Telecommun Syst
DOI 10.1007/s11235-011-9585-2

On formal reachability analysis in networks with dynamic behavior

Gayan de Silva · Ondřej Ryšavý · Petr Matoušek · Miroslav Švéda

Abstract Recently, several researches have suggested an application of formal methods for identifying configuration errors, unveiling design problems and predicting network behavior. In this paper, we contribute to this research area by defining a method able to efficiently check reachability properties in dynamically routed networks. We define a notion of network state that captures different network conditions. Each network state represents a unique combination of link availability. The naive enumeration of network states leads quickly to intractability even for small networks as the number of possible combinations grows exponentially. Instead, we enumerate all available paths and, for each path, we search for state aggregation, in which this path is active.

Keywords Formal modeling and analysis · Network service reachability · Dynamic routing · Configuration validation

1 Introduction

Modern computer networks are large, complex and expected to provide a numerous kind of demanded services. Designing, implementing and maintaining such a network is difficult task. Rarely occurred situations, which happened as the effect of a certain combination of device and/or link failures can be hard to troubleshoot. As the networks are often required to deliver a high-level of availability, early detection or prevention of unwanted situations is desirable. Online techniques, such as monitoring, logging and triggered notifications help administrators be aware of occurrences of disaster events. To mimics consequences of service outages network designs usually incorporate various failure overcoming techniques, which are implemented, e.g., by duplicating critical network paths or devices. Redundant links or active network devices if they are properly configured can provide an increased level of reliability.

In this paper, we focus on the effect of link (crash) failures to network reliability. Even an isolated link failure in a small network segment can theoretically be propagated to other network segments by dynamic routing protocols. Thus, it is sometimes difficult to predict an overall contribution of any single failure. Harder than that is the impact analysis of multiple failures. It is practically impossible to check all network conditions simulating every link failure in a real network. Instead, a method based on network analysis of topology and configurations can be employed to compute all possible network behavior.

Firewalls are deployed in networks to prevent unwanted communications. Data delivery and thus service availability depends on the behavior of a network. Essentially, this is determined by two factors, namely, routing and filtering. While routing implemented by dynamic routing protocols attempts to find any possible path to deliver the data to its target, the role of filtering is to drop packets that do not match the network security policy. In this sense, certain configurations of routing and filtering may lead to an unexpected network behavior. It is possible that there are some network states that violate either requirements on service availability or a security policy. Finding these incorrect states is difficult using conventional tools, such as ping or traceroute. Static analysis of configuration files seems to be a more viable option. By

G. de Silva · O. Ryšavý (✉) · P. Matoušek · M. Švéda
Faculty of Information Technology, Brno University of Technology, Brno, Czech Republic
e-mail: rysavy@fit.vutbr.cz

M. Švéda
e-mail: sveda@fit.vutbr.cz

Published online: 24 August 2011

creating a model for network behavior by analysis of contents of configuration files it is possible to validate configurations against the defined set of requirements.

1.1 Related work

Research in the area of network security and vulnerability detection has been conducted since the beginning of the Internet. Recently, formal approach has found its role in this area aiming primarily at the verification of security properties and network designs.

The detection of hosts vulnerability and their protection against network attacks is elaborated by Tidwell et al. [24], Zakeri et al. [22, 28]. Ou et al. in [26] introduce an automatic method for inferring required network security and implement it using Prolog language. The authors define reasoning rules that express semantics of different kinds of exploits. The rules are automatically extracted from the OVAL scanner and the CVE database [17].

Bartal et al. in [3] introduce the tool for automatic generation of network protection in form of firewall rules. First, the security policy is modeled using Model Definition Language. Then, a model of network topology is translated into firewall-specific configurations. These configuration files can be immediately loaded into real devices (firewalls).

Ritchey and Ammann in [20] explain how model checking can be used to analyze network vulnerabilities. They build a network security model of hosts, connections, and attackers. Exploits and security properties are described by temporal logics and verified using the SMV model checker. They are able to verify if network hosts are vulnerable to attacks.

In 1997, Guttman defined a formal method to compute a set of filters for individual devices given a global security policy [10]. To achieve a feasibility, the network is abstracted such that only network areas and border routers occur in the model. This decision reflects a real situation as internal routers usually do not participate in data filtering. Similarly, data flow model is defined in terms of abstract packets, which are described by abstract source and destination addresses and service type. An algorithm computes a feasibility set of packets that can pass all filtering rules along the path. The rectangle abstraction of packet representation makes the procedure practical and efficient.

Yan et al. have developed a tool called FIREMAN [27], which allows them to detect misconfigurations in firewall settings. The tool performs symbolic model checking of firewall configurations for all possible IP packets and along all possible data paths. The underlaying implementation depends on a Binary Decision Diagram (BDD) library, which efficiently models firewall rules. This tool can reveal intra-firewall inconsistencies as well as misconfigurations that lead to errors at inter-firewall level. The tool can analyze Access Control List (ACL) series on all paths for end-to-end connection thus offering network-wide firewall security verification.

Jeffrey and Samak in [13] aims at analysis of firewall configurations using bounded model-checking approach. They focus on reachability and cyclicity properties. To check reachability, it means to find for each rule r a packet p that causes r to fire. To detect cyclicity of a firewall configuration, it means to find a packet p which is not matched by any rule of the firewall ruleset. They implemented an analysis algorithm by translating the problem to SAT instance and showed that this approach is efficient and comparable to tools based on BDD representation. Similar result was achieved by Pozo, Ceballos and Gasca [19] who provided a consistency checking algorithm that can reveal four consistency problems, namely, shadowing, generalization, correlation and independence.

Liu et al. developed a method for formal verification and testing of (distributed) firewall rules (see [14] and [9]) against user provided properties. They represent firewall rules in a structure called firewall decision diagram (FDD), which is processed by a verification algorithm. A property rule, which describes a property that is checked, e.g. description of a set of packets that should pass the firewall is verified by a single traversing an FDD from the root to a leave.

Xie et al. reports in [25] on their extensive work on static analysis of IP networks. They define a framework able to determine lower and upper approximations on a network reachability considering filtering rules, dynamic routing and packet transformations. The method computes a set of packets that can be carried by each link. By combination of these sets along all possible paths between two end-points, it is possible to determine the end-to-end reachability. The upper approximation determines the set of packets that could potentially be delivered by the network, while the lower approximation determines the set of packets that could be delivered under all possible forwarding states. In their paper, the authors also present a refinement of both upper and lower approximations by considering the effect of dynamic routing.

Bera, Dasgupta and Ghosh (see [5] and [4]) define a verification framework for filtering rules that allows one to check the correctness of distributed ACL implementations against the given global security policy and also to check reliability (or fault tolerance) of services in a network. To check the correctness, the filtering rules are translated to assertions in the form of first order logical formulas. They are together with logical description of global security policy sent to SAT solver that mechanically checks the satisfiability. In the case of an inconsistency, the SAT solver produces a counter example that helps debugging ACL rules. To check the reliability, the framework accepts a description of a global security policy, a collection of ACL rules

and a network description and computes whether the rules are consistent with the given policy. Policy is understood as a description of service availability with respect to defined network zones. The method used computes first a network access model, which is a directed graph with ACLs assigned to its edges. Next, the service flow graphs (SFG) are generated for all services in the interest, e.g. SFG for ssh traffic. An SFG is a subgraph of the network access graph. The fault analysis is performed by computing minimum cuts in all SFGs. These values then represent how many link failures can be tolerated.

Gan and Helvik in [8] employee probabilistic methods to reduce the space of possible network states. They use stochastic activity networks to describe the failures and repairs of network components and other dynamic issues of the network. This restricts the state space to the subset of the most probable situations.

Another algorithmic framework based on probabilistic calculations is discussed by Menth et al. in [16]. They present a framework for the analysis of ingress-egress unavailability and link congestion. The framework is able to deal with link and device failure, changes of user behavior, and rerouting.

1.2 Contribution

This paper introduces a method dealing with the problem of exponential grow of states that have to be enumerated when exhaustively checking end-to-end reachability. Instead of checking every single state, the method collects all available paths first, and then for each path the set of associated states is determined. Because of using a compact state representation, this set is represented by a finite and usually small number of representative elements.

For network description, we use a unified model defined in [25], but employ a different approach during the analysis. Unlike [15], the analysis does not precompute all routing tables in order to verify network reachability. The probabilistic data are not used in our analysis as suggested in [8] and [16].

The present paper is the extension of work published in [23]. The main difference to that paper is a simplification of the presented method and the definition of a reachability analysis method.

1.3 Organization of the paper

In Sect. 2, we define preliminary information needed for the development presented in the paper. In Sect. 3, we introduce the notion of the Modified Topology Table (MTT), which is the key data and functional structure for compact state space representation and source of information of analysis method. In Sect. 4, we explain the reachability analysis process which can be automatized in a software tool. In Sect. 5, we draw the conclusions and discuss the future work.

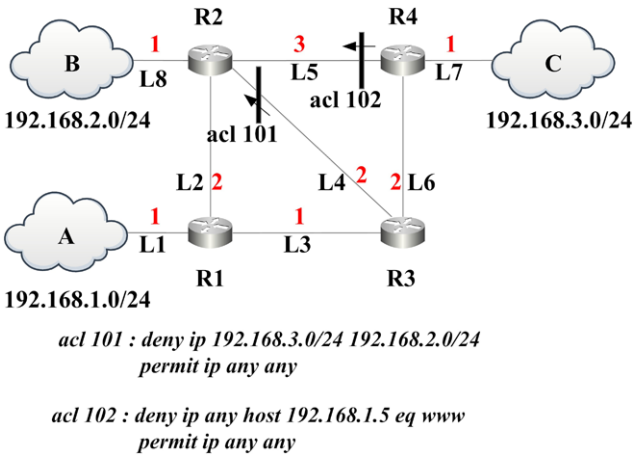

Fig. 1 Dynamic network—devices, links and ACLs

2 Preliminaries

In this section, we introduce basic definitions used in further development. We will use a simple network topology shown in Fig. 1 as a running example.

2.1 Network model

A network topology is modeled as a graph $N = \langle R, L \rangle$, where R is a set of network devices, and $L \subseteq R \times R$ is a set of communication channels between adjacent devices. For every physical link between two adjacent devices R_i and R_j there is a pair of channels $l_{ij} = \langle R_i, R_j \rangle$ and $l_{j,i} = \langle R_j, R_i \rangle$, such that $l_{i,j} \in L$ and $l_{j,i} \in L$.

Modeling a single physical link as a pair of communication channels simplifies assignment of filters to links and computation of path costs. Further, the employed algorithm enumerating available paths expects directed graphs being its input.

Links have associated costs whose metrics depend on the configured routing protocol, e.g., RIP uses hop-count while OSPF uses values proportional to available link bandwidth. We define a *cost function* $C_T(l)$ for every routing protocol T, e.g., $C_{RIP}(l) = 1$ for every $l \in L$.

A proposed analysis method performs packet-based reachability checking. Consider P be a set of packet descriptions, each communication channel can be assigned a *filter* $f_{i,j} = \{p \in P : p$ is permitted by access control lists $A^i_{i,j}$, $A^j_{i,j}\}$. A filter defines a largest set of packets that are permitted by access control lists $A^i_{i,j}$ and $A^j_{i,j}$. ACL $A^i_{i,j}$ is configured at router R_i and it is active for the outgoing traffic on the interface connecting router R_i with neighboring router R_j. ACL $A^j_{i,j}$ is configured at router R_j and it is active for the incoming traffic on the interface connecting router R_j with neighboring router R_i.

Methods dealing with efficient representation of access control lists can be found, for instance, in [15] or [11], and the brief overview is given in Sect. 4.

2.2 Computing available paths

Different routing protocols use different algorithms for selecting the shortest path. Routing protocols cannot establish virtual paths without a physical connection. Therefore, as the first step, we enumerate all available physical paths in the network. Then according to the routing protocol configured, the specified path selection criteria are used to identify the best paths for communication. We tested the approach using Rubin's algorithm [21] for enumerating all simple paths[1] in a graph. The efficient implementation encodes vertices and edges occurring in a path using bit vectors. The pseudocode of the algorithm appears as Algorithm 1. This algorithm requires N^3 matrix operations.

A path π is a sequence of links and devices along the available physical connection between a source and a destination. Let R_0 be the source, and R_n be the destination of path π, then the k-th existing path between R_0 and R_n is defined as follows:

$$\pi^k_{\langle R_0,R_n\rangle} = R_0 l_1 R_1 \dots R_{i-1} l_i R_i \dots R_{n-1} l_n R_n,$$

such that $\forall i, l_i \in L, R_i \in R$ and $\langle R_{i-1}, R_i\rangle = l_i$. A path may be compactly represented by a path descriptor $d(p^k) = (v, e)$, where v is a vertex vector and e is an edge vector. Vertex vector v for a set of vertices R has 1 in position i for all i such that $r_i \in R$ and 0 in all other positions. Similarly, edge vector e for a set of vertices L has 1 in position i for all i such that $l_i \in L$ and 0 in all other positions.

A cost of path π represented as a path descriptor $d(\pi) = (v, e)$ can be computed by counting weight of the edge cost vector. An edge cost vector is obtained by replacing 1 in the edge vector by the corresponding link costs.

A filter of path π is $f_\pi = f_{l_1} \cap f_{l_2} \cap \dots \cap f_{l_n}$, where $l_1, l_2, \dots, l_n \in \pi$. Intersection naturally describes the fact that the packet can be carried by the path only if it is permitted by all access control lists along the path.

2.3 Cost function for RIP, OSPF and EIGRP

The proposed approach for modeling and analysis does not depend on the type of dynamic routing protocol in use. Only the computation of the cost function differs for each protocol as shown below. In our example, we have considered the OSPF as the configured protocol.

1. *RIP* [7]—RIP has default link cost of value one and hence for any link l, $C(l) = 1$. Therefore the shortest path has the lowest hop count.

[1]The path is called simple if all of the vertex of the path are distinct.

Algorithm 1 Rubin's algorithm enumerates all simple paths in a graph

Require: List of edges $(i, j) \in E$ for graph $G = (V, E)$.
Ensure: Matrix D whose cells $D(i, j)$ contain all paths from i to j.

```
for every (i, j) ∈ E do
   D(i, j) = [d(i, j)]
end for

for j = 1 ... N do
  for i = 1 ... N do
    for k = 1 ... N do
      for every (v, e) ∈ D(i, j) and (w, f) ∈ D(j, k)
      do
        if v · w · I_j = 0 then
          append (v + w, e + f) to D(i, k)
        end if
      end for
    end for
  end for
end for
```

The following notation is used in the algorithm:

- $d(i, j) = (v, e)$ is a path descriptor consisting of vertex vector v, which has set only bits i and j, and edge vector, which has set only bit corresponding to index of edge (i, j) in E.
- Operations $x \cdot y$ and $x + y$ are Boolean vector AND and OR, respectively.
- I_j is a vector of size N which has 0 in position j and 1 in all other positions.

2. *OSFP* [18]—OSPF requires the bandwidth function (BW) to compute the cost function, $BW(l_i) : L \to \mathcal{N}$, where $BW(l)$ is the bandwidth of the link l. The cost function for a link is defined by Cisco [1] as $C(l) = [10^8/BW(l)]$. The cost of the path $C(\pi)$ has the accumulated link costs along the path and least cost path will be used for the communication.
3. *EIGRP* [12]—EIGRP uses delay and bandwidth of the whole path to calculate the cost function of the path. Delay function D is $D(l_i) : L \to \mathcal{N}$. The delay function for the path is defined as $D(\pi) = D(l_0) + D(l_1) + \dots + D(l_n)$, where $l_0, l_1, \dots, l_n \in \pi$. Bandwidth function BW is $BW(l_i) : L \to \mathcal{N}$. The bandwidth function for the path is defined as $BW(\pi) = BW(l_i)$ where $\forall l_i, l_j \in \pi, BW(l_i) \le BW(l_j)$. Then the cost function for the path π is given by $C(\pi) = [10^8/BW(\pi) + D(\pi)]$.

3 Modified topology table

Individual routing information tables maintained by forwarding devices are searched during packet forwarding for the best available information. Content of these tables reflects the actual network condition, which is determined from availability of links and devices. In this work we assume only availability of links, which can be expressed using two values, e.g., down and up.

To get the global view on the network, it is possible to approximate contents of individual routing tables by computing all available paths and select paths that would be most probably used in a given network states for data delivery.

To check the end-point reachability in a certain network state one needs to compute the best path from a source network to a target network in this state and verify that a set of desired packets is included in a set of permitted packets for this path.

Attempting to find all network states, in which the reachability is guaranteed using a naive approach would mean to enumerate and verify the reachability in all network states. This soon leads to a problem known as state explosion, as there are $2^{|L|}$ possible states considering only two-valued link condition representation.

Therefore as a solution to this problem, we design a structure called the modified topology table (MTT) which is calculated for given network and allows to quickly determine the best path for any network state. Using the MTT, it is easy to derive an actual connectivity for any network state, a routing table for any network state, available paths for any network state, available paths from any source to any destination, filters applied for any path, costs of paths, and critical links.

Features of the Modified Topology Table are summarized as follows:

- Topology for any network state
- Routing table for any device
- Set of all available paths for any network state
- Set of all path between two end points
- Set of similar network states
- Set of filter applied in any network state
- Series of filters applied to any path
- Overall costs of any path
- Set of critical links

The idea behind MTT is not to enumerate network states, but instead to find all paths and assign sets of corresponding network states to these paths. The approach is summarized as follows:

- To enumerate all possible simple paths for a network state with all links in state *up*. This is the maximal set of simple paths possible in the network at any state. Computation can be performed using Rubin's algorithm as discussed in the previous section.
- Depending on the routing protocol employed, the corresponding cost function is used to sort paths in the ascending order.
- For each path, its covering lattice is found. This lattice contains all network states, in which the path is selected by network routers.

3.1 Computing network states

In this section we define a process of computing network states associated to precomputed paths. Network states are either aggregated or atomic.

An *aggregated network state* is a three-valued vector $S = \langle c_1, c_2, \ldots, c_i, \ldots, c_n \rangle$, of length n, where $c_i \in \{0, 1, \times\}$ represents the link state of $l_i \in L$. Value $\times$ expresses that a link state is invariant in representation of the network state. An aggregated network state s_x defines a lattice with bottom element $s_x[0/\times]$ and top element $s_x[1/\times]$, where $s[a/b]$ stands for substitution of a for all b in vector s.

An *atomic network state* is a network state represented as a vector that contains only 0,1 but no $\times$ bits. Given an atomic state s it is possible to check whether this state belongs to an aggregated state q, written $s \in q$.

An example of aggregated state is $\langle 1, \times, 1, \times, 0 \rangle$. A set of atomic states that belongs to this state is:

$$\{\langle 1, 1, 1, 1, 0 \rangle, \langle 1, 0, 1, 1, 0 \rangle, \langle 1, 1, 1, 0, 0 \rangle, \langle 1, 0, 1, 0, 0 \rangle\}.$$

Consider $\Pi^N_{i,j}$ be a set of all paths found in network $N = \langle R, L \rangle$ from source r_i to destination r_j, where $r_i, r_j \in R$. We define $\Pi^N_{i,j}|s$ to be a subset of paths such that these paths are available in a state s. Path $\pi = \langle v, e \rangle$ is available in state s if $s_x[0/\times] \cdot e = e$. It means that the path has to be available in state s if all invariant links of state s are down. Further, we define an ordering

$$(\Pi^N_{i,j}|s, \leq)$$

that sorts available paths based on their costs. Paths with the minimal cost are called *active paths* at state s.

States associated with found paths are enumerated by performing the following steps:

1. Take the best path $\pi_0 = (v_0, e_0)$ and compute its aggregated state, $e_0[\times/0]$. This yields to a set of only a single item, $s_0 = \{e_0[\times/0]\}$.
2. Take the second best path $\pi_1 = (v_1, e_2)$. This path is active in states $e_1[\times/0]$ except states s_0, which is $s_1 = e_1[\times/0] \ominus s_0$.
3. Take the third best path $\pi_2 = (v_2, e_2)$. This path would be active in states $e_2[\times/0]$ except the states $e_0[\times/0]$ and $e_1[\times/0]$, which is $s_2 = e_2[\times/0] \ominus (s_0 \cup s_1)$.
4. Continue until the last path was processed.

x	y	x $\prec_b$ y
×	×	×
×	0	0
×	1	⋆
1	×	1
1	1	1

x	y	x $\bowtie_b$ y
0	1	1
1	0	1
×	y	×
x	×	×
0	0	0
1	1	0

Operation $\prec_b$ can be extended for vectors of length n, such that $x = s \prec q$ iff $x[i] = s[i] \prec_b q[i], 1 \leq i \leq n$. Operation $\bowtie_b$ can be extended for vectors of length n, such that $x = s \bowtie q$ iff $x[i] = s[i] \bowtie_b q[i], 1 \leq i \leq n$.

Fig. 2 Definition of operations $\prec$ and $\bowtie$

Operation $s \ominus R$ is defined as follows:

$$s \ominus R = \{x : \forall q, r \in R : \mathit{diff}(s, q, x) \wedge \mathit{disj}(x, q)\},$$

where difference relation $\mathit{diff}(s, q, x)$ is defined based on the bit-wise operation $\prec$ that finds all possible positions of difference between individual bits of vectors. The $\prec$ operator together with $\bowtie$ operator, which is used in *disj* relation, are defined in Fig. 2.

Let $x = s \prec q$, then the difference relation $\mathit{diff}(s, q, x_j)$ relates all s, q and x_j such that the following holds:

- x_j is get from x by replacing a single occurrence of $\star$ by 0 and all other by $\times$, and
- vector x has to contain at least one $\star$.

The $\mathit{disj}(s, r)$ is satisfied if $s \bowtie r$ contains at least a single 1. Vectors $\langle 1, 1, \times, 0, \times\rangle, \langle 1, 0, 1, 0, 1\rangle$ are disjunctive, but $\langle 1, 1, \times, \times, \times\rangle, \langle 1, 1, 0, 1, \times\rangle$ are not.

The operator $\prec$ is undefined for $1 \prec 0$. This reflects the situation when there is an attempt to find differences for state $s = e[\times/0]$ representing path $\pi = (v, e)$ to state q such that π is not available in q.

The algorithmic solution is given in Algorithm 2. In the rest of the section, we provide an illustrative example to better explain this method.

3.2 Example

In Table 1, the whole content of the MTT for example network from Fig. 1 is shown. We show the computation of several items in the MTT. As required by Algorithm 2, the input to the method is an ordered list of available paths. We will demonstrate the calculation for networks B and C, only.

- The initialization step takes the first path, which is $\pi_0 = [l_1, l_3, l_6, l_7]$. Corresponding edge vector is $e_0 = \langle 1, 0, 1, 0, 0, 1, 1, 0\rangle$. The aggregated state for this path is $s_0 = \{\langle 1, \times, 1, \times, \times, 1, 1, \times\rangle\}$.
- In the second step, we choose path $\pi_1 = [l_1, l_2, l_5, l_7]$. The edge vector is $e_1 = \langle 1, 1, 0, 0, 1, 0, 1, 0\rangle$. Then $s_1 = \langle 1, 1, \times, \times, 1, \times, 1, \times\rangle \ominus s_0$. This amounts to find difference states x such that

$$\mathit{diff}(\langle 1, 1, \times, \times, 1, \times, 1, \times\rangle, \langle 1, \times, 1, \times, \times, 1, 1, \times\rangle, x).$$

Term $\langle 1, 1, \times, \times, 1, \times, 1, \times\rangle \prec \langle 1, \times, 1, \times, \times, 1, 1, \times\rangle$ is defined and gives $\langle 1, 1, \star, \times, 1, \star, 1, \times\rangle$. Thus, the states in which path π_1 is active are

$$s_1 = \{\langle 1, 1, 0, \times, 1, \times, 1, \times\rangle, \langle 1, 1, \times, \times, 1, 0, 1, \times\rangle\}.$$

- There are two remaining paths with the same cost. These should be computed in the same context, as they can be used both, in case of equal cost load balancing, or one of them is selected arbitrary. Therefore the third step is to evaluate these paths. Path $\pi_2 = [l_1, l_2, l_4, l_6, l_7]$ has an edge vector $e_2 = \langle 1, 1, 0, 1, 0, 1, 1, 0\rangle$.

$$s_2 = \langle 1, 1, \times, 1, \times, 1, 1, \times\rangle \ominus (s_0 \cup s_1).$$

The found difference states are:

$x_1 = \langle 1, 1, 0, 1, \times, 1, 1, \times\rangle$ with s_0
$x_2 = \langle 1, 1, 0, 1, 0, 1, 1, \times\rangle$ with s_1

Only x_2 is considered as a state for path π_2 as x_1 is not disjunctive with states in s_1.

The last path, $\pi_3 = [l_1, l_3, l_4, l_5, l_7]$ is evaluated in a similar manner. Edge vector $e_3 = \langle 1, 0, 1, 1, 1, 0, 1, 0\rangle$ is used in computation of state s_2:

$$s_3 = \langle 1, \times, 1, 1, 1, \times, 1, \times\rangle \ominus (s_0 \cup s_1).$$

Algorithm 2 Computation of MTT for a single pair of endpoints.

Require: An ordered list of paths $L = (\Pi_{i,j}^N | s, \leq)$.
Ensure: A table consisting of n rows, (s_i, c_i, p_i, f_i).

$(v_0, e_0) := \texttt{head}(L)$
$L := \texttt{tail}(L)$
$s_0 := \{e_0[\times/0]\}$
$p_0 := (v_0, e_0)$
$c_0 := \texttt{cost}(e_0)$
$f_0 := \texttt{filters}(e_0)$

for $i = 1 \ldots (n-1)$ **do**
  $(v_i, e_i) := \texttt{head}(L)$
  $L := \texttt{tail}(L)$
  $c_i := \texttt{cost}(e_i)$
  $p_i := (v_i, e_i)$
  $f_i := \texttt{filters}(e_i)$
  $s_i := e_i[\times/0] \ominus \bigcup\{s_j : 0 \leq j < i \text{ and } c_j < c_i\}$
end for

Table 1 Modified topology table

	Filter	ACL expression
	f_1	Permit ip any any
	f_2	Deny ip 192.168.3.0/24 192.168.2.0/24, permit ip any any
	f_3	Deny ip any host 192.168.1.5 www, permit ip any any

No	Source	Destination	General Sate	Cost	Path	Filter
1	192.168.1.0/24	192.168.2.0/24	1,1,X,X,X,X,X,1	4	$l_1l_2l_8$	f_1
2	192.168.1.0/24	192.168.2.0/24	1,0,1,1,X,X,X,1	5	$l_1l_3l_4l_8$	f_2
3	192.168.1.0/24	192.168.2.0/24	1,0,1,0,1,1,X,1	8	$l_1l_3l_6l_5l_8$	f_3
4	192.168.1.0/24	192.168.3.0/24	1,X,1,X,X,1,1,X	5	$l_1l_3l_6l_7$	f_1
5	192.168.1.0/24	192.168.3.0/24	1,1,0,X,1,X,1,X + 1,1,X,X,1,0,1,X	7	$l_1l_2l_5l_7$	f_3
6	192.168.1.0/24	192.168.3.0/24	1,1,0,1,0,1,1,X	8	$l_1l_2l_4l_6l_7$	f_1
7	192.168.1.0/24	192.168.3.0/24	1,0,1,1,1,0,1,X	8	$l_1l_3l_4l_5l_7$	$f_3 \vee f_2$
8	192.168.2.0/24	192.168.1.0/24	1,1,X,X,X,X,X,1	4	$l_8l_2l_1$	f_1
9	192.168.2.0/24	192.168.1.0/24	1,0,1,1,X,X,X,1	5	$l_8l_4l_3l_1$	f_2
10	192.168.2.0/24	192.168.1.0/24	1,0,1,0,1,1,X,1	8	$l_8l_5l_6l_3l_1$	f_3
11	192.168.2.0/24	192.168.3.0/24	X,X,X,X,1,X,1,1	5	$l_8l_5l_7$	f_3
12	192.168.2.0/24	192.168.3.0/24	X,X,X,1,0,1,1,1	6	$l_8l_4l_6l_7$	f_2
13	192.168.2.0/24	192.168.3.0/24	X,1,1,0,0,1,1,1	7	$l_8l_2l_3l_6l_7$	f_1
14	192.168.3.0/24	192.168.1.0/24	1,X,1,X,X,1,1,X	5	$l_7l_6l_3l_1$	f_1
15	192.168.3.0/24	192.168.1.0/24	1,1,0,X,1,X,1,X + 1,1,X,X,1,0,1,X	7	$l_7l_5l_2l_1$	f_3
16	192.168.3.0/24	192.168.1.0/24	1,1,0,1,0,1,1,X	8	$l_7l_6l_4l_1$	f_2
17	192.168.3.0/24	192.168.1.0/24	1,0,1,1,1,0,1,X	8	$l_7l_5l_4l_3l_1$	$f_3 \vee f_2$
18	192.168.3.0/24	192.168.2.0/24	X,X,X,X,1,X,1,1	5	$l_7l_5l_8$	f_3
19	192.168.3.0/24	192.168.2.0/24	X,X,X,1,0,1,1,1	6	$l_7l_6l_4l_8$	f_2
20	192.168.3.0/24	192.168.2.0/24	X,1,1,0,0,1,1,1	7	$l_7l_6l_3l_2l_8$	f_1

This state is computed at the same context as state s_2 because these paths π_2 and π_3 have the same cost. The found difference states are:

$x_3 = \langle 1, \times, 1, 1, 1, 0, 1, \times \rangle$ with s_0
$x_4 = \langle 1, 0, 1, 1, 1, 0, 1, \times \rangle$ with s_1

Again, x_3 cannot be considered as a state for π_3 because x_3 is not disjunctive to states in s_1. Therefore, only x_2 is accepted as a state of π_3, that is, $s_3 = \{x_3\}$.

Results of computation for other endpoints in the network is shown in Table 1.

4 Reachability analysis

In this section, we describe basic principles of reachability and security property analysis that can employ the proposed MTT structure. The MTT captures information on all available paths in the network and states in which these paths are active. Therefore the analysis that verifies packet-reachability in a given set of network states can obtain necessary information by querying the MTT.

We assume that analysis begins with a definition of interesting end-point networks. For these networks, all possible paths can be obtained by selecting rows from the MTT that correspond to any of these networks.

First, we show how the MTT can be used to obtain some simple network metrics that assess the network facilities by their importance for packet reachability.

Then, we define a straightforward method for evaluating packet reachability by computing a set of packets allowed by filters assigned to paths that are active in the given set of states. We give two approximations, namely, the least set of reachable packets and the greatest set of reachable packets.

4.1 Infrastructure metrics

Based on the collection of paths $\Pi_T = \{\pi_1, \ldots, \pi_n\}$ that connects the set of examined networks T, it is possible to split the network intermediate devices and network links into three groups:

1. Group of Critical Facilities (*CF*) is a subset of intermediate devices and communication links which occur at every path. The links and devices of *CF* are essential for

communication. *CF* can be represented using vertex and edge vector, $\pi_i = (v_i, e_i)$, that is

$$CF_T = (v_1 \cdot \cdots \cdot v_n, e_1 \cdot \cdots \cdot e_n)$$

2. Group of Supportive Facilities (*SF*) is a subset of intermediate devices and communication links which occur at least along a single path. The links and devices of *SF* are not critical for communication. CF can be represented using vertex and edge vector:

$$SF_T = (v_1 + \cdots + v_n, e_1 + \cdots + e_n).$$

Here we define operation + to be addition in arithmetical sense, e.g., $\langle 0, 1, 0, 1\rangle + \langle 1, 2, 1, 0\rangle = \langle 1, 3, 1, 0\rangle$.

3. Group of Redundant Facilities (*RF*) is a subset of intermediate devices and communication links that do not occur at any path.

$$RF_T = \neg SF_T$$

Vector *CF* gives us the information on the critical facilities of the network for assuring availability of destination end-points. Vector *SF* tells us quantitative information on how each facility is important to provide availability of destination end-points. From these values, importance ratios can be computed, which is for each device r and link l, $SF_T.v[r]/|\Pi_T|$ and $SF_T.e[l]/|\Pi_T|$, respectively.

4.2 Packet reachability analysis

Based on the MTT, it is possible to find the greatest set of packets that are carried between specified networks:

$$\nu(n_1, n_2) = \bigcup_{\pi \in \Pi_{n_1,n_2}} f_\pi$$

Similarly, it is possible to find the smallest set of packets that are carried between specified networks:

$$\mu(n_1, n_2) = \bigcap_{\pi \in \Pi_{n_1,n_2}} f_\pi$$

These two approximations give lower and upper bounds on transmittable packets. More refined result is obtained if we compute sets of transmittable packets in different network states.

$$\nu(n_1, n_2, s) = \bigcup_{\pi \in \Pi^s_{n_1,n_2}} f_\pi$$

$$\mu(n_1, n_2, s) = \bigcap_{\pi \in \Pi^s_{n_1,n_2}} f_\pi$$

Here, term $(\Pi^s_{n_1,n_2})$ yields to a set of all active paths in (aggregated) state s. There can be indeed more than one active path in aggregated state s, e.g., consider state $\langle 1, \times, \times, \times, \times, \times, \times, 1\rangle$ from running example. Two paths, namely, $[l_1, l_2, l_8]$ and $[l_1, l_3, l_4, l_8]$ are active in this aggregated state. In the previous, $\mu(n_1, n_2$ is the special case when $\mu(n_1, n_2, \langle \times, \ldots, \times\rangle)$.

The role of MTT in the process of computation of $\mu(n_1, n_2, s)$ and $\nu(n_1, n_2, s)$ sets is following:

- The MTT is used to select all paths that connects n_1 and n_2.
- The MTT is queried for selecting a set of paths, $(\Pi^s_{n_1,n_2})$ for the given aggregated state s. This can be done by testing whether s is in relationship with states q_i associated to path π_i:
  - if s contains some state q_i, then path belongs to $\Pi^s_{n_1,n_2}$,
  - if q_i contains state s than path π_i is the only path in set $\Pi^s_{n_1,n_2}$,
  - otherwise π_i does not occur in $\Pi^s_{n_1,n_2}$.

Matching source and destination addresses in the MTT can be improved by using a method described in [15] which employs Interval Decision Diagrams [6].

In the next subsection, we review an existing methods for packet filter representation and develop a simple but efficient method for capturing the semantics of packet filters. The logical representation of filters is suitable for efficient checking of the inclusion of a set of examined packets in transmittable packets, which is used in reachability verification.

4.3 List-based packet filters

A list-based packet filter (firewalls) consists of rules imposing network security policies ordered by the priority, which depends on the rule's position within the list. Although there may be other kinds of packet filters, we assume the most common case, in which evaluation of packet filters is based on the first match principle. This means that an action of the highest priority rule that matches the analyzed packet is executed.

Each rule is a multidimensional structure. Dimensions are sets of network fields, e.g., source and destination addresses, port numbers, protocol type, or an action field, e.g., accept, deny, redirection. A typical rule can be formally defined as a tuple $\langle src, dst, srv, act\rangle$, where *src* and *dst* are set of addresses, *srv* is a set of services, and *act* is an action.

Packet filters can suffer from conflicts and dependencies, which complicate the analysis. The goal of packet filter preprocessing is to remove conflicts, redundancies and dependencies such that we avoid the need to evaluate the rules in the imposed order. If the resulting rule set is completely disjoint then it is possible to use a straightforward transformation into logical representation. First, we discuss a method to obtain the disjoint rule set from an ordinary rule set.

We discuss the method proposed in [19]. The method consists of two steps. The first step isolates the possibly conflicting rules and figures out their dependencies. Two rules are potentially conflicting, if both rules have different actions and one rule is either subsumed by another or there is an nonempty intersection in one or more dimensions. The result of the first step is a conflicting graph.

The second step analyses the conflicting graph to identify a minimal set of rules that generate inconsistencies with another rules. The result is a collection of trees. Each pair of root and leaf in this trees defines a conflict that needs to be removed making these rules independent.

The common approach to make the rule independent (disjoint) is to split conflicting rules into more rules and remove conflicting parts. This phase may be followed by merging process in order to optimize the rule set representation as the previous splitting may increase the rule set size. A similar approach is taken also for redundancy elimination as shown in [2].

4.4 Logical representations of rule sets

Semantics of rule set consisting only of independent rules is invariant to rule ordering. We will use this property to define for each rule set its logical representation. This representation has form of propositional logic formula in disjunctive normal form.

Recall that filtering rule is a tuple with network fields. In the simplest case it consists of selectors, namely, source address set, destination address set, service set, and the action. A logical formula that is a translation of a simple rule $r = \langle s, d, v, a \rangle$ consists of a conjunction of all selectors. A selector is represented by a predicate that extracts required network fields from some packet p. Thus, for rule r the formula is written as follows:

$$src\_adr(p) \in s \wedge dst\_adr(p) \in d \wedge service(p) \in v.$$

A list of possible selector functions in shown in Table 2

We adapt network-mask convention for representation of a set of continuous addresses. For instance, it allows us to consider 147.229.12.0/24 as a set of addresses ranging from 147.229.12.0 to 147.229.12.255. We can use the standard set operations, e.g., $src\_adr(p) \in 147.229.12.0/24$ or

$$dst\_adr(p) \in 147.12.28.0/24 \cup 147.12.30.0/24.$$

This can be further expanded to

$$dst\_adr(p) \in 147.12.28.0/24 \\ \vee dst\_adr(p) \in 147.12.30.0/24,$$

which allows us to use the network-mask format for canonical address set representation.

Table 2 Network Field Selectors

Function	Description
$dst\_adr(p)$	Destination address of a packet p
$src\_adr(p)$	Source address of a packet p
$dst\_port(p)$	Destination port of UDP or TCP datagram carried in packet p
$src\_port(p)$	Source port of UDP or TCP datagram carried in packet p
$service(p)$	Service of a packet p

Often, rule sets implicitly assume the existence of a default rule, which has the lowest priority. It matches all packets that were not hit by the preceding rules. While the previously described method copes with default rules transparently, it may cause to split the default rule into a large number of rules appearing in a disjoint rule set. To overcome this issue we ignore the default rule in the process of conflicting rule elimination and consider it again when we compute a logical representation.

A logical representation can be either *positive*, representing all accepted packets or *negative* representing all denied packets. The most commonly, we want to compute a positive representation of rule sets with default deny all rule. In this case we only select all rules with *allow* action and calculate the logical representation for these rules. It completely eliminates the need to explicitly deal with default rule.

5 Conclusions

5.1 Summary

We presented a method for efficient reachability checking in dynamically routed networks. The method does not require to calculate routing tables for forwarding device in every network state. Instead, the number of iterations is reduced by grouping network states which use the same forwarding paths. This can be estimated by computing infrastructure metrics and explicitly enumerated using a simple algorithm. If a device or a link is a member of RF set with respect to the given reachability property it is possible to completely ignore the effect of a potential failure that these elements can have on satisfying of the property.

We have shown how the MTT is used to identify the communication paths for combinations of different link failures. The MTT contains a description of a complete state space in a compact form and all available communication paths of a network, therefore it enables us to quickly check end-to-end reachability, for instance. Considering filters, it is possible to validate implementations of security policy, as proposed in [4].

5.2 Future work

We designed the MTT and proposed the reachability analysis method. The further work is oriented towards extensions and practical evaluations of the presented approach:

- Routing approximation is considered in the computation of the MTT. We assume that routing protocols are configured to provide full connectivity. We do not handle the case in which routing processes can be customized by imposing routing update filters, for instance. Also we do not elaborate on the issue of route selection nor route redistribution.
- The practical aspect of the presented method was only briefly tested. We would like to find the limits of the method by exercising it on a collection of real examples. This involves to implement an experimental tool suite that allows us to automatize processing of input configurations and producing adequate results.
- In the present paper, we only deal with security properties that can be expressed in terms of end-to-end reachability. The future work should be oriented towards relaxing the imposed limits on expressiveness of the security properties specification language, which allows us to define a broader class of security related properties.

Acknowledgements This project has been carried out with a financial support from the Czech Republic state budget through the CEZ MMT project no. MSM0021630528: Security-Oriented Research in Information Technology, by the Grant Agency of the Czech Republic through the grant no. GACR 102/08/1429: Safety and Security of Networked Embedded System Applications, and by the Brno University of Technology, Faculty of Information Technology through the specific research grant no. FIT-10-S-1: Secured, Reliable and Adaptive Computer Systems. Also, the fourth co-author was supported by the grant no. FR-TI1/037 of Ministry of Industry and Trade: Automatic Attack Processing.

References

1. (2006). *Ospf design guide*. (Tech. rep.) Available at url: http://www.cisco.com/warp/public/104/1.pdf.
2. Acharya, S., Wang, J., Ge, Z., Znati, T., & Greenberg, A. (2006). Simulation study of firewalls to aid improved performance. In *39th annual, simulation symposium, 2006* (p. 8).
3. Bartal, Y., Mayer, A., Nissim, K., & Wool, A. (1999). Firmato: a novel firewall management toolkit. In *IEEE symposium on security and privacy* (pp. 17–31). citeseer.ist.psu.edu/article/bartal99firmato.html.
4. Bera, P., Ghosh, S., & Dasgupta, P. (2009). Formal analysis of security policy implementations in enterprise networks. *International Journal of Computer Networks and Communications*, *1*(2), 56–73.
5. Bera, P., Ghosh, S., & Dasgupta, P. (2009). Formal verification of security policy implementations in enterprise networks. In *ICISS '09: Proceedings of the 5th international conference on information systems security* (pp. 117–131). Berlin: Springer.
6. Christiansen, M., & Fleury, E. (2004). An interval decision diagram based firewall. In *3rd international conference on networking (ICN'04)*. Los Alamitos: IEEE Comput. Soc.
7. Hedrick, C. L. (1988). *Routing information protocol*. RFC 1058.
8. Gan, Q., & Helvik, B. (2006). Dependability modelling and analysis of networks as taking routing and traffic into account. In *2nd conference on next generation internet design and engineering, NGI '06* (pp. 8–32).
9. Gouda, M., Liu, A. X., & Jafry, M. (2008). Verification of distributed firewalls. In *Proceedings of the IEEE global communications conference (GLOBECOM)*, New Orleans, Louisiana.
10. Guttman, J. D. (1997). Filtering postures: local enforcement for global policies. In *Proceedings, 1997 IEEE symposium on security and privacy* (pp. 120–129). Los Alamitos: IEEE Computer Society.
11. Guttman, J. D. (1997). Filtering postures: local enforcement for global policies. In *IEEE symposium on security and privacy* (pp. 120–129).
12. Doyle, J. (2006). CCIE professional development routing TCP/IP, vol. 1. Cisco Systems, Inc.
13. Jeffrey, A., & Samak, T. (2009). Model checking firewall policy configurations. In *IEEE international workshop on policies for distributed systems and networks* (pp. 60–67).
14. Liu, A. X. (2008). Formal verification of firewall policies. In *Proceedings of the 2008 IEEE international conference on communications (ICC)*, Beijing, China.
15. Matoušek, P., Ráb, J., Ryšavý, O., & Švéda, M. (2008). A formal model for network-wide security analysis. In *15th IEEE symposium and workshop on ECBS*.
16. Menth, M., Duelli, M., Martin, R., & Milbrandt, J. (2009). Resilience analysis of packet-switched communication networks. IEEE/ACM Transactions on Networking **17**(6).
17. Mitre: Common vulnerabilities and exposures database. Available from http://cve.mitre.org/; accessed on Feb 2008.
18. Moy, J. (1998). *OSPF Version 2*. RFC 2328.
19. Pozo, S., Ceballos, R., & Gasca, R. (2008). Fast algorithms for consistency-based diagnosis of firewalls rule sets. In *Proceedings of the 3rd international conference on availability, reliability and security (ARES)*.
20. Ritchey, R. W., & Ammann, P. (2000). Using model checking to analyze network vulnerabilities. In *IEEE symposium on security and privacy*, Washington, USA.
21. Rubin, F. (1978). Enumerating all simple paths in a graph. *IEEE Transactions on Circuits and Systems*, *25*(8), 641–642.
22. Shahriari, H. R., Sadoddin, R., Jalili, R., Zakeri, R., & Omidian, A. R. (2005). Network vulnerability analysis through vulnerability take-grant model (VTG). In *LNCS: Vol. 3783. Proceedings of 7th international conference on information and communications security (ICICS2005)* (pp. 256–268). Berlin: Springer. citeseer.ist.psu.edu/749214.html.
23. de Silva, G., Sveda, M., Matousek, P., & Rysavy, O. (2010). Formal analysis approach on networks with dynamic behaviours. In *Proceeding of the 2nd international workshop on reliable networks design and modeling*.
24. Tidwell, T., Larson, R., Fitch, K., & Hale, J. (2001). Modeling Internet attacks. In *Proc. of the IEEE workshop on information assurance and security*, West Point, NY.
25. Xie, G. G., Zhan, J., Maltz, D. A., Zhang, H., Greenberg, A., Hjalmtysson, G., & Rexford, J. (2005). On static reachability analysis of IP networks. In *Proc. IEEE INFOCOM*.
26. Ou, X., Govindavajhala, S., & Appel, A. W. (2005). MulVAL: a logic-based network security analyzer. In *Proc. of the 14th USENIX security symposium*, Baltimore. citeseer.ist.psu.edu/article/bartal99firmato.html.
27. Yuan, L., & Chen, H. (2006). Fireman: a toolkit for firewall modeling and analysis. In *Proceedings of IEEE symposium on security and privacy* (pp. 199–213).
28. Zakeri, R., Shahriari, H., Jalili, R., & Sadoddin, R. (2005). Modeling TCP/IP networks topology for network vulnerability analysis. In *2nd int. symposium of telecommunications* (pp. 653–658). citeseer.ist.psu.edu/749214.html.

Gayan de Silva received his bachelor degree (BSc) in Electronics and Telecommunications Engineering at University of Moratuwa, Sri Lanka in 1999 and his master degree (MSc) in Information Technology at Keele University, UK in 2005. He is currently undertaking a PhD at Brno University of Technology, Czech Republic in the area of network security. Gayan has 10 years industrial experience in networking and he is presently employed at IBM Delivery Centre, Central Europe.

Ondřej Ryšavý earned his PhD from Brno University of Technology in 2005. Currently he is an assistant professor at Brno University of Technology. His research interest includes the field of modeling, design and verification of embedded software. He also collaborates on research of network modeling techniques and analysis methods for validating network designs.

Petr Matoušek completed his PhD at Brno University of Technology in 2005 and since that he is at the position of assistant professor. He spent several months as a researcher at LIAFA institution in France during 2003 and as a system administrator in CERN, Geneva, Switzerland during 1997 and 1998. Before that he also undertook the technical training at Digital Engineering Ltd. in Northern Ireland, UK. His research interests include management, services and transmission technologies of computer networks, network security, IDS systems, modeling and analyzing network security, and application of formal specification and verification techniques.

Miroslav Švéda has been professor at Brno University of Technology since 2002. His research interests include embedded systems, formal specifications, engineering of computer-based systems, and engineering computer networks and communication protocols. Also, he has published several papers in the area of security analysis of networks. Prof. Sveda is the author of numerous publications presented at international conferences and in journals.