the items step-by-step. Another selection type is the border based selection [9] which selects the samples with small difference between their highest and the second-highest degrees of cluster membership confidence, i.e. the ones that are around the border between clusters. The combination of center-based selection and border-based selection is called by hybrid selection. Authors of [30] selected half of the instances with center-based, another half with border-based selection, and they achieved this by alternating between the two methods. The centroid, border and hybrid selections were implemented and compared to SCBS during the experiments. The aim of CBBSF is not only to select the initial labeled training dataset, but to uniformly select the instances among the categories to get a balanced labeled training dataset.

3 Clustering Based Balanced Sampling Framework

In this section the Clustering Based Balanced Sampling Framework (CBBSF) active learning query strategy framework is presented. The aim of CBBSF is to select the initial labeled dataset in the special zero initialized situation, where the initial labeled training dataset L is empty. This condition designates a few guidlines: (i) only an unsupervised machine learning algorithm can be used, (ii) the balance of labeled items between the classes is important, (iii) the query strategy should select an item whose class label the learning system is assured of. Satisfying these criteria, CBBSF can be used as a selection strategy for both balanced and imbalanced datasets (see Section 5). After CBBSF selects the initial L set, the active learning could proceed by using another query strategy that is more focused on optimizing the accuracy, but CBBSF could also be used as an end-to-end strategy.

A CBBSF query strategy first performs a clustering algorithm on the unlabeled dataset U, then selects an unlabeled instance to be labeled by an oracle, as can be seen in Figure 1. The selected item should maintain the balance in L; however, in order to achieve this, the class membership probabilities are required so that an item that presumably belongs to the most underrepresented class could be selected. On the other hand, class membership probabilities can not be calculated explicitly because L is empty, and thus supervised machine learning techniques can not be performed.

The clustering algorithm used in CBBSF must return a cluster membership matrix Q, see Equation 1, where q_{ij} is the probability for the i^{th} item to belong to the j^{th} cluster.

$$\begin{gathered} Q = (q_{ij}) \in \mathbb{R}^{n \times k}, \\ 0 \leq q_{ij} \leq 1, \quad \sum_{j=1}^{k} q_{ij} = 1. \end{gathered} \tag{1}$$

Let P be the class membership probability matrix, see Equation 2, where p_{ij} is the probability for the i^{th} item to belong to the j^{th} class. It is important to note that $P \neq Q$, since cluster identifiers are not related to class identifiers. As it was mentioned above, determining P is essential to sustain balance in L, and the elements of P can be derived from matrix Q with an appropriate assignment solution between clusters and classes. During the active learning process, there is no true information about the connection scheme, but this can be estimated based on only the labeled items.

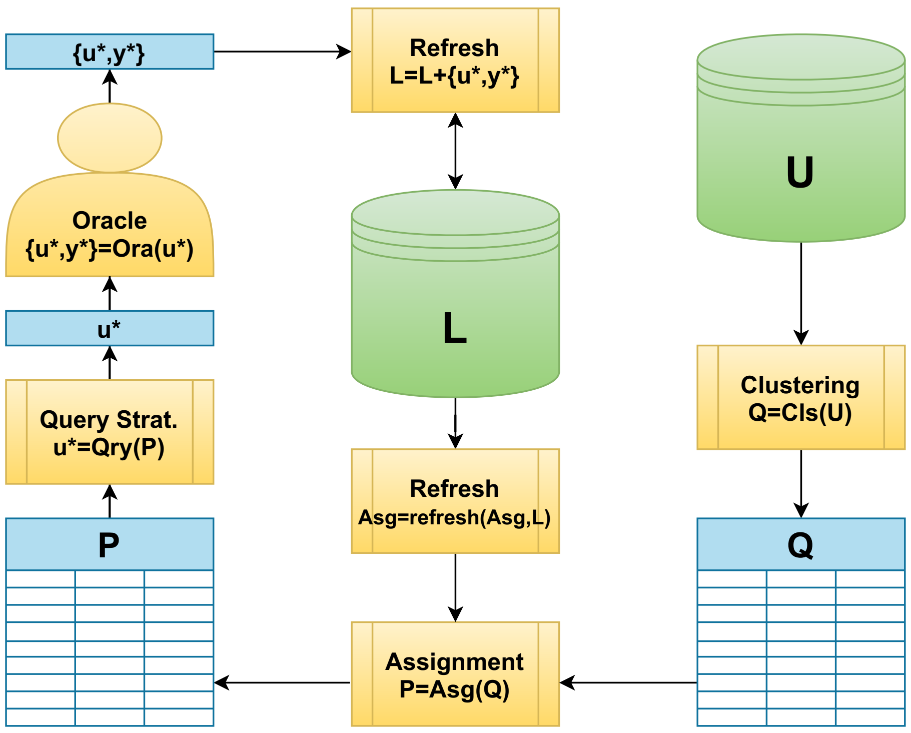


Figure 1: Process of the Clustering Based Balanced Sampling Framework

$$\begin{gathered} P = (p_{ij}) \in \mathbb{R}^{n \times k}, \\ 0 \leq p_{ij} \leq 1, \quad \sum_{j=1}^{k} p_{ij} = 1. \end{gathered} \tag{2}$$

After P becomes available, the most informative unlabeled instance (denote it by u^*) can be selected, and then query its label y^* from an oracle (e.g. a human expert or an all knowing entity). Note that, in this case, most informative means that most likely to preserve the balance in the labeled dataset. The last step is to refresh L by adding the $\{u^*, y^*\}$ pair to it, and based on the new L refresh the assignment pattern as well. The process of CBBSF can be seen in Figure 1, where the datasets, sub-processes and matrices are represented by green, yellow and blue shapes, respectively. It is worth mentioning that U is excluded from the iterative part of this process, since the clustering algorithm is only performed at the beginning to get Q. The reason for this is that the more data is available for the clustering method to work with, the more accurately it can form the clusters.

Nevertheless, a fully iterative variant of this framework could also be used (where L influences the clustering of the remaining items in U), but in this paper such configuration is not examined.

4 Spectral Clustering Based Sampling

In this section, the Spectral Clustering Based Sampling (SCBS) active learning query strategy is presented, which belongs to CBBSF, and thus suitable to be performed in the special zero initialized environment. First, the spectral clustering [26] algorithm is briefly reviewed, and then the realization of CBBSF modules is discussed. Furthermore, Algorithm 1 shows a concise pseudocode for the CBBSF and SCBS based zero initialized active learning algorithm.

4.1 Clustering Module

Given a set of data points $x_1, ..., x_m$, pairwise similarities are calculated based on Euclidean distances, and then a similarity graph G is built to model local neighborhood relationship between the data points. Based on the constructed G graph, a similarity matrix $S = \{s_{ij}\}(i, j = 1...m)$ is derived, where s_{ij} corresponds to the weight of the edge between x_i and x_j in G (if those points are not connected by an edge in G, then $s_{ij} = 0$). Let D be a diagonal degree matrix with $D_{ii} = \sum_j s_{ij}$.

The fundamental step of spectral clustering is calculating the graph Laplacian matrix from the matrices S and D [8] For example, the unnormalized graph Laplacian matrix can be computed as expressed in Eq. 3, and this is the variant used in the SCBS algorithm. Another two popular Laplacians are the symmetric normalized and left normalized [5].

$$\Lambda = D - S \tag{3}$$

Let matrix V be defined as the matrix containing the first k eigenvectors $v_1, ..., v_k$ of Λ as columns. At this point, SCBS applies ClusterGAN [15] to form clusters $C_1, ..., C_k$. The input of ClusterGAN are the rows of V, so the spectral representation of the m datapoints. ClusterGAN is a relatively new clustering approach that performs clustering using generative adversarial networks (GAN). ClusterGAN uses a mixture of distributions (combination of discrete and continuous) to generate latent vectors and to identify different groups in the latent space (the space of latent variables). Besides, it uses a specific clustering error function to train the generator model. Once the data is transformed into latent space, they are clustered using the k-means algorithm. One advantage of using ClusterGAN is that it provides a probabilistic interpretation of the clustering. It outputs so-called cluster decision vectors $q_1, ..., q_m$ from which the cluster membership probability matrix Q can be built (i.e., $q_1, ..., q_m$ vectors are the rows of Q). This algorithm is performed on the initial unlabeled dataset U, and after that, items can be selected by the query strategy.

Algorithm 1 Zero initialized active learning with CBBSF using SCBS

```
input:
  $U$: unlabeled image set
  $k$: number of categories / number of clusters
  $iter$: number of active learning iterations
initialize:
  $C_1, ..., C_k \leftarrow$ Spectral ClusterGAN on $U$ with $k$ clusters
  $v_N$: $k$-long zero vector
  $L = \emptyset$
output: $L$ initial labeled training dataset ($|L| = iter$)

for $N = 1...iter$ do
  if $L \neq \emptyset$ then
    Build the occurrence matrix $A_o$ (Eq. 4)
    $A \leftarrow$ Hungarian algorithm based on $A_o$
    $\hat{P} = Q \times A$ (Eq. 5)
    $h = \text{argmax}(A[:, \text{argmin}(v_N)])$
  else
    $\hat{P} = Q$
    $h = \text{random}(1...k)$
  end if
  $bestValue = \infty$
  $bestIdx = 0$
  for $\forall u_i \in U$ do
    Calculate the informativeness value of $u_i \rightarrow \text{val}(u_i)$ (Eq. 7 or Eq. 8)
    if $(\text{val}(u_i) < bestValue)$ AND $(u_i \in C_h)^\dagger$ then
      $bestValue = \text{val}(u_i)$
      $bestIdx = i$
    end if
  end for
  $u^* = u_{bestIdx}$
  $y^* = \text{query}(u^*)$
  $v_N[y^*]$ += 1
  $L = L \cup \{u^*, y^*\}$
  $U = U \setminus \{u^*\}$
end for
```

Note that L-SCBS uses the condition marked with † symbol, while G-SCBS considers only the condition before the AND operator.

4.2 Assignment Module

SCBS uses single-assignment procedure to implicitly calculate P, so that an appropriate unlabeled item can be selected that maintains even distribution in L.

The class identity and the cluster identity of the labeled items are known. This information can be structured in a table, based on which an occurrence matrix A_o is introduced, as can be seen in Equation 4, where a_{ij} is the number of the items that belong to class j while they are part of cluster i.

$$A_o = (a_{ij}) \in \mathbb{N}^{k \times k} \tag{4}$$

To find the best assignment in the matrix A_o, the Hungarian algorithm [12] is used, although in this case the sum of the entries in the assignment was maximized, instead of the minimization (as it originally happens in the Hungarian algorithm). The connection between Q and $\hat{P}$ is characterized by this best assignment A, which is actually a permutation matrix, thus Q is multiplied by A to get $\hat{P}$, where $\hat{P}$ is an approxiamtion of P; see Equation 5.

$$\hat{P} = Q \times A \tag{5}$$

4.3 Selection Module

Let $C_1, ..., C_k$ denote the k different clusters, and $Y_1, ..., Y_k$ denote the k different classes. Furthermore, introduce the vector $v_N = (N_1, ..., N_k)$ to contain the number of labeled items in the different categories, after N active learning iterations, where N_h is the number of items in Y_h $(h = 1, ..., k)$. The assignment module creates the bijection between C_g and Y_h $(g, h = 1, ..., k)$; hence the number of labeled items in the clusters are also known, at each step. Two variants of SCBS were developed: the Global SCBS (G-SCBS) and Local SCBS (L-SCBS); both of them essentially operates the same way. However, the former minimizes the informativeness metric over every element of U, while the latter examines only a reduced unlabeled set U_{C_g}, which contains the elements of a single cluster. Thus the local version of the algorithm aims to balance L directly by investigating only U_{C_g}, where C_g corresponds to the most underrepresented category in L, denoted by Y_h, as can be seen in Equation 6.

$$Y_h : h = \text{argmin}(v_N) \tag{6}$$

In situations when v_N has multiple minimum values, one of them was randomly selected to designate Y_h.

In order to find the most informativeness unlabeled instance (u^*), two different techniques were used: (i) the first one maximizes the probability of the most probable class, and (ii) the second one minimizes the information entropy over all categories; as can be seen in Equation 7 and Equation 8, respectively.

$$u^* = \underset{i}{\text{argmin}} \left(1 - \hat{p}^*\right), \tag{7}$$

$$u^* = \underset{i}{\text{argmin}} \left(- \sum_{j=1}^{k} \hat{p}_{ij} \times \log \hat{p}_{ij} \right), \tag{8}$$

where $\hat{p}_{ij}$ is an element of $\hat{P}$ and $\hat{p}^*$ represents the probability of the most probable category. Despite that traditional active learning query strategies objective is usually to pick instances with maximum variance, the purpose of CBBSF is to evenly choose the instances from the classes. Consequently, SCBS must be confident that y^* ($\in \{Y_1, ..., Y_k\}$) is the true label of u^* ($\in U$), so that the assignment and the balancing could be feasible. This implies that the most representative unlabeled instance is the most informative for SCBS, i.e. the one which has minimal uncertainty about its true class label.

Table 1: Numer of items in balanced and imbalanced MNIST datasets.

	$\lvert Y_1\rvert$	$\lvert Y_2\rvert$	$\lvert Y_3\rvert$	$\lvert Y_4\rvert$	$\lvert Y_5\rvert$	$\lvert Y_6\rvert$	$\lvert Y_7\rvert$	$\lvert Y_8\rvert$	$\lvert Y_9\rvert$	$\lvert Y_{10}\rvert$	Sum
Balanced (B)	500	500	500	500	500	500	500	500	500	500	5000
Imbalanced1 (I1)	387	516	300	429	482	603	503	700	405	675	5000
Imbalanced2 (I2)	209	850	472	641	558	150	730	354	804	232	5000

5 Experimental evaluation

In this section the experiments are presented that were conducted on the MNIST [13] database of handwritten digits, which consist of 60,000 train and 10,000 test images. The train and test sets were combined into a 70K dataset, and then 5 Balanced (B), 5 Imbalanced1 (I1) and 5 Imbalanced2 (I2) subsets were randomly selected from this dataset, each of them contained 5,000 images (see Table 1). During the experiments, the following 4 SCBS method variants were tested:

- G-SCBS using minimal entropy (G-SCBS 1)
- G-SCBS using most confident (L-SCBS 1)
- L-SCBS using minimal entropy (G-SCBS 2)
- L-SCBS using most confident (L-SCBS 2)

Several additional methods proposed in the literature were also tested: the Centroid [11], the Border [30] and the Hybrid [30] active learning query strategies; furthermore, the Random sampling [9], which selects a random item at each iteration. The results of these competitor methods are compared to the results of the proposed SCBS based techniques.

The tests were performed in the special zero initialized situation, so at the start of the active learning process U contained the total 5,000 images of the test dataset and L was empty. At the testing of each dataset, the goal was to select the initial labeled image collection with a fix size: $|L| = 100$; therefore, in ideal situation each category should contain 10 labeled items. Consequently, only the first 100 active learning iterations were investigated, and in each iteration only one unlabeled instance was selected (i.e., the batch size was one).

In order to evaluate the balancedness in L, two new measures are introduced in this paper: the Average Cardinality Error (ACE, see Equation 9) and the Actual Balancedness (AB, see Equation 10). The latter expresses the amount of balance in L, at the actual active learning step. In case of perfect balance $AB = 1$, while in the worst case (when every item belongs to the same class) $AB = 0$. On the other hand, ACE can be calculated by taking the average of the deviation of actual state from the optimal one.

$$ACE = \sum_{j=1}^{k} \left(\frac{1}{k} \times \left| \left\lfloor \frac{N}{k} \right\rfloor - N_j \right| \right) \tag{9}$$

$$AB = \left(1 - \frac{1}{N} \times \left(\max_j \{N_j\} - \min_j \{N_j\} \right) \right) \tag{10}$$

where N_j is the cardinality number of class Y_j in L, and N is the number of active learning steps. After the evaluation of AB for each individual results got on MNIST datasets, the average of them were calculated, denoted by AAB, as can be seen in Table 2. Furthermore, the accuracy (ACC) was also measured at each iteration on the remaining items in U. ACC is the ratio of the correct decisions and all decisions, where the different types of decisions come from the confusion matrix: True Positive, False Positive, True Negative and False Negative. Note that since at zero initialized active learning there is not enough labeled items to perform supervised learning (i.e. classification), the predicted elements of the confusion matrix are derived from the clustering results by the assignment solution.

In Figures 2-4 the cardinality numbers (N_j) of the classes in L are presented, at iterations 20, 50 and 100, obtained on Balanced, Imbalanced1 and Imbalanced2 MNIST datasets, respectively. Figure 5 shows the average accuracy at each iterations, where SCBS methods are represented with dark, competitor methods are represented with gray lines; each strategy with different markers. The results show that L-SCBS 1 and L-SCBS 2 strategies could achieve higher accuracy than every other method, moreover, in case of balanced datasets both of them were able to perfectly balance L after 100 active learning steps. Regarding imbalanced datasets, L-SCBS 2 seems to perform slightly better, than L-SCBS 1. On the other hand, G-SCBS 1 and G-SCBS 2 could not balance $\{N_j\}$, and therefore it can be concluded that reducing U to only one cluster at a time by leveraging the assingment solution is advantageous. Competitor methods were also unable to reach equilibrium, although at balanced datasets Centroid seems to be promising, since it surpassed the global variants of SCBS. Border technique resulted the highest deviation in $\{N_j\}$, while it gave the highest accuracy on average, after 100 iterations (see Figure 5). This could be explained by analyzing the way it operates, Border method selects instances on the border of clusters, and thus it eliminates uncertain choices, which increases the accuracy. Other methods reached the same level of accuracy, L-SCBS 1 and L-SCBS 2 at around 10-11 steps, while for other approaches it took a longer time.

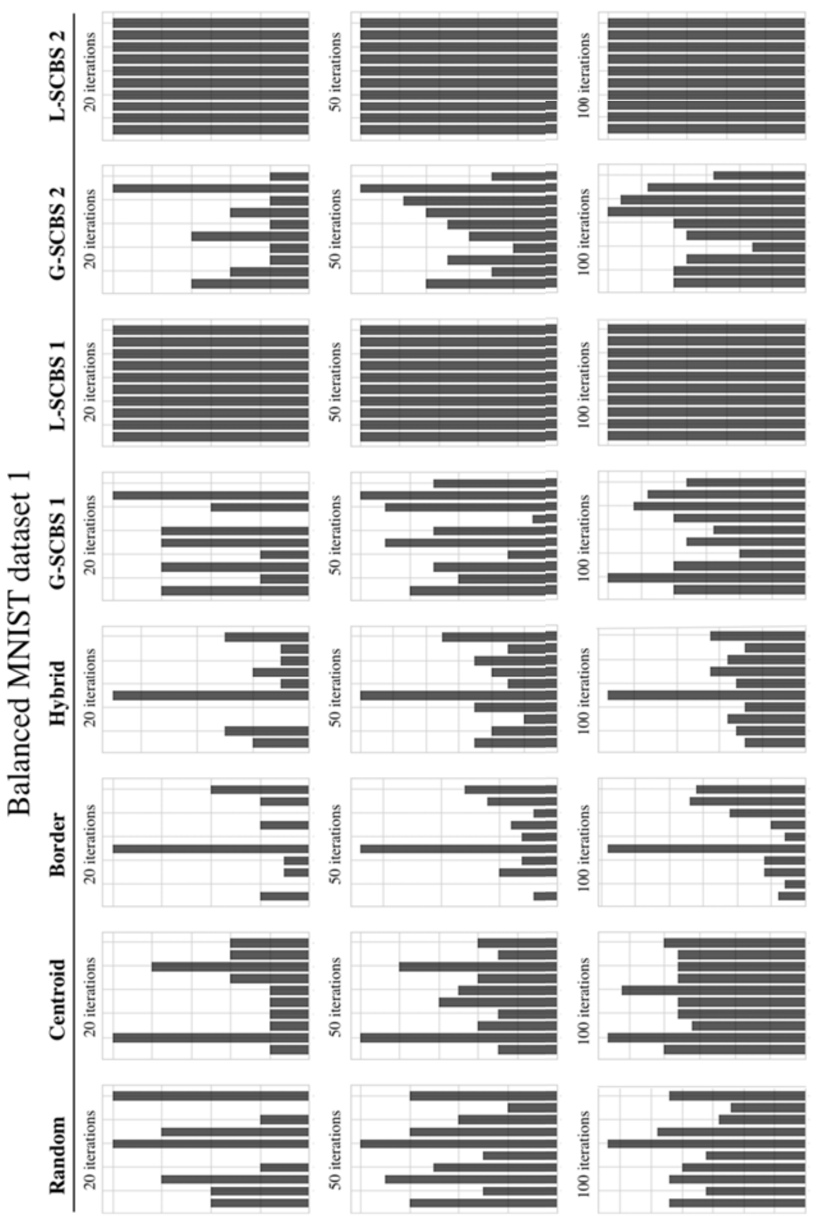


Figure 2: Distributions of the labeled instances among the categories got on one of the Balanced MNIST dataset at iterations 20, 50 and 100.

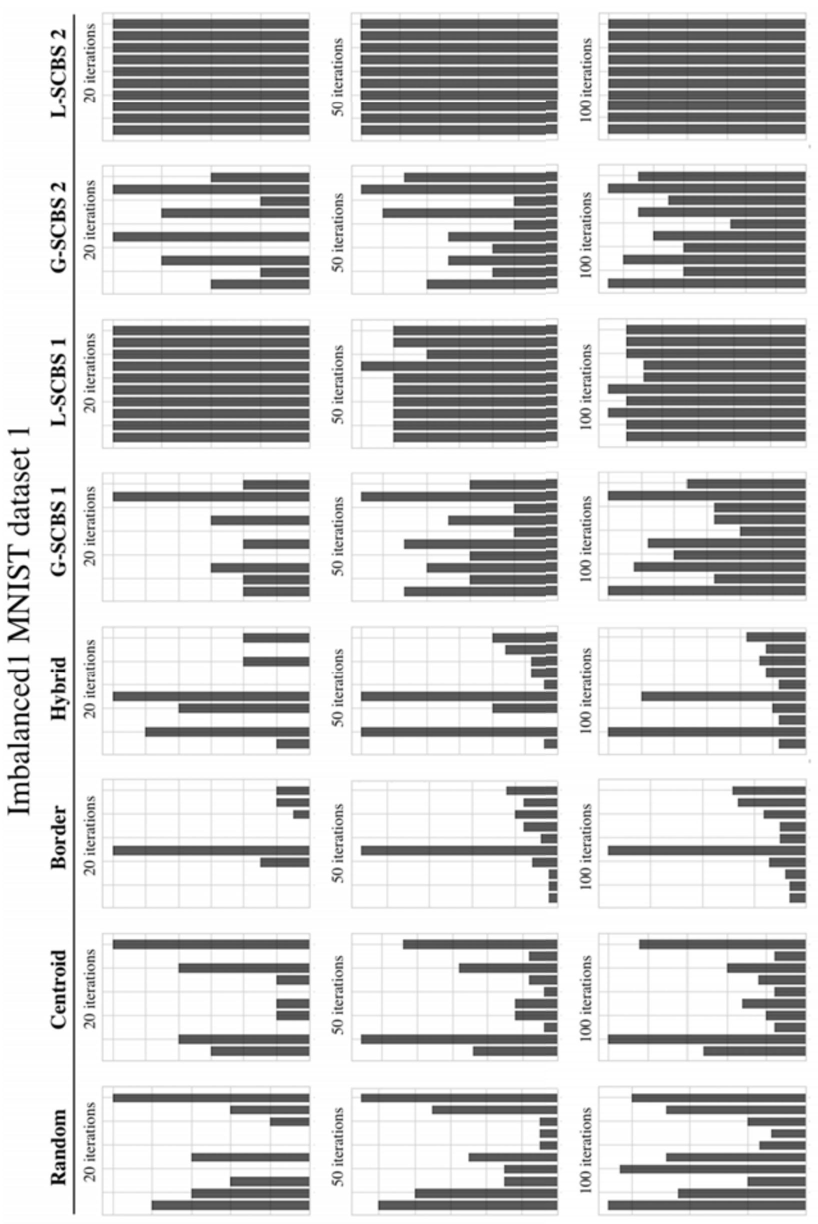


Figure 3: Distributions of the labeled instances among the categories got on one of the Imbalanced1 MNIST dataset at iterations 20, 50 and 100.

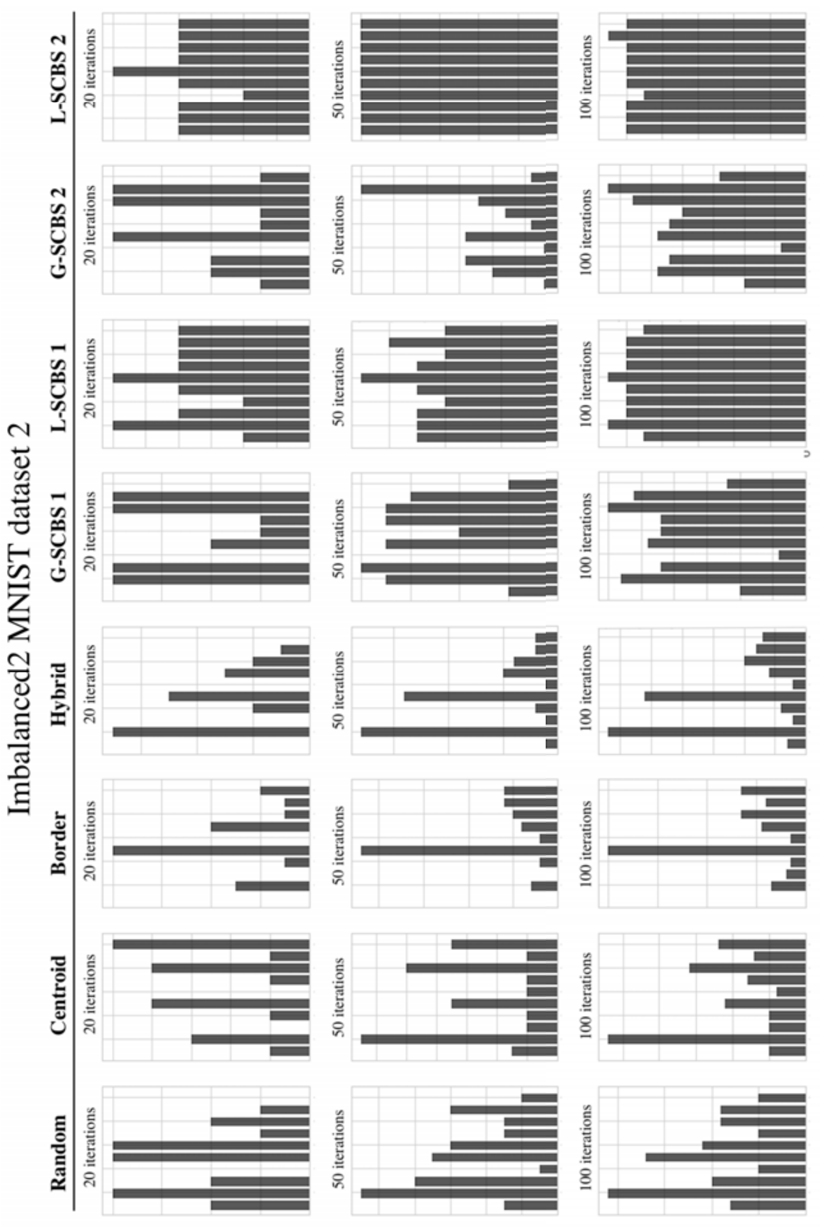


Figure 4: Distributions of the labeled instances among the categories got on one of the Imbalanced2 MNIST dataset at iterations 20, 50 and 100.

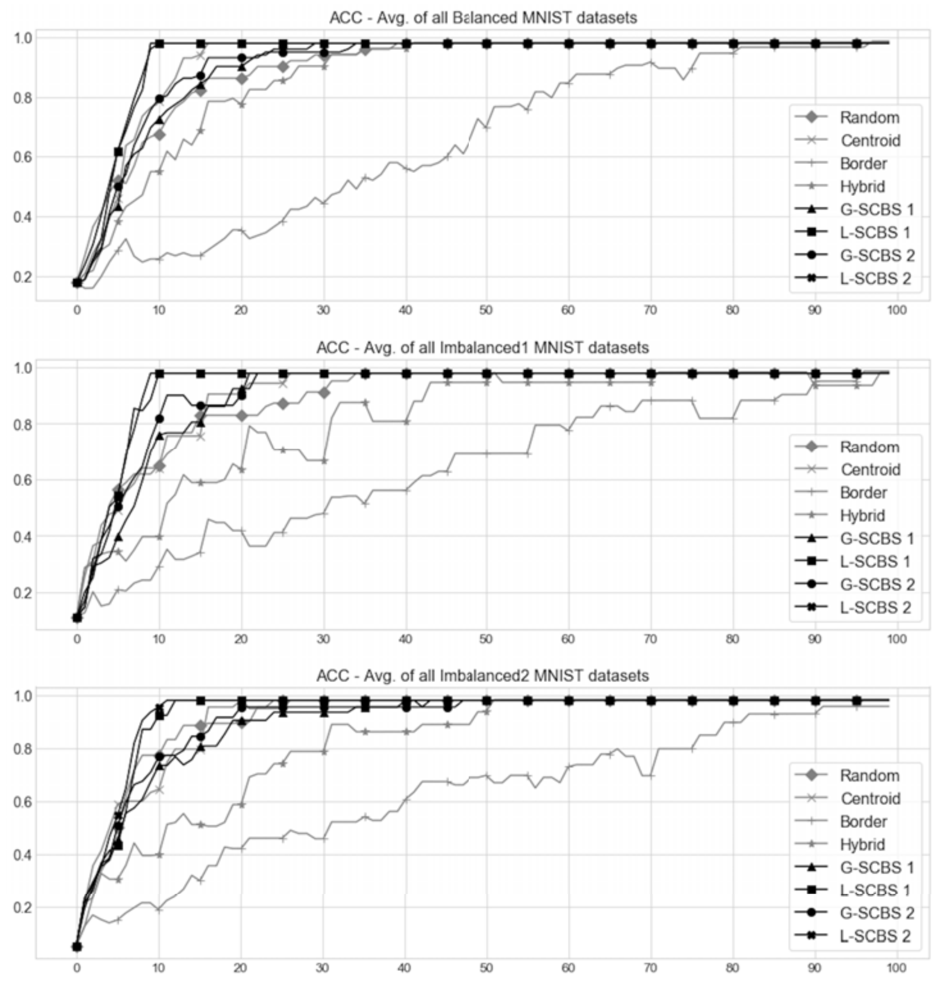


Figure 5: Average of accuracies got on the MNIST datasets at each active learning iteration; different query strategies are denoted by different markers, additionally, the darker lines correspond to the SCBS variants.

As can be seen on the figures, different datasets resulted different label distributions in L, however, taking the average of the different MNIST datasets regarding this aspect would be highly misleading and difficult to interpret. The reason for this is that the outcome of taking the average of low and high cardinality numbers could be around the perfect result, even though the difference between the individual results could be colossal. Consequently, each test were evaluated separately and deviations from the optimal cardinality number were calculated as errors. Table 2 summarizes the results, where the maximum and minimum $\{N_j\}$ are shown along

Table 2: Maximum, minimum cardinality numbers, and Average Cardinality Errors got on each MNIST dataset, additionally, last row of each block show the Average Actual Balancedness.

		Random	**Centroid**	**Border**	**Hybrid**	**G-SCBS 1**	**L-SCBS 1**	**G-SCBS 2**	**L-SCBS 2**
MNIST B 1	max	16	14	29	23	15	**10**	15	**10**
	min	6	8	3	7	5	**10**	4	**10**
	ACE	2.2	1.4	6.6	3	2	**0**	2.2	**0**
MNIST B 2	max	14	12	22	17	13	**10**	14	**10**
	min	8	8	4	6	7	**10**	6	**10**
	ACE	1.6	1	3.2	3.4	1.8	**0**	2.2	**0**
MNIST B 3	max	21	14	25	18	14	**10**	13	**10**
	min	4	7	4	7	6	**10**	7	**10**
	ACE	3.8	2	4.6	2.2	2.2	**0**	1.6	**0**
MNIST B 4	max	13	12	16	19	14	**10**	20	**10**
	min	4	9	5	5	6	**10**	5	**10**
	ACE	2.2	0.8	3.6	3.4	2	**0**	3	**0**
MNIST B 5	max	14	15	28	18	15	**10**	16	**10**
	min	1	8	5	7	6	**10**	5	**10**
	ACE	2.8	1.2	5	3	3	**0**	2.4	**0**
AAB		0.899	0.948	0.814	0.890	0.907	**1.000**	0.905	**1.000**
MNIST I1 1	max	17	25	38	30	15	11	13	**10**
	min	3	4	3	4	5	9	5	**10**
	ACE	4.6	5.8	7	7	3	0.4	2	**0**
MNIST I1 2	max	17	24	37	31	22	**11**	21	**11**
	min	5	4	2	4	5	**9**	5	**9**
	ACE	2.6	5	7.4	6.8	3.6	**0.2**	3.2	**0.2**
MNIST I1 3	max	16	27	24	31	21	11	20	**10**
	min	1	3	6	5	3	9	4	**10**
	ACE	3.6	6.2	3.4	5.6	4.2	0.4	3.8	**0**
MNIST I1 4	max	17	21	23	31	18	11	16	**10**
	min	3	5	2	3	2	9	2	**10**
	ACE	4	4.4	5.6	5.2	4	0.4	3.4	**0**
MNIST I1 5	max	15	24	21	30	15	**11**	14	**11**
	min	5	5	5	5	5	**9**	3	**9**
	ACE	2.6	4.8	3.6	4.8	3.2	**0.4**	2.4	**0.4**
AAB		0.870	0.800	0.750	0.736	0.858	0.980	0.870	**0.992**
MNIST I2 1	max	16	23	15	31	16	11	17	**10**
	min	5	4	6	5	2	9	2	**10**
	ACE	2.8	6.2	2.8	4.8	3.6	0.2	3.8	**0**
MNIST I2 2	max	21	27	40	32	15	**11**	16	**11**
	min	5	4	0	2	2	**9**	2	**9**
	ACE	3.8	5.2	7.2	7.6	3.4	0.4	3.2	**0.2**
MNIST I2 3	max	20	27	22	30	17	13	23	**10**
	min	1	4	5	3	3	9	2	**10**
	ACE	4.8	3.8	5	6.2	4	**0.6**	3.6	**0**
MNIST I2 4	max	23	23	43	31	18	11	18	**10**
	min	3	4	2	2	2	9	2	**10**
	ACE	5	5.4	8.6	7.6	5.2	0.4	5	**0**
MNIST I2 5	max	33	21	34	29	18	12	17	**10**
	min	3	4	3	4	4	9	3	**10**
	ACE	5.6	5.2	6.4	6.6	3.2	0.6	3.6	**0**
AAB		0.808	0.798	0.724	0.726	0.858	0.974	0.840	**0.996**

with the ACE for each MNIST dataset (indicated in the left column, where B, I1 and I2 refers to the type of MNIST dataset). Furthermore, the AAB measure was calculated for each query strategy, and presented in the last row of each block of Table 2. As can be seen in the table, L-SCBS 1 and L-SCBS 2 had zero deviation from the optimal distribution in all balanced cases, in addition, L-SCBS 2 could achieve perfect balance, even in imbalanced situations, while L-SCBS 1 performed marginally worse; as the values of the AAB metric shows. Therefore, L-SCBS 2 is the best method (among the tested ones) to employ for the zero initialized active learning task.

6 Conclusion

A novel active learning query strategy framework and an acitve learning query strategy that belongs to this framework were elaborated in this paper, the Clustering Based Balanced Sampling Framework (CBBSF) and the Spectral Clustering Based Sampling (SCBS), respectively. CBBSF focuses on the problem of zero initialized active learning, hence it selects the initial labeled training dataset and balances the items among the categories. The framework consists of three modules, (i) a clustering module, (ii) an assignment module and (iii) a selection module. SCBS realizes this framework, it utilizes ClusterGAN integrated in spectral clustering process to form the clusters and then Hungarian method is used during the assignment, after that it selects unlabeled items based on the class membership probabilities. Global and local variants of the SCBS method were developed, futhermore, two different techniques were applied to calculate the informativeness of the unlabeled instances, and thus four different SCBS approaches were examined. Average Cardinality Error (ACE) and Actual Balancedness (AB) new measures were introduced in the paper. During the experimental evaluation on MNIST datasets, ACE, AB and accuracy (ACC) were evaulated using each SCBS variant, moreover, state-of-the-art zero initialized active learning query strategies were also tested and compared to the results of SCBS, namely the Random, Centroid, Border and Hybrid approaches. The results showed that local versions of SCBS achieve high accuracy faster than every other method, and they are able to perfectly balance the labeled training dataset. In future work, the proposed approach will be extended with a solution that handles wrong clustering, i.e., when two categories are merged or one category is splitted. With this addition, the usability of the algorithm in real world scenarios could be improved significantly.

References

[1] B., Settles and M., Craven. An analysis of active learning strategies for sequence labeling tasks. *In Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1069–1078, 2008. DOI: `10.3115/1613715.1613855`.

[2] Cai, W., Zhang, Y., and Zhou, J. Maximizing expected model change for active learning in regression. *In: IEEE 13th International Conference on Data Mining*, pages 51–60, 2013. DOI: `10.1109/icdm.2013.104`.

[3] Cebron, N. and Berthold, M. R. Active learning for object classification: from exploration to exploitation. *Data Mining and Knowledge Discovery*, 18(2):283–299, 2009. DOI: `10.1007/s10618-008-0115-0`.

[4] Chowdhury, M. and Faisal, M. Disease mention recognition with specific features. In *In Proceedings of the 2010 workshop on biomedical natural language processing*, pages 83–90, 2010.

[5] Chung, F. R. and Graham, F. C. *Spectral graph theory*, volume 92. CBMS, Philadelphia, 1997. DOI: `10.1007/978-3-642-58058-1`.

[6] D., Lewis and W., Gale. A sequential algorithm for training text classifiers. In *In Proceedings of the ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 3–12, 1994. DOI: `10.1007/978-1-4471-2099-5_1`.

[7] Gavves, E., Mensink, T., Tommasi, T., Snoek, C. G., and Tuytelaars, T. Active transfer learning with zero-shot priors: Reusing past datasets for future tasks. *In Proceedings of the IEEE International Conference on Computer Vision*, pages 2731–2739, 2015. DOI: `10.1109/iccv.2015.313`.

[8] HU, P. Spectral clustering survey. Technical report, The Chinese University of Hong Kong, 2012.

[9] Hu, R., Mac Namee, B., and Delany, S. J. Off to a good start: Using clustering to select the initial training set in active learning. *In Twenty-Third International Florida Artificial Intelligence Research Society Conference (FLAIRS 2010)*, pages 26–31, 2010.

[10] Kang, J., Ryu, K. R., and Kwon, H. C. Using cluster-based sampling to select initial training set for active learning in text classification. In *In Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 384–388. Springer, 2004.

[11] Kang, J., Ryu, K. R., and Kwon, H. C. Using cluster-based sampling to select initial training set for active learning in text classification. *In Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 384–388, 2004. DOI: `10.1007/978-3-540-24775-3_46`.

[12] Kuhn, H. W. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1–2):83–97, 1955. DOI: `10.1002/nav.3800020109`.

[13] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. DOI: `10.1109/5.726791`.

[14] Mac Aodha, O., Campbell, N., Kautz, J., and Brostow, G. Hierarchical subquery evaluation for active learning on a graph. *In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 564–571, 2014. DOI: `10.1109/cvpr.2014.79`.

[15] Mukherjee, S., Asnani, H., Lin, E., and Kannan, S. Clustergan: Latent space clustering in generative adversarial networks. *In Proceedings of the AAAI Conference on Artificial Intelligence*, 33:4610–4617, 2019.

[16] Panda, N., Goh, K. S., and Chang, E. Y. Active learning in very large databases. *Multimedia Tools and Applications*, 31(3):249–267, 2006. DOI: 10.1007/s11042-006-0043-1.

[17] Papp, D. and Szűcs, G. Balanced active learning method for image classification. *Acta Cybernetica*, 23(2):645–658, 2017. DOI: 10.14232/actacyb.23.2.2017.13.

[18] Papp, D. and Szűcs, G. Extended margin and soft balanced strategies in active learning. *In European Conference on Advances in Databases and Information Systems*, pages 69–81, 2018. DOI: 10.1007/978-3-319-98398-1_5.

[19] Papp, D., Szűcs, G., and Knoll, Zs. Difference based query strategies in active learning. *In Proceedings of the IEEE 17th International Symposium on Intelligent Systems and Informatics (SISY 2019)*, pages 35–39, 2019. DOI: 10.1109/sisy47553.2019.9111587.

[20] Settles, B. Active learning literature survey. Technical report, University of Wisconsin-Madison Department of Computer Sciences, 2009.

[21] Souza, V., Rossi, R. G., Batista, G. E., and Rezende, S. O. Unsupervised active learning techniques for labeling training sets: an experimental evaluation on sequential data. *Intelligent Data Analysis*, 21(5):1061–1095, 2017.

[22] Szűcs, G. Decision trees and random forest for privacy-preserving data mining. *In Research and Development in E-Business through Service-Oriented Solutions*, pages 71–90, 2013. DOI: 10.4018/978-1-4666-4181-5.ch004.

[23] Szűcs, G. and Henk, Z. Active clustering based classification for cost effective prediction in few labeled data problem. *Academy of Economic Studies. Economy Informatics*, 15(1):5–13, 2015.

[24] Szűcs, G. and Tamás, B. Body part extraction and pose estimation method in rowing videos. *Journal of computing and information technology*, 26(1):29–43, 2018. DOI: 10.20532/cit.2018.1003802.

[25] Tsai, Y.L., Tsai, R.T.H., Chueh, C.H., and Chang, S.C. Cross-domain opinion word identification with query-by-committee active learning. *In: Cheng, S.M., Day, M.Y. (eds.) TAAI 2014. LNCS*, 8916:334–343, 2014. DOI: 10.1007/978-3-319-13987-6_31.

[26] Von Luxburg, U. A tutorial on spectral clustering. *Statistics and computing*, 17(4):395–416, 2007. DOI: 10.1007/s11222-007-9033-z.

[27] Xie, S. and Philip, S. Y. Active zero-shot learning: a novel approach to extreme multi-labeled classification. *International Journal of Data Science and Analytics*, 3(3):151–160, 2017. DOI: 10.1007/s41060-017-0042-5.

[28] Xie, S., Wang, S., and Yu, P. S. Active zero-shot learning. *In Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, pages 1889–1892, 2016. DOI: `10.1145/2983323.2983866`.

[29] Yang, Y., Ma, Z., Nie, F., Chang, X., and Hauptmann, A.G. Multi-class active learning by uncertainty sampling with diversity maximization. *Int. J. Comput. Vis.*, 113(2):113–127, 2015. DOI: `10.1007/s11263-014-0781-x`.

[30] Yuan, W., Han, Y., Guan, D., Lee, S., and Lee, Y. K. Initial training data selection for active learning. *In Proceedings of the 5th International Conference on Ubiquitous Information Management and Communication*, page 5, 2011. DOI: `10.1145/1968613.1968619`.

[31] Zhu, X. *Semi-Supervised Learning with Graphs*. PhD thesis, Carnegie Mellon University, 2005.

Acta Cybernetica **25** (2021) 421–434.

Energy-Efficient Routing in Wireless Sensor Networks*

Dániel Pásztor[ab], Péter Ekler[ac], and János Levendovszky[de]

Abstract

Efficient data collection is the core concept of implementing Industry4.0 on IoT platforms. This requires energy aware communication protocols for Wireless Sensor Networks (WSNs) where different functions, like sensing and processing on the IoT nodes is supported only by local battery power. Thus, energy aware network protocols, such as routing, became one of the fundamental challenges in IoT data collection schemes. In our research, we have developed a novel routing algorithm which aims at increasing the lifetime of the IoT network subject to pre-defined reliability constraints. Assuming that the data is split and transmitted in the form of packets, we seek the optimal paths over which packets can reach the Base Station (BS) with effective energy usage subject to the condition that the probability of successful packet arrival to the BS exceeds a pre-defined threshold (reliability parameter). As far as the radio propagation is concerned we use Rayleigh-fading in our model. The new algorithm will guarantee an increased longevity and information throughput of the network due to the efficient energy balancing in the IoT network. The performance of the new protocol has also been studied and confirmed by simulations.

Keywords: IoT, WSN, energy-efficient, routing, WiFi

*The research reported in this paper was supported by the BME Artifcial Intelligence TKP2020 IE grant of NKFIH Hungary (BME IE-MI-SC TKP2020) and by the Ministry of Innovation and the National Research, Development and Innovation Office within the framework of the Artificial Intelligence National Laboratory Programme. Project no. FIEK 16-1-2016-0007 has been implemented with the support provided by the National Research, Development and Innovation Fund of Hungary, financed under the Centre for Higher Education and Industrial Cooperation – Research infrastructure development (FIEK 16) funding scheme.

[a]Department of Automation and Applied Informatics, Budapest University of Technology and Economics, Hungary

[b]E-mail: `daniel.pasztor@aut.bme.hu`, ORCID: `0000-0002-7565-3941`

[c]E-mail: `peter.ekler@aut.bme.hu`, ORCID: `0000-0002-2396-3606`

[d]Depatment of Networked Systems and Services, Budapest University of Technology and Economics, Hungary

[e]E-mail: `levendov@hit.bme.hu`, ORCID: `0000-0003-1406-442X`

DOI: `10.14232/actacyb.288351`

1 Introduction

Industry 4.0 is the newest stage of the fourth industrial revolutions, where the primary objective is digital data acquisition and performance enhancement of complex manufacturing processes by using the concept of "digital twins". This performance enhancement requires a number of different sensors and communication equipments to measure and transmit the information obtained about the underlying industrial process.

In most of the cases, connecting each sensor to a wired network proves to be physically infeasible, thus wireless IoT devices can provide an efficient solution for controlling the observed industrial process. The wireless nodes transmit the collected data in the form of packets to a Base Station (BS) where the complex processing of data and system evaluation is carried out. This also gives flexibility as additional sensors can easily be added to or redundant nodes can be removed from the network as needed. We refer to this combination of the sensor and an IoT device with wireless transceiver as a *node* in the forthcoming discussion.

Unfortunately, wireless devices need to be powered by built-in batteries which need to be recharged periodically, if possible. Under these circumstances, network longevity and energy efficiency becomes a driving force when one wants to maximize the throughput of IoT networks.

The power consumption of these devices can be divided into two main categories: the energy required to operate the sensor and the energy required to transmit data. The energy consumption of the sensor depends on several parameters, such as the consumption of the electrical components, the operation mode of the sensor and various parameters of the controlling electronics. In contrast, the energy required for reliable communication can be well defined and typically depends on the distance between the communicating nodes and the environmental noise and this energy used for communication is by far the most significant in energy consumption.

For this reason, it may often prove to be disadvantageous for a particular device to send its message directly to the BS due to the increased energy consumption of long distance communication. Instead, it may be useful to implement multi-hop packet transfers from the sender node to the BS via some relay nodes, thus in this paper, we develop a novel routing algorithm for packet transfer that ensure the extended lifetime of the network.

The rest of the paper is organized as follows. In Section 2 the related work is summarized. In Section 3 the model is defined. Section 4 introduces the two-hop and k-hop algorithms with numerical performance evaluation. The results of the algorithm are shown in Section 5. Section 6 concludes the paper and proposes further research directions.

2 Related Work

In the literature, several different algorithms have been proposed for efficient wireless communication in IoT networks.

LEACH [3] assigns nodes to be cluster heads periodically whose responsibilities are to collect the messages in their region. After compressing the received packets into a single message, every cluster head transmits its message to the base station, which is considered to be farther away from the nodes. If every node were to send their message directly to the base station, the energy required to transmit over the longer distances would quickly deplete every node. Collecting the messages in a region by the cluster heads have low energy requirements (because nodes are generally closer to each other), and this allows the node heads to compress multiple messages, decreasing the payload size. Due to these observations, the sensor network will stay alive longer in this scenario.

While the gathering and compressing of messages was a novel idea at the time, several new algorithms have been proposed with the aim of increasing the efficiency of LEACH. In [11], the authors suggest two modification to the original algorithm. In one algorithm, dubbed *energy-LEACH*, the cluster head selection algorithm was changed from the original random selection to selecting the nodes with the highest residual energy levels. Meanwhile, the other algorithm (called *multihop-LEACH*) changed the cluster head message routing: instead of directly sending the compressed message to the base station, the heads are allowed to send the message to other cluster heads. When implemented in simulations, both of these modifications showed better performance than the original LEACH algorithm.

Another modification for LEACH, called LEACH-B [9] (shortened from LEACH-Balanced), extends the cluster head selection algorithm. The authors based their modification on another paper [2], where it has been shown that with the original LEACH algorithm, the highest efficiency can be reached when for every round, around 3-5% of the nodes are selected to be cluster heads. LEACH-B introduces another round for the cluster head selection with the aim of making the number of cluster heads equal to this ideal number. Through simulations, the authors showed that this modification significantly extended the lifetime of the system.

Looking at another popular WSN routing algorithm, PEGASIS [6] creates a chain between the nodes close to each other using a greedy algorithm. In each round, the measured values from the nodes are aggregated and sent towards one particular node (the *leader* node) through the chain, which in turn transmits to the base station. This transmitting node changes every round. This means that at the end of the round, the base station does not receive every message sent by the nodes, rather only an aggregation of every measured value. This makes it efficient, but limits the use cases, and makes it harder to compare to other routing strategies.

Several improvements have been proposed for the original PEGASIS algorithm as well. Clustering has been used to break up the single, long chain into multiple shorter part based on the distance to the base station [4]. Through simulations, the authors showed an improvement of 35% in energy efficiency. EEPB (Energy-Efficient PEGASIS Based protocol) [5] and IEEPB (Improved EEPB) algorithms modify the chain creation algorithm by imposing a constraint on the maximal link distance between nodes, and change the leader selection algorithm to account for the residual energy levels.

There are many other routing algorithms in the context of Wireless Sensor

Networks, LEACH is usually preferred due to its simplicity. While other algorithms perform better compared to LEACH, they generally introduce significant overhead to the clustering and network maintenance stages, making them unideal for low powered IoT devices. A survey has been conducted which compares most of the available WSN routing algorithm [10].

The key difference between the research reported above and our work is that our solution achieves energy-awareness and extends network lifetime subject to meeting a pre-defined quality of service criterion.

3 The model

In our model, we consider the network as a 2D graph. Each node is stationary, meaning that the distances between any two nodes remain constant in time, and each node has a current residual energy level calculated as the starting energy level minutes the energy used up forwarding the past packets. For modeling the radio propagation, we use the Rayleigh-fading model, which gives us the connection between the transmission energy g_{ij} and the probability of successful packet transfer P_{ij} for nodes i,j, based on [1] in a zero-interference network.

$$g_{ij} = -d_{ij}^2 \frac{\theta \sigma_Z^2}{\ln P_{ij}} \tag{1}$$

where d is the distance between the communicating nodes, θ and σ are the parameters of the environment and communication.

The equation above can be simplified to the following relationship, as the nodes are stationary:

$$g_{ij} \ln P_{ij} = \omega_{ij} \tag{2}$$

where $\omega_{ij} = -d_{ij}^2 \theta \sigma_Z^2$, is a constant dependent on the distance between the nodes and the parameters of the environment. Because θ and σ_Z^2 are positive values, ω_{ij} will have a negative value. Each packet transmission requires this energy from the the sending node, and in this paper we assume that the receiving node can receive the packet without any energy consumption (in the future work the reception energy will also be taken into account). In order to ensure a given reliability, we also define the probability P_s as the probability with which the base station must receive the message sent by a node.

Our proposed routing algorithm works on the principle that nodes with higher energy levels are supposed to participate more frequently in packet forwarding by being relay nodes. In this way low energy nodes are allowed to have short distance communication on low energy with other nodes and there is no need to send their packets directly to the BS. In order to achieve this, we introduce a new property into our model, called Minimal Path Residual Energy (*MPRE* for short). This is defined over a given path between nodes on which a message is transmitted. For a given path, MPRE is the energy level of the node which has the least energy remaining after the message transfer is successfully completed. We would like to develop an algorithm which maximizes MPRE. This can be accomplished if the nodes participating in the packet transfer will have uniform remaining energies.

We prove this with a proof of contradiction: let us suppose that the energy levels after the routing are uniform, but the MPRE value is also maximized. Take the node with the most energy remaining. If this node was routing the message with more energy (without reaching the MPRE), the other nodes would be able to transmit the message with less energy usage to reach the same probability, increasing the value of MPRE. This contradicts our initial assumption, proving our statement.

4 The proposed algorithm

Based on the observation above, the objective of our algorithm is to make the residual energy of the nodes participating in the packet transfer as uniform as possible, while still satisfying the reliability constraint (guaranteeing that the packet will reach the BS with a pre-defined probability). We investigate the following routing strategies:

- *Direct sending*, i.e. the source node sends the packet directly to the base station without using an intermediate node. This is the simplest strategy which serves as a baseline.
- *2-hop* strategy, when an intermediary node may be used for sending the message to the base station.
- *K-hop* strategy, in which case at most k-1 intermediate nodes form the path for packet transfer.

The strategies are described by the next sections.

4.1 Two-hop routing

For the sake of simplicity, let us first assume that a single packet is forwarded to the BS over a two-hop path, and at time instant k a sender node denoted by index s sends this packet to the BS via a relay node denoted by index l. Knowing that after the transmission, both nodes must arrive at the same energy level, and that the successful transfer probability is P_s is given, we can write the following equation (where c is the common energy level after the transmission, c_s is the starting energy level of the source node and c_l is starting energy level of the relay node):

$$\frac{\omega_{s,l}}{c_s - c} + \frac{\omega_{s,BS}}{c_l - c} \geq \ln(P_s) \tag{3}$$

Arranging the equation for c will get us the following quadratic formula:

$$c^2 A + cB + C = 0 \tag{4}$$

where

$$A = \ln(P_s) \tag{5}$$

$$B = \omega_{s,l} + \omega_{l,BS} - (c_s + c_l)\ln(P_s) \tag{6}$$

$$C = c_s c_l \ln(P_s) - \omega_{s,l} c_l - \omega_{l,BS} c_s \tag{7}$$

where ω_{ij} is defined in the following way (from Section 3): $\omega_{ij} = -d_{ij}^2 \theta \sigma_Z^2$. Taking into account the constraints on the variables ($c >= 0$, $\omega < 0$), only one solution is possible:

$$c = \frac{-B + \sqrt{B^2 - 4AC}}{2A} \tag{8}$$

If $c < 0$, then the needed energy for the communication is higher than the nodes can provide, and so the packet can not be sent with the given confidence. Our strategy requires us to find an intermediary node which maximises the common energy level c. This can be done by calculating the resulting common energy level c for every possible intermediary node, and using the one with the maximum value to route the message through.

The algorithmic complexity of this strategy is $O(\mid V \mid)$.

4.2 K-hop routing

Calculating the solution for k-hop routing requires significantly more calculations. We have to calculate the optimal common energy level for a given set of the intermediary nodes, which requires finding the roots of a k-degree complete polynomial which satisfies the constraints, then we also have to check different paths containing k hops to find the one with the highest PMRE. Because of this computational complexity we rather develop an approximate solution for the problem.

This approximation can be broken up into two main parts. First, for a given node energy distribution vector c, let us find the highest probability with which a message can be sent between a chosen source node and the base station. Formally, this can be written in the following way:

$$\max_g \sum_{j=0}^{m} \frac{\omega_{j,j+1}}{g_{j,j+1}} \tag{9}$$

Since ω must be a negative number, we can see that for a given path, the maximum transmission probability can be reached if $g_{l_j l_{j+1}} = c_j$, meaning that every node along the path is using their remaining energy to send the message. Since we know the energy level of every node before the transmission occurs, we can calculate $\gamma_{j,j+1} \equiv \frac{\omega_{j,j+1}}{g_{j,j+1}}$, making the problem:

$$\max \sum_{j=0}^{m} \gamma_{j,j+1} = \min \sum_{j=0}^{m} -\gamma_{j,j+1} \tag{10}$$

which makes this problem equivalent to finding the shortest path in a graph with at most k edges, in which an edge between node j and $j+1$ have a weight of $-\gamma_{j,j+1}$.

This can be solved using the Bellman–Ford algorithm, and stopping after the kth iteration, which has the worst case complexity of $O(k \mid V \mid^2)$.

With this, we can approximate the optimal common energy level for k-hop routing the following way. Instead of every node sending with its remaining energy, let us choose a common energy level c_{common}, and the aim is that every node participating in the transmission reaches this energy level after the transmission. This gives us the energy for every node with which they can participate in the transmission: $g_{j,j+1}(k) = c_j - c_{common}$, from which the previously presented approach gives us the maximum transmission probability.

Looking at the relation between the chosen common energy level and the maximum transmission probability, we can intuitively see that if we lower the energy level, the transmission probability rises since nodes can use more energy in the transmission. Because of this, we can use binary search over the interval $(0, c_s)$ for the common energy level where the maximum transmission probability reaches the given $\ln(P_s)$. This gives us an approximate solution for the optimal common energy level with complexity $O(\mid V \mid^2 \ln \frac{c_s}{\delta c_{common}})$, where δc_{common} is the maximum guaranteed absolute error between the optimal and approximated solution.

The presented approach will give us an approximate solution for the k-hop routing strategy.

5 Numerical results

To evaluate the performance of our energy efficient algorithm, we used the following simulation environment: Our wireless sensor network consists of N stationary nodes and a BS collecting the messages sent by the nodes. The location of the nodes are chosen randomly in a unit square. An example of a random network can be seen on Figure 1.

We have chosen the number of nodes to represent 3 different sizes:

1. 10 nodes for a *small* network
2. 100 nodes for a *medium* network
3. 1000 nodes for a *large* network

Every node starts with the same initial energy. The parameters of the Rayleigh-fading model were chosen to mimic real-life circumstances. We have chosen the probability value for successful message transfer to be 95%.

In every step of the simulation, the source node is selected randomly, which transmits a message towards the base station subject to the given probability criterion. We repeat this procedure until a node depletes its energy, and count the messages received by the base station.

To make comparisons between LEACH and other strategies, the simulation consists of separate rounds. In every round, every node in the network must send one message to the base station. For LEACH, this is the original algorithm, meaning no modification is needed. For our energy-efficient algorithm, the order of nodes

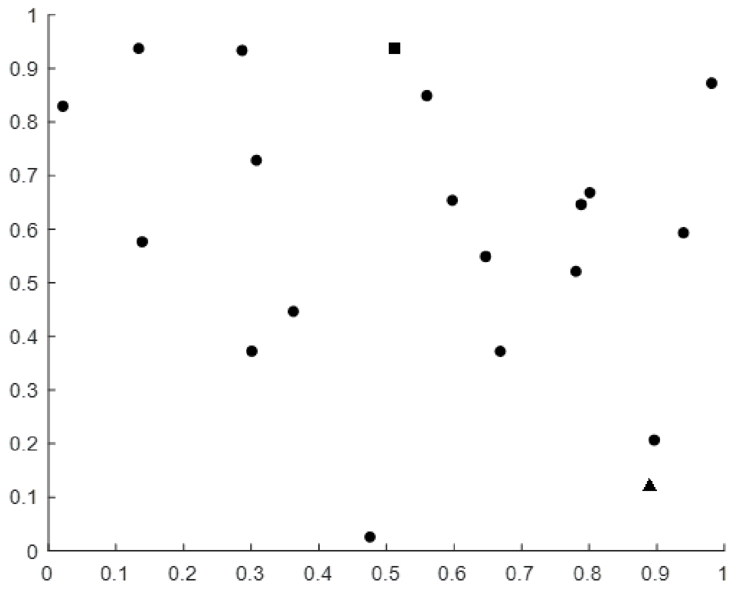


Figure 1: An example of one WSN. The square is the base station, while the triangle is a node currently sending a message.

sending their message is randomized in a given round. Like previously, we run as many rounds as possible before one node depletes its energy, and count the messages received by the base station.

We implemented the LEACH algorithm as discussed in the original paper [3]. We have chosen the probability for a node to become a cluster head each round to be 15%.

The simulation environment and proposed strategies were implemented in MATLAB. During our research we have not found sources for the exact values of the θ and σ_Z^2 environmental parameters. However, these parameters have no impact on the relative performance of one strategy compared to another. Looking at equation 1, we can see that increasing the product of these parameters by a ratio of x would increase the required energy as well, meaning that on average, the sent message count would also decrease by this ratio as well when sending with the same transfer probability. In our simulations, we have estimated the product of θ and σ_Z^2 to be 2.14 based on the transmission parameters of the nrf24l01 chip, but this will be refined in future work. We have generated a hundred different random network topologies, and for each topology we have run the simulation ten times. We average the results over these runs.

For our first set of simulations, we are only comparing the 2-hop, 3-hop, 4-hop and 5-hop strategies to each other.

As can be seen on Figure 2, under these circumstances, the two-hop routing performed better than either the direct routing or the k numbered strategies ($^*k > 2$).

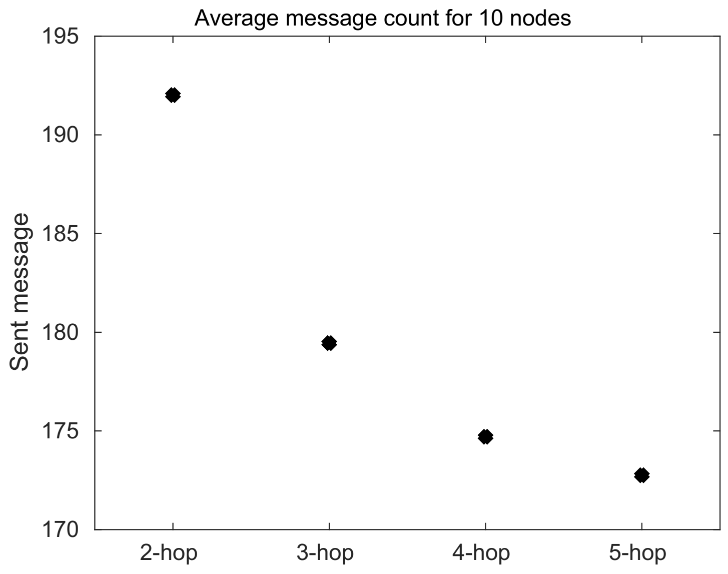


Figure 2: Result for small network

Compared with the direct routing, the two-hop strategy can make use of an intermediary node, so nodes farther away or with lower energy are able to conserve their energies. This is in contrast with the results of higher k numbered strategies, where the extended use of more intermediary nodes leads to shorter lifetime. Examining the energy levels after each message, we concluded that while the remaining energy levels are indeed higher compared to the two-hop strategy, the use of multiple nodes results in an overall higher energy usage which depletes the network faster.

To give a simple example of this effect, let us suppose that we have a network with ten nodes, each having 4 unit of energy. After choosing a random source node, we run the different strategies on this network, and we find that for two-hop, the optimal solution is sending the message through one intermediary node to the base station with 4 units of energy, while with 4-hop, the optimal solution is sending the message through 4 nodes with 3 unit of energy. In this case, two-hop used 8 units of energy, while 4-hop used a total of 12 energy unit. This results in higher overall energy usage for 4-hop, draining it faster.

For the medium sized network with 100 nodes, 2-hop still performed better with regards to the average message count seen on Figure 3. However, 3-hop is closer to the 2-hop result, under performing only by 4%.

For our final simulation, we ran the network with 1000 nodes. In this case, Figure 4 shows that 3-hop strategy outperformed the 2-hop routing by around 3%.

Looking at the results of different network with regards to the network size, we can conclude that as the network size increases, higher numbered k algorithm will become more efficient.

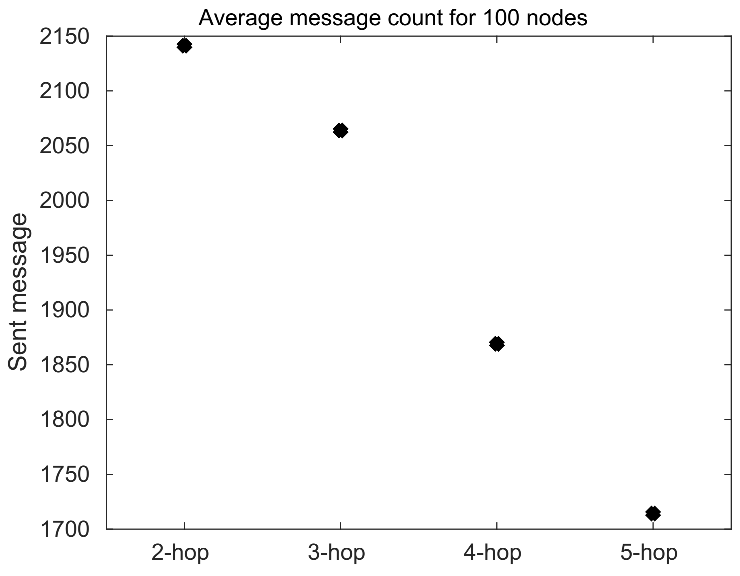


Figure 3: Result for medium network

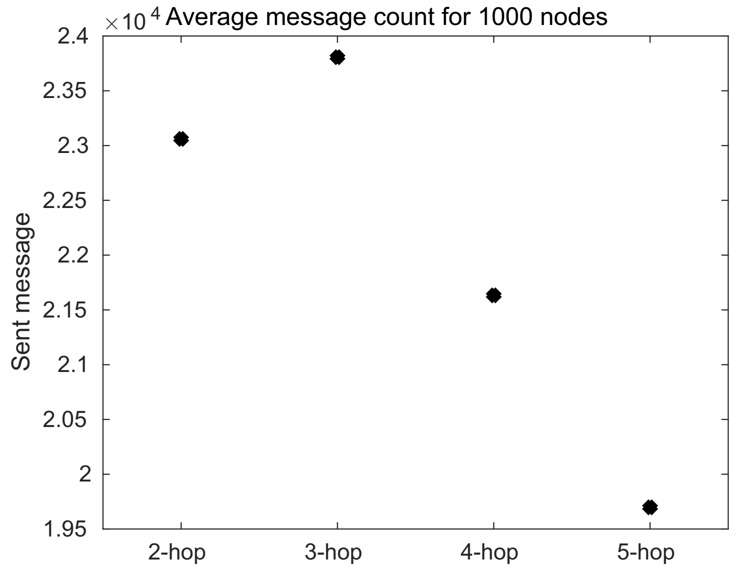


Figure 4: Result for large network

For our next set of runs, we compare LEACH with the proposed strategies as well as with direct sending.

As can be seen on Figure 5, LEACH under performs even when compared to direct sending. This can be explained by the fact that originally, LEACH was proposed for networks where the base station is farther away from the nodes. Also, LEACH is typically run until every node in the network dies, while we are focusing on the first dead node.

Looking at the energy-efficient strategies, we can see slightly higher results compared to the previous simulations. This is because of the way we run the simulations with LEACH: due to the presence of rounds, the selected nodes are more evenly spread out around the network (due to the fact that in every round, every node will be selected exactly once for message sending). This spreads out the energy differences more evenly, increasing the longetivity of the network.

Looking at the results for medium networks in Figure 6, we can see the same results as with the small network.

In the result for large networks seen on Figure 7, the other strategies still outperform LEACH by a huge margin. Another interesting point that can be seen is that in this case, 2-hop still performed better compared to 3-hop, while in the previous runs, 3-hop outperformed 2-hop in large networks.

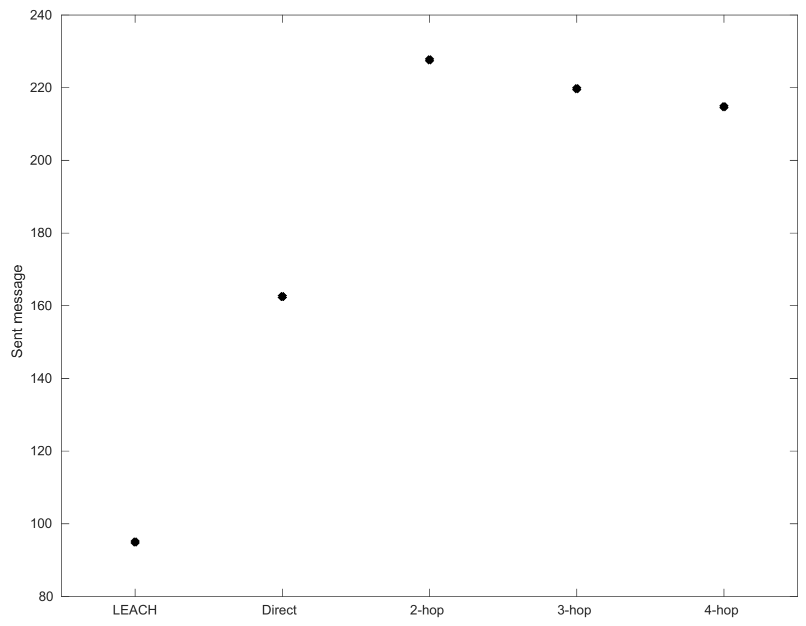


Figure 5: Result for small network with LEACH

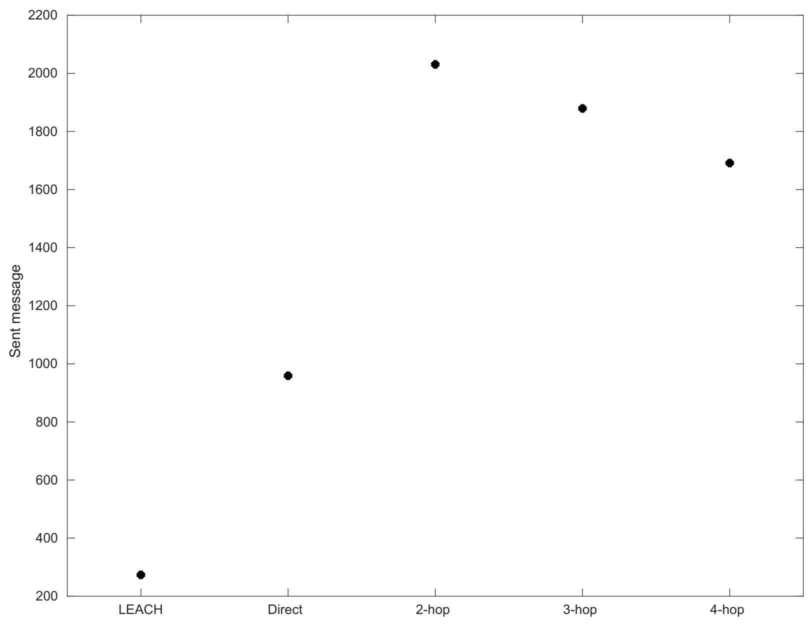


Figure 6: Result for medium network with LEACH

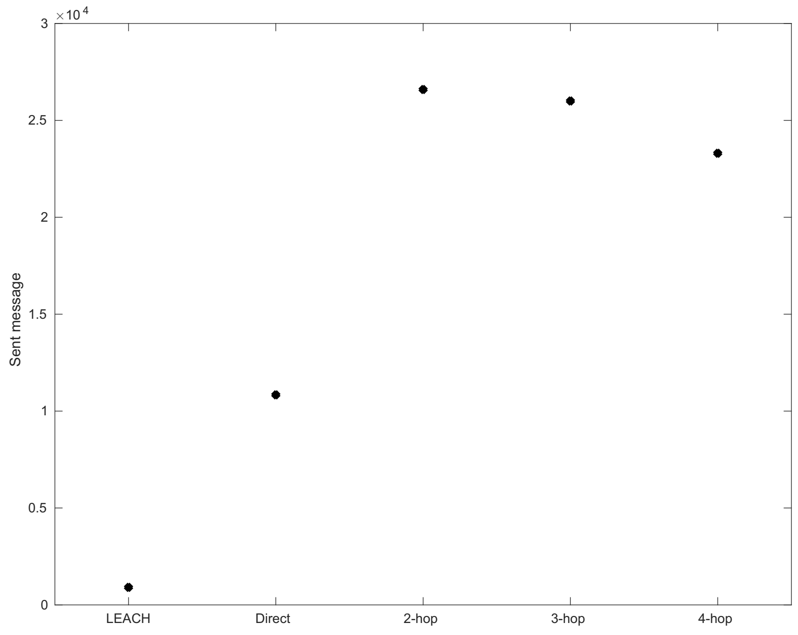


Figure 7: Result for large network with LEACH

6 Conclusion and future works

In this paper we have developed a novel routing algorithm for energy aware IoT data communication where the successful packet transfer from the nodes to BS is guaranteed with a given level of reliability. As the performance analysis have revealed, 2-hop performed well for any network size, while 3-hop outperformed 2-hop in a large network.

As can be seen, LEACH performed poor due to multiple reasons, such as in our environment, the base station was placed near the other nodes, while LEACH was proposed with the base station being placed farther away. Also, the original LEACH does not take into account the current energy levels of the nodes, for a low energy node to become cluster head instantly depletes that node. Since we are running the simulations until the first node goes flat (i.e. runs out of battery power) , this also results in lower performance. We also plan on making comparisons with other, improved versions of LEACH such as E-LEACH [8] and ME-LEACH [7], where the residual energy is taken into account for cluster head selection.

Our planned future work will take into account the network topology, as well. The results given above were achieved with randomly placed nodes but we can further tailor them if the network topology is known in advance. In the future, we would like to consider special network topologies including indoor transmission, as well as different packet sending frequencies, when optimizing the routing algorithm. The model can be further expanded by introducing barriers between nodes (such as buildings). We plan to apply the findings of our research in the wireless sensor network deployed at ZalaZone (being a test environment for autonomous vehicles).

References

[1] Haenggi, M. On routing in random rayleigh fading networks. *IEEE Transactions on Wireless Communications*, 4(4):1553–1562, 2005. DOI: `10.1109/TWC.2005.850376`.

[2] Heinzelman, W. B., Chandrakasan, A. P., and Balakrishnan, H. An application-specific protocol architecture for wireless microsensor networks. *IEEE Transactions on Wireless Communications*, 1(4):660–670, 2002. DOI: `10.1109/TWC.2002.804190`.

[3] Heinzelman, W. R., Chandrakasan, A., and Balakrishnan, H. Energy-efficient communication protocol for wireless microsensor networks. In *Proceedings of the 33rd Annual Hawaii International Conference on System Sciences*, pages 10 pp. vol.2–, 2000. DOI: `10.1109/HICSS.2000.926982`.

[4] Jung, S., Han, Y., and Chung, T. The concentric clustering scheme for efficient energy consumption in the pegasis. In *The 9th International Conference on Advanced Communication Technology*, volume 1, pages 260–265, 2007. DOI: `10.1109/ICACT.2007.358351`.

[5] Li, T., Ruan, F., Fan, Z., Wang, J., and Kim, J. An improved pegasis protocol for wireless sensor network. In *2015 3rd International Conference on Computer and Computing Science (COMCOMS)*, pages 16–19, 2015. DOI: `10.1109/COMCOMS.2015.20`.

[6] Lindsey, S. and Raghavendra, C. S. Pegasis: Power-efficient gathering in sensor information systems. In *Proceedings, IEEE Aerospace Conference*, volume 3, pages 3–3, 2002. DOI: `10.1109/AERO.2002.1035242`.

[7] M. Abdurohman, Y. Supriadi and Fahmi, F. Z. A modified e-leach routing protocol for improving the lifetime of a wireless sensor network. *Information Processing Systems*, 16:845–858, 2020. DOI: `10.3745/JIPS.03.0142.`

[8] Patel, H. B. and Jinwala, D. C. E-leach: Improving the leach protocol for privacy preservation in secure data aggregation in wireless sensor networks. In *2014 9th International Conference on Industrial and Information Systems (ICIIS)*, pages 1–5, 2014. DOI: `10.1109/ICIINFS.2014.7036607`.

[9] Tong, M. and Tang, M. Leach-b: An improved leach protocol for wireless sensor network. In *2010 6th International Conference on Wireless Communications Networking and Mobile Computing (WiCOM)*, pages 1–4, 2010. DOI: `10.1109/WICOM.2010.5601113`.

[10] Watteyne, T., Molinaro, A., Richichi, M. G., and Dohler, M. From manet to ietf roll standardization: A paradigm shift in wsn routing protocols. *IEEE Communications Surveys Tutorials*, 13(4):688–707, 2011. DOI: `10.1109/SURV.2011.082710.00092`.

[11] Xiangning, F. and Yulin, S. Improvement on leach protocol of wireless sensor network. In *2007 International Conference on Sensor Technologies and Applications (SENSORCOMM 2007)*, pages 260–264, 2007. DOI: `10.1109/SENSORCOMM.2007.4394931`.

Acta Cybernetica **25** (2021) 435–468.

Traquest Model — A Novel Model for ACID Concurrent Computations*

Dániel B. Rátai[ab], Zoltán Horváth[ac], Zoltán Porkoláb[ad], and Melinda Tóth[ae]

Abstract

Atomicity, consistency, isolation, and durability are essential properties of many distributed systems. They are often abbreviated as the ACID properties. Ensuring ACID comes with a price: it requires extra computing and network capacity to ensure that the atomic operations are done perfectly or they are rolled back.

When we have higher requirements on performance, we need to give up the ACID properties entirely or settle for eventual consistency. Since the ambiguity of the order of the events, such algorithms can get very complicated since they have to be prepared for any possible contingencies. Traquest model attempts to create a general concurrency model that can bring the ACID properties without sacrificing a too significant amount of performance.

Keywords: ACID, concurrency, consistency, atomicity, concurrency model, fault tolerance

1 Introduction

In the case of the microservices architecture [5] when we send a request, it can initiate some modifications in the global state in a transactional way. Microservices are mainly based on the request-response model. When the Request returns with no errors, that means the modifications in the global state are done, and the transaction is over. While if the response is an error, that means there were no

*The research was supported by the ÚNKP-20-3 New National Excellence Program of the Ministry for Innovation and Technology from the source of the National Research, Development and Innovation Fund. The research is part of the "Application Domain Specific Highly Reliable IT Solutions" project that has been implemented with the support provided from the National Research, Development and Innovation Fund of Hungary, financed under the Thematic Excellence Programme no. 2020-4.1.1.-TKP2020 (National Challenges Subprogramme) funding scheme.

[a]Eötvös Loránd University, Faculty of Informatics, Budapest, Hungary
[b]E-mail: `danielratai@inf.elte.hu`, ORCID: `0000-0002-9587-571X`
[c]E-mail: `hz@inf.elte.hu`, ORCID: `0000-0001-9213-2681`
[d]E-mail: `gsd@inf.elte.hu`, ORCID: `0000-0001-6819-0224`
[e]E-mail: `toth_m@inf.elte.hu`, ORCID: `0000-0001-6300-7945`

DOI: `10.14232/actacyb.288765`

modifications in the global state at all. Requests can make other requests, and more complex transactions can be assembled.

In a request-response model, the ACID properties come at a high price. The service which calculates the response has to guarantee that any write operations arising must be synchronized, committed, and persisted before it can reply with a response. Of course, on the other hand, if ACID is not a requirement, the service can be very fast. In this case, the service can just read and write the state of the local server and answer to the client immediately. Later, the server can synchronize the writes if eventual consistency is a requirement, but this does not block or decelerate the original process.

However, it is not just the performance that can cause a problem. It is very challenging to ensure atomicity itself when we nest the services. Figure 1 shows a scenario where we have service A calling two other services, B and C. The order of the network events is marked on the figure. The client sends a request to service A, which sends requests to service B and C. Service B responds correctly, but C responds with an error. In this case, we can assume that C has rolled back correctly, but B should be rolled back as well. There is no mechanism to roll back a request in the request-response model after it has been responded. Therefore it is hard to chain more services properly when atomicity is a requirement. We can be sure to have a proper response if the happy path happens, but if there is an error arising at some of the chained requests, our system can get easily stuck into an invalid intermediate state.

It seems there is a hard dilemma between ACID properties and efficiency. The proposed Traquest model attempts to resolve this dilemma and therefore improve the efficiency of the ACID systems.

The phrase Traquest [28] comes from the words Request and Transaction. The core of the idea comes from the microservices architecture, and the Traquest model

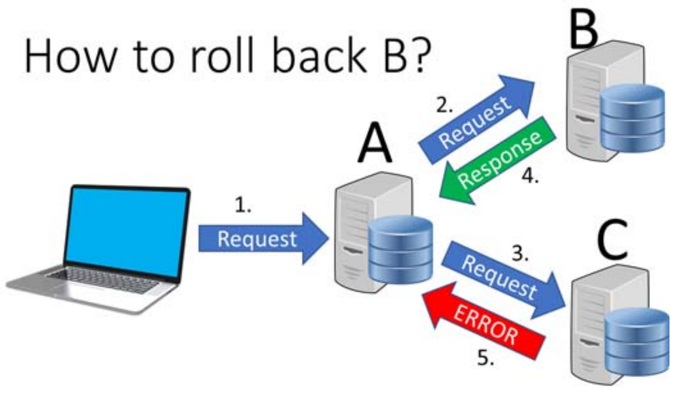


Figure 1: Nested rollback issue

is something similar to the request-response model. We can send requests to a Traquest, and the Traquest replies with an answer, but here the answer is not a simple response message, but rather an established parent-child connection between the two Traquests with a temporary response, a so-called Trasponse. When a Traquest gets a request, it can immediately carry out read and write operations on the local server. It can immediately reply with a Trasponse; however, of course, that still might take time to synchronize the effects of the operations with other servers. Therefore Trasponse is only a temporary response.

Before creating the Traquest model, we have examined many of the existing technologies and solutions, including multitier architectures, actor model based systems, different consistency protocols, and different papers discussing the limitations of distributed ACID systems. We have found that the current systems have the strict limitations we described above. We decided to investigate whether it is possible to create a system based on the idea that a response can have a temporary nature. The Traquest model was realized during this research process.

We created the concept of the Traquest model, and we also built an experimental prototype in TypeScript. Adjustments on the model might become necessary later as further, and more comprehensive implementations will be created in different programming languages. However, the current results show that the general concept of the Traquest model is viable. Traquests can provide ACID computations using magnitudes fewer network messages in some concurrency scenarios than the current technologies.

This paper is structured as follows. In Section 2, we give an explanation of the Traquest model. In Section 3, we discuss some state-of-the-art solutions and how the current technologies were used to solve problems related to the ACID properties. We explain further the Traquest model through an exemplary case and compare it to the current technologies. In Section 4, we will highlight the current challenges and further research directions. Finally, this paper concludes in Section 5.

2 The Traquest model

The request-response model is used on a local level as well and not only between different computing nodes. Asynchronous callback functions can behave equivalently. We can send the request content and the callback function as an argument, and the callback function can contain the response in an argument. This mechanism is often used to wrap network-based request responses, but for local asynchronous operations as well.

However, callbacks can get complicated when they are heavily used, and we want to handle exceptional scenarios. To this end in computer science, *Future*, *Promise*, *Delay*, and *Deferred* refer to constructs used for synchronizing program execution in some concurrent programming languages. They describe an object that acts as a proxy for a result that is initially unknown, usually because the computation of its value is not yet complete. The term *Promise* was proposed in 1976 by Daniel P. Friedman and David Wise [9] and Peter Hibbard called it *Eventual* [16]. A somewhat similar concept *Future* was introduced in 1977 in a paper by Henry Baker and Carl Hewitt [3].

Traquests behave most similarly to *Promises*; therefore, we use them as a baseline for the explanation. Traquests, just like Promises, are placeholders for a temporarily unknown value. Traquests, just like Promises, can be nested and depend on each other. However, once a Promise returns with a response, this response is final, and it cannot be modified afterwards. On the other hand, Traquests can be strongly bonded together to form a tree structure, a so-called Traquest tree. A Traquest tree creates the transaction, and if any Traquest fails in the Traquest tree, all the Traquests are failing. When the Traquests are failing, they are not just returning an error, but they are ensuring that if they created any modification, it would be appropriately rolled back so that the global state of the system will not be affected by half-done transactions. To be able to achieve this, Traquests are containing some additional mechanisms.

2.1 Structure

To understand how Traquests are working, first, we need to see the fundamental structure of the state of art Promises.

2.1.1 Promises

Figure 2 shows the fundamental structure of Promises. Deferred describes a yet unfinished work which is the asynchronous process that has to be done to get the

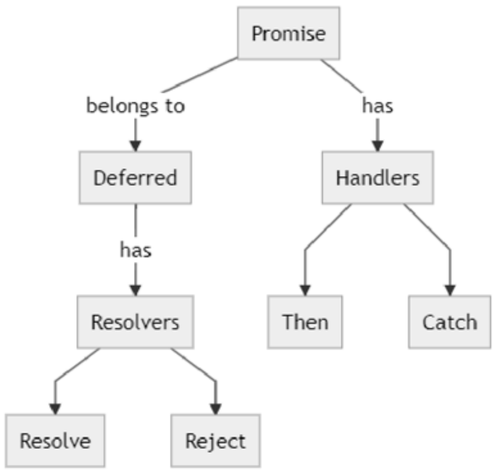


Figure 2: Promise structure

final value of the Promise. A Promise belongs to a Deferred. When the Deferred finishes, it can call different resolvers depending on whether the execution was successful or some exceptions were arising. If the execution was successful, the Deferred calls the Resolve resolver; otherwise, it calls the Reject resolver. The Promise itself is the placeholder of the yet unknown value. It has two handlers to handle the event when the unknown value becomes known. The Then handler is responsible for handling the successful resolution of the Promise, and the Catch handler is for handling the exceptions.

2.1.2 Traquests

Figure 3 show the fundamentals structure of Traquests. A Traquest also belongs to a Deferred that, similarly to Promises, describes a yet unfinished work. Traquests have handlers just as Promises to handle the event when the Deferred returns. However, Traquests has a third significant component as well, the Binding mechanism. The binding can permanently bind together Traquests in a parent-child tree structure. This binding holds until the whole atomic transaction finishes. Like that, a Traquest tree can act as a single entity, and it can form a complete atomic transaction, which can be distributed to many computing nodes.

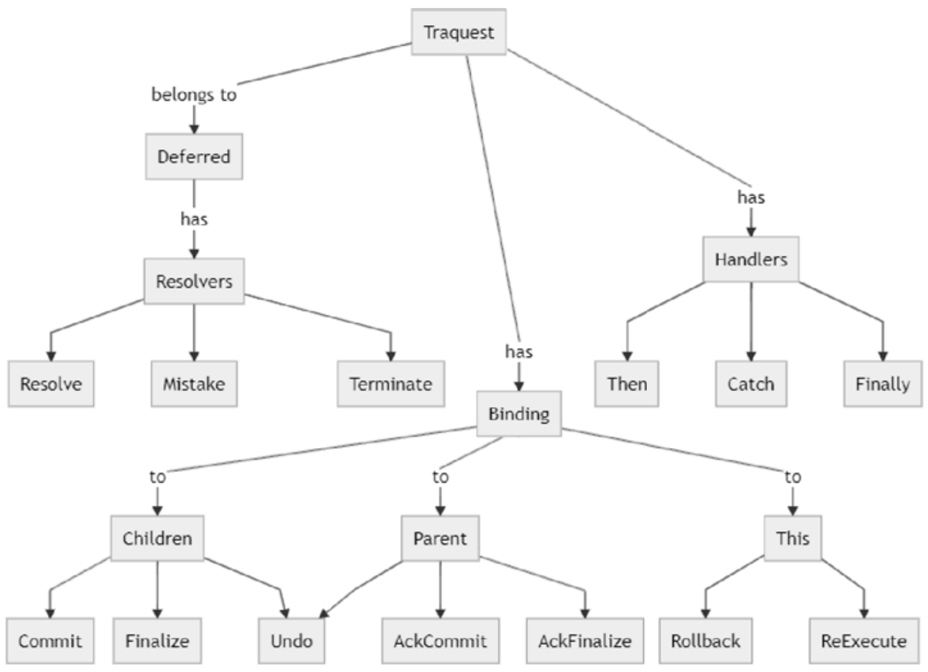


Figure 3: Traquest structure

The Deferred has the following resolvers. The Response resolver is the same as the Resolve resolver at the Promises. This should be executed when the asynchronous Deferred process successfully finishes with the proper value. The Mistake resolver is slightly different from the Reject resolver of the promises. The Mistake is called when a temporary failure happens. If there is a chance that the failure has occurred only because of the wrong order of the asynchronous operations, then Mistake should be triggered. Mistakes can be undone later, and the Traquests might rerun in proper order. The Terminate resolver is used in case of final failures. This resolver terminates the whole Traquest tree and tries to roll back all the Traquests in the Traquest tree.

The Then handler is the same as the Then handler of the Promises. The Catch handler is similar to the Catch handler of the Promises, but it is used explicitly for the mistakes. It can also avoid spreading up the Mistake to parent Traquests or let it spread further. The Finally handler is called no matter if the Traquest was properly committing or it was terminated.

The Binding mechanism of the Traquests has the following concepts:

Parent-child binding – When a Traquest had been created, the reference to the parent Traquest should be defined. If it is not defined, that means the created Traquest will be the root of the Traquest tree.

Undo – A mistake happens when an exception occurs because the Traquests are executed out of order. However, it can happen that the Traquest has already responded with a seemingly correct response, and an out-of-order conflict turns out only later. In this case, an undoing mechanism can be executed, which rolls back the necessary Traquests on the affected branch of the Traquest tree and re-executes them.

Rollback – A callback is provided for the case when the Traquest needs to revert the changes it has made so far. If the Traquest did not create any changes directly to the global state, just by calling other Traquests, this part could be omitted because the rollbacks spread automatically on the Traquest tree.

Finalizing – It is a mechanism used when all the Traquest in the tree have returned, and the result of the Traquests can be finalized. This happens completely hidden and automatically when all the Traquests in the tree have returned.

Committing – It is a mechanism used when all the Traquests in the tree have been finalized, and a final commit can be initiated. This happens completely hidden and automatically. The Committing mechanism combined with the Finalizing mechanism gives a similar process to the two-phase commit protocol [35]; however, there are differences because the Finalizing and Finalized states are handling the potential rollbacking Tail Traquests as well.

2.2 States

Promises and Traquests have different states throughout their life-cycle, which describes their behaviour.

2.2.1 Promises

Figure 4 is a state diagram that shows the possible states of a Promise. In the case of the Promises, we have three very simple states. We have an Unfulfilled or Pending state while the Deferred process is running, and the Promise value is not known. From this state, the Promise can step only to Fulfilled or Rejected state. This happens when the Deferred process finishes depending on whether an exception was arising or not.

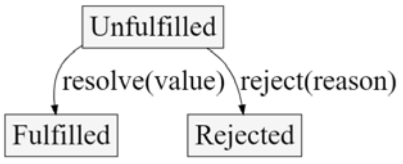


Figure 4: Promise state diagram

2.2.2 Traquests

Figure 5 shows the simplified state diagram of the Traquests. Traquests have to handle more complex scenarios; therefore, they can have more different states. For the sake of simplicity, the Terminated state and some state transitions are not

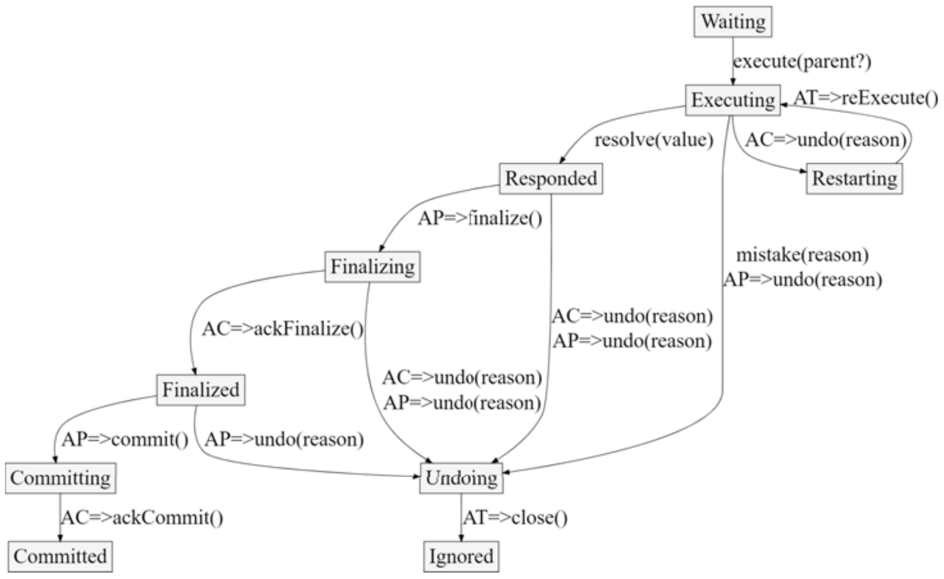


Figure 5: Simplified Traquest state diagram

shown here, only the most relevant ones which are necessary to understand the working mechanism of the Traquests. In Figure 5 the explanation for the state transition prefixes are the followings:

- **AC**: Automatically triggered by the child Traquest
- **AP**: Automatically triggered by the parent Traquest
- **AT**: Automatically triggered by the current ("this") Traquest
- **No prefix**: Manually triggered in the Deferred process

As one can see, Traquests are much more complex and have much more states than Promises have. However, Traquests have only two main manual resolvers, just like the Promises. When we have nested Promises, and an exception occurs, we should ensure manually that the proper Reject resolver is called, and it is handled at the parent Promise [31]. Furthermore, the parent Promise should manually escalate the exception further by calling its own Reject resolver. Traquests, on the other hand, are bound together and escalate the Mistakes automatically. A traditional exception in the Deferred process can be automatically converted into a Mistake, or a Mistake can automatically come from a child Traquest. In either case, no manual intervention is needed. Most of the time, it is enough to define only the happy path in a Deferred process and call the Resolve resolver. Interestingly this means that even though Traquests are more complex than Promises, still using Traquests can be even more convenient. To understand more how Taquests work, let us investigate them state by state in more detail.

Waiting state: The Waiting state is the initialization state of the Traquest. In this state, the Traquest gets initialized by defining the Deferred process belonging to it. The Deferred process is not added automatically to the event-loop like in the case of the Promises [20]; instead, it has to be manually executed. During the execute call, we need to provide the parent Traquest optionally. If the parent Traquest is missing, the current Traquest will be the root of the Traquest tree; therefore, the current Traquest will represent the overall transaction.

Executing state: When the execute method is called on the Traquest, the Traquest begins to execute the Deferred process, and it steps into the Executing state. During this state, the Deferred process can create new child Traquests and bind them to the current Traquest. The transaction can grow this way recursively.

During the execution, the Traquest can be interrupted by parent or child Traquests. The interruption happens if a Mistake arises at the ascending or descending Traquests. If a parent Traquest has a Mistake, that means the current Traquest should be ignored, including its children. Therefore if the parent Traquest triggers an undo operation on the current Traquest, it changes its state to Undoing, and triggers an undo operation on all the descendant Traquests.

If a Mistake is coming from one of the children of the current Traquest, that means that the Mistake is still not escalated to upper levels in the Traquest tree, and the Traquest can try to re-execute itself. In this case, the Traqest steps into the Restarting state.

The Mistake can come not only from the bounded Traquests, but it can also happen locally in the Deferred process of the current Traquest. In this case, the Traquest should roll back its changes; therefore, it has to switch into Undoing state.

Suppose no Mistakes are coming from the bounded Traquests, and the current Traquest is not running into a Mistake either. In that case, the Deferred process should call the resolve operation with the proper resulting value when it finishes. When the resolve operation is called, that means the current Traquest has completed its desired asynchronous task, and it can respond with the required value; therefore, the Traquest should step into the Responded state.

Responded state: The Traquest can get into the Responded state only from the Executing state by the Deferred process calling the Resolve resolver. Before stepping to the Responded state, the Traquest sends the result value coming from the Resolve operation to the handler of the Traquest. The handler executes the callbacks bounded to the Traquest by the Then operation right after the Traquest steps into the Responded state. However, there is one exception. If the current Traquest is the root of the Traquest tree, then no handlers are called, and the Traquest steps into the Finalizing state immediately.

Otherwise, the Traquest waits in the responded state until the parent Traquest asks for a commit, or a Mistake is coming from any of the bounded Traquests. Parent Traquests obviously can still be in execution when the current Traquest responds. However, for synchronization purposes and keeping the system's overall consistency, children Traquests can still be in Executing state as well. We discuss this scenario in more detail in Section 2.6. This means that a Mistake can trigger an undo operation from any direction. When an undo operation is called on a Traquest, which is in a Responded state, it should not be able to commit anymore.

If no Mistakes are coming from any of the bounded Traquests, then the current Traquest waits until it gets a finalize operation from the parent Traquest. The Traquest can and should trigger a finalize on itself only if the current Traquest is the root Traquest.

Finalizing state: When a finalize operation is triggered on a current Traquest by the parent Traquest and the current Traquest is in the Responded state, the current Traquest switches to the Finalizing state and calls the finalize operation on its child Traquests. If there are no child Traquests, the current Traquest steps into Finalized state and sends an ackFinalize operation to the parent Traquest.

If the current Traquest is still in Executing state, it makes no action. In this case, when the Deferred process returns, the parent Traquest calls again the finalize operation on the current Traquest; hence the finalizing mechanism can continue. This scenario happens with the Tail Traquests, which we will discuss later in more detail.

An Undo operation can come from any bounded Traquests during the Finalizing state.

Finalized state: The Traquests step to Finalized state when they get an ackFinalize acknowledgement from all of their children. If the Traqest steps into the Finalized state, it calls an ackFinalize operation on its parent. If it has no parent – meaning that the current Traquest is the root of the Traquest tree – the

Traquest steps in to Committing state and sends a commit operation to all of its child Traquests.

Undo operation can come only from a parent Traquest during the Finalized state because this state means that no descendant Traquests are being in Executing state, which could serve as a root for launching the undoing chain.

Committing state: When a Traquest steps into the Committing state, it triggers a commit on its child Traquests and waits until it gets an acknowledgement from them. When all the children have responded with an acknowledgement using the ackCommit operation, the Traquest steps into the Committed state. If the Traquest has no children, it immediately steps to the Committed state without waiting for any other processes or Traquests.

Traquests cannot roll back if the Traquest has already stepped into the Committing state.

Committed state: When a Traquest gets acknowledgements from all the child Traquests, it steps into the Committed state. The Committed state is a final state, meaning that the life-cycle of the Traquest was finished.

At the Response state, we discussed that if the current Traquest is the root Traquest the callbacks bounded to the handlers are not called. The root Traquest calls any handlers only at the end of its life-cycle. Therefore, when the root Traquest enters the Committed state, it calls the callback bounded to the Then handler, responding the final result of the whole transaction.

Undoing state: When a Traquest enters the Undoing state, that means that all the modifications in the global state done by the current Traquest or its descendants should be rolled back. Traquests can enter the Undoing state from almost any state. The exceptional states are the Waiting, Committing, Committed, Ignored, and Terminated states. From Waiting, there would be no point in entering the Undoing state since without executing the Deferred process, there cannot be any modifications to be rolled back.

The Committing state is already part of the committing process. Here all the temporary values are finalized. Suppose any errors are happening at this phase. That means a more serious issue that can affect the consistency. Therefore rolling back cannot be an option from the Committing state.

The Committed, Ignored, and Terminated states are the final states of the Traqest, and they cannot be rolled back. In all the other states, except the five states mentioned above, stepping to the Undoing state is possible.

Stepping to the Undoing state can be triggered manually in the Executing state, during the execution of the Deferred process – we can use the mistake operation for this purpose – or it can be triggered automatically by the bounded Traquests. The bounded Traquests can use the undo operation for initiating a rollback and trigger the current Traquest to step into the Undoing state. However, when the current Traquest is still in Executing state, and an undo operation comes from its child, the Traquest can still be re-executed, and in this case, the Traquest will step into the Restarting state.

Traquests can spread up the undoing chain even if they already have responded. This is important because the undo operation means that the responded value is not valid anymore; therefore, the parent Traquest, which already consumed this value, should be conflicted.

Ignored state: The Traquest steps into the Ignored state from the Undoing state when it has finished calling all the undo operations on its bounded Traquests. The Ignored state is a final state, and it means the life-cycle of the Traquest has ended.

Restarting state: When the current Traquest gets an undo operation from its child Traquest while the current Traquest is still in the Executing state, the Deferred process of the current Traquest should be re-executed. In the Restarting state, an undo operation is called on each of the child Traquests, the rollback callback is called for the current Traquest – if it was defined – to roll back every state change that has been made so far. When the rollbacks are finished, a reExecute operation is called on the current Traquest, which sets back the state of the current Traquest to Executing, and re-executes the Deferred process.

Terminated state: The Terminated state was not discussed in depth so far, and it is missing from Figure 5, because it implies high complexity. Figure 10 in the Appendix shows all the possible states and state transitions a Traquest can have, including the Terminated state. The Terminate state can be initiated by calling the terminate operation of the Traquest. The Terminated state means that an unsolvable error has happened, and the system's consistency cannot be guaranteed.

When a Traquest enters the Terminated state, it tries to roll back itself and calls the terminate operation on all its bounded Traquests.

2.3 Timestamps

Traquests contain a logical timestamp of their creation to be able to resolve conflicts. The timestamp of a Traquest inherits all the timestamps of the ascendant Traqests. This means that the timestamp of a parent Traquest represents the logical time of its whole branch when compared with Traquests from other branches. The timestamp of an ascendent Traquest is always earlier than the timestamps of descendant Traquests. The order of the sibling Traquests is decided in the order of their creation.

To clarify how the order of the Traquests should be considered, Figure 6 shows two Traquest trees, and on the horizontal axis, their physical time of creation is represented. In this case, the physical order of the Traquests is the following: T1; T2; T3; T4; T5; T6; T7; T8; T9. However, since Traquests trees represent atomic operations, and the branches of the trees represent atomic sub-operations, the logical order of the Traquests cannot be equivalent to the physical order. The logical order of the Traquests in this particular case is the following: T1; T3; T6; T8; T4; T7; T2; T5; T9.

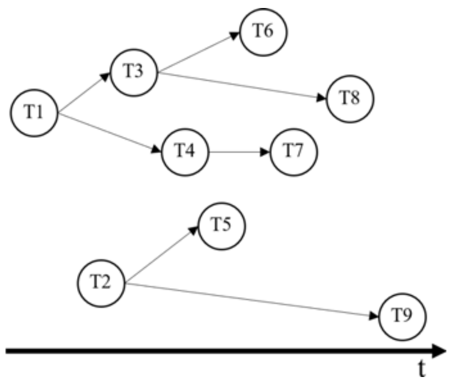


Figure 6: Order of Traquests

2.3.1 Current logical timestamps

There are already several solutions for creating timestamps and clocks in a distributed environment when more processes are running parallelly. The Lamport timestamp [27] algorithm or the Vector clocks [37] algorithm are designed to handle timing issues when more processes are existing at the same time which can interact with each other. In the case of the Lamport timestamp, creating the timestamp is very fast and easy but compare two timestamps is very hard and costs a lot of computation time. For the comparison, we also must decide if there is a joining path between the processes. It is not possible to use it for the Traquests. With the vector clocks, the result is similar. However, at the vector clocks, creating the timestamp is highly costly because we need as many dimensions for the clock as many processes we have. In our case, every Traquest is a process, and there can be millions of them or even more. Furthermore, Traquests can be created in real-time, which means at the point where we should give a timestamp for a Traquest, we do not even know how many processes should we consider to create a Vector Clock based timestamp. Therefore, the Vector clock works neither for the Traquest model.

2.3.2 Hierarchical timestamp

We need a hierarchical time stamping mechanism which is a more special case. The naive algorithm for creating a required hierarchical timestamp would be simply using an array with integers. All the Traquest can count how many children they have already, and whenever a child is created, they increase the counter; therefore, each child knows where they are in the queue of the order. The root Traquests can get their number from a global counter, from the Unix time, or a combination of the two. The array used as a timestamp can store all the ancestor's order numbers and also its own order number in the last record. Figure 7 shows the naive algorithm timestamps for the Traquest tree example presented in Figure 6.

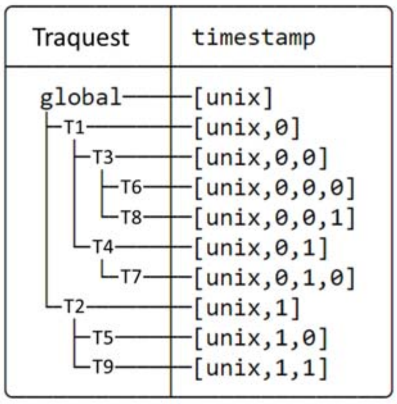


Figure 7: Naive hierarchical timestamps

In the example, we supposed that all the Traquests were created very close in physical time; therefore, they all have the same Unix time in the first record. After the global timestamp, all the timestamps contain their ancestors' timestamps; they are just extending it with their own order number. For flat hierarchies, it works well because there are only a few levels, and the length of the timestamp is short. However, the timestamps can get very long when the hierarchy is tall.

To address this issue, we have created a new timestamp algorithm that can reduce the necessary size of the timestamp drastically. Explaining our research on the optimized timestamps in more detail would be complex, and it is out of the scope of the current paper. Therefore, we rely on the naive timestamps for the explanation of the Traquest model.

2.4 Data protectors

We introduced that Traquests are forming a tree structure. However, a tree structure by itself would never result in any conflicts, which would be the core of the Traquest model to handle them effectively. Conflicts are happening when two different processes are trying to read or write the same part of the global state. To this end, Data protectors were constructed. Data protectors are entities responsible for managing a given segment of the global state. They protect the given global state particle from conflicting reads and writes.

The goal of each branch of the Traquest tree is to interact somehow with the global state. Therefore, each branch, at some point, ends up in a Data protector. Traquests can call CRUD [33] operations on the Data Protectors. When a Traquest calls a CRUD operation to a Data protector, the Data protector generates a new Traquest containing the operation and replies with the generated Traquest. This new Traquest can be bounded to the original Traquest as a child; therefore, it becomes part of the Traquest tree.

When more Traquests are using the same Data protector, the Data protector can use the logical timestamps of the Traquests to decide which read or write operation should be answered first. If a Traquest with an earlier timestamp comes after a Traquest with a later timestamp has been already responded to, the Data protector can call the undo mechanism of the already responded Traquest and serve the newly requesting Traquest. Therefore, the Data protector can efficiently resolve any conflicts.

Furthermore, because all the conflicts are recognized and resolved at the Data protectors, most of the conflict resolving features of the Traquests are used only by the Data protectors themselves. Data Protectors can trigger a Mistake if they have conflicting Traquests, and they can define the callback for the Rollback. This way, the increased complexity of the Traquests can be mostly hidden from the developers, and they do not need to care about the failure handling parts at all, except taking care of the Finally handler of the root Traquest. As a result, using Traquests can be as straightforward as using Promises or even more.

Data Protectors are the only entities who can contact directly to the global state storage. Therefore, the way we physically store the data can be abstracted away. Data Protectors can store the state using any databases, the local storage, the memory, or even any mixture of these solutions. Using traditional databases can be helpful to provide compatibility with other systems; however, in this case, we should be aware of the risk of corrupting the consistency of the global state. The most efficient solution is to use the local storage or memory for storing the state.

2.5 Consistency and Fault tolerance

Traquests interact with each other using serializable data constructs only. Therefore, Traquests can be located on different computing notes as well, and they still can interact. Fault tolerance requires the replication of the different particles of the global state to several computing nodes. Traquests are perfect for creating replicas of a desired global state particle and consistently managing them. It is enough to add new Traquest tree branches to each write operation that replicates the operation on different computing nodes. Thanks to the atomic property of the Traquest tree, the state will always remain consistent. For the read operations, we do not need such replication since the writes are already ensuring consistency.

2.6 Tail Traquests and network buffering

Tail Traquest is not a new separate feature in the Traquest model; it is instead a useful design pattern. Tail Traquests are simply Traquests where the Then handler of the Traquest is not defined. This implies that the parent Traquest of a Tail Traquest can finish its Deferred process without waiting for the current Traquest to finish with its task. This also means that the resulting value of a Traquest is independent of its child Tail Traquests.

Tail Traquests are primarily helpful for synchronizing the modifications in the global state made by the Traquest tree with other servers in a consistent way for reaching fault tolerance. Because Tail Traquests are not blocking the execution of the core logic of the Traquest tree, the overall Traquest tree can run very fast. Suppose every part of the global state that needs to be used is replicated locally. In that case, the whole logic of the Traquest tree can even execute in-memory time, and the network messages for the synchronizations are buffered automatically. They can be synchronized lazily in only two round trip messages for finalizing and committing. This optimization can happen even in the case of very complex algorithms, when there are many global state reads and writes depending on each other.

The communication of the Traquests between different computing nodes and the buffering can be separated and managed automatically. Therefore, the protocol for communication is abstracted away. The background implementation can use TCP [17], UDP [39], REST [21], WebSocket [8], WebRTC [18], long polling [19], SSE [36] or any other technologies what the infrastructure allows. This leaves many possibilities for optimizations. For example, if two nodes are communicating less frequently, they can use REST calls. However, if there are two nodes with frequent communication between them, they can switch to WebSocket. This way, the Traquest model attempts to create a new layer on top of the OSI layers [14], where the network communication itself can be abstracted.

2.7 A basic exemplary case

We have discussed how the basics of the Traquest model are working. To have a deeper understanding, let us examine how we can increment a simple integer value in the global state.

2.7.1 Basic scenario with no conflicts

To examine a basic scenario with no conflicts, let us discuss a simple incrementation of a value in the global state. In the Appendix Figure 11 illustrates such an example of an incrementation. The incremented global state variable is named i. The global process creates a Traquest for the transactional incrementation. A "T" prefix marks the Traquests, and their postfix is their logical timestamp. "DP_i" is the Data Protector of the i variable, and "Storage_i" is the physical location of the i variable. The "Storage_i" can be a local storage, it can be stored directly in the memory, or it can represent any kind of database as well. The sequence diagram notes are marking the actual states of the Traquests. The global process creates the "T1" Traquest, and the other two Traquests are generated by the Data Protector, one for reading the i variable and one for updating it. The diagram shows the operations between the entities. The initial value of the i variable is 10.

2.7.2 Conflict resolving

To examine how Traquests behave in a conflicting scenario, let us continue with a similar incrementation, but in this case, we have two conflicting global processes (e.g., two threads). In the Appendix Figures 12a and 12b illustrate such an example of a conflicting incrementation. The first global process begins an incrementation on variable i, and the second process begins a read on the same i variable only a little later. In this case, we have two Traquest trees with T1 and T2 Traquests at the root. The T1 tree performs a read on variable i, next the T2 tree performs a read, and after that, T1 performs the write with the incremented value. This is a conflicting scenario because the T2 tree should read the value of i only after the T1 has completely finished with the incrementation.

Figures 12a and 12b in the Appendix show that despite the conflict, the global processes get only the correct result at the end, and the correct final global state value is persisted on the storage. Examining more the "DP_i" Data Protector, it is also visible that the Data Protector reads and writes to the storage only once. It is interesting if we consider that it had to serve several CRUD operations coming from the Traquests. Data Protectors can effectively aggregate those CRUD operations and reduce the number of CRUD operations necessary to call on the storage itself. This has higher importance if we consider that the storage is a component that can be located on different computing nodes; therefore, calling an operation on the storage can be the slowest element of the overall process.

3 Related work

3.1 Current technologies

There are many technologies for providing ACID concurrent systems. Hereby we discuss the most relevant and most widely used directions.

3.1.1 Multitier architectures

The "Layers" architectural pattern has been described in various publications [6], and it is the most widely used pattern in the case of enterprise web applications. When the Business layer executes the desired algorithm, it continuously has to access the Data access layer to read the global state and write back the changed state. When the algorithm requires only a few iterations depending on each other with the Data access layer, this causes no problem. On the other hand, each read and write requires a roundtrip on the network when there are several depending steps. Although many databases – e.g., most of the SQL databases – can easily handle atomic transactions, the number of the necessary roundtrips implicates a massive limitation in the overall performance. This is a strict limitation in any architecture where we separate the location where we execute the business logic from the location where we store the global state.

To give an example, one might select any use cases where there are several dependent reads or writes to any databases to compare the multitier architectures. One of those examples is the Geographical Information System using large point clouds. For this purpose traditional PostgreSQL [23] is often used. Using space partitioning algorithms is a necessity to be able to manage the point cloud. Octree [22] is such an algorithm that can let us manage the points efficiently and easily.

Figure 8 shows an example when a new point is added to an Octree. We assumed that adding the new point requires searching down the Octree structure for ten levels. We assumed that the whole data set is replicated to two servers.

Database servers cannot execute algorithms in multitier architectures. They are only responsible for storing the data. The application is executed by the application server. Therefore, each node has to be first read to the application server from one of the database servers. Each node refers to its children; therefore, all the parents should be read before the child can be reached, and pipelining [29] cannot be used. Furthermore, to ensure atomicity and consistency, each node must be checked and locked on both servers. The example showed in Figure 8 is a simplified one. In real life, there can be much more and complex network messages between the servers. Still, even in this simplified case, we can count up to 44 network messages between the servers.

This example shows that any solutions built on using databases can have strict performance limitations if the operations sent to the database are depending on each other. If they are independent, buffering and pipelining can be used, and many queries can be sent within a single RTT (Round Trip Time) over the network. In this case, the number of network messages can be $O(1)$.

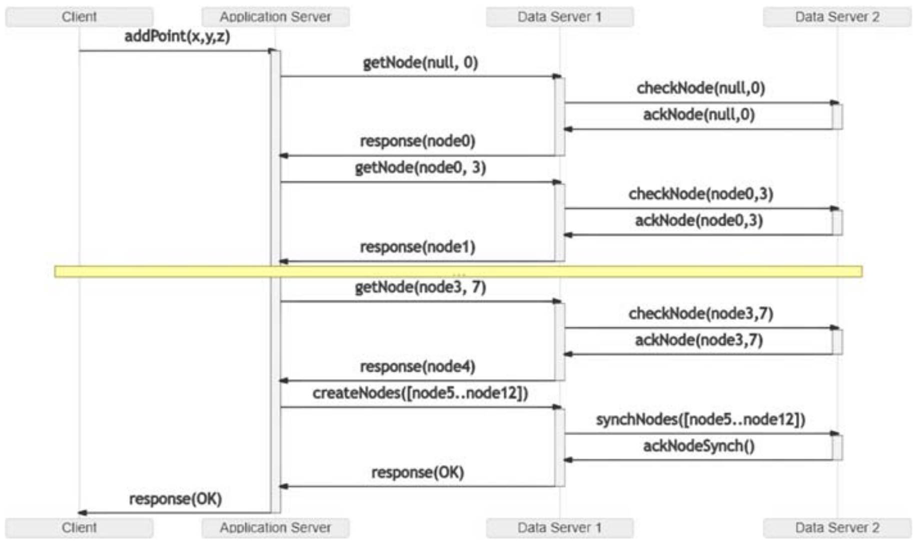


Figure 8: Add point to an octree in a multitier architecture

However, in many cases, we need to know the result of a query to be able to send the following query to the database. In such cases, the number of network messages grows on a $O(n)$ scale.

3.1.2 Serverless architecture

Serverless is one of the most trending architecture types nowadays. Serverless computing has emerged as a new compelling paradigm for the deployment of applications and services. It represents an evolution of cloud programming models, abstractions, and platforms and is a testament to the maturity and wide adoption of cloud technologies [4].

The serverless architecture is built on using stateless cloud functions in a managed way. The developer does not need to manage any server-side infrastructures. The number of cloud functions can scale horizontally automatically. These cloud functions can call other managed database systems to reach out to the application's global state. Serverless is getting more and more traction, and all the leading cloud providers are offering their serverless solutions: AWS Lambda [2], Google Cloud Functions [11], or Azure Functions [24].

However, in serverless architectures, the cloud functions have strict performance limitations thanks to the stateless nature of the cloud functions. The cloud functions cannot store any part of the global state. They need to call a database, a distributed file system, or other external services for that purpose. Therefore the cloud functions cannot be executed in-memory time, and they need to communicate on the network to finish their task. This means that from a performance perspective, they share the same limitations with the multitier architectures.

3.1.3 Actor model

Carl Hewitt first described the actor model in 1973 [15]. Actors can execute business logic and store state simultaneously; therefore, they do not share the limitations of the classical multitier architectures. Actors are excellent for solving problems where we have many independent processes that can work in isolation and only interact with other Actors through message passing. This model fits many problems. However, unfortunately, the actor model is not a favorable model for implementing truly shared state [34], when we need to have a consensus and a stable view of state across many components. Actors can get information about each other only through messages; therefore, it is hard to maintain atomicity.

The Actor model is a more general concept. Many algorithms can be implemented using Actors. Therefore we cannot directly compare the Actor model itself with the Traquest model because it depends on the algorithm that we create using the Actor model. However, there are standardized solutions for creating atomic transactions using the Actor model. The Akka toolkit suggests Transactors [34] for creating transactions. Under the hood, it uses a CommitBarrier, similar to a Java CountDownLatch [26] which is a blocking mechanism. Therefore, Transactors also have no specific timestamping mechanism. They have to lock and await each change in the global state; therefore, we would not have fewer network messages with the Transactors than what multitier architectures have.

3.1.4 Consistency protocols

The Traquest model can ensure atomicity and is also a promising way to ensure consistency. Therefore, hereby we take the most relevant consistency protocols [32] under investigation respective to the Traquest model.

Continuous consistency: Continuous consistency ensures that the numerical deviation of a specific global state particle does not go above a certain threshold on the different computing nodes. This can be applied only in the case of numerical values, and it ensures only an approximate consistency; therefore, it is out of scope for the Traquest model.

Primary-Based Protocols: Primary-Based protocols provide proper consistency for an arbitrary type of data. To keep the data consistent, they have to synchronize each write at least with the primary server. For instance, in such a case, the algorithm has to be blocked until a read it depends on gets a confirmation from the primary server. This requires many iterations of roundtrip messages; therefore, Primary-Based Protocol implies a strict limitation in the performance.

Quorum-Based Protocols: Quorum-Based Protocols have very similar limitations to Primary-Based Protocols. Each read and write operation must be confirmed by other computing nodes before the executed algorithm can rely on the operation and step forward. The only exception is the Read-One, Write-All scheme. In this case, it is enough to read the local state of the data; however, it requires even more messages to write synchronizations. This scenario can only be suitable in the case of very read-heavy applications.

The mentioned Primary-Based and Quorum-Based protocols share the same problems with the Transactors and multitier solutions. Each read or write should be crosschecked with other servers before we can rely on the locally stored data, and the locks are blocking the process. Therefore, these protocols cannot reduce the necessary network messages either.

3.2 Traquest model compared to the current technologies

In the following, we will discuss the exemplary GIS-octree case described in Section 3.1.1 to understand more and compare the Traquests.

With the Traquest model, we do not need a persistence layer because the Traquests themselves can already ensure the ACID properties. The data protectors can store the global state on the local storage or even in the memory. Therefore, there is no need for network communication for the reads in the case of an optimal topology. Moreover, write operations do not need network communication either, only for synchronization, which can be buffered and postponed. This way, all the network events can be done in only three RTTs. This results in $O(1)$ – or even less depending on the proportion of reads and writes – number of network messages even for operations that depend on each other.

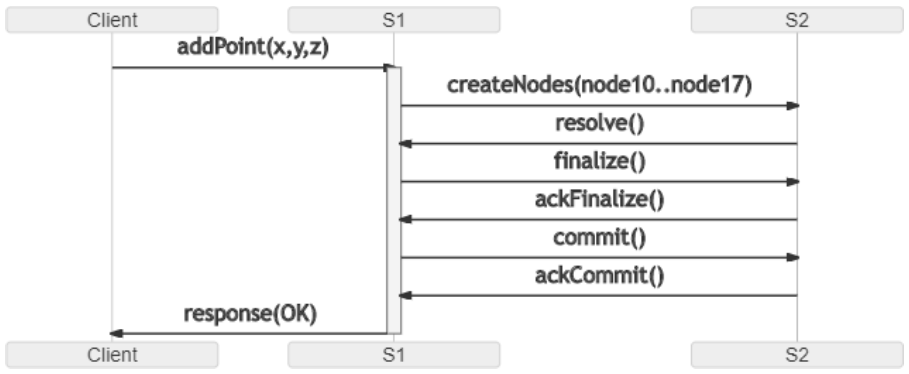


Figure 9: Add point to an octree in a Traquest based architecture

Figure 9 shows the same use case that was presented in Figure 8, but it uses the Traquest model. We do not need to use separate application and data servers since the Traquest model has constructions for both processing and storing. We have two servers for storing the replicated data on two different physical computing devices, just like the multitier example. As we discussed earlier, Traquests do not need to synchronize the read operations because if a conflict arises, the responded value can be later undone. Furthermore, the writes can also be aggregated and buffered and be sent in one message on the network. In this particular case, there are only six network messages between the servers. Compared with the optimistic estimation of 44 for the multitier architectures, this is a significant reduction. This difference is even bigger if there are more dependent iterative operations necessary for the database. For example, suppose we would have a graph as an example, and we wanted to find the shortest distance between two points. In that case, there could be thousands of dependent operations to the database, but with the Traquest model, we would still need only six network messages.

Here we need to emphasize that this optimal scenario is valid only if the necessary particles of the global state have replications locally. These numbers also depend on the actual infrastructure topology, the used components, the concrete use case, and many other factors.

However, even if not every data is available locally, the execution of the necessary Traquest tree branches can be delegated to other servers since all the servers are running the Traquest environment. Therefore the location of the processing can move to the data location and not backwards. This means much less communication over the network, even in this case.

Suppose we have a worst-case scenario, where the data stored over the servers is randomly fragmented. In that case, the number of the messages in the Traquest model can grow on a $O(n)$ scale, which is the best-case scenario for the multitier architectures. However, this would mean already an unrealistic and completely randomized worst-case topology. In general, we can say that Traquests has the potential to reduce the network load by magnitudes.

Table 1: Comparing the Traquest model

Case #	Circumstances				Response type		
	Concurrency	**Dependency**	**Topology**	**Operations**	**Final**	**Temporary**	
1	Optimistic	Independent	Ideal	Read only	$O(1)$	$O(0)$	
2	Optimistic	Independent	Ideal	Read & Write	$O(1)$	$O(1)$	
3	Optimistic	Independent	Random	Read only	$O(n)$	$O(n)$	
4	Optimistic	Independent	Random	Read & Write	$O(n)$	$O(n)$	
5	Optimistic	Dependent	Ideal	Read only	$O(n)$	$O(0)$	!
6	Optimistic	Dependent	Ideal	Read & Write	$O(n)$	$O(1)$	
7	Optimistic	Dependent	Random	Read only	$O(n)$	$O(n)$	
8	Optimistic	Dependent	Random	Read & Write	$O(n)$	$O(n)$	
9	Pessimistic	Independent	Ideal	Read only	$O(n)$	$O(0)$	!
10	Pessimistic	Independent	Ideal	Read & Write	$O(n)$	$O(n)$	
11	Pessimistic	Independent	Random	Read only	$O(n)$	$O(n)$	
12	Pessimistic	Independent	Random	Read & Write	$O(n)$	$O(n^2)$	
13	Pessimistic	Dependent	Ideal	Read only	$O(n)$	$O(0)$	!
14	Pessimistic	Dependent	Ideal	Read & Write	$O(n)$	$O(n)$	
15	Pessimistic	Dependent	Random	Read only	$O(n)$	$O(n)$	
16	Pessimistic	Dependent	Random	Read & Write	$O(n)$	$O(n^2)$	

Comparing the Traquest model with the existing technologies and models is challenging because there can be many different distributed scenarios. Table 1 shows a more complex comparison of the Traquest model. We showed four different kinds of circumstances that can highly influence the properties of a distributed system. The "Concurrency" column shows whether the examined distributed system has an optimistic or pessimistic concurrency scenario. We consider a scenario optimistic when there are no conflicts or the global state operations are executed in proper order or affect different records of the global state. In a pessimistic scenario, all the global state operations are executed in reverse order on the same global state records; therefore, we have the highest possible amount of conflicts. Realistically, most the distributed systems are much closer to the optimistic scenario.

The "Dependency" column shows whether the global state operations are dependent on each other. They are dependent if a global state operation has to await a previous one to be executed. For instance, incrementing a value is a dependent operation since the value first has to be read to increment it.

The "Topology" column shows whether the topology of global state particles is ideal or not. The topology is ideal if all the corresponding global state particles are stored on the same servers; therefore, the processes can reach them at one step. The topology is random if the global state particles are spread across the servers without considering the probability of them being used together. For instance, consistent hashing used for sharding by many of the most popular databases (AWS DynamoDB [7], Redis [30], etc.) creates such a random topology.

The "Operations" column shows whether the examined processes are write-heavy or they only contain read operations.

There are two "Response type" columns in the table. The Traquest model is built on introducing the temporary responses. As we described earlier, the Traquest model can collect all the conflicts and resolves them lazily in a buffered way. The column "Final" refers basically to all the currently existing technologies (multitier, serverless, actor, etc.) where the response to a request is final; therefore, all the conflict resolution must be finished in advance. It is usually done by using a locking mechanism. The "Temporary" column refers practically to the Traquest model as it introduces the temporary responses. However, we do not exclude the possibility that later new models or concepts will arise and utilize this idea as well.

The limitation in the performance of a distributed system is usually coming from the overall time needed to get network messages from one computing node to the other. A larger amount of data can be sent over the network relatively fast in a buffered way, but when there are more messages, each message has a roundtrip time and an overhead. Also, in-memory calculations are magnitudes faster than network communication. Therefore, to compare the Traquest model, we choose the necessary number of messages compared to the number of global state operations as the primary indicator for the performance of the distributed system.

The rows in the table describe different cases respective to the different circumstances. The last two columns show how the necessary number of network messages is growing as we increase the global state operations. The cells with green background mean easier circumstances or better performance, and the red background table cells have the opposite meaning. Table cells with white background mean no significant difference between traditional architectures and the Traquest model. The exclamation marks at the end of Case 5, 9, and 13 highlight that the Traquest model performs better not only by one but also by two magnitudes.

When the global state operations come strictly in order, they are independent, and the topology is ideal, the traditional databases can use pipelining. In this ideal scenario, the read and write operations can be sent over one single network message. This implies that at Case 1 and 2, the number of messages can scale on a $O(1)$ level ideally. In all the other cases, there is at least one network message per global state operation necessary. When the operations come entirely out of order and have a pessimistic concurrency scenario, traditional architectures using final responses still need only $O(n)$ messages. However, they also need to use locking mechanisms actively. This slows down the execution significantly. We did not show this effect in the table because we assumed that the servers could utilize this freed resource for other tasks or processes.

With the Traquest model, the processing and storing of the data can be performed on the same server, supposing we have read operations only and the topology is ideal. Therefore, in Case 1, 5, 9, and 13, there is no need for messages on the network at all. This means the number of the messages scales on a $O(0)$ scale. In an ideal case, when there are writes, the Traquest model can immediately process the writes in-memory time and postpones the conflict resolving and data replication lazily to buffer them to one single message. The committing mechanism runs in a buffered way as well, meaning that Case 2 and 6 can scale on a $O(1)$ level. In the worst cases, the Traquest model needs $O(n)$ network messages, except when we need to consider pessimistic concurrency. When we have pessimistic concurrency, random topology, and write-heavy global state operations, the Traquest model can require $O(n^2)$ number of messages. There is a big difference between read and write operations because reads do not generate conflicts thanks to the ability of the Data protectors to store the history of the global state records and serve the read operations, respectively.

We can conclude that the Traquest model performs better than the traditional architectures when considering some level of topology optimization and a more optimistic concurrency scenario. In this case, the advantage can reach several magnitudes. The Traquest model is not ideal when there is a limited amount of global state records getting many concurrent, and conflicting writes or the topology of the global state records is random.

3.3 Comprehensive solutions in the literature

3.3.1 Consistency Choices in Distributed Systems

Gotsman et al. [12] give a comprehensive overview of the consistency issues and the compromises that should be considered when designing a distributed system. It argues that only the necessary level of consistency should be used to avoid unnecessary loss in the performance. Using different levels of consistency in the same distributed system in such a mixed way is called hybrid consistency. The authors have designed a modular methodology to help developers decide the necessary level of consistency, and they presented proof for the methodology.

Despite the comprehensiveness of the paper, the different consistency issues are all discussed on the database level. All the consistency issues are discussed as database-related issues. This pattern can be recognized in the literature in general since consistency problems are associated with the consistency of the global state, and most of the distributed systems are managing the global state in the persistence layer. On the other hand, Traquests can ensure consistency – and atomicity – on the data processing level, not only on the data storage level. This is a major difference that allows a significant leap forward in the potentially reachable performance.

The authors of the paper also discuss the usefulness of hybrid consistency strategies. Through the enhanced performance, the Traquest model can highly reduce the consistency level dilemma and reduce the necessity of using hybrid strategies. Nevertheless, the Traquest model still supports hybrid consistency implicitly. When we wish to create strong consistency, we can build a fully bound Traquest tree as an atomic operation, and all the state changes and replica synchronization steps will remain strongly consistent.

However, we are not always restricted to define the parent of a Traquest. This way, we can separate different branches from the Traquest tree, and we can create less consistent operations to gain more performance. Although, in the Traquest model, it can rarely have clear benefits because strong consistency can already perform equally fast. Separating Traquest trees can be beneficial when we need to face highly conflicting use cases. In such a pessimistic concurrency case, the continuous rollbacking of the branches could slow down the execution of the Traquest tree, and hybrid consistency can be an effective solution.

3.3.2 Performance of Transactional Distributed Systems

Eric Brewer introduced the idea that there is a fundamental tradeoff between consistency, availability, and network partition tolerance. This tradeoff, known as the CAP [10] theorem, has been widely discussed ever since. Some of the interest in CAP derives from the fact that it illustrates a general tradeoff in distributed computing: the impossibility of guaranteeing both safety and liveness in an unreliable distributed system.

Ahsan et al. [1] discuss the CAP theorem in more depth. The authors highlight some of the limitations in the practical usage of the CAP theorem, and they propose a new impossibility theorem called the CAT theorem.

The paper refers to Jim Gray's paper from 1996 [13] which showed that the rate of transaction aborts increases at least proportional to the square of the TPS (throughput) of the system, and the third to the fifth power of the number of actions in the transaction.

The Traquest model cannot violate the CAP theorem either, but it can provide a workaround. The Traquest model can inherently provide consistency and partition tolerance. Transaction availability is also basically provided thanks to the Tail Traquests. Availability is provided for the whole transaction as well; just the final commit of the root Traquest has to be awaited. We will have the correct result, and only the response time can be higher for the final commit if the writes need to be synchronized.

The workaround nature of the Traquest model for the CAP theorem confirms that alternative and more practical impossibility theorems can be essential. The CAT theorem stands for Contention, Abort Rate, and Throughput. The Traquest model tries to keep the abort rate minimal, thanks to introducing the temporary mistakes. If an exception arises, that would typically trigger the abortion of the whole transaction. Instead, in the Traquest model, only the affected Traquest tree branch gets aborted, rolled back, and re-executed. Therefore the Traquest model balances between Contention and Throughput.

In the Traquest model, resolving a conflict is relatively expensive, thanks to the undoing mechanism. Therefore, the Traquest model shows the most significant potential in the case of optimistic concurrency cases. This refers to cases where we can assume that most of the operations are not conflicting. We can say that regarding the CAT theorem, the Traquest model prefers Contention and low Abort Rate over Throughput. However, this decision is not architecturally predefined. When the system gets fewer conflicting concurrent operations, it automatically achieves higher throughput.

4 Self-reflection and further research

4.1 Current status

To be able to verify the Traquest model, we have built an experimental prototype in TypeScript. The Traquest model itself has no language dependencies. It can be implemented practically in almost any programming language. However, using TypeScript gave us more benefits in the current phase of the research. The concept of Promises is highly used in JavaScript. Typescript is a superset of JavaScript [25]; therefore, it simplified the experimentation process with Promises and Traquests. The flexibility of JavaScript and the strong typing, the strictness, and the expressivity of TypeScript made it an ideal solution. Furthermore, TypeScript compiles to JavaScript, which can be executed on the client and server side, enabling the experimentation with hybrid architectures balancing between cloud, edge, and peer to peer.

Building this experimental prototype helped clarify that the general concept of the Traquest model is feasible and viable. We could try the different state transitions, callbacks and write unit tests and integration tests to verify whether the Traquests behave as expected. The functional tests have worked as they were expected, and they gave a positive result. However, the non-functional tests gave a slower execution time than we were expecting. The complete correctness of the final version of the model can be verified only after more tests at a larger scale or a complete formalization of the model. This requires further research and optimization. However, we expect no significant changes in the general concept. Only slight modifications are expected for edge cases we could not consider in advance. These edge cases also might vary slightly depending on the programming language used for the implementation.

4.2 Local boilerplate

One of the main disadvantages of the Traquest model is that it increases the computing power necessary on a local level. Managing all the Traquests, calculating the timestamps, maintaining all the states can consume significant computing. The non-functional tests of the Traquest model clearly showed this issue. When we created only simple standalone Traquests, then 100,000 Traquests could be executed in 460 msec. However, when we had a more complex Traquest tree where we created a binary tree from the Traquests, we measured 990 msec to execute only 5,998 Traquests which means less than one Traquest/msec.

Thanks to the results of the non-functional tests, the current focus of our research is to optimize the Traquests performance.

Some parts of this increased computing resource consumption come from the programming language used for the prototype implementation. Namely, if we dynamically change the schema of an object in runtime, the V8 JavaScript engine changes to a much slower general implementation in the background. We measured that this deceleration can be even on a 100X times slower level. We have built many new concepts to mitigate this issue. To describe them in detail would be out of scope for the current paper, but we mention the most important ones. We have created a custom Promise implementation which we measured to be 30X times faster than the standard JavaScript Promise implementation. We also created a custom callback mechanism which we measured to be around 14X times faster than the standard JavaScript anonymous callbacks. We also prepared a second prototype of the Traquest model, avoiding object schema changes. With these optimizations, we could significantly accelerate the Traquests and execute 100,000 standalone Traquests in 54 msec.

Some parts of the increased computing resource are coming from a more general algorithmic level. We have also conducted more research that would be out of scope to discuss in more detail, but we mention the most significant achievements. One of the most resource-consuming parts of the Traquest model is the hierarchical timestamping mechanism. We highlighted this issue in Section 2.3.2. To address this, we created a new timestamp algorithm what we named Interval Timestamp, which uses a logical time interval to define inheriting timestamp relations instead of arrays used by the naive algorithm.

We plan to expand the research on aggregating the Traquests that can be executed sequentially. If a group of Traquests is executed by one single thread, that means there can be no conflicting operations from elsewhere; therefore, they could be handled as one aggregated Traquest. This optimization could reduce the execution boilerplate of the Traquests almost completely. Only the Traquest communicating on the network would break this aggregation and end up in physically new Traquests.

After this research phase, we will have a final Traquest implementation, and we will be able to test Traquests in larger quantities. This will let us create a more rigorous validation of the Traquest model and make more definitive performance tests comparing Traquests with architectures based on the currently existing technologies.

The Traquest model in its current status is already a unique approach utilizing the idea that a response to a request can contain temporary information. A general system can be built based on this principle. This system can postpone synchronization and conflict resolution phases lazily, enabling us much more buffering on the network and in-memory time speed even in the dependent operations where network communication is inevitable using the current technologies.

4.3 Formalizing

We tried to model every possible scenario that can happen with a Traquest tree to be sure about the correctness of the model. However, the Traquest model is a new concept, and in the future, building formal semantics and reasoning for the Traquest model using tools like the Coq [38] formal proof managing system would give us an absolute certainty about the correctness of the model.

5 Conclusion

Providing ACID properties can be crucial for many applications, but it requires a massive compromise in performance. The Traquest model is a proposed potential solution for this problem. We have discussed in detail the general concept of the Traquest model.

Traquests are unifying the location of the data storing and processing; they are using specialized timestamps and history tracking of the global state changes that can potentially cause conflicts. By creating temporary responses and building up a mechanism for rolling back the conflicting parts of a running distributed algorithm, Traquests can ensure atomicity in a very efficient way. The synchronization steps over the network for replication and fault tolerance do not block the business logic executed in the Traquests, giving a massive advantage in the performance. Furthermore, this lazy synchronization allows an effective buffering of the network messages; therefore, the number of the necessary network messages can be decreased by magnitudes.

In our investigated concrete use case, the classical multitier architecture required 44 messages between the servers, where the Traquest model only needed six. The difference can grow magnitudes as the complexity of the algorithm grows. We showed that classical architectures require at least $O(n)$ network messages as the number of operations grows in the case of depending operations. Simultaneously, the Traquest model can scale with only a $O(1)$ factor.

We showed how the Traquest model could mitigate the dilemma of choosing between the different consistency levels and how the Traquests can provide hybrid consistency.

We investigated the Traquest model through the CAP theorem and realized that the Traquest model does not violate but gives a workaround for the CAP theorem. We considered another impossibility theorem as well, called the CAT theorem, and identified the characteristics of the Traquest model respectively.

We discussed the main current challenges and future research directions to come over those limitations. The biggest challenge is to improve the local performance of the Traquests, and our suggested solution is the aggregation of those Traquests to reduce the boilerplate execution.

The presented Traquest model can be a good foundation for making distributed ACID computation magnitudes faster and easier.

References

[1] Ahsan, Shegufta Bakht and Gupta, Indranil. The cat theorem and performance of transactional distributed systems. In *Proceedings of the 4th Workshop on Distributed Cloud Computing*, DCC '16, New York, NY, USA, 2016. Association for Computing Machinery. DOI: `10.1145/2955193.2955205`.

[2] Amazon. AWS Lambda. `https://aws.amazon.com/lambda/`, 2021. Accessed: 2021-04-14.

[3] Baker, Henry and Hewitt, Carl. The incremental garbage collection of processes. In *SIGPLAN Notices 12, 8*, pages 55–59. ACM, 1977.

[4] Baldini, Ioana, Castro, Paul, Chang, Kerry, Cheng, Perry, Fink, Stephen, Ishakian, Vatche, Mitchell, Nick, Muthusamy, Vinod, Rabbah, Rodric, Slominski, Aleksander, and Suter, Philippe. *Serverless Computing: Current Trends and Open Problems*. In *Research Advances in Cloud Computing*, pages 1–20. Springer Singapore, Singapore, 2017. DOI: `10.1007/978-981-10-5026-8_1`.

[5] Bogner, Justus, Zimmermann, Alfred, and Wagner, Stefan. Analyzing the relevance of SOA patterns for microservice-based systems. In *10th Central European Workshop on Services and their Composition*, volume 2072, pages 9–16, 2018. `http://ceur-ws.org/Vol-2072/`.

[6] Buschmann, Frank, Meunier, Regine, Rohnert, Hans, Sommerlad, Peter, and Stal, Michael. *Pattern-Oriented Software Architecture, Volume 1, A System of Patterns*. Wiley, 1996.

[7] DeCandia, Giuseppe, Hastorun, Deniz, Jampani, Madan, Kakulapati, Gunavardhan, Lakshman, Avinash, Pilchin, Alex, Sivasubramanian, Swaminathan, Vosshall, Peter, and Vosshall, Werner. Dynamo: Amazon's Highly Available Key-Value Store. *SIGOPS Oper. Syst. Rev.*, 41(6):205–220, 2007. DOI: `10.1145/1323293.1294281`.

[8] Fette, Ian and Melnikov, Alexey. The WebSocket Protocol. *RFC*, 6455:1–71, December 2011.

[9] Friedman, Daniel and Wise, David. The impact of applicative programming on multiprocessing. Technical report, Computer Science Department, Indiana University, Bloomington, 1976.

[10] Gilbert, S. and Lynch, N. Perspectives on the cap theorem. *Computer*, 45(02):30–36, 2012. DOI: `10.1109/MC.2011.389`.

[11] Google. Google Cloud Functions. `https://cloud.google.com/functions`, 2021. Accessed: 2021-04-14.

[12] Gotsman, Alexey, Yang, Hongseok, Ferreira, Carla, Najafzadeh, Mahsa, and Shapiro, Marc. 'Cause I'm Strong Enough: Reasoning about Consistency Choices in Distributed Systems. *SIGPLAN Not.*, 51(1):371—384, 2016. DOI: `10.1145/2914770.2837625`.

[13] Gray, Jim, Helland, Pat, O'Neil, Patrick, and Shasha, Dennis. The dangers of replication and a solution. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, page 173–182. Association for Computing Machinery, 1996. DOI: `10.1145/233269.233330`.

[14] Henshall, John and Shaw, Sandy. *OSI explained: end-to-end computer communication standards*. Ellis Horwood Chichester, 1990.

[15] Hewitt, Carl, Bishop, Peter, and Steiger, Richard. A universal modular actor formalism for artificial intelligence. In *IJCAI'73: Proceedings of the 3rd international joint conference on Artificial intelligence*, pages 235–245, 1973.

[16] Hibbard, Peter. Parallel processing facilities. In Schuman, Stephen A., editor, *New Directions in Algorithmic Languages*. IRIA, 1976.

[17] Kozierok, Charles M. *The TCP/IP Guide: A Comprehensive, Illustrated Internet Protocols Reference*. William Pollock, 2005.

[18] Loreto, Salvatore and Romano, Simon Pietro. *Real-time communication with WebRTC: peer-to-peer in the browser*. O'Reilly Media, Inc., 2014.

[19] Loreto, Salvatore, Saint-Andre, P, Salsano, Sd, and Wilkins, G. Known issues and best practices for the use of long polling and streaming in bidirectional http. *Internet Engineering Task Force, Request for Comments*, 6202(2070-1721):32, 2011. DOI: `10.17487/RFC6202`.

[20] Madsen, Magnus, Lhoták, Ondřej, and Tip, Frank. A Model for Reasoning about JavaScript Promises. *Proc. ACM Program. Lang.*, 1(OOPSLA), October 2017. DOI: `10.1145/3133910`.

[21] Masse, Mark. *REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces*. O'Reilly Media, Inc., 2011.

[22] Meagher, Donald. Geometric modeling using octree encoding. *Computer Graphics and Image Processing*, 19(2):129–147, 1982. DOI: `10.1016/0146-664X(82)90104-6`.

[23] Meyer, T and Brunn, A. 3D Point Clouds in PostgreSQL/PostGIS for Applications in GIS and Geodesy. In *Proceedings of the 5th International Conference on GISAM - Volume 1*, pages 154–163. SciTePress, 2019. DOI: `10.5220/0007840901540163`.

[24] Microsoft. Azure Functions. `https://azure.microsoft.com/en-us/services/functions/`, 2021. Accessed: 2021-04-14.

[25] Microsoft. TypeScript Documentation. `https://www.typescriptlang.org/docs/handbook/typescript-from-scratch.html`, 2021. Accessed: 2021-04-14.

[26] Oracle Corporation. Java documentation - Class CountDownLatch. `https://docs.oracle.com/javase/10/docs/api/java/util/concurrent/CountDownLatch.html`, 2018. Accessed: 2021-04-14.

[27] Plakal, Manoj, Sorin, Daniel J., Condon, Anne E., and Hill, Mark D. Lamport Clocks: Verifying a Directory Cache-Coherence Protocol. In *Proceedings of the Tenth Annual ACM Symposium on Parallel Algorithms and Architectures*, SPAA '98, page 67–76. ACM, 1998. DOI: `10.1145/277651.277672`.

[28] Ratai, Daniel Balazs, Horvath, Zoltan, Porkolab, Zoltan, and Toth, Melinda. Traquest model – a novel model for ACID concurrent computations. In *The 12th Conference of PhD Students in Computer Science – Proceedings*, pages 44–48. Institute of Informatics, University of Szeged, 2020.

[29] Redis Labs Ltd. Redis documentation – Using pipelining to speedup Redis queries. `https://redis.io/topics/pipelining`, 2020. Accessed: 2021-04-14.

[30] Redis Labs Ltd. Redis documentation – Partitioning: how to split data among multiple Redis instances. `https://redis.io/topics/partitioning`, 2021. Accessed: 2021-04-14.

[31] Sarieddine, Rami. *JavaScript Promises Essentials.* Packt Publishing Ltd., 2014.

[32] Tanenbaum, Andrew S. and Steen, Maarten Van. Consistency protocols. In *Distributed Systems – Principles and Paradigms*, pages 306–317. Prentice Hall, 2007.

[33] Truica, C., Radulescu, F., Boicea, A., and Bucur, I. Performance Evaluation for CRUD Operations in Asynchronously Replicated Document Oriented Database. In *2015 20th International Conference on Control Systems and Computer Science*, pages 191–196, 2015. DOI: `10.1109/CSCS.2015.32`.

[34] Typesafe Inc. Transactors. `https://doc.akka.io/docs/akka/2.1/scala/transactors.html`, 2013. Accessed: 2021-04-14.

[35] Uyanık, H. and Ovatman, T. Enhancing Two Phase-Commit Protocol for Replicated State Machines. In *2020 28th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, pages 118–121, 2020. DOI: `10.1109/PDP50117.2020.00024`.

[36] Vinoski, S. Server-sent events with yaws. *IEEE Internet Computing*, 16(5):98–102, 2012. DOI: `10.1109/MIC.2012.117`.

[37] Zakeryfar, Maryam and Grogono, Peter. Static Analysis of Concurrent Programs by Adapted Vector Clock. In *Proceedings of the International C* Conference on Computer Science and Software Engineering*, C3S2E '13, page 58–66, New York, NY, USA, 2013. Association for Computing Machinery. DOI: `10.1145/2494444.2494476`.

[38] Zhang, Xiyue, Hong, Weijiang, Li, Yi, and Sun, Meng. Reasoning about connectors in Coq. In *International Workshop on Formal Aspects of Component Software*, pages 172–190. Springer, 2016.

[39] Zheng, Haitao and Boyce, Jill. An improved UDP protocol for video transmission over internet-to-wireless networks. *IEEE Transactions on Multimedia*, 3(3):356–365, 2001. DOI: `10.1109/6046.944478`.

Appendix

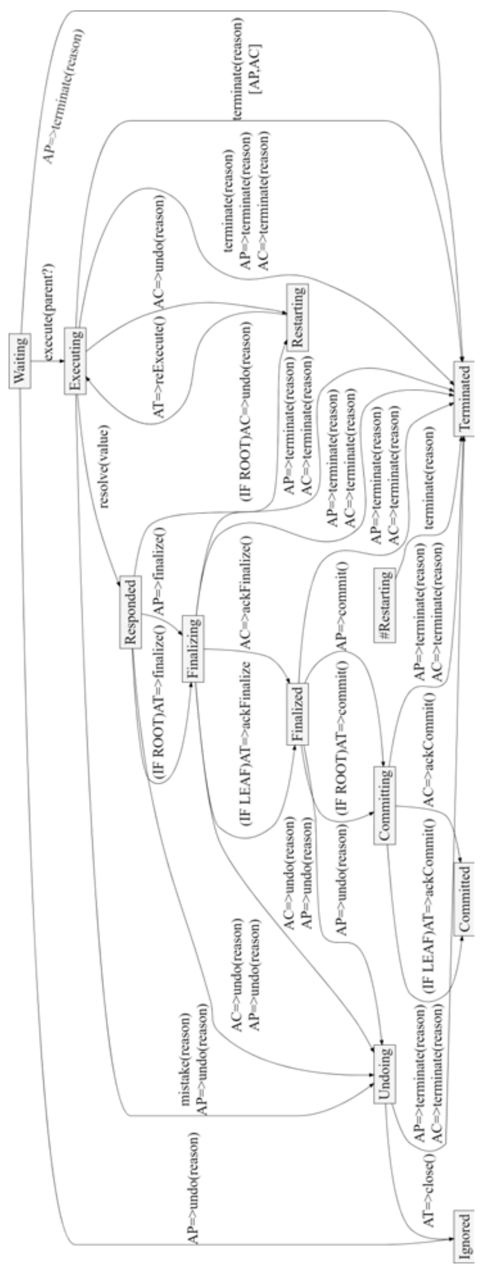


Figure 10: Comprehensive Traquest state diagram

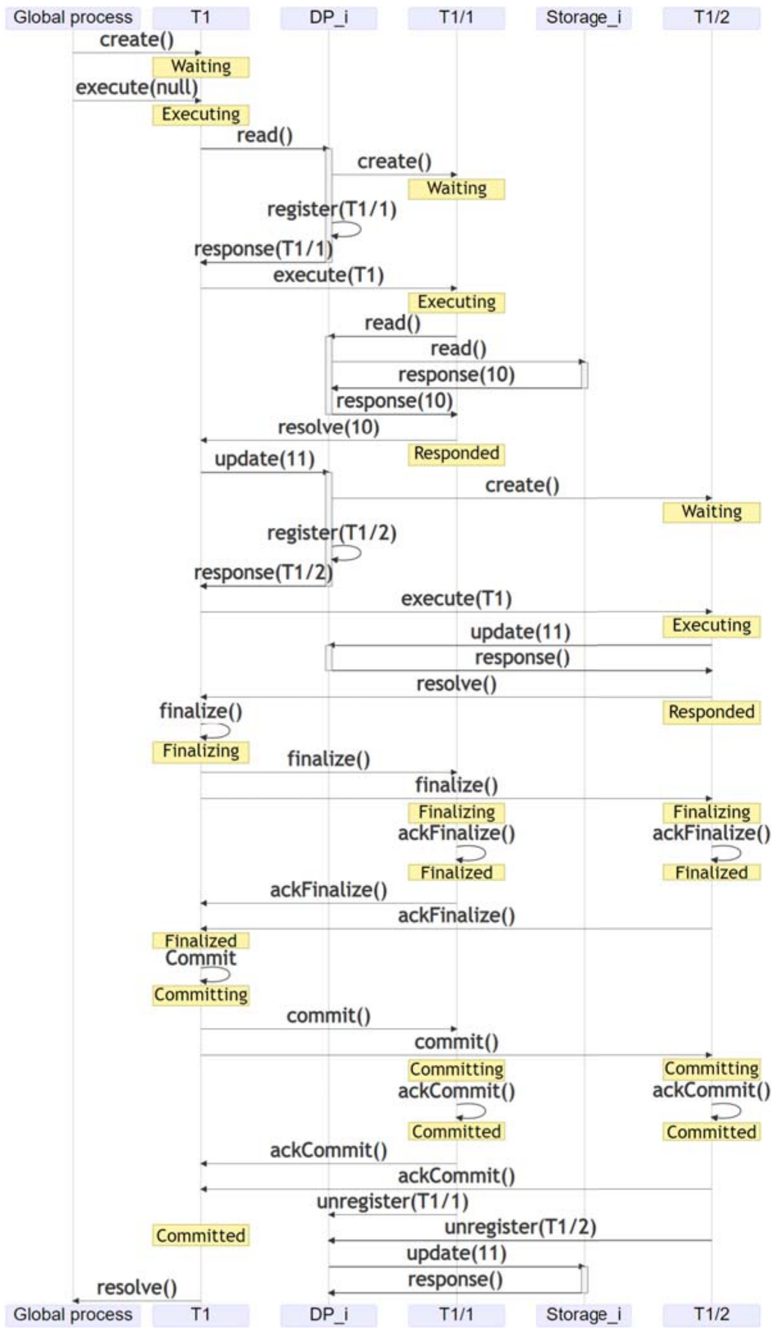
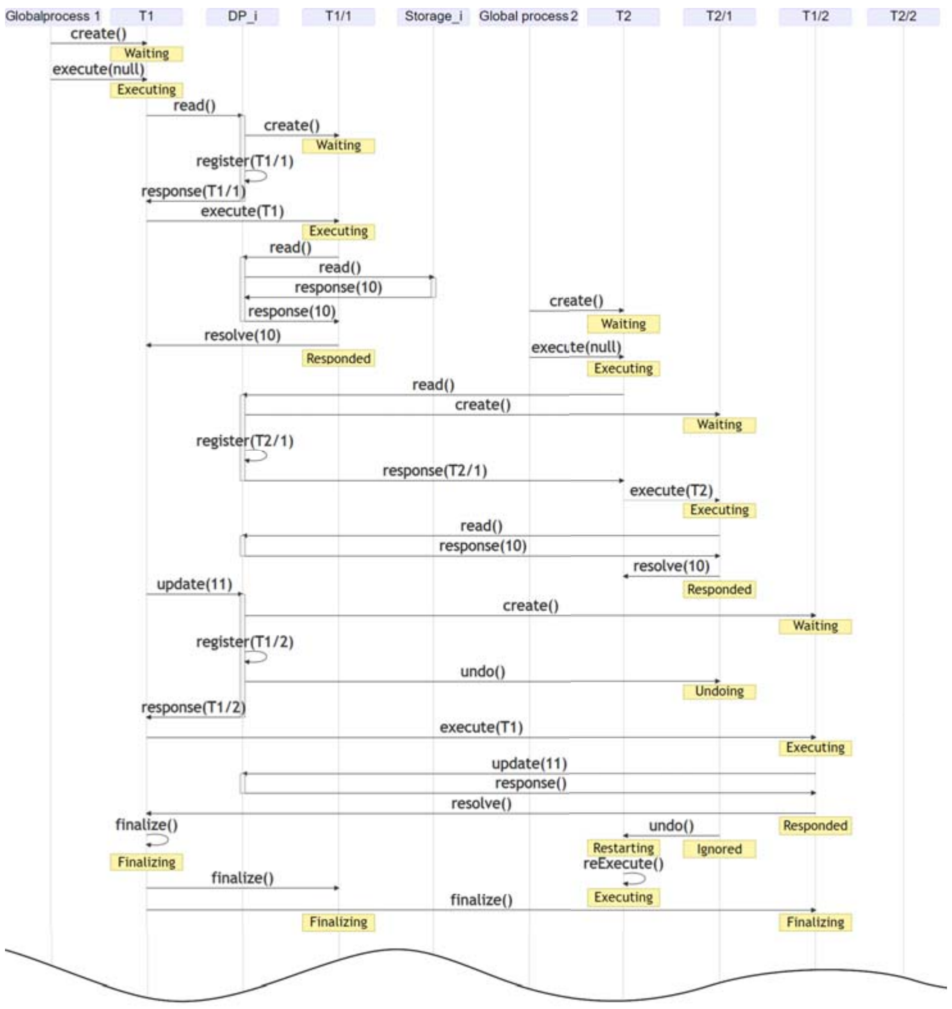


Figure 11: Sequence diagram of an incrementation



...

Figure 12a: Sequence diagram of an incrementation with a conflict (Part 1/2)

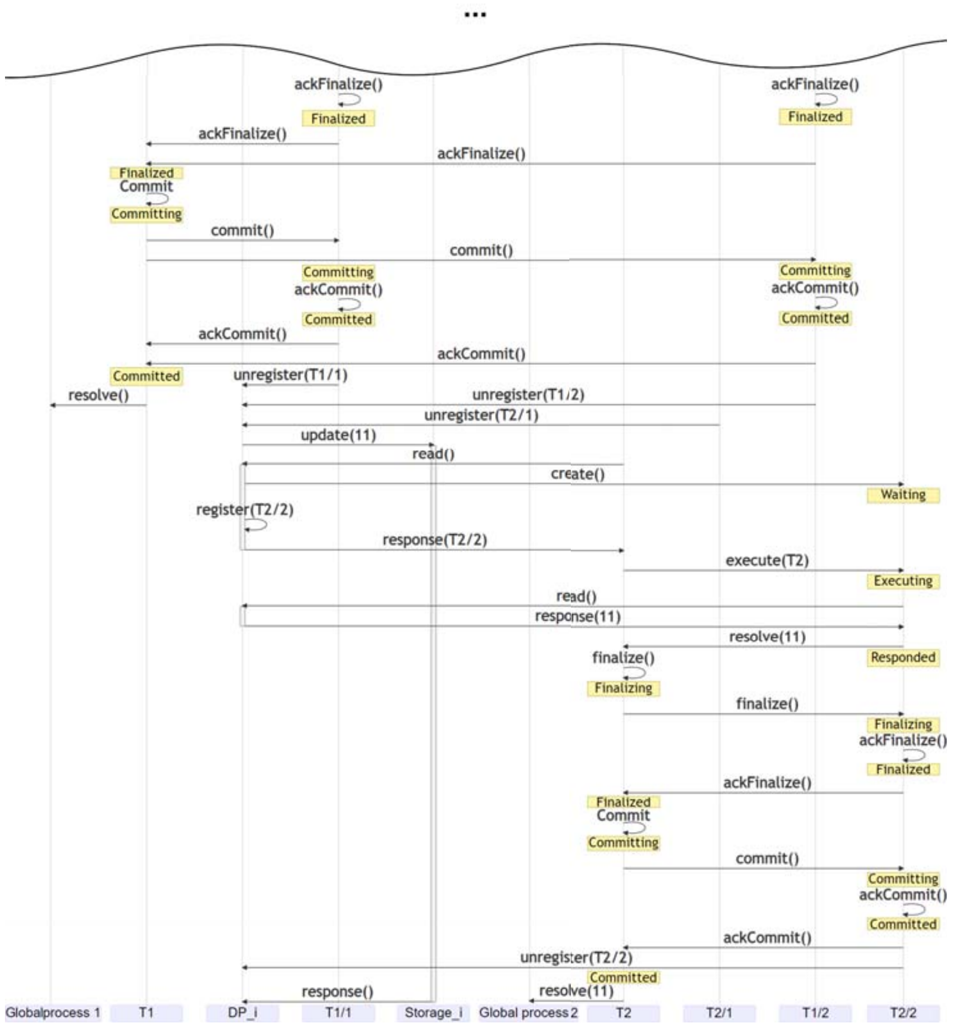


Figure 12b: Sequence diagram of an incrementation with a conflict (Part 2/2)

Acta Cybernetica **25** (2021) 469–484.

A Parallel Event System for Large-Scale Cloud Simulations in DISSECT-CF*

Dilshad Hassan Sallo[ab] and Gabor Kecskemeti[acd]

Abstract

Discrete Event Simulation (DES) frameworks gained significant popularity to support and evaluate cloud computing environments. They support decision-making for complex scenarios, saving time and effort. The majority of these frameworks lack parallel execution. In spite being a sequential framework, DISSECT-CF introduced significant performance improvements when simulating Infrastructure as a Service (IaaS) clouds. Even with these improvements over the state of the art sequential simulators, there are several scenarios (e.g., large scale Internet of Things or serverless computing systems), which DISSECT-CF would not simulate in a timely fashion. To remedy such scenarios this paper introduces parallel execution to its most abstract subsystem: the event system. The new event subsystem detects when multiple events occur at a specific time instance of the simulation and decides to execute them either on a parallel or a sequential fashion. This decision is mainly based on the number of independent events and the expected workload of a particular event. In our evaluation, we focused exclusively on time management scenarios. While we did so, we ensured that the behaviour of the events should be equivalent to realistic, larger-scale simulation scenarios. This allowed us to understand the effects of parallelism on the whole framework, while we also shown the gains of the new system compared to the old sequential one. With regards to scaling, we observed it to be proportional to the number of cores in the utilised SMP host.

Keywords: cloud computing, parallel simulation, DISSECT-CF

1 Introduction

There are several obstacles that stop increasing the performance of DES frameworks. First of all, most are designed to execute sequentially. The need of simulat-

*This research was supported by the Hungarian Scientific Research Fund under the grant number OTKA FK 131793.

[a]Institute of Information Technology, University of Miskolc, Miskolc, Hungary

[b]E-mail: `sallo@iit.uni-miskolc.hu`, ORCID: `0000-0003-3672-4601`

[c]Department of Computer Science, Liverpool John Moores University, Liverpool, UK

[d]E-mail: `kecskemeti@iit.uni-miskolc.hu`, E-mail: `g.kecskemeti@ljmu.ac.uk`, ORCID: `0000-0001-5716-8857`

DOI: `10.14232/actacyb.287937`

ing multiple events in parallel, is now essential for several scenarios. For example, simulating Internet of Things (IoT) involving millions or more devices worldwide; or simulating billions of service invocations and their interactions in serverless computing situations. Introducing a parallel approach to the core event handling in DESs aimed at distributed systems simulations would be the first step towards the support of the aforementioned scenarios.

Always applying parallel execution to the simultaneously occurring events in the simulation does not necessarily lead to a well scaling DES though. When only a few events occur simultaneously, sequential execution is often times beneficial as we can avoid the overheads of parallel constructs; otherwise, the parallel execution can lead to better performance. The necessity of determining at a specific simulated time instance, whether the events will execute sequentially or parallel manner is crucial to increase the performance and to avoid unnecessary overhead.

Despite several DESs support simulating parallel and distributed computing, the majority lack of parallel execution. For instance, Cloudsim [3] and GroundSim [14] execute computing tasks sequentially. This raises challenges when trying to simulate novel technologies (e.g., serverless) that require large scale simulation to be used as a support tool. DISSECT-CF [10] is one of the frameworks capable to simulate internal components and processes of distributed systems (ranging from cloud and fog infrastructures to even IoT systems). Although the execution time of DISSECT-CF is significantly faster than the most prominent simulator in the field CloudSim [13], this performance advantage is still not sufficient for the most demanding current research use cases (e.g., simulating millions of IoT devices and their continuum with clouds). Its sequential execution is a significant bottleneck, thus parallelisation is needed for scaling its performance efficiently to meet the newest challenges in the field.

This paper introduces a parallel execution mode for the event subsystem of DISSECT-CF. Our approach automatically switches between this new mode and the old one based on the number of simultaneous events that occur at a given time instance of the simulation. To avoid the overhead of applied parallel constructs under low workloads, our approach keeps using the original sequential mode for situations when only a few simultaneous events are detected. Otherwise, a parallel event executor will be selected. This executor divides and distributes the simultaneous events equally over the available processors and balances the load across the system. These two operations avoid idle CPUs (or cores) behind the simulator. To avoid issues with the initial event distribution, our parallel approach also uses work stealing to further reduce the contention among threads.

We have designed several experiments to evaluate the scalability and performance of the new approach. These experiments focus on core functionality and time management mechanisms of the event subsystem in DISSECT-CF. The evaluation had independent control on the following four properties: (*i*) event independence (no influence on future events); (*ii*) pattern of events throughout a simulation (i.e., how many events do we have in total and when should they happen); (*iii*) number of simultaneous events (degree of parallelism) happening at an average time-simulated instance; (*iv*) the single event workload (i.e., how compute heavy is a particular event). We instrumented and measured the behaviour of realistic simulations in terms of these properties. Then, we implemented simple synthetic event patterns (that are only exercising the event subsystem of DISSECT-CF) for the simulator which we calibrated to imitate the properties of the previously measured realistic simulations. To ensure the quality of our experiments, we collected the synthetic event pattern's properties with the same measurement approach that we applied for the realistic setting to compare and analyse them. We also evaluated with random event patterns to test the behaviour of parallel version under unforeseen conditions.

We have executed our experiments on 12 core SMP hosts. Our experiments have been conducted with different degrees of parallelism and single event workload size. With respect to the number of cores, evaluation results show that two factors have affected the performance of the parallel version. First, if we have at least two simultaneous events for more than 50% of the simulated time instances, then the parallel version already runs two times faster than the sequential. Second, increasing the single event workload leads to 2.4 times faster simulation execution than sequential.

The remainder of this paper is structured as follows. In Section 2, we discuss work related to our approach. Section 3 discusses our methodology of employing parallelism in DISSECT-CF. Section 4 covers the evaluation of the parallel version. Finally, Section 5 concludes the paper and identifies future work.

2 Related work

Over the last decade, several DES frameworks have been designed to offer researchers an opportunity to evaluate and predict the behaviour of cloud computing applications. Each framework was designed with a specific purpose and having unique features that able solve some challenges.

CloudSim [3] is mostly used as general purpose cloud simulation environment. Due to the extensible nature of CloudSim, several extensions have been developed to integrate new features to it. DISSECT-CF [10] is a simulation framework that improved the modelling of resource-, network utilisation, power consumption and data centre configurations, by providing the capability of simulating IaaS internal behaviour. GroudSim [14] is a platform mainly focused on scientific application modelling (e.g., workflows) in cloud and grid computing. GreenCloud [11] is a simulator specifically built for estimating the energy consumption of cloud data centres. In addition to the above, the authors [1, 2] have conducted the detailed survey of over 33 simulators. Each of these is built for a specific purpose and is having unique features around cloud simulation. Although these simulators offer several features for cloud computing, the majority were built in a sequential fashion. Thus, they are all struggling to address recent challenges such as simulating millions of IoT devices.

Parallel discrete event simulation (PDES) approach has been applied in various fields such as simulation of networks, with the primary goal of performance. For example ROSS [4] and GWT [6] are parallel discrete event simulators that execute on shared-memory multiprocessor systems. They mostly used in large-scale networking simulation models and telecommunication networks. DaSSF [12] is also parallel simulator targeting network simulation and it achieves high performance through parallel processing. Unfortunately, these frameworks have limited applicability in the research areas surrounding cloud computing. Parallelising existing systems remains a challenge [7]. Moreover, the prominent language for cloud simulators [1, 2] is Java, while current PDESs are not easily adoptable to this language. However, one of the frameworks called Cloud2Sim [9] supports concurrent and distributed simulations of clouds, based on the following libraries: Hazelcast, Infinispan and Hibernate.

In [7] the author raises many challenges that researchers could face in a PDES. One of these challenges is the complexity of using a parallel implementation correctly and simplifying code to understand it easily. Agreeing with this, our approach aims to keep the original sequential APIs while making a parallel solution in the background. In [8] the authors suggested the initial steps towards cloud supporting PDESs, unfortunately these steps were not yet adopted by current simulators. Introducing parallel execution to simulators needs easy simulation control as well as repeatable tests. In [5] the authors explained that sequential execution can be insufficient for modelling real complex systems, and parallel execution could manage resources efficiently. Sequential approaches are unable to fulfil many requirements, and lead to trade-off between the cost and performance. However, there is opportunity for applying parallelism to DISSECT-CF simulator to gain better performance. Introducing parallel executions of its event system can benefit all subsystems built on top of it. Finally, in [15] the authors showed a possibility to execute simulations over multiple virtual machines.

The previous works show the minimal advances towards parallel execution in cloud computing frameworks, which are needed to address the present challenges that accompanied modern technologies in this field. Therefore, our proposed solution provides parallel execution to the event system of DISSECT-CF simulator to speed up the simulation to foster simulating larger scale systems and technologies (e.g., IoT). Although the techniques proposed here are likely to be applicable to other frameworks as well, this paper is solely focused on DISSECT-CF to show our approach's applicability.

3 Methodology

DISSECT-CF simulator introduced substantial features to foster the rapid evaluation of IaaS clouds and its extensibility lead to support for other concepts such as IoT and fog computing. Although, DISSECT-CF reduces the execution time of equal quality/detail simulations done compared to several other frameworks in the field, it still does so in a sequential fashion. In the past, DISSECT-CF was shown simulating hundreds of thousands of computing entities within a few hours. But it has little chance to sequentially simulate recent systems within an acceptable time frame. With this research, we aim to set the foundations to support simulations where the number of simulated computing components can easily reach over a billion of devices (like the IoT cloud continuum, or serverless computing).

3.1 Overview of DISSECT-CF simulator

DISSECT-CF is a simulation framework that offers insight into advanced cloud concepts supporting modern technologies. DISSECT-CF provides an amalgamation of several features that hardly exist in any previous simulators such as capturing low-level resource sharing behaviour and introducing an adequate energy consumption model. This aims to support previously problematic IaaS simulation scenarios that require all these advanced features to be available in the same framework.

The extensible core of the DISSECT-CF simulator consists of five major subsystems that mostly implement different concepts around clouds and distributed systems in a layered fashion [10]. Generally, each layer attempts to provide a comprehensive implementation for a particular concept without being dependent on the rest of the framework. The lowest subsystem, **event system** provides an appropriate mechanism to manage the behaviour of regular and irregular events as well as controlling the basic state of the simulation in a given time instance (so called tick in DISSECT-CF terminology). This subsystem is the foundation of all layers and introducing substantial features here such as parallelism has the highest potential impact on higher level subsystems. Next, **Unified resource sharing** subsystem introduces a holistic approach to establish a central resource provider able to share behaviour among low-level computing concepts. Then, the **Energy modelling** subsystem provides a unique approach that allows monitoring and analysing energy usage of all simulation resources by decoupling energy modelling from resource simulation (i.e., allows performance gains by only offering selective energy monitoring). On a layer above, the **Infrastructure simulation** subsystem deals with modelling the behaviour of typical distributed system components like virtual machines, physical machines, storage and networking. Finally, the highest layer of abstraction is provided in the **Infrastructure management** subsystem which contains major IaaS components such VM scheduler and PM scheduler that simulate the management of users requests and fosters the creation of custom internal IaaS behaviours. It also provides components such as Repository and the IaaS service to interact with users of the simulator.

Although the subsystems of DISSECT-CF have originally been written to execute sequentially, most of them can be executed in a parallel fashion as well. As all subsystems depend on the event subsystem, it comes as a natural point to adopt parallelism. As most of the operations in the higher level components are driven by events delivered from the event subsystem, these operations will be automatically parallelised with the parallelisation of the event subsystem itself.

3.2 Prominence of recurrent events

The lowest (event) subsystem of DISSECT-CF has two main classes: (*i*) the `Timed` class, used for recurring events; and (*ii*) `DeferredEvent` class used for irregular events. Recurring events are events that the simulator invoked them regularly based on a specified frequency. Thus, recurring events can subscribe notifications, when subscribing, an event frequency must be specified to determine how many ticks must pass to get repeated notifications. As the other subsystems are built on the top of mostly recurring events, our target is enhancing the performance of this subsystem, which will reflect the outcome over the rest of the framework and its extensions. The event sub-system had a sequential execution design. Based on the existing API of DISSECT-CF, parallelisation could happen for executing of simultaneously happening events (i.e., events that should happen in the same time instance or tick of a simulation).

To understand such simultaneous events, we have provided a simple example scenario with three event objects with various frequencies (this demonstrates events derived from the `Timed` class of DISSECT-CF which allows defining events that can happen repeatedly). Table 1 shows the basic details of our simple example scenarios. The first three rows show the event objects and their behaviour. The last line shows the time instances in our simple simulation. In the table, we can see for every time instance when the events will be processed. E.g., the second event ($e2$) is processed in time instances 3,6 etc. Figure 1 shows how the event queue will look like at any particular time instance in case we execute the events defined in the previous table.

The degree of parallelism denotes the number of events that happens at a specific time instance (tick). Which mainly depends on the frequencies of subscribed objects

Table 1: Three events with different frequencies.

Events	Freq	Next events of e1, e2 and e3 based on their frequencies(Freq)														
e1	1	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
e2	3	3	3	3	6	6	6	9	9	9	12	12	12	15	15	15
e3	5	5	5	5	5	5	10	10	10	10	10	15	15	15	15	15
Time		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Degree(%)		33	33	66	33	66	66	33	33	66	66	33	66	33	33	100

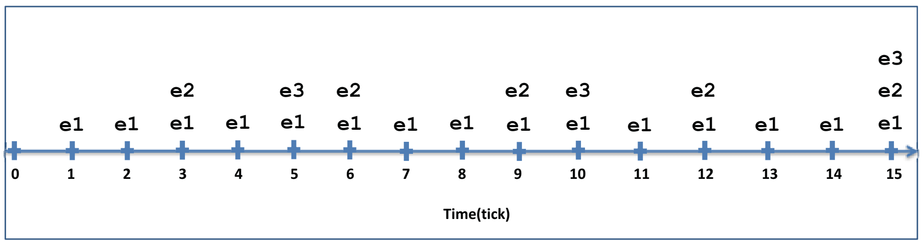


Figure 1: Representing multiple events in Table 1 occur at a specific time

that determine how frequently these events occur. When all subscribed events happen at a specific time, the degree of parallelism is 100%. When half of them occurs at one time, the degree is 50% and so on. Thus, the degree of parallelism varies according to the occurrence of events at each tick. Therefore, the average degree of parallelism in a single simulation run is deduced from all ticks for the whole system. In Figure 1, there are 15 simulated time instances, out of these 7 are having parallel events, making the example's average degree of parallelism 50.66%. If we execute simultaneously occurring events (e.g., $e1$ and $e2$ in time instance 3 in the Figure) in a sequential fashion, then we pay a penalty of using a sequential simulator. This observation will guide the next the sub-section where we discuss how we identify these kinds of events and how we execute them in parallel.

3.3 The parallelisation of simultaneous events

Figure 2 shows the basis of our extension. The diagram shows only the relevant parts of the original `Timed` class, and the new `Parallel` class. The `Parallel` is created as an inner class within `Timed` class, to ensure easy access to the original data structures within the event subsystem's main class. The user of the system is still expected to interface with the existing methods of the `Timed` class (thus all previous extensions to the simulator would benefit from our parallelisation approach). Note that inside the simulator, all higher level subsystems (e.g., those which simulate virtual machines) are considered as users of the `Timed` class. As parallelisation is automatically executed depending on the state of the event queue, the higher level subsystems benefit from the improvement on `Timed`.

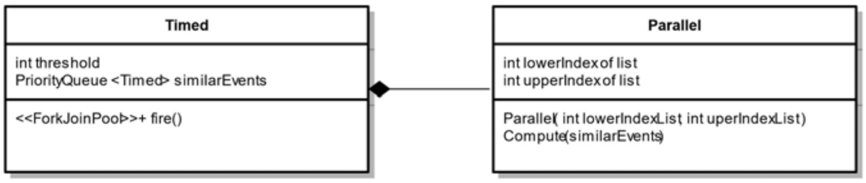


Figure 2: Diagram of Timed class and Parallel class

In DISSECT-CF, time is measured in ticks [10] and users of the simulator are free to interpret ticks the way they want. The events taking place in a particular tick are handled with the `fire()` method (see Figure 2). Our approach changes the behaviour of this method by introducing Algorithm 1. Here we first collect the list of simultaneously occurring events at each particular tick (see line 2) – note that this list was not needed for the sequential sub-system as that would only work with one event at a time. As a result, the collection of this list is an overhead of the new parallel algorithm. The discussed approach below aims at minimising this overhead.

Our new `fire()` method now checks the size of the list to determine if we need to execute in sequential or parallel fashion. The old, sequential execution is shown in the loop of line 4, this is still kept and used if we have too few simultaneous events

Algorithm 1 Determining the need for parallelism

```
 1: $threshold$ = specified size
 2: $list$ = all simultaneous events
 3: if $list.size <= threshold$ then
 4:    while $list.notEmpty$ do
 5:       $event$ = get single event from $list$
 6:       Execute $event$
 7:    end while
 8: else
 9:    invoke Parallel($list.lowIndex$, $list.upperIndex$)
10: end if
```

in the queue. The parallel execution utilises our new `Parallel` class to distribute the work over threads, that will be created implicitly according to the number of available cores. This is done by passing the `lowerIndex` and `upperIndex` that specify the indices of first and last elements of the list (see line 9). The `threshold` (minimal size of the list which leads to parallel execution) is configurable by the user of the simulator. To aid the user determining the threshold, an auto-tuning approach is also going to be offered for the threshold which determines its value when suitably long running simulations are executed. The auto-tuning approach bases its decision on the threshold on the typical single event workload. As the auto tuning approach also needs some compute time it is possible to disable it for simulations where the threshold is known to the user.

Algorithm 2 Mechanism of Parallel class

```
 1: Procedure Parallel( $list.lowIndex$, $list.upperIndex$ )
 2: $lowerIndex = list.lowIndex$
 3: $upperIndex = list.upperIndex$
 4: Funct compute ()
 5: if $upperIndex$ - $lowerIndex <= threshold$ then
 6:    while $list.notEmpty$ do
 7:       Execute events of $list$
 8:    end while
 9: else
10:    $midIndex = (lowerIndex + upperIndex / 2)$
11:    invoke all (Parallel($lowIndex$, $midIndex$), Parallel($midIndex$, $upperIndex$))

12: end if
```

After the decision to parallelise, the actual parallelisation is organised by the `Parallel` class according to Algorithm 2. Instances of this class are executed in their own threads. Thus, they will likely run on another CPU core compared to the original `fire()` method. When a `Parallel` instance is instructed to compute, it again uses our the previously discussed and determined `threshold` value to decide if the workload assigned to the thread is sufficiently small or not.

If the sublist of simultaneous events is short enough (see line 5), the sublist is executed in the current thread entirely. This sublist execution is done just like the sequential one was discussed before (see Algorithm 1's line 4). But instead of going through the entire list of simultaneous events, now we have a shorter list to process which was assigned only to the thread of this `Parallel` object in the parallel invocations of Algorithms 1 and 2.

In contrast, when there are more simultaneous events than a single thread should handle, we sub-divide the list of events based on its size in equal parts and pass them on to further threads (see line 11). We repeat this process until the list of events divided into sublists (sublists size become less than or equal threshold) and all threads have sufficiently short lists, then the threads are scheduled according to a fork-join model. This list division method ensures that we execute on all available processors in the current machine and also offers an initial load balance. The fork-join model uses work-stealing algorithm by allowing the thread to steal workloads from others. Although each thread has an almost equal number of sublists, work-stealing approach ensures that the threads workloads are almost equal to avoid wasting time.

4 Evaluation

A private cloud at LJMU was used for the evaluation of our parallel DISSECT-CF. For our experiments, we used a VM with the following specifications: Intel (R) Core (TM) i7-8700 CPU @ 3.2GHz (6 cores + 6 hyper threaded cores), 64GB memory, 1T SSD, 1T HDD, Debian Linux Buster 10.4, OpenJDK 11.0.6. We have designed several scenarios to test the performance of the parallel version by focusing on time management while ensuring complete control over event occurrence. We also made sure the evaluation was validating the parallel version: we used the complete API of the `Timed` class to verify if the parallel version produces results matching output from the unmodified sequential code.

4.1 Validation

To ensure that the behaviour of our evaluation is following real life simulation patterns, we have instrumented the `JobDispatchingDemo` class of the dissect-cf-examples project. This class was already validated before to produce realistic simulations e.g., comparable to CloudSim (see [10]). Our instrumentation focused on how the realistic simulation utilises the lowest abstraction layer of DISSECT-CF. We measured, the degree of parallelism, the typical event behaviour, the number of events in total and the average execution time of a single tick method call in nanoseconds (i.e., the single event workload). To enable the comparison, we have also instrumented our parallel `Timed` class in the same way allowing us to acquire the typical workload of our synthetic tick methods.

We have set up our realistic simulation with `JobDispatchingDemo` as follows: (*i*) maximum number of jobs that exist in parallel was set to 2; (*ii*) the amount of seconds the job startup times was set to 10; (*iii*) minimum execution time of a single job was set to 10s; (*iv*) maximum execution time of a single job was set to 90s; (*v*) minimum and maximum gaps between the last and the first job submission of two consecutive parallel batches were set to 200s; (*vi*) minimum number of processors for a single job was set to 1; (*vii*) maximum number of processors for a single job was set to 2; (*viii*) total number of processors usable by all parallel jobs was set to 4; (*ix*) total number of jobs was 100000; (*x*) the number of nodes was 5000.

To allow our evaluation to focus at the lowest abstraction layer (and our parallelism evaluations not to be distracted by upper layer behaviour), we set out to capture the event workload behaviour of the above complex simulation, but with a synthetic workload. Our synthetic tick method (implemented in the class `TimeRandomGenerator`), does a busy waiting loop by calculating the following formula:

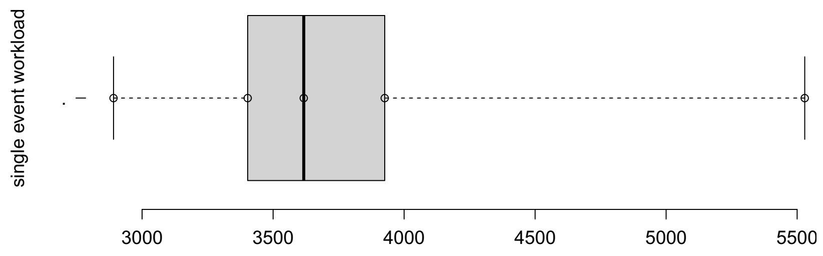
$$SyntheticEventWorkload(size) := \sum_{i=0}^{size} \left(2e^{i}\sqrt{i}\right) \mod \left|\left\lfloor \frac{i+5}{i+1} \right\rfloor\right| , \qquad (1)$$

where *size* can control the single event workload, while the denoted operations ensure that the distribution of the single event execution time is closely matching the above mentioned more realistic simulation.

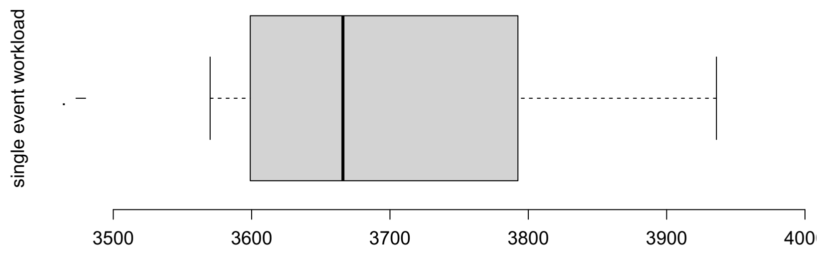
To ensure that the workload produced by this busy waiting loop is equivalent to the realistic simulation, we have executed the same number of events we have recorded in the realistic simulation and repeated the measurement 100 times. The repetition allowed us to collect several statistical properties of the single event workload in both the synthetic and the realistic simulations. We present our findings for the realistic simulation in the box plot of Fig 3a. Our best approximation of this realistic workload was captured by our synthetic workload parametrised with $size = 49$.

Fig 3b shows the behaviour of our best approximate synthetic workload. Our median duration is within 3% of the realistic. The distribution of our workload is a bit narrower and more even, but the upper and lower whiskers of our synthetic experiment are within the typical range of the realistic simulation's values. As a result, from this point onwards, we will refer to synthetic workloads set up with this particular parameter as the *original* single event workload.

Note, that later we have evaluated the system with other single workloads. For example, changing the *size* to 147, leads to a threefold increase in single event workload compared to the realistic setting. In contrast, changing it to 16, leads to a three fold reduction in single event workload again compared to the realistic setting. These two values will be the extremes used in Figure 5.



(a) The execution time(ns) of single event workload using JobDispatchingDemo class



(b) The execution time(ns) of single event workload using our experiments classes

Figure 3: Boxplot diagrams for JobDispatchingDemo class and our classes

4.2 Performance

Our evaluation scenarios create 35,000 recurrent event objects. The object count was set so the minimum execution time of the sequential version is at least 5 minutes, allowing sufficient time for the parallelisation to take effect. The recurring events subscribe with different frequencies so we have control over the degree of parallelism. We provided controls to these scenarios, so we can easily adjust the degree of parallelism (through event subscription changes) and the single event workload (through changing the *size* in Equation 1). The evaluation scenarios are publicly available in the ParallelTimed package released in the dissect-cf-examples project on GitHub[1] using the GPL 3.0 license.

The invocation of `Parallel` class depends on the threshold value (see Algorithm 1 for details) to determine the maximum length of the event list processed by a single thread. To determine the ideal setting for the threshold, we evaluated our solution with four different values: 8, 16, 32 and 64. We have also generated recurring events with four different degrees of parallelism as shown in table 2. Based on our analysis of the execution times in the table. Even though the differences are not big, it is recommended to use a threshold equal or exceed 32 to enhance the performance.

With the respect to the number of cores, there are two factors that influence the performance of the parallel version. First, the degree of parallelism plays a

[1]https://github.com/dilshadsallo/dissect-cf-examples

Table 2: The execution time(s) of parallel version using four different sizes of list

Degree of Parallelism				
Threshold	**25%**	**50%**	**75%**	**100%**
8	235	412	549	657
16	234	411	548	657
32	231	408	545	654
64	229	406	541	652

significant role and it is shown in Fig 4 that the parallel version can significantly improve performance. We evaluated both the parallel and sequential versions of the simulator with four different degrees of parallelism (25%, 50%, 75%, 100%). Even though the evaluation of this scenario has been done with the same number of aforementioned objects, the number of events that occur, and the number of events that occur at the same time significantly increase. This is because we simulated for the same amount of simulation time, but with increasing subscription frequency each object receives more event notifications. E.g., to increase the degree of parallelism on the scenario in table 1, we can change the subscription frequency of event 2 to 1. In this example, the degree of parallelism increases to 73%, but we see more event notifications delivered as we will have 15 notifications for event 2 as well.

With regards to Fig 4, in 25% of parallelism, the parallel version runs 1.72 faster than the sequential. When the degree reaches 50%, the ratio increased to 1.74. The parallel version executes simulations 1.84 faster than the sequential version when 75% of all subscribed events occur recurrently during a simulation time. Finally, the parallel version reaches 2 times faster than the sequential version when the degree of parallelism is 100%. Even with a high degree of parallelism and using multi-core, we cannot use all cores because there is still a performance cost such as coordinating threads that introduced by multi-thread compared to a single-threaded approach.

Now let's analyse the effect of the size of the single event workload (as per Equation 1). We tested both of the parallel and sequential versions with various single event workload sizes, commenced with threefold lower than the *original* one to show the behaviour of simulating very low single event workload. Then reaching to threefold higher than the *original* single event workload to demonstrate the advantage of parallel version as shown in Fig 5. When the single event workload is threefold lower than *original* one, the parallel version runs 1.2 faster than the sequential version. This ratio increases to 1.6 when the single event workload is two times lower than the *original* single event workload. The parallel version even runs 2.1 faster than the sequential version using *original* single event workload. When the single event workload size doubled, the parallel version executes the simulation 2.3 times faster than the sequential version. The ratio increases to 2.4 when the single event workload size becomes threefold higher than the *original* one.

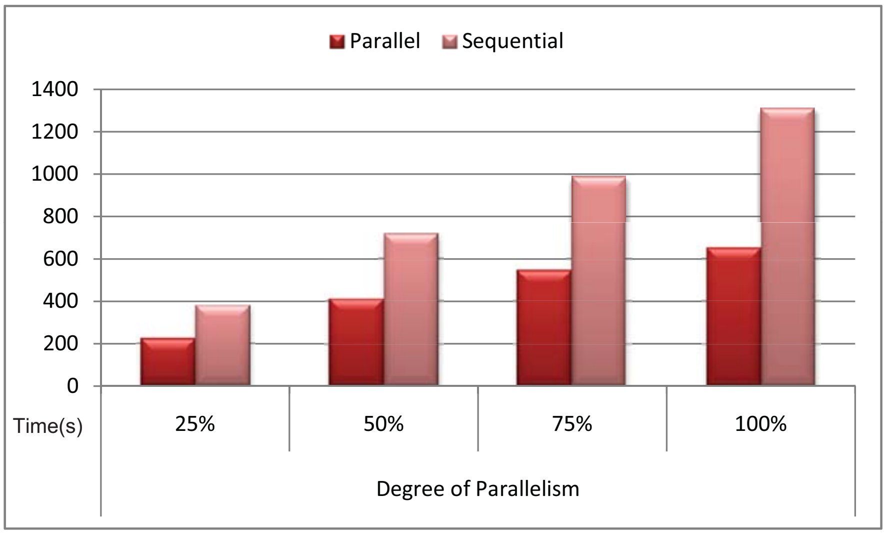


Figure 4: The execution time(s) of parallel and sequential versions in four different degrees of parallelism

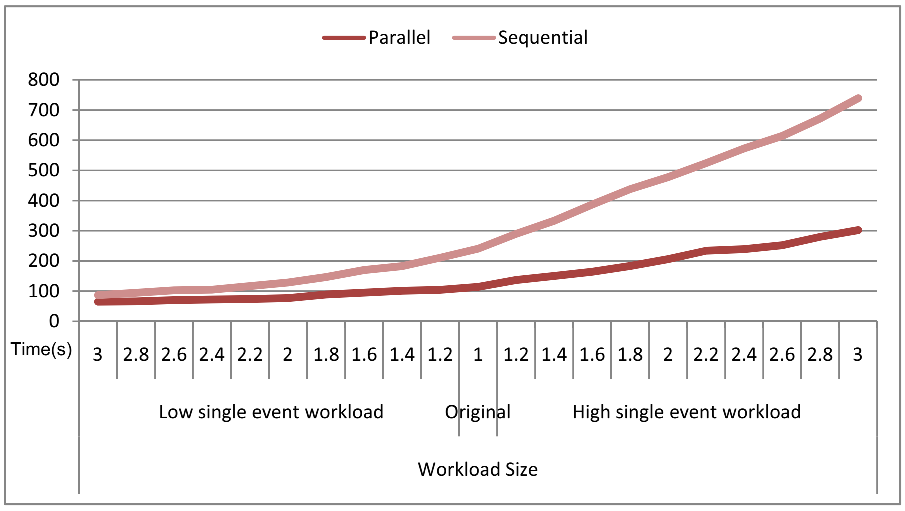


Figure 5: The execution time (s) of different workload sizes simulated by parallel and sequential versions

Thus, the parallel version speeds up the performance of simulation by using the additional cores of the host. The biggest advantages of the parallel version can be exploited when there are larger simultaneously occurring event queues and when the single event workload is larger as well. In the realistic simulations, we have seen that simultaneously occurring event queues are typically larger when advanced features of the simulator are fully utilised (e.g., cloud wide energy metering or virtual machine consolidation).

5 Conclusion and future work

Mostly DES frameworks are used to simulate and evaluate cloud computing environments. The majority executes sequentially. DISSECT-CF is one of the frameworks that brought several features to improve the performance of IaaS simulation. It is built to accompany the latest technology with easy extensibility. In terms of execution, DISSECT-CF was already fast and reliable but still targeted a single core. We devised a parallel version to handle this issue focusing on the use of multi-core when simultaneous events happen in the simulation. The parallel version scales well and leads to significant speed up. The performance of the parallel version is dependent on the number of simultaneous events at a particular time instance in the simulation, as well as on the workload a single event's processing causes. Our introduced parallel execution mode is focused on the event subsystem, as this is the lowest layer in DISSECT-CF all other components benefit from our improvements.

Future work will focus on the simulator's second most heavily used component: the unified resource sharing subsystem. As this subsystem is having high compute complexity, its parallelisation will enable the rapid estimation of resource sharing on even larger scale distributed systems. Applying these will lead to the seamless transition of DISSECT-CF into simulating more communication intensive systems, or evaluating fog computing & IoT device behaviour.

References

[1] Ahmed, Arif and Sabyasachi, Abadhan Saumya. Cloud computing simulators: A detailed survey and future direction. In *2014 IEEE International Advance Computing Conference (IACC)*, pages 866–872. IEEE, 2014. DOI: `10.1109/IAdCC.2014.6779436`.

[2] Byrne, James, Svorobej, Sergej, Giannoutakis, Konstantinos M, Tzovaras, Dimitrios, Byrne, Peter J, Östberg, Per-Olov, Gourinovitch, Anna, and Lynn, Theo. A review of cloud computing simulation platforms and related environments. In *International Conference on Cloud Computing and Services Science*, volume 2, pages 679–691. SCITEPRESS, 2017. DOI: `10.5220/0006373006790691`.

[3] Calheiros, Rodrigo N, Ranjan, Rajiv, Beloglazov, Anton, De Rose, César AF, and Buyya, Rajkumar. Cloudsim: a toolkit for modeling and simulation

of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and experience*, 41(1):23–50, 2011. DOI: `10.1002/spe.995`.

[4] Carothers, Christopher D, Bauer, David, and Pearce, Shawn. Ross: A high-performance, low-memory, modular time warp system. *Journal of Parallel and Distributed Computing*, 62(11):1648–1669, 2002. DOI: `10.1016/S0743-7315(02)00004-7`.

[5] D'Angelo, Gabriele and Marzolla, Moreno. New trends in parallel and distributed simulation: From many-cores to cloud computing. *Simulation Modelling Practice and Theory*, 49:320–335, 2014. DOI: `10.1016/j.simpat.2014.06.007`.

[6] Das, Samir, Fujimoto, Richard, Panesar, Kiran, Allison, Don, and Hybinette, Maria. Gtw: a time warp system for shared memory multiprocessors. In *Proceedings of Winter Simulation Conference*, pages 1332–1339. IEEE, 1994. DOI: `10.1109/WSC.1994.717527`.

[7] Fujimoto, Richard M. Research challenges in parallel and distributed simulation. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 26(4):1–29, 2016. DOI: `10.1145/2866577`.

[8] Fujimoto, Richard M, Malik, Asad Waqar, Park, A, et al. Parallel and distributed simulation in the cloud. *SCS M&S Magazine*, 3:1–10, 2010.

[9] Kathiravelu, Pradeeban and Veiga, Luis. Concurrent and distributed cloudsim simulations. In *2014 IEEE 22nd International Symposium on Modelling, Analysis & Simulation of Computer and Telecommunication Systems*, pages 490–493. IEEE, 2014. DOI: `10.1109/MASCOTS.2014.70`.

[10] Kecskemeti, Gabor. Dissect-cf: a simulator to foster energy-aware scheduling in infrastructure clouds. *Simulation Modelling Practice and Theory*, 58:188–218, 2015. DOI: `10.1016/j.simpat.2015.05.009`.

[11] Kliazovich, Dzmitry, Bouvry, Pascal, and Khan, Samee Ullah. Greencloud: a packet-level simulator of energy-aware cloud computing data centers. *The Journal of Supercomputing*, 62(3):1263–1283, 2012. DOI: `10.1007/s11227-010-0504-1`.

[12] Liu, Jason, Nicol, David, Premore, Brian, and Poplawski, Anna. Performance prediction of a parallel simulator. In *Proceedings Thirteenth Workshop on Parallel and Distributed Simulation. PADS 99.(Cat. No. PR00155)*, pages 156–164. IEEE, 1999. DOI: `10.1109/PADS.1999.766172`.

[13] Mann, Zoltán Ádám. Cloud simulators in the implementation and evaluation of virtual machine placement algorithms. *Software: Practice and Experience*, 48(7):1368–1389, 2018. DOI: `10.1002/spe.2579`.

[14] Ostermann, Simon, Plankensteiner, Kassian, Prodan, Radu, and Fahringer, Thomas. Groudsim: an event-based simulation framework for computational grids and clouds. In *European Conference on Parallel Processing*, pages 305–313. Springer, 2010. DOI: `10.1007/978-3-642-21878-1_38`.

[15] Yoginath, Srikanth B and Perumalla, Kalyan S. Optimized hypervisor scheduler for parallel discrete event simulations on virtual machine platforms. In *SimuTools*, pages 1–9, 2013. DOI: `10.4108/icst.simutools.2013.251736`.

Acta Cybernetica **25** (2021) 485–516.

Evaluation of EHR Access Control in a Heterogenous Test Environment[*]

Zoltán Szabó[ab] and Vilmos Bilicki[ac]

Abstract

Since the advent of smartphones, IoT and cloud computing, we have seen an industry-wide demand to integrate different healthcare applications with each other and with the cloud, connecting multiple institutions or even countries. But despite these trends, the domain of access control and security of sensitive healthcare data still raises a serious challenge for multiple developers and lacks the necessary definitions to create a general security framework that addresses these issues. Taking into account newer, more special cases, such as the popular heterogeneous infrastructures with a combination of public and private clouds, fog computing, Internet of Things, the area has become evermore complicated. In this paper we will introduce a categorization of the required policies, describe an infrastructure as a possible solution to these security challenges, and then evaluate it with a set of policies based on real-world requirements.

Keywords: EHR, FHIR, telemedicine, cloud, access control, security

1 Introduction

In the mid-2010s with the emergence of the Fast Healthcare Interoperability Resources (FHIR) standard from HL7 [1], it seemed that we finally had the necessary tools to create e-health applications and databases that not only meet their respective institutional requirements, but also conform to international standards, making a networked health infrastructure more feasible. FHIR achieved this by defining a set of over 90 document templates that can be implemented in both JSON and XML formats and used to describe the entire healthcare workflow from administration to the daily events that a general practitioner or nurse is confronted

[*]This research was supported by the EU-funded Hungarian grants EFOP-3.6.1-16-2016-00008, EFOP-3.6.3-VEKOP-16-2017-00002, 2018-1.1.1-MKI-2018-00249 and GINOP-2.2.1-15-2017-00073

[a]Department of Software Engineering, University of Szeged, Hungary

[b]E-mail: `szaboz@inf.u-szeged.hu`, ORCID: `0000-0003-3863-7595`

[c]E-mail: `bilickiv@inf.u-szeged.hu`, ORCID: `0000-0002-7793-2661`

DOI: `10.14232/actacyb.290283`

with. FHIR has also made these documents customizable to meet specific requirements and cover specific areas. These attractive aspects have made FHIR the most popular and widely used healthcare communications standard from HL7 to date.

However, the FHIR standard had some alarming shortcomings [7, 10]. Although some of these have been addressed in the course of the various updates to the standard, one of the most pressing is still an open problem - namely the lack of clearly defined access control and security. While FHIR generally accepts custom extensions and adaptations of its standardized document types, it provides only a light template and some minor guidelines for security policy enforcement. This has led to a "free-for-all" problem in the development of e-Health applications, with almost everyone developing their own solutions, which greatly corrupts the original concept of interoperability. With the introduction of the GDPR [12], the increasing integration of IoT and intelligent devices into the healthcare workflow [18] and in some cases the decision to use a heterogeneous backend [32] for handling accessibility and sensitive data, the complexity of this issue has increased. Another complicating factor is that the most popular current technologies for the backend are serverless and native cloud infrastructures. These present two main problems: with the advent of fog/edge computing, data processing takes place much closer to the end devices and user locations, and in these cases a large amount of sensitive data is stored in a public cloud.

The two main approaches recommended by the official documentation of the FHIR standard for these challenges are the attribute- or role-based policy controls, ABAC [36] and RBAC [15], respectively. With RBAC, the developer is able to assign specific roles to users that determine the level of access and possible operations in the system. Some typical roles in a medical system would be those of a doctor, nurse, patient, family member, and so on. In contrast, ABAC uses specific attributes of the user or the requested data to determine whether access should be granted.

However, in the current network topology, which combines IoT, intelligent devices, edge computing, private and public clouds, these methods in themselves are far from sufficient. To meet these needs, it is essential to develop a hybrid approach that combines the strengths of these two classical methods. Furthermore, it is important that these enforcement points can be placed at any part of the infrastructure to deal with the sensitive nature and processing requirements of the data. For example, while fog endpoints require a complete FHIR object, the connecting IoT devices may not be able to handle such complex data structures. In the case of a hybrid cloud solution, the data could also pass through a public cloud between the private cloud and end users, where naturally stricter policies and encryption are required than for the isolated, private parts of the infrastructure, as shown in Figure 2.

A popular concept for such enforcement points is the concept of Policy Enforcement Point (PEP), developed by the standards organization OASIS as part of its eXtensible Access Control Markup Language standard [14], an extension of the classic ABAC model, also known as policy-based access control or PBAC. While we are committed to developing a custom, fine-grained security solution that is

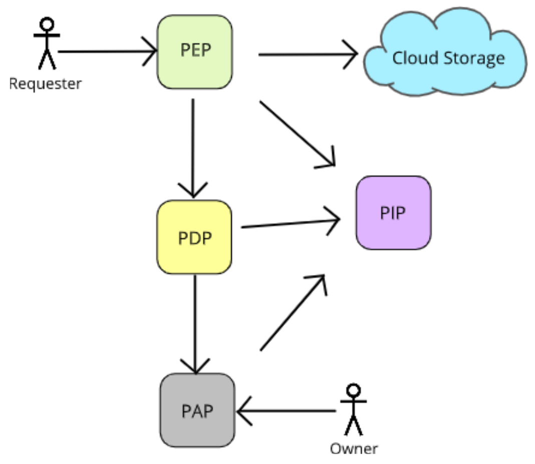


Figure 1: The Policy Enforcement Flow between end users and the cloud

not subject to the strict limitations of the XACML standard, the concept of the PEP-based architecture is well suited to our needs. In the complete model shown in Figure 1, the responsibilities for access control and security enforcement are distributed across several components. The Policy Enforcement Point (PEP) is the key to the model that enforces policies and allows or denies access to resources. Administrators can define given policies at the Policy Administration Point (PAP), which are evaluated and stored by the Policy Decision Point, based on the user's identity or multiple identities, and recognized by the Policy Identification Point (PIP). The PDP's decision is handled and enforced by a PEP. This model also provides some room for customization, because the exact structure of these nodes can be defined by the developers, and the nodes have the ability to fuse multiple elements of the infrastructure into one.

The main requirements of such a PEP solution in our infrastructure are the following:

- **Transparency**: It should have as little impact as possible on the performance and latency of the system;
- **Efficiency**: Since several elements of the infrastructure do not have the memory and CPU capacity to perform complex transformations and an analysis of the data, the enforcement engine should spare them from the more demanding operations;
- **Portability**: It should be possible to place it at any point on the infrastructure. The main strength of edge computing is its ability to provide functionality even when the cloud is not available. This also means that it should be able to work between the edge and the cloud, between the edge and end-

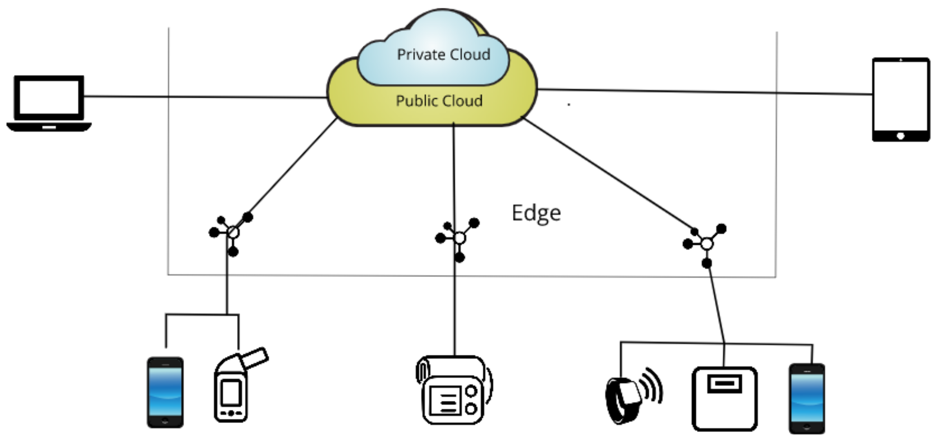


Figure 2: A hybrid edge computing infrastructure with a public and a private cloud

points, in the cloud or in some scenarios even on endpoints if they have the necessary resources;

- **Adaptability**: The domain of telemedicine requires strict, very specific guidelines to protect sensitive data. The reason why ABAC and RBAC by themselves are not enough is that the requirements for interoperability and interchangeability demand a much more dynamic and fine-grained approach. The PEP should support the formulation and assessment of even the most specific needs.

Our work seeks to combine these two approaches while separating the access control process from the backend and frontend and putting it on the path of the data between the cloud and the end users. In our previous study [33] we introduced the concept of a hybrid access control methodology, taking the classical roles of the telemedicine environment and assessing its requirements based on the content of the documents and the contextual information. Later, we described a theoretical infrastructure [34] which, based on our assumptions, should allow us to effectively implement this methodology in a clinical environment. In the latter case, we not only defined the four main categories of required policies, but also selected a potential candidate as our policy enforcement point and performed various tests on FHIR documents while monitoring CPU load and memory usage. These early evaluations demonstrated that our concept is viable.

Since then, we have been able to develop a prototype of this infrastructure, which integrates the chosen PEP between a small client application and a scalable NoSQL database with over 500,000 FHIR documents.

In this paper we investigate the performance and relative latency of a PEP engine as part of the larger infrastructure. To this end, in the Related Work section we give an overview of several possible solutions and research projects that, with at least some degree of similarity to our study, also sought to combine access control methods and define complex, interoperable security frameworks for health care. In the next section we describe our results obtained so far in greater detail and present the test environment and the various policies for the evaluation process. Then, in the Results section we share and analyze the results of our tests to determine trends in the evaluation process and possible patterns, anti-patterns for our future work. Finally, in the Conclusions section we summarize these results and provide an outlook on future steps and possible new directions for research.

2 Related Work

While the authors of a 2013 comparative study based on 775 reviewed articles found that RBAC [13] was the most popular approach to manage access control in healthcare, this trend changed significantly with FHIR, and the preference between ABAC and RBAC became the de facto choice of the development team rather than industry standards.

For example, the developers of the application atHealth [31] succeeded in implementing a role-based methodology for their mobile application in 2017, recognising the lack of security in FHIR. However, there also were implementations of the ABAC model for access control to health records [21] in the same year.

To further complicate the issue of these two models, as early as 2008 [23] there were critics who noted that access control in healthcare systems is sufficiently complex to justify situation-based decisions, with the classical concept of roles and attributes oversimplifying the issue. When we conducted our first experiments in 2018 in this area [33], we also found that neither ABAC nor RBAC as such is sufficient to meet the needs of practitioners and clinical applications, because even though roles are important elements of security, they cannot cover every situation without specific, contextual information.

Although there are platforms, such as the popular SMART project [11], which offer a solution in the form of a full OAuth2 integration into their FHIR database, the use of such frameworks usually comes at the expense of a certain degree of freedom in the choice of health infrastructure components. There have been several attempts to define hybrid solutions both in healthcare [24] and more generally in multi-modal, heterogeneous environments [9]. A key concept of the domain is the requirement to control access not only to entire documents, but also to specific fields and attributes in documents. The proposed architecture by Rezaeibagha, F. et al [28] is specifically designed to move sensitive data from a secure private cloud to a public one, while maintaining security.

In 2016, Pusselwalage H. S. G. et al. [25] published their approach for an ABAC methodology that bases its policies not only on the attributes of the data but also on the attributes of the user, treating the different levels of access and the classical roles in healthcare as attributes. They combined the two models to some extent, while also highlighting special cases such as unregistered users or registered users without a specific role. In 2018 Joshi M. et al. [20] used a similar approach with roles treated as attributes, but instead of granting full access, their solution also transformed the requested data to match the requester's access level. The developers of the SOCIAL platform [29] also discussed some interesting ideas about treating the requesting device as an important component of ABAC with a combination of the user attributes.

During our review we also found some studies that appeared to combine elements of the RBAC and ABAC models without clearly categorizing their methodology as a hybrid. The developers of the [26] H-Plane Framework, which follows the terminology of the ABAC model, also apply several attributes in a way that is almost identical to some aspects of RBAC. In their publication they also pointed out the importance of the IoT in this domain. In 2019, Alnefaie, S. et al. [6] after reviewing the possible alternatives for access control they thought ABAC was much better suited to the needs of healthcare in combination with edge computing, but also suggested modifying the infrastructure of the methodology to bring the point of evaluation closer to the edge and place more emphasis on the identity of the IoT device itself. Tasali Q. et al. [35] extended this concept by covering not only medical data, but also the authorization process for real-time communication between IoT devices.

The solution proposed by the developers of the mHealth application [22] is also quite similar to ours, the main differences being that its policy engine is deployed as part of the cloud services and the engine is an implementation of the NIST NGAC framework, with the evaluation process based on traversing a Neo4j graph database. The infrastructure and principle designed by Ray, I et al. [27] also have similar features, with policy enforcement based on the XACML format.

To summarize the state-of-the-art based on these sources:

- A modern solution should either extend the traditional access control ABAC model or develop a custom hybrid solution to meet the needs of the domain;

- Heterogeneous storage should be taken into account and the sensitive documents must be transformed before they enter the public cloud;

- The IoT raises brand new challenges. The security solution must be able to handle the different capabilities and requirements of these tools when evaluating and converting the healthcare data.

It is clear that our approach is only one of many proposals that seek to resolve the security issue of EHR. Our goal is to combine the best ideas and elements of the domain - combining RBAC and ABAC, establishing the included PEP nodes as a middle layers between the private and public clouds, public cloud and edge network, etc. - and also to improve and extend them, to provide support for every database and application that uses FHIR, and to provide users and developers with a trusted, verified solution to the security problem.

3 Our Approach

3.1 Telemedicine Security Infrastructure

The goal of our research is to create a solution that is able to support several different storage providers and at the same time make the infrastructure shown in Figure 3 transparent to the end users. While the use of such heterogeneous backends is recommended in various use cases, the field of telemedicine is the prime example of how the strengths of this model can be brought to bear. Even before the GDPR, the storage and encryption of data was a major challenge, and while public cloud providers proved to be very popular with telemedicine developers, the storage in such solutions required high-level encryption and data transformation. However, with heterogeneous storage, it is possible to store the sensitive data in a private, more secure cloud and the less sensitive information in a public cloud or even in a custom database to improve and optimize the efficiency of the system as a whole. To achieve this, we needed to place our entire policy enforcement flow - the Policy Enforcement Point, Policy Decision Point, and Policy Identity Point - between the back-end and front-end and connect it to a proxy. We chose a Squid proxy [30] implementation to achieve the latter and configured it to forward each request, but upon receipt of a response containing FHIR structures, send it to the PEP for filtering and (if necessary) transforming before forwarding it to the end user. This provides a necessary middlelayer, unlike most solutions we discussed in the previous section. In this way, policy enforcement can take place outside the cloud, which allows the use of a heterogeneous storage solution (provided that it uses the FHIR standard as the format of the stored documents), but it also relieves the burden on the end systems. This approach is not only better optimized in terms of efficiency and capacity, as some of the end systems may not have the required capabilities, but it also ensures that sensitive information never reaches the end users without prior assessment and filtering. It should be added that with such a proxy, developers are also able to log in detail the various operations on the telemedicine records in order to comply with the GDPR.

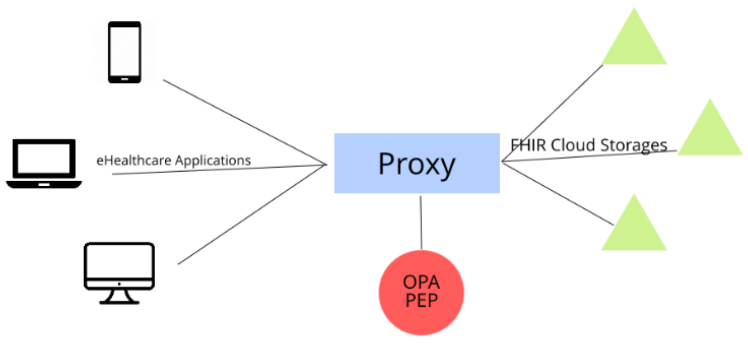


Figure 3: A general outline of our proposed solution

To test our concept of policy enforcement, we chose a promising new solution called Open Policy Agent [4], available in Go and WebAssembly, which could play every role in the enforcement process. There is also an OPA implementation in WebAssembly - this way, if it turns out to be an effective PDP/PEP solution, it could be placed on any part of the infrastructure (or even in multiple locations simultaneously) to meet another one of our requirements. OPA also permits us to store the information necessary for decision making in JSON format and define the various policies in its own scripting language, Rego, which can later be accessed via a well-defined REST interface with an HTTP POST request containing the contextual information to be filtered or evaluated (in our case, the medical records).

3.2 Test Environment

At this stage of our research, we decided to use the cleanest possible test environment. To achieve this, we created the prototype infrastructure on a local network connected via WiFi (at 5 GHz frequency with only the elements of the prototype infrastructure connecting) to the various nodes hosting different actors of our model, instead of testing with a well-known cloud provider like Google Firebase. This means having:

- A desktop PC running Windows 10 on an AMD Ryzen 5 processor at 3.59 GHz speed and 16 GB DDR4 memory ran the client application on a Kingston SSDNow V300 SSD with 120 GB capacity and 450 MB/s reading speed, acting as the controller node of the environment;
- A laptop running Windows 10 on an Intel i5 processor at 2.49 GHz speed with 8 GB DDR4 and a 120 GB SSD with a reading speed of 423 MB/s memory hosted the MongoDB v3.2.1 [2]-based backend along with a lightweight REST API that handled the requests and query parameters;
- A secondary laptop with similar attributes hosted the Squid proxy written in NodeJS 10.14;
- An iMac running macOS Catalina with an Intel i5 processor at 2.9 GHz speed and 20 GB DDR3 memory hosted the OPA v0.23.2 runtime with a hard disk of size 1 TB and reading speed of 210 MB/s.

The database was loaded with over 500,000 different FHIR Observation documents, based on properties from the MIMIC3 database [19], involving 200 patients, 30 doctors and 12 nursing teams, all signed with a different time stamp between 2015 and 2020. The template and structure of these Observations were taken from one of our industry projects to simulate the size and complexity of healthcare documents in a real system. The measurements were performed by a monitoring host that issued the restarts and reinitializations of each element of the architecture between measurements. During the experiment, each component was configured so that its output was logged in separate files that were collected and evaluated by the monitor at the end of a round. Each rule ran on 8 different input sizes: 10, 20, 50, 100, 200, 500, 1000 and 2000 data sets. The PEP engine received exactly the same amount of data in each round, but of course the size of the output varied from case to case, depending on the selected policy and the content of the input.

3.3 Evaluation Role Set

In our previous paper [34], we defined the four main categories of policies required by the domain to evaluate health data before it reaches the external network and end-user applications. We based this categorization on several sources and overviews of the domain [8] [17] [16] [5], from which we were able to determine the basic and extended safety requirements of the health sector. In accordance with the GDPR, every user of the system must naturally have full control over their data. The patient is the primary owner, the physician who wrote the document or assisted in its creation is the secondary, while other practitioners and relatives can have access to it to some extent. The system must also handle indirect access when the applicant, as a member of a group, tries to access the file. These respective types of access must be identified based on a combination of user roles, role groups and the attributes of the FHIR documents.

In some cases, contextual information is also required to determine the degree of access. For example, while a general practitioner should be able to access patient records at any time (logging the exact time and nature of such access), a nurse or assistant should not allowed to exceed the prescribed office hours. For some especially sensitive information, other contextual information such as the physical location of the requester, the ID of the device from which the request originates, should also be used in the evaluation, and expanding this set compared to a simple role definition is enough to justify a separate category.

A key requirement in the field of healthcare is that access does not mean full access to every element of the given document. In many cases it is strictly forbidden to grant access to such information from which a third party might be able to reconstruct very sensitive events and elements. For example, if one receives a list of a patient's medicines from a certain period of time, it is easy to infer vital information that would otherwise be prohibited for that particular third party. The evaluation process in a healthcare environment should be able to determine access at a very fine granularity, essentially at the field-by-field level, and to mark or even remove certain fields that should not be available at that security level. This is also the reason why the standard security solutions of several large cloud providers and databases fails, as they can only provide this functionality by including lambda functions, trigger functions, and the like.

The last requirement is also the most unique and difficult aspect of healthcare security. The break-the-glass case requires an access control model that provides immediate access to key patient information to ensure the receiving of the necessary, possibly life-saving care. This is essentially what happens in an emergency, when life-saving surgery is required and neither the patient nor the doctor recording and processing their health data is available to grant access. In a break-the-glass situation usually only a few records are required, but in that case it is important to use very complex transformations. Only vital information should be accessed, while every other element of the document must be either removed or encrypted. Without the effective implementation of break-the-glass, no healthcare security system can be used in real-life situations.

Based on these requirements, the definitions of our policy categories are:

- **Role Evaluation**: The policy has to determine whether based on the user's role or roles in the system, partial or full access should be provided;
- **Contextual Evaluation**: The policy has to determine whether the combination of the user's role, various attributes and contextual information form the basis for partial or full access;
- **Contextual Modification**: Aside from providing access, the policy should also transform the data, removing or altering specific fields;
- **Break-the-Glass**: A specific requirement of a healthcare application. In the case of an emergency, the policy should provide immediate access, while also encrypting or removing sensitive information.

Our first step during the evaluation process was to test our evaluation concept with different policies and different pressures. During this first phase we ran our tests on OPA without including other elements of the proposed infrastructure and stored the FHIR documents in its database, making it essentially a temporary FHIR database. We measured attributes such as CPU load and memory usage, while increasing the size of the input data set by a power of ten after each iteration, up to a data set of one million records. The results of these experiments demonstrated that the combined PDP/PDE/PIP concept was an acceptable candidate.

These results paved the way for the next step of our research: the integration of the combined PDP/PDE/PIP node (OPA) with a prototype infrastructure and the further evaluation of its effectiveness and latency in this environment.

3.4 The Telemedicine Security Abstract Role Set

First and foremost we defined a formal specification of each category, with the following notations:

- $\mathbb{F} := \{f_1, ..., f_k\}$ marks the telemedicine record in question, where each f_x is a valid key-value pair of the record.
  For example:
  $\mathbb{F} := \{('subject', 'PAT/1'), ('systolic\_bloodpress', 120), ...\}$
- $\mathbb{UR} := \{r_1, ..., r_l\}$ where $\mathbb{UR} \subseteq \mathbb{F}$ is a subset containing the key-value pairs describing various primary or secondary owners of the record
  For example:
  $\mathbb{UR} := \{('subject', 'PAT/1'), ('practitioner', 'PR/A013'), ...\}$

- $\mathbb{EX} := \{e_1, ..., e_m\}$ marks the external context of the system at the time of the policy evaluation as key-value pairs
  For example:
  $\mathbb{EX} := \{('datetime','2020{-}09{-}12T12:20:33'), ('ip\_addr','223.134.22.1'), ...\}$
- $\mathbb{CX} := \{c_1, ..., c_k\}$ is a set of conditional functions, which take an atomic value as an argument and transform it to a boolean value. Each function is represented as an (op, val) pair where op is a conditional operation, $op \in \{<, >, \leq, \geq, =\}$ and $c_i(n) := n\ op_i\ val_i$
  For example:
  $c_1 := (>, 12), x := 5 \implies c_1(x) := 5 > 12 \implies c_1(x) := \text{false}$
  $c_2 := (=, 'bloodpressure'), x := 'bodyweight' \implies$
  $c_2(x) :=$ 'bodyweight' = 'bloodpressure' $\implies c_2(x) := \text{false}$
- $\mathbf{P}(n)$ is a function describing a policy to be enforced by a PEP engine, where $f_x \in \mathbb{F}$ and $\mathbf{P}(f_x) = Allow|Modify|Deny$ produces the decision regarding the evaluated key and $\mathbf{P}(\mathbb{F}) := \{\mathbf{P}(f_1), ..., \mathbf{P}(f_k)\}$

3.4.1 Formal Definition of the Role Evaluation Policy

Definition 1. ***P(n)*** *describes a Role Evaluation policy, if* $\mathbb{UR} \neq \emptyset$ *and for a given user identifier* $\exists key(key, id) \in \mathbb{UR}$*, then* $\boldsymbol{P}(n) := \forall f_i \in \mathbb{F}\ \boldsymbol{P}(f_i) := Allow$*, else* $\boldsymbol{P}(n) := \forall f_i \in \mathbb{F}\ \boldsymbol{P}(f_i) := Deny$

3.4.2 Formal Definition of the Contextual Evaluation Policy

Definition 2. ***P(n)*** *describes a Contextual Evaluation policy if* $\mathbb{G} := \mathbb{F} \cup \mathbb{EX}$, $\mathbb{CE} := \{ce_1, ..., ce_x\}$ *is a set of contextual conditions where* $0 \leq i \leq x$, $ce_i := (key_i, c_i)$, $c_i \in \mathbb{CX}$ *and* $\exists x : (key_i, value_i) \in \mathbb{G}$ *and* $\boldsymbol{P}(n) := \forall f_i \in \mathbb{F}\ \boldsymbol{P}(f_i) := Allow$*, if* $\forall x : (key_i, c_i) \in \mathbb{CE} : \exists (key_i, value_i) \in \mathbb{G}$ *and* $c_i(value_i) := true$*, else* $\forall f_i \in \mathbb{F}\ \boldsymbol{P}(f_i) := Deny$

3.4.3 Formal Definition of the Contextual Modification Policy

Definition 3. ***P(n)*** *describes a Contextual Modification policy if similarly to the Contextual Evaluation policy,* $\mathbb{G} := \mathbb{F} \cup \mathbb{EX}$, $\mathbb{CE} := \{ce_1, ..., ce_x\}$ *is a set of contextual conditions where* $0 \leq i \leq x$, $ce_i := (key_i, c_i)$, $c_i \in \mathbb{CX}$ *and* $\exists x : (key_i, value_i) \in \mathbb{G}$ *but there is also a* ***FX*** *mapping, which* $\boldsymbol{FX} : \mathbb{CE} \implies \mathbb{F}' \subseteq \mathbb{F}$*, with* $\cap_{0 \leq i \leq x} \boldsymbol{FX}(ce_i) := \emptyset$*.* $If 0 \leq i \leq x\ ce_x(g_x) := false\ then$
$\forall f'_i \in \boldsymbol{FX}(ce_i)\ \boldsymbol{P}(f'_i) := Deny,\ else\ \forall f'_i \in \boldsymbol{FX}(ce_i)\ \boldsymbol{P}(f'_i) := Allow$

3.4.4 Formal Definition of the Break-the-Glass Policy

Definition 4. ***P(n)*** *describes a Break-the-Glass policy if* ***P(n)*** *satisfies the requirements of the Contextual Modification with the further addition of a* ***TX*** *mapping, identifying the attributes which have to be encrypted or modified* $\boldsymbol{TX} : \mathbb{CE} \implies \mathbb{F}'' \subseteq \mathbb{F}$*, with* $\cap_{0 \leq i \leq x} \boldsymbol{TX}(ce_i) := \emptyset$ *and* $\cup_{0 \leq i \leq x} \boldsymbol{FX}(ce_i) \cap \cup_{0 \leq i \leq x} \boldsymbol{TX}(ce_i) := \emptyset$.

$If 0 \leq i \leq x\ ce_x(g_x) := false\ then\ \forall f_i'' \in \boldsymbol{FX}(ce_i)\ \boldsymbol{P}(f_i'') := Deny,\ else\ \forall f_i'' \in \boldsymbol{FX}(ce_i)\ \boldsymbol{P}(f_i'') := Modify$

3.5 Implementation Details

To achieve this goal, we defined a new set of guidelines based on the actual needs of a healthcare system, instead of the proof-of-concept drafts from our previous study. We defined two rules for each of the four categories - one simpler and one more complex, the latter containing more operations or more resource-intensive operations, or both. All algorithms run on multiple Observations received as part of the input, but only returned those in their original or modified state which were allowed by the policy.

The two policies of the Role Evaluation category included in algorithms 1 and 2 both focus on the identity of the practitioner. However, while *role_simple* only checks to see that the specified role identifier matches the practitioner's identifier in the document, *role_complex* checks the care teams responsible for the patient and only grants access if the requester is a member of those teams.

The main difference between the policies of the Contextual Evaluation category, shown in algorithms 3 and 4, is the nature of the contextual attribute.

In *context_simple* we check a high-level attribute, the status of the Observation and an external attribute, the hour. Here *context_complex* requires the PEP iterating through the component array of each Observation, finding each medical value based on the defined LOINC identification code, and checking to see if the exact value is greater than the threshold.

A key aspect to be evaluated was the efficiency of the PEP when iterating and handling arrays, since in the current version of the OPA, the developers noted in the official documentation [3] that the performance of such an evaluation engine is the most efficient when it works with objects, and weakest when it must iterate through non-indexed arrays.

Algorithm 1 A Role Evaluation policy in Rego returning only the Observations where the current user is the Practitioner

```
Policy role_simple[$shell$]
1: $pract := input.practitioner$ ▶ We get the Practitioner id from the request
2: $shell := input.observations[\_]$ ▶ We create a working copy from the list of Observations
3: $observation := shell.data$ ▶ We iterate through the shell array and map the Observation inner object
4: $performer := observation.performer[\_]$ ▶ We map the list of Practitioners of the current observation
5: $performer.identifier.value == pract$ ▶ We check if one of the Practitioners has the same id as the input. If so, it remains in the shell array and will be returned, if not, it will be filtered out
```

Algorithm 2 A Role Evaluation policy in Rego returning only the Observations where the current user is member of one of the CareTeams, who received access from the Patient

```
Policy role_complex[$shell$]
1: $pr := input.practitioner$ ▶ We get the Practitioner id from the request
2: $shell := input.observations[\_]$ ▶ We create a working copy from the list of Observations
3: $observation := shell.data$ ▶ We iterate through the shell array and map the Observation inner object
4: $performer := observation.performer[\_]$ ▶ We map the list of Practitioners of the current observation
5: $cteam == performer.identifier.value$ ▶ We get the identifier of the Performer
6: contains('CareTeams', $cteam$) ▶ We call the built-in function to check whether the identifier belongs to a CareTeam, and only to proceed if it is true. If the Observation has no CareTeam at all, the evaluation returns false.
7: count(practition_member_of_careteam($pr$, $cteam$)) $> 0$ ▶ We call a simple function which checks whether the identifier of the Practitioner is listed as a member of a the CareTeam. If there is a match for even one of the CareTeams, to access is provided
```

Algorithm 3 A Contextual Evaluation policy where access is granted only if the status of the Observation is active and the time of access takes places between 8:00 and 19:00

```
Policy context_simple[$shell$]
1: $timearray$ := time.clock([time.now_ns(), 'Europe/Budapest']) ▶ Using the built in functions of Rego, we generate the array containing the parts of a time string
2: $hour := timearray[0]$ ▶ We retrieve the first element of the timearray which will always be the hour from the chosen timezone
3: $shell := input.observations[\_]$
4: $shell.data.status ==$ 'active'
5: $hour \geq 8$
6: $hour \leq 19$ ▶ These last three lines are executed at the same time, if all of them return as true, the Observation will be part of the result set
```

Algorithm 4 A Contextual Evaluation policy where the internal structure of the Observation is analyzed and only a certain type with its value above a predetermined limit can be accessed

```
Policy context_complex[$shell$]
1: $shell := input.observations[\_]$
2: $component := shell.data.component[\_]$ ▶ We iterate through the inner attributes of the Observation
3: $component.code.coding[0].code ==$ '32419-4'
4: $component.value > 5$
```

Algorithm 5 A Contextual Modification policy where we remove the Patient identifiers from Observation

```
Policy modif_simple[$shell$]
1: $observation$ := $input.observations$[_]
2: $clean$ := object.remove($observation$, ['patient', 'data']) ▶ We remove the external patient identifier and the entire nested data object - since in the current version of Rego we cannot modify existing key-value pairs only remove existing or add new ones
3: $inner\_new$ := object.remove(observation.data, ['subject']) ▶ We create a new nested data object without the subject fields
4: $shell$ := object.union($clean$, {'data': $inner\_new$}) ▶ We create the result Observation by merging the two filtered versions
```

Algorithm 6 A Contextual Modification policy where we remove a specific nested component from every Observation

```
Policy modif_complex[$shell$]
1: $observation$ := $input.observations$[_]
2: $clean$ := object.remove($observation$, ['data'])
3: $no\_components$ := object.remove($observation.data$, ['component']) ▶ We need to filter the nested object as well
4: $new\_components$ := [$component$|
5:     $component$ := $observation.data.component$[_];
6:     $component.code.coding$[0] != '18748-4'] ▶ We filter the nested object based on the LOINC codes
7: $new\_data$ := object.union($clean$, {'component': $new\_components$})
8: $shell$ := object.union('$new_data$', {'data': $new\_data$})
```

The majority of the documents in the FHIR standard use the array structure very often, and if the performance of array operations is significantly worse than in any other case, this would provide a strong argument against adapting our concept. The modification operation in a PEP node is very demanding in itself, since the engine treats every variable as a constant due to their non-imperative behavior. For this reason, if we need to modify or redefine a particular field, we must first create a copy of the original object without the value, then create a new value, and finally create the response by adding the new value to the filtered object copy. The Contextual Modification policies are shown in algorithms 5 and 6. Here *modif_simple* simply removes the patient data from the Observation and the encapsulating shell, while *modif_complex* must create a new component array for each Observation that does not contain urls pointing to sensitive patient documents.

The Break-the-Glass Policies shown in algorithms 7 and 8, are the most complex. These policies are expected to be fast, accurate, and effective, because they are most commonly used in emergency scenarios when a doctor or nurse needs to access limited patient information to provide the necessary care.

Algorithm 7 A Break-the-Glass policy where we hash the identifier of the CareTeam in the Observations

```
Policy break_simple[$shell$]
1: $observation$ := $input.observations$[_]
2: $clean$ := object.remove($observation$, ['data', 'careteam'])
3: $no\_performer$ := object.remove($observation.data$, ['performer'])
4: $doctors$ := [$performer$| ▶ We create a performer array without the CareTeams
5:     $perfomer$ := $observation.data.performer$[_]
6:     $performer.type$ == 'http://hl7.org/fhir/practitioner.html']
7: $hash\_careteams$ := [$perf$| ▶ We create an array from the hashed CareTeams
8:     $performer$ := $observation.data.performer$[_]
9:     not $performer['type']$ ▶ CareTeams do not have the optional type attribute
10:    $perf$ := {'identifier': {'value': crypto.md5($performer.identifier.value$)}}]
11: $new\_performers$ := array.concat($doctors$, $hash\_careteams$)
12: $new\_data$ := object.union($clean$, {'data': $new\_data$, 'careteam': {'identifier':
    {'value': crypto.md5($observation.careteam.identifier.value$)}}})
```

Algorithm 8 A Break-the-Glass policy where we hash every identifier for Patients, Practitioners and CareTeams

```
Policy break_complex[$shell$]
1: $observation$ := $input.observations$[_]
2: $clean$ := object.remove($observation$, ['data', 'careteam', 'practitioner', 'patient'])
3: $removed\_body$ := object.remove($observation.data$, ['performer', 'subject'])
4: $new\_body$ := object.union($removed\_body$,
5:     {'subject': {'display': crypto.md5($observation.data.subject.display$), identifier: {'value':
       crypto.md5($observation.data.subject.identifier$) }},
6:     'performer': [$perf$|
7:         $performer$ := $observation.data.performer$[_]
8:         $perf$ := {'identifier': {'value': crypto.md5($performer.identifier.value$)}}]})
           ▶ We also tried whether a nested instruction set would increase or stabilize
           resource consumption
9: $shell$ := object.union($clean$,
10:    {'data': $new\_body$, 'patient': crypto.md5($observation.patient$),
11:    'practitioner': crypto.md5($observation.practitioner$),
12:    'careteam: {'identifier': {'value':
       crypto.md5($observation.careteam.identifier.value$)}}}})
```

In these scenarios, the system must remove or encrypt everything else that goes beyond the most necessary attributes, and with two policies we tested how resource consumption varies when we wish to encrypt a single attribute that is deeply embedded in the document and requires filtering in *break_simple* and in *break_complex* when we wish to encrypt every identifier in the document that could later be used to identify users in the system.

4 Results

4.1 PEP Performance between Categories

After evaluating the average performance of the various categories, we observed several interesting trends, compared to what we originally expected. The most notable of these is the relatively faster evaluation time of the Contextual Evaluation policies compared to the Role Evaluation category, as seen in Figure 4.

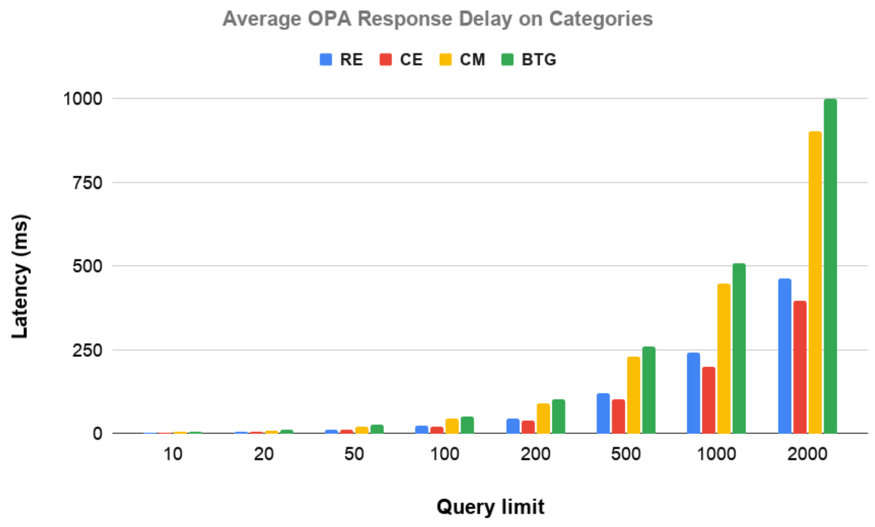


Figure 4: Average Delay on PEP by Categories

While the CE policies are more complex in nature, it seems that if the contextual information sets the result as true or false, the evaluation is significantly quicker than the cases when an internal examination of the input documents is required.

We observed a similar trend with the average CPU load of the categories, shown in Figure 5 with the Contextual Evaluation policies demanding slightly less percentage of the CPU time compared to the Role Evaluation policies, while the Break-the-Glass policies remain the most demanding ones. However, the overall difference between the first two and latter two categories is not as big as on the response delay. Another key observation is that while the size of the input was below 1000 documents (which is already an unrealistically large query size for a real-life application), not even the Break-the-Glass policies required more than 50% of the CPU.

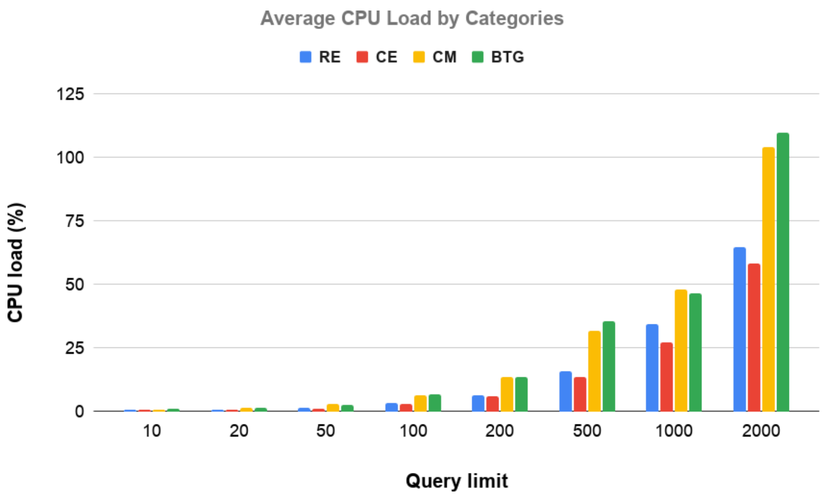


Figure 5: Average CPU Load of OPA by Categories

The memory usage of the categories, shown in Figure 6 on the other hand, while still showing the trends of the previous figures and requiring only manageable amount of memory when evaluation smaller inputs (not even Break-the-Glass policies demanding more than 50-60 MB while the input size is around 50 documents), this demand shows a sudden jump after the input size reaches 1000 documents, with even the Role Evaluation policies requiring around 100-150 MB to evaluate inputs between sizes 1000 and 2000.

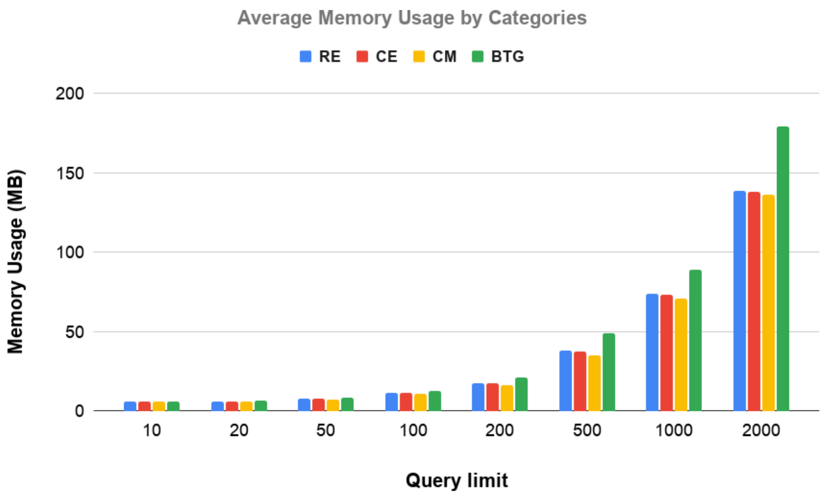


Figure 6: Average Memory Usage of OPA by Categories

The measurements were taken with the PEP solution restarted and reinitialized between each measurement, since, as we have shown in our previous paper, OPA employs a very lazy approach towards garbage collecting, cleaning the memory only when it is required by the system or an especially large evaluation. This fact makes these sizes even more alarming for a real-life scenario, since the size of OPA in the memory can grow significantly during a series of evaluations.

Based on these results, while the CPU load and the response delay seem to be manageable requirements, the memory demand, combined with the experienced lazy garbage collecting process of OPA might requires a custom build or external process that manages and frees the memory after the evaluations are finished to optimize this aspect of the PEP nodes.

4.2 PEP Performance in Categories

We ran each policy with each size at least 30-50 different times to collect the raw data for the statistics shown in the tables below, and these group the policies belonging to the same category. For each policy, we calculated from the collected data sets the mean value of CPU load, memory usage and response delay on the PEP (OPA) node.

Although the question of whether to use the same constant inputs for each evaluation, or use HTTP(S) requests that simulate a real-world application is a complex element for this phase of our research, we decided to use dynamic inputs to get more precise, realistic results.

4.2.1 Role Evaluation Policies

A comparison between Role Evaluation policies is shown in Figure 7. While, as we have assumed, the complexity of *role_complex* induces a higher latency, the total difference between the two policies is not very significant. Even with an input of 2000 records the PEP was able to filter out the restricted ones in half a second.

The effects on CPU and memory, as shown in Figures 8 and 9, are somewhat more demanding – when 2000 documents are sent, 70% of the processor is required to evaluate the policy and about 140 MB of the memory – a clear indication of how costly it is to perform subqueries in isolated structures, such as the list of careteams and their members.

Based on these results, it is evident that the proposed solution is capable of handling more complex Role Evaluation policies without difficulty, but it is advisable to store the teams, groups, institutions in indexed objects rather than in arrays.

4.2.2 Contextual Evaluation Policies

A comparison of the two Context Evaluation policies also produced some interesting results, which are presented in Figures 10, 11 and 12. Our main objective here was to determine what kind of contextual evaluation is more demanding, and on the basis of the data it is clear that array-based evaluations are generally more complex,

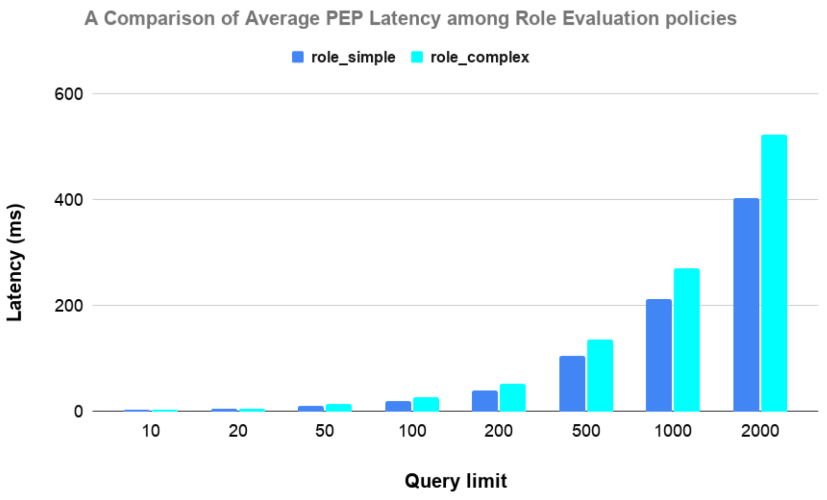


Figure 7: Average PEP Latency in Role Evaluation category

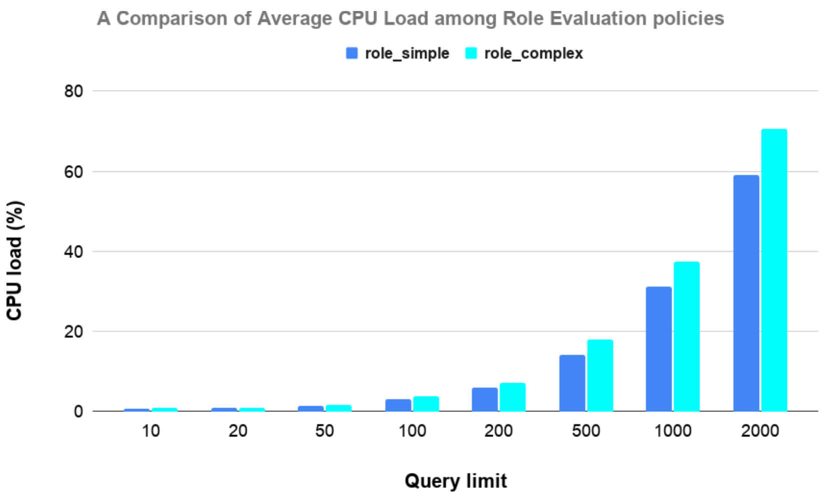


Figure 8: Average CPU load in Role Evaluation category

but in small evaluations they are actually cheaper than collecting and comparing external information such as dates.

This calls into question some notable architectural aspects of the infrastructure, such as whether this context information should be collected and forwarded by the proxy as part of the input data. Seeing that in some cases it is possible on a larger scale that the PEP deployment can handle traffic from end nodes in different time zones, it might be a good idea to omit such internal queries as a general design pattern of the policies.

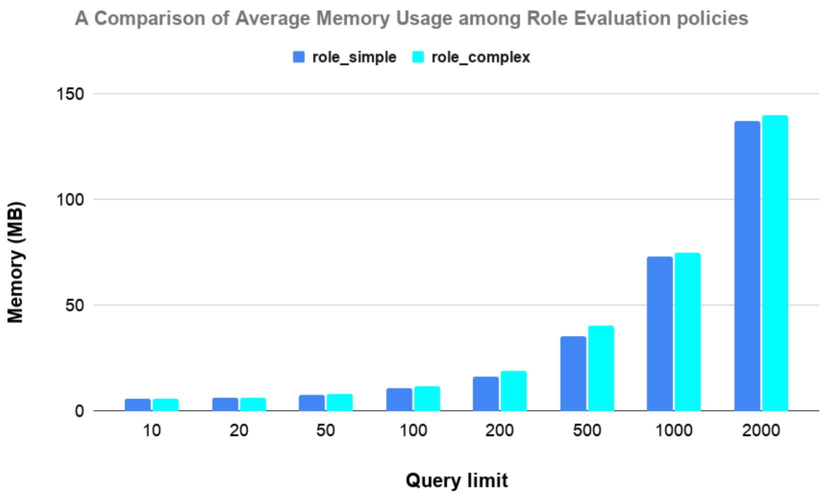


Figure 9: Average Memory usage in Role Evaluation category

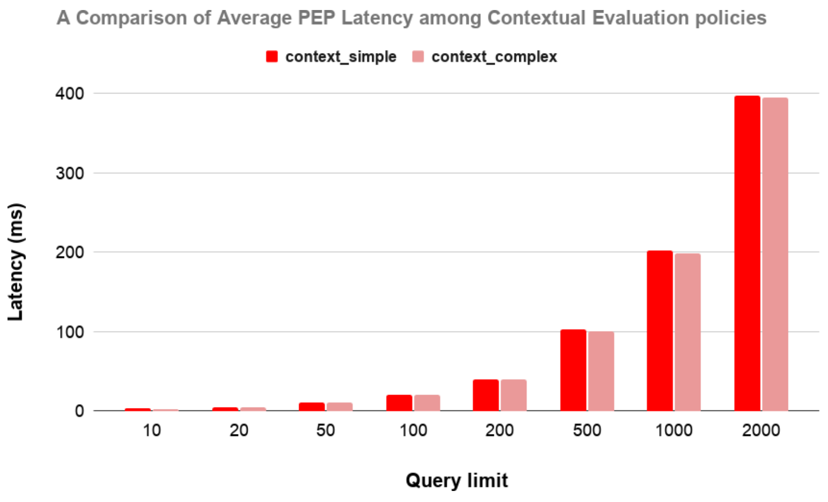


Figure 10: Average PEP Latency in Contextual Evaluation category

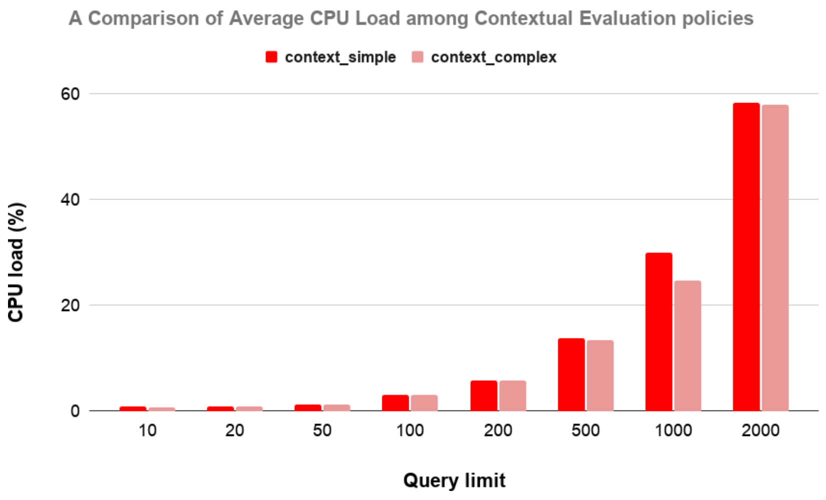


Figure 11: Average CPU load in Contextual Evaluation category

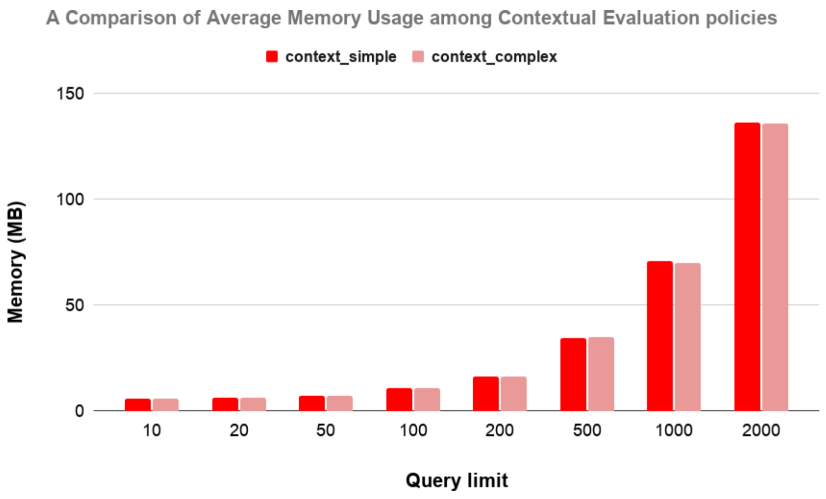


Figure 12: Average Memory usage in Contextual Evaluation category

Moreover, it is interesting to note that after some minor differences, besides inputs with more than 20 documents, the metric values start to converge, since after a certain point in the measurement set a significant portion of the documents is rejected during the evaluation of context1 due to their inactive status, thus the value is set false, before the engine starts evaluating the relational conditionals.

Our interpretation of these results can be summarized as follows: It is clear that policies with contextual evaluation may be as effective as simple Role Evaluation policies, but where possible, contextual information must be provided as part of the input, rather than being queried on the PEP node.

4.2.3 Contextual Modification Policies

The results of the Contextual Modification policies are included in Figures 13, 14 and 15. These results showcased another important aspect of the evaluation engine that could be one of the pillars of the design patterns for defining the policies. It was expected that *modif_complex* would be the more complex of the two.

Instead of this expected result, the measurements clearly indicate that neither is significantly more demanding. In some cases *modif_complex* consumes more CPU, and it has slightly more latency than *modif_simple* due to the demanding array copy and filter mechanisms, while *modif_simple* requires slightly more memory.

Modification policies are much more demanding than simple access evaluations. Nevertheless, they can be implemented effectively if we take into account the increased costs. We also wish to investigate the possible patterns and anti-patterns in order to write more effective policies for this type.

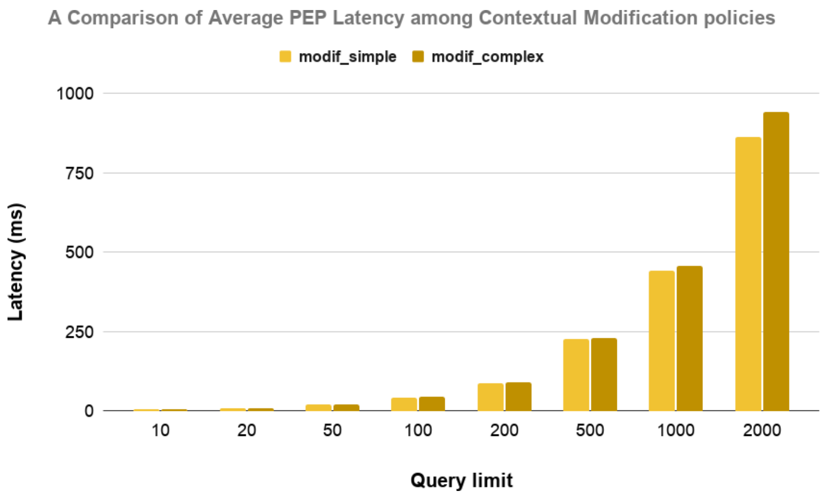


Figure 13: Average PEP Latency in Contextual Modification category

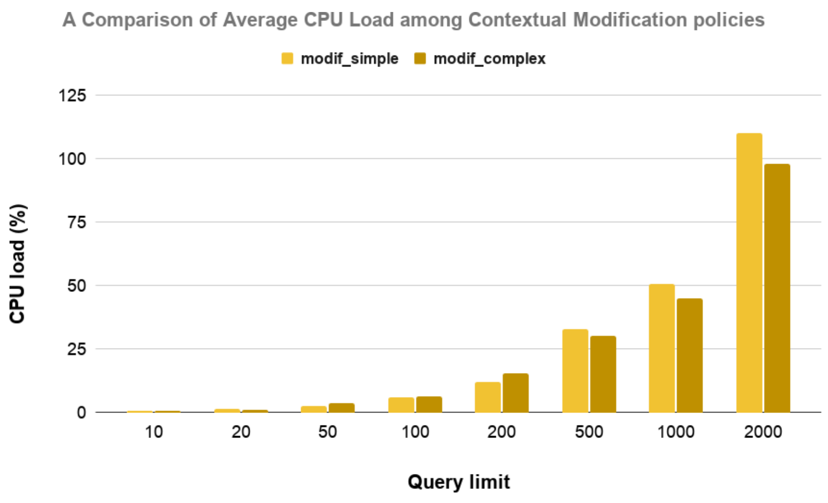


Figure 14: Average CPU load in Contextual Modification category

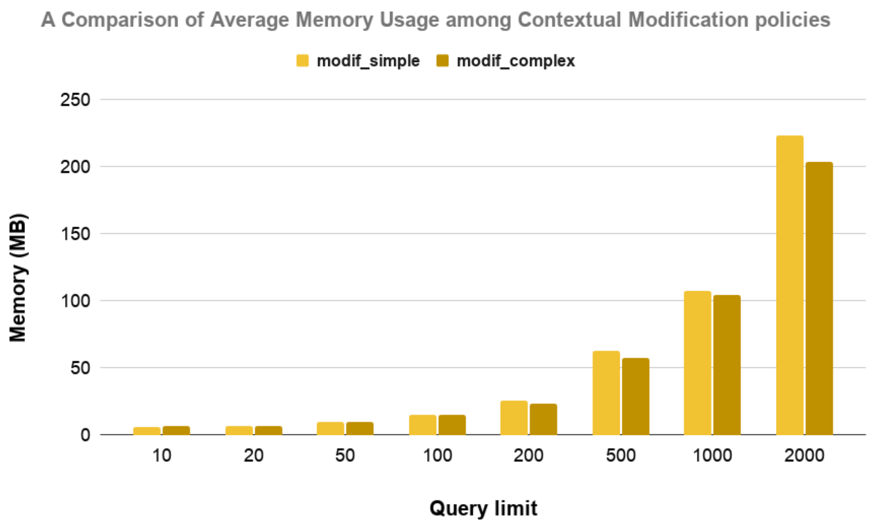


Figure 15: Average Memory usage in Contextual Modification category

4.2.4 Break-the-Glass Policies

Based on the results of our previous evaluations, we assume that Break-the-Glass policies will be the most resource-intensive portion of our evaluation set, and the results (see Figures 16, 17 and 18) were as we thought they would be, but again with some minor differences from our original expectations.

While the CPU load and memory usage of the two policies are almost identical, it actually takes a little longer to evaluate *break_simple* than *break_complex*, although based on the sheer number of operations (especially the number of encryption operations) in *break_complex*, it should have been a much more expensive policy compared to *break_simple*. The answer lies in the nature of operations that the policy executes: It filters an array, then creates a new array to store an encrypted value, and then concatenates two arrays to be embedded in an object. Apparently, these array operations, with the emphasis on the concatenation operation, which is unique in our evaluation set for this policy, is just as resource-demanding as its pair.

Just as we expected, the Break-the-Glass policies are the most demanding ones that can be evaluated on the PEP node, but even with the increased cost they can achieve the expected results. While the essence of these policies is to transform and encrypt the data, it is important to avoid array operations as much as possible, as they only further increase the cost when potentially cheaper workarounds might be available.

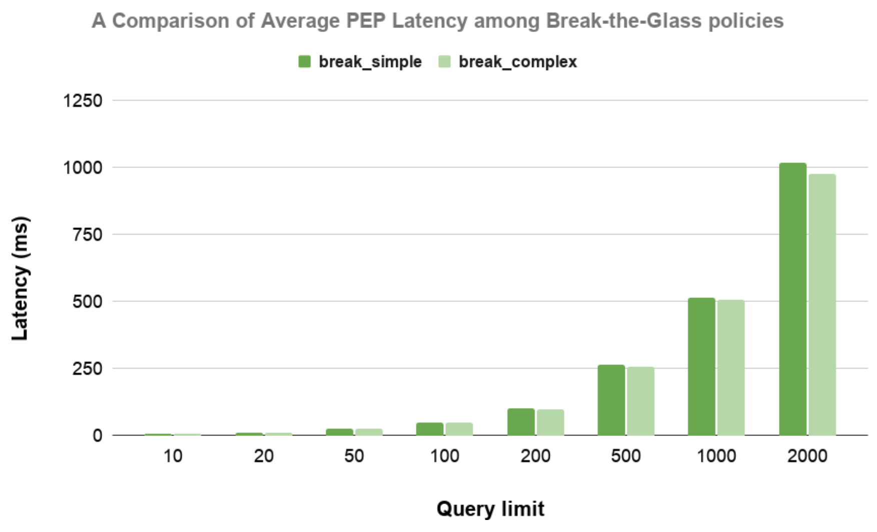


Figure 16: Average PEP Latency in Break-the-Glass category

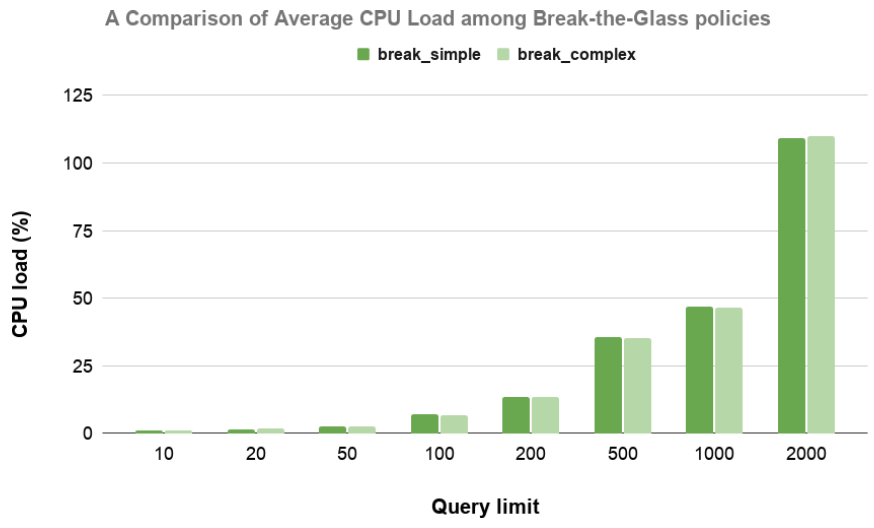


Figure 17: Average CPU load in Break-the-Glass category

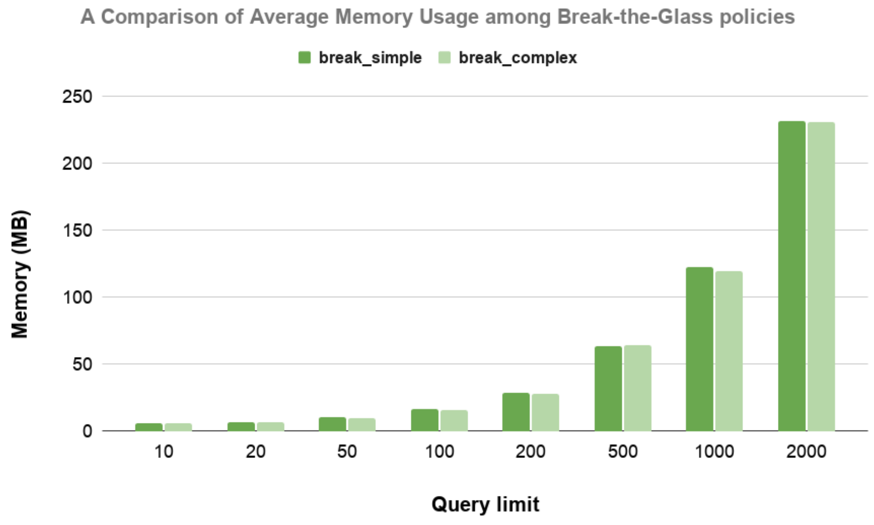


Figure 18: Average Memory usage in Break-the-Glass category

4.3 PEP Performance in Infrastructure

It is interesting to see how the latency of the PEP node affects the latency of the entire infrastructure. From each policy pair we took the one with the greater response delay and compared it with the system latency in Figures 19, 20, 21, and 22.

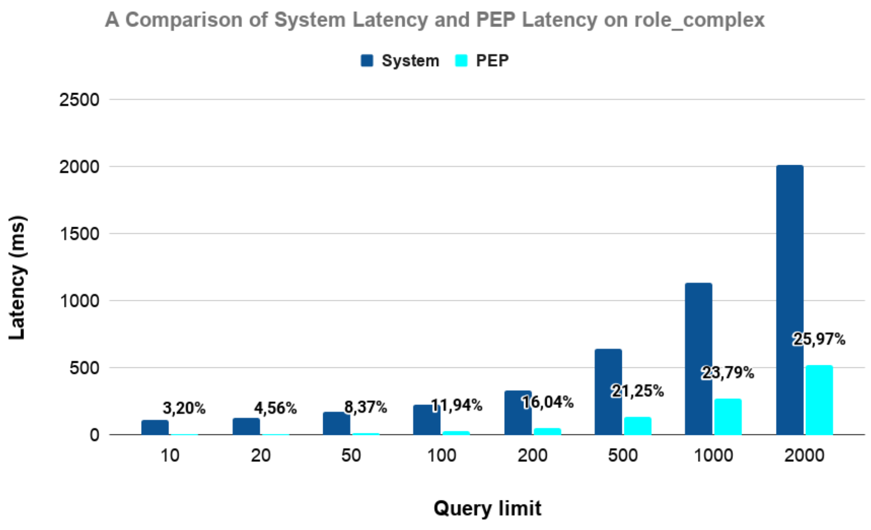


Figure 19: System Latency and PEP Latency on role_complex

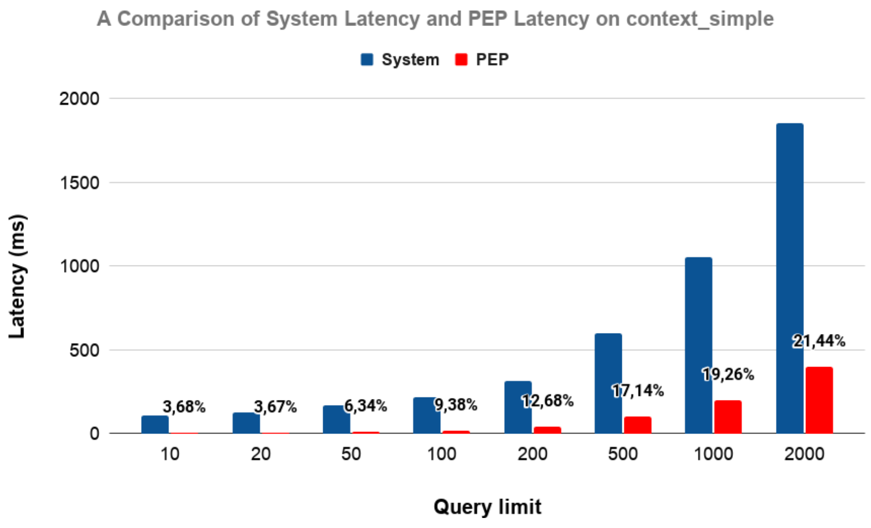


Figure 20: System Latency and PEP Latency on context1

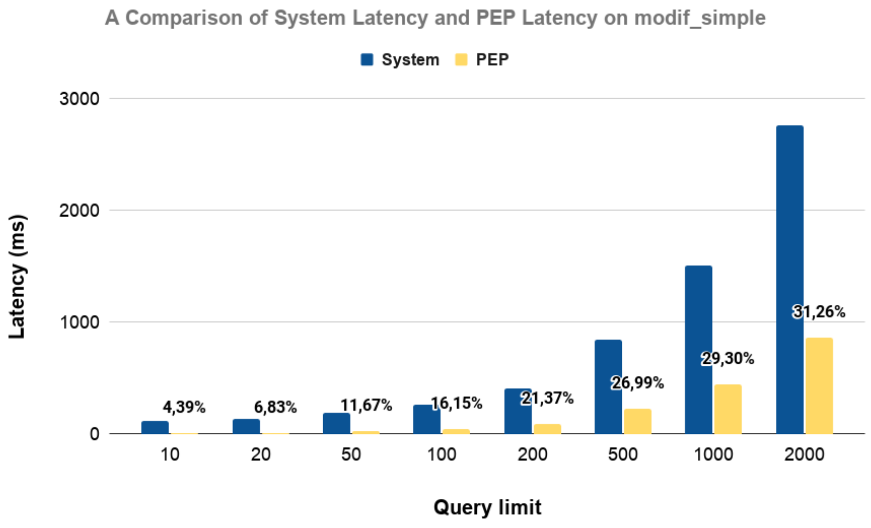


Figure 21: System Latency and PEP Latency on modif_simple

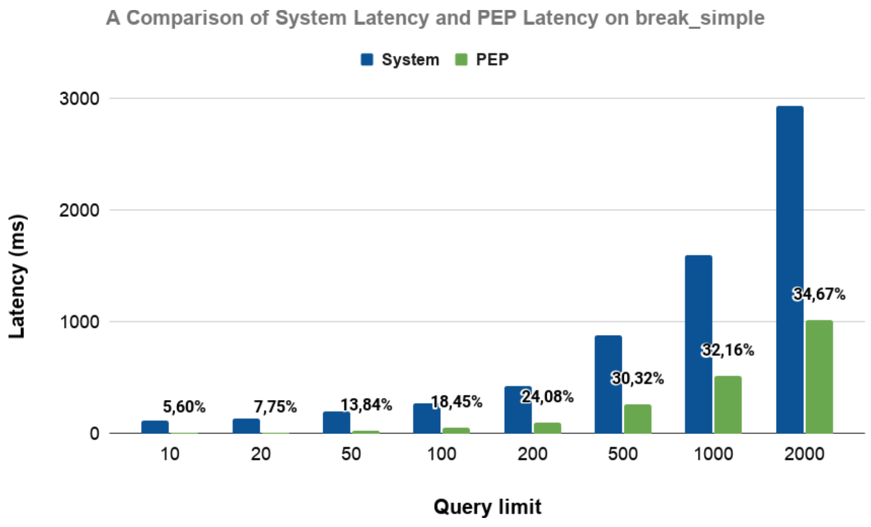


Figure 22: System Latency and PEP Latency on *break_simple*

Based on these results, we may conclude that when the data set is increased, the increase in system-wide latency, PEP latency and the relative latency of the two components are all nonlinear.

The difference between the complexity of the various policy categories, on the other hand, is not always as clear as we initially assumed. The results of *role_complex* and *context_simple*, for example, are almost identical, and the care team identifying *role_complex* even turns out to be somewhat more demanding than context1, which has to iterate and filter the contents of an embedded array, and with an input of size 2000 on *context_simple*, the PEP only provides the 21.4370% of the full system latency and 25.9672% on *role_complex*.

However, most of our expectations were confirmed by the results we obtained. Although it is clear that the identification of good practices, patterns and anti-patterns is necessary in the next phase of our research to further optimize the use of the PEP, the relative complexity and cost of the different categories were as expected. There is an overall latency and efficiency of the prototype infrastructure – a barely noticeable increase when we consider that the majority of healthcare applications, including our client application in its unmodified state. This requests 100 or 200 documents in a single operation, and we found the PEP (and OPA as its implementation) to be a very effective component of our security solution.

5 Conclusions

On the basis of our results, we conclude that the PEP (OPA) is indeed a suitable choice for our architecture and that the concept can be further developed. In the previous sections, we presented our proposed infrastructure and methodology, the categorization of security policies required in the healthcare sector, and a set of policies for different evaluations. When interpreting the results, we demonstrated the effectiveness of our concept, with only a relatively small increase in the overall latency in exchange for an effective healthcare security solution that is independent of the exact back- and front-end applications. In addition, we have also identified some good or definitely avoidable practices in the OPA policy definition that require further research to make it feasible in practice. To summarize these results based on the requirements described in the Introduction for an appropriate policy enforcement element in the domain of telemedicine:

- We showed that our proposed PEP can function and evaluate access in a telemedicine infrastructure with a minimal impact on the overall latency of the system, without any component explicitly being aware of its presence. Even with the largest input size, the evaluation delay caused by its inclusion was at most 35% of the total response delay;

- We showed that it can evaluate and transform the input with manageable CPU load and memory usage even for inputs that are unrealistically large in the healthcare workflow;

- Since our PEP configuration only depends on the input structure and our current selection is also available in Go and WebAssembly languages, it can be effectively used at any point in the infrastructure. Because of its internal database it can also work when the upper levels of the infrastructure are unavailable.

- We shoved that every category of our policy terminology can be effectively implemented in the script language of the OPA engine.

In the future we plan to continue our research in several directions. When interpreting the results, we found several possible patterns and anti-patterns for writing PEP policies. We intend to investigate these further in simulated an real-world scenarios as well, identify the possible bottlenecks and bad practices for deployment and provide other developers and researchers with a good guideline for implementing healthcare policies.

Since our infrastructure prototype turned out to be successful, the next step will be to further evaluate the capabilities of our proposed PEP integration in terms of portability, place it at different points in the infrastructure and record their behavior and efficiency.

If we succeed with these future steps, we hope to be a step closer to solving the problem of security and access control in telemedicine.

References

[1] Index – FHIR v4.0.1. `https://www.hl7.org/fhir/`. (Accessed on 09/16/2020).

[2] MongoDB official documentation. `https://docs.mongodb.com/manual/`. (Accessed on 09/16/2020).

[3] Open Policy Agent – policy performance. `https://www.openpolicyagent.org/docs/latest/policy-performance/`. (Accessed on 09/16/2020).

[4] Open Policy Agent official site. `https://www.openpolicyagent.org/`. (Accessed on 09/16/2020).

[5] Alhaqbani, Bandar and Fidge, Colin. Access control requirements for processing electronic health records. In *International Conference on Business Process Management*, pages 371–382. Springer, 2007. DOI: `10.1007/978-3-540-78238-4_38`.

[6] Alnefaie, Seham, Cherif, Asma, and Alshehri, Suhair. Towards a distributed access control model for IoT in healthcare. In *2019 2nd International Conference on Computer Applications & Information Security (ICCAIS)*, pages 1–6. IEEE, 2019. DOI: `10.1109/CAIS.2019.8769462`.

[7] Altamimi, Ahmad. SecFHIR: A security specification model for fast healthcare interoperability resources. *International Journal of Advanced Computer Science and Applications*, 7(6), 2016. DOI: `10.14569/IJACSA.2016.070645`.

[8] Andrew, Marcus. Security in FHIR at DevDays Redmond 2019. `https://tinyurl.com/ryk9zlu`, 2019. (Accessed on 07/20/2020).

[9] Attia, Hasiba Ben, Kahloul, Laid, and Benharzallah, Saber. A new hybrid access control model for security policies in multimodal applications environments. *Journal of Universal Computer Science*, 24(4):392–416, 2018. DOI: `10.3217/jucs-024-04-0392`.

[10] Carter, Gracie, Chevellereau, Ben, Shahriar, Hossain, and Sneha, Sweta. OpenPharma Blockchain on FHIR: An interoperable solution for read-only health records exchange through blockchain and biometrics. *Blockchain in Healthcare Today*, 2020. DOI: `10.30953/bhty.v3.120`.

[11] Chaballout, Basil Harris, Shaw, Ryan Jeffry, and Reuter-Rice, Karin. The SMART healthcare solution. *Advances in Precision Medicine*, 2(1), 2017. DOI: `10.18063/apm.v2i1.213`.

[12] Conley, Ed and Pocs, Matthias. GDPR compliance challenges for interoperable health information exchanges (HIEs) and trustworthy research environments (TREs). *European Journal of Biomedical Informatics*, 14(3), 2018. DOI: `10.24105/ejbi.2018.14.3.7`.

[13] Fernández-Alemán, José Luis, Señor, Inmaculada Carrión, Lozoya, Pedro Ángel Oliver, and Toval, Ambrosio. Security and privacy in electronic health records: A systematic literature review. *Journal of Biomedical Informatics*, 46(3):541–562, 2013. DOI: `10.1016/j.jbi.2012.12.003`.

[14] Ferraiolo, David, Chandramouli, Ramaswamy, Kuhn, Rick, and Hu, Vincent. Extensible access control markup language (XACML) and next generation access control (NGAC). In *Proceedings of the 2016 ACM International Workshop on Attribute Based Access Control*, pages 13–24, 2016. DOI: `10.1145/2875491.2875496`.

[15] Ferraiolo, David, Cugini, Janet, and Kuhn, D Richard. Role-Based Access Control (RBAC): Features and motivations. In *Proceedings of 11th Annual Computer security Application Conference*, pages 241–48, 1995.

[16] Finance, Beatrice, Medjdoub, Saida, and Pucheral, Philippe. Privacy of medical records: From law principles to practice. In *18th IEEE Symposium on Computer-Based Medical Systems (CBMS'05)*, pages 220–225. IEEE, 2005. DOI: `10.1109/CBMS.2005.89`.

[17] Gajanayake, Randike, Iannella, Renato, and Sahama, Tony R. Privacy oriented access control for electronic health records. In *Data Usage Management on the Web Workshop at the Worldwide Web Conference*. ACM, 2012.

[18] Islam, SM Riazul, Kwak, Daehan, Kabir, MD Humaun, Hossain, Mahmud, and Kwak, Kyung-Sup. The Internet of Things for health care: A comprehensive survey. *IEEE access*, 3:678–708, 2015. DOI: `10.1109/ACCESS.2015.2437951`.

[19] Johnson, Alistair EW, Pollard, Tom J, Shen, Lu, Li-Wei, H Lehman, Feng, Mengling, Ghassemi, Mohammad, Moody, Benjamin, Szolovits, Peter, Celi, Leo Anthony, and Mark, Roger G. MIMIC-III, a freely accessible critical care database. *Scientific Data*, 3(1):1–9, 2016. DOI: `10.1038/sdata.2016.35`.

[20] Joshi, Maithilee, Joshi, Karuna, and Finin, Tim. Attribute based encryption for secure access to cloud based EHR systems. In *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, pages 932–935. IEEE, 2018. DOI: `10.1109/CLOUD.2018.00139`.

[21] Mukherjee, Subhojeet, Ray, Indrakshi, Ray, Indrajit, Shirazi, Hossein, Ong, Toan, and Kahn, Michael G. Attribute based access control for healthcare resources. In *Proceedings of the 2nd ACM Workshop on Attribute-Based Access Control*, pages 29–40, 2017. DOI: `10.1145/3041048.3041055`.

[22] Pal, Shantanu, Hitchens, Michael, Varadharajan, Vijay, and Rabehaja, Tahiry. Fine-grained access control for smart healthcare systems in the Internet of Things. *EAI Endorsed Transactions on Industrial Networks and Intelligent Systems*, 4(13), 2018. DOI: `10.4108/eai.20-3-2018.154370`.

[23] Peleg, Mor, Beimel, Dizza, Dori, Dov, and Denekamp, Yaron. Situation-based access control: Privacy management via modeling of patient data access scenarios. *Journal of Biomedical Informatics*, 41(6):1028–1040, 2008. DOI: `10.1016/j.jbi.2008.03.014`.

[24] Premarathne, Uthpala, Abuadbba, Alsharif, Alabdulatif, Abdulatif, Khalil, Ibrahim, Tari, Zahir, Zomaya, Albert, and Buyya, Rajkumar. Hybrid cryptographic access control for cloud-based EHR systems. *IEEE Cloud Computing*, 3(4):58–64, 2016. DOI: `10.1109/MCC.2016.76`.

[25] Pussewalage, Harsha S Gardiyawasam and Oleshchuk, Vladimir A. An attribute based access control scheme for secure sharing of electronic health records. In *2016 IEEE 18th International Conference on e-Health Networking, Applications and Services (Healthcom)*, pages 1–6. IEEE, 2016. DOI: `10.1109/HealthCom.2016.7749516`.

[26] Ray, Indrakshi, Alangot, Bithin, Nair, Shilpa, and Achuthan, Krishnashree. Using attribute-based access control for remote healthcare monitoring. In *2017 Fourth International Conference on Software Defined Systems (SDS)*, pages 137–142. IEEE, 2017. DOI: `10.1109/SDS.2017.7939154`.

[27] Ray, Indrakshi, Ong, Toan C, Ray, Indrajit, and Kahn, Michael G. Applying attribute based access control for privacy preserving health data disclosure. In *2016 IEEE-EMBS International Conference on Biomedical and Health Informatics (BHI)*, pages 1–4. IEEE, 2016. DOI: `10.1109/BHI.2016.7455820`.

[28] Rezaeibagha, Fatemeh and Mu, Yi. Distributed clinical data sharing via dynamic access-control policy transformation. *International Journal of Medical Informatics*, 89:25–31, 2016. DOI: `10.1016/j.ijmedinf.2016.02.002`.

[29] Rosa, Marco, Faria, Cristiano, Barbosa, Ana Madalena, Caravau, Hilma, Rosa, Ana Filipa, and Rocha, Nelson Pacheco. A fast healthcare interoperability resources (FHIR) implementation integrating complex security mechanisms. *Procedia Computer Science*, 164:524–531, 2019. DOI: `10.1016/j.procs.2019.12.215`.

[30] Rousskov, Alex and Soloviev, Valery. A performance study of the Squid proxy on HTTP/1.0. *World Wide Web*, 2(1-2):47–67, 1999. DOI: `10.1023/A:1019240520661`.

[31] Sánchez, Yaira K Rivera, Demurjian, Steven A, and Baihan, Mohammed S. Achieving RBAC on RESTful APIs for mobile apps using FHIR. In *2017 5th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)*, pages 139–144. IEEE, 2017. DOI: `10.1109/MobileCloud.2017.22`.

[32] Sun, Jianfei, Xiong, Hu, Zhang, Hao, and Peng, Li. Mobile access and flexible search over encrypted cloud data in heterogeneous systems. *Information Sciences*, 507:1–15, 2020. DOI: `10.1016/j.ins.2019.08.026`.

[33] Szabó, Z. and Bilicki, V. Felhőben tárolt egészségügyi adatok védelme ABAC modellel. In *Orvosi informatika 2018 — A XXXI. Neumann Kollokvium konferencia-kiadványa*, pages 134–139. Neumann János Számítógép-tudományi Társaság (NJSZT), 2018.

[34] Szabó, Z. and Bilicki, V. Achieving RBAC on RESTful APIs for mobile apps using FHIR. In *CSCS — The Twelfth Conference of PhD Students in Computer Science*. Institute of Informatics, University of Szeged, 2020. `https://www.inf.u-szeged.hu/~cscs/proceedings.php`.

[35] Tasali, Qais, Chowdhury, Chandan, and Vasserman, Eugene Y. A flexible authorization architecture for systems of interoperable medical devices. In *Proceedings of the 22nd ACM on Symposium on Access Control Models and Technologies*, pages 9–20, 2017. DOI: `10.1145/3078861.3078862`.

[36] Yuan, Eric and Tong, Jin. Attributed based access control (ABAC) for web services. In *IEEE International Conference on Web Services (ICWS'05)*. IEEE, 2005. DOI: `10.1109/ICWS.2005.25`.

Acta Cybernetica **25** (2021) 517–535.

Protocol State Machine Reverse Engineering with a Teaching–Learning Approach*

Gábor Székely[abc], Gergő Ládi[ade], Tamás Holczer[af], and Levente Buttyán[ag]

Abstract

In this work, we propose a novel solution to the problem of inferring the state machine of an unknown protocol. We extend and improve prior results on inferring Mealy machines, and present a new algorithm that accesses and interacts with a networked system that runs the unknown protocol in order to infer the Mealy machine representing the protocol's state machine. To demonstrate the viability of our approach, we provide an implementation and illustrate the operation of our algorithm on a simple example protocol, as well as on two real-world protocols, Modbus and MQTT.

Keywords: automated protocol reverse engineering, state machines, Mealy machines

1 Introduction

In today's world, IT systems are seldom standalone and monolithic. Each system is comprised of up to several tens of parts or modules that somehow communicate with each other, usually via a network. The rules of communication, i.e. the formats of the messages and the possible valid sequences of messages are established by protocols that are defined in specifications. Many systems use proprietary or closed protocols whose specifications are not made publicly available. Examples typically include industrial control systems (ICS) and in-vehicle embedded networks.

*The research presented in this paper has been supported by the European Union, co-financed by the European Social Fund (EFOP-3.6.2-16-2017-00013), the Hungarian National Research, Development and Innovation Fund (NKFIH, project no. 2017-1.3.1-VKE-2017-00029), and the IAEA (CRP-J02008, contract no. 20629).

[a]Laboratory of Cryptography and System Security, Department of Networked Systems and Services, Budapest University of Technology and Economics, Hungary

[b]Ukatemi Technologies

[c]E-mail: `gabor.szekely@ukatemi.com`, ORCID: `0000-0001-6148-3948`

[d]BME Balatonfüred Student Research Group, Hungary

[e]E-mail: `gergo.ladi@crysys.hu`, ORCID: `0000-0002-0318-2175`

[f]E-mail: `holczer@crysys.hu`, ORCID: `0000-0003-0953-5397`

[g]E-mail: `buttyan@crysys.hu`, ORCID: `0000-0003-4233-2559`

DOI: `10.14232/actacyb.288213`

However, it would often be largely beneficial to have at least an approximate idea of how these closed protocols work. For instance, with this knowledge, it would be possible to build network anomaly detection tools that detect potential cyberattacks against industrial systems, design more specialized honeypots, detect malicious or malfunctioning components in in-vehicle networks, or build new components that can be more easily integrated into existing systems.

One may attempt to reconstruct the specification of an unknown protocol by applying various protocol reverse engineering methods. The goal of such methods is two-fold: first, the type and format of the messages used by the protocol need to be understood, second, this information may be used to recover the state machine of the protocol. The process of reverse engineering a protocol is often a tedious task; therefore, some degree of automation is required to make it practical.

In a previous paper [14], we focused on the problem of determining the message types and message formats of undocumented binary protocols. We developed a tool that can process captured network traffic containing messages of a protocol, then retrieve the identified message types as well as the semantics of the message fields for the different message types. In this paper, we address the problem of inferring the state machine of a previously unknown protocol.

Our method is based on prior work on inferring Mealy machines, in particular, on the work of Shahbaz and Groz [20]. We take their Mealy machine inference algorithm and extend it with elements that make it possible to use their conceptual results in practice so as to reverse engineer the state machine of real-world protocols in an automated fashion. This method requires access to and interaction with a system that runs the unknown protocol. In addition, we assume that the message types and formats have already been reverse engineered, e.g., by using our previously published method. We process and use previously recorded network traces as well as data obtained from this system, efficiently and intelligently choosing input for the above-mentioned Mealy machine inference algorithm.

2 Related work

The principles of protocol reverse engineering date back to the 1950s, where reverse engineering was typically used for fault analysis, and it involved analyzing and understanding electrical circuits that implemented a finite state machine representing a protocol [15]. With computers and computerized accessories becoming more and more widespread around the turn of the century, the number of network applications, thus the number of network protocols increased. Some of these were proprietary with no available documentation, meaning that they had to be reverse engineered in order to develop compatible applications.

The first well-known project that aimed at restoring the specifications of such a protocol was the Samba Project (2003) [1] that has taken 12 years to finish. The exact methods they used are not detailed, but it is known that they sniffed network traffic and employed random probing to identify message types and semantics. The Samba Project was soon followed by M. A. Beddoe's Protocol Informatics Project

[5] in 2004, which used bioinformatical algorithms on network traces to infer the message types of the text-based protocol HTTP. RolePlayer (2006) [12], Discoverer (2007) [11], Biprominer (2011) [22], ReverX (2011) [3], ProDecoder (2012) [21], and AutoReEngine (2013) [17] soon followed, all of which relied solely on network traffic.

The early works typically focused on message type and message format inference. They did not put much emphasis on field semantics inference, nor did they attempt to recover the protocol state machine. Those that tried to infer field semantics did not achieve significant results – Discoverer admits to achieving between 30-40% accuracy [11], and not even Netzob exceeds 50% [7]. FieldHunter (2015) [6] was the first to achieve over 80% accuracy on semantics. In a previous paper [14], we presented GrAMeFFSI (2020) that may achieve over 90% accuracy on binary protocols if high-quality network captures are available.

While most algorithms aimed at reversing both text-based and binary protocols, some specialized in one or the other, usually achieving better performance metrics compared to the more general solutions of their time. Biprominer, as its name suggests, targeted binary protocols, as did GrAMeFFSI, while ReverX targeted text-based protocols. The methods employed vary – RolePlayer and Discoverer rely on sequence alignment, Biprominer and AutoReEngine leverage data mining approaches, ProDecoder makes use of natural language processing algorithms, while GrAMeFFSI employs graph analysis.

An alternative to network traffic analysis based protocol inference is binary analysis based inference. Such methods often rely on dynamic taint analysis, marking sections of code in the memory of the binary being executed that are hit when processing or responding to a given message, then making assumptions about the message formats and semantics based on what and how was marked. It has been proven [19] that binary analysis based approaches can achieve better results; however, purely traffic analysis based approaches are also important as binaries may not always be at our disposal, and legal agreements may prevent us from analyzing or reverse engineering these.

In order to reverse engineer the protocol state machine, at least a partial understanding, a classification of the message types is needed. State machine inference methods must either produce the message classes themselves, or rely on existing message format inference methods, perhaps in a slightly modified manner. So far, there have been fewer attempts to reconstruct protocol state machines than to determine message types and semantics [13]. The majority of the network trace based solutions are passive, meaning that only recorded traffic is used as input, and no live systems (running original protocol implementations) are contacted.

ScriptGen (2005) [16] was the first notable method for state machine inference. It uses the Needleman-Wunsch sequence alignment algorithm (similarly to Beddoe's previously mentioned project) along with micro- and macroclustering to build a protocol state machine from captured network traffic. The state machine is then used to formulate responses for a honeypot. It was later followed by Cho et al.'s work on reverse engineering the protocol of the MegaD botnet (2010) [8] that leveraged and optimized Angluin's L^* algorithm [2] to infer a Mealy machine based on network traces and live queries. They reached between 96% and 99% accuracy on various protocols. Another important result was Veritas (2011) [23] that employs statistical analysis on captured network traffic to build probabilistic protocol state machines that are claimed to be 92% accurate on average. There also exist approaches that rely on program execution instead of network traces, the most significant ones being Prospex (2009) [10] and MACE (2011) [9].

3 The ${L_M}^+$ algorithm and its application for inferring protocol state machines

We use Mealy machines to represent the state machine of a protocol as they can be used to model the behaviour of protocols using requests and responses (which are quite typical in practice) in a simpler way than finite state machines or Moore machines. Mealy machines differ from simple finite state machines in that for every state transition that is triggered by an input, an output is defined. The set I of possible inputs is called the input alphabet and the set O of possible outputs is called the output alphabet.

Angluin described an algorithm in [2] that can be used to infer minimal finite state machines, and this algorithm can be adapted for inferring Mealy machines as well. In this work, we adopt the techniques of Shahbaz and Groz described in [20], and from this point forward, we refer to the Mealy machine inferring algorithm described in [20] as ${L_M}^+$.

Since we use the ${L_M}^+$ algorithm as a black box, a high-level overview of its operation is sufficient for our purposes here. The ${L_M}^+$ algorithm is executed by a *learner*, and it requires a *teacher*. The teacher knows the Mealy machine to be inferred, and the task of the learner is to infer that machine. The teacher can answer two types of queries for the learner: first, for a certain sequence of input characters, the teacher returns the output of the machine to be inferred (input query); second, the teacher can determine whether a certain Mealy machine conjectured by the learner is the same as the one to be inferred (equivalence query). If the conjectured machine differs from the real one, then the teacher returns a counterexample: a sequence of input characters for which the real and the conjectured machines produce a different output.

A Mealy machine can be used to model the state machine of a client-server protocol in a fairly straightforward manner: the input alphabet of the machine contains the possible messages that the client may send to the server (i.e., the requests) and the output alphabet contains the possible messages that the server may send to the client (i.e., responses, acknowledgements, errors, *etc.*).

Clearly, including all possible individual messages in the input and output alphabets can easily lead to problems: a huge resulting Mealy machine and a very long running time of the ${L_M}^+$ algorithm. For instance, if a message contains a 4-byte timestamp, then the alphabet would contain at least 2^{32} elements to represent all possible messages containing different timestamp values. To bring the size of the alphabets to a manageable range, we represent message types by the elements of the input and output alphabets instead of individual messages. A message type models a group of messages that have the same format but may differ in the specific values in the fields of the given message type.

In order to work with message types, we use two helper functions: a message classifier and a message generator. The message classifier function takes a particular message as input and returns its message type. The message generator function takes a message type as input and generates a valid message that has the specified message type. While the message classifier function should be deterministic, the message generator can be non-deterministic: the values of the message fields can be randomly generated as long as the message remains well-formed (i.e., consistent with its type). An additional function, a message updater is also useful, which takes as input the set M of all previously sent messages and a particular message m of this set, and returns a new message m' of the same type as m, such that the values of certain fields in m' are the mutations of the corresponding values in m, and M does not include m' yet. The updater function takes the semantics of certain fields into account: for instance, constant fields and identifiers are not mutated, a counter is mutated by incrementing it, *etc.* Such an awareness of semantics improves the efficiency and accuracy of our algorithm.

Recall that we aim at reverse engineering the protocol state machine of a system under test (SUT). To achieve this goal, we use the ${L_M}^+$ algorithm as the learner that infers the unknown Mealy machine representing the protocol, and we need to provide the teacher that answers the queries of the learner. We construct the teacher by using the helper functions defined above, and by sending messages to and observing the responses of the actual implementation of the protocol provided by the SUT.

This works in the following way: we start by using the message generator helper function to produce messages for every message type. We use these pre-generated messages to avoid a deterministic protocol appearing to be non-deterministic due to freshly generated random values used in the same message at different stages of our algorithm. Then, we run the ${L_M}^+$ learner and we respond to its queries. Input queries are answered by first resetting the SUT, then sending the pre-generated messages (after running the message updater helper function on them) corresponding to the learner's input query to the SUT, and finally running the message classifier helper function on the SUT's responses to get the message types that the learner can understand. Equivalence queries are answered by generating random input queries, running them against both the SUT and the conjectured machine, and comparing their outputs. The number of queries needed to decide about equivalence with a given confidence level has been studied in [2], and we follow those guidelines. Once the ${L_M}^+$ learner conjectures a Mealy machine that is deemed correct, our algorithm terminates with that machine as the output.

4 Extending the $L_M{}^+$ algorithm

The above-described version of our algorithm may return an incomplete protocol state machine because only a single message of each type is generated, which always triggers the same type of response, while it may be possible that other variants of the same message would result in a different response. Consider, for example, a request for reading the content of some memory address; the response can be the data found at the specified address or an error if the address was invalid. The extended version of our algorithm attempts to find these differing behaviours by finding messages of the same type that trigger different responses from the SUT. This further divides the set of messages that belong to a particular message type. We will call these new sets message subtypes. Our goal is to find as many subtypes as possible, ideally all of them. This is done by generating multiple messages of each type for the input alphabet I, then running the simple version of our algorithm with them. The resulting Mealy machine is analyzed, and messages that have the same type but do not produce different behaviour are removed (we call this step *deduplication*). More precisely, we say that two messages do not produce different behaviour, if, for all states of the Mealy machine, the two inputs result in the same output and the same new state. This means that deduplication only leaves a single message instance from every message subtype. After deduplication, new messages are generated and added to the set of possible inputs, and the simple algorithm is executed again. This is repeated again and again to reach more and more complete representations of the protocol as more subtypes are discovered (see Algorithm 1). A straightforward criterion for stopping could be requiring a number of runs where no new subtypes are found.

```
Initialization: $I := \emptyset$
Do
    $I := I \cup generateNewMessages()$;
    $M_{out} := L_M^+(I)$;
    $I, M_{out} := deduplicate(I, M_{out})$;
While stopping criterion is not met;
Result: $M_{out}$
```

Algorithm 1: Pseudocode of the extended algorithm

This method keeps the advantage of reducing the size of the input alphabet while still allowing us to discover inputs that are not considered different by message type categorization but reveal different behaviour of the protocol. In addition, not having an absolutely perfect message format becomes less of a problem, since message types that are not correctly separated by the used message formats will be automatically separated into message subtypes. However, it has two possible major pitfalls. The first problem arises if specific message subtypes only reveal their different properties if they are used in combination. As an example, let us look at a message type that changes some kind of operating mode. Let there be three modes: A, B and C, where there is a subtype for changing into each of these modes (a, b and c respectively). All other message instances that belong to the mode changing message type do not have any effect. Starting out, the protocol is always in mode A, and each mode can only be changed into from a neighbouring mode (A is only accessible from B, B is accessible from both A and C, while C is only accessible if already in mode B). Now, suppose that only a and c is in the input alphabet along with some other messages from the same message type (but not b). In that case, c might be falsely discarded since it does not do anything unless the protocol is in mode B, but there is no way to make that happen with this collection of messages. This might be countered by increasing the number of message instances that are added each time, but this increases the queries needed to run the algorithm.

The second shortcoming is related to the generator function. As long as the generator produces message instances at a uniform distribution from each subtype, there is no problem; however, it is not trivial to write such a generator. For example, if a generator is generating read messages that read from a specific memory address, and fills the target address at random, the rate at which it will produce "valid" messages that result in a response with the read value and "invalid" messages that return an error because of an out of bound read depends on the memory layout of the device that is being tested. This could lead to never discovering one of the subtypes, or extending the runtime of the algorithm by selecting from only one subtype for many rounds and only later finding the other.

To deal with these problems, we need to have access to previously recorded traffic of the protocol that is being reversed – with information regarding resets. It is not unrealistic to have access to such traffic since network captures are usually used to reproduce the message types and semantics that are direct inputs to our algorithm. Of course, the quality and diversity of this capture will directly affect the results of the algorithm. We use the capture to find message instances that do not belong to any subtypes we have previously found. To do so, after each round, we compare the recorded communication with the predictions that the Mealy machine of the last round would make from the inputs. We classify the messages in the recorded traffic with the classifier, creating a sequence of message types, keeping track of which were queries (from client to server) and which were responses (from server to client). We feed the queries into a Mealy machine produced from the output of the last round and compare the outputs with the responses. If there is a difference, we know that a message in the communication up to that point is not covered in the Mealy machine, so all the queries in the flow are added to the input alphabet, and a new round is started. The modification is necessary because the Mealy machines produced by the ${L_M}^+$ algorithm in this setting expect message subtypes as inputs, but the message classifier can only produce message types.

To be precise, we actually convert the output Mealy machine into a simple Deterministic Finite Automaton (DFA) that takes inputs that are pairs of query and response messages. The DFA accepts a sequence if it could be produced by the Mealy machine. As a first step, we take the original machine and construct a machine that is the exact copy of the Mealy machine, except each transition in the new Finite State Machine (FSM) is the pair of input and output message types in the original Mealy machine (the subtype is stripped here and duplicate transitions are discarded). Every state of this new machine is an accept state since every transition that is present here is represented in the original Mealy machine. We analyze the resulting machine to collect its alphabet. If the machine is incomplete (i.e., there are states where there is no transition with all letters of the alphabet, meaning there are some cases where the behaviour of the machine is not specified), we fully specify it by inserting the missing letters as transitions to the *FAIL* state. This is a new state that only has loop transitions, and is not an accept state. At each state, a missing input-output pair means that the original Mealy machine could not produce the given output for the given input in that state, so if this is encountered, the sequence is not covered. While analyzing a message sequence the first time the machine enters the *FAIL* state, the sequence up to that message pair can be used as new message instances for the next round. The same is true if we encounter a query/response pair that is not in the alphabet of the machine.

Now we have a machine that can handle the input we have, however, since we stripped away the subtypes, it may be non-deterministic (there may be multiple transitions from the same state with the same letter). This is not a problem, as there are known algorithms to transform non-deterministic state machines into deterministic ones. However, the resulting DFA may be exponentially large, which may be a problem for big inputs. It may be useful to run a minimization algorithm after determinization to get a minimal DFA.

One more improvement can be made by not completely stripping away the subtype information in the first step, but rather adding a transition with only the message type and also keeping the one with the subtype specified. The rest of the steps of creating the DFA remain the same. This is useful because for some of the messages, we can determine their subtype: these are the message instances that were used in the last round of learning. If there is an exact match with one of these, we can leverage this additional information.

5 Implementation and evaluation

We implemented the ${L_M}^+$ algorithm and our algorithm in Python using the NetworkX[1] package to represent Mealy machines. We designed the implementation to be modular, such that the different components are well separated and easy to replace. This is important for future improvements and to be able to easily plug in the functions that are different for each protocol (e.g., the message generator, the message classifier, and the part of the teacher that handles communication with the SUT).

[1]`https://networkx.github.io/`

5.1 Evaluation on a simple test protocol

For test and illustration purposes, we constructed and used a simple protocol which is illustrated in Figure 1. The protocol has 3 states: the starting state *DISCONNECTED*, the state *BASE*, where only *get* is a valid input, and the state *WRITE*, where both *get* and *write* are valid inputs. The difference between *get* and *bad_get* is that the parameter (target address) of the *get* message is valid, while it is invalid in a *bad_get*, and likewise with *write* and *bad_write*. Messages *get* and *bad_get* are of the same type, and similarly, messages *write* and *bad_write* have the same type. These two message types are used by the message generator to generate messages with random addresses; no recorded traffic analysis was used in this test.

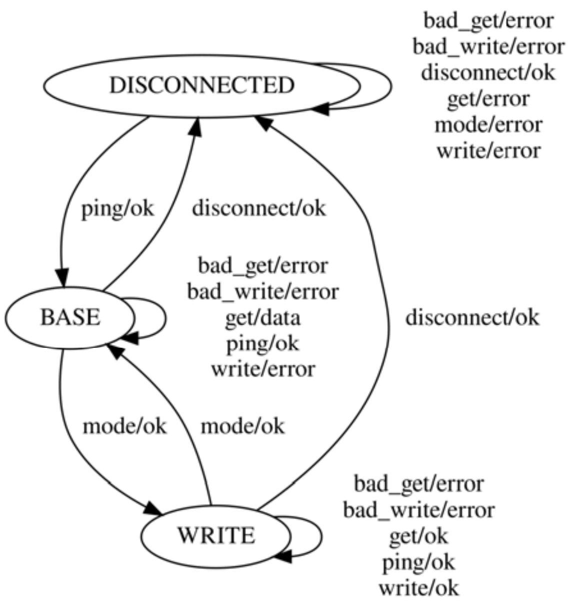


Figure 1: The Mealy machine of the simple test protocol

Figures 2, 3, and 4 show the inferred Mealy machine in different rounds after deduplication. The request messages are postfixed with a number; this is a counter showing how many times the message generator was called when the given message was generated. The starting state is *, and every other state is labelled by the messages that can be used in sequence to reach that state. In the first round, the algorithm generated two instances of each message type; however, each instance of the same message type produced the same behaviour, therefore, only one of them was kept (see Figure 2). In the second run, a new instance was generated from each message type, but these did not show new behaviour either, so they were discarded too. In the third round, a variant of the *write* message type was generated that resulted in an *ok* response, as opposed to the *error* response triggered by the

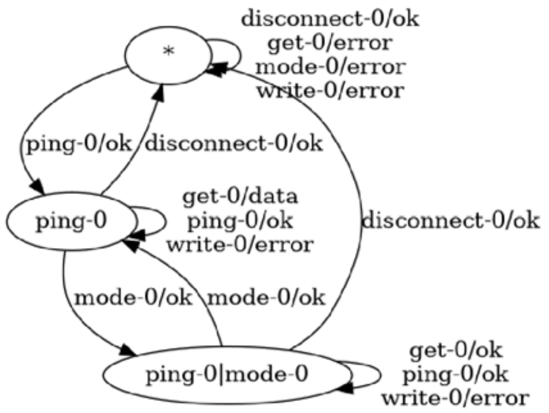


Figure 2: Output of the first round

other variant of the *write* message, so this was kept (see Figure 3). Finally, in the fourth round, the algorithm also finds a *get* message variant that results in different response observed so far, hence it is retained (see Figure 4). After five rounds with no new behaviour found, the algorithm stopped. As we can see in Figure 4, in our example, the Mealy machine inferred is identical to the state machine of the example protocol (except for the names of the states and message variants, of course).

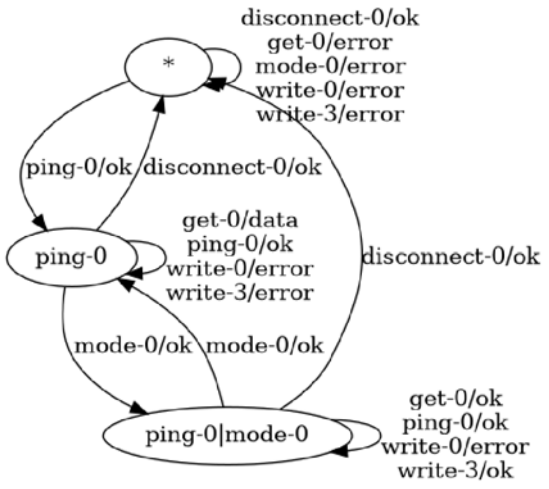


Figure 3: Output of the third round

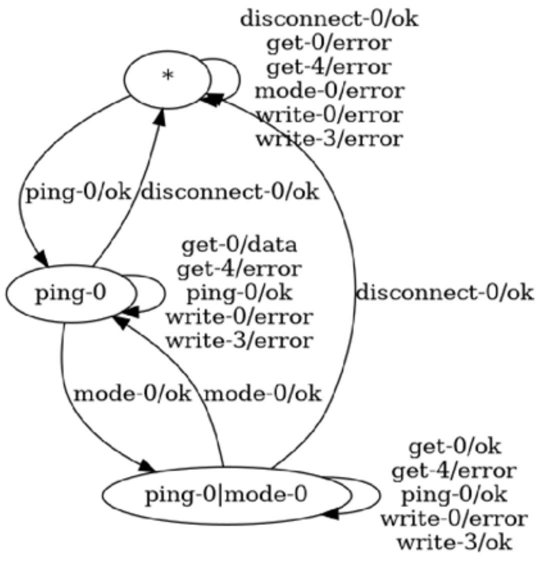


Figure 4: Output of the fourth round

5.2 Evaluation on Modbus

Modbus is a widely used industrial protocol that allows a client to read data from and write data to the server, usually a Programmable Logic Controller (PLC). For our tests, we used a custom implementation for both the client and the server.

Modbus messages begin with a fixed 8-byte header that is followed by a data part of variable length. For the classifier, we only used the last byte of the header as this byte determines the packet type. We constructed the generation function by classifying messages in the recorded network traffic and selecting from them when necessary, using the strategy described in Section 4.

In order to bridge the gap between real TCP communication and the abstract message instances described until now, some additional *artificial* message types were defined in the proxy that translates between the two. The first one is necessary because, to some queries, the server might simply not reply. When a certain timeout is reached without any response from the server, we conclude that it will not respond at all and return a *noresp* message as the response. The other possibility is the server terminating the connection because of some protocol violation, or because of a *DISCONNECT* packet. To model the communication properly, the proxy starts out with no open socket, and as long as there is no socket, it returns *noresp* to all queries. We define the *sockconn* query message, which results in the proxy opening a new TCP connection to the server and storing the socket – replacing the previously stored socket, if there was one –, and returning *sockconn.*

The output of our algorithm for Modbus can be seen in Figure 5. The input

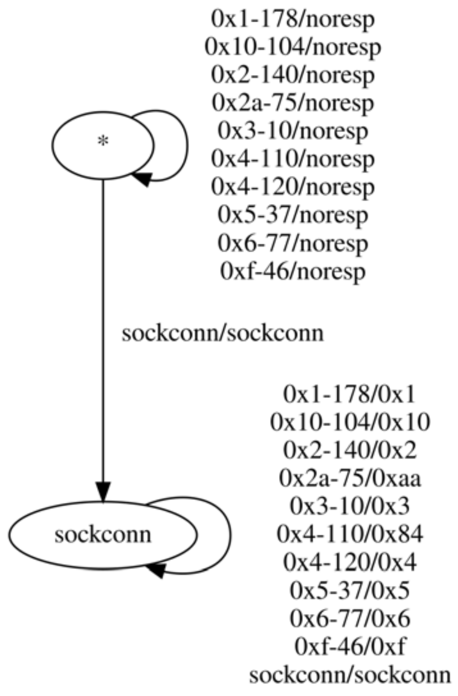


Figure 5: Resulting Mealy machine for Modbus

letters on the Mealy machine are the various function codes of Modbus, postfixed with a unique identifier to separate message subtypes. The output letters are the function codes that the server replied with. The figure shows that from a protocol state machine point of view, the Modbus protocol is very simple as it has two states: in the starting state only the *sockconn* message results in any response, as there is no connection to send or receive the other messages, and in the other state, the server always responds to queries. For each request (function code), the response has the same code as the query, except if an error is detected, in which case the response code is the query code plus $0x80$.

In our case, for request code $0x2a$, we always get sent back an error because it is an invalid function code, but for $0x4$ there is a subtype for a correct version of the query and one for the incorrect version. The network capture we used did not contain invalid uses of the other function codes, so those are not present in the produced Mealy machine.

Based on the open specification of the protocol [18], these results are 100% accurate.

5.3 Evaluation on MQTT

MQTT is a messaging protocol implementing a publish-subscribe model. Clients can subscribe to channels and will receive all messages posted to these channels. This protocol is ideal for evaluation because its simplicity and open specification allow easy verification. There are also plenty of free implementations for servers and client libraries, making it convenient to set up a test environment. For the client-side, we used the Eclipse Paho™ MQTT Python Client[2] library, and for the server-side we used Eclipse Mosquitto[3].

MQTT messages are comprised of a fixed header, a variable header and a payload. We only used the first four bits of the fixed header for the classifier as these bits determine the packet type according to the documentation. The second byte was used as well, to determine the length of the message, in case more than one message was sent in a single IP packet. For the classifier and the generation functions, we have taken the same approach as in the case of Modbus, with the exception that a third artificial message that is specific to MQTT was also defined. This message, called *pubtrigger*, simulates other clients connected to the MQTT server, and triggers a publish to one of the channels.

The MQTT protocol has three different Quality of Service (QoS) levels that govern how messages are published. As the level of QoS increases, so do the number of roundtrips and the confidence of the message reaching its destination. We used only messages with QoS 0, and in separate tests, we generated traffic that used one, two and three channels for posting messages. We configured the MQTT server with two different username and password combinations and forbade anonymous connections. Then we performed randomized actions through the client library by selecting from connecting, publishing, subscribing, unsubscribing, and disconnecting from the server. The channel and the message, where applicable, were also randomly selected. The generated network traffic was captured and used for the construction of the generation function and to prioritize the selection of message instances. As it can be seen on final results of the algorithm (Figures 6 and 7), at this level of QoS, the protocol is relatively simple: one can subscribe and publish to any channel after connecting with the correct credentials.

The outputs produced by the algorithm are correct based on the MQTT protocol specifications [4], taking into account that we only used a subset of the possible functions of the protocol.

We can see that membership in the various channels is entirely independent. As the number of potential channels increases, so do the number of states needed to keep track of how many of them the client is currently subscribed to. One state is necessary for each combination of channels that are joined, meaning two states when only one channel is used (the client is either subscribed or not), four with two channels (subscribed to none, either one, or both channels), and eight if there are three channels. Generally, the formula is 2^n, where n is the number of channels that are used. After reaching a high enough n, this is not very useful as it makes the

[2]`https://github.com/eclipse/paho.mqtt.python`

[3]`https://github.com/eclipse/mosquitto`

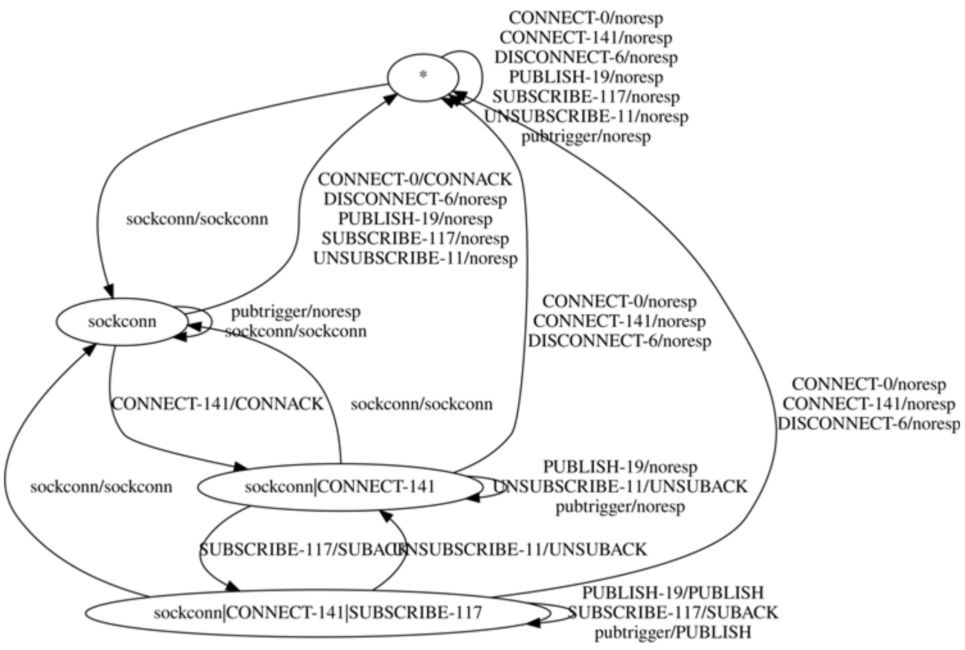


Figure 6: Resulting Mealy machine for MQTT with a single channel and QoS 0

analysis of the Mealy machine difficult because of its size, and it does not provide any new information. This is a limitation of using Mealy machines to represent the protocol. To counteract this problem, the number of such possible combinations should be kept to a low number. In our case, this means keeping the number of possible channels low.

We came up with a method that may help find such independent message subtypes. First, we define a *projection* of a Mealy machine on a set S of input letters. For all input letters in the input alphabet that are not in S, we ignore the output letter. In the practical implementation, we modify the output on the transitions that contain these letters, we change the output letter to a fixed value, in our case *noout*. After this, we run a minimization algorithm on the new Mealy machine, and the result will be the projected machine. During the minimization, all input letters that are not in S and do not influence the outputs of the transitions that contain letters that are in S completely disappear. At first, we project to single letters of the input alphabet, and then for each of these projections, we try to add another input letter to S in addition to the original one. If the resulting Mealy machine has the same number of states as the previous projection, we keep the letter in S; otherwise we remove it. After repeating with all input letters, we can take all the projections with a different set S and analyze them. These Mealy machines can be used together to replace the original Mealy machine, giving the inputs to all of

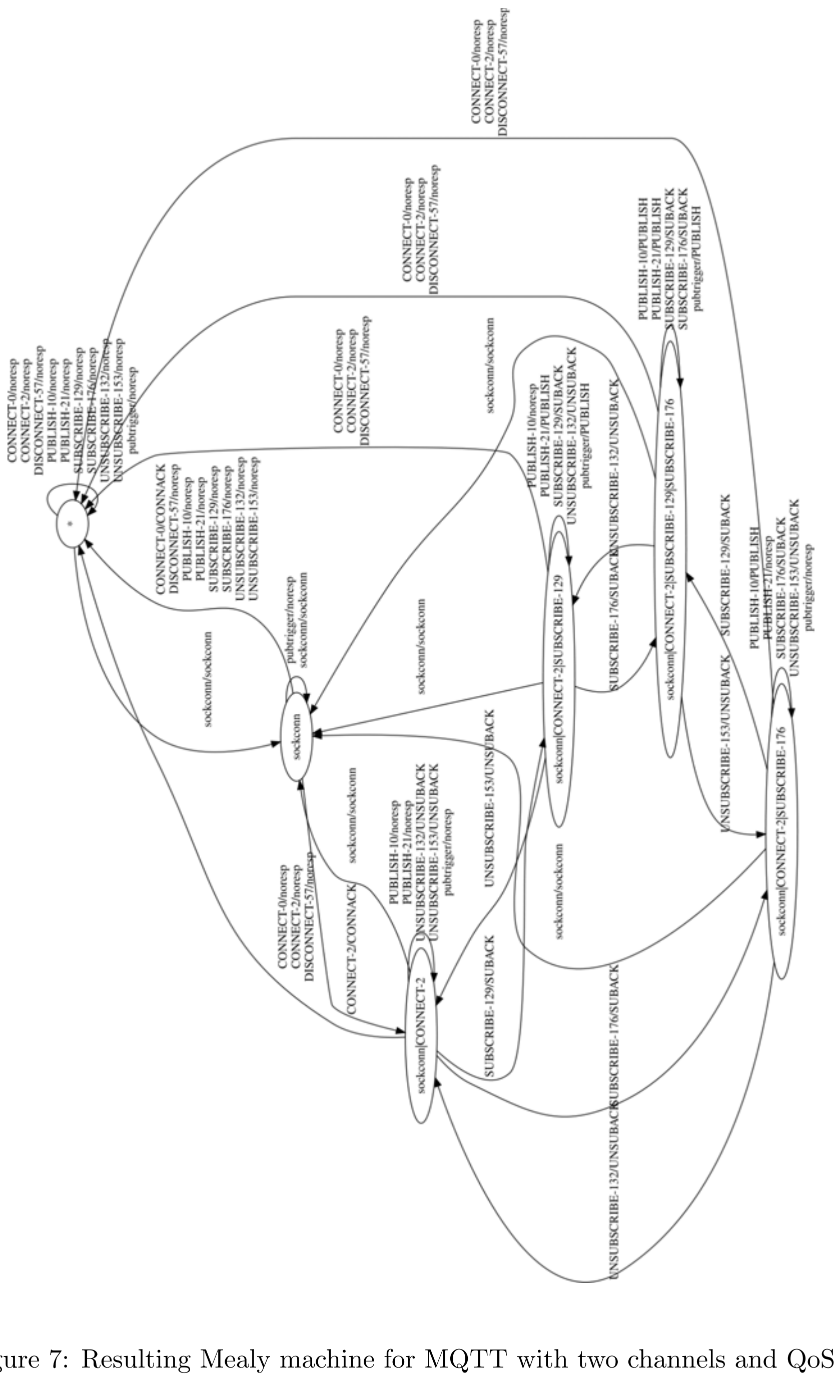

*
sockconn
sockconn|CONNECT-2
sockconn|CONNECT-2|SUBSCRIBE-129
sockconn|CONNECT-2|SUBSCRIBE-129|SUBSCRIBE-176
sockconn|CONNECT-2|SUBSCRIBE-176
CONNECT-0/noresp CONNECT-2/noresp DISCONNECT-57/noresp PUBLISH-10/noresp PUBLISH-21/noresp SUBSCRIBE-129/noresp SUBSCRIBE-176/noresp UNSUBSCRIBE-132/noresp UNSUBSCRIBE-153/noresp pubtrigger/noresp
CONNECT-0/CONNACK DISCONNECT-57/noresp PUBLISH-10/noresp PUBLISH-21/noresp SUBSCRIBE-129/noresp SUBSCRIBE-176/noresp UNSUBSCRIBE-132/noresp UNSUBSCRIBE-153/noresp
pubtrigger/noresp sockconn/sockconn
sockconn/sockconn
CONNECT-0/noresp CONNECT-2/noresp DISCONNECT-57/noresp
CONNECT-0/noresp CONNECT-2/noresp DISCONNECT-57/noresp
CONNECT-0/noresp CONNECT-2/noresp DISCONNECT-57/noresp
CONNECT-0/noresp CONNECT-2/noresp DISCONNECT-57/noresp
sockconn/sockconn
sockconn/sockconn
PUBLISH-10/noresp PUBLISH-21/PUBLISH SUBSCRIBE-129/SUBACK UNSUBSCRIBE-132/UNSUBACK pubtrigger/PUBLISH
PUBLISH-10/PUBLISH PUBLISH-21/PUBLISH SUBSCRIBE-129/SUBACK SUBSCRIBE-176/SUBACK pubtrigger/PUBLISH
UNSUBSCRIBE-132/UNSUBACK
SUBSCRIBE-176/SUBACK
SUBSCRIBE-129/SUBACK
PUBLISH-10/PUBLISH PUBLISH-21/noresp SUBSCRIBE-176/SUBACK UNSUBSCRIBE-153/UNSUBACK pubtrigger/noresp
UNSUBSCRIBE-153/UNSUBACK
CONNECT-2/CONNACK
sockconn/sockconn
PUBLISH-10/noresp PUBLISH-21/noresp UNSUBSCRIBE-132/UNSUBACK UNSUBSCRIBE-153/UNSUBACK pubtrigger/noresp
UNSUBSCRIBE-153/UNSUBACK
sockconn/sockconn
SUBSCRIBE-129/SUBACK
SUBSCRIBE-176/SUBACK
UNSUBSCRIBE-132/UNSUBACK

Figure 7: Resulting Mealy machine for MQTT with two channels and QoS 0

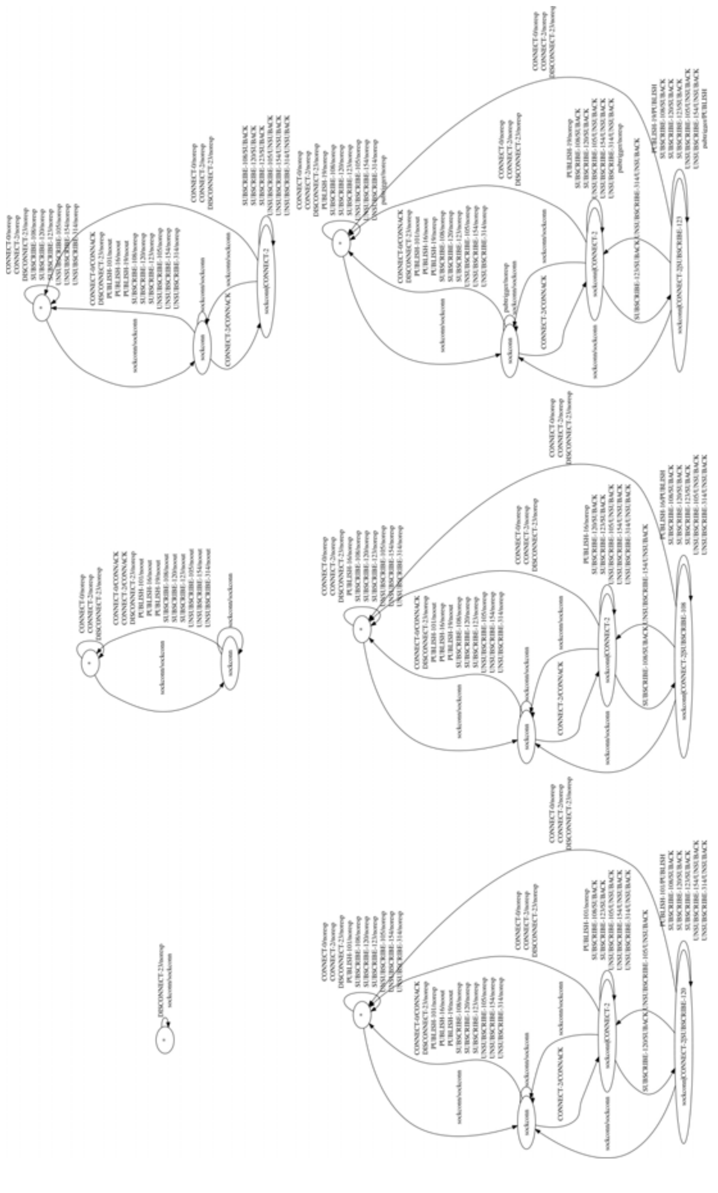


Figure 8: Results of the projection generation algorithm on the Mealy machine of MQTT with three channels with various sets of S. Each of the last three machines corresponds to a single channel. The figure is intended to illustrate the complexity of the resulting state machines only, the reader is not expected to be able to read the labels of the states or transitions.

them, and taking the output from either one which can produce it (if either Mealy machine does not have a transition with the particular letter at its current state, it should just do nothing). For the projections produced in this manner for QoS level 0 and three channels, the Mealy machine can be seen in Figure 8. The three machines in the bottom row can be grouped as implementing the same logic, only with different subtypes, and indeed, each corresponds to a single channel.

When testing with higher QoS, the same problems come up as with using multiple channels, but this time around the state of publishing is the culprit behind the state explosion. The above-described technique did not work so well in separating any independent parts of the machine.

6 Conclusion

In this work, we build on Shahbaz and Groz's L_M^+ algorithm (an adaptation of Angluin's L^* algorithm) and make use of captured network traffic as well as live queries to a known good protocol implementation in order to infer the state machine of an unknown protocol. We assume that message types and semantics have already been reverse engineered, and we use this information to intelligently and efficiently choose the possible inputs to use when querying the system under test. We demonstrate our methods on an example protocol created for this purpose, in addition to two real-world protocols, Modbus and MQTT. We show that our approach works by comparing the resulting automata to the original specifications.

In the future, we plan to investigate more complicated Mealy machine decomposition algorithms to produce hierarchical Mealy machines that can represent the protocol state machine more concisely. In addition, we aim to measure and further optimize input selection strategies.

References

[1] Andrew, Tridgell. How samba was written. `https://download.samba.org/pub/tridge/misc/french_cafe.txt`, 2003.

[2] Angluin, Dana. Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2):87–106, nov 1987. DOI: `10.1016/0890-5401(87)90052-6`.

[3] Antunes, João, Neves, Nuno Ferreira, and Verissimo, Paulo. ReverX: Reverse engineering of protocols. *12th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT)*, 2011.

[4] Banks, Andrew and Gupta, Rahul. MQTT Version 3.1.1 Plus Errata 01 (OASIS Standard Incorporating Approved Errata 01). `http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html`, 2015.

[5] Beddoe, Marshall A. Network protocol analysis using bioinformatics algorithms, 2004. http://www.4tphi.net/~awalters/PI/PI.html.

[6] Bermudez, Ignacio, Tongaonkar, Alok, Iliofotou, Marios, Mellia, Marco, and Munafò, Maurizio M. Towards automatic protocol field inference. *Computer Communications*, 84:40–51, 2016. DOI: 10.1016/j.comcom.2016.02.015.

[7] Bossert, Georges, Guihéry, Frédéric, and Hiet, Guillaume. Towards automated protocol reverse engineering using semantic information. *Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security*, pages 51–62, 2014. DOI: 10.1145/2590296.2590346.

[8] Cho, Chia Yuan, Babi ć, Domagoj, Shin, Eui Chul Richard, and Song, Dawn. Inference and analysis of formal models of botnet command and control protocols. In *Proceedings of the 17th ACM Conference on Computer and Communications Security*, CCS '10, page 426–439, New York, NY, USA, 2010. Association for Computing Machinery. DOI: 10.1145/1866307.1866355.

[9] Cho, Chia Yuan, Babić, Domagoj, Poosankam, Pongsin, Chen, Kevin Zhijie, Wu, Edward XueJun, and Song, Dawn. MACE: Model-inference-assisted concolic exploration for protocol and vulnerability discovery. *SEC'11 Proceedings of the 20th USENIX Conference on Security*, pages 139–155, 2011.

[10] Comparetti, Paolo Milani, Wondracek, Gilbert, Kruegel, Christopher, and Kirda, Engin. Prospex: Protocol specification extraction. *30th IEEE Symposium on Security and Privacy*, pages 110–125, 2009. DOI: 10.1109/SP.2009.14.

[11] Cui, Weidong, Kannan, Jayanthkumar, and Wang, Helen J. Discoverer: Automatic protocol reverse engineering from network traces. *SS'07 Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium*, 2007. https://www.usenix.org/conference/16th-usenix-security-symposium/discoverer-automatic-protocol-reverse-engineering-network.

[12] Cui, Weidong, Paxson, Vern, Weaver, Nicholas C., and Katz, Y H. Protocol-independent adaptive replay of application dialog. In *Network and Distributed System Security Symposium*, 2006. https://www.ndss-symposium.org/ndss2006/protocol-independent-adaptive-replay-application-dialog/.

[13] Duchêne, Julien, Guernic, Colas Le, Alata, Eric, Nicomette, Vincent, and Kaâniche, Mohamed. State of the art of network protocol reverse engineering tools. *Journal of Computer Virology and Hacking Techniques*, 14(1):53–68, 2017. DOI: 10.1007/s11416-016-0289-8.

[14] Ládi, Gergő, Buttyán, Levente, and Holczer, Tamás. GrAMeFFSI: Graph analysis based message format and field semantics inference for binary protocols using recorded network traffic. *Infocommunications Journal*, 12(2):25–33, August 2020. DOI: `10.36244/ICJ.2020.2.4`.

[15] Lee, David and Yannakakis, Mihalis. Principles and methods of testing finite state machines – A survey. *Proceedings of the IEEE*, 84(8):1090–1123, 1996. DOI: `10.1109/5.533956`.

[16] Leita, C., Mermoud, K., and Dacier, M. ScriptGen: an automated script generation tool for Honeyd. *21st Annual Computer Security Applications Conference*, pages 203–214, 2005. DOI: `10.1109/CSAC.2005.49`.

[17] Luo, Jian-Zhen and Yu, Shun-Zheng. Position-based automatic reverse engineering of network protocols. *Journal of Network and Computer Applications*, 36(3):1070–1077, 2013. DOI: `10.1016/j.jnca.2013.01.013`.

[18] Modbus Organization, Inc. Modbus application protocol specification v1.1b3. `http://www.modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf`, 2012.

[19] Narayan, John, Shukla, Sandeep K., and Clancy, T. Charles. A survey of automatic protocol reverse engineering tools. *ACM Computing Surveys*, 48(3), 2016. DOI: `10.1145/2840724`.

[20] Shahbaz, Muzammil and Groz, Roland. Inferring mealy machines. In *International Symposium on Formal Methods*, pages 207–222. Springer, 2009. DOI: `10.1007/978-3-642-05089-3_14`.

[21] Wang, Y., Xiaochun Yun, Shafiq, M. Z., Wang, L., Liu, A. X., Zhang, Z., Yao, D., Zhang, Y., and Guo, L. A semantics aware approach to automated reverse engineering unknown protocols. *20th IEEE International Conference on Network Protocols (ICNP)*, pages 1–10, 2012. DOI: `10.1109/ICNP.2012.6459963`.

[22] Wang, Yipeng, Li, Xingjian, Meng, Jiao, Zhao, Yong, Zhang, Zhibin, and Guo, Li. Biprominer: Automatic mining of binary protocol features. *12th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT)*, pages 179–184, 2011. DOI: `10.1109/PDCAT.2011.25`.

[23] Wang, Yipeng, Zhang, Zhibin, Yao, Danfeng, Qu, Buyun, and Guo, Li. Inferring protocol state machine from network traces: A probabilistic approach. *ACNS 2011: Applied Cryptography and Network Security*, pages 1–18, 2011. DOI: `10.1007/978-3-642-21554-4_1`.

Acta Cybernetica **25** (2021) 537–554.

Component-Based Error Detection of P4 Programs[*]

Gabriella Tóth[a] and Máté Tejfel[b]

Abstract

P4 is a domain-specific language to develop the packet processing of network devices. These programs can easily hide errors, therefore we give a solution to analyze them and detect predefined errors in them. This paper shows the idea, which works with the P4 code as a set of components and processes them one by one, while calculating their pre- and postconditions. This method does not only detect errors between the components and their connections, but it is capable to reveal errors, which are hidden in the middle of a component. The paper introduces the method and shows its calculation in an example.

Keywords: P4, error detection, component

1 Introduction

We introduce a component-based formal method to detect errors in P4 programs. The antecedent of the method is an error detection, which is based on a rule system [13]. With this idea, we approach the detection from backward and process the code from the smallest units to the biggest ones. This solution can not only check the error possibilities but give additional information about the code for the developer. The prototype of the tool, which is based on the method, is being implemented[1].

P4 programs We work with the P4 language [5, 2, 6, 1], which is a domain-specific programming language to develop the packet processing of network devices. When a packet arrives at a device as a bitstream, the P4 program gets that bitstream as the input and starts to work with it. In Figure 1 there is an example program. P4 programs work with the header information of a network packet. The developers can define what kind of header information they work with (rows 1-16). P4 programs have three main processing parts: parser, modifier, and deparser. The

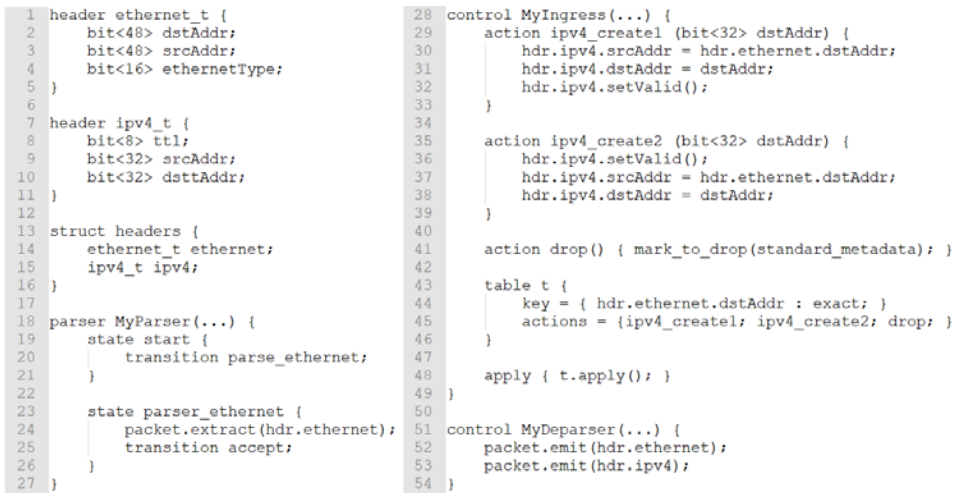
[*]This work has been supported by the European Union, co-financed by the European Social Fund (EFOP-3.6.3-VEKOP-16-2017-00002)

[a]Faculty of Informatics, Eötvös Loránd University, Budapest, Hungary, E-mail: `kistoth@inf.elte.hu`, ORCID: `0000-0001-9657-7231`

[b]Department of Programming Languages and Compilers, Eötvös Loránd University, Budapest, Hungary, E-mail: `matej@inf.elte.hu`, ORCID: `0000-0001-8982-1398`

[1]`https://github.com/tothgabi/ELTE_P4_Analyzer`

DOI: `10.14232/actacyb.289731`



```
 1 header ethernet_t {
 2     bit<48> dstAddr;
 3     bit<48> srcAddr;
 4     bit<16> ethernetType;
 5 }
 6
 7 header ipv4_t {
 8     bit<8> ttl;
 9     bit<32> srcAddr;
10     bit<32> dsttAddr;
11 }
12
13 struct headers {
14     ethernet_t ethernet;
15     ipv4_t ipv4;
16 }
17
18 parser MyParser(...) {
19     state start {
20         transition parse_ethernet;
21     }
22
23     state parser_ethernet {
24         packet.extract(hdr.ethernet);
25         transition accept;
26     }
27 }
28 control MyIngress(...) {
29     action ipv4_create1 (bit<32> dstAddr) {
30         hdr.ipv4.srcAddr = hdr.ethernet.dstAddr;
31         hdr.ipv4.dstAddr = dstAddr;
32         hdr.ipv4.setValid();
33     }
34
35     action ipv4_create2 (bit<32> dstAddr) {
36         hdr.ipv4.setValid();
37         hdr.ipv4.srcAddr = hdr.ethernet.dstAddr;
38         hdr.ipv4.dstAddr = dstAddr;
39     }
40
41     action drop() { mark_to_drop(standard_metadata); }
42
43     table t {
44         key = { hdr.ethernet.dstAddr : exact; }
45         actions = {ipv4_create1; ipv4_create2; drop; }
46     }
47
48     apply { t.apply(); }
49 }
50
51 control MyDeparser(...) {
52     packet.emit(hdr.ethernet);
53     packet.emit(hdr.ipv4);
54 }
```

Figure 1: Example P4 code

parser (rows 18-27) gets the input and extracts the header information from the packet. The modifier part (rows 28-49) modifies the header information – based on match-action tables, the content of which comes from an external controller – and after the modification, the deparser (rows 51-54) creates the new packet with the calculated header information to forward it to the network.

Software-defined networks have two main parts: the control plane – which controls the network traffic – and the data plane – which describes the process of a network packet. As we can see from the example, P4 programs can define the data plane. Therefore, the behavior of the programs depends on the control plane, which cannot see from the P4 code – for example, there is a program structure, the match-action table, which is only a frame in the code, and its content is filled by an external controller during the execution.

2 Related Work

There are different approaches to analyze P4 programs.

There are tools, which work with the previous version of P4 – the P4$_{14}$ – although the main concept of these projects is similar to the new version. One of them is P4V [9], which deals with errors caused by reading or writing of invalid headers, the arithmetic overflow, and proper accessing of header stacks. It creates an annotated program from the input source and verifies it by checking the correctness of a first-order formula, which describes the execution of the program.

Vera [11] is another one, which works with symbolic execution, which starts with the generation of the parsable input packet, and then checks all possible execution path with the Symnet [12] static analysis tool. They investigate similar error cases, for example, header validity, implicit drops.

Assert-P4 [8] also works with $P4_{14}$, and uses symbolic execution too, although they check program-specific errors, which can be described by the developer. Therefore it can check errors, which can not be detected only from the simple P4 source, but from an annotated P4 code. It translates the annotated P4 code to a C-model, and then it checks the different properties, which were given in the code, with symbolic execution.

There are other tools, which work with the actual version of P4 – the $P4_{16}$. There is an approach, which does data flow analysis of P4 programs [4]. In this research, they used an extended version of a control-flow graph, the def-use graph. For all possible execution paths, they execute the data flow analysis, which collects the usage of the different variables and header information. Based on this collected data, their solution can detect errors like using an undefined field.

Although the previous projects are based on static analyzes, there are other solutions too. P4RL [10] does runtime verification, therefore it is able to analyze behaviors which are come from the control plane. The behavior of the network should be described by the user, therefore it can check if all of them are correct during the execution.

Another approach is SafeP4 [7], which is a domain-specific language, which was created to develop correct P4 programs.

Our solution works with the new version of P4 – the $P4_{16}$. It is a static analyzer method, which uses only the P4 code as an input. Based on the code, we can extract the specification of the program, as pre- and postconditions, and we detect errors by analyzing the pre- and postconditions of the different components of the program and checking their relationship with the specification of the program. We detect errors, which can be caused by the usage of invalid header and uninitialized fields.

3 Motivation

P4 gives a new possibility for the protocol-independent development of the programmable switches. However developers can create more flexible programs, it is easier to make a mistake in them. That is the reason, why we need a tool, which can detect errors in our code.

As it was mentioned in the Introduction, the P4 programs do not define the whole behavior of the network, but a part of it. Therefore, some verification tools expect annotated programs and additional information about the planned behavior of the program, and they can check if those properties are correct or not. We would like to detect errors based on only the raw P4 code, therefore the developers will not have any other job besides developing the P4 program.

We would like to create a tool, based on our theoretical results, to help the developers to write correct P4 programs. In this first version of the method, we concentrate only on those errors to detect, which are caused by using invalid headers and uninitialized fields. Using them can cause undefined behavior by unknown values. However, we still work with the subset of the P4 language, but we plan to extend the rule system to work with a bigger language and detect more error cases.

This method is based on the precondition and postcondition of the program units. As we see in Figure 1, there are different programming structures in P4: actions, tables, control functions. It works with them one by one, and calculates condition pairs for each of them, starting from the smallest ones – like actions – finishing with the biggest ones – like control functions.

4 Method of detection

The input of the method is the source code that will be checked. The whole process contains three main phases: the Pipeline Analyzer, the Parser/Deparser Analyzer, and the Final Checker. The Pipeline Analyzer has two parts: the call graph, and the Condition calculator. The Pipeline Analyzer checks all of the components – actions, tables, and control functions – in the P4 source and calculates pre- and postcondition pairs for them while checking their correctness. The Parser/Deparser Analyzer works with the parser – gets the main precondition from it – and the deparser – gets the main postcondition from it. The Final Checker works with the calculated pieces of condition and checks that if every main precondition matches with at least one needed precondition. If the matching is not correct then it means that there are some errors in the code, because after the given parsing – which is described by the main precondition – there is not any execution path, which could work properly. However, if the matching is right in the preconditions then it checks the same with the postconditions.

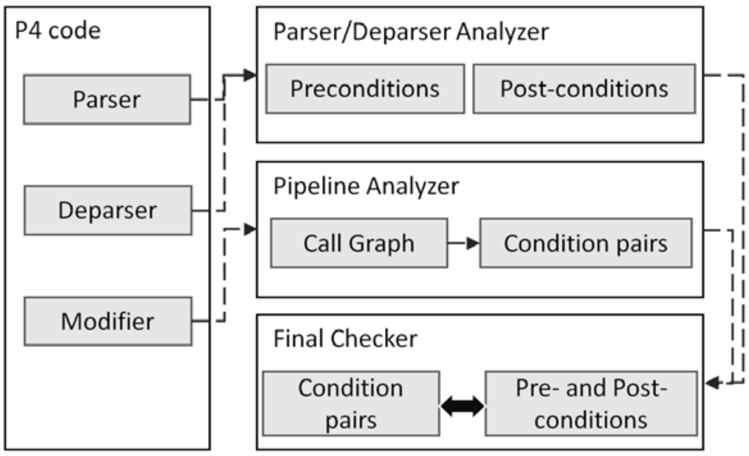


Figure 2: Model of the method

4.1 Pipeline Analyzer

The analyzer consists of a call graph and the Condition Calculator. The call graph contains the relationship of the call between the control functions, tables, and actions. Based on the graph it will make an order of these components to be processed by the Condition Calculator. The Condition Calculator goes through all of them and creates their pairs of pre- and postcondition, which describes the correct working of the program.

This phase processes the modifier part of the P4 code, and calculate pre- and postconditions for the different components. The result of this phase shows the claim – what kind of header information it needs to work well – and the offering – what kind of header information it will create after the processing – of the component.

Call Graph The call graph handles the modifier part of the input. Its vertices are the components of this part – the control functions, match-action tables, and actions – and the directed edges describe a calling relation between them.

The processing of the modifier part, in this version of the method, is the simplest packet processing path, where there is only an ingress pipeline. Therefore there is an initial vertex, which is the main ingress control function. The forwarding steps to process the whole control flow and whenever a component calls a component – if that has not been added to the graph as a vertex, it will create a vertex for it – and will give an edge from the vertex of the calling component and the vertex of the called one.

After the processing of the code, it will create a list of the components. This list starts with those components, which do not depend on any other components, – they have no outgoing edges –, it continues with components, which calls the components, which have already been added to the list, until the last two components, which will be the main function of the ingress. In the call graph there can not be any loop, because of the specification of the P4 language [6], which defines that neither direct recursion nor mutual recursion can be between actions and controls.

Condition calculator The list of the components gives a hint for the Condition Calculator in which order it should process the components, therefore whenever a command of a component's calling is processed, the condition of that component will have been calculated. During the processing of the components, it uses a rule system, which will calculate a *ConditionState*.

$$\begin{aligned}\text{ConditionState} \in \{name : [(&pre : \{valid : [ids], invalid : [ids]\},\\ &post : \{valid : [ids], invalid : [ids]\})]\}\end{aligned}$$

Figure 3: Condition State Describer

The type of the *ConditionState* – Figure 3 – is a set of pairs, where a pair shows the name of the component and its list of pre- and postconditions as pairs.

Every condition consists of a *valid* and an *invalid* container, both of them contain a list of headers' and fields' names. During the method, we use the *valid* word as a synonym of the property of initialized fields.

$$\vdash\text{: (ConditionState} \times \text{Name} \times \text{Program)} \rightarrow \text{ConditionState}$$

Figure 4: Type of $\vdash$

The Condition Calculator is defined by the operator $\vdash$, the type of which can be seen in Figure 4. It works with a known *ConditionState*, a name of a component – the one, that is processed when it uses this rule – and the program code. The rules are based on the structure of the program, therefore they have a deterministic usage. In this version of our method, we only work with the main structures of the program, therefore the rules only use a subset of the P4 language.

We define the behavior of the Condition Calculator with a rule system. It uses rewriting rules – we have an expression in the bottom and we rewrite it to the top of the rule. In Figure 5, there are the basic rules, which can be used for the subset of the P4 program. The rules for the assignment, sequence, table, skip, branch, table calling, and validity settings.

The preprocessing of the usage of the rules is to create the initial *ConditionState*. It is an empty state, where all of the components are stored, but all of them have only one condition, which says that the *drop* is invalid in the postcondition. The *drop* is a unique name in the conditions, which shows if the packet is dropped – it is valid, if it is dropped, and invalid if it is not dropped.

In the rules, we use some notations:

- $(A \parallel B)$ shows a rewriting from A to B. If the rewrote element has already been in a set, then it will be deleted from that place, and it will be only in the new set.
- *ids* means the type of the identifications. For example $\{ipv4, ipv4.ttl\}$ is a set of ids i.e. {ids}.
- Ids is a function, which gives the names from an expression. For example the Ids of the "$ipv4.ttl - 1$" is a $\{ipv4, ipv4.ttl\}$, which contains the identification of the header and the field of the expression.
- *newIds* is a function, which gives a subset of the Ids. These elements have not been used in any condition state.
- *oldIds* is a function, which gives a subset of the Ids. These elements have already been used in a condition state.
- $fields$ is a set of the identification of the fields of the given expression.
- *header* is a set of the identification of the headers of the given expression.

- $C(\mathit{conditionState}, \mathit{condition})$ calculates those pairs from the condition state, which fit to the condition.
- $H.n$ shows the pre- and postcondition pairs of the component named n in the Condition State H.
- $\mathit{Empty}(H.n)$ creates an empty block for the new component.
- $V : \{\mathit{ids}\} \rightarrow \mathit{Bool}$; $V(I) = \forall i \in I(H.n.post.Valid(i))$. It checks if every header, field or variable identification is valid in the postconditions of the actual component – the examined *ConditionState* H and component n come from the rule.
- $A\dot{\cup}B ::= \mathit{ifcond}(A.post =:= B.pre, A.pre, A.post \times B.post \uparrow)$ and

  $A \uplus B ::= \mathit{ifcond}(A.pre =:=, A.pre \times B.pre, A.post \times B.post \uparrow)$, where

  * $\mathit{ifcond}(\mathit{condition}, \mathit{precondition}, \mathit{postcondition})$ - it checks if the given condition is true. If it is true, then it creates a pre- postcondition pair, where the precondition is defined in the second and the postcondition is defined in the third argument.

  * $=:=$ checks if two given conditions fit or not.

  * $\times$ merges the given conditions. If there is an $\uparrow$ in one side of the merging it means to use that condition as a prior one – if there is a header information, which appears in both conditions, the condition with the $\uparrow$ will be stored.

The rules The rules are based on the structure of the program, therefore the calculation is deterministic. On the sides of the rules, we can see two types of possible calculations. In the rules, there is the form $H, n \vdash S$, where H is a *ConditionState*, n is the name of the processed component and S is the *Program* that it processes. In one case, these rules can be recursively rewritten by the same form of expression – like in Rules 2, 6 – or in the other case it can be rewritten with a *ConditionState* – as in Rules 1, 3, 4, 5, 7, 8.

Rule 1 shows that the *Skip* does nothing, therefore the result will be the calculated condition state.

Rule 2 defines the processing of the sequence construct with composition – first, it processes the $S1$ program, and from the result of this calculation it processes the $S2$ program.

Rule 3 describes the assignment. Here we can see a side condition of the rule, which says to check if the identification of the expression *expr* – right side – and the header of the e – left side of the assignment – is valid. The side condition in this usage means, it will continue the calculation with those pairs of conditions, which are correct based on the side condition – in the pairs of conditions, where the side-condition is false, the process shows an error case. If the condition is true

$$1.\ \frac{H}{H, n \vdash Skip}$$

$$2.\ \frac{(H, n \vdash S1) \vdash S2}{H, n \vdash S1; S2}$$

$$3.\ \frac{\begin{gathered}H[H.n || H.n[pre \to Valid(expr.newIds, e.newIds.header),\\ post \to Valid(e.Ids, expr.newIds)]]\end{gathered}}{H, n \vdash e = expr}$$

$$\text{if } V(expr.oldIds) \& V(e.oldIds.header)$$

$$4.\ \frac{H[H.n || H.n[post \to \{Valid(h), Invalid(h.fields)\}]]}{H, n \vdash h.setValid()}$$

$$5.\ \frac{H[H.n || H.n[post \to Invalid(h, h.fields)]]}{H, n \vdash h.setInValid()}$$

$$6.\ \frac{(H[C(H.n, cond)]), n \vdash S1 \cup (H[C(H.n, !cond)]), n \vdash S2}{H, n \vdash \text{if } (cond)\ S1 \text{ else } S2}$$

$$7.\ \frac{H[H.n || \bigcup_{i=1}^{n} (H.n[pre \to Valid(k.Ids), post \to Valid(k.Ids)] \uplus a_i)]}{H, n \vdash keys : k\ \ actions : \{a_1, ..., a_n\}}$$

$$8.\ \frac{H[H.n || (H.n\ \dot{\cup}\ H.table\_name)]}{H, n \vdash table\_name.apply()}$$

Figure 5: Definition of $\vdash$ in a subset of the language

then it refreshes the condition state of the processed component by giving the new identifications with a valid property to the precondition and sets them valid in the postcondition too – because this command has not changed them – and sets the left side of the assignment to valid too – because it has just got a new value.

Rule 4 and 5 describe the validity setting. These functions set the validity of the header to valid or invalid – depends on the called function *setValid*() or *setInvalid*() – and set all of the fields to uninitialized.

Rule 6 gives the calculation of the branches. It needs to work with two different execution path – one where the condition is true, and one where it is false. If we check the actual condition state, it may contain several pairs of pre- and post-condition, therefore it describes both of the possible executions. In this rule, the calculation divides into two ways, these are calculated separately and their result will be joint.

Rule 7 calculates the conditions for the tables. It is important for the tables to read valid headers and initialized fields as keys. Although when this rule is used, we process a fresh component, therefore it does not need to use any side-condition, because this condition will be described in the precondition. First, it creates the precondition, which contains the condition of valid keys, and then it uses the precalculated conditions of called actions and merges them into several pairs of conditions. It checks if the preconditions of the actions fit the precondition of the table, and stores the merged conditions – uses the merged preconditions and the merged postconditions, where the postconditions of the actions are the stronger.

Rule 8 describes a table calling. In this case, the conditions for of the table has already been calculated, therefore it only needs to merge them to the actual conditions. In this merge, it has to check if the actual postcondition and the precondition of the table fit, and when it is correct it can merge them – uses the actual preconditions and merge the postconditions, where the postcondition of the table is the stronger.

4.2 Parser/Deparser Analyzer

The Parser/Deparser Analyzer is based on our previous paper [13]. It uses the parser to describe the main preconditions and the deparser to defines the main postconditions of the whole program. Therefore it describes the possible header information of the input and the output packets. The result of this is not pairs of conditions, but sets of pre- and postconditions. These conditions are not connected – are not in pairs –, they describe that what kind of initial states we would like to start our program and which final states are proper for the end.

There can be more main pre- and postconditions because parser and deparser can contain branches. In the main postconditions besides the conditions from the branches, there is another unique condition, which describes the case, when the packet is dropped. In this case, the *drop* is valid, and the validity of the other header information is irrelevant.

4.3 Final Checker

The task of the Final Checker to verify if the expected main pre- and postconditions of the program – which come from the Parser/Deparser Analyzer – and the actual pre- and postconditions fit well to each other. For this calculation we can generate a formula – Figure 6 from the calculated conditions and check if the formula is valid.

$$\begin{aligned}&\forall Pr \in MainPre,\ \forall p \in \mathit{pipeline\_condition}\,,\\ &\exists Po \in MainPost :\ ((Pr \supset p.pre) \wedge (p.post \supset Po))\end{aligned}$$

Figure 6: Formula for verification

It checks if the program from every initial state – which are states, where the precondition is true – there is a final state – which are states, where the postcondition is true – to reach. Therefore it checks for all precondition if they have at least one postcondition to fit based on the condition of the components. These components describe the pre- and postconditions for the correct working of the component. It checks, if any of the preconditions of the biggest component – in this version, it is the main ingress control function – is correct based on the main precondition. If there is one, then it checks the postcondition of it and checks the correctness of the actual postcondition.

If there is an actual precondition, which does not fit the main precondition, it means, there is an execution path, which will not work properly. If there is an actual precondition, which fits a main precondition, but the postconditions are not fitting, it means, during the calculation of that execution path, there will not be any problem, but the final state will not be correct for the specification of the program – in this case, the deparser of the program.

4.4 Error detection cases

The method can detect error cases in four different ways.

Condition Calculator It can detect errors, those are caused by the code of a component or can detect if the components are not matching. Three examples can be seen in Figure 7.

In the left one, *act1* and *act2* contains inside errors, which come from the bad usage of the method *setValid*() and *setInvalid*(). The side-effect of both methods is to change the uninitialized fields of the header. Therefore in the first case, when it would like to read the right side of the assignment, it will get a valid header and uninitialized field, so the read value is unknown. In the second case, on the left side of the assignment, it would like to write a field of an invalid header, which is an error case too. Both of them will be detected when it uses Rule 3, when the side-condition will be false in the process of the assignment.

On the right side of Figure 7, there is an example of the components, which are not matching. There are two components: the table *t* and a control function. The precondition of the table *t* is: $Valid(ipv4, ipv4.dstAddr)$, because it reads the *ipv4.dstAddr* field, so it need a valid header and initialized field. Although the true case of the branch says that the ipv4 is invalid – the postcondition of the control function stores it. Therefore when it uses Rule 8, it will not be able to merge the conditions, because the precondition of the table and the postcondition of the control function will not fit.

Final Checker The other case, when errors can be detected, is the process of the Final Checker. It can detect if the specification of the program does not fit the actual conditions i.e. starting the program from any initial state – where the main

```
action act1 () {
   hdr.ipv4.setValid();
   hdr.ipv4.srcAddr = hdr.ipv4.dstAddr();
}

action act2 () {
   hdr.ipv4.setInValid();
   hdr.ipv4.srcAddr = hdr.ethernet.dstAddr();
}
```

```
table t {
 key = { hdr.ipv4.dstAddr : exact; }
 ...
}

apply{
 if (!hdr.ipv4.isValid(()) {
   t.apply() }
}
```

Figure 7: Error cases in Condition Calculation

precondition is true – it can reach one of the final states – where the postcondition is true. In this phase, there can be two types of errors.

The first case is when the preconditions – of the main and the actual conditions – are matching, but there is no main postcondition, which fits the actual postcondition. In this case, although the program can start the execution from a good state, it will reach a state, which is not expected by the deparser – the calculated header information will be not correct for the program.

The second case is when there is no matching actual precondition for the main precondition. This means that starting from that input header information, the program will reach an error, the execution will be incorrect.

5 Case study

Figure 1 is an example P4 program which will be used to show an example usage of the method.

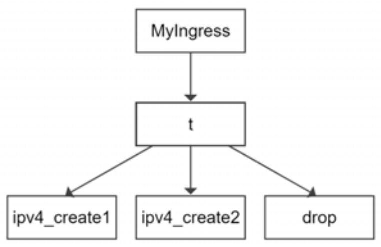


Figure 8: Call graph of the example program

Pipeline Analyzer First it creates the call graph. The graph of the example can be seen in Figure 8. There is only an ingress pipeline – *MyIngress* –, which calls one table – *t*– , which can call one of the three actions – *ipv4_create1*, *ipv4_create2* and *drop*. Based on the graph it can create the following list to define the order of the processing:
$[ipv4\_create1, ipv4\_create2, drop, t, MyIngress]$.

```
{ MyIngress: [(
        pre:  { valid: [], invalid: []},
        post: { valid: [], invalid: [drop]})],
  t: [(
        pre:  { valid: [], invalid: []},
        post: { valid: [], invalid: [drop]})],
  ipv4_create1: [(
        pre:  { valid: [], invalid: []},
        post: { valid: [], invalid: [drop]})],
  ipv4_create2: [(
        pre:  { valid: [], invalid: []},
        post: { valid: [], invalid: [drop]})],
  drop: [(
        pre:  { valid: [], invalid: []},
        post: { valid: [], invalid: [drop]})]}
```

Figure 9: Initial ConditionState

Based on the order of the list, the Condition Calculator starts the calculation from the expression, which is in Figure 10, where *Empty* is an initial *ConditionState*, which can be seen in Figure 9, which is mentioned in section 4.1.

$$(((\text{Empty}, \text{"ipv4\_create1"} \vdash \text{ipv4\_create1}), \text{"ipv4\_create2"} \vdash \text{ipv4\_create2}),$$
$$\text{"drop"} \vdash \text{drop}), \text{"t"} \vdash \text{t}), \text{"MyIngress"} \vdash \text{MyIngress}$$

Figure 10: Initial expression

Condition calculator The action *ipv4_create1* consists of sequences – Rule 2 – of two assignments – Rule 3 – and one setting of validity – Rule 4. Therefore there will be three rewritings of the conditions. When it starts the processing, there is only one pre- and postcondition pair, where the only condition says that the *drop* is invalid. The process of the first assignment – $hdr.ipv4.srcAddr = hdr.ethernet.dstAddr$ – does not contain any side-condition, because it has just started the calculation. Therefore it will add the *ethernet*, *ethernet.dstAddr* and *ipv4* to the Valid conditions in both of the pre- and postcondition – because it would read the right side, and it can only give value to a field of a valid header in the left side of the assignment. During the calculation, the postcondition describes the actual state. Therefore we will try to continue the process with the second assignment – $hdr.ipv4.dstAddr = dstAddr$. The side-condition checks, if the header of the written field is valid – *ipv4* is valid in the actual state or in the postcondition –, and if the read expression is valid – *dstAddr* is the parameter of the action, therefore the side-condition is not working with it. The validity is correct, so it adds the *ipv4.dstAddr* to the Valid side of the postcondition – because after the assignment, its value will be initialized. After the assignments, the program tries to set the validity of the header *ipv4* to valid. It changes only the postcondition by make all of the fields of *ipv4* – *ipv4.srcAddr*, *ipv4.dstAddr* and *ipv4.ttl* – invalid.

The calculation of the other actions similar to it, their results can be seen in Figure 11.

```
ipv4_create1: [(
       pre: {
         valid: [ethernet, ethernet.dstAddr, ipv4],
           invalid: []},
       post: {
         valid: [ipv4, ethernet, ethernet.dstAddr],
         invalid: [ipv4.ttl, ipv4.srcAddr, ipv4.dstAddr, drop]})],
ipv4_create2: [(
       pre: {
         valid: [ethernet, ethernet.dstAddr],
         invalid: []},
       post: {
         valid: [ipv4, ipv4.srcAddr, ipv4.dstAddr,
                 ethernet, ethernet.dstAddr, drop],
         invalid: [ipv4.ttl]})],
drop: [(
       pre: {
         valid: [ethernet, ethernet.dstAddr],
         invalid: []},
       post: {
         valid: [drop, ethernet, ethernet.dstAddr],
         invalid: []})],
```

Figure 11: Conditions of the actions

After the actions, it processes the table *t*. It has one key, so first, it sets its header *ethernet* and field *ethernet.dstAddr* to Valid, as a precondition, because the program would like to read them – and of course, as a postcondition too, because that shows the actual condition. Then it checks and merges its conditions and the conditions of the actions together. Therefore there will be three pairs of conditions – because of the three actions. The precondition of action *ipv4_create1* matches with the newly defined precondition of the table, therefore it can create the merged pre- and postcondition, where the condition of the action is the stronger one – in this case there is no conflict, so the condition pair of the action will be one of the table conditions.

The process of the other actions is similar. Their preconditions fit the precondition of the table, therefore it can merge their conditions. The conditions of the ingress *MyIngress* are the same as the table because it just simply calls it. The conditions of the table and ingress pipeline can be seen in Figure 12.

Parser/Deparser Analyzer The specification of the program is calculated from the parser and deparser. Both the parser and the deparser are really simple. None of them contains any branch, therefore there is only one possible precondition and two postconditions – Figure 13.

```
1.   [(
2.       pre: {
3.         valid: [ethernet, ethernet.dstAddr, ipv4],
4.         invalid: []},
5.       post: {
6.         valid: [ipv4, ethernet, ethernet.dstAddr],
7.         invalid: [ipv4.ttl, ipv4.srcAddr, ipv4.dstAddr, drop]}),(
8.       pre: {
9.         valid: [ethernet, ethernet.dstAddr],
10.        invalid: []},
11.      post: {
12.        valid: [ipv4, ipv4.srcAddr, ipv4.dstAddr,
13.                ethernet, ethernet.dstAddr],
14.        invalid: [ipv4.ttl, drop]}),(
15.      pre: {
16.        valid: [ethernet, ethernet.dstAddr],
17.        invalid: []},
18.      post: {
19.        valid: [ethernet, ethernet.dstAddr, drop],
20.        invalid: []})]
```

Figure 12: Conditions of table t and ingress pipeline *MyIngress*

The precondition describes the case when the parser extracts only the *ethernet* header, therefore this header and all of its fields will be valid, and the other header *ipv4* and its fields will be invalid.

There can be two postconditions. One, which is defined by the deparser, where the packet is not dropped and both of the headers *ethernet* and *ipv4* and all of their fields are valid – because the program will use their value for the new packet. The other case is when the packet is dropped.

```
Pre: [{
  Valid: [ethernet, ethernet.allFields() ],
  InValid: [ipv4, ipv4.allFields(), drop ]}]

Post: [{
  Valid: [ethernet, ethernet.allFields(), ipv4, ipv4.allFields() ],
  InValid: [drop] },

  { Valid: [drop], InValid: [] }]
```

Figure 13: Specification of the program

Final Checker There is only one main precondition, therefore it needs to check if the possible actual preconditions – the preconditions of the ingress pipeline *MyIngress* – fit, then their postcondition fits one of the main postconditions.

The main precondition assures only valid *ethernet* header information, although the first condition pair – rows 2-7. Figure 12 – of the actual condition needs a valid *ipv4* header too. The first part of the formula is false, therefore the method can detect an error, which shows that this execution path of the program will not be able to execute well, because of the invalid *ipv4* header.

In the second condition pair – rows 8-14. Figure 12–, the precondition fit to the main precondition, therefore it needs to check if the actual postcondition assures one of the main postconditions. The actual condition describes an execution when the packet was not dropped, therefore only the first main postcondition can be good, but we can see that in the actual postcondition the *ipv4.ttl* field is invalid, but the main postcondition needs a valid one. It is another error case, which is detected and it shows the case when the execution is correct, but the final header information which is calculated does not match with the expected result by the deparser.

In the third condition pair – rows 15-20. Figure 12, the preconditions fit, and this is the case when the packet was dropped during the execution, therefore its postcondition will ensure the second main postcondition. This path shows a correct execution path.

6 Evaluation

We created a prototype for the Condition Calculator. The critical part of the implementation is the calls of the tables because this is the step when more and more condition states can be defined. Therefore Figure 14 shows the runtime based on different numbers of table calls. During the measurement only one table was used, which has 5 action calls - one *NoAction*, which is like a *skip* program; one *drop* action, which only signs the package to drop it; and three others, all of them contain sequences of simple assignments.

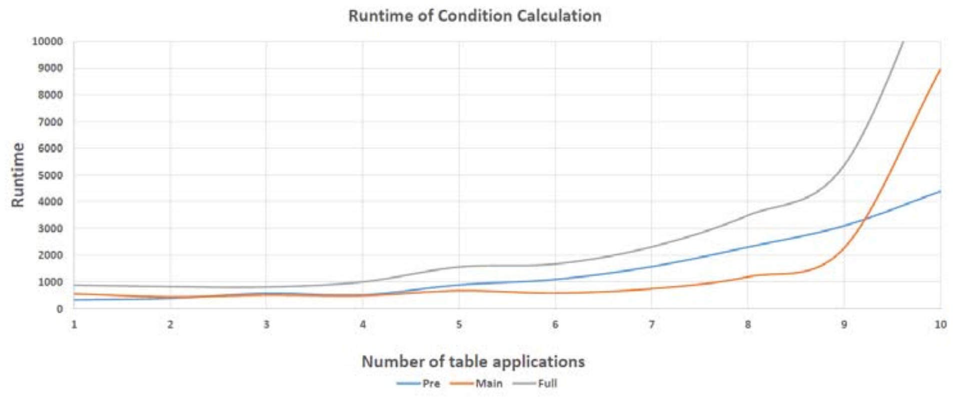


Figure 14: Runtime based on different number of table calls

In the graph of Figure 14, the vertical information is the number of table applications, and the horizontal data is the runtime in milliseconds. We can see three different lines because the implementation of the calculator is separated into two parts: a pre-calculator, which only prepares the obtained information; and the main condition calculator, which uses the prepared data to create the conditions. In the graph, the three different lines show the runtime of the pre-calculator (blue), the main calculator (orange), and the full time of them (grey). This graph shows the worst cases when all of the calculated conditions are different, therefore the number of the conditions is equal to the number of the actions to the power of the number of the table applications i.e. in our table there were 5 actions, and when we applied this table five times, then we could get 3125 conditions. However, we can see that, when we used 10 table applications, the runtime was only 13 seconds.

This presented data only give the worst runtime of this calculation, but when we use a normal P4 code then most of the conditions are the same, do the program will not work with this number of conditions.

7 Summary

In this paper, we introduced a formal method to detect errors in P4 programs by checking the pre- and postconditions. It has three main analyzers: the Parser/Deparser Analyzer, the Pipeline Analyzer, and the Final Checker. The method only works with the P4 source, therefore it extracts the specification of the program from the parser and deparser of it – by the Parser/Deparser Analyzer. The Pipeline Analyzer checks the definition of the pipeline, and calculated condition pairs for the different components of it – actions, match-action tables, and control functions. It can detect four different types of errors, which can be caused by the incorrect usage of invalid headers and uninitialized fields or components, which do not match.

Future Work The method is a base of a full solution for P4 verification. Until this goal, we need to extend the checked subset of the P4 language, therefore we will be able to work with more programs.
When both of our approach is ready – this, component-based and our previous idea [13] –, then it could be an interesting work to check which one is better in the specified error detections, and combine them for a more optimized solution.
P4 programs can use different architectures. In this approach, we only showed the Protocol Independent Switch Architecture, which use a single pipeline, but there are other architecture models – for example Portable Switch Architecture [3] which can work with more pipelines. Our solution will be able to used as a part of a research, which checks if a P4 program is correct for a given architecture, while checking the correctness of a pipeline.

Acknowledgement

We thank Dániel Lukács (PhD Student at ELTE) for the preparation of the base of the tool, therefore we could integrate our implementation of this verification idea into it!

References

[1] $P4_{16}$ Language Specification. `https://p4.org/p4-spec/docs/P4-16-v1.1.0-spec.pdf`, 2018. Online; accessed 7 December 2020.

[2] The P4 Language Specification. `https://p4.org/p4-spec/p4-14/v1.0.5/tex/p4.pdf`, 2018. Online; accessed 7 December 2020.

[3] $P4_{16}$ Portable Switch Architecture (PSA). `https://p4.org/p4-spec/docs/PSA.html`, 2020. Online; accessed 7 December 2020.

[4] Birnfeld, K., da Silva, D. C., Cordeiro, W., and de França, B. B. N. P4 switch code data flow analysis: Towards stronger verification of forwarding plane software. In *NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium*, pages 1–8, 2020. DOI: `10.1109/NOMS47738.2020.9110307`.

[5] Bosshart, Pat, Daly, Dan, Gibb, Glen, Izzard, Martin, McKeown, Nick, Rexford, Jennifer, Schlesinger, Cole, Talayco, Dan, Vahdat, Amin, Varghese, George, and Walker, David. P4: Programming protocol-independent packet processors. *SIGCOMM Comput. Commun. Rev.*, 44(3):87–95, 2014. DOI: `10.1145/2656877.2656890`.

[6] Budiu, Mihai and Dodd, Chris. The p416 programming language. *SIGOPS Oper. Syst. Rev.*, 51(1):5–14, September 2017. DOI: `10.1145/3139645.3139648`.

[7] Eichholz, Matthias, Campbell, Eric, Foster, Nate, Salvaneschi, Guido, and Mezini, Mira. How to avoid making a billion-dollar mistake: Type-safe data plane programming with safep4. *CoRR*, abs/1906.07223, 2019.

[8] Freire, Lucas, Neves, Miguel, Leal, Lucas, Levchenko, Kirill, Schaeffer-Filho, Alberto, and Barcellos, Marinho. Uncovering Bugs in P4 Programs with Assertion-based Verification. In *Proceedings of the Symposium on SDN Research*, SOSR '18, pages 4:1–4:7, New York, NY, USA, 2018. ACM. DOI: `10.1145/3185467.3185499`.

[9] Liu, Jed, Hallahan, William, Schlesinger, Cole, Sharif, Milad, Lee, Jeongkeun, Soulé, Robert, Wang, Han, Caşcaval, Călin, McKeown, Nick, and Foster, Nate. P4v: Practical Verification for Programmable Data Planes. In *Proceedings of*

the 2018 Conference of the ACM Special Interest Group on Data Communication, SIGCOMM '18, pages 490–503, New York, NY, USA, 2018. ACM. DOI: `10.1145/3230543.3230582`.

[10] Shukla, Apoorv, Hudemann, Kevin Nico, Hecker, Artur, and Schmid, Stefan. Runtime verification of p4 switches with reinforcement learning. In *Proceedings of the 2019 Workshop on Network Meets AI & ML*, NetAI'19, page 1–7, New York, NY, USA, 2019. Association for Computing Machinery. DOI: `10.1145/3341216.3342206`.

[11] Stoenescu, Radu, Dumitrescu, Dragos, Popovici, Matei, Negreanu, Lorina, and Raiciu, Costin. Debugging P4 Programs with Vera. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '18, pages 518–532, New York, NY, USA, 2018. ACM. DOI: `10.1145/3230543.3230548`.

[12] Stoenescu, Radu, Popovici, Matei, Negreanu, Lorina, and Raiciu, Costin. Symnet: Scalable Symbolic Execution for Modern Networks. In *Proceedings of the 2016 ACM SIGCOMM Conference*, SIGCOMM '16, pages 314–327, New York, NY, USA, 2016. ACM. DOI: `10.1145/2934872.2934881`.

[13] Tóth, Gabriella and Tejfel, Máté. A formal method to detect possible p4 specific errors. In *Position Papers of the 2019 Federated Conference on Computer Science and Information Systems*, volume 19 of *Annals of Computer Science and Information Systems*, pages 49–56. PTI, 2019. DOI: `10.15439/2019F355`.

Acta Cybernetica **25** (2021) 555–573.

Footvector Representation of Curves and Surfaces*

Gábor Valasek[ab], Csaba Bálint[ac], and András Leitereg[ad]

Abstract

This paper proposes a foot mapping-based representation of curves and surfaces which is a geometric generalization of signed distance functions. We present a first-order characterization of the footvector mapping in terms of the differential geometric invariants of the represented shape and quantify the dependence of the spatial partial derivatives of the footvector mapping with respect to the principal curvatures at the footpoint. The practical applicability of foot mapping representations is highlighted by several fast iterative methods to compute the exact footvector mapping of the offset surface of constructive solid geometry (CSG) trees. The set operations for footpoint mappings are higher-order functions that map a tuple of functions to a single function, which poses a challenge for GPU implementations. We propose a code generation framework to overcome this that transforms CSG trees to the GLSL shader code.

Keywords: computer graphics, constructive solid geometry, signed distance function

1 Introduction

This paper introduces a foot mapping based representation of shapes. The footpoint is the closest point of the shape boundary to an arbitrary query position and the footvector is the displacement from the latter to the footpoint. The proposed representation leverages the footvector mapping of the encoded shape and it is a geometric generalization of signed distance functions.

We present the theoretical background of this approach in Section 3 and show how the differential geometric invariants at the footpoint govern the development of the footvector mapping in space. More specifically, we highlight the connection

*EFOP-3.6.3-VEKOP-16-2017-00001: Talent Management in Autonomous Vehicle Control Technologies — The Project is supported by the Hungarian Government and co-financed by the European Social Fund. Supported by the ÚNKP-20-3 New National Excellence Program of the Ministry for Innovation and Technology.

[a]Eötvös Loránd University, Budapest, Hungary
[b]E-mail: `valasek@inf.elte.hu`, ORCID: `0000-0002-0007-8647`
[c]E-mail: `csabix@inf.elte.hu`, ORCID: `0000-0002-5609-6449`
[d]E-mail: `leanil@inf.elte.hu`, ORCID: `0000-0003-0705-0214`

DOI: `10.14232/actacyb.290145`

between the first-order behaviour of the footvector mapping and the offsets of the shape and, as such, the principal curvatures. These results are presented for curves in the plane and surfaces in space.

Following the theoretical discussion, we introduce a method to create CSG models with precise offset operations using footvector mappings. In particular, we present two iterative algorithms on foot mappings in Section 4 and Section 5 to compute the footvector function of the intersection of two shapes. We discuss the robustness/speed trade-off between these in Section 1 where we apply these methods to the offset of intersections.

In general, implicit surface representations ($g : \mathbb{R}^3 \to \mathbb{R}$) usually describe 'inside' region as the set of negative values ($\{g < 0\} := \{\boldsymbol{x} \in \mathbb{R}^3 : g(\boldsymbol{x}) < 0$), and the 'outside' as the positive $\{g\rangle 0\} \subseteq \mathbb{R}^3$ region of space. Set operations can be defined with minimum and maximum operations on the argument implicit functions.

A special case of implicit representations are the signed distance functions (SDFs) that map the signed distance from the boundary to every point in space. They offer efficient real-time visualization with sphere tracing [7]. Unfortunately, they are not closed under the min/max representation of set operations, that is the minimum and maximum of SDF arguments may not be a precise SDF. Even though the shape of the surface and the convergence of sphere tracing are invariant under these operations, the loss of the exact SDF property is disadvantageous for the offset operation.

We define the offset surface as the set of points that are r distance away from the original surface. To take an offset of a surface given by a signed distance function, one must only subtract the offset radius [2]. However, if g_1 and $g_2 : \mathbb{R}^3 \to \mathbb{R}$ are SDFs, then the $r > 0$ offset of their intersection is

$$\boldsymbol{x} \mapsto \max(g_1(\boldsymbol{x}), g_2(\boldsymbol{x})) - r \iff \boldsymbol{x} \mapsto \max(g_1(\boldsymbol{x}) - r, g_2(\boldsymbol{x}) - r)$$

which is the same as taking their offsets and then their intersection. However, these operations are not supposed to be interchangeable.

For example, the positive offsets of spheres are larger spheres, so their intersection is an intersection of larger spheres with a sharp edge, yet, the offset of the original intersection set supposed to be a pill-shaped oval surface. This is a wrong result as these operations should not be interchangeable. The reason is that the intersection operation is imprecise near-surface edges and corners and the function values do not increase correctly. Our algorithm solves this issue allowing real offset operations on set operation results. Figure 1 demonstrates this key difference between signed distance functions and footvector mappings.

We extend the precision of analytic distance functions for foot mappings by defining the primitives, set operations, and offset on them. The set operations, however, do not operate on a single implicit function value, but operate on whole functions. This means that set operations on footvector mappings are higher order functions that also produce a function from the input functions:

$$\cap, \cup \in (\mathbb{R}^3 \to \mathbb{R}^4) \times (\mathbb{R}^3 \to \mathbb{R}^4) \to (\mathbb{R}^3 \to \mathbb{R}^4) \, .$$

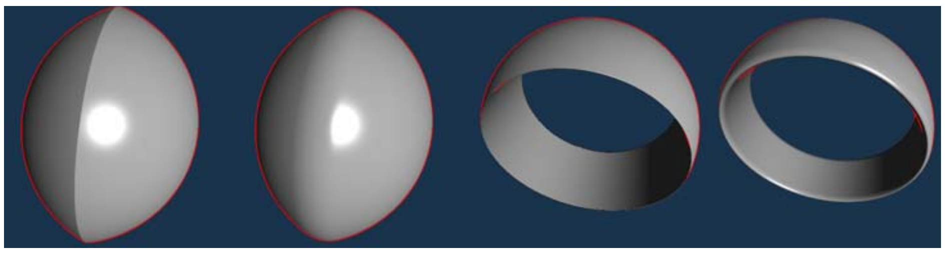

(a) SDF (b) Foot mapping (c) SDF (d) Foot mapping

Figure 1: The offset of the intersection of two spheres and the offset of a cylinder subtracted from a sphere. The offset operation should leave the surface smooth for which the distance representation is inadequate.

Our representation maps to $\mathbb{R}^4$ because besides the footvector we also have to encode whether the point is inside or outside the geometry. The footvector is assigned to the first three coordinates, and the signed distance value is in the fourth.

There are many other ways to compute the offset surface. Most of these operate on parametric representations, which may not be available in our case. For implicit representations, fast distance transforms are commonly used. These are efficient to execute, but the visualization is not direct because a grid of values must be computed and stored. Thus, these algorithms are limited in accuracy by the available memory. Yet, the resulting distance representation is fast, and offset surfaces can be created quickly making this a viable choice for e.g. text rendering in 2D. Three dimensional distance fields are even more expensive to store which motivated our grid-free approach.

Moreover, methods based on distance transform rely on a discretization of the distance function and marching the distance values by approximating the distance to the surface from the neighbouring values [5, 13] or use simplified proxy geometries to infer a more precise distance value [14]. The local computations introduce errors that can be amplified by set operations, making them worse than the min and max SDF operations. To increase precision and reduce memory usage, we devised iterative algorithms that compute the distance to the intersection set from any point $\boldsymbol{x} \in \mathbb{R}^3$ to the intersection object. In exchange, the presented algorithms require a different representation of the surfaces.

To achieve real-time direct visualization of such offset surfaces, the GPU is required. However, our set operations operate on functions, and we cannot simply pass functions as arguments in shader code, because GPUs only support inlineable functions. For this reason, we implemented a code generator that created the implementation for the footvector mapping from a given CSG tree. The higher-order set operations had to be implemented into the code generator to run our iteration on various argument footvector mappings. These results are presented in Section 6.

2 Preliminaries

From any sample point $\boldsymbol{x} \in \mathbb{R}^n$ a signed distance function (SDF) $g : \mathbb{R}^n \to \mathbb{R}$ is a continuous function that evte aluates to the signed Euclidean distance measured from the surface. That is

$$\big|g(\boldsymbol{x})\big| = d(\boldsymbol{x}, \{g = 0\}) \qquad (\forall\, \boldsymbol{x} \in \mathbb{R}^3),$$

where $d(\boldsymbol{x}, A)$ is the point-to-set distance, and $\{g = 0\}$ is the zero level-set. The dimension is $n = 2$ for the plane and $n = 3$ for 3D space. The sign encodes whether $\boldsymbol{x}$ is inside $\{g \leq 0\}$ or outside $\{g > 0\}$ which allows set-operations to be defined on SDFs. Let us take g_1 and g_2 signed distance functions, then according to [7],

$$\begin{aligned} d\big(\boldsymbol{x}, \{g_1 \leq 0\} \cup \{g_2 \leq 0\}\big) \;&\geq\; \big|\min\{g_1(\boldsymbol{x}), g_2(\boldsymbol{x})\}\big| \qquad (\forall\, \boldsymbol{x} \in \mathbb{R}^n)\ , \\ d\big(\boldsymbol{x}, \{g_1 \leq 0\} \cap \{g_2 \leq 0\}\big) \;&\geq\; \big|\max\{g_1(\boldsymbol{x}), g_2(\boldsymbol{x})\}\big| \qquad (\forall\, \boldsymbol{x} \in \mathbb{R}^n)\ . \end{aligned} \tag{1}$$

The $\min(g_1, g_2) = \boldsymbol{x} \mapsto \min\{g_1(\boldsymbol{x}), g_2(\boldsymbol{x})\}$ function is an implicit function of the union of $\{g_1 \leq 0\}$ and $\{g_2 \leq 0\}$ objects. The resulting function is a good lower-estimate of the distance on the inside of the union, whereas it is exact on the outside of the union. For this reason, many SDF representations use min and max operations to combine primitive geometries into complex scenes [6, 8]. However, the approximation is imprecise on the union for the min operation, and only exact within the intersection for the max intersection SDF approximation.

The precision can be quantified for any $g : \mathbb{R}^n \to \mathbb{R}$ SDF by comparing the real distance-to-surface value to that of the function:

$$q_g(\boldsymbol{x}) := \frac{\big|g(\boldsymbol{x})\big|}{d(\boldsymbol{x}, \{g = 0\})} \qquad (\forall\, \boldsymbol{x} \in \mathbb{R}^n) \tag{2}$$

Signed distance function estimates (SDFEs) are defined using the above local precision. If there exists a $c > 0$ global precision such that $0 < c \leq q_g(\boldsymbol{p}) \leq 1$, then $g : \mathbb{R}^n \to \mathbb{R}$ is a signed distance function estimate.

Distance representations can be directly ray-traced via various sphere tracing algorithms [1, 3, 7, 10]. The precision of the SDFE measures the slowdown of the sphere tracing algorithm; however, computing $q_{\max(g_1,g_2)}(\boldsymbol{x})$ for the intersection operation can be expensive.

The function $\boldsymbol{f} : \mathbb{R}^n \to \mathbb{R}^n$ is a footvector mapping if

1. $\boldsymbol{x} \mapsto \big\|\boldsymbol{f}(\boldsymbol{x})\big\|$ $(x \in \mathbb{R}^n)$ is a distance function
2. $\boldsymbol{f}(\boldsymbol{x} + \boldsymbol{f}(\boldsymbol{x})) = \boldsymbol{0}$ for all $\boldsymbol{x} \in \mathbb{R}^n$

This means that $\boldsymbol{f}$ returns a vector pointing to one of the closest points on the surface it defines. Thus $\boldsymbol{x} + \boldsymbol{f}(\boldsymbol{x})$ is the corresponding footpoint, but note that the $\boldsymbol{f}$ function is not unique for a given shape. This is because some points will have more then one closest points from the geometry, each providing an allowed return value for the $\boldsymbol{f}$ function.

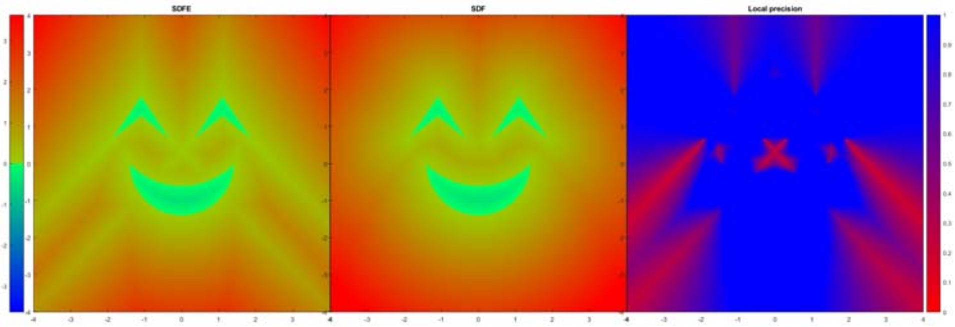


Figure 2: Left: 2D SDFE obtained through min and max set operations using transformations of a half-plane (line) and a circle primitive. Local precision is the ratio of the SDFE (left) and the exact SDF (middle) is displayed on the right signaling the slowdown of sphere tracing. Our footpoint iteration produced the middle image.

Similarly to SDFs, the footpoint representation requres volumetric, i.e. solid geometric information for the set-operations to be defined. Let us assume we can decide if $\boldsymbol{x}$ is inside $\boldsymbol{x} \in G \subseteq \mathbb{R}^n$ closed set or outside $\boldsymbol{x} \in \mathbb{R}^n \setminus G$, where the boundary set is $\partial G = \{\boldsymbol{x} \in \mathbb{R}^n \mid \|\boldsymbol{f}(\boldsymbol{x})\| = 0\} \subseteq G$. For example, having the corresponding signed distance function $g : \mathbb{R}^n \to \mathbb{R}$ such that $G = \{g \leq 0\}$ and $\{\boldsymbol{f} = 0\} = \{g = 0\} = \partial G$ will allow the set operations.

3 Differential geometry and footpoint mappings

In this section, we investigate the relation between the derivatives of the footpoint mapping and the local differential geometry at the footpoint.

Let $G \subseteq \mathbb{R}^n$ denote a geometry of interest in either the plane or space, i.e. $n = 2, 3$, and $\operatorname{int} G \subseteq G$ its interior points. Let us assume that its boundary, ∂G, is a sufficiently smooth set in the sense of geometric continuity, that is, it can be covered by local parametrizations of the desired parametric smoothness [4].

The stationary condition of distance [12] states that the vector from the query position $\boldsymbol{x}$ to the footpoint $\boldsymbol{x}^*$ is perpendicular to the tangent entity of the shape, which is the tangent line in plane and the tangent plane in space. In other words, this means that the footvector mapping $\boldsymbol{f} = \boldsymbol{x}^* - \boldsymbol{x}$ is parallel to the curve or surface normal $\boldsymbol{n}^*$ at the footpoint $\boldsymbol{x}^*$.

Let $\boldsymbol{c}(t) : \mathbb{R} \to \mathbb{R}^2$ and $\boldsymbol{s}(u, v) : \mathbb{R}^2 \to \mathbb{R}^3$ be the parametric representation of ∂G in the plane and in space, respectively. In both cases, we denote any suitable implicit representation of G by $g : \mathbb{R}^n \to \mathbb{R}$ $(n = 2, 3)$. Using these notations, the perpendicularity of the footvector to ∂G is expressed as:

representation	plane	space
parametric	$\boldsymbol{f} \cdot \boldsymbol{c}' = 0$	$\begin{cases} \boldsymbol{f} \cdot \boldsymbol{s}_u = 0 \\ \boldsymbol{f} \cdot \boldsymbol{s}_v = 0 \end{cases}$
implicit	$\boldsymbol{f} \times \nabla g = \boldsymbol{0}$	$\boldsymbol{f} \times (\boldsymbol{s}_u \times \boldsymbol{s}_v) = \boldsymbol{0}$

where $\boldsymbol{c}'$ denotes differentiation with respect to the particular curve parameter t, $\boldsymbol{s}_u = \partial_u \boldsymbol{s}$, and $\boldsymbol{s}_v = \partial_v \boldsymbol{s}$.

A more intuitive characterization of the footvector mapping comes from observing the behaviour of its first degree Taylor expansion. Using $\boldsymbol{x} = [x, y, z]^T, \boldsymbol{x}_0 = [x_0, y_0, z_0]^T$, this is formally

$$\boldsymbol{f}(\boldsymbol{x}) \approx \boldsymbol{f}(\boldsymbol{x}_0) + (x - x_0)\partial_1 \boldsymbol{f}(\boldsymbol{x}_0) + (y - y_0)\partial_2 \boldsymbol{f}(\boldsymbol{x}_0) + (z - z_0)\partial_3 \boldsymbol{f}(\boldsymbol{x}_0) \ .$$

Our goal is to quantify the interplay between the footpoint mapping and the differential geometry at the footpoint. The most natural setting for the study of the local differential geometry of shapes is the parametric representation. As such, in the following two subsections we derive our results for parametric curves in the plane and parametric surfaces in space.

3.1 Derivatives of the footpoint mapping: $\mathbb{R}^2$

In the case of parametric plane curves, the footvector mapping $\boldsymbol{f} : \mathbb{R}^2 \to \mathbb{R}^2$ is decomposed as

$$\begin{aligned} \boldsymbol{f}(\boldsymbol{x}) &= \boldsymbol{c}(t(\boldsymbol{x})) - \boldsymbol{x} \\ &= \boldsymbol{c} \circ t - \mathbf{id} \ , \end{aligned} \tag{3}$$

where $\boldsymbol{c} : \mathbb{R} \to \mathbb{R}^2$ is a parametrization of the boundary, $\boldsymbol{x} = [x, y]^T$ is the query position, and $t : \mathbb{R}^2 \to \mathbb{R}$ is the footparameter mapping that assigns the parameter value corresponding to the closest curve point to $\boldsymbol{x}$. In the second equation, $\mathbf{id}$ denotes the identity mapping of $\mathbb{R}^n$, i.e. $\partial_1 \mathbf{id} = \boldsymbol{e}_1$, $\partial_2 \mathbf{id} = \boldsymbol{e}_2$, where $\boldsymbol{e}_i$ are the canonical basis vectors of $\mathbb{R}^2$.

Let $i \in \{1, 2\}$ denote an arbitrary coordinate axis and let us take the partial derivative of $\boldsymbol{f} \cdot \boldsymbol{c}' = 0$ with respect to i as

$$\begin{aligned} 0 &= \partial_i \Big(\big(\boldsymbol{c} \circ t - \mathbf{id}\big) \cdot \boldsymbol{c}' \circ t \Big) \\ &= \partial_i \Big(\boldsymbol{c} \circ t - \mathbf{id} \Big) \cdot \boldsymbol{c}' \circ t + \Big(\boldsymbol{c} \circ t - \mathbf{id} \Big) \cdot \partial_i \boldsymbol{c}' \circ t \\ &= \Big(\boldsymbol{c}' \circ t \cdot \partial_i t - \boldsymbol{e}_i \Big) \cdot \boldsymbol{c}' \circ t + \Big(\boldsymbol{c} \circ t - \mathbf{id} \Big) \cdot \boldsymbol{c}'' \circ t \cdot \partial_i t \end{aligned}$$

After rearrangement, the partial derivative of the footparameter mapping is

$$\partial_i t = \frac{\boldsymbol{c}' \circ t \cdot \boldsymbol{e}_i}{\boldsymbol{c}' \circ t \cdot \boldsymbol{c}' \circ t + \boldsymbol{f} \cdot \boldsymbol{c}'' \circ t} \ .$$

Omitting the point of evaluation, the gradient of the footparameter mapping is

$$\nabla t = \frac{1}{\boldsymbol{c}' \cdot \boldsymbol{c}' + \boldsymbol{f} \cdot \boldsymbol{c}''} \boldsymbol{c}'$$

proving the simple intuition that the largest change in the footparameter happens when the query position is displaced parallel to the tangent line at the footpoint. More importantly, this equation also quantifies the amount of the largest change.

To give a geometric interpretation to this, let us consider the $\widehat{\boldsymbol{c}} : [0, L] \to \mathbb{R}^2$ arc-length parametrization of the boundary. This is done without loss of generality. The gradient of the footparameter mapping is then

$$\nabla t = \frac{1}{1 + \boldsymbol{f} \cdot \kappa \widehat{\boldsymbol{n}}} \widehat{\boldsymbol{t}} \ ,$$

where $\widehat{\boldsymbol{t}}, \widehat{\boldsymbol{n}}$ denote the Frenet frame unit tangent and normal vectors of the curve and κ is its curvature function. Noting that $\boldsymbol{f} \parallel \widehat{\boldsymbol{n}}$ and thus can be expressed as $\boldsymbol{f} = d \cdot \widehat{\boldsymbol{n}}$ for some $d \in \mathbb{R}$, the above becomes

$$\nabla t = \frac{1}{1 + d\kappa} \widehat{\boldsymbol{t}} \tag{4}$$

that is, a unit displacement of the query position along $\widehat{\boldsymbol{t}}$ results in a $\frac{1}{1+d\kappa}$ displacement in the footpoint parameter.

Equation (4) is also remarkable in the sense that it shows that the change in the footpoint parameter is the reciprocal of the change of the parametric speed of the original boundary curve offset by d. In Kallay's terms [9], it is a first order pull-back onto the parametrization of the offset curve.

To translate our results on the footparameter mapping to the footpoint mapping, let us consider the first degree Taylor expansion of Eq. (3) as

$$\boldsymbol{f}(\boldsymbol{x}) \approx \boldsymbol{f}(\boldsymbol{x}_0) + (x - x_0)\partial_x \boldsymbol{f}(\boldsymbol{x}_0) + (y - y_0)\partial_y \boldsymbol{f}(\boldsymbol{x}_0) \ .$$

From Eq. (3), the partial derivatives of $\boldsymbol{f}$ are

$$\partial_i \boldsymbol{f} = \boldsymbol{c}' \circ t \cdot \partial_i t - \boldsymbol{e}_i \ ,$$

that is,

$$\begin{aligned} \boldsymbol{f}(\boldsymbol{x}) &\approx \boldsymbol{f}(\boldsymbol{x}_0) + (x - x_0)\big(\boldsymbol{c}'(t(\boldsymbol{x}_0))\partial_1 t(\boldsymbol{x}_0) - \boldsymbol{e}_1\big) + (y - y_0)\big(\boldsymbol{c}'(t(\boldsymbol{x}_0))\partial_2 t(\boldsymbol{x}_0) - \boldsymbol{e}_2\big) \\ &= \boldsymbol{f}(\boldsymbol{x}_0) - (\boldsymbol{x} - \boldsymbol{x}_0) + (x - x_0)\boldsymbol{c}'(t(\boldsymbol{x}_0))\partial_1 t(\boldsymbol{x}_0) + (y - y_0)\boldsymbol{c}'(t(\boldsymbol{x}_0))\partial_2 t(\boldsymbol{x}_0) \end{aligned}$$

This is written more succinctly omitting the evaluation points, denoting $\boldsymbol{f}_0 = \boldsymbol{f}(\boldsymbol{x}_0)$, $\nabla t_0 = \nabla t(\boldsymbol{x}_0)$, $\boldsymbol{c}'_0 = \boldsymbol{c}'(t(\boldsymbol{x}_0))$, and substituting Eq. (3) as

$$\boldsymbol{f}(\boldsymbol{x}) \approx \boldsymbol{f}_0 - (\boldsymbol{x} - \boldsymbol{x}_0) + \big((\boldsymbol{x} - \boldsymbol{x}_0) \cdot \nabla t_0\big)\boldsymbol{c}'_0 \ ,$$

or, using arc-length parametrization,

$$\boldsymbol{f}(\boldsymbol{x}) \approx \boldsymbol{f}_0 - (\boldsymbol{x} - \boldsymbol{x}_0) + \frac{(\boldsymbol{x} - \boldsymbol{x}_0) \cdot \widehat{\boldsymbol{t}}_0}{1 + d\kappa_0} \widehat{\boldsymbol{t}}_0 \ .$$

According to this, there is no change in the footpoint if $\boldsymbol{x} - \boldsymbol{x}_0$ is perpendicular to $\widehat{\boldsymbol{t}}$, but this only holds as long as the footpoint mapping is a function, that is, the footpoint is unique. As soon as we reach the cut locus, i.e. a point in the plane that has multiple closest points, the footpoint mapping becomes discontinuous and the footpoint can change arbitrarily along the circumference of the unbounding circle.

3.2 Derivatives of the footpoint mapping: $\mathbb{R}^3$

Now, let us consider the case when the geometry is a volume and $\boldsymbol{s} : \mathbb{R}^2 \to \mathbb{R}^3$ is the parametric representation of its boundary surface. The stationary condition of the footpoint is written as

$$\boldsymbol{f} \cdot \boldsymbol{s}_u = 0 \tag{5}$$

$$\boldsymbol{f} \cdot \boldsymbol{s}_v = 0 \ , \tag{6}$$

where the footpoint mapping is decomposed as

$$\boldsymbol{f} = \boldsymbol{s} \circ \boldsymbol{u} - \mathbf{id} \ , \tag{7}$$

that is,

$$\boldsymbol{f}(x, y, z) = \boldsymbol{s}(u(x, y, z), v(x, y, z))$$

with the two footparemeter mappings $u, v : \mathbb{R}^3 \to \mathbb{R}$, $\boldsymbol{u} = (u, v)$.

Differentiating Eq. (5) with respect to an arbitrary coordinate axis $i \in \{1, 2, 3\}$ yields

$$\begin{aligned}\partial_i(\boldsymbol{f} \cdot \boldsymbol{s}_u \circ \boldsymbol{u}) &= \partial_i\big((\boldsymbol{s} \circ \boldsymbol{u} - \mathbf{id}) \cdot \boldsymbol{s}_u \circ \boldsymbol{u}\big) \\ &= (\boldsymbol{s}_u \circ \boldsymbol{u} \cdot \partial_i u + \boldsymbol{s}_v \circ u \cdot \partial_i v - \boldsymbol{e}_i) \cdot \boldsymbol{s}_u \circ \boldsymbol{u} \\ &\quad + \boldsymbol{f} \cdot \big(\boldsymbol{s}_{uu} \circ \boldsymbol{u} \cdot \partial_i u + \boldsymbol{s}_{uv} \circ \boldsymbol{u} \cdot \partial_i v\big)\end{aligned}$$

Similarly, differentiating Eq. (6) is

$$\begin{aligned}\partial_i(\boldsymbol{f} \cdot \boldsymbol{p}_v \circ \boldsymbol{u}) &= (\boldsymbol{s}_u \circ \boldsymbol{u} \cdot \partial_i u + \boldsymbol{s}_v \circ u \cdot \partial_i v - \boldsymbol{e}_i) \cdot \boldsymbol{s}_v \circ \boldsymbol{u} \\ &\quad + \boldsymbol{f} \cdot \big(\boldsymbol{s}_{uv} \circ \boldsymbol{u} \cdot \partial_i u + \boldsymbol{s}_{vv} \circ \boldsymbol{u} \cdot \partial_i v\big)\end{aligned}$$

As such, the partial derivatives of the footparameter mappings with respect to the coordinate axes of $\mathbb{R}^3$ are found from the following system of six linear equations in the unknowns $\partial_i u, \partial_i v$, $i \in \{1, 2, 3\}$:

$$\begin{aligned}(\boldsymbol{s}_u \circ \boldsymbol{u} \cdot \partial_i u + \boldsymbol{s}_v \circ u \cdot \partial_i v - \boldsymbol{e}_i) \cdot \boldsymbol{s}_u \circ \boldsymbol{u} + \boldsymbol{f} \cdot \big(\boldsymbol{s}_{uu} \circ \boldsymbol{u} \cdot \partial_i u + \boldsymbol{s}_{uv} \circ \boldsymbol{u} \cdot \partial_i v\big) = 0 \\ (\boldsymbol{s}_u \circ \boldsymbol{u} \cdot \partial_i u + \boldsymbol{s}_v \circ u \cdot \partial_i v - \boldsymbol{e}_i) \cdot \boldsymbol{s}_v \circ \boldsymbol{u} + \boldsymbol{f} \cdot \big(\boldsymbol{s}_{uv} \circ \boldsymbol{u} \cdot \partial_i u + \boldsymbol{s}_{vv} \circ \boldsymbol{u} \cdot \partial_i v\big) = 0\end{aligned}$$

Recalling that the footvector mappings are perpendicular to the tangent plane, we can make the substitution $\boldsymbol{f} = d \cdot \boldsymbol{m}$, where $d \in \mathbb{R}$ and $\boldsymbol{m}$ is the unit surface normal of $\boldsymbol{s}$, e.g. $\boldsymbol{m} = \frac{\boldsymbol{s}_u \times \boldsymbol{s}_v}{||\boldsymbol{s}_u \times \boldsymbol{s}_v||_2}$ at regular points of $\boldsymbol{s}$. The resulting system takes its final form as

$$\begin{aligned}\mathbb{E} \cdot \partial_i u + \mathbb{F} \cdot \partial_i v - \boldsymbol{e}_i \cdot \boldsymbol{s}_u \circ \boldsymbol{u} + d \cdot \mathbb{L} \cdot \partial_i u + d \cdot \mathbb{M} \cdot \partial_i v = 0 \\ \mathbb{F} \cdot \partial_i u + \mathbb{G} \cdot \partial_i v - \boldsymbol{e}_i \cdot \boldsymbol{s}_v \circ \boldsymbol{u} + d \cdot \mathbb{M} \cdot \partial_i u + d \cdot \mathbb{N} \cdot \partial_i v = 0\end{aligned} \tag{8}$$

using the first and second fundamental forms of

$$\mathbb{E} = \boldsymbol{s}_u \cdot \boldsymbol{s}_u \ , \ \mathbb{F} = \boldsymbol{s}_u \cdot \boldsymbol{s}_v \ , \ \mathbb{G} = \boldsymbol{s}_v \cdot \boldsymbol{s}_v \ ,$$
$$\mathbb{L} = \boldsymbol{s}_{uu} \cdot \boldsymbol{m} \ , \ \mathbb{M} = \boldsymbol{s}_{uv} \cdot \boldsymbol{m} \ , \ \mathbb{N} = \boldsymbol{s}_{vv} \cdot \boldsymbol{m} \ .$$

It is possible to derive a concise solution to the system of six equations in Eq. (8) by using a special parametrization of $\boldsymbol{s}$ that is analogous to the arc-length or natural parametrization of curves. This parameterization takes the lines of curvature as the u, v parameter axes where the parameter lines are also arc-length. This can be done without loss of generality because lines of curvatures cover the surfaces simply, without gaps [11]. Let us denote this parametrization by $\widehat{\boldsymbol{s}}$.

The normal curvature of a curve on the surface with tangent vector $a \cdot \boldsymbol{s}_u + b \cdot \boldsymbol{s}_v$ is

$$\kappa(a,b) = \frac{\mathbb{L} \cdot a^2 + 2 \cdot \mathbb{M} \cdot a \cdot b + \mathbb{N} \cdot b^2}{\mathbb{E} \cdot a^2 + 2 \cdot \mathbb{F} \cdot a \cdot b + \mathbb{G} \cdot b^2} \tag{9}$$

If the surface is parameterized as $\widehat{\boldsymbol{s}}$, then

$$\mathbb{E} = 1 \ , \ \mathbb{F} = 0 \ , \ \mathbb{G} = 1 \ ,$$
$$\mathbb{L} = \kappa_1 \ , \ \mathbb{M} = 0 \ , \ \mathbb{N} = \kappa_2 \ ,$$

where κ_1, κ_2 are the principal curvature functions, i.e. the minima and maxima of normal section curvatures of Eq. (9) at every point on the surface.

Now, the system of equations in Eq. (8) simplifies to

$$\partial_i u \cdot (1 + d \cdot \kappa_1) = \boldsymbol{e}_i \cdot \widehat{\boldsymbol{s}}_u \circ \boldsymbol{u}$$
$$\partial_i v \cdot (1 + d \cdot \kappa_2) = \boldsymbol{e}_i \cdot \widehat{\boldsymbol{s}}_v \circ \boldsymbol{u}$$

giving us the partial derivatives of the footparameter mappings as a function of distance from the surface and the geometric invariants of the surface at the footpoint:

$$\partial_i u = \frac{\boldsymbol{e}_i \cdot \widehat{\boldsymbol{s}}_u \circ \boldsymbol{u}}{1 + d \cdot \kappa_1} \qquad \partial_i v = \frac{\boldsymbol{e}_i \cdot \widehat{\boldsymbol{s}}_v \circ \boldsymbol{u}}{1 + d \cdot \kappa_2} \tag{10}$$

Let us consider the first degree Taylor expansion of the footvector mapping as

$$\boldsymbol{f}(\boldsymbol{x}) \approx \boldsymbol{f}(\boldsymbol{x}_0) + (x - x_0)\boldsymbol{f}_x(\boldsymbol{x}_0) + (y - y_0)\boldsymbol{f}_y(\boldsymbol{x}_0) + (z - z_0)\boldsymbol{f}_z(\boldsymbol{x}_0) \ ,$$

where, using Eq. (7), the $i \in \{1,2,3\}$ partial derivatives of $\boldsymbol{f}$ are

$$\boldsymbol{f}_i = \boldsymbol{s}_u \circ \boldsymbol{u} \cdot \partial_i u + \boldsymbol{s}_v \circ \boldsymbol{u} \cdot \partial_i v - \boldsymbol{e}_i \ .$$

Substituting this into the Taylor expansion, we get

$$\begin{aligned} \boldsymbol{f}(\boldsymbol{x}) \approx \boldsymbol{f}(\boldsymbol{x}_0) &+ (x - x_0)\Big(\boldsymbol{s}_u \circ \boldsymbol{u} \cdot \partial_1 u + \boldsymbol{s}_v \circ \boldsymbol{u} \cdot \partial_1 v - \boldsymbol{e}_1\Big) \\ &+ (y - y_0)\Big(\boldsymbol{s}_u \circ \boldsymbol{u} \cdot \partial_2 u + \boldsymbol{s}_v \circ \boldsymbol{u} \cdot \partial_2 v - \boldsymbol{e}_2\Big) \\ &+ (z - z_0)\Big(\boldsymbol{s}_u \circ \boldsymbol{u} \cdot \partial_3 u + \boldsymbol{s}_v \circ \boldsymbol{u} \cdot \partial_3 v - \boldsymbol{e}_3\Big) \end{aligned}$$

Assuming a natural parametrization of the surface, this is further developed using Eq. (10) as

$$\begin{aligned}\boldsymbol{f}(\boldsymbol{x}) \approx \boldsymbol{f}(\boldsymbol{x}_0) &+ (x - x_0)\left(\widehat{\boldsymbol{s}}_u \circ \boldsymbol{u} \cdot \frac{\boldsymbol{e}_1 \cdot \widehat{\boldsymbol{s}}_u \circ \boldsymbol{u}}{1 + d \cdot \kappa_1} + \widehat{\boldsymbol{s}}_v \circ \boldsymbol{u} \cdot \frac{\boldsymbol{e}_1 \cdot \widehat{\boldsymbol{s}}_v \circ \boldsymbol{u}}{1 + d \cdot \kappa_2} - \boldsymbol{e}_1\right) \\ &+ (y - y_0)\left(\widehat{\boldsymbol{s}}_u \circ \boldsymbol{u} \cdot \frac{\boldsymbol{e}_2 \cdot \widehat{\boldsymbol{s}}_u \circ \boldsymbol{u}}{1 + d \cdot \kappa_1} + \widehat{\boldsymbol{s}}_v \circ \boldsymbol{u} \cdot \frac{\boldsymbol{e}_2 \cdot \widehat{\boldsymbol{s}}_v \circ \boldsymbol{u}}{1 + d \cdot \kappa_2} - \boldsymbol{e}_2\right) \\ &+ (z - z_0)\left(\widehat{\boldsymbol{s}}_u \circ \boldsymbol{u} \cdot \frac{\boldsymbol{e}_3 \cdot \widehat{\boldsymbol{s}}_u \circ \boldsymbol{u}}{1 + d \cdot \kappa_1} + \widehat{\boldsymbol{s}}_v \circ \boldsymbol{u} \cdot \frac{\boldsymbol{e}_3 \cdot \widehat{\boldsymbol{s}}_v \circ \boldsymbol{u}}{1 + d \cdot \kappa_2} - \boldsymbol{e}_3\right)\end{aligned}$$

or, using the notational shorthands $\widehat{s}_{ui} = \widehat{\boldsymbol{s}}_u \circ \boldsymbol{u} \cdot \boldsymbol{e}_i$, $\widehat{s}_{vi} = \widehat{\boldsymbol{s}}_v \circ \boldsymbol{u} \cdot \boldsymbol{e}_i$,

$$\boldsymbol{f}(\boldsymbol{x}) \approx \boldsymbol{f}(\boldsymbol{x}_0) - (\boldsymbol{x} - \boldsymbol{x}_0) + \sum_{i=1}^{3}((\boldsymbol{x} - \boldsymbol{x}_0) \cdot \boldsymbol{e}_i)\left(\widehat{\boldsymbol{s}}_u \circ \boldsymbol{u} \cdot \frac{\widehat{s}_{ui}}{1 + d \cdot \kappa_1} + \widehat{\boldsymbol{s}}_v \circ \boldsymbol{u} \cdot \frac{\widehat{s}_{vi}}{1 + d \cdot \kappa_2}\right).$$

The first two terms of the approximation are interpreted similarly to the curve case. Since $\boldsymbol{f}(\boldsymbol{x}_0) = \boldsymbol{x}^* - \boldsymbol{x}_0$, $\boldsymbol{f}(\boldsymbol{x}_0) - (\boldsymbol{x} - \boldsymbol{x}_0) = \boldsymbol{x}^* - \boldsymbol{x}$, i.e. the approximation starts from the vector pointing to the footpoint of $\boldsymbol{x}_0$ from the new position $\boldsymbol{x}$. To distill the geometric meaning of the sum in the above approximation, let us rewrite $\boldsymbol{x} - \boldsymbol{x}_0$ in the spherical coordinate system about $\boldsymbol{x}_0$ with axes $\widehat{\boldsymbol{s}}_u$, $\widehat{\boldsymbol{s}}_v$, $\widehat{\boldsymbol{m}}$ as

$$\begin{aligned}\boldsymbol{x} - \boldsymbol{x}_0 &= l \cdot \left(\cos\alpha \sin\beta \cdot \widehat{\boldsymbol{s}}_u + \sin\alpha \sin\beta \cdot \widehat{\boldsymbol{s}}_v + \cos\beta \cdot \widehat{\boldsymbol{m}}\right) \\ &= l \cdot \left(\Delta x \cdot \widehat{\boldsymbol{s}}_u + \Delta y \cdot \widehat{\boldsymbol{s}}_v + \Delta z \cdot \widehat{\boldsymbol{m}}\right),\end{aligned}$$

where $l \geq 0, \alpha \in [0, 2\pi), \beta \in [0, \pi]$. This yields

$$\begin{aligned}\boldsymbol{f}(\boldsymbol{x}) &\approx \boldsymbol{x}^* - \boldsymbol{x} + l \cdot \sum_{i=1}^{3}(\Delta x \cdot \widehat{s}_{ui} + \Delta y \cdot \widehat{s}_{vi} + \Delta z \cdot \widehat{m}_i) \cdot \frac{\widehat{s}_{ui}}{1 + d \cdot \kappa_1} \cdot \widehat{\boldsymbol{s}}_u \circ \boldsymbol{u} \\ &\qquad + l \cdot \sum_{i=1}^{3}(\Delta x \cdot \widehat{s}_{ui} + \Delta y \cdot \widehat{s}_{vi} + \Delta z \cdot \widehat{m}_i) \cdot \frac{\widehat{s}_{vi}}{1 + d \cdot \kappa_2} \cdot \widehat{\boldsymbol{s}}_u \circ \boldsymbol{u} \\ &= \boldsymbol{x}^* - \boldsymbol{x} + \left(\frac{l}{1 + d \cdot \kappa_1} \cdot \sum_{i=1}^{3}(\Delta x \cdot \widehat{s}_{ui}^2 + \Delta y \cdot \widehat{s}_{vi}\widehat{s}_{ui} + \Delta z \cdot \widehat{m}_i\widehat{s}_{ui}) \cdot \widehat{\boldsymbol{s}}_u \circ \boldsymbol{u}\right) \\ &\qquad + \left(\frac{l}{1 + d \cdot \kappa_2} \cdot \sum_{i=1}^{3}(\Delta x \cdot \widehat{s}_{ui}\widehat{s}_{vi} + \Delta y \cdot \widehat{s}_{vi}^2 + \Delta z \cdot \widehat{m}_i\widehat{s}_{vi}) \cdot \widehat{\boldsymbol{s}}_v \circ \boldsymbol{u}\right) \\ &= \boldsymbol{x}^* - \boldsymbol{x} + \frac{l \cdot \widehat{\boldsymbol{s}}_u \circ \boldsymbol{u}}{1 + d \cdot \kappa_1} \cdot \left(\Delta x \sum_{i=1}^{3}\widehat{s}_{ui}^2 + \Delta y \sum_{i=1}^{3}\widehat{s}_{vi}\widehat{s}_{ui} + \Delta z \sum_{i=1}^{3}\widehat{m}_i\widehat{s}_{ui}\right) \\ &\qquad + \frac{l \cdot \widehat{\boldsymbol{s}}_v \circ \boldsymbol{u}}{1 + d \cdot \kappa_2} \cdot \left(\Delta x \sum_{i=1}^{3}\widehat{s}_{ui}\widehat{s}_{vi} + \Delta y \sum_{i=1}^{3}\widehat{s}_{vi}^2 + \Delta z \sum_{i=1}^{3}\widehat{m}_i\widehat{s}_{vi}\right)\end{aligned}$$

Note that $\widehat{\boldsymbol{s}}_u, \widehat{\boldsymbol{s}}_v, \widehat{\boldsymbol{m}}$ form an orthonormal basis, that is, for example $\widehat{\boldsymbol{s}}_u \cdot \widehat{\boldsymbol{s}}_u = \sum_{i=1}^{3} \widehat{s}_{ui}^2 = 1$ and $\widehat{\boldsymbol{s}}_u \cdot \widehat{\boldsymbol{s}}_v = \sum_{i=1}^{3} \widehat{s}_{ui}\widehat{s}_{vi} = 0$. Carrying out the resulting simplifications gives us the final form of the first degree Taylor expansion of the footvector mapping as

$$\begin{aligned} \boldsymbol{f}(\boldsymbol{x}) &\approx \boldsymbol{x}^* - \boldsymbol{x} + \frac{l \cdot \Delta x}{1 + d \cdot \kappa_1} \cdot \widehat{\boldsymbol{s}}_u \circ \boldsymbol{u} + \frac{l \cdot \Delta x}{1 + d \cdot \kappa_2} \cdot \widehat{\boldsymbol{s}}_v \circ \boldsymbol{u} \\ &= \boldsymbol{x}^* - \boldsymbol{x} + \cos\alpha \sin\beta \frac{l}{1 + d \cdot \kappa_1} \cdot \widehat{\boldsymbol{s}}_u \circ \boldsymbol{u} + \sin\alpha \sin\beta \frac{l}{1 + d \cdot \kappa_2} \cdot \widehat{\boldsymbol{s}}_v \circ \boldsymbol{u} \end{aligned} \tag{11}$$

Changing the query position $\boldsymbol{x}$ along the footpoint surface normal $\widehat{\boldsymbol{m}}$, i.e. when $\beta \in \{0, \pi\}$, does not alter the footpoint until we pass the cut locus of the geometry. The footvector is scaled according to the distance between $\boldsymbol{x}$ and $\boldsymbol{x}_0$.

The largest change in the footpoint mapping occurs when the query position is displaced parallel to the tangent plane at the footpoint, in the direction of the smaller principal curvature in magnitude. In this case $\beta = \frac{\pi}{2}$ and Eq. (11) becomes

$$\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{x}^* - \boldsymbol{x} + l \cdot \left(\frac{\cos\alpha}{1 + d \cdot \kappa_1} \cdot \widehat{\boldsymbol{s}}_u \circ \boldsymbol{u} + \frac{\sin\alpha}{1 + d \cdot \kappa_2} \cdot \widehat{\boldsymbol{s}}_v \circ \boldsymbol{u} \right)$$

Interestingly, the magnitude of the change is only equal to the change in the planar footvector mapping of a normal section curve when we are moving $\boldsymbol{x}$ parallel to either of the principal curvature directions. In all other cases, the two quantities will be different since the normal section curvature is $\cos^2 \alpha\kappa_1 + \sin^2 \alpha\kappa_2$ by Euler's theorem.

4 Footpoint Intersection Iteration

Let $G_1 \subseteq \mathbb{R}^n$ and $G_2 \subseteq \mathbb{R}^n$ be two objects with the foot mapping $\boldsymbol{f}_1 : \mathbb{R}^n \to \mathbb{R}^n$ and $\boldsymbol{f}_2 : \mathbb{R}^n \to \mathbb{R}^n$, respectively. Our task is to produce a foot mapping $\boldsymbol{f}_{12} : \mathbb{R}^n \to \mathbb{R}^n$ with $G_{12} = G_1 \cup G_2$ or $G_{12} = G_1 \cap G_2$ similar to Eq. (1). This paper only describes the intersection since the complement geometry has the same foot mapping and $G_{12} = G_1 \cup G_2 = \mathbb{R}^n \setminus \big((\mathbb{R}^n \setminus G_1) \cap (\mathbb{R}^n \setminus G_2)\big)$.

If $\boldsymbol{x} \in G_1 \cap G_2$, then the intersection approximation is precise in Eq. (1), so

$$\boldsymbol{f}_{12}(\boldsymbol{x}) = \begin{cases} \boldsymbol{f}_1(\boldsymbol{x}) & \text{if } \|\boldsymbol{f}_1(\boldsymbol{x})\| \le \|\boldsymbol{f}_2(\boldsymbol{x})\| \\ \boldsymbol{f}_2(\boldsymbol{x}) & \text{otherwise} \end{cases} \qquad (\boldsymbol{x} \in G_1 \cap G_2)\,. \tag{12}$$

If the closest point to G_1 from $\boldsymbol{x} \notin G_1 \cap G_2$ is inside the G_2 set, then that point is the closest point to $\boldsymbol{x}$ in the $G_1 \cap G_2$ intersection. Thus,

$$\boldsymbol{f}_{12}(\boldsymbol{x}) = \begin{cases} \boldsymbol{f}_1(\boldsymbol{x}) & \text{if } \boldsymbol{x} + \boldsymbol{f}_1(\boldsymbol{x}) \in G_2 \\ \boldsymbol{f}_2(\boldsymbol{x}) & \text{if } \boldsymbol{x} + \boldsymbol{f}_2(\boldsymbol{x}) \in G_1 \end{cases} \qquad \big(\boldsymbol{x} \in \mathbb{R}^3 \setminus (G_1 \cap G_2)\big)\,. \tag{13}$$

However, this still leaves some $\boldsymbol{f}_{12}(\boldsymbol{x})$ values for us to define via iterative algorithms. The idea of this naive midpoint approach is to step closer to the intersection and re-evaluate $\boldsymbol{f}_{12}$:

$$\boldsymbol{f}_{12}(\boldsymbol{x}) := \frac{\boldsymbol{f}_1(\boldsymbol{x}) + \boldsymbol{f}_2(\boldsymbol{x})}{2} + \boldsymbol{f}_{12}\left(\boldsymbol{x} + \frac{\boldsymbol{f}_1(\boldsymbol{x}) + \boldsymbol{f}_2(\boldsymbol{x})}{2}\right) \qquad \text{if not (12) or (13).} \tag{14}$$

One can stop evaluating the recursion when one of Eq. (12) or Eq. (13) yield a value or after a predefined number of iterations. Figure 3 illustrates the convergence.

Since $\|\boldsymbol{f}_1\|$ and $\|\boldsymbol{f}_2\|$ are distance functions, there are no surface points inside the corresponding unbounding spheres $\mathcal{K}_{\|\boldsymbol{f}_i(\boldsymbol{x})\|}(\boldsymbol{x}) = \{\boldsymbol{y} \in \mathbb{R}^3 : d(\boldsymbol{x}, \boldsymbol{y}) < \|\boldsymbol{f}_i(\boldsymbol{x})\|\}$:

$$\{\boldsymbol{f}_i = \boldsymbol{0}\} \cap \mathcal{K}_{\|\boldsymbol{f}_i(\boldsymbol{x})\|}(\boldsymbol{x}) = \emptyset \quad (\boldsymbol{x} \in \mathbb{R}^3, i = 1, 2) .$$

The point $\boldsymbol{x}$ is not in the intersection set $G_1 \cap G_2$ in the recursive Eq. (14). Therefore,

$$\begin{aligned} \big(G_1 \cap G_2\big) \cap \big(\mathcal{K}_{\|\boldsymbol{f}_1(\boldsymbol{x})\|}(\boldsymbol{x}) \cup \mathcal{K}_{\|\boldsymbol{f}_2(\boldsymbol{x})\|}(\boldsymbol{x})\big) &= \emptyset \\ \big(G_1 \cap G_2\big) \cap \mathcal{K}_{\max(\|\boldsymbol{f}_1(\boldsymbol{x})\|, \|\boldsymbol{f}_2(\boldsymbol{x})\|)}(\boldsymbol{x}) &= \emptyset \end{aligned}$$

Which means that we can take a larger heuristic step without overstepping with the following bisector iteration:

$$\boldsymbol{f}_{12}(\boldsymbol{x}) := \frac{\boldsymbol{f}_1(\boldsymbol{x}) + \boldsymbol{f}_2(\boldsymbol{x})}{\|\boldsymbol{f}_1(\boldsymbol{x}) + \boldsymbol{f}_2(\boldsymbol{x})\|} + \boldsymbol{f}_{12}\left(\boldsymbol{x} + \frac{\boldsymbol{f}_1(\boldsymbol{x}) + \boldsymbol{f}_2(\boldsymbol{x})}{\|\boldsymbol{f}_1(\boldsymbol{x}) + \boldsymbol{f}_2(\boldsymbol{x})\|}\right) \quad \text{if not (12) or (13).} \tag{15}$$

Figure 3 compares Eq. (14) and Eq. (15) iterative methods, with the upcoming deltoid iteration. This bisector iteration was sufficiently robust for 3D scenes as well. Figure 4 demonstrates the generated surface as a function of the iteration number.

5 Deltoid iteration

For this heuristics, we take advantage of the fact that the footvectors, $\boldsymbol{f}_1(\boldsymbol{x})$ and $\boldsymbol{f}_2(\boldsymbol{x})$, are perpendicular to the surfaces. First, we look for a

$$\boldsymbol{v}(f_1(\boldsymbol{x}), f_2(\boldsymbol{x})) = \alpha \cdot \boldsymbol{f}_1(\boldsymbol{x}) + \beta \cdot \boldsymbol{f}_2(\boldsymbol{x})$$

vector which is in the same plane as the footvectors. Second, the $\boldsymbol{x}+\boldsymbol{v}(\boldsymbol{f}_1(\boldsymbol{x}), \boldsymbol{f}_2(\boldsymbol{x}))$ should lie on each of the tangent planes at the footpoints.

Note that this quadrilateral is not necessarily a deltoid. To be precise, the evaluation point $\boldsymbol{x}$, the footpoints $\boldsymbol{x} + \boldsymbol{f}_1(\boldsymbol{x})$ and $\boldsymbol{x} + \boldsymbol{f}_2(\boldsymbol{x})$, and the next heuristic evaluation point $\boldsymbol{x} + \boldsymbol{v}(\boldsymbol{f}_1(\boldsymbol{x}), \boldsymbol{f}_2(\boldsymbol{x}))$ forms a right angular cyclic quadrilateral.

Therefore, the desired vector $\boldsymbol{v}(\boldsymbol{a}, \boldsymbol{b}) = \alpha \cdot \boldsymbol{a} + \beta \cdot \boldsymbol{b}$ has to be perpendicular to both $\boldsymbol{a} := \boldsymbol{f}_1(\boldsymbol{x})$ and $\boldsymbol{b} := \boldsymbol{f}_2(\boldsymbol{x})$, which means the following:

$$\begin{aligned} (\boldsymbol{v} - \boldsymbol{a}) \cdot \boldsymbol{a} &= 0 \\ (\boldsymbol{v} - \boldsymbol{b}) \cdot \boldsymbol{b} &= 0 \end{aligned} \quad \Longleftrightarrow \quad \begin{bmatrix} \boldsymbol{a}^\top \\ \boldsymbol{b}^\top \end{bmatrix} \cdot \boldsymbol{v} = \begin{bmatrix} \boldsymbol{a}^\top \\ \boldsymbol{b}^\top \end{bmatrix} \cdot \begin{bmatrix} \boldsymbol{a} & \boldsymbol{b} \end{bmatrix} \cdot \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} aa \\ bb \end{bmatrix} .$$

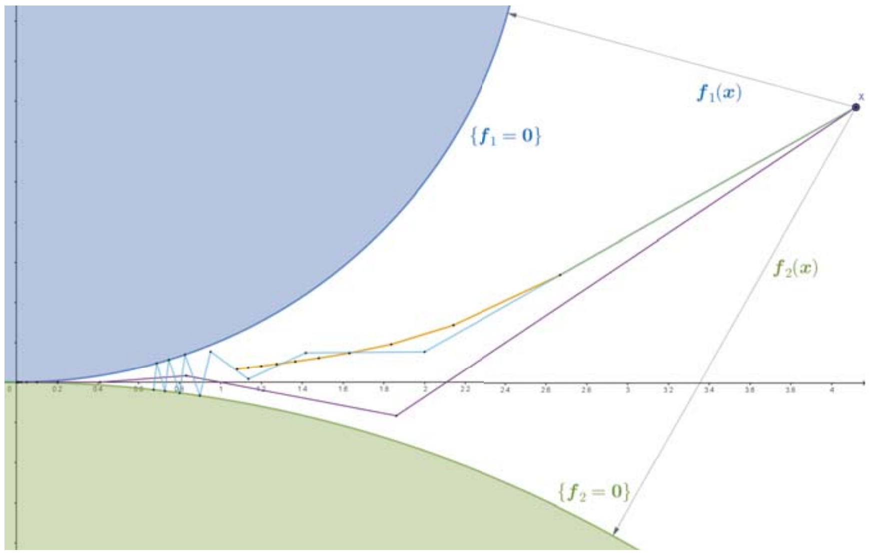


Figure 3: Comparison of the midpoint and deltoid footpoint iterations in a 2D scene where F and G are tangential circles, and the intersection is a single point. The midpoint method in Eq. (14) is the yellow iteration, the blue iteration is improved version from Eq. (15), the purple is the deltoid method from Section 5. The deltoid method converges much faster because of the surface linear approximation and the unrestricted step size.

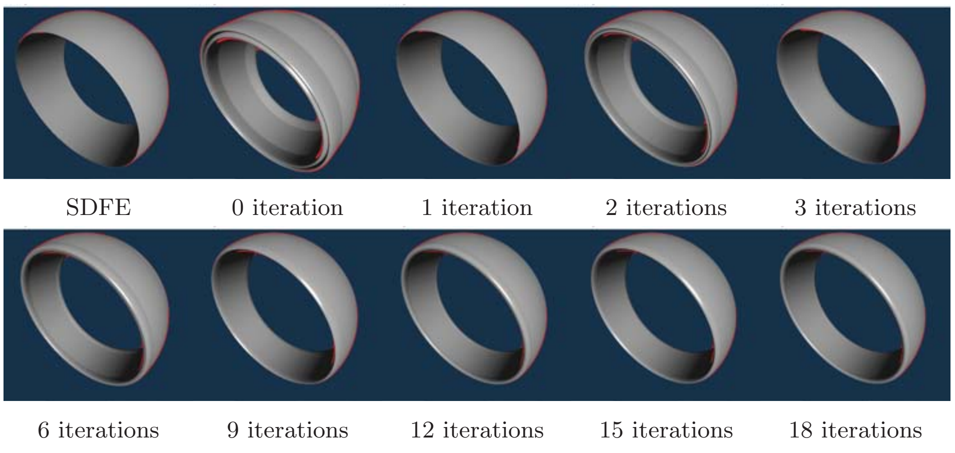


Figure 4: Convergence of the bisector iteration for an offset of a challenging intersection of a sphere and a cylinder. The iteration quickly starts to bounce from one surface to the other, resulting in similar error patterns in every second image, but converges nevertheless.

Where the dot products $aa = \boldsymbol{a} \cdot \boldsymbol{a}, bb = \boldsymbol{b} \cdot \boldsymbol{b}, ab = \boldsymbol{a} \cdot \boldsymbol{b}$. Solving the equation for α and β assuming $d := aa \cdot bb - ab \cdot ab \neq 0$ gives:

$$\begin{bmatrix}\alpha\\ \beta\end{bmatrix} = \begin{bmatrix}aa & ab\\ ba & bb\end{bmatrix}^{-1} \cdot \begin{bmatrix}aa\\ bb\end{bmatrix} = \frac{1}{d}\begin{bmatrix}bb & -ab\\ -ab & aa\end{bmatrix} \cdot \begin{bmatrix}aa\\ bb\end{bmatrix} = \frac{1}{d}\begin{bmatrix}bb \cdot (aa - ab)\\ aa \cdot (bb - ab)\end{bmatrix}. \quad (16)$$

For the deltoid footpoint iteration, we substitude $\boldsymbol{v} = \alpha \cdot \boldsymbol{a} + \beta \cdot \boldsymbol{b}$ from Eq. (16) into Eq. (14), so $\boldsymbol{g}_{12}(\boldsymbol{x}) := \boldsymbol{v} + \boldsymbol{g}_{12}(\boldsymbol{x} + \boldsymbol{v})$ unless Eq. (12) or Eq. (13) provide a foot vector.

The presented algorithms are visualized in 2D on Figure 3. In 2D, the deltoid iteration was the most efficient and was used to produce Figure 2. However, this method diverged and produced too many artifacts in 3D to be applicable, so the bisector iteration was used in our test scenes.

6 Shader code generation from CSG trees

To represent CSG trees in code, we first define the type of a tree node as the union of

- some primitive shape types, like Box, Sphere or Cylinder, and
- some standard CSG operations, like Move, Rotate, Union and Intersect.

Each of these node types contains the specific parameters necessary to describe the object (size, color) or the operation (vector, angle) as fields. Once we have the union type (called Expr), we can build any CSG tree, where the nodes are all Expr objects: the leaves are instances of the primitive shape types, and the internal nodes are operations.

Now we can traverse this tree in a bottom-up or top-down manner to collect (or distribute) information about it. For example, we can count the number of red objects in the tree using this algorithm:

```
def alg(node):
    if node in [Box, Sphere, Cylinder]:
        return 1 if node.color is "red" else 0
    elif node in [Move, Rotate]:
        return alg(node.child)
    elif node in [Union, Intersect]:
        return alg(node.child_1) + alg(node.child_2)
```

Even though the algorithm is run from the root of the tree, what it actually does is breaking down and converting the tree into a single number, starting from the bottom. It begins with the leaf nodes because it can directly convert those. Then for each internal node, it first recursively converts the subtrees, then combines the obtained partial results into a single value using some node-specific logic.

We can use the same approach to derive much more complex information from the tree, like a program code that renders the represented model. We just need to change the so-called carrier type – the type of the value each subtree is reduced into, and the node-specific reduction logic. Our carrier type is a structure that contains a block of code and the name of the register that holds the result of the computation.

The generated block of code for the primitive shapes is simple: it's a function that takes a $\boldsymbol{x} \in \mathbb{R}^3$ position as an argument, and returns four scalars: the footvector and the signed distance value. Code generation in internal nodes combines the code generated for their subtrees. E.g. for a Union node this means that the resulting function executes the functions corresponding to the subtrees, collects their results, then calculates and returns the footvector and the distance for the union.

The intersection operation needs to evaluate its arguments multiple times, so we had to generate actual functions to calculate the subtrees, couldn't just directly calculate the results. Since we cannot define higher order functions in GPU shader code, each occurrence of the intersection operation had to be specialized for its actual arguments as a separate function. Writing all these specializations all by hand would have been tedious and error prone, this is one of the main reasons why we decided to generate the shared code.

The code generation also made it possible to implement optimizations which would be difficult to do manually. Such optimization is pushing down the Move and Rotate operations to the leaf nodes. The intuitive implementation of a CSG evaluator would first compute operand of a Move or a Rotate, and then apply the operation on the result. However, we can render the primitive objects in the leaf nodes with the same cost, regardless of their position or orientation, so we rather aggregate these operations while traversing down the tree, and apply them directly on the leaf objects.

This recursive approach to code generation is well known. What we recognized and used to our advantage is that the CSG model representation is a much simpler tree than usual syntax trees, we do not need any sophisticated state management to process it, which resulted in clean and reusable code generation.

7 Exact offset surfaces

All variants of the footpoint iteration algorithm are heuristic optimization iterations. For this reason, there are cases when the iteration does not converge to the correct solution. In essence, there is a trade-off between robustness and speed. Although we can visualize the above surfaces in real-time on an Nvidia 1080ti GPU, the visualization is expensive and three to ten times slower than the signed distance function representation.

Figure 5 demonstrates the most important advantage of our algorithm, that is, the offset operation will be exact afterwards. With enough iterations, the surface will be smooth because there is no voxelization. For simple surfaces such as the intersection of two objects, the raytracing of the offset surface can be real-time. For more complex scenes, we avoided running the iterative approach for the union operation because the standard minimum distanced union operation yields exact

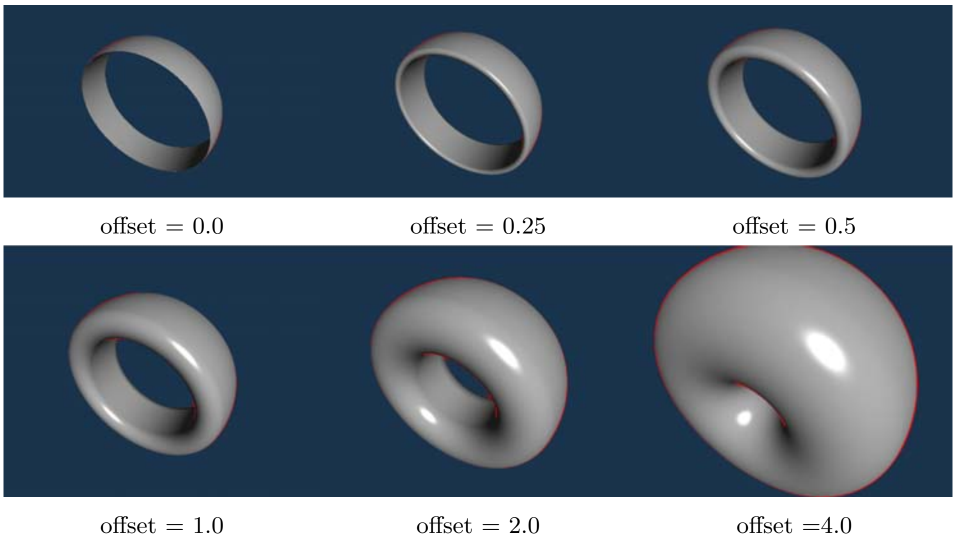


Figure 5: Different offsets of the intersection of a cylinder and a slightly larger sphere. Our bisector algorithm does a 100 iterations for each evaluation to produce the precise offset. The offset can be changed in real-time.

results outside the surface. Unless the result of the union is intersected with another object, the positive offset surface is correct.

Figure 6 showcases a few example surfaces and the drawbacks of the proposed method. Because the method is heuristic, the closest point of the intersection surface is not always found. In such cases, the surface may present artifacts or even appear elsewhere. Scene complexity both increases the likelihood of failure and the evaluation speed. When there are multiple intersection operations, the iterations are correctly inserted into each other by the code generator. This leads to an exponential slowdown in the number of nested intersection operations.

8 Conclusion

This paper proposed a footvector based representation of shapes. Section 3 provides a theoretical background for this, connecting the partial derivatives of the footvector mapping with the local differential geometry at the footpoint. The practicality of this representation, however, is provided by the iterative algorithms that make this representation closed under set theoretic and offset operations.

Offset operation of a signed distance function is as easy as subtracting the offset value from it, yet it is only precise on the CSG tree leafs, so-called primitives, in practise. This is because the SDF of combined objects are only signed distance lower bounds causing the offset surface to appear as if the offset was applied to the arguments of the set operation instead. In this paper, we devised algorithms

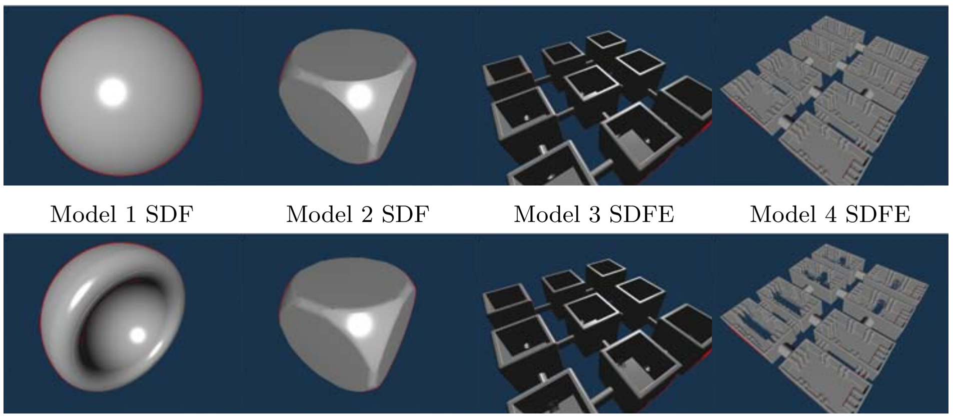


Figure 6: Several example scenes showcasing the strengths and limitations of the proposed methods. The SDFE of Model 1 completely deletes the subtracted sphere, yet the footpoint mapping offsets the correct surface. The improvements in smoothness is visible on Model 2, with some convergence artifacts. The iterative distance function of Model 3 and 4 introduce even more errors.

in Section 4 and 5 that iterate on input functions to produce the SDF of objects. Footvector mapping representations extended the distance information and provide search directions for the intersection operation iterations.

In two dimensions, the deltoid iteration outperformed the rest of the methods by a large factor. Computing the SDF in Figure 2 with the midpoint approach was about ten times slower compared to the deltoid method whilst achieving similar accuracy. Note that the iterations had to be nested to produce the CSG tree of set-operations causing exponential slowdown with CSG tree depth.

In three dimensions, our iterative methods are capable of producing high quality offset surfaces of intersection or difference of objects. The resulting footvector mapping can be visualized in real-time as a signed distance function despite the extra iterations within each intersection operation. Note that the expensive function evaluation time can amortized with better sphere tracing algorithms, such as enhanced sphere tracing [1] or quadric tracing which is an unpublished algorithm for reducing the number of function evaluations by pre-cacheing values.

For rendering purposes the SDF had to be implemented in shader code which does not support higher order functions. Hence, a CSG code generator was designed that created efficient implementations for our test scenes. In three dimensions, the bisector method performed the best because the deltoid iteration often did not converge to the right solution. Nevertheless, for most simple cases, the bisector method converged without artifacts, producing accurate offset surfaces.

References

[1] Bálint, Csaba and Valasek, Gábor. Accelerating Sphere Tracing. In Diamanti, Olga and Vaxman, Amir, editors, *EG 2018 - Short Papers*. The Eurographics Association, 2018. DOI: `10.2312/egs.20181037`.

[2] Bálint, Csaba, Valasek, Gábor, and Gergó, Lajos. Operations on signed distance functions. *Acta Cybernetica*, 24(1):17–28, May 2019. DOI: `10.14232/actacyb.24.1.2019.3`.

[3] Bán, Róbert, Bálint, Csaba, and Valasek, Gábor. Area Lights in Signed Distance Function Scenes. In Cignoni, Paolo and Miguel, Eder, editors, *Eurographics 2019 - Short Papers*. The Eurographics Association, 2019. DOI: `10.2312/egs.20191021`.

[4] do Carmo, Manfredo P. *Differential geometry of curves and surfaces.* Prentice Hall, 1976.

[5] Fabbri, Ricardo, Costa, Luciano Da F., Torelli, Julio C., and Bruno, Odemir M. 2D euclidean distance transform algorithms: A comparative survey. *ACM Comput. Surv.*, 40(1), February 2008. DOI: `10.1145/1322432.1322434`.

[6] Foley, James David. *12.7 Constructive Solid Geometry.* In *Computer Graphics: Principles and Practice*, pages 533–558. Addison-Wesley Professional, 1990.

[7] Hart, John C. Sphere tracing: a geometric method for the antialiased ray tracing of implicit surfaces. *The Visual Computer*, 12:527–545, 1996. DOI: `10.1007/s003710050084`.

[8] Hoffmann, Christoph M. *Boolean Operations on Boundary Representation.* In *Geometric and Solid Modeling: An Introduction*, pages 67–110. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1989.

[9] Kallay, Michael. A geometric Newton–Raphson strategy. *Computer Aided Geometric Design*, 18(8):797–803, 2001. DOI: `10.1016/S0167-8396(01)00070-X`.

[10] Keinert, Benjamin, Schäfer, Henry, Korndörfer, Johann, Ganse, Urs, and Stamminger, Marc. Enhanced Sphere Tracing. In Giachetti, Andrea, editor, *Smart Tools and Apps for Graphics - Eurographics Italian Chapter Conference.* The Eurographics Association, 2014. DOI: `10.2312/stag.20141233`.

[11] Martin, R.R. Principal Patches - A New Class of Surface Patch Based on Differential Geometry. In ten Hagen, P.J.W., editor, *Eurographics Conference Proceedings.* The Eurographics Association, 1983. DOI: `10.2312/eg.19831003`.

[12] Patrikalakis, Nicholas M. and Maekawa, Takashi. *Shape Interrogation for Computer Aided Design and Manufacturing.* Springer Publishing Company, Incorporated, 1st edition, 2009.

[13] Sethian, James A. Fast marching methods. *SIAM Rev.*, 41(2):199–235, June 1999. DOI: `10.1137/S0036144598347059`.

[14] Valasek, Gábor. Generating distance fields from parametric plane curves. In *Annales Mathematicae et Informaticae 48*, pages 83–91, 03 2018.

Contents continued from outside back cover

Contents

Contents continued on inside back cover

ISSN 0324—721 X (Print)
ISSN 2676—993 X (Online)

Editor-in-Chief: Tibor Csendes