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 tradeoff 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

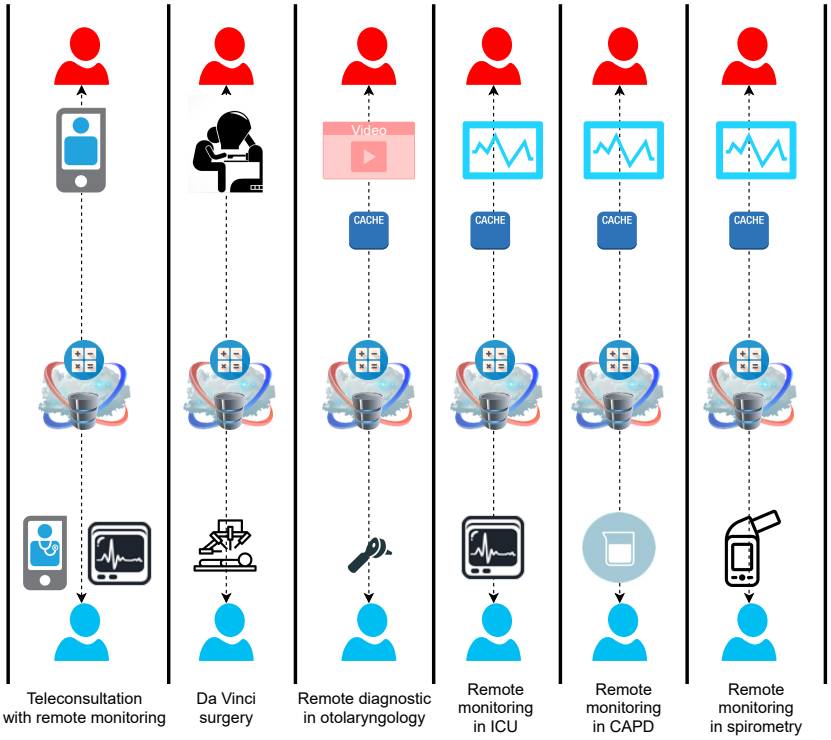

Figure 1: Data paths in real telemedicine use-cases

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.

$$ClientData := (\{x_m, op_m\}, \ldots, \{x_1, op_1\}) \tag{1}$$
$$DBData := (\{y_n, op_n\}, \ldots, \{y_1, op_1\}) \tag{2}$$
$$ProcData := (\{z_l, op_l\}, \ldots, \{z_1, op_1\}) \tag{3}$$

$$ClientWrite(x) := \{x_i, op_i\} \circ ClientData \tag{4}$$
$$DBWrite(y) := \{y_j, op_j\} \circ DBData \tag{5}$$
$$ProcWrite(z) := \{z_k, op_k\} \circ ProcData \tag{6}$$

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

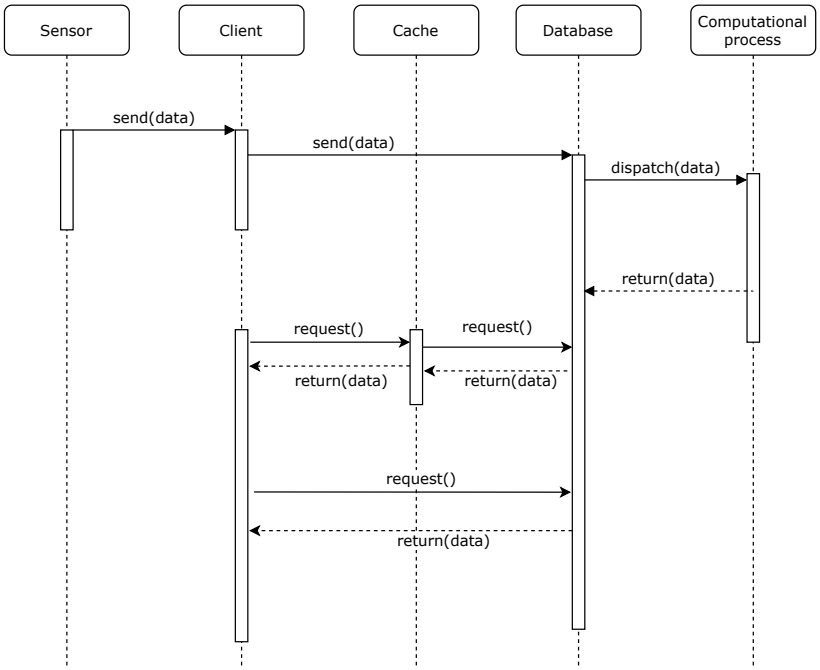

Figure 5: The simulation environment

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.

$$
\begin{array}{lll}
NO\_NS(self) & \triangleq & \wedge\ pc[self] = \text{“NO\_NS”} \\
& & \wedge\ \text{IF}\ (lat\_db[self] < db\_latency) \\
& & \quad \text{THEN}\ \wedge\ lat\_db' = [lat\_db\ \text{EXCEPT}\ ![self] = lat\_db[self] + 1] \\
& & \qquad \wedge\ pc' = [pc\ \text{EXCEPT}\ ![self] = \text{“NO\_NS”}] \\
& & \qquad \wedge\ \text{UNCHANGED}\ \langle db\_type,\ dbData,\ Cache \rangle \\
& & \quad \text{ELSE}\ \wedge\ db\_type' = \text{“PC/EL”} \\
& & \qquad \wedge\ Cache' = \text{FALSE} \\
& & \qquad \wedge\ dbData' = \langle [d \mapsto (Head(clientData).d),\ type \mapsto db\_type'] \rangle \circ dbData \\
& & \qquad \wedge\ pc' = [pc\ \text{EXCEPT}\ ![self] = \text{“DB”}] \\
& & \qquad \wedge\ \text{UNCHANGED}\ lat\_db \\
& & \wedge\ \text{UNCHANGED}\ \langle db\_latency,\ proc\_latency,\ clientData,\ procData, \\
& & \qquad readData,\ cachedData,\ K,\ num\_op,\ data,\ lat\_proc, \\
& & \qquad d \rangle
\end{array}
$$

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,

$$
\begin{array}{lll}
NO\_S(self) & \triangleq & \wedge\ pc[self] = \text{“NO\_S”} \\
& & \wedge\ \text{IF}\ (lat\_db[self] < db\_latency) \\
& & \quad \text{THEN}\ \wedge\ lat\_db' = [lat\_db\ \text{EXCEPT}\ ![self] = lat\_db[self] + 1] \\
& & \qquad \wedge\ pc' = [pc\ \text{EXCEPT}\ ![self] = \text{“NO\_S”}] \\
& & \qquad \wedge\ \text{UNCHANGED}\ \langle db\_type,\ dbData,\ Cache \rangle \\
& & \quad \text{ELSE}\ \wedge\ db\_type' = \text{“PC/EC”} \\
& & \qquad \wedge\ Cache' = \text{FALSE} \\
& & \qquad \wedge\ dbData' = \langle [d \mapsto (Head(clientData).d),\ type \mapsto db\_type'] \rangle \circ dbData \\
& & \qquad \wedge\ pc' = [pc\ \text{EXCEPT}\ ![self] = \text{“DB”}] \\
& & \qquad \wedge\ \text{UNCHANGED}\ lat\_db \\
& & \wedge\ \text{UNCHANGED}\ \langle db\_latency,\ proc\_latency,\ clientData,\ procData, \\
& & \qquad readData,\ cachedData,\ K,\ num\_op,\ data,\ lat\_proc, \\
& & \qquad d \rangle
\end{array}
$$

Figure 8: Formal definition of the process from the *NO_S* class

$$
\begin{array}{lll}
SO\_NS(self) & \triangleq & \wedge\ pc[self] = \text{“SO\_NS”} \\
& & \wedge\ \text{IF}\ (lat\_db[self] < db\_latency) \\
& & \quad \text{THEN}\ \wedge\ lat\_db' = [lat\_db\ \text{EXCEPT}\ ![self] = lat\_db[self] + 1] \\
& & \qquad \wedge\ pc' = [pc\ \text{EXCEPT}\ ![self] = \text{“SO\_NS”}] \\
& & \qquad \wedge\ \text{UNCHANGED}\ \langle db\_type,\ dbData,\ K,\ Cache \rangle \\
& & \quad \text{ELSE}\ \wedge\ db\_type' = \text{“PC/EL”} \\
& & \qquad \wedge\ Cache' = \text{TRUE} \\
& & \qquad \wedge\ K' = 3 \\
& & \qquad \wedge\ dbData' = \langle [d \mapsto (Head(clientData).d),\ type \mapsto db\_type'] \rangle \circ dbData \\
& & \qquad \wedge\ pc' = [pc\ \text{EXCEPT}\ ![self] = \text{“DB”}] \\
& & \qquad \wedge\ \text{UNCHANGED}\ lat\_db \\
& & \wedge\ \text{UNCHANGED}\ \langle db\_latency,\ proc\_latency,\ clientData,\ procData, \\
& & \qquad readData,\ cachedData,\ num\_op,\ data,\ lat\_proc,\ d \rangle
\end{array}
$$

Figure 9: Formal definition of the process from the *SO_NS* class

$$
\begin{array}{lll}
SO\_S(self) & \triangleq & \wedge\ pc[self] = \text{“SO\_S”} \\
& & \wedge\ \text{IF}\ (lat\_db[self] < db\_latency) \\
& & \quad \text{THEN}\ \wedge\ lat\_db' = [lat\_db\ \text{EXCEPT}\ ![self] = lat\_db[self] + 1] \\
& & \qquad \wedge\ pc' = [pc\ \text{EXCEPT}\ ![self] = \text{“SO\_S”}] \\
& & \qquad \wedge\ \text{UNCHANGED}\ \langle db\_type,\ dbData,\ K,\ Cache \rangle \\
& & \quad \text{ELSE}\ \wedge\ db\_type' = \text{“PC/EL”} \\
& & \qquad \wedge\ Cache' = \text{TRUE} \\
& & \qquad \wedge\ K' = 0 \\
& & \qquad \wedge\ dbData' = \langle [d \mapsto (Head(clientData).d),\ type \mapsto db\_type'] \rangle \circ dbData \\
& & \qquad \wedge\ pc' = [pc\ \text{EXCEPT}\ ![self] = \text{“DB”}] \\
& & \qquad \wedge\ \text{UNCHANGED}\ lat\_db \\
& & \wedge\ \text{UNCHANGED}\ \langle db\_latency,\ proc\_latency,\ clientData,\ procData, \\
& & \qquad readData,\ cachedData,\ num\_op,\ data,\ lat\_proc,\ d \rangle
\end{array}
$$

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-

$$
\begin{array}{lll}
O\_NS(self) & \triangleq & \land\ pc[self] = \text{“O\_NS”} \\
& & \land\ \text{IF } (lat\_db[self] < db\_latency) \\
& & \quad \text{THEN } \land\ lat\_db' = [lat\_db \text{ EXCEPT } ![self] = lat\_db[self] + 1] \\
& & \qquad \land\ pc' = [pc \text{ EXCEPT } ![self] = \text{“O\_NS”}] \\
& & \qquad \land\ \text{UNCHANGED } \langle db\_type,\ dbData,\ K,\ Cache \rangle \\
& & \quad \text{ELSE } \land\ db\_type' = \text{“PA/EL”} \\
& & \qquad \land\ Cache' = \text{TRUE} \\
& & \qquad \land\ K' = 5 \\
& & \qquad \land\ dbData' = \langle [d \mapsto (Head(clientData).d),\ type \mapsto db\_type'] \rangle \circ dbData \\
& & \qquad \land\ pc' = [pc \text{ EXCEPT } ![self] = \text{“DB”}] \\
& & \qquad \land\ \text{UNCHANGED } lat\_db \\
& & \land\ \text{UNCHANGED } \langle db\_latency,\ proc\_latency,\ clientData,\ procData, \\
& & \qquad\qquad readData,\ cachedData,\ num\_op,\ data,\ lat\_proc,\ d \rangle
\end{array}
$$

Figure 11: Formal definition of the process from the *O_NS* class

$$
\begin{array}{lll}
O\_S(self) & \triangleq & \land\ pc[self] = \text{“O\_S”} \\
& & \land\ \text{IF } (lat\_db[self] < db\_latency) \\
& & \quad \text{THEN } \land\ lat\_db' = [lat\_db \text{ EXCEPT } ![self] = lat\_db[self] + 1] \\
& & \qquad \land\ pc' = [pc \text{ EXCEPT } ![self] = \text{“O\_S”}] \\
& & \qquad \land\ \text{UNCHANGED } \langle db\_type,\ dbData,\ K,\ Cache \rangle \\
& & \quad \text{ELSE } \land\ db\_type' = \text{“PA/EL”} \\
& & \qquad \land\ Cache' = \text{TRUE} \\
& & \qquad \land\ K' = 3 \\
& & \qquad \land\ dbData' = \langle [d \mapsto (Head(clientData).d),\ type \mapsto db\_type'] \rangle \circ dbData \\
& & \qquad \land\ pc' = [pc \text{ EXCEPT } ![self] = \text{“DB”}] \\
& & \qquad \land\ \text{UNCHANGED } lat\_db \\
& & \land\ \text{UNCHANGED } \langle db\_latency,\ proc\_latency,\ clientData,\ procData, \\
& & \qquad\qquad readData,\ cachedData,\ num\_op,\ data,\ lat\_proc,\ d \rangle
\end{array}
$$

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.