Acta Cybernetica **17** (2005) 185–198.

Measurement and Optimization of Access Control Lists

Sándor Palugyai[*], Máté J. Csorba[*], Sarolta Dibuz[*], and Gyula Csopaki[†]

Abstract

This paper deals with the examination of Access Control Lists (ACLs) that are used in IP routers mainly for providing network admission control and maintaining a certain level of quality of service. In our work we present a method for measuring the performance impact of ACLs on the packet forwarding capabilities of a router. Besides, our study proposes new methods to model and optimize the operation and reduce the redundancy of ACLs.

1 Introduction

Nowadays the Internet usage is progressing at a great pace. More and more people become potential users and require faster connections. Recently, security has also become an important issue in business networks and at home as well. Because of these facts devices have to be designed and created, which allow us to build and maintain a more secure network. Their operation has to be optimized also. In this work a method is proposed for the optimization of Access Control Lists (ACLs) used in routers, which can maintain the operation of large networks. Besides, we present the model designed for the optimization process.

In the next section we describe how ACLs work and can be used in IP networks. We also give a short example on their usage. In Section 3 we outline our method for the measurement of ACL performance that was implemented in TTCN-3, and we present our measurement results with a conventional access router. In Section 4 we introduce a directed graph representation of ACLs and we give an algorithm based on the model for optimization of packet forwarding performance. In Section 5 experimental measurement results are presented to demonstrate the applicability of our method. Finally, in Section 6 conclusion is given together with the possible future developments.

[*]Ericsson Hungary Ltd., Test Competence Center, H-1117 Budapest, Irinyi J. u. 4-20. Email: {sandor.palugyai, mate.csorba, sarolta.dibuz}@ericsson.com

[†]Technical University of Budapest, Department of Telematics and Media Informatics Email: csopaki@tmit.bme.hu

2 The Application of Access Control Lists

Nowadays, TCP/IP is the most widely used networking protocol, so it is an important security issue to control or restrict TCP/IP access. To achieve the needed control over IP traffic and to prohibit unauthorized access, ACLs are a commonly used solution in firewall routers, border routers and in any intermediate router that needs to filter traffic.

ACLs are basically criteria put into a set of sequential conditions. Each line of such a list can permit or deny specific IP addresses or upper-layer protocols. Incoming or outgoing traffic flows can be classified and managed by a router using ACLs. There are two basic types of ACLs: standard and extended.

With standard IP access lists, a router is capable of filtering the traffic based on source addresses only. Extended access lists, on the other hand, offer more sophisticated methods for access control by allowing filtering based not only on source addresses, but also on destination addresses and other protocol properties. Hence the command syntax of an extended ACL can be far more complex than a standard one.

Standard ACL syntax:

```
Router1(config)# access-list acl-number {deny | permit} [host]
source-address [source-mask] [log]
```

Extended ACL syntax:

```
Router1(config)# access-list acl-number {deny | permit} protocol
[host] source-address [source-mask] [host] destination-address
[destination-mask] [precedence precedence-id] [tos tos-id]
[established] [log] [time-range tr-name]
```

Figure 1: **Standard and extended access list syntax**

In many cases ACLs are used for allocating resources needed by a user at a given time of a day, or to automatically reroute traffic according to the varying access rates provided by the ISPs. Service Level Agreements (SLAs), negotiated in advance, can be satisfied as well if time ranges are also specified in an access list. However time-based ACLs are not taken into consideration in this paper.

ACLs can be applied on one or more interfaces of the router and in both directions, but they work differently depending on which direction they are applied. When applied on outgoing interfaces, every received packet must be processed and switched by the router to the proper outgoing interface before checking against the appropriate list. And in case the rules defined in the list drop the packet, this results in a waste of processing power.

When the administrator defines the access lists needed, they must be applied on the proper interface by issuing the ip access-group command.

An application area for access lists is called session filtering. The main purpose of session filtering is to prevent (possibly malicious users on) outside hosts connecting to hosts inside, while still allowing users inside the protected network to establish connections to the outside world.

For the sake of clarity, consider the following example (Fig. 2). The administrator who is managing a local network wants to allow users of the corporate network to access the local web-server, but at the same time access to the local workstations must be prohibited. Besides, the workstations should be able to establish connections destined to the corporate network.

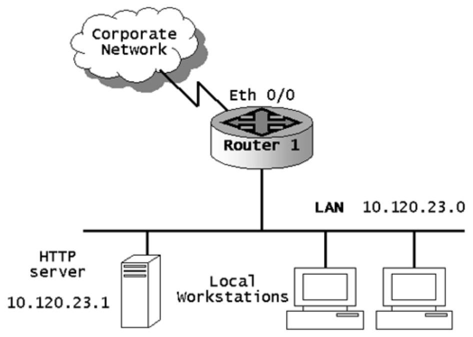

Figure 2: **Example network**

The solution to the problem introduced above is realized by the ACL numbered 111, which contains two separate lines. The first permits TCP traffic originated from any host, destined to the single host 10.120.23.1, which is an HTTP server. The destination TCP port is also restricted to 80, on which the HTTP server software is listening. The second line prohibits connections initiated by any host on the Corporate Network destined to the local network 10.120.23.0. Although it seems that this statement cuts the whole internal LAN from the outside world, the HTTP server is still available to connections because every incoming packet is checked against the statements sequentially.

Example ACL (two lines):

```
Router1(config)# access-list 111 permit tcp any host 10.120.23.1
eq 80
Router1(config)# access-list 111 deny any 10.120.23.0
0.0.0.255
```

Figure 3: **Setting up an ACL (an example)**

So, if the incoming packet belongs to a connection destined to the HTTP server, it matches the first line of the ACL and it is routed and transmitted to its destination. Any other packets that do not match the first line are checked against the second line and are discarded. In fact, every access list has a virtual line at the end that is called the implicit deny rule. The implicit deny discards every packet originated from any address destined to any other address. So, if the examined packet does not match any of the rules, at the end it matches the implicit deny rule and it is discarded. As a matter of fact the second line is not necessary. Finally, when the proper access list is constructed, it needs to be bound to an interface of the router (Fig. 4).

Applying the ACL on interface Ethernet0/0:

```
Router1(config-if)# ip access-group 111 in
```

Figure 4: **Setting up an ACL (continued)**

Although conventional ACLs are relatively static, dynamic access lists exist to allow the rules to be changed for a short period of time, but require additional authentication processes. In this case exceptions are granted for the user (possibly with a higher privilege-level) to access additional network elements. The current work does not consider these types of ACLs [1].

When an ACL is applied on a router's interface, the router is forced to check every packet sent or received on that interface depending on the type of the ACL (in or out). This can seriously affect the packet forwarding performance. A very simple solution to cope with the performance impact of ACLs is to use the null0 interface, which is implemented software-only and acts as a garbage bin or a virtual interface for the unwanted traffic.

The null0 interface can be used if and only if all of the traffic destined to a particular host or network destination needs to be restricted. In this case, a static route to the null0 interface can be added to the route table. This way the router forwards the unwanted traffic to the virtual garbage bin simply via a routing table entry without checking the packets against the ACL [2].

3 Measurement of ACL Performance

Once an ACL is bound to one of the router's interfaces it may have a serious effect on packet forwarding. To determine the properties of this effect and to be able to compare performance of routers from different vendors we developed a method for measuring the ACL performance.

The performance measurements were implemented in TTCN-3 (Testing and Test Control Notation version 3) language [3], which is originally used for conformance testing purposes and is very similar to the conventional C language. Accordingly, it is well equipped with constructs, supporting message-based communication with the particular implementation under test.

Basically, a TTCN-3 test program has the following necessary modules. A module containing the definition of packet and message types used during testing; another module contains the so-called templates, which are basically constraints to the incoming and outgoing communication; a main module contains the functions and test cases used during testing, and reads the configuration files for parameters that are alternating between each test execution.

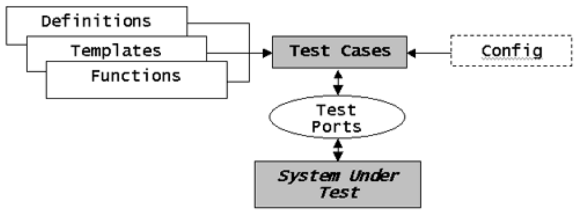

Figure 5: **Simplified diagram of the most important modules of a TTCN-3 program**

A very important part of the test system is the test port, which establishes connection to the operating system allowing the test program to establish communication with the implementation under test.

In traditional conformance testing methodology test ports are used simply as communication bridges without any further intelligence. Our measurement method utilizes modified IP test ports, which allow the use of precise timings and the transmission of IPv4 packets. But, beyond the original capabilities, the modified test port can cope with delay and round-trip-time measurements by using time stamps for any packet passing through.

The measurement method uses different traffic patterns to estimate the delay as a function of ACL size. In all cases artificial flows are generated using TTCN-3. One option is to apply a rough estimation and simulate the distribution of packets, according to a macroscopic view of real Internet traffic. The packet distribution is composed based on the data collected by the NLANR project [4]. During this project 342 million packets were observed and analyzed. The average packet size was 402.7 bytes.

The traffic according to this model consists of the following three main packet types:

— 40 bytes: TCP packets without payload (20 bytes IP header + 20 bytes TCP header). These packets can be observed typically at initiation of a TCP connection. Approximately 35% of the packets can be classified into this type, but because these packets are very small this type gives only 3.5% of the traffic.

— 576 bytes: TCP packets of obsolete implementations, which still use the MSS (Maximum Segment Size) value. 11.5% of the packets are this type though giving 16.5% of the traffic.

— 1500 bytes: packet size according to the Ethernet MTU (Maximum Transfer Unit). Most of the data flowing through the Internet consists of full sized Ethernet frames, though giving 10% of the packets and 37% of the overall traffic.

Considering packets with occurrence rate over 0.5%, the following packet sizes may occur (in order of frequency): 52, 1420, 444, 48, 60, 628, 552, 64, 56 and 1408 bytes. During the NLANR project 1.2% of the packets were smaller than 40 bytes. Although these packets are very small (only 0.1% of the traffic) the routers have to forward them also, and must be capable to handle the serious overhead caused by them.

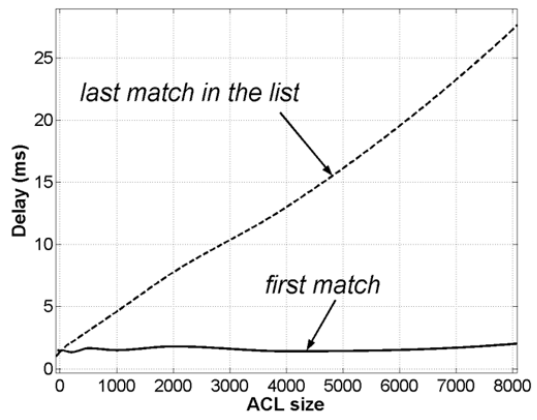

Figure 6: **Delay as a function of ACL size**

In the example (Fig. 6), 64 byte long UDP packets are generated every millisecond. This speed is relatively slow compared to the raw throughput capability of the router under test (Cisco 2600 series access router, with an approximate transmission capability of 15000 packets per second [5]), to avoid undesirable latency or packet loss. The results confirm the conjecture that the delay is increasing significantly with the increasing number of access list entries.

When examining ACL behavior, content carried in packet header fields carries relevant information. Accordingly measurement traffic is composed without respect to protocol payload, while the header fields are variable. Optimization is made based upon the match probabilities that the optimization algorithm reads as input for the process. By means of different match probabilities different traffic mixtures can be represented.

We have also measured the delay with test rules mapped to the routing table using the *null0* interface. In this case the average delay was only 0.32 ms with a variance of 0.1 us. However in this measurement, we had to constrain the rules to use destination address prefixes only, because of the limited capabilities of this filtering solution.

4 Optimization of ACL Performance

The first objective during our research was to find a suitable representation format for the access lists for further examination. Hazelhurst [6] proposed the usage of binary decision diagrams, first introduced by Bryant [7] to represent access lists

Table 1: **The example ACL in the original order with match numbers**

Node	**rule**	**prefix/length**	**# of matches**
C	deny	10.120.238.130/32	4299
O	deny	10.120.238.7/32	357
G	deny	10.120.240.0/24	2500
A	deny	10.120.238.0/28	1214
B	permit	10.120.238.0/26	4910
D	permit	10.120.238.128/26	1703
J	deny	10.120.0.0/16	2028
K	permit	10.121.130.0/24	125
L	deny	10.121.0.0/16	1380
I	permit	10.120.239.132/32	417
N	deny	10.120.239.128/26	1612
H	deny	10.120.239.0/24	3301
E	deny	10.120.238.0/24	3
M	permit	10.120.238.64/26	0
F	permit	10.120.0.0/16	3405

systematically. We decided to use directed graphs to describe the dependencies between the list entries.

The example graph is constructed considering the following parameters: type of rule (permit/deny), network prefix, network mask length and the number of times the rule matched. Every node represents one line of the access list. In this quite simple example we assume that filtering is based on destination addresses only, and more importantly there is no rule querying any upper-layer protocol information such as TCP and UDP port numbers or protocol IDs.

From this representation of the access list, the following graph can be constructed identifying which network prefix is more general and contains the other prefix as well (Fig. 7). The nodes are labeled by the capitals A..O each one representing a single rule (one line of the list).

At first, existing redundancy can be decreased in the list, by eliminating nodes, which depend from a node with a wider prefix having the same rule, and the wider rule is not represented by a star-like node. For example, a node can be deleted if its rule is preceding the other rule in the original list and its rule is completely a subset of the other rule at the same time. In this case, the weight of the actual rule (number of matches) is added to the weight of the more general rule. Secondly, suspicious nodes with 0 match can be checked and eliminated if needed. For example, node M permits traffic to 10.120.238.64/26 but network 10.120.238.0/24 is prohibited by node E, which comes before M in the original list, so M can be spared.

We optimize an extended ACL, so each rule may contain port, protocol and address related constraints. The graph representing an extended ACL may contain cycles, as for example in the following basic rule-set.

Table 2: **Cycle in the ACL**

No.	rule	Source address	Destination address	Protocol
1	deny	10.120.0.0/24	any	any
2	deny	any	10.200.20.0/24	any
3	deny	any	any	TCP

In this example every list entry is a deny, but none of them can be deleted, because the rules are not a complete subset of each other. The constructed graph will contain the entries the following way (Fig. 8).

Since the rules are examined in a sequential order, obviously the order in which they are specified has a semantic meaning [8]. Accordingly, during the optimization process the edges in the composed graph are directed based on these dependencies. In the example (Fig. 7), we consider the delay that a packet suffers equal for every list entry check operation, because the rules are simple and very similar to each other. But, generally the delay caused by a rule in a list is varying. However, the overall delay of the traffic can be estimated only if we also consider the arrival intensity of the traffic and the queuing at the in/out interfaces [9]. Hence, we estimate the delay of the traffic in this example with a ratio, which is equivalent to the case when the traffic is slow enough that no queuing is present at the router's interfaces. In turn, we are able to compare the improvement we can gain by reordering the list entries.

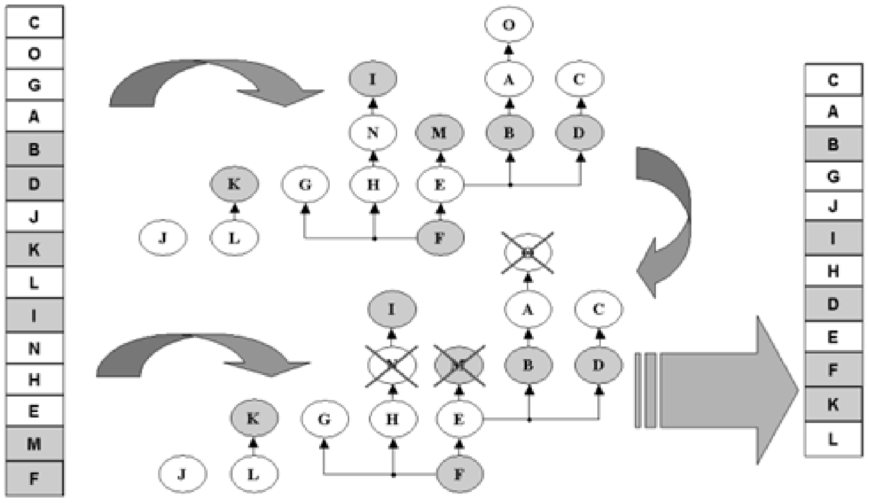

Figure 7: **The graph representing the example and the optimization process**

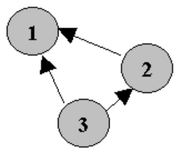

Figure 8: **Cycles in the constructed graph**

We define the following parameters:

L :	the set of actual list entries;
T :	the traffic that is matched against list L;
$n(L)$:	the number of list entries in L;
$r_i(L)$:	the delay of rule number i. in list L;
$m_i(L,T)$:	the probability that a packet in T matches rule number i. in L.

Considering these parameters the total delay a packet suffers that matches rule number i. in list L can be estimated as:

$$d_i(L) = \sum_{k=1}^{i} r_k(L) \tag{1}$$

Furthermore, the total delay traffic T suffers while filtered through list L if there is no queuing present at the network interfaces:

$$Delay(L,T) = \sum_{i=1}^{n(L)} m_i(L,T) \cdot d_i(L) = \sum_{i=1}^{n(L)} m_i(L,T) \cdot \sum_{k=1}^{i} r_k(L) \tag{2}$$

According to (2) the example input ACL (in Table 1 and Fig. 7) has a delay value of 192381, while the resulting ACL (in Fig. 7) has a value of 144840. These values do not have a unit, since they represent a ratio only for comparison. According to them the delay has been reduced by approximately 25%. The following algorithms use these formulas for comparing runtime results.

At first, we have developed a brute force algorithm to optimize the graph representing the ACL. This algorithm is executed after the graph has been built up in the memory and existing redundancy is removed. The algorithm evaluates every possible layout of the graph depending on the meanings of the rules and preserving the order of nodes that are dependent on each other. Afterwards, the theoretic delay of every layout is calculated using the weights of the nodes. At the end, the layout with the lowest calculated delay is chosen as the best solution. The reordered graph is then transformed back to a sequential list and can be uploaded to the router.

However, the applicability of this algorithm is highly limited because of the time it takes to evaluate every possible set-up. For example, to check 14 nodes lasts 5 sec and for 15 nodes it is already 74 sec, 16 nodes last 1310 sec and for 20 nodes it would take approximately 1763 days.

The graph optimizing processes were implemented in C++, because of the computationally intense calculations. Besides, the on-line communication with the router is implemented in *Perl* language and uses a telnet connection. This way the software can connect to routers from various vendors including Cisco, besides there is no need to implement the optimization inside the router, but it can be performed remotely from the connected network.

In order to overcome the limitations of the brute force algorithm we have to consider a more efficient way of rebuilding the Access Control List. But first, we need to fabricate a criterion for a basic building block of our optimization process, namely to estimate the resource demand of merging two simple sub-graphs of the ACL.

So, let us consider two separate sub-graphs of an ACL, namely list K and L. Let us also assume that the number of list entries is k and l respectively. In this case the two lists can be merged s_{kl}ways:

$$s_{kl} = g_{l+1}^T \cdot A_{l+1}^{k-1} \cdot e_{l+1} \tag{3}$$

Whereas matrices g_{l+1}, e_{l+1} and A_{l+1} are the following:

$$\mathbf{g}_{l+1} = \begin{bmatrix} 1 \\ 2 \\ \vdots \\ l+1 \end{bmatrix} \quad ; \quad e_{l+1} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix} \quad ; \quad A_{l+1} = \begin{bmatrix} 1 & 1 & \cdots & 1 & 1 \\ 0 & 1 & \cdots & 1 & 1 \\ \vdots & & \ddots & & \vdots \\ 0 & 0 & \cdots & 0 & 1 \end{bmatrix} \quad ;$$

More importantly, separate entries of K and L preserve their order in the resulting list. Using Equation 3 we constructed the following algorithm:

Algorithm 1 (The pseudo-code of the optimization algorithm).

```
1. Establish an authenticated connection towards the router;
2. Query the Access List data;
   {
     2.1. FOREACH list entry
        {
         2.1.1. Store the actual rule and number of matches into the
            memory
        }
   }
3. IF the newly created list is not the same as the one we have stored
   previously
   {
     3.1. Construct a (possibly non-continuous)graph structure according
        to the ACL rules;
```

```
      3.2. Assign weights to every node based on the number of matches on
         the particular list entry and on the delay of the actual rule;
      3.3. Eliminate possible redundant entries in the list
         {
          3.3.1. IF an entry exists in the list that has never been
             matched, or incidentally the list contains an error, the entry
             is removed from the list
         }
   }
4. FOREACH node starting from the leaves towards the root of the graph
   {
      4.1. IF the sub-graph starting from the actual node can be reordered
         in reasonable time, according to (3), THEN the actual sub-graph is
         arranged into one branch with the brute-force algorithm.  (The
         amount of reasonable time is estimated based on measurements and
         it is heavily hardware dependent, consequently short enough to
         allow us to neglect the time needed for reordering the nodes.)
   }
5. FOREACH node starting from the leaves towards the root of the graph
   {
      5.1. <weight of the actual node>:= <the original number of matches it
         has received> + <the weight of the underlying branches divided by
         the distance from the actual node>
         }
6. FOREACH node of the graph
   {
      6.1. <$l_i$> := the leaf with the most significant weight;
      6.2. Move <$l_i$> to the end of the list;
      6.3. Remove the selected leaf from the list (if <$l_i$> was the last node
         of a branch, zero or more new leaves appear)
   }
7. Replace the current Access Control List in the router with the newly
   created one (this operation needs packet forwarding to be suspended
   for a very short period of time for security reasons).
```

After the execution of this algorithm we compared the results with the results produced by the previous, brute force method. But, since the brute force algorithm can only be executed for a small amount of list entries the comparison is valid only for a few entries. However, we also conducted measurements with the method mentioned in Section 3, and found that with periodic optimization (using the algorithm detailed above) we can decrease the delay resulting from the use of very long ACLs, whilst still keeping the time needed for the execution of our optimization process below a reasonable level.

As ACLs are usually defined once by a network administrator with respect to the given policies in the organization, and might be upgraded several times by hand, the process lacks any kind of feedback or optimization based on the actual traffic in the network. In contrast, our method monitors list entry hit rates according to the traffic and can modify the list and upload a new one if it shows to be faster. The measurements show that our algorithm can be executed in an insignificant time (not more than 1 second) below 3000 access list entries, which is typically enough for routers used by Internet Service Providers. Moreover the time needed for the optimization can be kept below 70 seconds even for 10000 list entries.

5 Application Results

Initially, we developed four different algorithms called A1-A4. Afterwards, we made a thorough comparison regarding their performance and efficiency and decided to use algorithm A4.

All four of the algorithms perform the following steps. A node is chosen at each step and transferred into the final re-ordered list. Algorithm A1 chooses nodes according to leaf weights. A2 evaluates paths towards a certain leaf while summarizing node weights at the same time. Similarly A3 evaluates paths, but this algorithm decreases node weights along the path according to the place a node has in the path, e.g. the weight of the fourth node in the path is divided by four and the weights are summarized. A4 four is very similar to A3, but this time a node weight is divided by two at the power of node place.

We generated list representations with rules in random initial order for testing the algorithms. Efficiency was calculated by comparing the resulting list of each algorithm to the initial list (4).

$Result_{original} = overall\ weight\ of\ the\ random\ generated\ list$

$Result_{algorithm} = overall\ weight\ after\ optimization$

$$Efficiency = 100 \cdot \left(\frac{Result_{original} - Result_{algorithm}}{Result_{original}} \right) \tag{4}$$

The scripts that generated random lists could generate random and balanced graphs also. Test runs were performed on the generated graphs with the four algorithms automated also by scripts. The averaged results are shown in the following two diagrams. Fig. 9 represents the efficiency of the algorithms as a function of the number of list entries that is the number of nodes in the graph.

The algorithms were also compared to the Brute Force method, in which case every possible layout of the graph is evaluated and the layout with the least weight is chosen.

According to the results, algorithm A4 performs the best, in most cases the same order is chosen as by the Brute Force method. A few deviations do exist, e.g. one pair of rules is different. However, this comparison was made only with short lists, since the Brute Force method has an unacceptable execution time (Fig. 10).

Fig. 10 shows the execution times as a function of list size. Algorithm A4 was chosen, because it is the most effective and it is fast too. Since even in core routers

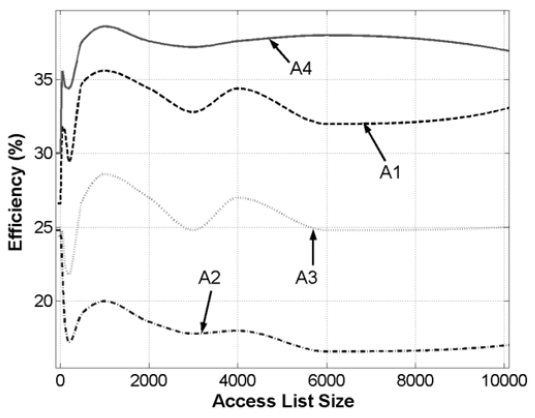

Figure 9: **Efficiency of the new algorithms**

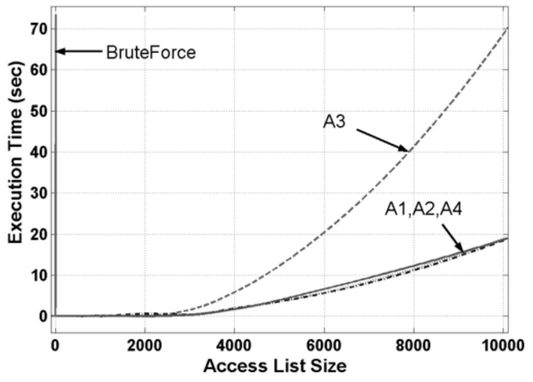

Figure 10: **Execution time of the new algorithms**

more than 10000 entries are quite rare, but few thousands are possible it is notable that algorithm A4 has an execution time of nearly zero till a few thousand entries.

6 Conclusions

Firstly, we have developed a method to measure the performance impact of network management with ACLs. Our measurement method uses regular PCs and it is a software-only solution. Compared to industrial solutions for the same problem, like the RouterTester from Agilent [10], it has similar capabilities combined with relatively cheapness and flexibility of a software solution. Using our test method it is possible to produce several streams to specified destinations, to test the functionalities of access lists. Detailed PDU (Protocol Data Unit) building is available as well. As during an ACL test, raw transmission performance of the tester is not the most important issue, since the performance impact of ACLs is measured instead of raw throughput capabilities. Our software solution satisfies the requirements of ACL testing. From the measurements, it can be concluded that the number and nature of access list entries have a significant impact on packet transmission in routers, so optimization might be needed.

In the second part of this work we proposed a method to optimize performance of access lists. The method uses directed and weighted graphs to represent ACL rules. We developed an algorithm to optimize the layout of the graph representing the ACL and this way to minimize the latency caused by access lists. Our software is implemented in C++ and Perl.

Our current work focuses on developing new, more efficient and faster algorithms to optimize ACLs, moreover we would like to examine the performance of ACLs using IPv6. Besides, we also would like to develop our method to be able to handle more general scenarios and to build a framework that is capable of examining more general list topologies applied for example in firewall systems or other software architectures as well.

References

[1] Scott Hazelhurst: A proposal for Dynamic Access Lists for TCP/IP Packet Filtering [Sortened Version In Proc. of SAICSIT 2001]

[2] S. Convery: Network Security Architectures [1rst edition, ISBN: 158705115X. Cisco Press (2004)]

[3] ETSI: Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; TTCN-3: Core Language [ETSI ES 201 873-1]

[4] NLANR: National Library for Applied Network Research. http://www.nlanr.net/

[5] Cisco: http://www.cisco.com/

[6] Scott Hazelhurst: Algorithms for Analysing Firewall and Router Access Lists [Workshop on Dependable IP Systems & Platforms, In Proc. ICDSN, June 2000]

[7] R Bryant: Symbolic Boolean Manipulation with Ordered Binary-Decision Diagrams [ACM Computing Surveys, 24(3) (September 1992)]

[8] Jeff Sedayao: Cisco IOS Access Lists [1rst edition, ISBN: 1-56592-385-5. O'Reilly]

[9] S. Palugyai, M. J. Csorba. Modeling Access Control Lists with Discrete-Time Quasi Birth-Death Processes. Submitted to the The 20th International Symposium on Computer and Information Sciences (ISCIS 2005)

[10] Agilent RouterTester: http://advanced.comms.agilent.com/RouterTester/