Modeling and Performance Evaluation of List Structures

PhD thesis

Sándor Palugyai

Advisors

dr. János Miskolczi
dr. Gyula Csopaki
dr. Sarolta Dibuz

Budapest, Hungary
2006
Abstract

These thesis groups deal 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.

The first thesis group presents a heuristic optimization of the rule entries of the ACLs according to the router traffic. I propose here an idea and a heuristic to solve an NP-hard problem, what is the optimization of these lists. I also introduce the other publications on this field.

After optimization I introduce the mathematical modeling of Access Control Lists. With the help of the model, important performance parameters can be derived from the Access Control Lists, such as overall latency and packet loss probability, which are affected by ACLs. Following examinations of the created model and following models for other types of list structures in the telecommunication are also presented in the second thesis group.

Since in telecommunication routers and personal computers the system interfaces play significant role on the performance parameters, I present the Markovian modeling of these interfaces in my third thesis group. For development an applicable model, some novelty have to be brought into the models. New ideas, state-space reductions and new algorithms for deriving the parameters are required by the applicable and calculable model.

All of my results are formally or with simulations proved, and deduction of the results are given in my thesis groups.
Acknowledgements

First of all, I would like to thank my advisors, dr. János Miskolczi, my direct advisor for his guidelines and for his help to find the proper scientific direction, dr. Gyula Csopaki for his valuable support from the TMIT Department, and dr. Sarolta Dibuz, who ensured the research possibilities at the Test Competence Center at Ericsson Hungary.

I also would like to thank dr. Miklós Telek and dr. Sándor Rácz for their significant ideas and help in the field of mathematics.

Last, but not least I would like to thank my colleague and friend Máté József Csorba, with who we worked and published together.
# Contents

Abstract                                                                                     ii  
Acknowledgements                                                                            iii  
Contents                                                                                      iv  
List of Figures                                                                              vi  
List of Tables                                                                               viii

## 1 Introduction

1.1 List-structures in telecommunication ........................................... 2
   1.1.1 Access Control List system ................................................. 2
   1.1.2 Access Control List variations ............................................ 6
   1.1.3 Other list-structures ...................................................... 7

## 2 Optimization of Routers using ACLs

2.1 Problem outline ................................................................. 13
2.2 Solution of the optimization problem ........................................... 14
   2.2.1 Definitions for the optimization ....................................... 14
   2.2.2 The algorithm of the heuristic ....................................... 19
2.3 Conclusion .............................................................................. 22

## 3 Modeling of Access Control Lists

3.1 The ACL filter model ............................................................. 25
   3.1.1 The models of ACLs ............................................................. 26
   3.1.2 Solution of the models ...................................................... 28
   3.1.3 Simulation results ............................................................. 34
3.2 Comparison of the finite and infinite model solutions ................................. 37
## CONTENTS

3.3 Modeling of hierarchical structures ........................................... 39
   3.3.1 The BSD filter list .................................................. 39
   3.3.2 Hierarchical lists ................................................... 42
   3.3.3 TTCN alternatives .................................................. 43
   3.3.4 Solution of the models .............................................. 44
   3.3.5 Simulation results .................................................. 44

3.4 Conclusion ................................................................. 44

4 Modeling affect of hardware interfaces ..................................... 46
   4.1 Affect of the interfaces on the overall delay ......................... 47
   4.2 The examined system .................................................. 48
   4.3 The interface model .................................................. 50
      4.3.1 The model ........................................................ 51
      4.3.2 The normalized interface model .................................. 56
      4.3.3 Solution of the model with approximation ....................... 60
   4.4 The punctual solution of the model .................................. 64
   4.5 State-space reduction ................................................ 68
      4.5.1 The idea of the state-space reduction .......................... 68
      4.5.2 The proof of the state-space reduction ......................... 69
      4.5.3 Affect of the reduction on the model .......................... 74
   4.6 Simulation results .................................................... 76
   4.7 Conclusion .............................................................. 78

5 Conclusion remarks ........................................................... 79
   5.1 Summary of thesis .......................................................... 79
   5.2 Future research .......................................................... 80

Abbreviations ................................................................. 81

Glossary of Symbols .......................................................... 82

References ................................................................. 84
List of Figures

1.1 An Access Control List (example) ...................................... 4
1.2 Example network .......................................................... 4
1.3 Rule sequence by the admin (example) .............................. 5
1.4 Reordered rule sequence .................................................. 6
1.5 BSD rule-groups ............................................................ 8
1.6 The BSD filter system ...................................................... 8
1.7 Hierarchical sub-groups ................................................... 9
1.8 Hierarchical filter systems ............................................... 9
1.9 Balanced Tree systems .................................................... 10
1.10 Illustration of the TTCNv3 alternative behavior .................. 11
1.11 TTCNv3 alternatives (example) ...................................... 12

2.1 Cycles in the constructed graph (example) ......................... 17
2.2 The graph representing the example and the optimization process . 18
2.3 Efficiency of the new algorithms ....................................... 21
2.4 Execution time of the new algorithms .................................. 22
2.5 Flowchart of the $A_4$ algorithm ....................................... 23

3.1 Logical representation of a DQBD process .......................... 27
3.2 A general PH-type renewal process .................................... 28
3.3 The infinite DQBD representing the logical behavior of ACLs .......... 32
3.4 The last levels of the finite DQBD model ............................. 33
3.5 Error of the model on the packet delay ............................... 35
3.6 Error of the model on the packet loss .................................. 35
3.7 Packet delay of the ACL .................................................. 36
3.8 Packet loss of the system ................................................. 36
3.9 The delay as a function of arrival intensity (example) .............. 36
3.10 Applicability of the infinite model .................................... 38
**LIST OF FIGURES**

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.11</td>
<td>The dependency of $y$ from the buffer size ($S_q$)</td>
<td>38</td>
</tr>
<tr>
<td>3.12</td>
<td>The DQBD model of the BSD filter system</td>
<td>40</td>
</tr>
<tr>
<td>3.13</td>
<td>The DQBD model of the Hierarchical filter system</td>
<td>42</td>
</tr>
<tr>
<td>4.1</td>
<td>The effect of the ACLs without interfaces</td>
<td>47</td>
</tr>
<tr>
<td>4.2</td>
<td>The delay refined with the effects of the network interfaces</td>
<td>48</td>
</tr>
<tr>
<td>4.3</td>
<td>The system under investigation</td>
<td>48</td>
</tr>
<tr>
<td>4.4</td>
<td>The interface model</td>
<td>49</td>
</tr>
<tr>
<td>4.5</td>
<td>Packet time-line without interrupts</td>
<td>50</td>
</tr>
<tr>
<td>4.6</td>
<td>Packet time-line with interrupts</td>
<td>50</td>
</tr>
<tr>
<td>4.7</td>
<td>The corresponding part of rule-behavior in the ACL model</td>
<td>53</td>
</tr>
<tr>
<td>4.8</td>
<td>The model-part of the interface delay</td>
<td>53</td>
</tr>
<tr>
<td>4.9</td>
<td>One example level of the interface model</td>
<td>54</td>
</tr>
<tr>
<td>4.10</td>
<td>The model of ACLs together with interfaces</td>
<td>55</td>
</tr>
<tr>
<td>4.11</td>
<td>Model structure, when $S_A \geq (S_q - 1)$</td>
<td>55</td>
</tr>
<tr>
<td>4.12</td>
<td>Model structure, when $S_A &lt; (S_q - 1)$</td>
<td>56</td>
</tr>
<tr>
<td>4.13</td>
<td>The structure of the normalized Interface model</td>
<td>57</td>
</tr>
<tr>
<td>4.14</td>
<td>Packets in the waiting queue</td>
<td>65</td>
</tr>
<tr>
<td>4.15</td>
<td>The triangle structure of delay calculation</td>
<td>66</td>
</tr>
<tr>
<td>4.16</td>
<td>Calculation order of the triangle structure</td>
<td>66</td>
</tr>
<tr>
<td>4.17</td>
<td>One level of the model and a drain state (example)</td>
<td>68</td>
</tr>
<tr>
<td>4.18</td>
<td>How to convert the input interface state-group ($IN$) into one state</td>
<td>69</td>
</tr>
<tr>
<td>4.19</td>
<td>Separation of the states in the two different cases.</td>
<td>70</td>
</tr>
<tr>
<td>4.20</td>
<td>Conversion of the input interface state-group ($IN$).</td>
<td>74</td>
</tr>
<tr>
<td>4.21</td>
<td>Calculation time of the model ($S_A = 5$)</td>
<td>75</td>
</tr>
<tr>
<td>4.22</td>
<td>Calculation time of the model ($S_A = 10$)</td>
<td>76</td>
</tr>
<tr>
<td>4.23</td>
<td>Accuracy of the model regarding the Delay</td>
<td>77</td>
</tr>
<tr>
<td>4.24</td>
<td>Accuracy of the model regarding the PacketLoss</td>
<td>77</td>
</tr>
<tr>
<td>4.25</td>
<td>The delay refined with the effects of network interfaces</td>
<td>78</td>
</tr>
<tr>
<td>4.26</td>
<td>The packet loss refined with the effects of network interfaces</td>
<td>78</td>
</tr>
</tbody>
</table>
## List of Tables

2.1 Notations for definition of the rule-weights .................................. 15  
2.2 Cycle in the ACL (example) ....................................................... 16  
2.3 The example ACL in the original order with match numbers .......... 17  
2.4 Notations for efficiency calculation ........................................... 20  

3.1 System variables of the list ......................................................... 29  
3.2 Model parameters of the list ....................................................... 29  
3.3 Calculation elements .................................................................. 34  
3.4 Notations for BSD modeling ......................................................... 41  
3.5 Notations for Hierarchical-list modeling ..................................... 42  

4.1 Packet processing time ............................................................... 50  
4.2 Interface system variables .......................................................... 51  
4.3 Interface model parameters ......................................................... 52  
4.4 Parameters of the delay calculation .............................................. 61  
4.5 Calculation parameters of the exact delay ................................... 65  
4.6 Sets of the state-separation ......................................................... 70  
4.7 Notations in connection with state-separation ............................. 71  

Chapter 1

Introduction

Nowadays, Internet usage is progressing at a great pace. More and more people become potential users and require faster connections. Security has also become an important issue in business and home networks, as well. As a result, such devices must be designed and created, which allow us to build and maintain a more secure network. Their operation must also be optimized.

Access Control Lists (ACLs) use sequential lists as data structures, and search through these lists by means of various search methods. ACLs are used in IP routers for providing network protection and access control. Access Lists are built from rules as list entries. Using these rules ACLs can filter the incoming/outgoing packets of the router. When a packet arrives into the router, the processor will match it against the rules in the list. For long lists execution-time plays a significant role and it is added to the packet delay. For these lists a waiting queue is also used. Because of the searching through the list and the waiting in the queue the packet delay may increase significantly caused by the Access Control List.

There are several other fields in telecommunication where sequential lists and their variations are also used. These fields are for example some database structures, programming alternatives or filter systems (like firewall) in line and mobil network systems equally.

My research was focused on the problems and possibilities of these kinds of lists and variations. The general aim was to find and solve open research problems in this field.

During the examination of ACLs I faced a significant proposition. I optimized the packet delay in these systems (Chapter 2) by reordering the list entries (rules). In this case simple analysis was enough and also mean values of packet delays were
adequate. However, I also want to derive performance indices, such as packet delay and packet loss more precisely. Predicting these kinds of performance parameters can be very important in testing and during system development to give a feedback to router hardware or software designers. Because of this fact I created and examined mathematical models of ACLs (Chapter 3). Apart from the sequential lists some types of Access Control Lists and other list structures also use other structures, e.g. hashes, balanced tree structures [1], etc. The replacement of these data structures might solve a few bottlenecks if they can be applied, but I want to model existing systems and their performance. These alternates are also dealt in Chapter 3. Through, when the examined system is loaded, queuing also appears at the system’s interfaces and that also has a serious impact on the system’s delay and loss [2]. Chapter 4 introduces the modeling of systems extended with the system interfaces. Connecting to the interface modeling I had to use some novelty which are also introduced in Chapter 4. These new ideas, state-space reductions and new algorithms for deriving the parameters are required by the applicable and calculable model.

1.1 List-structures in telecommunication

In this section the list-structures are introduced, that are used in the field of telecommunication. All of the mentioned novelties in this thesis is connected to these lists.

1.1.1 Access Control List system

Today, 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. Access Control Lists (ACLs) are a commonly used solution in firewall routers, border routers and in any intermediate router that needs to filter traffic [3].

ACLs are basically criterions 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 packet flow can be classified and managed by a router using ACLs. There are two basic types of ACLs: standard and extended [3],[4].

With standard IP access lists, a router is capable of filtering the traffic based on source addresses only. A typical rule, in the syntax of the Cisco Internetwork
Operating System (IOS) [5],[6] e.g.:

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

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 [3].

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]

An example for a quite simple and short Access Control List is depicted in Figure 1.1.

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. 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.

One of the application areas 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 [7].

For the sake of clarity, consider the following example (Figure 1.2). 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.

The solution to the problem introduced above is realized by the ACL numbered 111, which contains two separate lines (Figure 1.3) and an implicit one. The first one
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.

```
access-list 111 permit tcp any host 10.120.23.1 eq 80
access-list 111 deny any 10.120.23.0 0.0.0.255
{access-list 111 deny all} {implicit}
```

Figure 1.3: Rule sequence by the admin (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.

Applying the ACL on interface Ethernet0/0:
```
Router1(config-if)# ip access-group 111 in
```

If Figure 1.4 is seen, the importance of the order of list-rules can be realized. In this case, the order of the two specific rules are changed (compared to Figure 1.3). In Figure 1.3 the HTTP on a specific host is accepted (first rule), and everything else is denied (second rule). But in Figure 1.4 with the first rule a whole netmask is denied, so in the second rule there will not be any matches. So, we cannot reorder the rules of list without considering the dependencies in the list.

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 [4],[8].
1.1.2 Access Control List variations

There are some Access Control List types, what also provide other features, such as timing or dynamic access.

Time-Based ACLs

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 (Internet Service Provider). Service Level Agreements (SLAs), negotiated in advance, can be satisfied as well if time ranges are also specified in an access list [9].

Dynamic ACLs

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 [10].

The Null0 interface

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 [3].

access-list 111 deny any 10.120.23.0 0.0.0.255
access-list 111 permit tcp any host 10.120.23.1 eq 80
{access-list 111 deny all} {implicit}
1.1.3 Other list-structures

In the field of telecommunication we can meet other structures, that have similar behavior as Access Control Lists. These structures are introduced in this section. All of these systems use some kind of list-structure for filtering or searching in a rule-sets. There are some, which use sequential lists, like the Access Control Lists, and also there are some, which use other structures, like binary-trees or hierarchical structures.

Ipchains and Iptables

The generally used filter structures by LINUX systems are the *ipchains* and *iptables*. Working of these structures are highly similar to the behavior of the Access Control Lists, but they are using an other type of command syntax [11],[12].

BSD Packet Filter

The BSD systems, like FreeBSD [13] or OpenBSD [14] use a little trick in the searching list structures.

The BSD filter system is called to BPF (Berkeley Packet Filter), which use an efficient packet capture method [15]. Here every packet is compared against the filter rule set. The rule set consists of a linked list of rules. Each rule contains a set of parameters that determines the set of packets the rule applies to [16]. The parameters may be the source or destination address, the protocol, port numbers, etc. For a packet that matches the rule, the specified pass or block action is taken. Block means that the packet is dropped by the filter, and pass means that the packet is forwarded to its destination.

Although alternative containers like hash tables allow searches in constant time, they also have their drawbacks. Hash tables have a fixed size by definition. As the number of entries grows, collisions occur (if two entries have the same hash value) and several entries end up in the same hash bucket. An attacker can trigger the worst case behavior by opening connections that lead to hash collisions and state entries in the same hash bucket. To accommodate this case, the hash buckets themselves would have to be binary search trees, otherwise the worst case behavior allows for denial of service attacks. The hash table would therefore only cover a shallow part of the entire search tree, and reduce only the first few levels of binary search, at considerable memory cost.
So BSD systems use an other structure for decreasing the searching time. The packet filter in BSD automatically optimizes the evaluation of the rule set. If a group of consecutive rules all contain the same parameter, e.g., source address equals 10.1.2.3, and a packet does not match this parameter when the first rule of the group is evaluated, the whole group of rules is skipped, as the packet can not possibly match any of the rules in the group [1]. It can be said, that BSD collect the similar rules of the list and make groups from them. General rules are created for these groups. At the beginning of the search on the list, the packet chooses first the proper group-rule, and after choosing it searches through the list-entries in the group. This searching method is depicted in Figure 1.6. In this example in Figure 1.5 three rule-groups are given and they contain $K$, $L$ and $M$ rule-elements respectively.

![Figure 1.5: BSD rule-groups](image)

Based on Figure 1.5 the structure of the BSD filter system can be described by Figure 1.6.

![Figure 1.6: The BSD filter system](image)

With this trick the performance of the searching in the list can increase seriously [1].
Hierarchical structures

In general case the BSD trick can be improved and generalized. The BSD create groups from the rules, but it use only one hierarchical level. It is possible to create other sub-groups inside one group and so on. This theory is depicted in Figure 1.7.

Based on Figure 1.7 the structure of a hierarchical filter system can be described by Figure 1.8.

Theoretically in these structures infinite number of hierarchical levels can be created.
Balanced Tree structure

An example for hierarchical list structures is the Balanced Tree structure. Balanced Tree is in fact a special type of hierarchical list, where every group in the structure has maximum two sub-groups, as it is depicted in Figure 1.9.

![Figure 1.9: Balanced Tree systems](image)

This structure is very important in telecommunication and it is widely used. For example in [17] and [18].

TTCNv3 testing language

TTCNv3 uses a structure which also can be described by the hierarchical list-structures, but it has some specialities.

TTCNv3 is the standardized test specification language, the Testing and Test control Notation version 3 [19]. TTCNv3 is widely used in different areas of testing including different fields of telecom and datacom and is gaining acceptance even in automotive systems testing. The language itself has a variety of applications including testing various systems for interoperability, robustness and conformance. Recently, there has been a sore need to assess the capabilities of TTCNv3 as a testing solution not only for conformance and interoperability testing but for performance evaluation of telecommunication systems as well. Since TTCNv3 is a rather high level specification language, concerns have arisen regarding its applicability in load tests that require a significant amount of processing power, e.g. a high number of packets per second generated. On the other hand, tests and numerous pioneer projects show us that the language is capable of working at the edges of what the underlying hardware is capable of. Besides, emerging real-time extensions to the original notation also exist [20], [21], [22]. So, usage of TTCNv3 in load tests should
not imply a bottleneck if tests are designed carefully. For this careful design we have
to handle the possible events inside the test components well.

In TTCNv3, the *alt* statement is used to handle events possible in a particular
state of the test component. The *alt* statement uses a so-called snapshot logic [23].
This means, that before evaluating the actual alternatives in the *alt* a snapshot
of the test component containing any information that is relevant (e.g. status of
the test ports involved in the *alt*, running timers is taken). Branches of the *alt*
might have a Boolean guard expression assigned to them that is evaluated before
the branch is examined. The guard expression might be based on the snapshot as
well. A complex form of behavior is where sequences of statements are expressed
as sets of possible alternatives to form a tree of execution paths, as illustrated in
Figure 1.10.

![Figure 1.10: Illustration of the TTCNv3 alternative behavior](image)

Different types of branches may exist in an *alt* statement (e.g. timeout, receive)
as it is shown in Figure 1.11. Each time a receiving branch is found during execution
a matching operation is done first. In case the incoming message, that is the first in
the corresponding FIFO, matches the criteria the *alt* branch will be executed and
the message will be removed from the top of the queue. Otherwise, the execution
continues and the next branch will be examined. However, execution does not stop
after a snapshot was taken, so the state of the test component and the queues
assigned to it might change in between. However, these events do not change the
actual snapshot, until the *alt* statement is not executed again.

From the description of the behavior of TTCNv3 alternatives and from Figure
1.10 and Figure 1.11 it can be seen that this *alt* structure is really a kind of
hierarchical list structure.
EXAMPLE:

```c
// Use of nested alternative statements

alt {
    [] l1.receive(DL_REL_CO:*) {
        setverdict(pass);
        T&AC.stop;
        TNOAC.start;
    }
    [] l1.receive(DL_EST_IN) {
        TNOAC.stop;
        setverdict(pass);
    }
    [] T&AC.timeout {
        l1.send(DEL_EST_RQ:*)
        T&AC.start;
    }
    alt {
        [] l1.receive(DL_EST_CO:*) {
            ThC.stop;
            setverdict(pass)
        }
        [] T&AC.timeout {
            setverdict(inconc);
        }
        [] l1.receive {
            setverdict(inconc)
        }
    }
    [] l1.receive {
        setverdict(inconc)
    }
    [] T&AC.timeout {
        setverdict(inconc)
    }
    [] l1.receive {
        setverdict(inconc)
    }
    }
```

Figure 1.11: TTCNv3 alternatives (example)
Chapter 2

Optimization of Routers using ACLs

[PalugyaiJ3], [PalugyaiC3]

In this section an optimization heuristic of Access Control List systems and its description structure is introduced. According to the transit traffic on the list, the rules of the list have an optimal order. After I made some definitions according to the described system, I created a build up method, which helps us to assign the optimized rule order in the list. With the help of this method I present algorithms which approach this optimal rule order well.

Optimization of the presented lists is an NP-complete problem, because of the dependencies between the rules. As a result we need a heuristic method, which is approach the optimal order well, but it is much faster. Besides, telecommunication routers have low resources. Thus we need a fast and efficient algorithm, which does not use many system resources. Chosen of the best algorithm is also introduced in this section. The algorithms were compared focusing on the efficiency and execution time.

2.1 Problem outline

It is a very important task, albeit largely unaddressed until now, to optimize filter rules in routers [24]. As it is mentioned in Section 1.1.1, the existing of these lists largely affect the performance of a router. It was also mentioned, that the rule order cannot be simply reordered according to rule weights, because of the dependencies
CHAPTER 2. OPTIMIZATION OF ROUTERS USING ACLS

The rules. With the help of the optimization of the rule order in the list, we can decrease the mean time packets spent in these lists.

As it is mentioned above, the rules of the list have an optimal order, according to the transit traffic on the list. This transit traffic can be observed, measured, analyzed and can be predicted. Articles discuss for example general internet traffic measurement. One of these is the NLANR project [25]. This project collected a huge amount of traffic and data was analyzed. From these data the general traffic on the internet can be predicted. In the case of a specific router the transit traffic also can be measured and predicted. With the help of the predicted traffic the rule order in the filter list can be optimized.

There are some discussions and articles which are dealing with this problem already, but a good algorithm is not proposed yet [24],[26]. The Cisco System [27] has some proposals to make the list order better, but these proposals only use simple ideas, e.g. if a rule is used more, than it should be moved to the beginning of the list [2].

It is proved, that this optimization problem is NP-complete [24]. It is a huge problem, because telecommunication routers have low resources, like memory, processor speed and other hardware capacities [27]. Since, solving this problem is necessary to speed up the network traffic, however the problem is NP-complete, so we need a heuristic [24].

In the following thesis group I propose a structure and an algorithm, which approach the optimal list order well. In the same time it is fast enough, and it does not need so many system resources. It also does not harm the dependencies between the list rules.

2.2 Solution of the optimization problem

[PalugyaiJ3],[PalugyaiC3]

2.2.1 Definitions for the optimization

In real case scenarios it is general, that a new packet arrives into the system before the actual packet departures. This cause queuing in the examined systems. The overall delay on a given list in case of a given traffic can be correctly assigned, when we measure the intensity of the traffic and queuing processes also.
Table 2.1: Notations for definition of the rule-weights

<table>
<thead>
<tr>
<th>symbol</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$L$</td>
<td>the actually given list representation</td>
</tr>
<tr>
<td>$T$</td>
<td>traffic, which going through the list $L$</td>
</tr>
<tr>
<td>$n(L)$</td>
<td>size of the list $L$</td>
</tr>
<tr>
<td>$r_i(L)$</td>
<td>delay of the list-rule $i$ (independent from other rules)</td>
</tr>
<tr>
<td>$m_i(L, T)$</td>
<td>matching probability on the rule $i$ of list $L$ in case of traffic $T$</td>
</tr>
</tbody>
</table>

But, for the comparison the delay of different rule orders in the same list in case of the same traffic the punctual delay is not needed. So, I defined a ratio for comparing rule orders. This ratio is used for my optimization. As a matter of fact this ratio is the rarely given case, when the traffic is so slow, that demands are not queued in the waiting queue.

The situation is more difficult, since in general the delay in different rules is not equal. To define this ratio I used the notations in Table 2.1

I defined a ratio number to be able to compare the delay of a given traffic $T$ and list $L$ in case of different rule orders. In this way I examined the goodness of lists according to its latency.

**Definition 2.1.**

Let $d_i(L)$ be the full delay of demand on list $L$, which matches against the rule $i$ in list $L$ and before rule $i$ it does not matches against any rule [PalugyaiJ3],[PalugyaiC3].

\[ d_i(L) = \sum_{k=1}^{i} r_k(L) \]  

(2.1)

**Definition 2.2.**

Used the expression (2.1), let the overall latency on list $L$ in case of traffic $T$ be defined in (2.2), if there is no queuing [PalugyaiJ3],[PalugyaiC3].

\[ \text{Delay}(L, T) = \sum_{i=1}^{n(L)} m_i(L, T) \cdot d_i(L) = \sum_{i=1}^{n(L)} m_i(L, T) \cdot \sum_{k=1}^{i} r_k(L) \]  

(2.2)

Be this ratio the weight of the specified rule $i$ in the list $L$. 
In the following I present, how I defined the dependencies between list-rules, the direction of these dependencies and the redundancy of the list. Dependencies show us which rules cannot be reordered in the list.

**Definition 2.3.**

*Two list-rules depend from each other, if they have common parts, so there are demands that would match against both of them. Point the direction of the dependency into the first rule in the list from these two rules, according to (2.3) [PalugyaiJ3],[PalugyaiC3].*

\[
A \leftarrow B \text{ if } \exists C, \text{ that } C \subseteq A \text{ and } C \subseteq B \tag{2.3}
\]

With the help of the definition of redundancy we can delete rules from the list. As a result, we can decrease the size of the list.

**Definition 2.4.**

*A rule in the list is redundant, if it directly depends on an other rule and the direction of the dependency points into the redundant rule. This redundancy is true if and only if the dependency generator rule is not in indirect dependency with the redundant element, and the element is the full-subset of the dependency generator element, as it is described by (2.4) [PalugyaiJ3],[PalugyaiC3].*

\[
A \not\Rightarrow \text{ if } A \leftarrow C \text{ directly, and not } \exists B, \text{ that } A \leftarrow B \leftarrow C \tag{2.4}
\]

Definition 2.4 means, that in the example in Table 2.2 and in Figure 2.1 there is no redundant element. Rule 1 \(\leftarrow\) directly from rule 3, but they are also in an indirect connection through rule 2 (1 \(\leftarrow\) 2 \(\leftarrow\) 3), which cannot be deleted. So, there is no redundant element in the example list.

<table>
<thead>
<tr>
<th>No.</th>
<th>Rule</th>
<th>Source address</th>
<th>Destination address</th>
<th>Protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>deny</td>
<td>10.120.0.24</td>
<td>10.200.20.55</td>
<td>any</td>
</tr>
<tr>
<td>2</td>
<td>permit</td>
<td>10.120.0.24</td>
<td>10.200.20.0/24</td>
<td>any</td>
</tr>
<tr>
<td>3</td>
<td>deny</td>
<td>10.120.0.0/24</td>
<td>any</td>
<td>any</td>
</tr>
</tbody>
</table>

Using the definition of dependency (Definition 2.3) we can depict the rules of the example as a directed graph (Figure 2.1).
In Definition 2.4 I described the elements, which will never match against any demand, because all of the packets match against the rule, which is its redundancy generator rule.

**Thesis 2.1.**

*Based on my definitions (Definition 2.1-2.4) I created a structure, which describes the dependencies and connections between the rules of the list, and minimizes the redundancy in the structure. This structure is based on directed graphs which also may contain cycles in it [PalugyaiJ3],[PalugyaiC3].*

In my structure the list-rules will be graph nodes, dependencies will be arrows and matching numbers will be node weights in the directed graph.

To see the construction of this structure, let a short example be introduced. In Table 2.3 I give a short Access List as an example.

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

The example list is constructed considering the following parameters: type of
CHAPTER 2. OPTIMIZATION OF ROUTERS USING ACLS

Figure 2.2: The graph representing the example and the optimization process

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. These node are depicted in the directed graph in Figure 2.2. In this quite simple example I 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 [28] and UDP [29] port numbers or protocol IDs.

From this representation of the access list, the graph in Figure 2.2 can be constructed identifying which network prefix is more general and contains the other prefix as well. The nodes are labeled with the capitals A..O each one representing a single rule (one line of the list).

In Figure 2.2 I created the minimal directed graph with weights in the 1. and 2. steps. The 3. step (the third arrow) in Figure 2.2 is the optimization method, which is discussed in the next subsection.

During constructing the minimized, directed graph (Figure 2.2) from the original ACL rule-set (Table 2.3) I made the main steps what are depicted in Figure 2.2. After labeling the rules with capitals (A..O) I marked the meaning of the rules with white (deny) and gray (permit). In the first main step I built up the directed graph using the Definition 2.3 and I assigned weights to nodes of the graph according to Definition 2.2. In the second main step I eliminated the redundant nodes from the
graph according to the Definition 2.4. This step uses the original list order and the
pre-constructed directed graph, as it is shown in Figure 2.2. 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 (node
A and O in Figure 2.2). 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.

I optimize extended ACLs, so each rule may contain port, protocol and address
related constraints. The graph representing an extended ACL may contain cycles,
as it is demonstrated in the basic rule-set in Table 2.2.

2.2.2 The algorithm of the heuristic

In the third step of the Figure 2.2 I optimize the rule-order according to the con-
structed directed graph and weights (Definition 2.2), but I do not harm the depen-
dency directions in the constructed graph.

To find the best heuristic, I developed and examined four different algorithms
and improved them, according to its performance. I have chosen the best algorithm
from them by measurements, where I examined the execution time and the efficiency
of the algorithms.

At first, I have developed a brute force algorithm to optimize the graph repre-
senting 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 (Figure 2.4).

The graph optimizing processes were implemented in C++, because of the com-
putationally intense calculations. Besides, the on-line communication with the router is implemented in *Perl* language [30] and uses a Telnet [31] connection which uses TCP (Transmission Control Protocol). 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.

**Thesis 2.2.**

*I created algorithms, based on the constructed directed graph and weights (Thesis 2.1), which can be used to reduce the packet delay on the Access List. These algorithms are heuristics based on approximated traffic. I have chosen the best of them by experimental measurements according to execution time and efficiency [PalugyaiJ3],[PalugyaiC3].*


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 is very similar to A3, but this time a node weight is divided by two at the power of node place.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Res_{orig}$</td>
<td>overall weight of the random generated list</td>
</tr>
<tr>
<td>$Res_{alg}$</td>
<td>overall weight after optimization</td>
</tr>
</tbody>
</table>

I 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 (2.5) according to the notation of Table 2.4.

$$Efficiency = 100 \cdot \left( \frac{Res_{orig} - Res_{alg}}{Res_{orig}} \right)$$ (2.5)
I wrote many scripts that generate random lists, which 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, after many measurements, are shown in the following two diagrams. Figure 2.3 represents the efficiency of the algorithms as a function of the number of list entries that is the number of nodes in the graph.

![Image of Figure 2.3: Efficiency of the new algorithms](image)

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 (Figure 2.4).

Figure 2.4 shows the execution times as a function of list size. Algorithm A4 was chosen, because it is the most effective and it is fast as well. Since even in core routers 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.

After comparison the algorithms I have chosen the A4 algorithm (Figure 2.5), which is the fastest and which has the best efficiency.

Using this algorithm, according to (2.2), the example input ACL (in Table 2.3) has a delay ratio of 192381. The resulting ACL (in Figure 2.2) has a ratio of 144840. These values do not have a unit, since they represent a ratio only for comparison.
2.3 Conclusion

I have developed a method to measure the performance impact of network management with ACLs. My measurement method uses regular PCs and it is a software-only solution. Compared to industrial solutions for the same problem, like the Router-Tester from Agilent [32], it has similar capabilities combined with relatively cheapness and flexibility of a software solution. Using my 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. My 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 this section I proposed definitions and a method to optimize performance of Access Control Lists and other filter lists. The method uses directed and weighted graphs to represent ACL rules. I developed a structure and an algorithm to optimize the layout of the graph representing the ACL and this way to minimize the latency caused by Access Lists. My software is implemented in C++ and Perl.

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 manually, the process lacks any kind of feedback or optimization based on the actual traffic in the network. In contrast, my method monitors list entry hit rates according to the traffic and can modify the list. Also, it can upload a new list if it proves to be faster. The measurements show that my 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.

This solution can be used to decrease the latency in our routers, but it cannot calculate and foretell exact performance parameters of the examined system, because it uses only ratio numbers for optimizing. In the followings I will discuss exact performance modeling of these kinds of systems.
Chapter 3

Modeling of Access Control Lists

In the previous chapter I optimized the packet delay in ACLs and filter lists (Chapter 2) by rearranging the list entries (rules). In this case simple analysis was enough and also mean values of packet delays were adequate. However, I also want to derive performance indices, such as packet delay and packet loss more precisely. Foretelling these kinds of performance parameters can be very important in testing and during system development to give a feedback to router hardware or software designers. Because of this fact I created and examined mathematical models of ACLs. Apart from the sequential lists some types of Access Control Lists and other list structures also use other structures, e.g. hashes, balanced tree structures [17],[18], etc. Modeling of these alternate structures are also discussed in this chapter. The replacement of these data structures might solve a few bottlenecks if they can be applied, but I want to model existing systems and their performance.

In this chapter I present own mathematical models that can describe Access Control Lists and other filter systems (Section 3.1). I introduce precise and approximate models too, and I examine the applicability of the approximate model (Section 3.2). Here I give a limitation for using this model. In Section 3.2 I also deal with the modeling of other topologies, what are similar to the filter lists, or they are some kind of extended filters lists. These are the filter list of the system BSD, the TTCN alternatives and hierarchical list structures.

For examining the models what I created, I made lots of simulations. I proved the correctness of my models by simulation results at the end of this chapter.
For modeling ACL systems some mathematical solutions have been examined. E.g. M/G/1 type queues [33] to build a simple model of ACLs only, without considering the effect of the interfaces. Using this kind of description we can calculate the delay of arriving requests, but only in case of infinite system buffer. During modeling a real router I do not restrict my investigations in this manner. My approach uses finite queues, which is more realistic. With the M/G/1 solution we can neither calculate state-distributions, nor packet loss parameters.

The proposed model uses another Markovian structure, the well known Discrete-time Quasi Birth-Death (DQBD) process [34]. A DQBD can be described and solved in a matrix-geometrical way [35],[36]. With the aid of this type of process we can also calculate parameters that M/G/1 type queues are unable to handle, e.g. state-distribution or the effect of a finite buffer. Using DQBDs the model can derive performance parameters, like packet delay and packet loss ratio, from a parameterized ACL system. In my model parameters are the ACL rule-system together with the matching probabilities, interface parameters and the description of the incoming traffic. The input/output traffic can be described by a Markovian Arrival Processes (MAP) [37] or by a Phase-Type (PH) distribution [38], which are widely used in the field of Markovian performance modelling.

My method uses a mathematical model that can be evaluated and performance indices of the described system can be derived by existing solver techniques. The mathematical analysis in turn, is based on matrix-geometric solution techniques and matrix analytic methods [35]. In order to become acquainted with DQBD processes let us consider processes $N(t)$ and $J(t)$, where $\{N(t), J(t)\}$ is a DTMC (Discrete-Time Markov Chain). The two processes have the following meaning: $N(t)$ is the level process and $J(t)$ is the phase process. $\{N(t), J(t)\}$ is a DQBD if the transitions between levels are restricted to one level up or down, or inside the actual level, and we use a discrete time slot ($TU$). The structure of the transition probability matrix $P$ of a simple DQBD is in Equation 3.1. This transition probability matrix structure describes infinite Markov-chains, what are used for modeling infinite buffer sizes in telecommunication.
Matrix $A_0$ describes the demand arrivals in the model (transitions one level up), matrix $A_1$ describes transitions inside each level and matrix $A_2$ describes departures (transitions one level down). Matrix $A_1^*$ is an irregular transition matrix at level 0. The row sum of $P$ is equal to 1 (discrete-time model). The tangible meaning of levels and phases can be seen in Figure 3.1.

The two-dimensional property of a DQBD is utilized by my method significantly. On one hand, I map the sequential list checking mechanisms of the system I model, into the internal phases inside each level of the DQBD. Besides, I use the property of the system that has a relevant impact on the performance issue I want to investigate. In the example of ACLs internal phases represent the step-by-step list checking mechanism.

On the other hand, levels in the DQBD are assigned to properties that have a straight impact on the performance parameter we want to estimate. These properties typically include message sizes, buffer sizes, numbers of requests in queue, etc.

After the system's behavior is described by the DQBD model, with the aid of matrix analytic methods the following properties of the model can be calculated: steady state solution, queue length in steady state, phase distribution, mean value of time to transition back to level zero. As a result, a variety of performance indices can be derived from the DQBD model, such as packet loss ratios, delays, quantiles, and available throughput under stress conditions.
For wider information about Markovian modeling check the connected books and papers, like [33],[34],[35],[38], [39], etc..

3.1.1 The models of ACLs

In my behavioral model the transitions inside a given level of the DQBD process describe the functionality of the Access Control List of the router. The intra-level transitions represent the packet-matching mechanisms and can be described with a phase type process. In general a PH-type process [39] has a designated state, called the drain. In Figure 3.2, a general PH-type renewal process can be seen.

$$T$$

Figure 3.2: A general PH-type renewal process

In Figure 3.2, matrix $T$ describes transitions before entering into the drain and vector $t$ contains the probabilities of the process entering the drain. With my model it is possible to model the original ACL conception as well as grouped lists, quick search algorithms, hashes and other ACL implementations. In this section I present the model for the original sequential Access Control Lists.

Thesis 3.1.

I created mathematical models (based on my model parameters in Table 3.2), which can describe the ACL and other filter structures, and we can predict performance parameters from it. An infinite model is created (Figure 3.3 and Equations 3.5-3.10), which can be solved relatively fast, but it can only approximate the performance parameters. I also developed a finite model (Figure 3.4 and Equations 3.5-3.10 and 3.14), which is slower, but more accurate [PalugyaiC5],[PalugyaiLNCS1].
In my model all of the system variables, like input traffic or list size are parameters. Let the system be described by the system variables in Table 3.1.

### Table 3.1: System variables of the list

<table>
<thead>
<tr>
<th>symbol</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F_{tA}$</td>
<td>distribution of the inter-arrival time of incoming packets</td>
</tr>
<tr>
<td>$F_{LACL}$</td>
<td>distribution of matching time on the rules (list entries)</td>
</tr>
<tr>
<td>$N$</td>
<td>size of the ACL</td>
</tr>
<tr>
<td>$p$</td>
<td>matching probabilities of the list-entries (1 x $N$)</td>
</tr>
<tr>
<td>$Sq$</td>
<td>size of the waiting queue</td>
</tr>
</tbody>
</table>

With the help of the system variables the whole system can be described. The model parameters are built up using system variables. For example, from the size of the list ($N$) and from the rule matching probabilities vector $p$ I created matrix $T$. This matrix describes the possible behavior of the given list. The model parameters that are built up from the system variables are listed in Table 3.2.

### Table 3.2: Model parameters of the list

<table>
<thead>
<tr>
<th>symbol</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$TU$</td>
<td>time-unit: the smallest match time (the discrete time-slot)</td>
</tr>
<tr>
<td>$D_0, D_1$</td>
<td>description of the incoming MAP traffic</td>
</tr>
<tr>
<td>$T$</td>
<td>Phase-type description of the internal behavior of the list ($N$ x $N$)</td>
</tr>
<tr>
<td>$\tau$</td>
<td>rate vector to the absorbing state (list model) (1 x $N$)</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>initial distribution in the list model-part ($N$ x 1)</td>
</tr>
</tbody>
</table>

From the description of the incoming traffic, list size and rule-match probabilities I created an infinite Markovian chain. Routers use finite buffers, but it is also possible to approximate the finite behavior with an infinite model as well. In this model the states represent the rules of the ACL and model levels represent the packets in the buffer.

I use Discrete-Time Markov chains by choosing a specific time slot $TU$. The model describes processes that happen in the router using ACLs during $TU$. I have chosen $TU$ to be the time what is needed to examine a packet, whether it matches against a rule or it does not. This time period is considerably short and hardware dependent. The user of the model can get this parameter from measurements on a specific hardware platform or from other mathematical models.
With the help of the system variables (Table 3.1) I built up my model, I created the structures that are mentioned in Table 3.2. Matrix $T$, vector $t$ and $\alpha$ can be seen on (3.2),(3.3) and (3.4), where notations are independent from the general PH-type process depicted in Figure 3.2.

$$T = \begin{bmatrix}
0 & p_1 & 0 \\
0 & 0 & p_2 & 0 \\
0 & 0 & \cdots & 0 \\
0 & 0 & p_{N-1} \\
0 & 0 & 0 & 0
\end{bmatrix} \quad (3.2)$$

$$t = \mathbb{1} - T \cdot \mathbb{1} = \begin{bmatrix}
1 - p_1 \\
\cdots \\
\cdots \\
1 - p_{N-1} \\
1
\end{bmatrix} \quad (3.3)$$

The initial phase distribution (Equation 3.4) has the simple meaning, when a new packet arrives it is checked against the first list entry with probability of 1.

$$\alpha = \begin{bmatrix}
1 & 0 & \cdots & 0
\end{bmatrix} \quad (3.4)$$

According to (3.2) and (3.3), $1 - p_i$ is the probability of the examined packet matching against rule number $i$ in the Access Control List. The PH-type process enters the drain means that we have a departure. To be precise, a departure in my models physically means that the examined packet is handed over to the routing process (or to the outgoing interface) and will be forwarded or is simply dropped by the router depending on the meaning of rule number $i$ in the ACL. The routing process itself, and the routing table lookup is out of the scope of my models.

Using the notations of the defined model parameters (Table 3.2), I described ACLs with an infinite DQBD model for approximation. The transition matrix in this case is constructed according to (3.1), but in my model the whole first model level is irregular ($A^*_0$, $A^*_1$, $A^*_2$), not only the first element. Here I created the necessary blocks as it is described in (3.5)-(3.10).

$$A^*_0 = D_1 \otimes \alpha \quad (3.5)$$
I described the packet arrival with a Markov Arrival Process (MAP), where matrix $D_0$ describes phase transitions without arrival and matrix $D_1$ describes phase transitions with exactly one arrival. The $\otimes$ denotes the Kronecker product. The values of $D_0$ and $D_1$ can be calculated from the arrival intensity. For example in the case of discrete Poisson process as an input traffic with parameter $\lambda$, the matrices $D_0$ and $D_1$ are scalars (Equation 3.11 and 3.12).

$$D_0 = (1 - \lambda \cdot TU)$$

$$D_1 = (\lambda \cdot TU)$$

With the given structures I fully defined my system. The graphical representation of the infinite model, which describes the Access List part of the system is depicted in Figure 3.3.

For easier explanation during graphical presentation of the model (in Figure 3.3) I used a special type of process for input traffic, the Poisson process (exponential inter-arrival time distribution with parameter $\lambda$). In Figure 3.3, a packet either matches a rule entry or it does not and moves to the next entry during a time step. In the first case we have a departure and get the next packet from the waiting queue. So we will have one packet less in the queue. The next packet starts to be examined from the first list-entry. In the second case, when packet did not match against a specific rule, model steps to the following rule. Also during this time-step
a new packet can arrive into the system. With the combinations of the arrival and matching probabilities the model can be built up. In the proposed model there are two special conditions. When system examines the actual packet against the last rule the result has to be a match, because of the implicit permit or deny rule as it is described in the Section 1.1.1.

After approximating a finite router buffer and the effect of ACLs with an infinite DQBD, I also developed a finite model to derive more punctual results from it.

The finite model has the same levels, phases and overall structure as the infinite one, but the number of levels is restricted to $S_q$. This means, that when the model is in the last model-level (waiting queue is full) and a new packet arrives, the system has to drop this packet. When there are $i$ packets in the system, the DQBD is on level $i$, accordingly the finite $S_q$ represents the buffer size of the router and the DQBD has a finite number of levels. In this case, the graphical representation of the final levels is depicted in Figure 3.4, in case of Poisson traffic.

Consequently, the block structure of the transition matrix becomes the one in Equation 3.13.
In addition to the matrices I mentioned above I will have a new irregular matrix in my structure. $A_{1}^{**}$ in our transition matrix is built up as described in (3.14).

$$A_{1}^{**} = D_{0} \otimes T + D_{1} \otimes (T + t \cdot \alpha) \quad (3.14)$$

With this definition I successfully defined the whole finite DQBD model, which can describe the introduced Access Control List systems and other filter systems. With the model performance parameters can be calculated from the ACLs.

Now we have an infinite model and a finite one. The infinite can be calculated faster, but the finite model gives more punctual results. With an already existing method, called PhFit method, the state-space of the model can be reduced. In this case the error of the model increases [40].
3.1.2 Solution of the models

With the help of the proper description matrices (what I already created), the important performance parameters can be calculated from the model. I only need the starting expressions of the needed performance parameters. In the case of the presented finite model, the mean value of the packet delay in the model can be calculated according to (3.15).

\[
E(\text{fullDelay}) = \Pi_0 \cdot 1 \cdot m + m \cdot E(N) + \sum_{i=1}^{S_q} \Pi_i \cdot (I - T)^{-1} \cdot 1 \quad (3.15)
\]

The probability of the packet loss can be calculated based on the Equation 3.16.

\[
P(\text{packetLoss}) = \Pi_{S_q} \cdot 1 \quad (3.16)
\]

From these starting expressions ((3.15) and (3.16)) the exact performance parameters can be numerically calculated with the help of existing solving technics (e.g. in [33],[34] and [35]). The elements of (3.15) and 3.16 in Table 3.3 can be calculated by these technics.

<table>
<thead>
<tr>
<th>symbol</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>\Pi_i</td>
<td>steady state in level i</td>
</tr>
<tr>
<td>m</td>
<td>spent time inside a level</td>
</tr>
<tr>
<td>E(N)</td>
<td>mean value of the packet number in the waiting queue</td>
</tr>
</tbody>
</table>

3.1.3 Simulation results

In order to validate the numerical accuracy of the computational solutions and to get a feedback whether the assumptions made in the modeling phase were right I developed a simulation of ACLs in NS-2 [41]. The simulation was made without any assumptions or approximations and uses finite buffers to simulate real world ACL behavior. The comparison shows the same packet loss and delay values as the DQBD models with a negligible variation, especially when high numbers of packets are used in the simulation.

On both figures (Figure 3.5 and Figure 3.6) the x-axis represents the number of ACL entries and the y-axis the error-ratio. In the figures two curves are given, with the elementary time-unit \( TU \). The number of packets during a \( TU \) can be
0.01 and \( \frac{\text{packets}}{\text{TU}} = 0.05 \), in case of Poisson traffic. The model gives approximately the same results as the simulation. The relative error is lower than 0.7%.

The important parameters that can be derived from the models are the packet delay and loss ratio as a function of the arrival intensity (Figure 4.1 and Figure 3.8). The models also show us the influence of overall packet loss and the delays a packet can suffer. According to the models, for various arrival intensities, the characteristics of the delay have a soft-knee at the ACL size when the packet loss starts to be notable.

The delay can be derived from the models in quantities of \( \text{TU} \), which is the elementary time-unit described. When examining the delay as a function of the packet arrival intensity, the models show that the actual value of the delay is converging to an upper bound (Figure 3.9).

This upper bound can be calculated as (3.17), where \( N \) is the number of ACL
\[ E(Delay_{\text{max}}) = S_q \cdot \sum_{i=1}^{N} i \cdot p_i \]  

(3.17)

In the example depicted on Figure 3.9, with \( N = 100 \) ACL entries and with the packet matching probabilities \( p_{30} = 0.2, p_{70} = 0.4, p_{100} = 0.4 \), and \( p_i = 0 \) otherwise. The mean value of the upper bound of the delay is 5550 \( \cdot TU \).

The physical meaning of this characteristic is that at higher arrival intensities packet loss gets more and more significant, thus the delay parameter cannot be increasing anymore. Packets, which are not dropped by the router, suffer a constant delay, equivalent to the complete delay of a full buffer.
3.2 Comparison of the finite and infinite model solutions

After creating models to describe filter lists, I also examined and compared the models I created (finite and infinite one), because the infinite model is faster calculable, but the finite one is precise.

However, algorithms do exist that aim to decrease the time needed for the evaluation of a finite DQBD model, such as the folding algorithm from San-qi Li [42],[43], they can only be applied under certain circumstances and it is still faster to solve an infinite DQBD model instead. Although calculation of a finite DQBD requires more computational power, approximating with an infinite transition matrix can result in unacceptable error ratio in certain cases [34].

Thesis 3.2.

*I defined an upper boundary (Equation 3.20 and 3.21), under what the infinite model results (Thesis 3.1) can approximate well (below $10^{-5}$) the results of the finite model, in case of any kind of matching probabilities in the ACL. The boundary is given in case of Poisson traffic [PalugyaiLNCS1].*

Comparing the two models, we can estimate the applicability range of the infinite, thus the simpler and faster model. I assume that the infinite model is applicable by means of matrix-geometric methods if the relative error between the two models is below 0.001%.

In case of a list, what has $N$ list-rules, the matching probability distribution has two bounds. The first one is, when the packets of the traffic match against the first list-rule with probability 1, as it is shown by (3.18). In this case the applicability of the infinite model has no limit, because this model has no error in this case.

$$ P_r(\text{matching}) = \begin{bmatrix} 1 & 0 & \cdots & 0 & 0 \end{bmatrix} \quad (3.18) $$

The other bound is more interesting. Here, the traffic matches against the last rule with probability 1 (Equation 3.19). This case gives the limit of the applicability
of the infinite model. I examined this case and I approximated the limit under which
the infinite model might be used.

\[ P_r (\text{matching}) = \begin{bmatrix} 0 & 0 & \cdots & 0 & 1 \end{bmatrix} \] (3.19)

My examinations show that by comparing the computation times and results of
the models regarding the delay, the applicability of the infinite model is limited by
the function \( \frac{y(S_q)}{\lambda T U} \) (Figure 3.10). I compared and analyzed these results by using the
Matlab software [44].

![Graph showing the applicability of the infinite model.](image)

Figure 3.10: Applicability of the infinite model

![Graph showing the dependency of y from the buffer size (S_q).](image)

Figure 3.11: The dependency of \( y \) from the buffer size \( (S_q) \)

Accordingly, it is reasonable to use the infinite DQBD process for approximation
in the domain under the curve \( y(S_q) \) (Figure 3.10), on the other hand in the domain
above the finite model is more effective.

In Figure 3.10 the function \( y(S_q) \) depends on the buffer size of the router \( (S_q) \),
as depicted in Figure 3.11. However, \( y(S_q) \) never reaches the theoretical limit of 0. I also found that the function \( y(S_q) \) can be approximated under 150 ACL entries as 3.20 with a confidence bound of 95%. So it can be used in practise.

\[
y(S_q) = [0.843 \cdot e^{0.001 \cdot S_q} - 1.010 \cdot e^{-0.084 \cdot S_q}]^+
\]

(3.20)

The boundary of the applicability of the infinite model can be defined by (3.21) according to (3.20).

\[
boundary_{LOW}(S_q, \lambda, TU) = \left[\frac{0.843 \cdot e^{0.001 \cdot S_q} - 1.010 \cdot e^{-0.084 \cdot S_q}}{\lambda \cdot TU}\right]^+
\]

(3.21)

3.3 Modeling of hierarchical structures

[PalugyaiC8]

As it is mentioned in the Introduction, there are a few types of filter lists, which are different from the sequential list. These lists have different structures, but these structures are basically hierarchical structures. In this section I model the list-structures that I introduced in the Introduction.

Thesis 3.3.

I created mathematical models, and I have given a description method in mathematical form, for general modeling of hierarchical list-structures. A mathematical model of the BSD filter system is created (Equations 3.24, 3.25 and Figure 3.12), based on the notations of Table 3.4, and a general model to describe hierarchical-lists and TTCNv3 alternative behavior is also introduced with (3.26)-(3.28) and with Figure 3.13, based on Table 3.5. With the help of general Markovian methods performance parameters can be calculated from my models [PalugyaiC8].

3.3.1 The BSD filter list

The BSD is a widely known operation system. This system uses a solution-option to reduce the delay in the filter list. BSD collect the similar rules of the list and make groups from them. General rules are created for these groups. At the beginning
of the search on the list, the packet chooses first the proper group-rule, and after choosing it searches through on the list-entries in the group.

The Markovian description of this system is quite similar to the ACL system (Section 3.1). The transition probability matrix \((P)\), and the expressions of the build up matrices (3.8, 3.9, 3.10, etc.) are the same, because the input traffic and the system buffer can be described as in Section 3.1.

The different part is the internal behavior of the filter system (matrix \(T\)). The Markovian graphical representation of the internal behavior of the BSD solution is depicted in Figure 3.12.

![Figure 3.12: The DQBD model of the BSD filter system](image)

For describing the structure in Figure 3.12, it is recommended to use different matrix structures. For the whole description of \(T\) I used description structures, that are collected in Table 3.4.

I have to define some assistant matrices and vectors, like 3.22 and 3.23.

\[
x = \begin{bmatrix} 1 & 0 & \cdots & 0 \end{bmatrix} \quad (3.22)
\]
Table 3.4: Notations for BSD modeling

<table>
<thead>
<tr>
<th>symbol</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T$</td>
<td>description of the list behavior</td>
</tr>
<tr>
<td>$T_i$</td>
<td>description of the block $i$</td>
</tr>
<tr>
<td>$p^{(i)}$</td>
<td>matching probability vector (in the $i^{th}$ sublist)</td>
</tr>
<tr>
<td>$\Omega$</td>
<td>shifting matrix (the size of it equals to the size of block $i$)</td>
</tr>
<tr>
<td>$x_i$</td>
<td>assistant vector (the size of it equals to the size of block $i$)</td>
</tr>
<tr>
<td>$K$</td>
<td>size of the longest sublist</td>
</tr>
</tbody>
</table>

\[
\Omega = \begin{bmatrix}
0 & 1 & 0 & \cdots & 0 \\
0 & 0 & 1 & 0 & \cdots \\
\vdots & 0 & 0 & 1 & 0 \\
\vdots & \vdots & \vdots & \vdots & \cdots \\
\vdots & \vdots & \vdots & 0 & 0 \\
\end{bmatrix}
\] (3.23)

Using the mentioned notations (Table 3.4) I successfully described the whole internal behavior of the BSD solution. In the model first I described the behavior of the lists inside the groups. The structure of these internal lists (Equation 3.24) is the same as the structure of the Access Control Lists.

\[
T_i = \text{diag} < p^{(i)} > \cdot \Omega_i
\] (3.24)

These internal lists (Equation 3.24) build up the whole structure of the internal behavior ($T$). In (3.25) it can be seen, that $T$ has two parts. In the first part it describes the choosing of the suitable group. The second part of $T$ is a diagonal. In the diagonal the elements are that $T_i$ matrices that describe the internal list behaviors.

\[
T = x_0^T \bigotimes (E - \text{diag} < p^{(0)} >) \bigotimes (x_0 \cdot \Omega_0) + \sum_{i=0}^{K} [\text{diag} < x_i \cdot \Omega_i^T > \bigotimes T_i]
\] (3.25)

With the quite complex matrix structure of (3.25) I properly described the behavior of the BSD filter list solution.
3.3.2 Hierarchical lists

We can see, that the BSD solution is in fact a hierarchical list with one hierarchical level. The general hierarchical lists can have more levels, as it is shown in Figure 3.13.

![Hierarchical list diagram](image)

Figure 3.13: The DQBD model of the Hierarchical filter system

For describing these lists I use similar structures and notations, as in the previous section. But these structures are more complex and there are more of them (Table 3.5).

<table>
<thead>
<tr>
<th>symbol</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_i$</td>
<td>description of the whole block $i$</td>
</tr>
<tr>
<td>$T_{ij}$</td>
<td>description of the block $ij$</td>
</tr>
<tr>
<td>$p_i^{(ij)}$</td>
<td>matching probability vector (in the $ij^{th}$ sublist)</td>
</tr>
<tr>
<td>$\Omega_{ij}$</td>
<td>shifting matrix (the size of it equals to the size of block $ij$)</td>
</tr>
<tr>
<td>$\varpi_{ij}$</td>
<td>assistant vector (the size of it equals to the size of block $ij$)</td>
</tr>
<tr>
<td>$N$</td>
<td>size of the longest sublist (in the internal sublist)</td>
</tr>
</tbody>
</table>

Using the suitable notations (Table 3.5), these structures also can be described. But this description is quite complex (Equations 3.26, 3.27 and 3.28). As it can be
seen, I have to use as many complex matrix expression (like $T_i$) as many as the number of the hierarchical levels in the system. The structure components in the last hierarchical level can be described by simpler structures (like $T_{i,j}$ in (3.28)).

$$T = x^T_{00} \bigotimes (E - \text{diag} < p^{(00)} >) \bigotimes (x_{00} \cdot \Omega_{00}) + \sum_{i=0}^{K} \left[ \text{diag} < x_{i0} \cdot \Omega_{i0} > \bigotimes T_i \right]$$ (3.26)

$$T_i = x^T_{i0} \bigotimes (E - \text{diag} < p^{(i0)} >) \bigotimes (x_{i0} \cdot \Omega_{i0}) + \sum_{i=0}^{N} \left[ \text{diag} < x_{ij} \cdot \Omega_{ij} > \bigotimes T_{ij} \right]$$ (3.27)

$$T_{i,j} = \text{diag} < p^{(ij)} > \cdot \Omega_{ij}$$ (3.28)

When we have more hierarchical levels (like in the case of decision trees) the way of the description will not change. The description of the $T_{i,j}$ will also be similar to the description of $T_i$. The description of the next hierarchical level ($T_{i,j,k}$) will be similar to the $T_{i,j}$, as it is described by (3.28), and so on.

The balanced-tree and decision tree structures (what are also presented in the Introduction (Chapter 1)) are special types of Hierarchical lists. Since I successfully described the general type of Hierarchical lists, for describing balanced and decision tree structures, we just have to set the suitable parameters in my model.

### 3.3.3 TTCN alternatives

TTCN alternatives can almost be described as well, with the help of the Hierarchical model. The hierarchical behavior of TTCN can be described by the mentioned model, but TTCN alternatives have a special feature, as it is discussed in the Introduction. During the search in the TTCN alternative a snap-shot is taken. After snap-shot the packet searches trough the list. If it does not find any matches in the list, then packet will not be dropped, but processor will take a new snap-shot and search trough the list again.

In the mathematical description this means, that the basic build up structures of $T$ (description matrix of the internal behavior) are not the same as in the hierarchical case. Since in the case of normal lists and normal hierarchical list the matrix $T_f$ can be described by (3.29), in the case of TTCN alternatives this matrix is in
CHAPTER 3. MODELING OF ACCESS CONTROL LISTS

\[(3.30)\]

\[
T_f = \begin{bmatrix}
0 & p_1 & 0 \\
0 & 0 & p_2 & 0 \\
0 & 0 & \cdots & 0 \\
0 & 0 & \cdots & 0 & p_{N-1}
\end{bmatrix}
\]

\[(3.29)\]

\[
T_f = \begin{bmatrix}
0 & p_1 & 0 \\
0 & 0 & p_2 & 0 \\
0 & 0 & 0 & \cdots & 0 \\
0 & \cdots & 0 & 0 & p_{N-1}
\end{bmatrix}
\]

Everything else is the same in this structure, as in the case of normal hierarchical lists.

### 3.3.4 Solution of the models

After I built up the suitable description matrices \((A_0, A_1, A_2)\) the calculation of the needed performance parameters is the same as in the case of the Access Control Lists and normal filter systems (Section 3.1.2).

### 3.3.5 Simulation results

The result of simulations is also highly similar as in the previous section (Section 3.1.3).

### 3.4 Conclusion

In this thesis group I created some models which are useful to predict performance parameters (like packet delay and packet loss ratio) of existing filter systems.

I have shown the way I started to model these structures, and what I measured before I found the proper modeling method (DQBD). After this I introduced my models that describe the Access Control Lists properly. I also compared them to each other, and I gave a limit for the applicability of my infinite model. Finally, I modeled filter lists and list structures, which are different from the normal structure
(like BSD, TTCN and hierarchical lists). I also created a general model description for hierarchical list-structures.

All of my models are validated by NS-2 simulations, where I did not use any approximations. These models were calculable by using general Markovian solving technics, so in the performance parameter calculation I did not introduce any novelty.

The created structures also can be used in more complex models, which it is shown in the next section.
Chapter 4

Modeling affect of hardware interfaces

In the previous chapter models were described, which address the internal ACL system in a router. The extended model can include the effects of the input/output interfaces of a router or even of a single PC (Personal Computer) together with an internal Access List into the examination. Interfaces play a significant role in queuing and congestion. Besides, in the extended model the entries of the Access List also have a meaning, namely to forward or to drop the packets of the network traffic [4].

In the proposed model when a packet arrives into the incoming interface, the processor interrupts the actual running process. After this, the processor decodes the incoming packet, so it can be handed over to the process incorporating the access list. Thereafter, the execution continues exactly where the interruption appeared. But, in this case the packet delay in the system is dependent on the further packet arrivals into the system too. Because of this fact we cannot use ordinary Markov-chain solutions to calculate the packet delay.

In this thesis I propose a model which is able to describe the system of Access Control Lists with router environment. In the proposed model the impact of the system interfaces on the delay and loss is also included. The proposed model can handle a preemptive system.

After describing the modeled system I introduce the model that I developed. This model is very complex and it has a huge state-space. For longer lists or for
higher interface delay the calculation time of the model can be very long. Because of this fact I apply an assumption in my model, to reduce its state-space, and I created a model, which has much less states. With the model I also have an other problem. The solution of the model cannot be calculated by existing solving technics. First I used an approximation in the calculation of performance parameters from my model, but in the second step I developed a brand new solving technique. With the help of this new solving method a new group of the Markovian models can be calculated. Those, which describe preemptive systems. In the last step I successfully decreased the state-space of my model based on an other idea. My models, the calculation methods and the proof of the state-space reduction are introduced in this chapter.

To validate the models and results of this chapter, I also created simulations. I examined the measurement of these results, which is also described in this chapter.

4.1 **Affect of the interfaces on the overall delay**

In this section the effect of the preemptive behavior of routers using ACLs is presented shortly, through some experimental measurements. For easier explanation in the following examples a Poisson process is used for generating the incoming traffic. Delay values are depicted using time-units $TU$s, which is the discrete time-slot.

![Figure 4.1: The effect of the ACLs without interfaces](image)

The basic model (ACLs only) which disregards queuing at the hardware interfaces of a router gives the results depicted in Figure 4.1. This figure shows the
packet delay as a function of the list topology’s size for two different arrival intensities \( \frac{\text{packets}}{TU} \).

In Figure 4.2, packet delay is depicted in case I apply a more sophisticated router model. When this extended description of the router is applied, we can analyze the effect of the interfaces on the packet delay as well. In Figure 4.2 four different curves are presented, in each curve the intensity is \( 0.01 \frac{\text{packets}}{TU} \). Each curve is labeled with an \( S_A \) and an \( S_D \) value. These labels mean the delay factor at the input/output interfaces. For example, \( S_A = 5 \) and \( S_D = 10 \) means that the mean value of time spent at the input and output interfaces is 5 and 10 units respectively.

These results show that interfaces seriously affect the performance parameters of the system.

4.2 The examined system

A whole router using ACLs that can be described by the proposed model is
CHAPTER 4. MODELING AFFECT OF HARDWARE INTERFACES

depicted in Figure 4.3. There are two main issues that seriously affect the delay (beside the Access Control List) at the interfaces. First of all, decoding of the incoming packets and secondly the encoding of outgoing packets [45],[46]. In this system a packet arrives into the input interface of the router. After it suffers delay because of decoding at the input interface the packet gets into the Access Control List part of the system, where it is served sequentially. First the packet is enqueued into a FIFO queue of the list. When it reaches the list processor, it is matched against the list-entries. When the packet matches a given rule the packet will either be forwarded or dropped based on the rule entry. When the packet is forwarded it also suffers encoding delay at the output interface.

In the proposed model I consider a single processor, single bus description for modeling the system interfaces [47]. In this model no additional intelligence is included at the interfaces, and only one processor is in the system. The router has a shared memory (Figure 4.4). The processor tries to match the incoming packet against the rules in the Access List. Whenever a new packet arrives into this system, the processor interrupts the running process of ACL matching or packet encoding and the bus becomes free. Processor reads the new packet from the input interface into the memory via the bus. When this process is finished the processor returns to the interrupted process.

A new packet arrival can happen at any time. The packet arrival generates an interrupt in my system as I mentioned above (dashed arrows in Figure 4.3). After an interrupt the processor reads the new packet into the memory via the interface. During memory reading any other packet in the ACL system (for example in the queue) has to wait. While the actual packet is examined a huge amount of new packets can arrive (vertical arrows in Figure 4.6), which increases the packet delay.
significantly. In Figure 4.5, an example is shown for delays that a packet suffers in the router, when there is no other packet arrival during the processing of the actual packet. In the example the base delay of the packet is $t_i + t_Q + t_{acl} + t_o$, according to Table 4.1.

<table>
<thead>
<tr>
<th>symbol</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_i$</td>
<td>decoding time of the arrived packet</td>
</tr>
<tr>
<td>$t_Q$</td>
<td>time spent by the packet in the waiting queue</td>
</tr>
<tr>
<td>$t_{acl}$</td>
<td>time of the ACL processing</td>
</tr>
<tr>
<td>$t_o$</td>
<td>output encoding time</td>
</tr>
</tbody>
</table>

When new packet arrivals happen while the router is processing the actual packet (Figure 4.6), the actual one also suffers delay because of the interruptions, which are caused by packets read from the input interface. The delay caused by new packets is $4 \cdot t_i$ in the example. The full delay that the examined packet suffers in Figure 4.6 is $5 \cdot t_i + t_Q + t_{acl} + t_o$.

4.3 The interface model

In this section I introduce the base and normalized model of the described system. I modeled the interface effect on the performance parameters together with an internal
4.3.1 The model

In this section the steps of modeling of the whole system are introduced. Since the presented model is very complex, I introduce it based on the ACL model, which is described in the last Chapter.

In my model all of the system variables, like input traffic or list size are parameters. Let the system be described by the system variables in Table 3.1 connected to the ACL model, and by the ones in Table 4.2, which are connected to the interface part of the system.

<table>
<thead>
<tr>
<th>symbol</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F_{t_{input}}$</td>
<td>distribution of the input processing time at the interface</td>
</tr>
<tr>
<td>$F_{t_{output}}$</td>
<td>distribution of the output processing time at the interface</td>
</tr>
<tr>
<td>$p_v$</td>
<td>a vector describing the rules (&quot;permit&quot; or &quot;deny&quot;) (1 x N)</td>
</tr>
</tbody>
</table>

With the help of the system variables the whole system can be described. The model parameters are built up using system variables. For example, from the size of the list ($N$) and from the rule matching probabilities vector ($p_v$) I create matrix $T_v$. This matrix describes the possible behavior of the given list. The model parameters that are built up from the system variables are listed in Table 3.2 (ACL) and in Table 4.3 (interfaces).

In the first step I build up the Access Control List model, which was described in the previous Chapter.

For the interface-part modeling I also use Discrete-Time Markov chains by choosing a specific time slot $TU$. The model describes processes that happen in the router using ACLs during $TU$. I have chosen $TU$ to be the time what is needed to examine a packet, whether it matches against a rule or it does not. This time period is considerably short and hardware dependent. The user of the model can get this parameter from measurements on a specific hardware platform or from other mathematical models.

For easier explanation in Figure 4.7 I used a special type of process for input traffic, the Poisson process (exponential inter-arrival time distribution with parameter
Table 4.3: Interface model parameters

<table>
<thead>
<tr>
<th>symbol</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_A$</td>
<td>number of states in $IN$ (the input interface model)</td>
</tr>
<tr>
<td>$S_D$</td>
<td>number of states in $OUT$ (the output interface model)</td>
</tr>
<tr>
<td>$A$</td>
<td>description of the internal behavior of input interfaces ($S_A \times S_A$)</td>
</tr>
<tr>
<td>$D$</td>
<td>description of the internal behavior of output interfaces ($S_D \times S_D$)</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>rate vector to the absorbing state (input interface) ($1 \times S_A$)</td>
</tr>
<tr>
<td>$d$</td>
<td>rate vector to the absorbing state (output interface) ($1 \times S_D$)</td>
</tr>
<tr>
<td>$L_v$</td>
<td>probability of a &quot;deny&quot; list-entry in the list ($1 \times N$)</td>
</tr>
<tr>
<td>$L^*$</td>
<td>probability of a &quot;permit&quot; list-entry in the list ($1 \times N$)</td>
</tr>
<tr>
<td>$\alpha_A$</td>
<td>initial distribution at the input interface model-part ($S_A \times 1$)</td>
</tr>
<tr>
<td>$\alpha_D$</td>
<td>initial distribution at the output interface model-part ($S_D \times 1$)</td>
</tr>
</tbody>
</table>

$\lambda$). In Figure 4.7, a packet either matches a rule entry or it does not and moves to the next entry during a time step. In the first case we have a departure and get the next packet from the waiting queue. So we will have one packet less in the queue. The next packet starts to be examined from the first list-entry. In the second case, when the packet did not match against a specific rule, model steps to the following rule. Also during this time-step a new packet can arrive into the system. With the combinations of the arrival and matching probabilities the model can be built up. In the proposed model there are two special conditions. When the system examines the actual packet against the last rule the result has to be a match, because of the implicit permit or deny rule. The other condition is, when the model is in the last model-level (waiting queue is full) and a packet arrives. In this case system has to drop this packet.

Modeling the system interfaces together with the Access List is a difficult task. A packet in the described system suffers the decoding time (at the input interface) and it may suffer the encoding time (at the output interface) as well. During the interface modeling the meaning of the rules in the list has to be taken into consideration too, not only the matching probabilities. The packet may suffer the outgoing delay. This means, if packet matches against a "permit" rule it will be forwarded. In this case the packet suffers the outgoing delay. In the other case the rule is "deny", so the packet will be dropped and it does not suffer the outgoing delay. Probabilities that rules are "permit" or "deny" are described by vector $p_v$.

In the proposed model system interfaces are described by mean values of the
time packets spend inside them while they are encoded or decoded. These two parameters (S_A and S_D) are also hardware dependent. We can obtain these parameters via modeling or measurements. The mean values of the time spent at the input and output interfaces have to be divided by the discrete time-step. With the aid of these parameters the model can describe the input/output delay of the packets as Figure 4.8 shows. The encoding of a packet (in the output interface) may be interrupted by arrival of a new packet into the system. If we model the input interface to be preemptive, namely that an incoming packet might interrupt decoding of a packet, this can lead to a state space explosion.

Figure 4.7: The corresponding part of rule-behavior in the ACL model

Figure 4.8: The model-part of the interface delay

Thesis 4.1.
I have developed a mathematical model (Figure 4.9 and Figure 4.10), based on the parameters in Table 4.3, which can describe the effects of the hardware interfaces of a router, together with the embedded system in it. This model can handle the preemptive packet handling, which use interrupts.

To build up the proposed model I have to combine the state structure of Access Lists (Figure 4.7) and the structures of the interfaces together with the meanings of rule entries. This way the size of the state space grows to a relatively high level, because every state that corresponds to a rule entry connects to $S_A$ states describing the input interface. Also the same connections exist to the states, describing the output interfaces ($S_D$). In this case, it is hard to produce a graphical representation of the model.

After combining the list-part and the interface-part of the model, we can draw up a simple example of one level of this model. In Figure 4.9 I depicted a short example, where one model level contains only a few states. Here the ACL list contains only two rules, the input interface is modeled by three, the output one is modeled by two states. Let me denote the whole level of the model with a complex state (checked cycle in Figure 4.9), because it is needed for representing the whole model.

Figure 4.9: One example level of the interface model

With the help of the checked cycle notation in Figure 4.9, we are able to describe the model by the Figure 4.10, where general Markovian notations are used. This
model has a two-dimension behaviour as it is seen in the figure. The model is very complex and big, because we have to handle the interrupted interface mechanisms. These mechanisms are described by the matrices $F$, $S$, $L$ and $M$ next to the generally used $A_0$, $A_1$ and $A_2$ type matrices.

![Figure 4.10: The model of ACLs together with interfaces](image)

The model in Figure 4.10 has a huge structure for longer lists, interfaces and for real buffer sizes. The complexity of this model is introduced in Figure 4.11 and in Figure 4.12 besides different system parameters ($S_A$ and $S_q$). If the ratio of the input interface part is bigger or equal to the $(S_q - 1)$ (where $S_q$ is the buffer size), then the structure of the model, according to Figure 4.10, will be the one in Figure 4.11. Otherwise the model structure is the one in Figure 4.12. In both figures it can be seen, that there are regular elements (striped elements) in the middle of the model structure, but there are some irregular ones as well. These are the structure elements on the brink of the structure (white elements).

![Figure 4.11: Model structure, when $S_A >= (S_q - 1)$](image)
According to Figure 4.11, Figure 4.12 and the connected structures contain the created interface model as many states in general what is described by (4.1), where I used the notations of the model parameters (Table 3.2 and Table 4.3).

\[
X_1 = S_q \cdot (S_A + 1) \cdot (N + S_D) + S_A \cdot (N + S_D) \cdot \sum_{i=1}^{\min(S_A, S_q-1)-1} (S_q - i)
\]  

(4.1)

I created a complex mathematical model that can describe the effect of the interfaces on embedded systems. This model has a finite state-space. But for longer lists, for real buffer sizes and for longer interface delay, derivation performance parameters from this model can be very slow.

4.3.2 The normalized interface model

[PalugyaiC5],[PalugyaiC6]

Since the developed model has very big state-space I tried to decrease this number of states in the model structure.

Thesis 4.2.

I successfully decreased the state-space of the Interface model (Thesis 4.1). I applied the assumption (4.2) and I created a one-dimensional model (Figure 4.13) from the two-dimensional one (Figure 4.10).

I applied my assumption (Equation 4.2) during decreasing the state-space of my model. Namely that during packet decoding at most one new packet arrival can happen into the input interface. This way the state space explosion can be avoided. From the two-dimensional structure we can create a one-dimensional model.
By comparing the model results to simulation results I proved that this assumption was correct (Section 4.6).

The model still has many states and the graphical representation is still complicated. I depicted the normalized model in Figure 4.13, which uses the mentioned notation of Figure 4.9.

\[ X_2 = S_q \cdot (S_A + 1) \cdot (N + S_D) \]  

Since the graphical representation of the model is still too difficult in the general case (for long lists and real buffers) I use matrix-geometric description instead of the graphical one. The matrix-geometric description was introduced by Neuts in [35].

My proposed model is a finite Discrete-Time QBD with irregular first, second and last levels. It can be defined by its transition probability matrix \( \mathbf{P} \) (Equation 4.4).

\[
\mathbf{P} = \begin{bmatrix}
A_1' & A_0' & 0 & 0 & \cdots \\
A_2' & A_1^* & A_0^* & 0 & \cdots \\
0 & A_3^* & A_1 & A_0 & 0 & \cdots \\
0 & 0 & A_2 & A_1 & A_0 & 0 \\
\cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
0 & 0 & A_2 & A_1 & A_0 & 0 \\
0 & \cdots & \cdots & \cdots & \cdots & A_1^K
\end{bmatrix}
\]  

\[ P_r(\# \text{ of arrivals during } S_A > 1) \approx 0 \]  

(4.2)
Matrices $A'_1$ and $A'_0$ represent transitions inside a level and one level upwards at the first irregular level. Similarly, the second irregular level is described by matrices $A^*_1$, $A^*_0$ and $A'_2$, where they denote transitions inside the level, one level upwards and one level downwards respectively. All the other levels in the DQBD process are described by the regular matrices $A_0$ (4.11), $A_1$ (4.12) and $A_2$ (4.13), except $A^*_2$ that represents transitions back to the second irregular level. Beside the regular matrices and the ones describing the first and the second irregular levels one more irregular matrix has to be defined because of the finite nature of my DQBD model. Accordingly, $A^K_2$ represents transitions inside the last level of the DQBD, which is irregular because there are no further transitions upwards from this level. These matrices ((4.5),(4.6),(4.7),(4.8),(4.9),(4.10),(4.11),(4.12),(4.13),(4.14)) are built up using the Kronecker-product and with the model parameters described in Table 3.2 and Table 4.3.

\[
A'_0 = \begin{bmatrix}
D_1 \otimes \alpha_A & 0 & 0
\end{bmatrix}
\]  

(4.5)

\[
A'_1 = [D_2]
\]  

(4.6)

\[
A'_2 = \begin{bmatrix}
0 \\
\frac{D_2 \otimes t_v}{D_2} \\
\frac{D_2 \otimes d}{D_2}
\end{bmatrix}
\]  

(4.7)

\[
A^*_0 = \begin{bmatrix}
D^*_1 \otimes a \otimes \alpha_D \otimes \alpha_A & 0 & 0 & 0 \\
D_1 \otimes \alpha_D \otimes \alpha_A & 0 & 0 & D_1 \otimes t^* \otimes \alpha_D \otimes \alpha_A \\
0 & 0 & 0 & D_0 \otimes D \otimes \alpha_A
\end{bmatrix}
\]  

(4.8)

\[
A^*_1 = \begin{bmatrix}
I_D \otimes \alpha & D^*_1 \otimes a \otimes \alpha & 0 \\
D_1 \otimes \alpha_D \otimes \alpha_A & D_0 \otimes T & D_0 \otimes t^* \otimes \alpha_D \\
D_1 \otimes d \otimes \alpha_A & 0 & D_0 \otimes D
\end{bmatrix}
\]  

(4.9)
I distinguish between the four main state-groups in the model description: the
input interface, the list-part, the output interface state-group, and a special state-group, which contains input interface states connected to the output interface states. According to these state-groups the regular descriptor matrices (4.11,4.12,4.13) contain $4 \cdot 4 = 16$ matrix-blocks. For example a \{1, 2\} matrix-block describes the transition from the input interface part (first group) to the list-part (second state-group) of the model.

Here I explain three example blocks of the upper equations (4.11,4.12,4.13, etc.). The actions described happen during one time-unit (TU), and level means the Markovian model levels in the created DQBD. In (4.11) the \{1, 1\} matrix-block $(D_1^* \otimes L_N \otimes a \otimes \alpha A)$ means that in the input interface part of the model we can step one level up when a new packet arrives to the input interface. Here I introduce the four main parts of \{1, 1\}. $D_1^*$ describes the arrival process at the input interface. $L_N$ means that the position in the list remains unchanged. $a$ means that we can step into the list-part, and $\alpha A$ means we start in a new interface part, because a new packet has arrived. In (4.12) the \{2, 1\} part $(D_1 \otimes L_v \otimes a \otimes \alpha A)$ means that we step into the input interface part from the list-part, inside the same level, when a new packet arrives and at the same time the system emits another packet. This is because the actual processed packet matches against a deny rule. In \{2, 1\} $L_v$ describes that a packet departs and the rule is deny, and $\alpha$ means we start in the first state of the list-part of the model. Finally, $A_1(2, 2)$ (in Equation 4.11) $D_0 \otimes T$ means that we step ahead in the list to the next rule and no packet arrives and departs during TU. Here $D_0$ means no packet arrives during TU and $T$ describes the behavior of the list-part.

After an applied idea, I constructed a model, which is able to describe the whole system, which use ACLs and it has system interfaces. This normalized model can be used in practice, since it has much less states, than the first model.

4.3.3 Solution of the model with approximation

After the model has been built up performance parameters can be derived. In the case of the presented model the calculation of the delay is more complicated. With general Markovian solutions the delay of this model cannot be calculated. Since now a suitable calculation method for systems, which use preemptive packet handling, is not published. During this calculation we have to use an approximation to get the required information, the performance parameters.
In case of general Markov chains the performance parameters can be calculated with existing solving technics. When we examine the packet delay in the system after the packet arrives the input traffic can be stopped, because the following arrivals do not affect the delay of the examined packet. But in the case of preemptive packet handling this is not true. Because of this fact we need an approximation or a new calculation method.

In Table 4.4, I introduce the structures that I use for the approximated calculation.

<table>
<thead>
<tr>
<th>symbol</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$MVAP$</td>
<td>Mean Value of Arriving Packets</td>
</tr>
<tr>
<td>$\Pi_i$</td>
<td>steady state probabilities at level $i : (1 \times (S_A + 1) \cdot (N + S_D))$</td>
</tr>
<tr>
<td>$AP_i$</td>
<td>$A_0$ packet arrival at level $i : ((S_A + 1) \cdot (N + S_D) \times (S_A + 1) \cdot (N + S_D))$</td>
</tr>
<tr>
<td>$AN_i$</td>
<td>$A_1$ packet arrival at level $i : ((S_A + 1) \cdot (N + S_D) \times (S_A + 1) \cdot (N + S_D))$</td>
</tr>
<tr>
<td>$\mathbb{1}$</td>
<td>a column vector full of ones : $([1\ldots1]^T)$</td>
</tr>
<tr>
<td>$arp_i$</td>
<td>arrival probability vector at level $i : ((S_A + 1) \cdot (N + S_D) \times 1)$</td>
</tr>
<tr>
<td>$T_{IN}$</td>
<td>delay, what is caused by a packet arrival</td>
</tr>
</tbody>
</table>

Calculation starts from a generally applied delay equation (Equation 4.15). Here I summarize the probabilities that the system is at the level $i$ and phase $k$ after a demand arrived multiplied by the time that the system has a departure from this level and phase in the model.

$$E(delay) = \sum_{i=1}^{S_q} \sum_{k=1}^{(S_a+1)\cdot(N+S_d)} Pr(afterArrival : N = i, J = k) \cdot E(delay|N = i, J = k)$$  \hspace{1cm} (4.15)

The first part of this sum can be calculated by (4.16). In Equation 4.16, the probabilities inside a level are collected into a vector.

$$P(i) = P(afterArrival : N = i) = \frac{1}{MVAP} \cdot [\Pi_i - 1 \cdot AP_{i-1} + \Pi_i \cdot AN_i]$$  \hspace{1cm} (4.16)
Equation 4.16 contains steady state probabilities \( \Pi_i \), the Mean value of Arriving Packets (Equation 4.18) and special matrices, like \( AP \) and \( AN \) (Equation 4.17). \( AP \) and \( AN \) are created from \( A_0 \) and \( A_1 \) matrices. \( AP = A_0 \) and \( AN = A_1 \) when a packet arrives during the elementary time-slot.

\[
\begin{align*}
AN &= \begin{bmatrix}
0 & 0 & 0 \\
D_1 \otimes t_v \otimes \alpha \otimes \alpha_A & 0 & 0 & 0 \\
D_1 \otimes d \otimes \alpha \otimes \alpha_A & 0 & 0 & 0 \\
0 & 0 & 0 & 0
\end{bmatrix}
\end{align*}
\] (4.17)

The MVAP expression means the Mean Value of Arrival Probabilities that is described in Equation 4.18.

\[
MVAP = \Pi_0 \cdot D_1 \cdot 1_l + \Pi_1 \cdot arp_1 + \sum_{i=2}^{sq-1} \Pi_i \cdot arp + \Pi_sq \cdot arp_sq
\] (4.18)

In Equation 4.18, \( arp_i \) is an arrival probability vector, built up using matrices \( AP \) and \( AN \), as it is described in Equation 4.19.

\[
arp_i = AP_i \cdot 1_l + AN_i \cdot 1_l
\] (4.19)

At this point the unknowns are the steady state probabilities. The steady state is calculated in the followings. I use \( A_0, A_1, A_2 \) type of matrices and the \( R \) and \( S \) matrices. Background informations about \( R \) and \( S \) matrices, together with algorithms how to calculate them, can be found in [34].

Let \( Q, M \) and \( N \) be given according to the equations (4.20),(4.21) and (4.22), where I use the notations of [34] and [35].

\[
Q = \begin{bmatrix}
(A_1' - I) & A_0' \\
A_2' & (A_1^* - I) & A_0^* \\
0 & 0 & 0
\end{bmatrix}
\] (4.20)

\[
M = \begin{bmatrix}
\Pi_0 & \Pi_1 & \alpha & \beta
\end{bmatrix}
\] (4.21)
Now, with (4.23) the steady state can be calculated.

$$N = M \cdot Q$$  \hspace{1cm} (4.23)

With the aid of $\alpha$ and $\beta$ we can calculate $\Pi_i$ (steady state solution) according to Equation 4.24.

$$\Pi_i = \alpha \cdot R^{i-2} + \beta \cdot S^{S_q-i}, \text{ if } 2 \leq i \leq S_q$$ \hspace{1cm} (4.24)

The calculation of the second part of (4.15) is more complicated. Ordinarily, in a Markov chain at level $i$ and phase $k$ we can calculate the delay of demands without problem. The arrival process can be stopped and the time of a departure from this level and phase can be calculated. But, in the proposed model we cannot stop the arrival process, because a new packet arrival interrupts the departure and brings time delay with itself. In this case the arriving packets affect the delay of the actually examined packet, which is under service.

**Thesis 4.3.**

*I propose the approximation Equation 4.25 for the calculation of the second part of (4.15), the overall delay, what a packet spend in the described system. The second part of (4.15) describes the mean value of the packet delay, when the examined packet is at the level $i$ in the phase $k$.*

$$E(delay|N = i, J = k) = E(T|N = i, J = k) + E(delay|N = i, J = k) \cdot T_{IN} \cdot D_1$$  \hspace{1cm} (4.25)

In (4.25) two unknown elements can be found. $E(T|N = i, J = k)$ is the delay of the packet, in the case, when there is no arrival during it serves. This expression ($E(T|N = i, J = k)$) can be calculated by generally used Markovian solutions. The second unknown element ($T_{IN}$) is the time that a new packet arrival brings into the system.

It can be seen that (4.25) has a recursive structure. This is because when a packet arrives, it brings $T_{IN}$ time into the system. But during this $T_{IN}$ time a new packet may also arrive. This is even more obvious at higher input traffic intensities.
After rearrangement of (4.25), this part of the expression will be the one in (4.26).

\[
E(\text{delay}|N = i, J = k) = \frac{E(T|N = i, J = k)}{1 - \sum_{i=0}^{s_{q-1}} \Pi_i \cdot D_1 \cdot \mathbb{1} \cdot S_A}
\]

During introduction of my model I mentioned that the delay that a packet brings into the system is $S_A$. This time comes into the system every time when a packet arrives, except when the buffer is full, so we are at the last level in my model. This fact is described by the expression $\sum_{i=0}^{s_{q-1}} \Pi_i$ in (4.26).

For this approximation the delay of a packet in the introduced system can be calculated for lower traffic intensities.

### 4.4 The punctual solution of the model

[Natural, approximation of the real performance parameters was not adequate for me. For lower incoming traffic intensities the approximation works well. But for higher intensities this approximation has increasing error. Because of this face, I developed a new calculation method to derive the exact mean value of delay packets suffer in the described system.

**Thesis 4.4.**

*I propose a method and structure (Figure 4.15, Figure 4.16 and Equations 4.28-4.34) that is applicable to calculate the exact value of the second part of (4.15). This equation is used to assign the overall latency, what packets spend in the described system. This method can be generally used for DQBD structures that use preemptive behavior.*

The mean value of the delay a demand suffers can be derived exploiting the special structure of the underlying DTMC and using DQBD solving techniques. In the following I introduce the structure and method, that is applicable to calculate packet delay from the introduced model.
In Figure 4.14 the waiting queue and the server (which contains the ACL system) of the system is depicted. Here the full size of the waiting queue is $S_q$. The actually examined demand is in the $j$th place in the queue, and $i$ is the number of packets in the waiting queue. This means that since the actually examined packet arrived into the system, $i - j$ other packets have also arrived into the queue.

Using the notations of Figure 4.14, with the aid of the matrices in Table 4.5 I built up a triangle structure (Figure 4.15). This structure is used for calculating the departure time from the given level and phase in my model.

The necessary equations to calculate the departure time from level $i$ and phase $k$ ($D_{(i,j)}(k)$ in Figure 4.15) are presented in the following, based on (4.27).

The base equation is (4.27). This equation describes the general step-probabilities between the states in Figure 4.15.

$$D_{(i,j)} = 1 + K \cdot D_{(i,j)} + A_3 \cdot D_{(i+1,j)} + A_2 \cdot D_{(i-1,j-1)} + V \cdot D_{(i,j-1)}$$  \hspace{1cm} (4.27)

From (4.27) the following equations can be derived. These equations are the special equations on the sides of the triangle structure (Figure 4.15).
\[ D_{(i,j)} = (I - K)^{-1} \cdot \left[ I + A_0 \cdot D_{(i+1,j)} + A_2 \cdot D_{(i-1,j-1)} + V \cdot D_{(i,j-1)} \right] \quad (4.28) \]
if $2 < i < Sq; 1 < j < Sq$

\[
D_{(Sq,j)} = (\mathbb{I} - \tilde{K})^{-1} \cdot [\mathbb{I} + A_2 \cdot D_{(Sq-1,j-1)} + \tilde{V} \cdot D_{(Sq,j-1)}]
\]  

(4.29)

if $i = Sq; 1 < j$

\[
D_{(1,1)} = (\mathbb{I} - \tilde{K})^{-1} \cdot [\mathbb{I} + A_0 \cdot D_{(2,1)} + A_0^* \cdot D_{(1,1)}]
\]  

(4.31)

if $i = 1; j = 1$

\[
D_{(i,0)} = 0
\]  

(4.32)

if $i >= 1; j = 0$

\[
D_{(i,1)} = (\mathbb{I} - \tilde{K})^{-1} \cdot [\mathbb{I} + A_0 \cdot D_{(i+1,1)}]
\]  

(4.33)

if $1 < i < Sq; j = 1$

\[
D_{(i,j)} = (\mathbb{I} - \tilde{K})^{-1} \cdot [\mathbb{I}]
\]  

(4.34)

if $i = Sq; j = 1$

I observe the whole life-time of a packet in this system. When a packet arrives into the queue, its place is always the last place in the queue (FIFO: First In, First Out). To calculate the departure time of this demand (overall latency) we have to calculate $D_{(i,i)}$. According to Figure 4.14 we can also see that $j$ (the actual position of a packet in the queue) cannot be greater than $i$ (the number of packets in the queue). So my structure (Figure 4.15) will form a shape of a triangle. Figure 4.16 depicts the order of evaluation of the state dependent delay and the equations (enclosed in brackets) to calculate them. The calculation of the state dependent delay starts form the right-down corner. From this point we have to continue the calculation in the left direction, and then up until the upper point of the triangle structure.

When we have calculated the state dependent delay ($D_{(i,j)}$) for all states of Figure
4.15 the mean value of the overall latency of a packet can be calculated according to (4.15), where \( E(\text{delay}|N = i, J = k) = D_{m(i,k)}. \)

4.5 State-space reduction

[PalugyaiJ5]

4.5.1 The idea of the state-space reduction

In case of a long list the model I designed still has a quite large number of states, as it is presented in Section 4.3.2. Because of this fact, calculation of performance parameters is generally slow. States at one level of the model and a drain state is depicted in Figure 4.17. In this example, the list contains two rule-entries. The connected input and output state-groups are also depicted. Here, the system dependent behavior of input and output interfaces are described by three input and two output states. In this example two states can model the behavior of the list. But with the interface model parts I use \( 2 \cdot (1 + 3) \) (rules and connected input interface parts) plus \( 2 \cdot (1 + 3) \) states (output interface part and connected input interface parts). So in this example we have 16 states in one level (in case of two rule-entries). For longer lists and for more interface states the model may have much more states at a level.

Figure 4.17: One level of the model and a drain state (example)
My goal is to reduce the state-space without changing the calculated performance parameters. The idea is to minimize the input interface part of the model. Originally, there is only one entry point in the input interface model part and one exit point as it is shown in Figure 4.8 and in Figure 4.18. This is a point in the model, where we can reduce the state-space, without introducing errors in the examined performance parameters. Besides, at this point, a remarkable state-space reduction is possible. So, I replace the originally deterministic input interface state-group with only one state, which has geometric distribution (Figure 4.18). After this replacement, the number of states decreases to $(2 + 2) \cdot (1 + 1) = 8$ states. For longer lists and for bigger interface model parts (this is the typical case) the state-space can decrease more drastically.

![Figure 4.18: How to convert the input interface state-group (IN) into one state](image)

Naturally, we have to define the leave and stay properties of the new geometric distribution (question marks in Figure 4.18). Besides, I have to prove that the state-space reduction does not affect the calculated performance parameters, such as the mean value of packet delay or the probability of packet loss.

**Thesis 4.5.**

My given method (Figure 4.20 and Equation 4.50) reduce the state-space of the proposed model (Thesis 4.2) according to a new idea (Figure 4.18). I assumed that the derived performance parameters do not depend on the distribution of the input interface model-part. I assumed that the performance parameters (packet delay and loss) depend only on the mean value of this distribution. I have proved this assumption and I successfully decreased the state-space of the model.

### 4.5.2 The proof of the state-space reduction

For this proof we have to distinguish between two cases, the one with deterministic input interface state groups, and the other with geometric states.
To obtain this proof we have to separate the input interface (IN) state-groups from the other states on each level (Figure 4.19). Figure 4.19 shows, that after the state separation there are no differences between the states in \{U\} in the two cases. I collect different states into \{D\} and \{W\}, using notations of Table 4.6. These sets (\{D\} and \{W\}) contain the IN state-groups. \{D\} contains the ones of the deterministic model-part, and \{W\} contains the IN states of the geometric model-part. The differences between the two main sets (Figure 4.19) are only in these IN-sets (\{D\} and \{W\}), and at the transitions which go trough the \{D\} – \{U\} and \{W\} – \{U\} boundaries.

The probability of packet loss in the separate cases is the same if the stationary probabilities of the \{U\} states are the same. Besides, the mean value of packet delay is equal too, if the mean times spent in \{D\} and \{W\} are also equal in the two cases.

<table>
<thead>
<tr>
<th>symbol</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>{U}</td>
<td>sets of the common states</td>
</tr>
<tr>
<td>{D}</td>
<td>all input interface state (deterministic case)</td>
</tr>
<tr>
<td>{W}</td>
<td>all IN states (geometric case)</td>
</tr>
<tr>
<td>{d}</td>
<td>one IN state-group (deterministic case)</td>
</tr>
<tr>
<td>[w]</td>
<td>one IN state (geometric case)</td>
</tr>
</tbody>
</table>

My goal is to prove that the entries in \{D\} and \{W\} describe the same behavior and they give the same result regarding the examined performance parameters, i.e. this model does not cause error.

Let us examine the case when only one entry is represented in the Access List, and there is no output delay (here I do not take the output interface into consideration). In this case only one IN state-group exists in the model. So, we can derive the subsets of the general \{D\} and \{W\}. Let these subsets be \{d\} and \[w\] in this case (Equation (4.35)). Now \{d\} contains only one IN state-group, and \[w\] contains only
Table 4.7: Notations in connection with state-separation

<table>
<thead>
<tr>
<th>symbol</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>( T_d, T_w )</td>
<td>time spent in states of ( {d} ) and in state ( w )</td>
</tr>
<tr>
<td>( P_d, P_w )</td>
<td>probabilities of stay inside ( {d} ) and ( w )</td>
</tr>
<tr>
<td>( P_{dU}, P_{wU} )</td>
<td>probabilities of stepping into ( {U} ) from ( {d} ) and ( w )</td>
</tr>
<tr>
<td>( P_{Ud}, P_{Uw} )</td>
<td>probabilities of stepping into ( {d} ) and ( w ) from ( {U} )</td>
</tr>
<tr>
<td>( \Pi_d, \Pi_w^o )</td>
<td>initial distributions in ( {d} ) and ( w )</td>
</tr>
<tr>
<td>( \Pi^* )</td>
<td>vector of departure probabilities from the last state of ( {d} )</td>
</tr>
</tbody>
</table>

one state. If we can prove that \( \{d\} \) is commutable with \( w \), and this replacement does not affect the calculated performance parameters, than this assumption will also be true for the sets \( \{D\} \) and \( \{W\} \).

\[
\{d\} \subset \{D\}, \text{ and } w \subset \{W\} \tag{4.35}
\]

In (4.36) it is shown, that the distributions of the time when we step into \( \{U\} \) from \( \{d\} \) and \( w \) are not equal. If it would be true, every performance parameter we obtain would be equal in the two different cases. Here \( \gamma_U \) is the moment of stepping into a state of the set \( \{U\} \).

\[
P_r(X(\gamma_U) = j \in \{U\}, \gamma_U < t | X(0) = i \in \{d\}) \neq P_r(X(\gamma_U) = j \in \{U\}, \gamma_U < t | X(0) = i \in \{w\}) \tag{4.36}
\]

But condition (4.36) is not entirely true in our case, because the distribution of \( \gamma_U \) is not the same in the two cases. However, the conditional mean values are equal for arbitrary start and final states, as it is shown in (4.37). If condition (4.37) is true the steady state and the mean time to absorption equals in pairs in the mentioned two cases (in case of \( \{d\} \) and \( \{w\} \) sets).

\[
E(\gamma_U | X(0) = i \in \{d\}, X(\gamma_U) = j \in \{U\}) = E(\gamma_U | X(0) = i \in w, X(\gamma_U) = j \in \{U\}) \tag{4.37}
\]

Condition (4.37) is true, if the mean time, spent in the different groups, and the probability of stepping to \( j \subset \{U\} \) also equals in pairs. The mean time in group \( \{d\} \) \( (E(T_d)) \) can be calculated according to (4.38).
\[ E(T_d) = \sum_i \sum_j E(\gamma_U | X(0) = i \epsilon \{d\}, X(\gamma_U) = j \epsilon \{U\}) \cdot P_r(X(0) = i \epsilon \{d\}, X(\gamma_U) = j \epsilon \{U\}) \]  

(4.38)

The probability of stepping into \( j \subset \{U\} \) from \( \{d\} \) is shown in (4.39). The mentioned expressions use the notations of Table 4.7.

\[ P_r(X(\gamma_U) = j | X(0) = i) = [(I - P_d)^{-1} \cdot P_{dU}]_{ij} \]  

(4.39)

According to (4.38) and (4.39) conditions (for the different cases) are shown in (4.40) and (4.42).

\[ E(T_d | X(0) = d_1) = E(T_w) \]  

(4.40)

(4.40) means, that the times spent in \( \{d\} \) and \( w \) are the same, if the starting state in \( \{d\} \) is the first state of \( \{d\} \) (Equation 4.41). As it is mentioned above, in my model we start from the first state in \( \{d\} \), so this condition is true.

\[ P_r(we\ start\ from\ d_1 | we\ step\ into\ \{d\}) = 1 \]

(4.41)

\[ [(I - P_d)^{-1} \cdot P_{dU}]_{ij} = [(I - P_w)^{-1} \cdot P_{wU}]_{1j} \]  

(4.42)

As I mentioned above, if conditions (4.40) and (4.42) are true, the mean time, spent in \( \{d\} \) and \( w \), and the probability of stepping into \( j \subset \{U\} \) also equals in pairs. So, state-group \( \{D\} \) can be replaced with \( \{W\} \), and we can reduce the state-space of the model.

Let us convert state-group \( \{d\} \) into a state \( (w) \), and assign its state-transition probabilities to decrease the state-space of my model significantly.

Let \( \Pi^* \) (Equation 4.43) be the vector of departure probabilities from the last state of \( \{d\} \). From other states of \( \{d\} \) the departure probabilities are equal to zero.

\[ \Pi^* = \Pi^*_d \cdot (I - P_d)^{-1} \cdot P_{dU} = \Pi^*_w \cdot (I - P_w)^{-1} \cdot P_{wU} \]  

(4.43)

In (4.43), \( \Pi^*_d \) (Equation 4.44) and \( \Pi^*_w \) (Equation 4.45) are the initial distribution vectors in the sets \( \{d\} \) and \( w \).

\[ \Pi^*_d = \Pi^* \cdot (I - P_U)^{-1} \cdot P_{Ud} \]  

(4.44)
\[ \Pi^0_w = \Pi^* \cdot (I - P_U)^{-1} \cdot P_{Uw} \] (4.45)

Resulting from the previous equations, the sum of the state-probabilities of set \{d\} equals to the state-probability of \(w\) (it is also true for \{D\} and \{W\}), as it is shown in (4.46). Here \(\Pi_w\) and \(\Pi_d\) contain the state-probabilities of \(w\) and \{d\}.

\[ \sum_d \Pi_d = \sum_w \Pi_w, \text{ and } \sum_D \Pi_D = \sum_W \Pi_W \] (4.46)

The mean time spent in set \{d\} can be calculated as the mean value of a PH distribution with parameters \(\Pi^0_d\) and \(P_d\), as it is described in (4.47) [34]. Call this value \(S_A\), because I used this notation \((S_A)\) as a model parameter for describing the delay of the input interface.

\[ E(PH(\Pi^0_d, P_d)) = \Pi^0_d \cdot (I - P_d)^{-1} \cdot 1 = S_A \] (4.47)

The same time is also spent in state \(w\). The mean departure time from state \(w\) can be calculated according to (4.48), using the mean value of the geometric distribution.

\[ S_A = \frac{1}{1 - P_r(\text{stay in } w)} \] (4.48)

Rearranging (4.48) gives us the probability of stepping out from \(w\) (Equation 4.49).

\[ P_r(\text{stay in } w) = 1 - \frac{1}{S_A} \] (4.49)

Using (4.49), the state-transition probability from \(w\) to the set \{U\} can be calculated according to (4.50).

\[ v = (1 - P_r(\text{stay in } w)) \cdot \Pi^* = \frac{1}{S_A} \cdot \Pi^* \] (4.50)

If we normalize \(\Pi^*\), the vector of departure probabilities from the last state of \{d\} with \(S_A\), the description satisfies conditions (4.40) and (4.42). So, (4.37) is also true. Satisfying (4.37) allows us to replace set \{D\} with set \{W\}. With this state-group replacement, the same system is described without affecting examined performance parameters (delay of a demand, and packet loss probabilities).

The behavior of the input interface can be described with one state (connected to
each list-element), which has geometric distribution. The state-change probabilities
are \( S_A \) times smaller than in the original deterministic representation. Accordingly
the conversion, which is depicted in Figure 4.18 can be defined as in Figure 4.20.

\[
\begin{align*}
\begin{array}{c}
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot 
\end{array}
\end{align*}
\]

Figure 4.20: Conversion of the input interface state-group \((IN)\).

### 4.5.3 Affect of the reduction on the model

After applying the idea of state-space reduction, steps of performance parameter
derivation (Section 4.4) do not change, only the state-transition probability matrices.
The main matrices, what have changed are the (4.11), (4.12) and (4.13), are the
regular matrices. All of the other matrices (the irregular ones) have also changed,
but here I do not describe them, because the way of the reduction is visible from the
change of the regular ones. The new regular matrices are (4.51),(4.52) and (4.53).
The size of the quadratic description matrices (Equations 4.11, 4.12 and 4.13) is
\( D \cdot (S_A + 1) \cdot (N + S_D) \times D \) \( \cdot (S_A + 1) \cdot (N + S_D) \), where \( D \) is the quadratic size of
the input traffic description matrices \((D_0, D_1)\). The size of the new description
matrices (Equations 4.51, 4.52 and 4.53) is \( 2 \cdot D \cdot (N + S_D) \times 2 \cdot D \cdot (N + S_D) \). As it
is shown, the reduction of the state-space is significant. With this state-reduction,
the calculation of performance parameters is much faster than in the last case.

\[
A_0 = \begin{bmatrix}
D_0^* \otimes \mathbf{I}_N & 0 & 0 & 0 \\
D_0 \otimes \mathbf{T} & 0 & 0 & D_0 \otimes t^* \otimes \alpha_D \\
0 & 0 & 0 & D_0 \otimes \mathbf{D} \\
0 & 0 & 0 & D_0^* \otimes \mathbf{I}_{S_d}
\end{bmatrix}
\]  
(4.51)
\[ A_1 = \begin{bmatrix}
I_D \otimes L_N \otimes (1 - \frac{1}{S_A}) & D_{\alpha} \otimes L_N & 0 & 0 \\
D_1 \otimes L_v \otimes \alpha & D_0 \otimes T & D_D \otimes t^* \otimes \alpha_D & 0 \\
D_1 \otimes d \otimes \alpha & 0 & D_D \otimes D & 0 \\
0 & 0 & D_D^\ast \otimes I_{S_d} & L_D \otimes I_{S_d} \otimes (1 - \frac{1}{S_A})
\end{bmatrix} \quad (4.52) \]

\[ A_2 = \begin{bmatrix}
0 & 0 & 0 & 0 \\
0 & D_0 \otimes t_v \otimes \alpha & 0 & 0 \\
0 & D_0 \otimes d \otimes \alpha & 0 & 0 \\
0 & 0 & 0 & 0
\end{bmatrix} \quad (4.53) \]

As we see, the model has to work with bigger matrices in the last case. In Figure 4.21 and in Figure 4.22 I compare the calculation times of the model, as a function of Access Control List size. During both of the comparisons I used equivalent settings in the model. Poisson process is used for generating the incoming traffic. The intensity of this traffic is \(0.01 \text{ packets/TU}\) in both of the examples. All of the other system and model parameters are the same (\(S_q, S_A, \text{etc.}\)). Only the input interface delay is different between the comparisons in Figures Figure 4.21 and Figure 4.22. In the first comparison (Figure 4.21), the delay is \(S_A = 5\) unit. In the second example (Figure 4.22) the input interface delay is \(S_A = 10\) unit. As it is shown, the difference between the calculation times of the original and the minimized model is evident.

![Diagram](image-url)  

Figure 4.21: Calculation time of the model \((S_A = 5)\)
Meanwhile the calculation examples (Figure 4.21 and Figure 4.22) I derive the mean value of packet delay and the probability of packet loss. The calculations are implemented in Matlab language [44]. I have executed the Matlab code on a personal computer using an Intel(R) Pentium(R) M processor, 1500 Mhz, 512 Mb memory and Microsoft Windows 2000 operation system.

4.6 Simulation results

Similarly to the Section 3.1.3, in order to validate the numerical accuracy of the computational solution, I developed a simulation of the introduced model in NS-2 [41].

The following two diagrams show the accuracy of my minimized Markovian interface model. Figure 4.23 and Figure 4.24 prove that I managed to design an accurate model to derive performance indicators. For shorter lists a relatively small error can result higher relative error ratio, thus for smaller list sizes the relative error might be higher than for longer lists.

During examination of the model and its solution large amount of result were derived and simulated in case of different input parameters. In the presented figures only two curves are given, with the elementary time-unit (TU). The number of packets during a TU can be \(\frac{\text{packets}}{TU} = 0.01\) and \(\frac{\text{packets}}{TU} = 0.05\). The model gives approximately the same results as the simulation. The relative error is lower than \(8 \cdot 10^{-3}\).

This means that with the created model we can calculate very accurate performance parameters in case of different system parameters.
The two main attributes I derived from the model were the overall delay of a query, in this case an IP packet, suffers and the packet loss rate caused by the finite buffer sizes. As I mentioned my model can handle the interface delay as an input parameter. So we can examine different interface types with different speeds in case of different traffic. In Figure 4.25 and in Figure 4.26 four different curves are presented, each one labeled with an $S_A$ and an $S_D$ value. For example, $S_A = 5$ and $S_D = 10$ means that the mean value of time spent at the input and output interfaces is 5 and 10 time-units respectively, while the arrival intensity is constant $\frac{\text{packets}}{\text{TU}} = 0.01$. Figure 4.25 and Figure 4.26 show the affect of the hardware interfaces on the derived performance parameters, like packet delay and packet loss.

These results have shown, that interfaces seriously affect the performance parameters of the system.
4.7 Conclusion

In this thesis group, I propose a mathematical model of effect of hardware interfaces together with an embedded system, which is the already presented ACL system. I introduced the examination and state-space reduction of the model based on two own ideas. For calculating the model result, I proposed first an approximation and second I proposed a punctual calculation method of the packet delay. This punctual calculation method can also be used in the group of the Markovian models, which describe preemptive behavior.

The state-space reduced model and its calculations are validated by NS-2 simulations, where I did not use any approximations.
Chapter 5

Conclusion remarks

In this chapter I summarize my work, adding some ideas for future work.

5.1 Summary of thesis

My research was focused on the problems and capabilities of Access Control Lists and their variations. The general aim was to find and solve open research problems in this field. These thesis groups deal with the examination of ACLs that are used in IP routers mainly for providing network admission control and for maintaining a certain level of quality of service.

The first thesis group in Chapter 2 presents a heuristic optimization of the rule entries of ACLs taking the router traffic into consideration. Here, I propose an idea and a heuristic to solve an NP-hard problem that is the optimization of these lists.

In the second thesis group, after the optimization method I introduce the mathematical modeling of Access Control Lists and other types of lists that are used in telecommunications. With the aid of these models important performance parameters can be derived from the described structures, for example overall latency and packet loss probability, which are affected by these lists. Normal sequential list models and more complicated hierarchical models (BSD solution, general hierarchical lists and TTCNv3 alternative models) are also introduced in this thesis group (Chapter 3), together with their mathematical background. In this group, I also examined the infinite list model, and I proposed an upper bound for its applicability.

In the third thesis group (Chapter 4) I present a Markovian model of these interfaces, because in telecommunication routers and personal computers the system interfaces play significant role on the performance parameters. For development of
an applicable model, some novelty had to be brought into the models. New ideas, state-space reductions and new algorithms for deriving the parameters are required by the applicable and calculable model.

All of my results are proved formally or by simulations, and deductions of the results are given in my thesis groups.

5.2 Future research

In the future, I would like to work with Markovian modeling of other types of software structures in telecom and testing fields, such as parallel test components, and with modeling of more hardware related elements, like the interfaces in this thesis.

It can also be an interesting field to make a conversion between formal descriptions (like UML, SDL, TTCN) and mathematical description (Markovian structures). With this idea, formally specified software and hardware element models could be transformed to a mathematical model that is solvable from the point of view of performance parameters, for example packet delay and packet loss, that packets suffer in the specified system.
## Abbreviations

<table>
<thead>
<tr>
<th>symbol</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACL</td>
<td>Access Control List</td>
</tr>
<tr>
<td>BPF</td>
<td>Berkeley Packet Filter</td>
</tr>
<tr>
<td>DQBD</td>
<td>Discrete-time Quasi Birth-Death process</td>
</tr>
<tr>
<td>FIFO</td>
<td>First In First Out</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hyper Text Transfer Protocol</td>
</tr>
<tr>
<td>IN</td>
<td>Input states</td>
</tr>
<tr>
<td>IOS</td>
<td>Internetwork Operating System</td>
</tr>
<tr>
<td>IP</td>
<td>Internet Protocol</td>
</tr>
<tr>
<td>ISP</td>
<td>Internet Service Provider</td>
</tr>
<tr>
<td>LAN</td>
<td>Local Area Network</td>
</tr>
<tr>
<td>MAP</td>
<td>Markov Arrival Process</td>
</tr>
<tr>
<td>M/G/1</td>
<td>Markovian arrival/ General departure/ 1 server</td>
</tr>
<tr>
<td>NLANR</td>
<td>National Laboratory for Applied Network Research</td>
</tr>
<tr>
<td>NS2</td>
<td>Network Simulator version 2</td>
</tr>
<tr>
<td>OUT</td>
<td>Output states</td>
</tr>
<tr>
<td>PC</td>
<td>Personal Computer</td>
</tr>
<tr>
<td>PDU</td>
<td>Protocol Data Unit</td>
</tr>
<tr>
<td>PH</td>
<td>Phase-Type process</td>
</tr>
<tr>
<td>QBD</td>
<td>Quasi Birth-Death process</td>
</tr>
<tr>
<td>SDL</td>
<td>Specification and Description Language</td>
</tr>
<tr>
<td>SLA</td>
<td>Service Level Agreements</td>
</tr>
<tr>
<td>TCP</td>
<td>Transmission Control Protocol</td>
</tr>
<tr>
<td>TTCNv3</td>
<td>Testing and Test Control Notation version 3</td>
</tr>
<tr>
<td>UDP</td>
<td>User Datagram Protocol</td>
</tr>
<tr>
<td>UML</td>
<td>Unified Modeling Language</td>
</tr>
</tbody>
</table>
Glossary of symbols

<table>
<thead>
<tr>
<th>symbol</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathbf{A}$</td>
<td>Matrix A</td>
</tr>
<tr>
<td>$\mathbf{t}$</td>
<td>Vector t</td>
</tr>
<tr>
<td>$A \leftarrow B$</td>
<td>A depends from B</td>
</tr>
<tr>
<td>$\exists C$</td>
<td>exists C</td>
</tr>
<tr>
<td>$A \subset B$</td>
<td>A subset of B</td>
</tr>
<tr>
<td>$A \subseteq B$</td>
<td>A is a subset or equal to B</td>
</tr>
<tr>
<td>$A \mathcal{R}$</td>
<td>A is redundant</td>
</tr>
<tr>
<td>$TU$</td>
<td>Discrete Time Unit</td>
</tr>
<tr>
<td>$N(t)$</td>
<td>Level process</td>
</tr>
<tr>
<td>$J(t)$</td>
<td>Phase process</td>
</tr>
<tr>
<td>$\lambda$</td>
<td>Parameter of the Poisson traffic</td>
</tr>
<tr>
<td>$A_0$</td>
<td>Arrival process in DQBD</td>
</tr>
<tr>
<td>$A_1$</td>
<td>Transitions inside a level</td>
</tr>
<tr>
<td>$A_2$</td>
<td>Departure process in DQBD</td>
</tr>
<tr>
<td>$A^*$</td>
<td>An irregular description matrix</td>
</tr>
<tr>
<td>$\otimes$</td>
<td>Kronecker-product</td>
</tr>
<tr>
<td>$E(X)$</td>
<td>Mean value of $X$</td>
</tr>
<tr>
<td>$P_r(X)$</td>
<td>Probability of $X$</td>
</tr>
<tr>
<td>$I_{\mathbb{N}}$</td>
<td>Intensity matrix with size $N \times N$</td>
</tr>
<tr>
<td>$[X]^+$</td>
<td>Positive value of expression $X$</td>
</tr>
<tr>
<td>$\Omega_N$</td>
<td>Shifting matrix with size $N \times N$</td>
</tr>
<tr>
<td>$\text{diag} \left&lt; \mathbf{X} \right&gt;$</td>
<td>Diagonal matrix with the vector $\mathbf{X}$ in the diagonal</td>
</tr>
</tbody>
</table>
\[ \min(X, Y) \] The minimum of \( X \) and \( Y \)

\# of \( X \) Number of \( X \)

\( \approx \) Approximately

\( \{1, 1\} \) The 1,1 matrix block

\( \{X\} \) Set of states of \( X \)

\( \gamma_X \) The moment of stepping into a state of the set \( X \)

\( \neq \) Not equals

\( P(X|Y) \) Probability of \( X \) with condition \( Y \)
Bibliography

Own articles

Journals


LNCS Articles


Conference Papers


Other references


[14] The OpenBSD project.
http://www.openbsd.org


http://www.nlanr.net/


[27] Cisco System’s HomePage  
http://www.cisco.com


http://www.perl.com/


http://advanced.comms.agilent.com/RouterTester/


[41] The discrete event Network Simulator - NS-2, from the University of Southern California, Information Sciences Institute.
http://www.isi.edu/nsnam/ns/.


http://www.mathworks.com/

