A Formal Model for Network-wide Security Analysis

Petr Matoušek Jaroslav Ráb Ondřej Ryšavý
Miroslav Švéda
Brno University of Technology
Faculty of Information Technology
Bozetechova 2, 612 66 Brno, Czech Republic
{ matousp, rabj, rysavy, sveda }@fit.vutbr.cz

Abstract

Network designers perform challenging tasks with so many configuration options that it is often hard or even impossible for a human to predict all potentially dangerous situations. In this paper, we introduce a formal method approach for verification of security constraints on networks with dynamic routing protocols in use. A unifying model based on packet-filters is employed for modelling of network behaviour. Over this graph model augmented with filtering rules over edges verification of reachability properties can be made. In our approach we also consider topology changes caused by dynamic routing protocols.

1. Introduction

Security and safety of computer networks against different types of attacks or unexpected failures is an important task for network administrators. Network devices like firewalls, flow monitors, or intrusion detection/prevention systems become an important part of commercial LAN and WAN networks. These devices can mitigate possible attacks and failures but cannot prove that the network keeps its performance and response time when topology or state of links have changed. Traditional approaches to check the correctness of the network design encompass testing and traffic monitoring.

Using testing and network monitoring, we can analyse only the current state of the network. Test programs like traceroute or ping can reveal if ICMP packets can go from one end point to the other end point of the network. However, when a link on the route goes down, the network may converge into a new topology, and new connection between end points may be filtered by firewall rules on an intermediate router. Unfortunately, this behavior cannot be guessed using testing before the failure of the link have appeared.

An alternative approach is to build a formal model of the computer network and make a thorough analysis of all its behavior under certain conditions (links down, adding packet filters, etc.). For the analysis, formal verification techniques such as model checking [**?**] or static analysis [1] can be employed. In this way the required property is examined in all possible states of the network configuration that increases the confidence in proper functionality of the network. In the case of property violation the problem can be easily detected and the misconfiguration that caused the functional failure or security risk can be tracked.

1.1. Contribution of the Paper

The main contribution of the paper consists in the development of an analytic model and discussion on a possible analysis approach revealing issues that relate to the problem of verification of properties in a network with variable topology.

The network is modelled as a graph, where vertices are network devices and edges stand for communication links. ACLs and routing policies are reflected into the model by means of packet filtering functions that are associated with edges of the graph. This unified model of the network was first introduced in [2].

We provide a procedure that converts an Access Control List (ACL) in a packet filtering function as this representation is computationally more efficient. ACLs can be seen as an ordered sequence of filtering rules. To evaluate such a sequence of many rules (entries) with different policies (deny/permit) is a complicated issue. Instead of evaluating each rule one by one, we transform this ordered set of rules into a single quantifier-free first-order logical formula called a packet filter. We prove that the transformation is correct and present an algorithm of the transformation. We also propose to use Interval Decision Diagrams (IDDs)[3], an efficient data structure, to implement and manipulate these

formulas.

The topology of a network may change as links go up and down. The network state is expressed as the state of all links on the network. We define a network transition system as a graph of network states. This transition system models the behaviour of the network under link failures. When a link fails, the network state changes. There happens not only topological changes, but routes of data flows change too, according to routing information. To reflect these changes we need the model of a routing process that allows us to compute changes of network-wide routing information for each network state. This is important for reachability analysis of the network under link failures.

The state transition system comprises all states of the network under different failures of links. There are 2^l possible network states, where l is the number of links of the network that can possibly fail. This finite transition system can be analysed using model checking verification methods. As the number of states is exponential we need to tackle the state explosion problem. We discuss a possible approach bounding the number of states that need to be explicitly checked. It is based on the observation that some properties only hold in a continuous region of the state space.

1.2. Structure of the Paper

The structure of the paper is following. The next section deals with building a formal model of a network that allows for description of a network topology, ACLs and routing policies. Algorithms for conversion of an ACL to a packet filter function and routing information base to packet filter functions are defined and their correctness is proved. The third section introduces a network transition system, abstract model of distance vector routing protocol, and considers the verification method for the network reachability analysis computed for a complete set of network states. The last section summarizes the paper, overviews the related work and discusses the future work.

2. Formal Model of the Network

The aim of this section is to provide a formal model of a network topology that allows us to specify a set of attributes for security analysis. To do so we introduce a graph-based formal network model with packet filter functions classifying the message flow thus constraining the reachability of the entities on the network. For the rest of the paper we refer to the example of the network as given in figure 1.

2.1. Formal description of the network

One of contributions at this paper is the formal model of the network from point of view of routing processes and

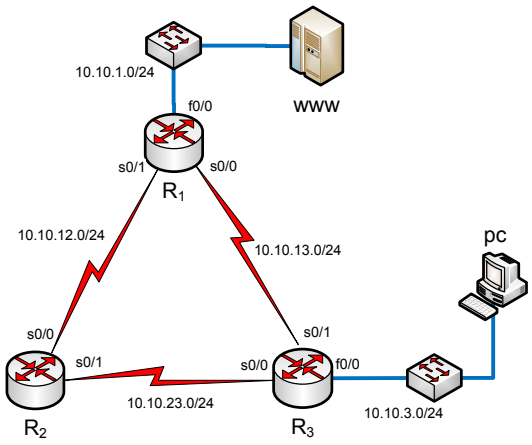


Figure 1. Network topology example

filtering. Our approach combines techniques introduced in [2], [4], and [5]. The network is modeled by a directed graph where vertices are routing devices and edges are communication channels that form abstraction of communication links. Each communication link is modeled by a pair of unidirectional communication channel.

Definition 1 (Network)**.** A network *is a tuple* $N = \langle \mathcal{R}, \mathcal{L}, \mathcal{F} \rangle$*, where*

- $\mathcal{R}$ *is a finite set of network devices,*
- $\mathcal{L} \subseteq \mathcal{R} \times \mathcal{R}$ *is a finite set of links between routers, such that for every physical link between* R_1 *and* R_2 *there is a pair of channels* $l_{12} = \langle R_1, R_2 \rangle$ *and* $l_{21} = \langle R_2, R_1 \rangle$*, and*
- $\mathcal{F}$ *is a finite set of filtering rules assigned to each edge of the graph.*

Because filters can be applied in both directions of the link, we suppose that the set $\mathcal{L}$ contains for every link two items $\langle R_i, R_j \rangle$ and $\langle R_j, R_i \rangle$.

On real networks there are other network device then routers. However, every end-point device like PC or Web server can be described as a router with only one interface, and one outgoing filtering rule representing routing all traffic to default gateway. According to the previous definition the network model for our running example is a graph as shown in figure 2.

Geoffrey G.Xie et al. in [2] show that routing information from the network can be added to the static model of the network using additional filtering rules. These filtering rules can change as the state of links change, so the filtering rules depend on the actual state of the network. We analyze the network states in section 3 of this paper.

In the rest of the paper, we thoroughly refer to the definitions of IP addresses and a structure describing an IP packet

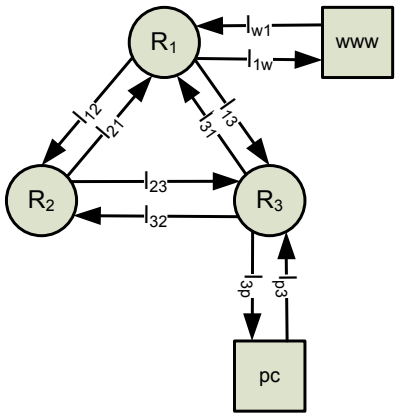


Figure 2. Network topology example
End-point devices are distinguished from routers by different symbol only for the clarity of the presentation.

header. In all definitions, a set of theoretical notations enriched by the notions of records and lists is used. A set of records is written as $\langle l_1 : A_1, \ldots, l_n : A_n \rangle$,

where l_i are labels and A_i are sets defining domains for fields of the record. An element of the set is a record $\langle l_1 = a_1, \ldots, l_n = a_n \rangle$,

where $a_i \in A_i$ for all $1 \leq i \leq n$. The list is written as $\langle a_1, \ldots, a_n \rangle$ and two predicates are defined. Predicate $head(l, a)$ tests if a is a first element of the list. Predicate $tail(l, t)$ tests if t is a tail of list a. Also the following abbreviations are used. Term $x \in (i..j)$ stands for $i \leq x \leq j$, and $Seq(i..j)$ stands for the set of all possible intervals with boundaries i and j. Term $[12..45]$ is an example of the element of interval set $Seq(0..100)$.

Definition 2 (IP Address Representation)**.** *The IP address structure is defined as four octets, usually delimited with dot separator.*

$$\text{IP} = \{a_1.a_2.a_3.a_4 \ : \ a_i \in (0..255), i \in \{1, 2, 3, 4\}\}$$

The IP address interval structure represents a contiguous address space defined as interval possibly in every octet.

$$\text{IPINT} = \{a_1.a_2.a_3.a_4 \ : \ a_i \in Seq(0..255), i \in \{1, 2, 3, 4\}\}$$

The IP address with network mask is a tuple whose first component is IP address and the second component is a number of network bytes.

$$\text{IPNET} = \{\langle a, m \rangle, a \in \text{IP} \wedge 0 \leq m \leq 32\}$$

For instance, an address of interface s0/0 on router R_2, 10.10.12.2, is an entity of IP set. IPINT is a set of intervals of IP addresses. A network connecting routers R_2 and R_3 can be expressed as an interval $10.10.12.[0..255] \in \text{IPINT}$. In this case, an alternative representation uses a network prefix length, e.g., $\langle 10.10.12.0, 24 \rangle$. We will use more convenient notation $10.10.12.0/24$. It is easy to see that interval representation is more general as there are intervals of addresses that cannot be represented using prefix notation.

The fields of header record defined bellow does not correspond directly to the structure of a real header of IP packets. Only source and destination address fields, and protocol identification are considered for IP header. Source port and destination port fields come from TCP/UDP header. This representation abstracts from the data carried in the IP header that are not significant for modeling and analysis shown later.

Definition 3 (Header Structure)**.** *An L3/L4 header is defined as a record of the following structure:*

$$\begin{aligned}\text{IPHDR} = \langle & proto \ : \ \{ip, icmp, tcp, udp\}, \\ & srcIp \ : \ \text{IP}, dstIp \ : \ \text{IP}, \\ & srcPort \ : \ (0..65535) \\ & dstPort \ : \ (0..65535)\rangle\end{aligned}$$

We also define an index set of header fields

$$\text{HDRITEMS} = \{proto, srcIp, dstIp, srcPort, dstPort\}.$$

Note that set of protocols contains only four elements. RFC 790[6] defines protocol numbers that can appear in the protocol fields of IP header. In this paper only protocols ICMP, TCP, and UDP are considered. All others are specified as just IP protocol without to distinguish upper layer protocols.

Considering the header structure defined above, an HTTP request message sent by device PC to a web browser on WWW is represented as $\langle proto = tcp, srcIp = 10.10.3.2, dstIp = 10.10.1.2, srcPort = 13244, dstPort = 80\rangle$ record.

2.2. Representation of ACL as a Filtering Function

An important issue for a formal automatic analysis is how to represent the data to be analysed. In our case, we need to find a way how to represent access control lists (ACLs) that are used to filter traffic on the routers. Their representation should be efficient for further analysis that includes adding new rules, searching (packet matching over the rules), test of equality over set of rules, or canonical representation. We consider ACLs as configured on Cisco Routers [7].

Briefly, an access control list (ACL) is an ordered sequence of filtering rules that permit or deny specific traffic from/to given nodes/networks. The following example describes an ACL 1 that permits only a HTTP traffic originating from network 10.10.0.0 by the first rule. Other traffic

from that network is prohibited by the middle rule. Nodes with other source addresses can communicate without any restrictions as permitted by the last rule.

```
permit tcp 10.10.0.0/16 any www
deny ip 10.10.0.0/16 any
permit ip any any
```

Our goal is to represent such ordered lists of filtering rules in a way that is efficient for a traffic analysis on the network. Christiansen and Fleury show in [5] that filtering rules on firewalls can be viewed and represented as lists of logical formulas.

Filtering function on header h can be expressed as a predicate in conjunctive form. For example, functions η_1, η_2 and η_3 are representatives for ACL rules above.

$$\begin{aligned}\eta_1(h) &= (h.proto \in Tcp) \wedge (h.srcIp \in 10.10.0.0/16) \\ &\quad \wedge (h.destPort = www) \\ \eta_2(h) &= (h.proto \in Ip) \wedge (h.srcIp \in 10.10.0.0/16) \\ \eta_3(h) &= (h.proto \in Ip)\end{aligned}$$

We define $Ip = \{ip, udp, tcp\}$ since every TCP or UDP traffic can be classified as IP traffic as well, and therefore eligible to be filtered by an IP rule. For uniform representation we similarly define $Tcp = \{tcp\}$, $Udp = \{udp\}$,and $Icmp = \{icmp\}$.

Definition 4 (ACL Rule)**.** *An ACL rule is given as follows:*

$$\begin{aligned}\text{RULE} = \langle &action : \{permit, deny\}, \\ &match : \text{IPHDR} \rightarrow \text{BOOLEAN}\rangle\end{aligned}$$

Matching function $r.match$ over rule r is defined as a boolean expression in the conjunctive normal form:

$$r.match(h) = \bigwedge\nolimits_{x_i, i \in \text{HDRITEMS}} h.x_i \in c_i$$

Constants c_i are sets such that $c_i \subseteq Dom(x_i)$.

An ACL (or packet filter) is an ordered sequence of rules r_i. For ACL 1 (see above), filter φ_1 is given as follows:

$$\begin{aligned}&\varphi_1 = \langle r_1, r_2, r_3\rangle \\ &r_1 = \langle action = permit, match = \eta_1\rangle \\ &r_2 = \langle action = deny, match = \eta_2\rangle \\ &r_3 = \langle action = permit, match = \eta_3\rangle\end{aligned}$$

Definition 5 (Access Control List (ACL))**.** *An Access Control Lists (ACL) is a list of ACL rules. A set of ACLs is defined as follows:*

$$\text{ACL} = \{l : l = \langle r_1, \ldots r_n\rangle, r_i \in \text{RULE}, 1 \leq i \leq n\}$$

Such a representation requires a strict evaluation method. The order of the rules is important because if there is a match on the header η_i, the corresponding policy π_i is taken and no other matching is done. If η_i does not match a packet, the following rules are tested for matching until a match is found, or the last rule is examined. If no rule matches, no policy is applied on a packet, and the packet is allowed to pass. However, a real implementation of ACLs adds a default rule at the end. This rule of the form $r_\infty = \langle action = deny, match = (h \mapsto \text{TRUE})\rangle$ drops every traffic. Further we assume that ACL always contains such implicit deny rule.

Definition 6 (ACL Evaluation)**.** *An algorithm that evaluates ACL for the given header is a recursive function* $AclEval : \text{IPHDR} \times \text{ACL} \rightarrow \{Permit, Deny\}$ *defined as follows:*

- $AclEval(h, a) = Deny$ *if* a *is empty list,*
- $AclEval(h, a) =$ **if** $r.Match(h)$ **then** $r.Action$ **else** $AclEval(h, t)$ *if* $head(a, r)$ *and* $tail(a, t)$.

We show how to represent ACL as a single quantifier-free predicate formula that is more suitable for efficient memory representation and evaluation, for instance, using BDD like structures.

The procedure that computes such a predicate from the ACL structure takes iteratively all the rules from the beginning to the end of an ACL list, and for each rule it gives a boolean expression as follows:

- If an action of the rule is permit, it yields disjunction of a match expression of the rule with the rest of the ACL.
- If the action of the rule is deny, it yields conjunction of negation of a match part of the rule with the rest of the ACL.

When the end of the ACL is reached, an implicit deny rule is processed at the same way.

Definition 7 (Filter Function)**.** *A packet filter function* $\psi_a : \text{IPHDR} \rightarrow \text{BOOLEAN}$ *for ACL* a *is a function defined as follows:*

- $\psi_a(h) = \text{FALSE}$ *if* a *is an empty ACL list, or*
- $\psi_a(h) = \psi_t(h) \vee r.Match(h)$ *if* $r.Action = Permit$*, or*
- $\psi_a(h) = \psi_t(h) \wedge \neg r.Match(h)$ *if* $r.Action = Deny$ *when* r *is the head and* t *is the tail of ACL* a*, and* $\psi_t(h)$ *is a packet filter function for ACL* t.

Using this procedure, we can express filter φ_1 by equivalent predicate ψ_1 that is equivalent to ACL 1.

$$\psi_1(h) \quad = \quad \eta_1(h) \vee (\neg\eta_2(h) \wedge \eta_3(h))$$

Theorem 1 (Filter Function Correctness)**.** *Assume that* TRUE $= Permit$ *and* FALSE $= Deny$*. Then, for any ACL a and any header h functions* $\psi_a(h)$ *and* $AclEval(h, a)$ *give the same result. Formally,*

$$(\forall h \in \text{IPHDR}, a \in \text{ACL})\psi_a(h) = AclEval(h, a)$$

Proof is done by induction on the ACL list and then by case analysis.

2.3. IDDs as an efficient data structure for ACL

As showed before, ACL can be expressed as a single first-order logic formula. For computation of conjunction, disjunction, etc., we can represent this formula using different data structures – Binary Decision Diagrams (BDDs, [8]), Difference Bound Matrices (DBMs, [9]), Covering Sharing Trees (CSTs, [10]), Difference Decision Diagrams (DDDs, [11]), etc. By operations defined over these data structures, we can easily manipulate the formula.

Filtering rules usually work with a range of IP addresses or port numbers. To represent a range of such kinds, intervals can be very efficient in the matter of space storage and computational time. The structure *interval decision diagrams* (IDD) [3] allows us to perform an easier classification on integer numbers within a finite domain.

Definition 8 (Interval Decision Diagram)**.** *An Interval Decision Diagram (IDD) is a rooted, directed acyclic graph with two types of nodes (terminal and non-terminal) such, that*

- *One or two terminal nodes of out-degree zero are labeled 0 or 1.*
- *Nonterminal nodes form a set of* u *of out-degree* $deg(u) \leq |Dom(u)|$*. Variable* $var(u)$ *is associated with each node.*
- $val(u, v) = c$ *is a valuation of the edge from node* u *to node* v*,* $c \in Dom(u)$*.*
- $val(u, v) = \perp$ *if there is not an edge between nodes* u *and* v*.*

Now, we recall how IDD can be used for encoding a range of IP addresses. The IP address fields in ACLs are given using wildcard masks. For these fields, the intervals are deduced according to these wildcard masks, e.g.,

$$10.10.0.0/16 \quad \rightarrow \quad \{10\}.\{10\}.[0-255].[0-255]$$

The advantage of such representation will become evident when combining several IDDs. If some octets have the same value they can be merged. Octets whose values form a continuous range are represented by one interval. The following interval representation combines $10.10.0.0/16$ and $10.11.0.0/16$.

$$\{10\}.[10-11].[0-255].[0-255]$$

A filter function matches IP headers on the basis of several parameters, e.g, source address, destination address, protocol type, and port numbers. For IDD representation, it is important that we are able to define an interval cover on the domain of every header field. That allows us to split the domain into intervals that are covered in the IDD graph.

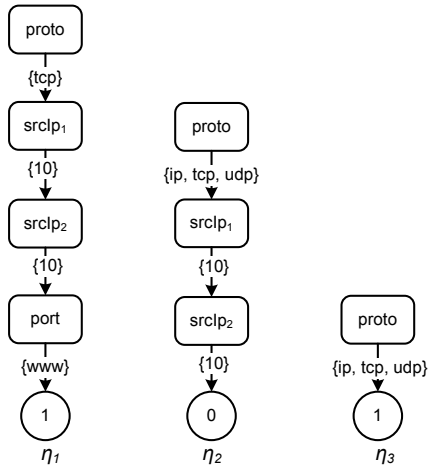


Figure 3. Example of IDD representation of ACL rules

Definition 9 (Intervals in Domains)**.** *The set* $I(\text{IPHDR}_i) = \{I_1, I_2, \ldots, I_r\}$ *represents an* interval cover of a domain IPHDR_i (for $i \in$ HDRITEMS) *iff (i)* $\forall j, k \in \{1, \ldots, r\}$: $I_j \cap I_k = \emptyset$ *(disjoint intervals), and (ii)* $\text{IPHDR}_i = I_1 \cup I_2 \cup \ldots \cup I_r$ *(a complete cover).*

Every matching function of the rule can be represented using IDD. For instance, the IDD representations of a matching functions η_1, η_2, η_3 are depicted in Figure 3.

Definition 10 (IDD Match)**.** *Let* $x = \langle x_1, \ldots, x_n \rangle, n = |\text{HDRITEMS}|$ *is an ordering of elements from the set* HDRITEMS*. IDD structure* σ_r^x *with nonterminals nodes* $x_i \in$ HDRITEMS *represents a matching function of ACL rule* r *if* $\forall i, 1 \leq i \leq n-1$*:*

- $val(x_i, x_{i+1}) = c_i$ *if* $r.match$ *contains* $h.x_i \in c_i$*,*
- $val(x_i, 0) = Dom(x_i) \setminus c_i$ *otherwise.*
- $val(x_n, 1) = c_n$ *if* $r.match$ *contains* $h.x_n \in c_n$*,*

- $val(x_n, 0) = Dom(x_n) \setminus c_n$ *otherwise.*

Similarly to the construction of ψ function, the corresponding IDD can be built using basic logical operators. The existence of an equivalent IDD structure for an arbitrary packet filter function is stated by the following theorem.

Theorem 2 (IDD Filter)**.** *For any ordering x of elements from set* HDRITEMS*, is is possible to construct an IDD structure σ_l^x for an arbitrary packet filter function ψ_l such that $\sigma_l^x(h) = \psi_l(h)$.*

Proof is done by construction of IDD structure σ_l that follows recursive structure of ψ_l predicate function. It assumes that all used logical operators ($\vee, \wedge, \neg$) are defined over IDD structure, and preserve the intended semantics.

Figure 4 shows an example of IDD representation for φ_1 using the ordering $x = \langle proto, srcIp, dstIp, srcPort, dstPort \rangle$. The IDD is a result of combination of matching functions η_1, η_2, η_3 and optimization steps that removed duplicities.

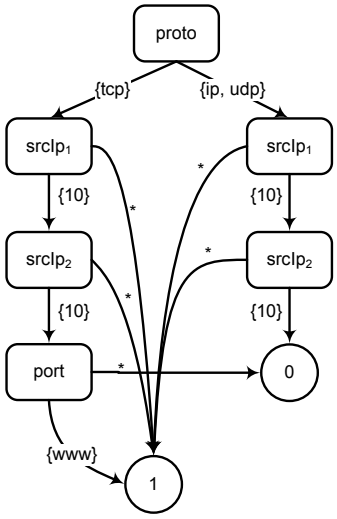


Figure 4. Example of IDD representation

It is known that the structure of IDD depends on the order of variables.

It is apparent from the example above, that using different order of variables can lead to more efficient representation. The study of the methods for determining the best ordering of variables is beyond the topic of this paper.

2.4. Converting a Routing Table to a Packet Filter Function

On real networks, data is delivered over various links connecting a transmitting device with an end host. Layer 3 of OSI model provides end to end connection over intermediate nodes. It uses special network devices, routers, to find "the best route" to the receiver. Router has usually several interface cards that interconnect neighbors networks. Router also keeps routing table—a list of known networks with corresponding outgoing interface. If a router receives a packet, it lookups its destination network in the routing table. If found, it sends it by the best route on the corresponding interface.

A packet is forwarded to the destination based on the routing table entries. If no route is found in the table, packet is discarded and ICMP message is sent to the originator of the packet. The routing table in Table 1 contains routes to the networks 10.10.12.0/24, 10.10.23.0/24, and 10.10.13.0/24 with outgoing interfaces Fast Ethernet 0/1 (f0/1), serial 0/0 (s0/0), and serial 0/1 (s0/1) see Figure 1.

In Table 1, we can see that there are two routes to the same destination network 10.10.23.0—via interfaces s0/1 and s0/0. These routes are equivalent because they have the same metric. If one of the routes fails, the table keeps only an entry to the working network. If both links fail, there will not be any route to the destination network and packets headed to this network will be discarded.

```
C 10.10.12.0/24 is directly connected, s0/1
R 10.10.23.0/24 via 10.10.12.2, s0/1
                via 10.10.13.3, s0/0
C 10.10.13.0/24 is directly connected, s0/0
C 10.10.1.0/24  is directly connected, f0/0
R 10.10.3.0/24  via 10.10.13.2, s0/0
```

Table 1. Routing table for router R1

Routing tables may contain static and dynamic routing information. If we use dynamic routing and network topology is changed, routing process on the router detects the change. Then routing protocol distributes this change to all other routers on the network within an administrative domain. Distribution of routing information depends on type of the protocol–distance vector, or state link protocol. For our analysis, it means that the contents of routing tables can be changed at any time. We have to consider it in our analysis of the network behaviour.

It is very efficient to hold routing information as a special kind of packet filters – see unifying model by [2]. If a route to the destination exists in the routing table, we can add a permit rule as a new ACL to the outgoing interface. Other traffic is denied by a default deny rule.

Now we will formally define the routing table and show how to convert routing table into packet filter function defined in the previous section.

Definition 11 (Routing table entry)**.** A routing table entry *is a tuple $rt_i = \langle d_i, l \rangle$, where $d_i \in$* IP *is a destination net-*

work address with its mask, $l \in \mathcal{L}$ represents an outgoing interface, and i *is an index of the entry in the routing table.*

Definition 12 (Routing table)**.** The routing table $rt(R) = \{rt_1, \ldots, rt_n\}$ *is a set of routing table entries on router* $R \in \mathcal{R}$, n *is the number of routing table entries.*

Every routing table entry of the form $rt_i = \langle d_i, l \rangle$ can be converted into rule $r(h) = ((h.proto = ip) \wedge (h.dstIp = d_i))$[1]. This transformation is straightforward—the rule expresses semantically the same thing, i.e., a routing table entry forwarding packets with destination address d_i to the interface l corresponds to a firewall rule, that permit on l only packets with the destination address d_i.

If the rule $r(h)$ is applied on the interface connected by link $l \in \mathcal{L}$, we write $r_l(h)$.

Similarly, routing table $rt(R)$ of the router $R \in \mathcal{R}$ (i.e., a set of rules) can be converted into a set of packet filters $\psi_{li}(h), l_i \in \mathcal{L}$, that are applied on all interfaces $l_i \in \mathcal{L}$. Each packet filter $\psi_{li}(h)$ may contain routes to different networks that use the same outgoing interface.

Formally, routing table $rt(R)$ is transformed to a set of filter predicates $\psi_l(h)$:

$$\psi_l(h) = \bigvee_{\langle d_i, l \rangle \in rt} (h.proto = ip \wedge h.dstIp = d_i)$$

where $l \in \mathcal{L}$ is a link connected to the router R.

Consider the routing table $rt(R_1)$ from Table 1. This table can be converted into three packet filter function $\psi_1(h)$ (for s0/1), $\psi_2(h)$ (for s0/0), and $\psi_3(h)$ (for f0/0):

$$\begin{aligned}
\psi_1(h) &= (h.proto = ip \wedge h.dstIp = 10.10.12.0/24) \\
&\vee (h.proto = ip \wedge h.dstIp = 10.10.23.0/24) \\
\psi_2(h) &= (h.proto = ip \wedge h.dstIp = 10.10.13.0/24) \\
&\vee (h.proto = ip \wedge h.dstIp = 10.10.23.0/24) \\
&\vee (h.proto = ip \wedge h.dstIp = 10.10.3.0/24) \\
\psi_3(h) &= (h.proto = ip \wedge h.dstIp = 10.10.1.0/24)
\end{aligned}$$

2.5. Classless Inter–Domain Routing

Transformation of packet filter as showed in the previous part works fine for classful routing. However, if we consider classless routing with a variable subnet mask, it will give unprecise results. Suppose two networks—10.10.12.0/24 pointing to interface f0/0, and 10.10.12.192/26 pointing to interface f0/1. Packet filter functions for these networks will be $\psi_{f0/0}(h) = (h.proto = ip \wedge h.dstIp = 10.10.12.0/24)$, and $\psi_{f0/1}(h) = (h.proto = ip \wedge h.dstIp = 10.10.12.192/26)$. If there is a packet with destination address 10.10.10.12.193, it will match both predicates. However, the router should provide the longest prefix match on the IP address. That means, if the packet matches more than one rule, the rule with more matched bits is selected. In our case, only the second function should be applied.

To deal with classless routing, we first need to define a function that will test if an IP address A is a subnet of an IP address B.

Definition 13 (isSubnet function)**.** *Let* $isSubnet$: IPNET $\times$ IPNET $\rightarrow$ BOOLEAN *be a function such that* $isSubnet(A, B) =$ TRUE *if and only if network address* A *is subnet of network address* B.

In our case, $isSubnet(10.10.12.192/26, 10.10.12.0/24)$ is true because 10.10.12.192/26 is a subnet of 10.10.12.0/24.

Then, the packet filter function $\psi_l(h)$ has to contain a rule that permits the network address but forbids all its possible subnets that exist in routing table rt. Formally, $\psi_l(h) =$

$$\begin{aligned}
(i) \quad & \bigvee_{\langle d_i, l \rangle \in rt} \Big((h.proto = ip \wedge h.dstIp = d_i) \\
& \bigwedge_{\langle d_j, k \rangle \in rt, k \neq l} \neg (h.proto = ip \wedge h.dstIp = d_j) \Big) \\
& \text{iff } \exists \langle d_j, k \rangle : isSubnet(d_j, d_i) \\
(ii) \quad & \bigvee_{\langle d_i, l \rangle \in rt} (h.proto = ip \wedge h.dstIp = d_i) \\
& \text{otherwise}
\end{aligned}$$

In our example above, $\psi_{f0/1}$ will not be changed. Only packet filter function $\psi_{f0/0}$ will be extended as follows: $\psi_{f0/0} = ((h.proto = ip \wedge h.dstIp = 10.10.12.0/24)) \wedge \neg(h.proto = ip \wedge h.dstIp = 10.10.12.192/26)$.

2.6. Adding Routing Information and ACLs into the Network Model

In the previous text, we showed how both ACL and routing information can be represented by a packet filter function $\psi_l(h)$ applied on the interface $l \in \mathcal{L}$. Now we can join these two packet filter function together, i.e., we create a unified network model that contains both filtering and routing information. Let $\mathcal{R}$ is a set of routers of the network and $\mathcal{L}$ is a set of all links. Then, a unified model is a triple $N = \langle \mathcal{R}, \mathcal{L}, \mathcal{F} \rangle$, where $\forall l \in \mathcal{L} : \psi'_l(h) \wedge \psi_l(h) \in \mathcal{F}_l$, $\psi'_l(h)$ represents ACL related to interface l and $\psi_l(h)$ specifies routing information bound with l. If there is no ACL related to l, we omit $\psi'_l(h)$. If there is no routing information bound with l, we have to add a default rule $\psi_l(h) = \neg((h.proto = ip) \wedge (h.dstIp = any))$ to l that filters out all possible traffic. In our examples, we omit this default rule for brevity.

Suppose the network from Figure 5 with one ACL called ACL1 allowing DNS traffic[2] on link $\langle R_3, R_2 \rangle$ only. Then,

[1] We suppose routing protocols over IP only.

[2] protocol UDP, destination port 53

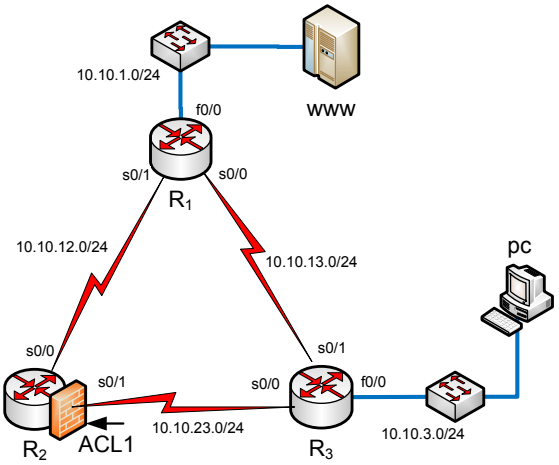


Figure 5. Model of the network N with one ACL

we get the following model: $N_1 = \langle \mathcal{R}, \mathcal{L}, \mathcal{F} \rangle$, where $\mathcal{R} = \{R_1, R_2, R_3, WWW, PC\}$, $L = \{\langle R_1, R_2\rangle, \langle R_2, R_1\rangle, \langle R1, WWW\rangle, \langle WWW, R1\rangle, \langle R_1, R_3\rangle, \langle R_3, R_1\rangle, \langle R_2, R_3\rangle, \langle R_3, R_2\rangle, \langle R3, PC\rangle, \langle PC, R3\rangle\}$, and $\forall i \in \mathcal{L}, F_i \in \mathcal{F}$ where F_i is defined as follows:

$$
\begin{aligned}
&F_{\langle R_1,R_2\rangle}(h) = \\
&\quad \vee\, (h.proto = ip \wedge h.dstIp = 10.10.12.0/24) \\
&\quad \vee\, (h.proto = ip \wedge h.dstIp = 10.10.23.0/24) \\
&F_{\langle R_1,WWW\rangle}(h) = \\
&\quad (h.proto = ip \wedge h.dstIp = 10.10.1.0/24) \\
&F_{\langle R_1,R_3\rangle}(h) = \\
&\quad \vee\, (h.proto = ip \wedge h.dstIp = 10.10.13.0/24) \\
&\quad \vee\, (h.proto = ip \wedge h.dstIp = 10.10.23.0/24) \\
&\quad \vee\, (h.proto = ip \wedge h.dstIp = 10.10.3.0/24) \\
&\ldots \\
&F_{\langle R_3,R_2\rangle}(h) = \\
&\quad \wedge\, (\vee\, (h.proto = ip \wedge h.dstIp = 10.10.23.0/24) \\
&\qquad \vee\, (h.proto = ip \wedge h.dstIp = 10.10.12.0/24)) \\
&\quad \wedge\, (h.proto = udp \wedge h.dstPort = 53) \\
&\ldots
\end{aligned}
$$

This model describes how packets on the network are filtered with respect to both current ACLs applied on links and routing information. However, that model describes only one network state where all links are up and working correctly. We will cover the case of failures within the following discussion about analysis approach.

3. Analysis Approach

For the network state shown above, we can verify reachability between any two routers from R. Suppose we are interested if there is a path for WWW requests (destination port number 80) coming from PC to WWW. This property can be expressed by the formula $\varphi(h) = (h.proto = tcp) \wedge (h.srcIp = PC) \wedge (h.dstIp = WWW) \wedge (h.dstPort = 80)$ stating, that every packet h with required header fields satisfies the property.

Before presenting how our network model can be analysed, we need to define several terms.

Definition 14 (Path)**.** *A path between two routers $R, R' \in \mathcal{R}$ on the network $N = \langle \mathcal{R}, \mathcal{L}, \mathcal{F} \rangle$ is a sequence of routers $r_i \in \mathcal{R}$ and links $l_i \in \mathcal{L}$ with filters $F_i \in \mathcal{F}$ as follows:*

$$
\begin{aligned}
\pi(R, R') \quad &= \quad (R, \langle R, R_1\rangle, R_1, \langle R_1, R_2\rangle, R_2, \ldots, R_k, \\
&\qquad \langle R_k, R'\rangle, R') \\
\textit{where} \quad &\qquad F_{\langle R,R_1\rangle} \wedge \ldots \wedge F_{\langle R_k,R'\rangle} \textit{ holds.}
\end{aligned}
$$

There can be more than one path between two routers.

Definition 15 (Network Reachability)**.** *We define network reachability $NetReach(R)$ on the network $\langle \mathcal{R}, \mathcal{L}, \mathcal{F} \rangle$ to be a set of routers reachable from router R:*

$$NetReach(R) = \{R' \in \mathcal{R} \mid \exists \pi(R, R_k), R_k = R'\}$$

Usually we put restrictions on the path between two routers, e.g., we verify if there exists a path between two routers for Web traffic. This restriction can be expressed by a formula that extends our definition of path, resp. reachability in the following way:

Definition 16 (Path under property)**.** *A path under property φ between two routers $R, R' \in \mathcal{R}$ on the network $N = \langle \mathcal{R}, \mathcal{L}, \mathcal{F} \rangle$ is a sequence of routers $r_i \in \mathcal{R}$ and links $l_i \in \mathcal{L}$ with filters $F_i \in \mathcal{F}$ as follows:*

$$
\begin{aligned}
\pi_\varphi(R, R') \quad &= \quad (R, \langle R, R_1\rangle, R_1, \langle R_1, R_2\rangle, R_2, \ldots, R_k, \\
&\qquad \langle R_k, R'\rangle, R') \\
\textit{where} \quad &\qquad F_{\langle R,R_1\rangle} \wedge \ldots \wedge F_{\langle R_k,R'\rangle} \wedge \varphi \textit{ holds.}
\end{aligned}
$$

The definition above restricts the set of possible paths from R to R' to those paths where property φ is satisfied on every link of the path.

Definition 17 (Network Reachability under property)**.** *Network Reachability under property φ on the network $N = \langle \mathcal{R}, \mathcal{L}, \mathcal{F} \rangle$ $NetReach_\varphi(R)$ is a set of routers reachable from router R under property φ:*

$$NetReach_\varphi(R) = \{R' \in \mathcal{R} \mid \exists \pi_\varphi(R, R_k), R_k = R'\}$$

3.1. Reachability Analysis

To analyse security property on the network $N = \langle \mathcal{R}, \mathcal{L}, \mathcal{F} \rangle$ we at first (i) define a property of the network by means of a packet filter φ, and then (ii) compute a network reachability set from the starting point R under the

given property φ, i.e., $NetReach_{\varphi}(R)$. If the set is non-empty, there is a path that covers the given property, and the property is valid on that path starting from R of the network N.

Suppose the network $N = \langle R, T, F\rangle$ from Fig.5 and property allowing WWW traffic from PC to WWW: $\varphi(h) = (h.proto = tcp) \wedge (h.srcIp = 10.10.3.2/32) \wedge (h.dstIp = 10.10.1.2/32) \wedge (h.dstPort = 80)$. Then, we compute a set of destination reachable from the source PC under property φ, i.e., $NetReach(PC)_{\varphi} = \{R_3, R_1, R_2, WWW\}$.

There are two paths between PC and WWW, π_1, going over $R3$ and $R1$, resp. π_2, going over $R3, R2$, and $R1$. By adding property φ that represents WWW request from PC to WWW, we get only one path under that property $\pi_{\varphi} = \pi_1$, that goes over links $\langle PC, R3\rangle, \langle R3, R1\rangle$, and $\langle R1, WWW\rangle$. For the path π_1 the formula $F_{\langle PC,R3\rangle} \wedge F_{\langle R3,R1\rangle} \wedge F_{\langle R1,WWW\rangle} \wedge \varphi$ is satisfied because destination address $10.10.1.2$ is in the range of interval $dstIP$ set by filtering rules, and no other restrictive rules are applied. The path π_2 is not valid under φ because $F_{\langle R_3,R_2\rangle} \wedge \varphi$ is not satisfied (DNS filtering on the link).

What happens if the link $\langle R_3, R_1\rangle$ goes down? The routing tables are recomputed. Still, all the networks in N are reachable from PC. However, $NetReach_{\varphi}(PC) = \{R3\}$ since path π_2 does not satisfies φ. Under such link states the property φ is not satisfied.

This transient behaviour of the network cannot be found by testing or simulation. In the example above we consider only one state of the network where all links are up. In the following text we will discuss transient behaviour. We will also show how this behaviour can be expressed using network states s, and analysed.

3.2. Transient Behaviour of the Network

If the state of any link changes, i.e. link goes down or up, it will alter the network topology and routing information may become obsolete. At that time it is necessary to recompute packet filtering functions on interfaces to reflect new routing information. This subsection introduces the notation of a state of the network and identifies the basic properties of the structure of states that aids in formal analysis.

For simplicity, we restrict now our model to converged states only, where the link is up or down, and the routing process successfully distributed converged routes into every router. Later we will show, that this abstraction is correct even for momentary unstable network. This simplification allows us to represent network states as a bit vector.

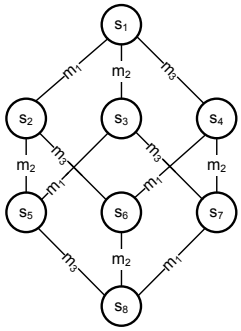
Definition 18 (Network State)**.** *A network state* s *is a vector of boolean values representing states of all links. A link state is a boolean value representing either "link up" or "link down" state.*

For the network with $l = |\mathcal{L}|$ links the state s is represented by an l-bit vector. Number of different states is given by all possible combinations of link states, that is $m = 2^l$. Suppose our running example from Fig.1, for brevity, only links between routers. We have a network with three links, represented by three pairs of connections in the corresponding network graph (see fig.2).

In this network, there are following network states: $s_8 = (0, 0, 0)$ (all links are down, network is disconnected), $s_5 = (0, 0, 1)$ (only one link is up, in particular, link represented by pair $\langle R_2, R_3\rangle, \langle R_3, R_2\rangle$), etc. The number of network states is $2^3 = 8$.

Definition 19 (Network Transition System)**.** *Behaviour of a network $N = \langle \mathcal{R}, \mathcal{L}, \mathcal{F}\rangle$ from point of view of topology changes, can be defined by* network transition system $\mathcal{T}_N = (\mathcal{S}_N, \rightarrow)$*, where*

- $\mathcal{S}_{N_1} = \{s_1, s_2, \ldots, s_m\}, m = 2^l$ *is a finite set of network states,*
- $\rightarrow$ *is a transition relation between network states such that $s_i \rightarrow s_j$ iff $\forall n \in \{1, \ldots, k-1, k+1, \ldots, l\} : s_n^i = s_n^j$ and $s_k^i \neq s_k^j$, where $s_i = (s_1^i, \ldots, s_k^i, \ldots, s_l^i)$ and $s_j = (s_1^j, \ldots, s_k^j, \ldots, s_l^j), s_i, s_j \in \mathcal{S}_{N_1}$.*



$s_1 = \langle 1, 1, 1\rangle$
$s_2 = \langle 0, 1, 1\rangle$
$s_3 = \langle 1, 0, 1\rangle$
$s_4 = \langle 1, 1, 0\rangle$
$s_5 = \langle 0, 0, 1\rangle$
$s_6 = \langle 0, 1, 0\rangle$
$s_7 = \langle 1, 0, 0\rangle$
$s_8 = \langle 0, 0, 0\rangle$

$m_1 = \langle l_{12}, l_{21}\rangle$
$m_2 = \langle l_{23}, l_{32}\rangle$
$m_3 = \langle l_{13}, l_{31}\rangle$

Figure 6. Network transition system

Transition system $\mathcal{T}_{N_1}$ in Fig. 6 precisely describes subset of possible network states of the network N. The focus is only on the state of links among the routers.

In general, verification of properties Ψ on network N requires analysis of the every network state. Formally, the verification of properties Ψ of the system N is $\forall s \in \mathcal{S} : N(s) \models \Psi$. For such analysis, a model checking technique can be applied. As the number of states is exponential to the number of links, the model checking faces the state explosion problem. The expected result of property verification is

a set of states for which the property holds. Above defined transition system can help to reduce a number of states that have to be checked.

A network transition system constructed according the definition above forms a lattice. If the verified property is closed under lattice operations the states we have to visit form a sub-lattice. We plan to examine these properties deeply in our future work.

3.3. Computing Routing Table Content

Computation of routing tables in routers that uses RIP, a dynamic routing protocol, is based on Bellman-Ford algorithm [12]. The implementation of this algorithm on routers is asynchronous, iterative and distributed. Distributivity comes from the behavior of RIP on the routers. Each router R receives information from one or more directly connected routers and calculates new routing table based on this received information. For link cost calculation RIP protocol uses hop count as metric that represents a number of nodes through which the message must go to reach the destination. Even in the case of more metrics, such as delay, bandwith, reliability, that are used by more complex protocols, the main requirement that it must be possible to represent the total metric as the sum of individual one hop metrics has to be satisfied. For our demonstration we use a variant of the distance vector algorithm described in [13].

Definition 20 (Distance Vector Algorithm)**.** *Distance vector algorithm computes a minimum reachable distance $D^i(j)$ from nodes R_i to R_j. It is defined to satisfy the following constraints:*

- $D^i{}_0(i) = 0$*, for all nodes* R_i*,*
- $D^i{}_0(j) = \infty$*, for* $i \neq j$*, and*
- $D^i{}_{n+1}(j) = min_k[d(i,k) + D^k_n(j)]$*, for* $i \neq j$*, all neighbors* R_k *of* R_i*, and* $d(i,k)$ *to be the cost of the direct connection between* R_i *and* R_k*.*

Figure 7 shows the two interative steps needed to compute routing tables for individual routers. In figure 8 the resulting routing tables are presented. Note that the computation is for the case $s_5 = (1, 0, 1)$, i.e. link between routers R_1 and R_3 is down.

An analysis in [14] gives a proof that this algorithm will converge to the correct estimates in finite time. As the assumption the authors consider that entities are reliable, i.e. they will not crash. If there is a problem with the entity it can be modelled as topology change. Also there are no constraints on the communication and it can be considered that entities can send updates asynchronously. The proved convergence under these assumptions make this abstraction suitable for the analysis considered in this paper.

$D_0^{R_1}$	R_2	R_3
(R_1, R_2)	0	∞
(R_1, R_3)	∞	∞

$D_1^{R_1}$	R_2	R_3
(R_1, R_2)	0	1
(R_1, R_3)	∞	∞

$D_0^{R_2}$	R_1	R_3
(R_2, R_1)	1	∞
(R_2, R_3)	∞	0

$D_1^{R_2}$	R_1	R_3
(R_2, R_1)	0	1
(R_2, R_3)	1	0

$D_0^{R_3}$	R_1	R_2
(R_3, R_1)	∞	∞
(R_3, R_2)	∞	0

$D_1^{R_3}$	R_1	R_2
(R_3, R_1)	∞	∞
(R_3, R_2)	1	0

Figure 7. Distance Vector Routing Database for s=(101).

Values on vertical axis represent interfaces, values on horizontal axis represent destinations, i.e. arguments for the distance vector function.

rt^{R_1}	next	cost
R_2	R_2	0
R_3	R_2	1

rt^{R_2}	next	cost
R_1	R_1	0
R_3	R_3	0

rt^{R_3}	next	cost
R_1	R_2	1
R_2	R_2	0

Figure 8. Resulting Routing Tables for s=(101)

4. Conclusions

This paper introduces a new methodology how to analyse dynamic behaviour of the network. The approach is based on graph theory. The paper shows how to create a network model extracted from routers' configurations and suggests how to automatically analyse it. In the model we consider access control lists and dynamic routing protocols. We show how ACLs and routing can be added to the graph model using quantifier-free first-order formulas. We present an algorithm that transforms a set of ACL rules to the formula and prove that the transformation is correct. We also show how routing information can be added to the model. We recommend interval decision diagrams for internal representation of such formulas. Using IDDs we can easily provide conjunction, disjunction, or inclusion of routing information and ACLs specified with formulas.

In the analysis part we show how network reachability can be computed. We consider not only static topology of the network but also link failures. Our model is general and can be used for verification of reachability properties even if some links go down.

4.1. Related work

The tools for testing and simulation of network behavior is well established. Instead the contributions on verification of network behavior by means of formal methods are rare. The closest work to ours is by Xie et.el. [2]. They provide a unified framework for reasoning about the effects of packet filters, routing policy, and packet transformations on the network's reachability. In that work system states are not explicitly evaluated nor examined by automatic methods, but lower and upper bounds on the reachability are defined by means of a set of packets allowed to pass through the network between given nodes.

The work on the packet classification is thoroughly examined for many years and many works were published, e.g. [15], [16],[17],or [18]. The use of IDD has been proposed as the efficient implementation of packet filter functions for resource constrained devices. In our work we consider to use it in a different context of model checking tool.

4.2. Future Work

Our future work is oriented mainly toward research of analysis techniques for the given network model. One issue has been already mentioned in the paper. We want to study the class of properties whose satisfiability is closed under lattice operations of the network transition system. The aim is to set criteria that would guarantee that the property belongs to this class. From the practical viewpoint, we want to experiment with model checking tools to show the feasibility of the proposed analysis technique. In the paper we considered a general distance vector protocol. For better approximation of the network behavior the models of different routing protocols, such as RIP, OSPF, EIGRP, and BGP, will be created and used in the verification procedures.

The other direction is to incorporate the routing policies different than those based on static routing information or routing information provided by dynamic routing protocol. It includes policy based routing, for instance. Finally, we plan to employ probabilistic verification techniques for networks with dynamic metrics, such as congestion or link reliability, appearing in routing information.

Acknowledgement

The work has been supported by the CEZ MMT project no. MSM0021630528 *Security-Oriented Research in Information Technology*. We also appreciate valuable suggestions and notes given us by our colleague Tomáš Vojnar during his thorough reading of the text.

References

[1] F. Nielson, H. Riis Nielson, and C. L. Hankin, *Principles of Program Analysis*, Springer, 1999.

[2] Geoffrey G. Xie, Jibin Zhan, David A. Maltz, Hui Zhang, Albert Greenberg, Gisli Hjalmtysson, and Jennifer Rexford, "On static reachability analysis of ip networks," in *Proceedings of the 24th Annual Joint Conference of the IEEE Computer and Communications Societies*, 2004.

[3] K. Strehl and L. Thiele, "Symbolic model checking using interval diagram techniques," Technical Report 40, Computer Engineering and Networks Lab (TIK), ETH Zurich, Feb. 1998.

[4] David Antoš, *Hardware-constrained Packet Classification*, Ph.D. thesis, Masaryk University, 2006.

[5] Mikkel Christiansen and Emmanuel Fleury, "An interval decision diagram based firewall," in *3rd International Conference on Networking (ICN'04)*. Feb. 2004, IEEE.

[6] Jon Postel, "*An Assigned numbers*," RFC 790, September 1981, Since 1994 at on-line IANA database at http://www.iana.org/.

[7] Cisco, *Configuring IP Access Lists*, white papers 23602 edition, July 2007.

[8] R.E. Bryant, "Graph-based algorithms for boolean function manipulation," *IEEE Transactions on Computers*, vol. C-35-8, pp. 677–691, 1986.

[9] D.L. Dill, "Timing assumptions and verification of finite-state concurrent systems," in *Proceedings of the 1st CAV*, J. Sifakis, Ed. 1989, vol. 407 of *LNCS*, pp. 197–212, Springer Verlag.

[10] G. Delzanno, J.-F. Raskin, and L. Van Begin, "Covering sharing-trees: a compact data-structure for parametrized verification," *Software Tools for Technology Transfer*, vol. 5, no. 2-3, pp. 268–297, 2004.

[11] J. Moeller, J. Lichtenberg, H.R. Anderson, and H. Hulgaa rd, "Difference decision diagrams," Technical Report IT-TR-1999-023, Technical University of Denmark, 1999.

[12] Richard Bellman, "On a routing problem," *Quarterly of Applied Mathematics*, vol. 16, no. 1, pp. 87–90, 1958.

[13] C. Hedrick, "Routing information protocol," Rfc 1058, Rutgers University, 1988.

[14] G. Malkin and F. Baker, "Rip version 2 mib extension," Rfc 1389, Bay Networks, 1993.

[15] Mikkel Christiansen and Emmanuel Fleury, "An mtidd based firewall using decision diagrams for packet filtering," *Telecommunication Systems*, vol. 27, no. 2-4, pp. 297–319, Oct. 2004, Kluwer Academic Publishers.

[16] T. V. Lakshman and Dimitrios Stiliadis, "High-speed policy-based packet forwarding using efficient multi-dimensional range matching," in *SIGCOMM*, 1998, pp. 203–214.

[17] Pankaj Gupta and Nick McKeown, "Packet classification on multiple fields," in *SIGCOMM*, 1999, pp. 147–160.

[18] A.Attar and S.Hazelhurst, "Fast packet filtering using n-ary decision diagrams," 2002, Technical Report, School of Computer Sciene, University of the Witwatersrand.