Ondřej Ryšavý

Formal Logic Based Framework for Network Configuration Analysis

March 16, 2015

Brno University of Technology

Abstract

Current network configurations are involved because they have to satisfy many different requirements. Network devices are configured using high-level declarative languages that control devices behavior shaping the overall network functionality. Because network parameters depend on the composition of these individual configurations, it is important that they are consistent and meet expected properties. From this perspective, providing correct device configurations for enterprise network is difficult task requiring advanced knowledge of various technologies comprising routing, security, access control, high-availability, quality of service and monitoring. Thus, it is necessary to provide a method that would help in delivering correct and network-wide consistent configurations. This needs were addressed in recent research that lead to proposal of methods for firewall configuration, service security control, policy languages as well as to development of systems for complex configuration analysis and synthesis, e.g., Config Assure, MulVal, FAME, FireCrocodile or Fireman among others.

This thesis aims at providing a formal logic-based framework for specification and validation of network policy, firewall rules, and network configurations. Each of these areas is presented by providing an overview of the current state followed by a novel contribution that consists of formalization of concepts and analysis methods. To achieve this, a fixed-point logic language is employed. Implemented logic programs can automatically analyze consistency of formalized domains as well as to check properties of concrete models of network configurations. It is shown, that the presented approach is expressive enough to be a suitable basis for further evolution as a practical network configuration validation tool.

Contents

Acknowledgements

I am grateful for the opportunity to work with Miroslav Švéda, Petr Matoušek, Jaroslav Ráb and Gayan de Silva. Thanks to them, the topic of this thesis was developed from shaping initial idea through identifying a problem to finding a workable solution. I am also indebted to colleagues from Department of Information Systems, Brno University of Technology for their kindness and support.

This thesis is based on results of collaborative research which were published in following papers:

- MATOUŠEK Petr, RÁB Jaroslav, RYŠAVÝ Ondřej and ŠVÉDA Miroslav. A Formal Model for Network-wide Security Analysis. In: Proceeding of the 15 IEEE International Symposium and Workshop on the Engineering of Computer-based Systems. Belfast: University of Ulster, 2008, pp. 171-181. ISBN 0-7695-3141-5.
- ŠVÉDA Miroslav, RYŠAVÝ Ondřej, MATOUŠEK Petr, RÁB Jaroslav and ČEJKA Rudolf. SECURITY ANALYSIS OF TCP/IP NETWORKS – An Approach to Automatic Analysis of Network Security Properties. In: Proceedings of the International Conference on Data Communication Networking ICETE-DCNET 2010. Athens: Institute for Systems and Technologies of Information, Control and Communication, 2010, pp. 5-11. ISBN 978-989-8425-25-6.
- ŠVÉDA Miroslav, RYŠAVÝ Ondřej, MATOUŠEK Petr and RÁB Jaroslav. An Approach for Automated Network-Wide Security Analysis. In: Proceedings of the Ninth International Conference on Networks ICN 2010. Les Menuires: IEEE Computer Society, 2010, pp. 294-299. ISBN 978-0-7695-3979-9.
- MATOUŠEK Petr, RYŠAVÝ Ondřej, DE Silva Gayan and DANKO Martin. Combination of Simulation and Formal Methods to Analyse Network Survivability. In: Proceedings of the IEEE 3rd International ICST Conference on Simulation Tools and Techniques. Malaga: International Communication Sciences and Technology Association, 2010, p. 6. ISBN 978-963-9799-87-5.
- DE SILVA Gayan, MATOUŠEK Petr, RYŠAVÝ Ondřej and ŠVÉDA Miroslav. Formal Analysis Approach on Networks with Dynamic Behaviours. In: 2010 International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT). Moscow: IEEE Computer Society, 2010, pp. 545-551. ISBN 978-1-4244-7285-7.
- ŠVÉDA Miroslav, RYŠAVÝ Ondřej, DE Silva Gayan, MATOUŠEK Petr and RÁB Jaroslav. Reachability Analysis in Dynamically Routed Networks. In: Proceedings of the IEEE ECBS 2011. Piscataway, NJ: IEEE Computer Society, 2011, pp. 197-205. ISBN 978-0-7695-4379-6.

- ŠVÉDA Miroslav, RYŠAVÝ Ondřej, DE Silva Gayan, MATOUŠEK Petr and RÁB Jaroslav. Static Analysis of Routing and Firewall Policy Configurations. e-Business and Telecommunications. Heidelberg: Springer Science+Business Media B.V., 2012, pp. 39-53. ISBN 978-3-642-25205-1.
- RYŠAVÝ Ondřej, RÁB Jaroslav and ŠVÉDA Miroslav. Improving security in SCADA systems through firewall policy analysis. In: Proceedings of the Federated Conference on Computer Science and Information Systems. Krakow: IEEE Computer Society, 2013, pp. 1435-1440. ISBN 978-1-4673-4471-5.
- DE SILVA Gayan, RYŠAVÝ Ondřej, MATOUŠEK Petr and ŠVÉDA Miroslav. On formal reachability analysis in networks with dynamic behavior. Telecommunication Systems. New York: Springer US, 2013, vol. 52, no. 2, pp. 919-929. ISSN 1018-4864.

This research was supported in part by MŠMT under the IT4Innovations Centre of Excellence (CZ.1.05/1.1.00/02.0070), a grant from GAČR Safety and security of networked embedded system applications, and Security-Oriented Research in Information Technology (MSM0021630528).

Chapter 1
Introduction

Network configuration management is a process of maintaining settings of computer network components [19], [35], [69]. The ultimate goal of network configuration management is to simplify the process of maintenance, repair, expansion and the upgrading network infrastructure and services [80].Network management systems, providing methods for verifying semantic correctness of configurations before these are applied to devices would significantly improve the quality of network management [93].

This thesis proposes a logic-based formal framework for analysis of network properties, such as connectivity, security, performance, reliability and their interaction, based on the analysis of network configurations [73]. The goal is to provide a foundation for practical tools detecting configuration errors to improve availability and security of networks. The network provides communication between hosts. Reachability implemented by a network provides a natural interpretation for many network properties. For instance, service availability depends on the possibility of reaching the service by all clients at operable states.

Protection against unauthorized access can be tested by checking reachability property [79]. Only the authorized clients should be able to reach the service. Static analysis of network reachability can be done at a different level abstraction, which depends on network specification and requirements [8]. In IP networks, network reachability is a main function of the network layer. At this layer three concepts, namely, packet forwarding, packet filtering and packet transform, express the network functionality providing data delivery function.

Of course, this is the only part view as other layers significantly contribute to reachability as well. For instance, reliable data delivery is possible if the network layer can find a path from source to destination. To achieve this, a distributed routing algorithm executes to supply the necessary information to routing databases. The transport layer then implements mechanisms for compensating packet loss. Validation of requirements is difficult because it has to consider different networking concepts. While existing methods do not perform the complete analysis, they can validate various network requirements [84]. The aim of the proposed approach is to provide an expressive formal framework for capturing various networking concepts. Analysis of a complex interaction of various concepts is possible by the common formal foundation.

The approach presented in this thesis performs analysis of network configuration to reveal errors that can demonstrate themselves in unwanted network behavior [38]. Networks with configuration errors may have a poor performance, cannot deliver expected services or can even expose serious vulnerabilities. Several studies reveal that 50%–80% of downtime and security issues are due to

configuration error (cf. [66], [63], [88]). Configurations of active network devices can be seen as a distributed program that through controlling individual network nodes implements the required network functionality. To work properly, configurations have to satisfy requirements on network security and functionality.

1.1 Network Requirements

Functional requirements are determined by the basic assumptions derived from design goals of the network [80]. A fundamental functional requirement on any network is to provide data delivery. Another example of functional requirement is the ability of a network to provide diagnostic data, e.g. in the form of logs from network devices or Netflow records.

Non functional requirements [42, 60] also known as dependability consists of the following classes:

- Availability – the degree to which network services is in a specified operable state.
- Reliability – the ability of a network to perform its required functions.
- Safety – the protection against damages, losses, harm, or any other undesirable consequences of network failure.
- Confidentiality – the absence of unauthorized disclosure of information from network communication.
- Integrity – the protection against improper network configuration modification.
- Maintainability – the degree which allows administrator to perform network modification and repairs.
- Quality – the capability of a network to provide a specified level of availability and reliability for selected functions or services.

In addition to network devices, other machines are commonly deployed to implement network security [17] and functionality. For example, various specialized security appliance can be deployed to monitor network traffic to improve network security. Providing accurate network model incorporating all deployed functionality is thus often impossible. Rather, abstract model inherited from configurations of network devices should capture properties significant for intention of an analysis. For instance, analysis of service availability one should check reachability of the service considering routing and access control configurations.

As conditions of the network are constantly changing, expected requirements may be violated. A network condition is a collective state of network devices and communication facilities. For instance, depending on the level of abstraction the state of communication facility can be represented by used link capacity. Nevertheless, even simpler model based on failures is useful in an analysis of network reliability with respect to accessibility of network services. In this model, the network condition is expressed by operational states of device and communication links.

Iannaccone, Chuah, Moriter, Bhattacharyya and Diot performed analysis of an effect of link failures in an IP backbone network in [40]. An impact of link or device failure in the environment with redundant paths is visible during a convergence period, which is the time needed for routing protocols to recompute routing information. From their analysis, it is evident that failures are very common in a typical network. Moreover, often failure events of short duration (about half of all

failures in the examined network) are caused by overloaded routers. These routers are temporarily unable to process all necessary events, e.g., maintaining adjacency between routing processes. Other nodes identify such device as failed.

1.2 Network Security

Packet filtering firewalls are supposed to enforce a network security policy by inspecting all packets flowing through them. Configurations of firewalls implement high-level security policy by deploying low-level rule-bases [14]. A significant effort was made to develop a method verifying that firewalls correctly enforce specified security policies (cf. [46], [12], [33], [27], [45]). As firewalls can contain thousands of rules, the verification methods have to be efficient. A firewall configuration usually consists of a collection of deny/permit rules. A rule that applies to a packet is selected by first-match semantics. Administrators often adjust firewall configurations to new demands by adding new rules and removing or modifying existing rules. It is not unusual that such modifications introduce an internal conflict within firewall rule-base. These conflicts are classified according their severity. A rule may hide another rule or overlap with other rule performing the same or a different action. Several algorithms were devised to find conflicting rules and to correct firewall rule-bases.

Bartal, Mayer, Nissim, and Wool in [9] introduce a tool for generating firewall rule bases from a security policy design. A security policy design is an abstract high-level description of security requirements. Their policy language introduces a concept of *Role*. A role defines a communication capability, e.g., a web server may accept any HTTP request from any computer on Internet. A role may restrict accessible services and allowed peers. Role groups aggregate individual roles, which enables to specify service availability for accumulations of hosts. A security policy comprises triples, each of them consisting of a group of source locations, a group of destination locations and permitted services. A policy compiler takes a specification of a security policy made by a security administrator and produces an appropriate firewall configuration files. Policy rules generation proceeds in two phases. First, a centralized collection of rules is generated. Second, this centralized rules are assigned to firewall devices from a network by analyzing network topology. By direct analysis of permitted services, it is possible to infer permitted application ports. From the analysis of network topology and mapping locations specified in roles to network locations it is possible to deduce IP addresses and also assignment of filtering rules to particular network firewalls.

Al-Shaer and Hamed analyze anomalies of firewall rule-base in [2]. They also present algorithms that uncover certain inconsistencies and conflicts by performing intra-firewall and inter-firewall analysis. The main declared contribution comprises the analysis of distributed firewalls. To classify anomalies, several kinds of relations among rules are determined, namely, completely disjoint, exactly matching, inclusively matching, partial disjoint, and correlated. A firewall policy is represented as a policy tree that enables to discover anomalies by checking coincidence of paths of any pair of rules. Rules whose paths do not coincide are disjoint. In the case of any coincidence, one need to apply an anomaly discovery algorithm to determine the relation between rules by comparing individual fields.

Inter-firewall anomaly discovery is a bit complicated task. Anomalies between different firewalls may contain:

- shadowing anomaly, if an upstream firewall blocks the traffic accepted by a downstream firewall,
- spuriousness anomaly, if an upstream firewall permits the traffic blocked by a downstream firewall,
- redundancy anomaly, if a downstream firewall denies the traffic already blocked by a downstream firewall,
- correlation anomaly, which can lead to shadowing or spurious anomaly.

An inter-firewall anomaly discovery algorithm reveals anomalies in a collection of firewalls. The algorithm considers all paths between analyzed domains as its input. For any path, it analyzes all firewalls along the path. First, firewalls are analyzed for intra-firewall anomalies. Then, a policy rule tree of the first firewall on the path is created. Rules of all consecutive firewalls extend the policy tree. Rules that applies to the processed path are marked. Finally, all added unmarked rules are reported as irrelevant. Any anomaly detected while adding new rules in the base policy rule tree is reported as warning or errors. Author also provide the thorough evaluation of presented algorithms demonstrating their efficiency on real scenarios.

Yan et.al have developed a tool called FIREMAN [91], which allows to detect misconfiguration in firewall settings. The FIREMAN performs symbolic model checking of the firewall configurations for all possible IP packets and along all possible data paths. The underlying implementation depends on a BDD library, which efficiently reduces firewall rules. This tool can reveal intra-firewall inconsistencies as well as misconfigurations that lead to inter-firewall conflicts. The tool can analyze ACL series of all paths for end-to-end connection thus offering network-wide firewall security verification.

Pozo, Ceballos, and Gasca [31] provided a consistency checking algorithm that can reveal four consistency problems called shadowing, generalization, correlation and independence. Liu et.al developed a method for formal verification and testing of (distributed) firewall rules (cf. [52], [53], and [56]) against user provided properties. They represent firewall rules in a structure called firewall decision diagram (FDD). A verification algorithm takes FDD and a requirement statement, e.g., a set of packets that should pass the firewall. The verification algorithm checks the given requirement by traversing an FDD from the root to leaf.

Jeffrey and Samak in [44] aims at analysis firewall configurations using bounded model-checking approach. They focus at 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 firewall configuration, it means to find a packet p which is not matched by any rule of the firewall rule set. They implemented analysis algorithm by translating the problem to SAT instance and showed that this approach is efficient and comparable to tools based on a BDD representation.

Techniques for detection of vulnerabilities of hosts and their protection against the network attack have been intensively studied (cf. [83], [92]). Ou, Govindavajhala, and Appel [67] introduce an automatic deduction of network security implemented in Prolog. The authors define reasoning rules that express semantics of different kinds of exploits. The implemented tool automatically extracts rules and facts from the OVAL scanner and the CVE database.

1.3 Contribution

This thesis introduces a novel approach that represents network policies, firewall configurations and network configuration as models interpreted against formalized context. Context expressed as domains of Formula system provides precise meaning and key assumptions about context's concepts. The presented thesis examines domain specifications for network configuration requirements at three different levels:

- *Network policy configuration* level represents a high-level view of network functionality. A network policy configuration expresses requirements on network functionality in terms of end-to-end reachability, security, and service availability.
- *Firewall configuration* level represents a detail view of access control implementation. At this level, individual firewall rules and relations between rules can be analyzed.
- *Network device configuration* level represents a declarative specification of device functionality. At this level, network functionality is analyzed as the combined effect of individual device configurations. Also, the detection of misconfiguration is possible at this level.

Various researchers have studied network configuration analysis problem from many aspects introducing several methods (cf. [6], [86], [28], [12], [47], [13][64], [90], [87]). The main contribution of this thesis is the proposal of a formal framework that can accurately explain many concepts from network policy, firewall configuration, and network device configuration domains. A constraint logic programming defines concepts in a form of logical constraints. A domain is an organizational unit that comprises related data types and associated constraints.

The proposed framework is extensible through specification of new domains. A domain precisely defines networking concepts. Due to advanced module system of Formula , a new domain can extend or include an existing domain, which reduces an amount of the necessary effort. Domain specification is challenging and time-consuming task. In addition to a concept formalization, an author has to provide all necessary assumptions and conformance rules.

1.4 Organization

The structure of the thesis itself is following:

- Chapter 2 gives an overview of Open-World Logic Programming (OLP) paradigm that provides the formal foundation for specifications of network concepts and their rigorous analysis. This chapter contains background information on a logic-based method used in this thesis. There are no novel concepts introduced in this chapter, but the explanation of the idea taken for modeling and analysis of network policy, network firewalls and network configurations. Chapter ends with a short overview of Formula language.
- Chapter 3 presents a network model intended for analysis of network layer flow reachability. The chapter provides the explanation of how to use this model to analyze whether a specified flow can reach a target node. Flow-based reachability can represent many network properties. Using this model, it is, for instance, possible to analyze routing configuration or access control enforcement. The chapter consists of state-of-the-art overview and specification of network reachability model

accompanied with illustrative examples. The chapter concludes by comparing this model with other related models intended for analysis of network functionality.

- Chapter 4 deals with network policy specification languages. It presents extended overview of existing languages, followed by analysis of their key features. Next, a novel policy specification language is introduced and formally defined. Formal semantics is built by using semantic anchoring approach that gives mapping for objects from language domain into interpretation domain.
- Chapter 5 starts with a description of the functionality of a typical firewall device as the introduction to the problem of implementing network security policy. Next, the chapter presents a unified formal model of a firewall device capturing the semantics of packet matching algorithm. Conflict classification framework and conflict classification algorithms have received substantial attention. The chapter finishes with the description of a new classification algorithm along with its performance evaluation.
- Chapter 6 describes an approach based on constraint relations for modeling and analysis of network configurations. A constraint model is built for the analyzed network with constraints representing functionality declared in configuration files. Configuration modeling by constraint relations is demonstrated using a collection of illustrative examples, e.g., representation of access control lists, network address translation, and other configuration features. The last part is devoted to the analysis of routing configuration, which is elaborated in details.

Finally, Chapter 7 provides conclusions by summarizing achieved results and discussing possible improvements and future work.

Chapter 2
A Constraint System

The use of logic programming is one of the approaches commonly used for configuration analysis. The advantage of logic programming approach is that it can directly express many networking concepts as relations and logic constraints. The approach presented in this thesis employs open-world logic programming paradigm. Open-world logic programming provides a suitable foundation for embedding domain specific languages supported with efficient reasoning techniques based on constraint solving methods. Using open-world logic programming, it is possible to analyze specifications by converting them to a set of constraints obtained from symbolic execution of the specifications. The output from quantifier elimination procedure is a formula that can be tested by state-of-the-art constraint solvers. The main purpose of this chapter is to provide an overview of *Open-World Logic Programming* (OLP) paradigm [41] that provides the formal foundation for specifications of network concepts and their rigorous analysis. Before presenting OLP, the notion of constraint relations and their importance for specification of network configuration features is discussed. A notion of a constraint relation comes from constraint databases and constraint logic programming. In the next section, constraint relations are defined using constraint Datalog as an example of CLP. For the development presented in this thesis a tool called Formula is used. Formula is a constraint logic programming system that offers some other useful constructs that simplify programming.

2.1 Constraint Relations

In this section, the notion of constraint relations is presented. This presentation employs a system often denoted as constraint Datalog [71] that is an example of constraint database systems [72]. Constraint databases and constraint programming emerged to tools able to solve practical problems [23]. Though Datalog was intensively researched in eighties, many applications emerged recently revived interest in this system [37]. Extending Datalog with constraint system enables to consider even more application domains [49],[75].

A constraint data model describes a logical view of data in a finite and compact way. A constraint database is a collection of constraint relations. A constraint relation is a finite set of constraint tuples. A Datalog sentence is a function-free Horn clause, where every variable in the head of a clause must appear in the body of the clause. Each sentence is either a fact or a rule. *Constraint* Datalog *facts* similarly to plain Datalog facts define ground information of the data model. In its constraint

version, each fact nevertheless can represent a (possibly infinite) set of items. A constraint Datalog fact has the following form:

$$\mathtt{f}(v_1, \ldots, v_n) \Leftarrow \phi_1, \ldots, \phi_m,$$

where $\mathtt{f}$ is the name of a constraint relation of this constraint tuple, v_i is either a constant value or a variable that must be bound to the constraint body. The body consists of a set of conjuncts in form of atomic constraints ϕ_j. An example of a constraint relation built from three facts is shown in Figure 2.1.

Constraint Datalog *rules* allow us to deduce new facts, which are refined with a set of constraints. A constraint rule has the following form:

$$\mathtt{r}_0(x_1, \ldots, x_n) \Leftarrow \begin{array}{l} \mathtt{r}_1(x_1, \ldots, x_n), \ldots, \mathtt{r}_m(x_1, \ldots, x_n) \\ \phi_1(x_1, \ldots, x_n), \ldots, \phi_k(x_1, \ldots, x_n) \end{array},$$

where r_0 represents a constraint relation being defined and $r_1, \ldots, r_n$ are constraint relations used to deduce new facts. In addition, rules can contain atomic constraints ϕ_j.

Constraints that can appear in constraint tuples are not arbitrary but have to be from constraint domains. A constraint domain is a set of objects together with a language for manipulating these objects. A constraint domain Φ is given by its signature Σ, interpretation $\mathcal{D}$, and a class of primitive constraints $\mathcal{L}$. For our purpose, we need two relatively simple constraint domains. Assuming that x, y are variables and c_i are constants, the generic domains are defined as follows:

- Range constraint domain Φ_R. Its primitive constraints have the form of $x = y, x = c$ and $x \in (c_1, c_2)$.
- Sets constraint domain Φ_S. The primitive constraints have the form of $x = y$ and $x \in \{c_1, \ldots, c_n\}$.

For most parts of this thesis, it is enough to instantiate these generic constraint domains with constants to get concrete constraint domains for representing constraints over IP addresses, port numbers, and protocol types.

```
%service(name, svc.ip, svc.pn, svc.pt)
service(web, 147.229.10.22, 8080, tcp).
service(ssh, 147.229.10.9, 22, tcp).
service(smb, x, 445, y) ⇐ x ∈ 147.229.10.0/24, y ∈ {tcp, udp}
```

Fig. 2.1 An example of a constraint relation

The `service` relation consists of four fields. The first field denotes a name of the service. Field `svc.ip` defines an IP address of a host, field `svc.pn` defines a port number and field `svc.pt` defines a transport protocol. The first two tuples are plain tuples. The last tuple is a constraint tuple. The first tuple specifies a host running the web service on port 8080. The second tuple specifies SSH service, and the third constraint tuple specifies that all computers in a subnetwork provide SMB service.

- $\Phi_R^{\texttt{ip}}$ – range constraint domain of IP addresses. It instantiates Φ_R with constants from range $0.0.0.0, \ldots, 255.255.255.255$ [1].
- $\Phi_R^{\texttt{pn}}$ – range constraint domain of port numbers. It instantiates Φ_R with constants from range $0, \ldots, 65535$.
- $\Phi_S^{\texttt{pt}}$ – set constraint domain of protocol types. It fixes the set of constants to those representing a valid protocol name, e.g., `ip`, `tcp`, `udp`.

Because of the finiteness of all domains, we can also easily define *range exclusion* and *set exclusion* constraints, which is useful when we need to deal with negation in rules or queries.

The computational complexity of constraint Datalog program[2] is determined by the complexity of used constraint domains. Li and Mitchell in [49] defined a notion of tractable constraint domains and showed that evaluation time complexity is polynomial in the size of the constraint Datalog program. They also proved that linearly decomposable domains are tractable. All above defined constraint domains have this property and thus are tractable. It means that Datalog queries in our data model can all be computed in polynomial time.

2.2 Open-World Logic Programming

In this section, Open-World Logic Programming (OLP) is briefly presented along with motivation for using OLP as a formal foundation of Logic-based Framework for Network Configuration Analysis.

Logic programming (LP) has been used for formal specification in many application areas. Adding constraints to an LP system is done by utilizing an external solver or extending the core system to dispatch constraints [21]. The difficulty in use of existing LP in problems that require constraint solving is that LP do not integrate smoothly with current decision procedures. For example, Prolog extended with the constraint solver [43] requires that users carefully introduce constraints within rules to guide proof search. It is because of the way Prolog evaluates subgoals of a rule. The position of a constraint in a rule can influence the execution, which in the worst case can lead to non-terminating programs.

Open-World Logic Programming attempts to overcome previously mentioned difficulties by incorporating decision procedures in the core engine to check if a goal is satisfied in a generated closure.

Consider the following trivial OLP program:

```
domain D {
     p(x) :- q(x), r(y), x > y.
}
```

A closure of a program is a collection of ground facts, e.g.:

```
model M of D {
    q(1). q(2). q(3). q(4). q(5).
```

[1] We use dot syntax for representing IP addresses, although internally it is represented as positive integer number. Often we also use a/m to write a range of IP addresses, where a represents an IP address and m represents mask.

[2] Queries expressible in Datalog without constraints can be computed in polynomial time.

```
                r(3), r(4), r(5).
}
```

Closed programs are expected to have a finite least fixpoint. The least fixpoint of the previous closure is the following set of facts:

$$\{q(1), q(2), q(3), q(4), q(5), r(3), r(4), r(5), p(5), p(4)\}$$

Evaluating an OLP query means to compute a least fixpoint of a closure and check if the goal is satisfied. For instance, query that proofs rather obvious property of the previously defined program

```
query count({x |  p(x)}) <= count({ y | q(y)})
```

is trivially satisfied in computed fixpoint of the closure. If a goal is not satisfied in the computed fixpoint then result of the query is false.

Computation is represented as a procedure that applies rules until a fixpoint of a closure is reached. To ensure efficient evaluation of programs, one needs to avoid the necessity to analyze an infinite number of elements. To detect programs that would introduce infinite number of elements in knowledge set, the OLP paradigm uses the following restrictions on programs:

- OLP checks safety of rules. A rule is safe if every variable used in a head of the rule occurs at the top-most scope of the rule. This means that following rule will be rejected:

$$p(x, y) \leftarrow q(y).$$

- Stratification is a syntactic restriction on rules that guarantees the existence of a least fixpoint. A stratified program does not permit cyclic dependencies that involve the negation. The negation is true if the set of expressions evaluates to the empty set. For example the following program that contains a cyclic dependency between n and m rules:

```
n ::= (Integer).
n(x) :- q(x), no m(x).
m ::= (Integer).
m(x) :- q(x), no n(x).
```

Open-world logic programming is an approach that deeply integrates modern constraint solvers in evaluation of logic programs. OLP works differently than classical LP when proving a goal. Instead of application of rules in backward or forward fashion, OLP attempts to find a goal by solving a satisfiability problem. The idea of solving this problem is to perform symbolic execution of the program extended with primitive relations and solve the resulting set of constraints in the state-of-the-art satisfiability modulo solver (SMT). As SMT contains various theories, e.g., linear arithmetic, bit vectors, arrays, formulas representing generated constraints can include variables ranging over infinite domains or complex structures. Consider, evaluation of a simple query `?- p(x)`. Symbolic execution of a program extended with a single fact $q(4)$ is represented by a formula:

$$p(x) \leftarrow q(x) \wedge r(y) \wedge x > y \wedge q(4)$$

SMT solver finds that this formula can be satisfied by finding values for both variables, $x = 4$ and $y > 4$. Based on this information, the OLP engine selects a next fact that extends the closure of the program and create a new formula to be dispatched by SMT:

$$p(x) \leftarrow q(x) \wedge r(y) \wedge x > y \wedge q(4) \wedge r(5)$$

The solver confirms satisfiability of this formula, which gives us the answer to the query, $p(4)$. Practical implementation of the OLP engine includes optimization techniques to perform efficient evaluation of queries and to avoid unnecessary operations, e.g., double checking same constraints.

The advantage of OLP approach is that a constraint solver is deeply integrated and used for evaluation of each constraint generated from symbolic execution of a logic program.

2.3 Formula Language

Microsoft Research's Formula is an LP system for open-world logic programs and algebraic data types. This tool is intended as a formal system for specifications of domain-specific languages. Formula implements efficient reasoning techniques about strongly typed open-world logic programs. Syntax of Formula is similar to the syntax of Prolog except that terms contain type annotations.

Formula has a small collection of built-in data types, e.g., Numbers and Strings, accompanied with (interpreted) operations on them. For example, usual arithmetic operations are available for all numeric types.

The power of Formula type system lies in the possibility to build derived data types by using data constructors that has form of:

```
ConstructorName ::= (Arg1 : TypeExpr1, ..., ArgN : TypeExpr1).
```

Formula also provides enumeration types that represent typed collections of symbols. Enumeration types are more flexible than in traditional programming languages, as any object can be a member of an enumeration. Enumerations can also include integer ranges:

```
EnumType ::= {RED, GREEN, "Foo", -1000..1000, 1001..1001, 1002}.
```

Formula also provides union types. Discriminated union types contain values that can be one of the values of specified types. Discriminated unions are useful for modeling heterogeneous data. Following type represents a nullable boolean type:

```
NullableBool ::= Boolean + { NULL }.
```

In Formula , discriminated unions are also necessary when one needs to represent recursive data structures, e.g., tree data structures or lists. A data type of integer lists is represented using union data type as follows:

```
IntList ::= new (hd:Integer, tl:any IntList + { NIL }).
```

This representation is close to the definition of list data type known from functional programming languages. Formula supports a manipulation with lists by providing a set of interpreted functions.

Subtyping in Formula is based on structural rather than a nominal equivalence. Subtyping relation (<:) is defined implicitly by the values of types. Type t_1 is a subtype of t_2 when set of all values of t_1 are subset of values of t_2. This subtyping scheme provides very flexible typing, for instance, the following is valid in Formula:

```
{ 1, 2 } <: PosInteger <: Natural + {"Foo","Bar"} <: Real + String
```

Possibility to define complex data types together with flexible typing system makes Formula suitable environment for modeling network concepts. For example, following types provide a direct representation of simplified IP header:

```
Protocol ::= { HOPOPT, ICMP, IGMP, ... , TCP, ..., UDP,  ... }
IP ::= {0..4294967295}
Dscp ::= new (class:{ BE, EF, AF, CS}, value:Priority).
IpPacket ::= new (pt:Protocol, srcIp:IP, dstIp:IP, dscp:Dscp).
```

Open-world logic programming paradigm enables to write logic programs where some parts of the program remain unknown until the program is analyzed. To analyze program P, its closure is first computed by adding ground facts that gives Herbrand base B_p.

In Formula, a logic program is implemented as a set of rules. Each rule has form of:

```
head :- body1, ..., bodyn.
```

Each *head* must be formed only from constants, constructors, and variables. A rule proves *head* if every *body* clause can be shown true in the current program, formally:

$$S \models H \leftarrow B_1, \ldots, B_n.$$

where S is Herbrand structure and $S \subseteq B_P$, for Herbrand base B_P of program P. Rules of a program together with facts and assumptions are organized in a module called *domain*. Open-world paradigm enables to avoid specifying all facts with the program. Following domain represents a logical program that computes paths in a network:

```
domain Paths {
    % Vertex data type
    V ::= new (lbl: Integer).
    % Edge data type
    E ::= new (src: V, dst: V).
    % following rule implements path computation
    path ::= (V, V).
    path(u, w) :- E(u, w); E(u, v), path(v, w).
}
```

As it can be seen, no information on network is provided. We only know that a network should be defined in terms of data type V standing for vertices and data type E representing edges.

Ground facts are provided in a separate module called *model*. When declaring a model the reference to a corresponding domain is required. The domain declares data types for which the model contains concrete instances. An example of a model for previously defined *Path* domain is following:

```
model N1 of Paths {
    v1 is V(1).
    v2 is V(2).
    v3 is V(3).
    v4 is V(4).
    v5 is V(5).
    E(v1, v2).
    E(v1, v3).
    E(v1, v4).
    E(v1, v5).
}
```

Model N_1 defines five vertices and four edges to represent a network that has a star topology. Separation of program and data is natural for formal specification and analysis, as it enables to reuse various programs for different data. Also, domains and models can be combined to create more complicated modules. Domain composition allows type declarations, rules, and conformance constraints to be combined. Model composition enables to add new assertions to existing models or combine more models into a single one. Module composition enables to apply modular specification approach.

Except modeling capabilities, FORMULA tool can be used to analyze models. For example, one may want to check if there is a path from node V_2 to node V_5 in model N_1 by executing the following query:

```
query N1 path(V(1),V(5))
```

When, it is necessary to know also why a query is true, a proof tree can be obtained. A proof tree provides information on rules used and values of matching constraints triggered rules.

Chapter 3
Network Reachability Analysis

This chapter presents a network model intended for analysis L3-level flow reachability. The network model is represented as a directed graph. Nodes describe network locations (interfaces). Edges describe connection among these interfaces. A path in this model thus depicts a data path in a network. Various operations that network devices can perform on traffic is specified by filter and transform constraint relations. Flow reachability can be analyzed in the network model. It means that it is possible to examine whether a specified flow can reach the target node. Many network properties can be expressed in terms of flow-based reachability. Using this model, it is, for instance, possible to analyze routing configuration or access control enforcement.

An early report on formal analysis of end-to-end network reachability was presented in 1997 by Guttman. He defined a formal method to compute a set of filters for individual devices given a global security policy [25]. He introduces an abstraction by including only network areas and border routers in a model. This natural decision mirrors the real situation as internal routers do not usually participate in data filtering. Similarly, data flow model is defined in terms of abstract packets. An abstract packet comprises of source and destination addresses and service types. Guttman also proposed an algorithm for computing a reachability set of packets that passes all filtering rules along any considered path. The employed abstract packets description make the procedure practically feasible and efficient. The Guttman's method can compute network reachability in a network model that counts only for filtering rules. He implemented a prototype of the algorithm in a Lisp-like language. This implementation can generate distributed firewall configuration based on a global security policy, and check the given policy implementation against its specification. A global security policy has form of reachability requirements that define a set of services, servers providing these services and clients authorized to access them.

Later on, Guttman and Herzog extended the previous approach to deal also with IPSec gateways [26] and then combine both approaches to a uniform framework [27]. The method deals with a simple network model that has the form of a bipartite graph, where nodes are routers and networks, and edges are interfaces, which have associated filters. For representing filters, an abstract packet representation is given. A packet consists of selected fields from the IP header. A term called *trajectory* is used for a pair composed of a packet and a path on which this packet is allowed. Trajectory can precisely represent a security policy. For instance, a typical policy rule reads, if p was ever in area A and later reaches area B, then p should be a TCP packet destined for port 25 at the target SMTP server. Authors claim that most of network security policies can be formulated in this way. To evaluate this approach, authors implemented an experimental program called Network Policy Tool. For

efficient evaluation, symbolic representation is used for network and host addresses. Each range of IP addresses has assigned a symbolic name in filtering rules. Filtering rules are converted to sets that describe their meaning using this symbolic representation. This tool finds its applications when the effect of firewalls on data delivery is to be examined.

IPSec configuration analysis checks that private traffic does not leak to the public network without application of a protection mechanism. The IPSec network model is a state machine. The transition relation encodes five basic packet operations:

- Create operation represents introducing a new packet by the source node.
- Discard operation represents deleting a packet either by target node or some filter in a network.
- Move operation represents forwarding a packet from a node to its neighbor.
- Prefix operation adds a new header to a packet.
- Pop operation removes the outermost header from a packet.

The notion of *trust set* is introduced expressing valid the network locations for a packet. A collection of trust sets formalizes security requirements. For IPSec secured traffic T, its trust set contains nodes from private networks connected by IPSec tunnels. IPSec configuration validation is performed by computing trust sets from a configuration and checking that it corresponds to trust sets of security requirements.

Xie et.al present approach for static analysis of IP networks [90]. They define a framework able to determine lower and upper approximations of network reachability. A network model uniformly describes filtering rules, dynamic routing, and packet transformations. The method computes symbolically a set of packets that can be carried by each link. By the combination of these sets along all possible paths between two endpoints, it is possible to determine the end-to-end reachability. The upper approximation fixes the set of packets that can be delivered by the network in *some* forwarding state. The lower approximation fixes the set of packets that can be delivered by the network in *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. They show how to convert dynamic routing information to filtering rules. Algorithms are presented for this conversion and for estimating lower and upper bounds on reachability. Finally, the problem of reachability analysis is discussed in larger context by exploiting possible applications, e.g., analysis of failure scenarios, validation of design patterns, and a combination with on-line data from network monitoring systems.

Bandhakavi et al. [8] extended the approach from [90] for practical analysis of end-to-end network. The difference is in separating routing and filtering, which simplifies the network model. Checking end-to-end reachability is a process that consists of four phases:

1. A network model consisting of device models is populated with configuration information.
2. Route advertisement graph are constructed for each target network. The computation is performed according rules specified by each participating device. Routing information base is filled with computed routes.
3. Information on possible routes is used to calculate all end-to-end connectivities.
4. End-to-end reachable paths are checked against the requirements. Any violation of the requirements is reported together with suggested fixes.

The effect of routing to end-to-end reachability is included in the analysis by computing route graphs. These graphs are computed for every target network. To calculate route graphs, an approach from [90] is employed. From route graphs, all paths for the given endpoints are selected and on these paths filtering rules are evaluated with respect to given security requirements. The method also extends to round trip flows, which is useful for analysis of stateful firewall rules.

Bera, Dasgupta and Ghosh (cf. [10], [12] and [11]) define a firewall verification framework. The framework can check the correctness of distributed ACL implementations against the given global security policy. The framework can also check a reliability (or fault tolerance) of a network. To check a correctness, filtering rules are translated into quantifier-free formulas that are together with the interpretation of the global security policy sent to a SAT solver. When filtering rules does not conform to the security policy, the SAT solver will produce a counter-example helping administrator to debug 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 to compute whether the rules are consistent with the given policy. A policy is understood as a description of service availability with respect to defined network zones. First, the method computes a network access model, which is a directed graph with filters assigned to its edges. Next, Service Flow Graphs (SFG) are generated for all services of a network, e.g., SSH, Web or E-Mail. An SFG is a subgraph of network access graph. A minimum cut for SFG is computed to determine what link and device failures can be tolerated.

Gan and Helvik in [22] propose to apply probabilistic methods to reduce the size of possible network states. They use stochastic activity networks [74] to describe the failures and repairs of network components and other dynamic issues of the network.

Another algorithmic framework based on probabilistic calculations is discussed by Menth, Duelli, Ruediger, and Milbrandt in [60]. They present a framework for the analysis of ingress-egress unavailability and link congestion. The framework can deal with three kinds of issues, namely, link of device failures, changes in user behavior and rerouting.

3.1 A Network Reachability Model

In this section, a network reachability is defined in terms of a constraint data model [72]. The idea behind this approach is that basic network operations on packets can be directly represented using constraint relations. A constraint relation is a finite set of constraint tuples, which enable compact representation of large and possibly infinite sets of values. For instance, network filters can be defined as constraint relations that restrict sets of permitted packets. Also packet transformations can be described by constraint relations, e.g., Network Address Translation (NAT) can be given as a constraint relation on pairs of packets. In general, by a combination of filters and transformations it is possible to express observable behaviors of network devices. This approach provides a general and expressive method for representing many operations on packets, e.g., tunneling, type of service marking, or policy based routing. The discussed approach is not limited to the flow-based model which is presented in this thesis. A flow-based model of communication assumes that packets are aggregated according to key header fields. Consequently, constraint relations are based on flows instead of individual packets.

A reachability data model comprises from the following basic constraint relations:

- `filter`(f, p) – constraints filter f to permit packet flow sets p,
- `transform`(t, p_i, p_o) – expresses packet transformation t as constraint relation between input flow sets p_i and output flow sets p_o,
- `subnet`(s) – defines s to be a subnetwork in a network topology,
- `interface`(l) – defines l to be an interface in a network topology,
- `idge`(i_s, i_d, t) – specifies an internal link inside forwarding devices between i_s and i_d interfaces and assigns it a transformation t,
- `edge`(e_s, e_d, f) – specifies an external link between interface and subnetwork e_s, e_d and assigns it filter f.

These constraint relations form *a network reachability domain* and are used as an input to a verification method.

A network model describes structural properties of a network, and interpretation of network device functionality that are relevant for reachability analysis. In particular, it takes a network topology and an internal structure of forwarding devices to form a bipartite graph labeled with constraints.

Definition 1. A network reachability graph $G = \langle S, L, I, E, \delta, \gamma \rangle$ consists of a set of *subnetwork vertices* S and a set of *interface vertices* L. Interface vertices are connected by *internal edges* (idges) $I \subseteq L \times L$. External *edges* $E \subseteq L \times S \cup S \times L$ connects interfaces with subnetworks. Labeling $\delta : I \to t$, maps internal edges to transform constraints. Labeling $\gamma : E \to f$ maps external edges to filter constraints.

Fig.3.1 contains an illustrative example of the defined network model. An internal structure of each device is a fully connected directed graph with all interfaces of a router as vertices. For modeling complicated device functionality, auxiliary internal vertices can be added. Each internal edge maps to a constraint. A sequence of internal edges thus represents a composition of constraints. By the constraint composition, complex processing can be modeled.

The network model is translated to a constraint data model by defining `subnet`, `interface`, `idge` and `edge` relations, which directly capture the topology of a network. These relations can be easily generated from the network reachability graph. A Formula domain that formalizes network reachability is shown in Fig.3.2.

3.2 Flow and Session Model

Expressing filters and transformations depend on a flow model. A flow model describes sets of packets by their characteristic attributes. It is, of course, more efficient to analyze whether a certain set of packets can reach the target node instead of computing reachability for individual packets. In addition to flow objects, a session object is defined. A session can be useful if properties are defined in terms of bidirectional communication, e.g., for analysis of network address translation or stateful firewalls.

A `flow` constraint relation describes unidirectional communication consisting of packets that share certain properties. The set of packet's attributes depends on the purpose and the scope of the analysis. In this chapter, following five keys describe a packet abstractly:

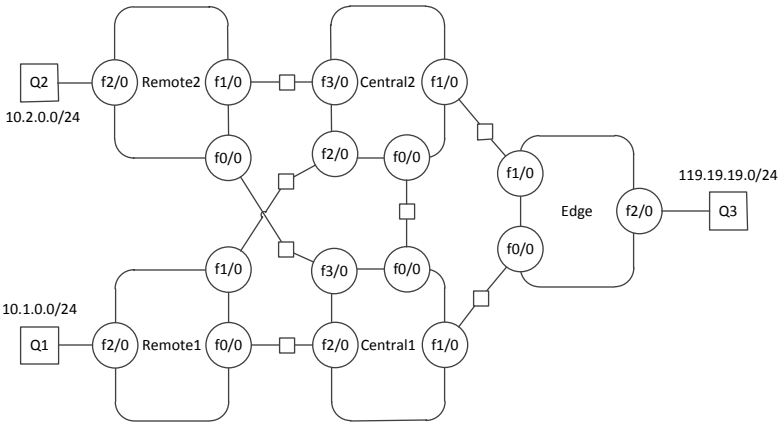


Fig. 3.1 An example of network model

Routers are visualized as rounded rectangles, labeled with their names, e.g. `Central1`. A router contains interfaces, which appear as discs with labels that correspond to their local identifiers. Rectangles stand for communication facilities, e.g. point-to-point links or destination networks.

```
1  domain Topology {
2      ID ::= Integer + String.
3      Device ::= new (id:ID).
4      Interface ::= new (id:ID, dev:Device).
5      Subnet ::= new (id:ID).
6      Idge ::= new (from:Interface, to:Interface).
7      EdgeOut ::= new (from:Interface, to:Subnet).
8      EdgeIn ::= new (from:Subnet,to:Interface).
9      Edge ::= EdgeOut + EdgeIn.
10
11
12     conforms no { e | e is EdgeOut, f is EdgeIn,
13                       e.from = f.to, e.to = f.from}.
14     conforms no { e | e is EdgeIn, f is EdgeOut,
15                       e.from = f.to, e.to = f.from}.
16     conforms no { d | d is Device, i is Interface(_,d),
17                       j is Interface(_,d), i != j, no Idge(i,j)}.
18 }
```

Fig. 3.2 Topology domain implementation

Topology domain defines new data constructors for device, interface, subnet, idge and edge concepts. Conformance of topology domain is asserted at lines 12-17. First pair of conformance constraints asserts that all edges are bidirectional. The last conformance constraint asserts full mesh connectivity among all interfaces within each device.

- `srcIp` - source IP address,
- `dstIp` - destination IP address
- `srcPn` - source port for UDP or TCP, unused for other protocols,
- `dstPn` - destination port for UDP or TCP, type and code for ICMP, unused for other protocols,

- `app` - protocol type, e.g., IP, ICMP, TCP, UDP.

In a constraint relation system, a flow f can be defined by providing a constraint $\phi(p)$:

$$\texttt{flow}(f, p) \Leftarrow \phi(p)$$

Notation $p[f]$ is used to operation that selects field f of record p. In the following example, a HTTP(S) traffic from subnetworks *12.34.56.0/24* to *91.83.74.65* is specified as flow f_1. It is written as a constraint on variable p, which assigns a set of packets to flow f_1.

$$\texttt{flow}(f_1, p) \Leftarrow \begin{array}{l} p[\texttt{srcIp}] \in 12.34.56.0/24, \\ p[\texttt{dstIp}] = 91.83.74.65 \\ p[\texttt{dstPn}] \in \{80, 443\} \\ p[\texttt{app}] = \texttt{tcp} \end{array}$$

One may notice that not all fields of a packet are constrained in the example. Those fields can contain any value from its domain, i.e., a missing field in the presented f_1 constraint tuple is implicitly constrained by $p[srcPn] \in \mathcal{D}_R^{pn}$.

It would be possible to formulate a constraint query:

$$\begin{array}{ll} ?- & \texttt{flow}(f_1, [\texttt{srcIp} = 12.34.56.123, \texttt{dstIp} = 91.83.74.65, \\ & \qquad \texttt{srcPn} = 6543, \texttt{dstPn} = 25, \texttt{app} = \texttt{tcp}]) \end{array}$$

that yields to `false` as the destination port in the query does not match to any of the specified number for flow f_1. Although constraints may contain complex logical expressions, in most cases simplistic properties are specified as shown in the example.

Formalization of the flow model as Formula domain is not direct. One of the possible approach is to define constraint relations for each of packet's fields. Then a collection of packets can be defined in terms of these constraints. Flows object is than defined to contain all packets that satisfy these constraints. See Fig.3.3 for implementation of `Flow` domain. A model of flow f_1 is

```
model F of Flow {
   p is Packet("f1").
   SrcIp(p,IpRange(203569152,203569407)).
   DstIp(p,IpRange(1532185153,1532185153)).
   DstPn(p,PnRange(80,80)).
   DstPn(p,PnRange(443,443)).
   App(p,TCP).
}
```

and query can be expressed as follows:

```
   query F flow("f1", 203569275, 1532185153, 6543, 25, TCP)
```

A session model aims at pairing flows that belong to a same bidirectional data communication. Monitoring data communication on a single locality in a network, a session observed at the transport layer consists of flows that have the same protocol and corresponding addresses and ports. This is

```
domain Flow includes Range {
   ID ::= Integer + String.
   Ip ::= {0..4294967295}.
   Pn ::= {0..65535}.
   Pt ::= {IP,ICMP,IGMP,TCP,UDP}.
   Packet  ::= new (id:ID).
   IpRange ::= new (l:Ip, r:Ip).
   PnRange ::= new (l:Pn, r:Pn).
   SrcIp ::= new (p:Packet, IpRange).
   DstIp ::= new (p:Packet, IpRange).
   SrcPn ::= new (p:Packet, PnRange).
   DstPn ::= new (p:Packet, PnRange).
   App   ::= new (p:Packet, Pt).

   packet ::= (p:Packet,srcIp:Ip,dstIp:Ip,srcPn:Pn,dstPn:Pn,app:Pt).
   packet (p,srcip,dstip,srcpn,dstpn,app) :-
        SrcIp(p,r_srcip),Range.in(srcip, r_srcip),
        DstIp(p,r_dstip),Range.in(dstip, r_dstip),
        SrcPn(p,r_srcpn),Range.in(srcpn, r_srcpn),
        DstPn(p,r_dstpn),Range.in(dstpn, r_dstpn),
        App(p,r_app), app = r_app.

   flow ::= (id:ID,srcIp:Ip,dstIp:Ip,srcPn:Pn,dstPn:Pn,app:Pt).
   flow(id,srcip,dstip,srcpn,dstpn,app) :-
       p is Packet(id), packet(p, srcip,dstip,srcpn,dstpn,app).
}
```

Fig. 3.3 Flow domain implementation

For simplicity, only ranges over IP domain and Port domain can be specified in constraints. Rule `packet` enumerates all packets associated with a specified packet group identifier. Rule `flow` can be used in queries to determine properties of flow built from packet group.

formalized as follows:

$$\texttt{session}(s_1, p_1, p_2) \Leftarrow \begin{array}{l} p_1[pt] = p_2[pt] \\ p_1[src.ip] = p2[dst.ip] \\ p_1[dst.ip] = p2[src.ip] \\ p_1[src.pn] = p2[dst.pn] \\ p_1[dst.pn] = p2[src.pn] \end{array}$$

Later, it will be shown, how the session model will become useful for analysis of network address translation mechanism. It is possible to give a different definition for a session relation. A session model for DNS communication also requires matching a DNS query identifier in addition to port numbers.

3.2.1 Packet Classification

A packet classification is needed by various mechanisms, e.g., packet filtering, packet routing, Quality-of-Service, or Network Address Translation. A packet classification problem according Eppstein and Muthukrishnan [20] is defined as to classify a packet P by finding the classification rule of the highest priority in a classification database F. A sufficient abstraction of packets and rules for IP networks is based on d-dimensional ranges.

A packet classification rule c is a collection of d-dimensional ranges $[l_1^c, r_1^c] \times \ldots \times [l_d^c, r_d^c]$, an action a^c and a priority r^c. Any packet p is then a d-dimensional vector $[p_1, \ldots, p_d]$. These values are derived from a packet header, e.g., source and destination address, port numbers, type of service, etc. The matching condition states that a packet p is classified by rule c, if and only if

$$\forall i \in 1..d : p_i \in [l_i^c, r_i^c] \wedge \nexists c' \in F : r^{c'} < r^c \wedge p_i \in [l_i^{c'}, r_i^{c'}].$$

A priority is a way to solve possible conflicts among the classification rules. In general, the conflict between rules arises if they overlap in all their dimensions. A precise and detailed taxonomy of conflicts was established by Al-Shaer and Hamed in [2] and is detailed elaborated in Chapter 5. It is convenient to use conflict free rule sets for representing packet classifiers in the form of constraint relations. In particular, for the efficient execution of a decision procedure the representation of a classifier should have the following properties:

- A set of filters should be disjoint. This eliminates multiple evaluation for the overlapping ranges.
- A set of filters should be complete in the sense that it covers the whole address space in all dimensions. This guarantees the completeness of the evaluation.

Presented approach follows the work proposed by Hari, Suri and Parulkar [30]. They devise an algorithm for detecting and eliminating conflicts in packet filters by constructing prefix trees for each dimension and identifying conflicts during processing new rule. If a conflict is detected then, a resolve filter rule is introduced, which represents an intersection of overlapping parts of conflicting filters. The approach based on the prefix tree construction requires that filters use prefix representation for their fields. It does not pose a problem as it is possible to represent an arbitrary range $[0, 2^k]$ into at most $2k$ prefixes as demonstrated by Srinivasan et al. in [76].

3.2.2 Packet Filtering

In the network reachability model, filters are a natural representation for access control lists. An access control list (ACL) consists of rules ordered by their priority. Each rule has an action and attributes that define matching condition. To find an action for packet p, rules are checked from the top of the list. The first rule that matches packet p is selected, and its action is executed. Overlapping rules are common in practice as they often make ACL shorter and better expressing the administrator's intention, but in several cases overlapping signalizes the configuration error.

A packet classification method discussed in the preceding subsection can be directly applied to represent ACLs. A result of an application of the algorithm on an example ACL is shown in Figure

```
ip access-list extended ACL-ACCESS-OUT
1 permit icmp 10.255.0.0 0.0.255.255 10.1.0.0 0.0.255.255
2 deny   icmp any 10.1.0.0 0.0.255.255
3 permit ip any 10.1.0.0 0.0.255.255
4 deny   ip any any
```

$$\texttt{filter}(f_{acl}, 4, deny, p) \Leftarrow p[\texttt{pt}] = \texttt{ip}, \begin{array}{l} p[\texttt{src.ip}] \in [0.0.0.0 - 255.255.255.255] \\ p[\texttt{dst.ip}] \in [0.0.0.0 - 10.0.255.255] \end{array}$$

$$\texttt{filter}(f_{acl}, 4, deny, p) \Leftarrow p[\texttt{pt}] = \texttt{ip}, \begin{array}{l} p[\texttt{src.ip}] \in [0.0.0.0 - 255.255.255.255] \\ p[\texttt{dst.ip}] \in [10.2.0.0 - 255.255.255.255] \end{array}$$

$$\texttt{filter}(f_{acl}, 2, deny, p) \Leftarrow p[\texttt{pt}] = \texttt{icmp}, \begin{array}{l} p[\texttt{src.ip}] \in [0.0.0.0 - 10.254.255.255] \\ p[\texttt{dst.ip}] \in [10.1.0.0 - 10.1.255.255] \end{array}$$

$$\texttt{filter}(f_{acl}, 2, deny, p) \Leftarrow p[\texttt{pt}] = \texttt{icmp}, \begin{array}{l} p[\texttt{src.ip}] \in [11.0.0.0 - 255.255.255.255] \\ p[\texttt{dst.ip}] \in [10.1.0.0 - 10.1.255.255] \end{array}$$

$$\texttt{filter}(f_{acl}, 3, permit, p) \Leftarrow p[\texttt{pt}] = \texttt{(udp,tcp)}, \begin{array}{l} p[\texttt{src.ip}] \in [0.0.0.0 - 255.255.255.255] \\ p[\texttt{dst.ip}] \in [10.1.0.0 - 10.1.255.255] \end{array}$$

$$\texttt{filter}(f_{acl}, 1, permit, p) \Leftarrow p[\texttt{pt}] = \texttt{icmp}, \begin{array}{l} p[\texttt{src.ip}] \in [10.255.0.0 - 10.255.255.255] \\ p[\texttt{dst.ip}] \in [10.1.0.0 - 10.1.255.255] \end{array}$$

Fig. 3.4 An example of filter constraints.

An extended ACL is a set of rules each consisting of action, protocol name, source address/wildcard and destination address/wildcard. This ACL is applied on *f2/0* interface of Remote1 router and secures destination network *10.1.0.0/16*. For Icmp it permits communication only from admin network and denies all other (rules no.1 and 2). Rule no.3 permits all communication to destination hosts. Finally, rule no.4 denies all communication.

3.4. There are three classification dimensions, namely, protocol type, source address, and destination address. While $\texttt{filter}(f_{acl}, 1)$ equals to rule no.1 in all of its attributes, filter tuples representing rules no.2 and no.4 were modified to remove overlaps in source and destination addresses. An interesting case is $\texttt{filter}(f_{acl}, 3)$, which was restricted to match only Tcp and Udp communication because rule no.2, which has higher priority captures all Icmp communication.

3.2.3 Transformations

Packet transformations represent operations that take input packets and create different or modified output packets, e.g., network address translation (NAT), IPSec protection, QoS marking. Network model enables to assign transformations on all internal edges. Modeling NAT functionality stands for assigning a transformation to those idges of the router that participate on NAT (see Figure 3.1, router Edge). Figure 3.6 contains an example NAT configuration and corresponding transformation constraints.

Static NAT is configured by providing mapping from source private IP addresses to public IP addresses. An example of static NAT is in Fig. 3.5. SNAT is encoded using a new relation $\texttt{snat}(Id, In, Out)$, which interprets address translation. The transform tuple consists of if-then-else macro expression. Its meaning is obvious. If p and q are related as prescribed by one of the specified static translation then the transform tuple is defined in this way. If none is applicable then p equals q and no translation is in action. Reverse translation can be encoded using a similar approach.

```
1   ip nat inside source static tcp 10.1.10.1 25 119.19.19.2 25
2   ip nat inside source static tcp 10.2.10.1 80 119.19.19.2 80
```

$$\texttt{snat}(1,p,q) \Leftarrow \begin{array}{l} p[\texttt{src.ip}] = \textit{10.1.10.1}, q[\texttt{src.ip}] = \textit{119.19.19.2}, \\ p[\texttt{src.pn}] = 25, q[\texttt{src.pn}] = 25, \\ p[\texttt{pt}] = \texttt{tcp}, q[\texttt{pt}] = \texttt{tcp}, p[\textit{dst}.*] = q[\textit{dst}.*]. \end{array}$$

$$\texttt{snat}(2,p,q) \Leftarrow \begin{array}{l} p[\texttt{src.ip}] = \textit{10.2.10.1}, q[\texttt{src.ip}] = \textit{119.19.19.2}, \\ p[\texttt{src.pn}] = 80, q[\texttt{src.pn}] = 80, \\ p[\texttt{pt}] = \texttt{tcp}, q[\texttt{pt}] = \texttt{tcp}, p[\textit{dst}.*] = q[\textit{dst}.*]. \end{array}$$

$$\texttt{transform}(t_{nat}^{fwd}, p, q) \Leftarrow \texttt{snat}(x, p, q)$$
$$\texttt{transform}(t_{nat}^{fwd}, p, q) \Leftarrow \neg\texttt{snat}(x, p, r), p = q$$

Fig. 3.5 An example of transform relation for SNAT

The NAT configuration provides that SMTP server at *10.1.10.1* host will be visible at router's public address at port 25 and HTTP server at *10.2.10.1* host will be reachable at router's public address at port 80. `transform` relation compose of two clauses. The first clause denotes an application of the defined translations. The second clause ensures that flows not translated by NAT remain unchanged.

Network Address Port Translation (NAPT), also called NAT overloading, allows mapping many private addresses to a single external address. To do so, also transport ports are involved in the translation. For an outbound packet flow, the NAPT translates the source IP address, the source transport identifier to a new values adjusting some other header fields, e.g., checksums. For inbound packets, the NAPT translates the destination IP address and the destination transport port. Constraint relation can specify additional constraints on translation, e.g., selection of port numbers based on appropriate port groups (0-511, 512-1023, or 1024-65535). The NAPT example is shown in Figure 3.6. For forward translation, t_{nat}^{fwd}, there are two options: If a flow matches an ACL, packets are translated. Otherwise, the packets are forwarded unchanged.

The first constraint tuple defines a translation of source addresses and ports of packets that matches ACL 1. The second constraint tuple deals with other packets that are processed without modification. For encoding reverse transformation, t_{nat}^{rev}, `session` relation is exploited.

3.3 Reachability Calculation

This section shows how to use the developed network reachability model for analysis. This analysis although limited only to determining a set of flows that can reach a given subnetwork can answer many interesting questions either from the perspective of ensuring service availability or guarantee service security. An analysis method determines if a supplied property on packet reachability is satisfied in a network model.

```
ip nat pool HOSTS 119.19.19.2 119.19.19.2 netmask 255.255.255.0
ip nat inside source list 1 pool HOSTS
access-list 1 permit 10.0.0.0 0.255.0.255
```

$$
\begin{aligned}
&\texttt{transform}(t^{fwd}_{nat}, p, q) \Leftarrow \begin{array}{l}\texttt{filter}(1, \_, \texttt{permit}, p),\\ q[\texttt{src.ip}] \in \texttt{HOSTS}, p[\texttt{dst.ip}] = q[\texttt{dst.ip}],\\ p[\texttt{dst.pn}] = q[\texttt{dst.pn}], p[\texttt{pt}] = q[\texttt{pt}].\end{array}\\
&\texttt{transform}(t^{fwd}_{nat}, p, p) \Leftarrow \texttt{filter}(1, \_, \texttt{deny}, p)\\
&\texttt{transform}(t^{rev}_{nat}, p, q) \Leftarrow \begin{array}{l}\texttt{session}(p, p^{rev}), \texttt{session}(q, q^{rev}),\\ \texttt{transform}(t^{fwd}_{nat}, q^{rev}, p^{rev})\end{array}
\end{aligned}
$$

Fig. 3.6 An example of transform relation for NAPT

The NAT configuration translates packets captured by ACL 1. Public addresses are taken from `HOSTS` pool. If packets are not permitted by ACL 1, then translation is not applied. Of three `transform` clauses, the first clause represents transformation of packets as defined by NAT, the second clause represents a case when no translation is applied, and the third clause represents backward transformation.

3.3.1 Restricted Join

The constraint relational algebra is equipped with six primitive operators, namely, set union ($\cup$), set difference ($\setminus$), set intersection ($\cap$), projection (π), selection (σ) and rename (ρ). Among derived operators, the join operator ($\bowtie$) is one of the most useful. Natural join operator $A \bowtie B$ is applied to an n-dimensional relation A and an m-dimensional relation B. If they have k common attributes then the natural join operator returns $(n + m - k)$-dimensional relation with projections $\pi_A(A \bowtie B) = A$ and $\pi_B(A \bowtie B) = B$.

For the purpose of reachability set calculations, a new operator called *restricted join* is introduced. The purpose of this operator is to allow for controlled composition of filters and transformations.

Definition 2. Given, transformations t, s and filters f, g. A restricted join operator relation that represents a composition of t,f,g and s is defined as:

$$t \overset{f,g}{\bowtie} s \triangleq \pi_{p_i,p_o}((\rho_{p_o/p}t \bowtie f) \bowtie (\rho_{p_i/p}s \bowtie g))$$

For example, restricted join $t_{1,6} \triangleq t_{1,3} \overset{f^o_3, f^i_4}{\bowtie} t_{4,6}$ expresses transformation performed between interfaces i_1 and i_6. Transformation $t_{1,6}$ is a compound transformation but has the same shape as atomic transformations. It can be used directly in queries or to define another compound transformation. There are several options to define compound transformation.

The presented compound transformation is defined between interface locations. It is useful and in queries often required to have compound transformation between hosts in a network. To define such transformation we use identity transformation, $\texttt{transform}(\iota_t, p, p)$, and permit-all filter, $\texttt{filter}(\iota_f, p)$. For instance, we can define transformation, $t^{r_1}_{a,b}$, between subnetworks a and b, which expresses how router r_1 forwards and filters the packets as follows:

$$t^{r_1}_{a,b} \stackrel{\Delta}{=} \iota_t \overset{\iota_f, f_1}{\bowtie} t_{1,3} \overset{g_3, \iota_f}{\bowtie} \iota_t$$

Network in Fig.3.1 has two possible paths connecting location a to location c. Transformations $t^1_{a,c}$ for path via r_2 and $t^2_{a,c}$ for path via r_3 can be computed. Extending the transformation to location d means to join $t^1_{a,c}$ and $t^2_{a,c}$ transformations with $t_{8,9} \overset{g_9, \iota_f}{\bowtie} \iota_t$ yielding $t^1_{a,d}$ and $t^1_{a,d}$, respectively. It is also possible to computed approximations of $t_{a,c}$ from different combinations of $t^1_{a,c}$ and $t^2_{a,c}$. Following idea introduced in [90], an upper bound approximation is $t^{\cup}_{a,c} \stackrel{\Delta}{=} t^1_{a,c} \cup t^2_{a,c}$, and a lower bound approximation is $t^{\cap}_{a,c} \stackrel{\Delta}{=} t^1_{a,c} \cup t^2_{a,c}$. Extending these approximations to reachability relations from a to d is by computing $t^{\cap}_{a,d} \stackrel{\Delta}{=} t^{\cap}_{a,c} \overset{\iota_f, \iota_f}{\bowtie} t_{c,d}$ and $t^{\cup}_{a,d} \stackrel{\Delta}{=} t^{\cup}_{a,c} \overset{\iota_f, \iota_f}{\bowtie} t_{c,d}$, respectively

Two versions of the reachability relation, $t^{\cap}_{i,j}$ and $t^{\cup}_{i,j}$, give us the lower and upper bounds on packet reachability between nodes i and j, respectively. The *lower bound* $t^{\cap}_{i,j}$ is a set of (p, q), such that if p originates at location i then *any path* that connects i to j is able to deliver packet q to location j. The *upper bound* $t^{\cup}_{i,j}$ is a set of (p, q), such that if p originates at location i then *some path* that connects i to j is able to deliver packet q to location j. It is possible to combine transformations arbitrary to analyze reachability within these bounds.

3.3.2 *Reachability Analysis*

A defined restricted join operation is implemented in REACH rule. In addition, PATH rule is implemented to compute paths between pairs of locations.

- PATH(m, n, p) asserts that p is a path from location m to location n. It is possible to use any path enumeration algorithm for obtaining a set of paths in the network between two specified locations.
- REACH(p, f, g) is an input to output packet flow relation computed for path p. This operation is defined as compound transformation computed along a single path p. Fig. 3.7 contains FORMULA implementation of this relation.

Using these two rules, it is possible to perform an analysis of network requirements. The analysis method follows the idea, that if one wants to proof a given reachability property she tries to find its violation. If no violation is found then, it is possible to assert that the property is satisfied. In a case of property violation, information consisting of a path and flow sets that violate the property is available. The spirit of the method is demonstrated on following examples:

- Each web traffic from a local network must be routed through a transparent proxy server `proxy`. Verifying that this statement is satisfied by a network model, a query trying to find its violation is executed:

$$\begin{aligned} \texttt{violate}(P_1, p, f, g) \Leftarrow\ & \text{PATH}(\texttt{local}, \texttt{inet}, p), \texttt{proxy} \notin p \\ & \text{REACH}(p, f, g) \\ & f[\texttt{dstIp}] \in \texttt{inet}, f[\texttt{dstPn}] = 80 \end{aligned}$$

  When the answer is `false`, then the requirement is satisfied. Otherwise, a set of paths p together with input and output flow pairs (f, g) are provided in a proof tree. Such result thus also contains diagnostic information that can help to reveal a configuration error.

```
domain Reach includes Topology, Flow {
    Filter ::= new (e:Edge, fp:Packet).
    Transform ::= new (i:Idge, inp:Packet, outr:Packet).
    Path ::= new (Edge+Idge, any Path + {NIL}).

    reach ::= (p:Path,f:flow,g:flow).

    reach(Path(edge,NIL),flw,flw) :-
        edge is Edge,
        Filter(edge,pck),
        packet_to_flow(pck,flw).

    reach(Path(idge,NIL),fin,fout) :-
        idge is Idge,
        Transform(idge,pin,pout),
        packet_to_flow(pin,fin),
        packet_to_flow(pout,fout).

    reach(Path(edge,path),fin,fout) :-
        edge is Edge,
        Filter(edge,pck),
        packet_to_flow(pck,fin),
        reach(path,fin,fout).

    reach(Path(idge,NIL),fin,fout) :-
        idge is Idge,
        Transform(idge,pin,pout),
        packet_to_flow(pin,fin),
        packet_to_flow(pout,fmid),
        reach(path,fmid,fout).
}
```

Fig. 3.7 A `Reach` domain implementation

Reach domain contains definitions of `Filter`, `Transform` and `Path` relations. Reachability evaluation is based on `reach` rule. `reach` computes an overall transform relation between input and output flows for the given path. A recursive rule implementation consists of four cases. Note that while filter and transform relations are defined on packet objects, `reach` computes a relation between flow objects.

- Allowed traffic to management interfaces of all intermediate devices must arise from administrators' computers and must be secured. Finding violations of this policy means either to find a location other than `admins` or ii) to identify a management connection that is not secure.

$$
\begin{aligned}
\texttt{violate}(P_2, p, f, g) \Leftarrow\ & \textsc{path}(n, \texttt{infstr}, p), \\
& \textsc{reach}(p, f, g), p \notin \texttt{admins} \\
& g[\texttt{dst.ip}] \in \texttt{infstr}, \\
& f[\texttt{src.ip}] \notin \texttt{admins} \\
\texttt{violate}(P_2, p, f, g) \Leftarrow\ & \textsc{path}(\texttt{admins}, \texttt{infstr}, p) \\
& \textsc{reach}(p, f, g) \\
& g[\texttt{dst.ip}] \in \texttt{infstr}, \\
& f[\texttt{src.ip}] \in \texttt{admins} \\
& g[\texttt{dst.pn}] \neq \texttt{ssh}
\end{aligned}
$$

By evaluating these two statements we get as many answers as there are violations of the property.

From both presented examples, it can be seen that there are more than one possible representation of a network requirement. Also, the amount of information in a requirement specification can differ depending on the precise intention of a user. The presented formalization and analysis method enables to expert query network reachability model to validate requirements.

3.4 Chapter Summary

Simple network reachability model was defined in this chapter. The purpose of the network reachability model is to facilitate a computation of flows between any pair of network locations. Presented reachability model provides a notion of end-to-end flows as a constraint relation between input and output flows along a specified path. Regardless its simplicity, this model is capable of interpreting meaning of many network concepts, e.g., access control, network address translation, quality of service marking and forwarding, or traffic tunneling.

It was also shown that Formula system can efficiently implement reachability calculation. The simplicity of the network reachability model expressed by constraint relations greatly simplifies reasoning about reachability properties. The reasoning can be performed directly in Formula environment.

The presented model and analytical method can be used for analysis of network behavior either at an abstract level during network design or as a part of network configuration analysis for validation of reachability properties. The method can determine for every path a reachability for any given packet definition. Based on this, it is, for instance, possible:

- to perform security policy verification as proposed by Gutmann [25],
- to verify the access control based security implementations in a similar way as done by Bera, Ghosh and Dasgupta in [12], and
- to compute reachability estimations as shown by Xie et al. in [90].

In general, computing upper and lower bound of flow reachability enables to verify a security policy implementation. Further refinement that includes routing information would allow us to find network states in which security policy is violated. Network designers thus have precise information about the issue that aids them to detect sources of the problem. Integration of routing model to the analysis process is presented in Chapter 6.

Chapter 4
Network Policy Specification

Every network is designed to meet certain functionality and security requirements. The purpose of network policy specification is to describe requirements that a network should meet. A simple view of a network policy is that it constraints network communication according to specified rules. The purpose is to allow a network administrator to manage intended relations between clients of the network services and network devices providing access to these services. This chapter deals with an overview of existing network policy languages and the design of a new language that can be used for specification of flexible network policies that can be analyzed by constraint solvers. The goal is to create a new network policy language supporting network design and control. Formula tool is employed to capture formal semantics of the network policy language as well as to provide a tool implementing consistency checking of specifications.

A policy specification is an aggregation of policy rules, each of which consists of condition statements and corresponding actions. The purpose of a network management and security policy is to constrain communication so that the policy meets the administrator intention. Network policy thus dictates the availability of services provided by servers and consumed by clients. It may be assumed that network policies defined at service level constrain data flows by classifying then into different service groups.

A network policy can be either used for specification of requirements on a network or as a program for network management systems. In the former case, network policy specification defines what is expected network functionality, and network devices need to be configured to meet this expectations. Traditional networking understands a role of network policy specification as a description of expected network behavior. Often such policy is given using informal language. Network administrators deploy network and configure device based on this (informal) network policy specification. In the latter case, a policy can be viewed as a program that is used by a policy engine to control the behavior of a network. Recently, software-defined networking become a new trend in networking. In SDN environment, policy-based network management is an option as SDN offers greater possibility to control network behavior by separating and moving control plane out of a network device.

The concept of real-time policy-based network management systems does not emerged from SDN paradigm. Convergence of various network application imposes additional requirements on network management. Policy Core Information Model were jointly developed by IETF and DMTF. It defines policy-driven management approach that assume centralized repository storing policies and distributed components implementing these policies. Policy Decision Point (PDP) and Policy

Enforcement Point (PEP) is a component that can be a part of the network device that applies the policy to the network traffic. If PEP needs to decide what action to apply to a new traffic, it asks PDP, which makes a decision on a proper action based on the network state and policies from the policy repository. Existing network management protocols are used in this architecture, e.g., LDAP for accessing policy repository, SNMP for communication between PEP and PDP. Policy rules have the form of conditional statements:

$$\text{IF } condition_1 \ldots \text{ AND } condition_n \text{ THEN } action_1 \ldots \text{ AND } action_n$$

Policy is an aggregation if policy rules. Priority of policies and rules is defined to resolve possible conflicts. For expression policy statements, Policy Framework Definition Language was proposed. It provides mapping of network service requirements to vendor independent policy specifications.

An overview of network policy specification language is provided by Stone, Lundy and Xie [77] and more recently by Hand and Lei [29]. An approach to integrating policy controls into the Internet was proposed by Clark [16]. He defined template called a policy term. This template considers that the Internet is split into administrative regions (AR) that contain devices, links, and networks. A policy term is represented as following tuple:

$$((H_s, AR_s, AR_{ent}), (H_d, AR_d, AR_{exit}), USI, C_g),$$

where

- (H_s, AR_s, AR_{ent}) represent host's source address, source AR and entry AR (previous hop),
- (H_d, AR_d, AR_{exit}) represent host's destination address, destination AR and exit AR (last hop),
- UCI represents the user class id (traffic class) and C_g are global conditions.

For example, the following policy term declares that all voip traffic from AR 14 can flow to network 14.229.8.* within AR 20.

```
((*,14,*)(14.229.8.*,20,), voip, *).
```

Guttman [25] introduced a simple language for expressing global network access policies. A policy consists of a set of statements that have the form of if-sentence:

> If packet p was in previously in zone A and later reaches zone B then it satisfies predicate $\phi(p)$.

The following sentence is an example of Guttman's policy statement:

> If p was ever in the external area and later reaches the engineering area, then p should be an SMTP packet with its destination the mail host.

Guttman introduced formal language having LISP-like syntax for capturing policy statements and devised an algorithm that computes access control lists for individual routers in a network that obeys specified network access policy. Hinrichs [32] presents a language for policy expressions in a form of guarded commands. Policy consists of a condition that selects network flows to which an action is applied. Policy actions are requirements associated with network flows and can include filtering, cryptographic requirements or quality of service requirements. The language has if-then-else structure. Guttman and Herzog [26] use notion of trajectories that record zones the flows visited between source and destination. Specified policies impose constraints on areas that may be visited.

For instance, it is required that if a packet arrives in zone A then it must not visit area B. Security policy specification proposed by Ou et al. [68] is defined as a set of triples expressing allowed actions that a client can perform on a server. Hinrichs et al. [34] introduce Flow-based Management Language (FML), which specifies permitted flows on a network. This language employs non-recursive DATALOG. Policy statements have the form of predicates defining properties of flows. Bera et al. [10] specifies security policy as tuples consisting of service, source zone, destination zone, time constraint and associated action.

4.1 Policy Languages

Major policy languages designed and published in the current literature that are related to the policy language developed in presented thesis is summarized in this section. These languages aim at abstract representation of security and network management systems. They differ in the scope of types of policies supported and level of abstraction employed for the description of real-world concepts managed. In the following text, each language is described using illustrative examples to grasp the character of the language itself and its applicability.

4.1.1 Policy Description Language

The Policy Description Language (PDL) is an event-based language proposed for real-time policy-based network management [57]. Policies applied to network management are used to specify configuration parameters, handle faulting states, ensure agreed level of performance, provide required security and handle accounting. Policies are functions that map actual network states to adequate predefined actions.

The PDL uses the *event-condition-action* paradigm for rules. A rule is triggered when specified event occurs, and the condition is satisfied. Policy rule is a proposition that has the following form:

event CAUSES *action* IF *condition*

It is thus possible to trigger a rule if a specified sequence of events occurs in a system. Events compose event streams. A stream separates events according time of their occurrence into intervals called epochs. Events in the same epoch are considered to occur simultaneously. Epochs are linearly ordered so that it is possible to relate any pair of events. They are either simultaneous, if they are in the same epoch or event e_1 precedes e_2 if e_1 is from an epoch that precedes the other epoch where e_2 occurs.

An action is an expression that consists of action symbol and its parameters. A condition is an expression that consists of predicates and logical operators necessary to express propositions in a system domain.

```
1    const: R_over, R_ok
2    events: normal_mode, restricted_mode, call_made, time_out
3    actions: restrict_calls, accept_all_calls
4    policy: normal_mode, ^(call_made | time_out )
5            triggers restricted_mode
6            if count(time_out) > R_over * count(call_made).
7            restricted_mode causes restrict_calls.
8            restricted_mode, ^(call_made | time_out)
9            triggers normal_mode
10           if count(time_out) < R_ok * count(call_made).
11           normal_mode causes accept_all_calls.
```

Fig. 4.1 An example of PDL Policy (from [57])
This specification consists of two policy rules that define actions for normal (line 11) and restricted (line 07) modes. Switching between these modes is defined using *policy defined events*. Switching from normal to restricted mode is at lines 04-06. Switching back to normal mode is at lines 08-10.

Except primitive events, which are defined using event symbols from a system or environment domain, the language also defines complex events called policy defined events, which have the form of:

```
event triggers pde(m1=t1,...,mk=tk)
      if condition
```

where *pde* is a policy defined primitive event symbol and $m_1, \ldots, m_k$ are policy attributes and $t_1, \ldots, t_k$ are terms assigned to policy attributes. The intuitive meaning of a policy defined events is that if an event occurs in the current epoch and condition is met then in the next epoch an instance of *pde* will occur.

Figure 4.1 shows an example of s simple policy that controls calls in a communication network depending on its current load. This example also demonstrates the use of aggregate operator Count. Other aggregate operators include Sum, Avg, Min, Max, etc. These operators are defined over streams of events.

The PDL has a formal semantics. Policies are interpreted over event streams. A policy defined event proposition, E **triggers** e **if** c, is satisfied if there is a satisfying trace found in event stream history. Similarly, E causes a if c can be established. Authors implemented PDL algorithm into policy server, which was being used for central administration of packet telephony networks. Evaluation of the algorithm shows that although the decision of the policy evaluation problem is NP-hard but for most of the practical policies the efficient evaluation exists. Further detail son semantics and complexity can be found in [57].

4.1.2 Ponder

The Ponder language provides and abstract specification of security policies and their mapping to access control mechanisms implemented in firewalls, operating systems or databases [18]. The

Ponder language is a declarative, object-oriented language intended to specify security and management policy for distributed systems. Following requirements drove the design of Ponder language:

- providing flexible security policies supporting delegation of access rights,
- scaling to large systems by managing policies for collection of objects rather individual ones,
- creating composite policies from basic security and management policies,
- precise semantics of policy models enabling for conflicts and inconsistencies checking, and
- extensibility and easy to use to provide a robust tool for network policy administrators.

Essential ingredients of the language comprise of access control, obligations, constraint expressions and policy composition mechanisms.

Access control policies represent basic and most widely applied security mechanism that limits activity of authenticated users in a system. Group of access control policies represent the following types of policies:

Access control policies represent basic and most widely applied security mechanism that limits activity of authenticated users in a system. Group of access control policies represent the following types of policies:

- *Authorization policies* that perform on specified target objects. While existence both positive and negative authorization policies may be a source of conflicts, and its complicates the enforcement of these rules, they increase flexibility because administrator can describe an authorization in the very natural way. Moreover, negative authorization policies can be used to represent (temporal) exceptions to the general system. Potential conflicts can be detected using static analysis. An example of a positive authorization is in Figure 4.2.
- *Information filtering policies* that aids in refining actions in authorization policies. Each filtering policy is associated with an action. The filter contains a condition determining when this filter is applied to the associated action. Using filters, it is possible to set the specific properties of the action based on input or output parameters of the action, attributes of the subject or target or system settings. An example of information filter policy is in Figure 4.3.
- *Delegation policies* that significantly increase flexibility of access control system enabling a user to the temporary transfer access rights to other subjects. The delegation has to be bound by authorization mechanism. Delegation permits subjects to transfer privileges to grantees that perform an action on their behalf. Positive delegation policies express the access rights that can be delegated. It is possible to limit rights that cannot be delegated using negative delegation policies. An example of delegation policy is in Figure 4.4.
- *Refrain policies* that define actions that must not be executed on targets objects. These policies are used when targets are not trusted to enforce authorization policies. Refrain policies are contrary to negative authorization policies imposed by subjects. An example of the refrain policy is in Figure 4.5. This example demonstrates that access is determined by the state related to the subject rather than character of the target object.

Obligation policies are used to define actions that must be performed when certain events occur in a system. An obligation policy specifies what action must a subject do on target object when a specified event occurs. Figure 4.6 shows an example of obligation policy, which requires that if there are three consecutive login fail events for user id then security administrator from NRegion/SecAdmin will

```
inst auth+ switchPolicyOps {
  subject /NetworkAdmin;
  target <PolicyT> /Nregion/switches;
  action load(), remove(), enable(), disable() ;
}
```

Fig. 4.2 An example of Positive authorization policy (from [18])
Members of NetworkAdmin group can execute listed function in the target domain. These functions belongs to PolicyT object

disable user account user id and generate and write log item using log() operation of the associated administrator object.

Policy constraints represent sets of conditions that further limit situations when the policy can be applied. There are several categories of policy constraints:

- *Basic Policy Constraints* have form of predicates over attributes of a policy, system, and an environment. Predicate must be evaluated to true for the policy to apply. Constraints are either state or time-based depending on attributes referenced. Separation of different types of constraints simplifies their handling by the policy compiler and when performing consistency checking. Figure 4.7 shows an example of policy with two fundamental constraints. It states that only manager can set up a video conference during the specified period. A collection of basic policies composes a complex policy.
- *Meta-Policies* specify policies that apply to basic or complex policies in the given scope. They are used to constrain the application of policies in the system by defining predicates over system-wide attributes. Meta-policies are expressed using OCL expressions.

Policy composition is a mechanism to build comprehensive policy-based systems. Composition simplifies the task of policy system management by grouping related policies and structuring them according organizational structure. Policy composition defines the following constructs:

- Groups are used to package related policies. Groups have assigned names that enable their reuse. Also, meta-policies may be associated with a group enforcing additional constraints within the scope of the group.
- Roles provide grouping of policies that applies to the same subject. Assigning policies to roles instead of concrete subjects, it is possible to manage subjects in a policy system efficiently. Each

```
inst auth+ filter1 {
  subject /Agroup + /Bgroup ;
  target USAStaff - NYgroup ;
  action VideoConf(BW,Priority)
         { in BW=2 ; in Priority=3 ; }
         if (time.after("1900")) {in BW=3; in Priority = 1; }
}
```

Fig. 4.3 An example of Information filter policy (from [18])
Members of AGroup and BGroup can set up a video conference with members of USAStaff except NYgroup. This vide conference use default parameters on bandwidth and priority until 7pm when these parameters changes to the specified.

```
inst deleg+ (switchPolicyOps) delegSwitchOps {
  grantee /DomainAdmin ;
  target /Nregion/switches/typeA ;
  action enable(), disable();
  valid time.duration(24) ;
}
```

Fig. 4.4 An example of Delegation policy (from [18])
This policy is associated with switchPolicyOps authorization from (a). It states that members of NetworkAdmin can delegate access to actions enable() and disable() on target objects from domain Nregion/switches/typeA to members of DomainAdmin. Note that target is restricted here to typeA objects only

role definition is a set of authorization, obligation, refrain and delegation policies that have the role as their subject.

- Inheritance enables specialization of roles. A new role may be derived from an existing role by extending it with new elements and by overriding existing elements with new content. In this manner, it is possible to create a hierarchy of roles without introducing necessary redundancy in policy specifications.
- Relationship provides a way for defining policies not directly associated with any role. A relationship associates policies that declare authorizations and obligations to relationship between roles. For example, this may include an obligation policy expressing that the project manager must provide a report to division manager every Monday.

The Ponder language provides construct for specifying policies for management and security of distributed systems. Policies can be declared in a form of authorizations, filters, delegations and obligations. In addition, composite policies enable to develop large scale policy-based management systems. Implementation of the PONDER Policy Based Management Toolkit consists of Policy Editor and Management Console Tool for creating and dynamically managing policies.

4.1.3 Path-Based Policy Language

Stone in [78] defines Path-Based Policy Language (PPL) which enables to create flexible policies targeting path-based traffic flows. Possibility to specify a path as the part of policy constraints offers a greater control over traffic flows that can be useful, e.g., for QoS policies. The PPL aims

```
inst refrain testingRes {
  subject s=/test-engineers ;
  target /analysts + /developers ;
  action discloseTestResults();
  when s.testing_sequence = "in-progress" ;
}
```

Fig. 4.5 An example of Refrain policy (from [18])
This policy states that disclosing test results to analysts and developers is forbidden by test engineers while testing sequence is in progress.

```
inst oblig loginFailure {
  on               3*loginfail(userid) ;
  subject          s = /NRegion/SecAdmin ;
  target  <userT>  t = /NRegion/users ^ {userid} ;
  do               t.disable() -> s.log(userid) ;
}
```

Fig. 4.6 An example of Obligation policy (from [18])

at unambiguous representation based on well-defined semantics that enables to detect potential conflicts in policies. To detect conflicts, policies are translated into formal logic, and a theorem prover is utilized. An intended application of PPL is capture abstract policies that can be used to generate a set of concrete rules for network devices. An abstract policy rule expression has following format:

```
policyId<userId>@{paths} {target} {conditions} {actions},
```

where:

- policyId is a unique policy identifier that can be used when referring to the policy,
- userId is identification of the creator of policy and can be used for resolving conflicts,
- paths represent a collection of network paths to which the policy is constrained,
- target determines a set of network flows to which the policy is applied,
- conditions represent a rule's postcondition in form any valid logical propositions, and
- actions consist of a collection of action items having form of condition:action. Each action is a statement that can set parameters, declare compromises or explicitly deny or accept the traffic. Action items can have optional preconditions that must be satisfied to execute the action.

The semantics of the policy *PolicyId* imposed by *userId* requires that traffic target may only use specified *paths* if *conditions* are satisfied after *actions* are executed. Next example defines a policy expressing that all data traffic in a network has to be marked with priority 10.

```
P1<admin> @ {*}{traffic-class=data} {*} {priority:=10}
```

While target specification, conditions, and actions are typical of most of the existing policy languages, the path component introduces a new way to refine constraints on data communication. For example, it is possible to declare that all accounting traffic will be lowered to priority five on all paths from node r_1 to node r_5 unless it is Friday:

```
inst auth+ filter2 {
  subject s = EUStaff
  action VideoConf(bw,prio);
  target USAStaff
  when {
    s.role = "manager";
    time.between("16:00","18:00");
  }
}
```

Fig. 4.7 An example of policy constraints (from [18])

```
P2<Bob> @ {<r1,*,r5>}{traffic-class={accounting}} {}
          {day!=Friday : priority:=5}
```

PPL provides various level of granularity for policy specification. In the previous example, the whole class of traffic was limited. The following policy restricts a traffic of the single user:

```
P3<admin> @ {<r1,*,r5>}{traffic-class={student}} {}
            {user-id="Gary" : deny}
```

Authors of PPL sees the natural representation for integrated services as one of the benefits of path-based policies. Also, path-based approach can simplify the problem of allocating resources for new requests that involve a combination of services constraints, e.g., acceptable delay and required throughput, which is essential an NP-complete problem. If these demands were specified using path-based QoS policies, it might be possible to analyze these requirements in static time rather than run-time.

4.1.4 Flow-based Security Language

Flow-based Security Language (FML) proposed by Hinrichs et al. [34] is a declarative policy language for managing configurations of enterprise networks. It employs features representing various mechanisms including access control lists, virtual local networks, network address translation, policy routing and admission control. Design of FML was driven by the aim of providing a simple language with efficient implementation suitable for management of large networks.

FML specifies policies with respect to network flows. An FML policy is a set of *if-then* statements. In essence, the system of FML is non-recursive DATALOG with negations. An FML rule in the context G has the following form:

$$h \Leftarrow [\neg]b_1 \wedge \cdots \wedge [\neg]b_n$$

where h is head of the rule that can be one of the predefined predicates for access control, quality of services, and address translation. Every b_i is atom in a body of the rule, which is used to express constraints on the state. A simple example of FML policy is shown bellow. It says that super users have no restriction in communication.

$$\textsc{allow}(U_s, H_s, A_s, U_t, H_t, A_t, P, R) \Leftarrow \textsc{superuser}(U_s)$$

Arguments of ALLOW rules are variables that refer to a user, host, access points addresses, protocol and indication if the flow is a request or response. Using these arguments, administrator can describe a flow to which a rule applies. In the case of presented example, there are no other constraints on flows except that the flows are created by a superuser.

Order of FML rules is irrelevant, which simplifies combination of policies. On the other hand, an explicit rule priority scheme needs to be defined to resolve potential conflicts. Conflict resolution for FML employs priority system that depends on the type of rules. For example, *deny* rules have

P_3 :
  ALLOW($\bar{F}$) $\Leftarrow$ $arp \vee Pt = dhcp$
  ALLOW($\bar{F}$) $\Leftarrow$ $H_t = authSvr \wedge Pt = http$
  ALLOW($\bar{F}$) $\Leftarrow$ $H_s = authSvr \wedge Pt = http$
P_2 :
  HTTPREDIRECT($\bar{F}, authSvr$) $\Leftarrow$ $U_s = unknown \wedge Pt = http$
P_1 :
  DENY($\bar{F}$) $\Leftarrow$ $U_s = unknown$

Fig. 4.8 An example of FML cascade specification (from [34])

the highest priority, *waypoint* and *avoid* have equal priority and take precedence over all *allow* rules. However to enable a definition of closed authorization policy[1], FML provides cascade construct. FML Cascade defines total ordering on a finite set of FML policies.

Hinrichs et al. [34] provides definition and examples for several networking mechanisms defined using FML:

- *Access Control* specifies whether flows are allowed on the network. It also can further constraint permitted routes. Head of a policy rule can contain one of the access control keywords, which include allow, deny, waypoint, avoid and rate-limit. Their names denote intended meaning. An example of access control policy rule follows:

  WAYPOINT($\bar{F}, ids$) $\Leftarrow$ GUEST(U_s) $\wedge$ WIRELESS(A_s).
  WIRELESS($wap1$).
  WIRELESS($wap2$).

  This rule dictates that guest communication must pass through ids device. Guest communication enters the network at one of the defined wireless access points *wap1* or *wap2*.
- *Quality of Service (QoS)* serves to specify how resources should be allocated to flow classes in terms of the relative importance of key QoS attributes: latency, jitter, and bandwidth. For example, the following rules defines QoS policy for VoIP traffic:

  LATENCY($\bar{F}$, 100) $\Leftarrow$ $Pt = voip$
  JITTER($\bar{F}$, 5) $\Leftarrow$ $Pt = voip$
  BAND($\bar{F}$, 3000) $\Leftarrow$ $Pt = voip$

- *Network address translation (NAT)* maps between two ranges of IP addresses. The NAT has to maintain a state, which requires that the policy engine contain a database of translated flows. Next example of the NAT policy consists of two rules. The first rule denotes that flows with private source address 10.10.10.1 are translated to use source public address 170.70.70.1. Argument *sw* refers to NAT device that connects a private network. Second statement contains fact that the destination has to match the original source.

[1] Closed policy is a policy where everything not explicitly allowed is denied.

$$\textsc{srcNat}(\bar{F}, 170.70.70.1, sw) \Leftarrow A_s = patio \wedge IP_s = 10.0.0.1$$
$$\textsc{unSrcNat}(\bar{F}) \Leftarrow A_d = sw \wedge IP_d = 170.70.70.1$$

- *Admission Control* serves for specifying authentication requirements for clients in order to gain access to network services. FML can represent admission control rules by defining default connectivity and authentication mechanisms that must be used to acquire privileged access. Figure 4.8 contains example representing FML Policy Cascade. Therein, policy P_3 defines default connectivity for authentication, policy P_2 adds possibility for redirection to a captive web portal, and P_1 denies all other traffic. FML Cascade defines priority chain $P_3 > P_2 > P_1$.

FML was implemented within NOX, which is a general control network platform. FML serves as the policy engine in NOX to manage newly created flows and to modify forwarding tables of NOX devices. During experiments, this implementation was successfully deployed to control two operational networks.

4.2 Network Policy Specification Language

Network and security policy languages cover many aspects of network management. They define access control, routing policies, security policies such as access control, admission control, or other specific security parameters. For instance, security policy can define a set of services that must be switched off at all network devices. Network Policy Specification Language (NPSL) is a language for defining network policies in a rigorous manner. The language enables to express routing, forwarding, security, and performance requirements. It offers the following constructs:

- *Flow-based policy* rules express constraints and actions on traffic flow in a network. They can be used for expressing routing policies, filter policies and access control.
- *Service-based policy* rules represent which services are enabled on a network and their security and performance parameters.
- *Composition rules* reveal how individual policy rules and policy groups are combined in the overall policy. Composition enables to resolve potential conflicts, scale system for large networks and reuse policy specification is necessary.

In the rest of this section, syntax and structure of policy rules are defined accompanied with illustrative examples.

4.2.1 Flow-based Policy Rules

In general, each flow-based policy rule expresses constraints that may be classified into the following groups:

1. *What* type of communication should be constrained, e.g., data, voice, video, management,
2. *Where/when* policy should be imposed, which defines network locations, path or zones, e.g., DMZ, gateways, or imposes some additional requirements, e.g., times span, triggered by some other event, conditional in a specified network state, and

3. *How* the policy is to be imposed by means of actions that represent the constraint, e.g., deny or permit action, QoS remarking.

These also correspond to a structure of a policy written in natural language such as English, where a subject imposes an action to an object at a specified place and time. Structure of a policy rule thus consists of flow specification that selects the targeted traffic enforced by the space and time constraints, and applicable action. Flow specification selects a group of flows that will be affected by the rule. Flows can be chosen according to their characteristics, namely, source and destination node or network, type of traffic, application name, specific user or a group of users. Space constraint restrains network locations, network zones as proposed by Guttman [25] or paths similarly to PPL by Stone [77]. Time constraints would either require existence of the standard time base or can be related to specific sources of time information. Time constraints often represent valid time intervals of the policy. An action can be permit, deny, or some other way of controlling the network behavior.

Specifying traffic type means to define a class of communication and communicating parties. In many network policy languages, it is often possible except specifying a service, to define user, security options, and other extended properties. The present approach can capture many other properties depending on the flow definition. In general, it is possible to constraint any aspect of flow that can be expressed in the form of constraints on a flow object.

Flow-based policy rules represent relations that associate flow objects with actions possibly under scope and additional constraints. A flow-based policy rule is defined as the direct compound of two constraints, namely, target flow and action item:

```
FlowRule ::= new (flow:Flow, space:SpaceScope, time:TimeScope,
                  action:FlowAction).
```

Flow constraint represents to which traffic class a policy should be applied. There are many possibilities to characterize traffic. In the present approach, traffic is expressed in terms of its source node, target node, and application protocol.

```
Flow ::= new (source:Netloc, target:Netloc, protocol: Protocol).
```

Source and target nodes can be specified in terms of zones or services. This is represented using type *Netloc*, which is defined as follows:

```
Netloc ::= Zone + Service.
```

Protocol is either a name of application protocol or an application category:

```
Protocol ::= Application + Category + {ANY}.
```

List of supported application protocols and application categories are according Cisco NBAR2 database. They represent a collection of constants, e.g.:

```
Application ::= { FLASHMYSPACE, FLASHVIDEO, FLASHYAHOO,
                  GOPHER, GSSHTTP, HTTP, ... }.
Category ::= { BROWSING, BUSINESS, EMAIL, FILESHARING, ... }.
```

Flow definitions can have an arbitrary level of aggregation. It is possible to designate a particular flow between two IP addresses and an application as well as to determine a group of flows between zones and a collection of applications of the particular category. For example,

```
Flow(Network("131.1.*.*", UI32Range(2197880832,2197946367)),
     anynet, BROWSING).
```

defines a flow from *Browsing* category (e.g. HTTP), originating in 131.1.0.0/16 network and without constraining target address, which is expressed as anynet object:

```
anynet is Network("*", UI32Range(0,4294967295)).
```

that represents a network often denoted as 0.0.0.0/0. In examples, network constructor use form of `[A.B.C.D]` instead of `Network("A.B.C.D", UI32Range(X,Y))`.

The aim of action part of policy definition is to express how the target flow should be handled in a network. There may be many possible ways to handle network traffic. Two most common kinds of actions are access control and traffic prioritization. It is achieved by defining FlowAction type in the form

```
FlowAction ::= Access + Priority + Qos + Priority
```

where

- *Access* defines Permit, Restrict and Deny actions. Permit or Deny actions state that given flow is allowed or denied on the specified path and during specified time. Restrict action represents and obligation that flow is only permitted if it takes the specified path and occurs during the given time interval.
- *Priority* enables to express an importance of the traffic using one of the defined values.
- *Qos* is a triple of parameters that define bounds on latency, jitter and bandwidth. For example,

  ```
  FlowRule(voip, ANY, ANY, Qos(Ms(200),Ms(10),Kbps(30))).
  ```

  represents a policy rule that defines QoS parameters for VoIP traffic. Latency is required to be less than 200ms, jitter must be bound within 10ms and required bandwidth is 30Kbps.

Presented policy definition can directly only pair constraints on target flows and associated actions. Additional constraints can be specified to provide scope of the policy and impose other conditions. Scope constraints defined by

```
Scope ::= new (space:SpaceScope,time:TimeScope).
```

assign space or time constraints to existing policy. The scope in path-based policy language is given as a set of paths conforming to specified criteria. A path can be given explicitly by enumerating all nodes between source and destination or by using a waypoint and avoid operations. Similarly to Stone's PPL the space scope can be defined in terms of full or partial path definition, which is accomplished by following statements:

```
Path ::= new (Netloc + {ANY}, any Path + {NIL}).
Waypoint ::= new (loc:Netloc).
Avoid ::= new (loc:Netloc).
SpaceScope ::= Path + Waypoint + Avoid.
```

Path constraint enables to specify a path as a sequence of network locations. Each node in a path may be either `Netloc` object or `ANY`, which stands for a wild character known from PPL paths. For example,

```
Path(Zone("Public"),Path(ANY,Path(Zone("Servers"))))
```

matches any path from Public zone to Servers zone with any number of nodes between these two end zones. In PPL syntax, this would be represented as `{Public,*,Servers}`. *Waypoint* constraint expresses that a path must contain specified node while Avoid constraint is satisfied for paths that do not contain the given node. Time scope constraint enables to specify an interval when a policy rule is valid. It is captured by *TimeScope*, and related relations defined as

```
Between ::= new (from:DateTime,until:DateTime).
Except ::= new (from:DateTime,until:DateTime).
TimeScope ::= Between + Except.
```

Type `DateTime` enables to represent various time information. For example,

```
Between(TimeOfDay(16,00),TimeOfDay(08,00))
```

represents a time interval from 16:00 to 08:00. Time scope is not limited only to the presented constructors. It may be possible to extend the domain with other time constraint constructors, for example

```
Day ::= { MON, THU, WED, THU, FRI, SAT, SUN }.
BetweenOnDay ::= new (from:DateTime,until:DateTime,day:Day).
```

offers possibility to express also day when a policy is only applied. Formal semantics of policy rules is discussed in section 4.4.1. Here, an informal overview is provided using an illustrative example. Policy rule

```
flow131_1 is Flow([131.1.*.*],anynet, ANY).
sc1 is SpaceScope(Waypoint(Zone("InternetEdge"))).
sc2 is SpaceScope(Waypoint(Zone("PublicServices"))).
tm  is TimeScope(Between(TimeOfDay(08,00),TimeOfDay(16,00))).
p4_1 is FlowRule(flow131_1, sc1, tm, Access(PERMIT)).
p4_2 is FlowRule(flow131_1, sc2, tm, Access(PERMIT)).
```

asserts that all traffic from 131.1 network is permitted between 08:00 to 16:00 if it passes *Internet-Edge* zone or *PublicServices* zone.

4.2.2 Service-based Policy Rules

Service based policy rules are not associated with any class of flows but rather with targets that are represented by sets of services. For example, a policy rule can require that an individual service is enabled and have some predefined parameters. Service based policy rules a defined in ServicePolicy domain:

```
domain ServicePolicy {
    Service ::= new (sid:ID).
```

```
    User ::= new (uid:ID).
    Group ::= new (gid:ID).
    Mode ::= new (g:User+Group,m:Mode).
    ServiceAction ::= Mode + Access.
    ServiceRule ::= new(s:Service, p:Scope, a:SrvAction).
}
```

Following policy rule controls accessing a web service in the public service zone. It admits anybody to read web pages and enables user Joe also to write

```
web is Service("Web Service").
public is Zone("PublicServices").
joe is User("Joe").
anybody is Group("Anybody").
sweb_1 is ServiceRule(web,public,Mode(anybody,READ)).
sweb_2 is ServiceRule(web,public,Mode(joe,WRITE)).
```

4.3 Examples

Examples of various network policies are presented in this section. These policies are split into different categories depending on their characteristic and purpose. All these policies are network-wide policies. Thus, they may be seen as global requirements on network functionality. Examples presented in this section are described in the context of a network design that roughly follows Cisco SAFE architecture guidelines for Small Enterprise Networks. High-level topology of the network structure is shown in Figure 4.9. The security design focuses on network foundation protection, Internet perimeter protection, data center protection, network security and control and secure mobility. In examples, the following set of facts is expected.

```
z_enterprise is Zone("Enterprise").
z_management is Zone("Management").
z_access is Zone("Access").
z_data is Zone("Data Center").
z_core is Zone("Core").
z_edge is Zone("Internet Edge").
z_public is Zone("Public Services").
z_internet is Zone("Internet").
z_branch is Zone("Branch").
z_partner is Zone("Partner Site").
z_tworker is Zone("Teleworker").

s_email is Service("Mail Service").
s_web is Service("Web Service").
s_esa is Service("Email Security Appliance").
s_wsa is Service("Web Security Appliance").
```

Zones split an enterprise network to different locations depending on their intended role. There are two special zones, namely, Enterprise and Internet. The first represent the whole corporate

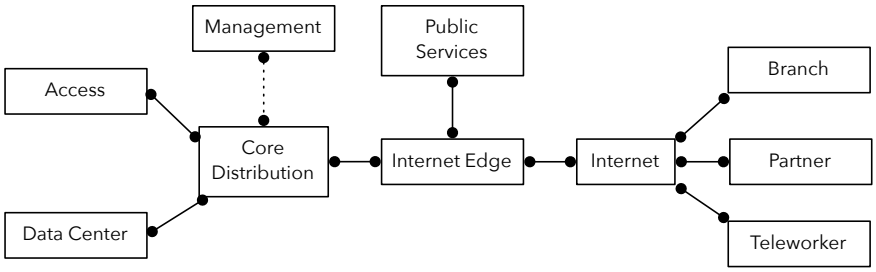


Fig. 4.9 Zone level diagram of Cisco SAFE network architecture

network and other zones, except Internet, Branch, Partner Site, and Teleworker are sub-zones of this zone. Internet zone represents any outside location to enterprise zone and is used when referring any Internet network or service. Services listed are only examples of all possible services that can be deployed in the network.

4.3.1 Availability

Availability policies define what services are accessible in a network. They also express network service availability and network resilience requirements, for instance, high availability design based on implementing routed access layer (see Figure 4.10). The role of access block is for policy enforcement and access control, route aggregation and an entry point for user populated access networks.

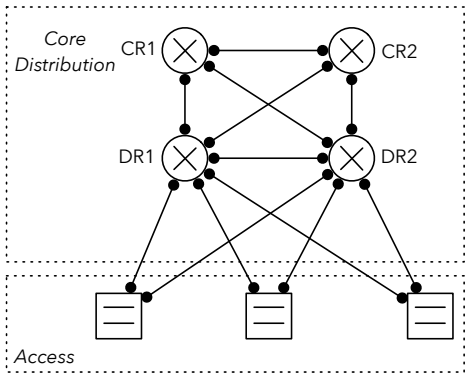


Fig. 4.10 Traditional design of Distribution Core zone

The core distribution block provides for high capacity transport between the connected access and data center blocks. Core layer employing routing design provides the necessary scalability, load sharing, fast convergence, and high-speed data forwarding. Full-mesh interconnection is provided between access and core blocks to achieve these parameters. High availability can be represented by the following policies:

- All access networks have redundant connectivity to enterprise services and internet. This policy enables flow on a path where core distribution device is avoided. Network topology has to contain redundant devices and interconnections to accommodate this. To assure that both of these rules can be satisfied an extra constraint called `ensure` is attached.

```
f is Flow(z_access, z_internet, ANY).
r_cr1 is Router("CR1").
r_cr2 is Router("CR2").
r1 is FlowRule(f, Avoid(r_cr1), ANY, Access(Permit)).
r2 is FlowRule(f, Avoid(r_cr2), ANY, Access(Permit)).
ensure(and(r1,r2)).
```

- Redundant paths are used for load balancing. Load balancing is a feature of forwarding service. The following expression requires that load balancing be enabled in a Distribution Core zone of the network for flows from access zones to the Internet:

```
f is Flow(z_access, z_internet, ANY).
s_fwd is Service("Forwarding").
LoadBalancing(s_fwd, f).
ServiceRule(s_fwd, z_core, ANY).
```

Network reachability policies can also impose additional constraints, such as required data paths for selected services. We consider email and web services as an example. To protect email communication, Email security appliance (ESA) is deployed in a network, for instance, as a part of Security Appliance device (see Figure X). To work properly, all email traffic should be inspected by ESA. To ensure this, the following rules are defined:

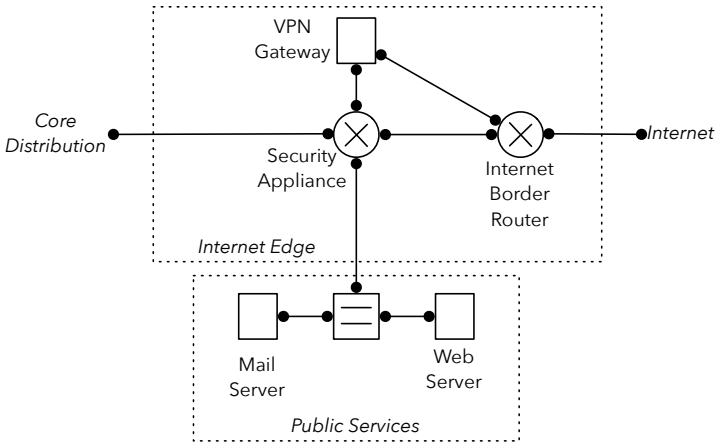


Fig. 4.11 A possible organization of Internet Edge Zone in SAFE

- All incoming SMTP traffic must be inspected by ESA. To enforce this, Restrict access permits SMTP flows if they go through ESA.

```
emailIn is Flow(z_internet, s_email,SMTP).
FlowRule(emailIn, Waypoint(s_esa),ANY,Access(Restrict)).
```

- Because ESA requires Internet access for updating its database, outbound HTTP and SSH should be enabled.

```
f_esa1 is Flow(s_esa, z_internet,HTTP).
f_esa2 is Flow(s_esa, z_internet, SSH).
FlowRule(f_esa1,Path(s_esa,z_edge,z_internet),ANY,Access(Permit)).
FlowRule(f_esa2,Path(s_esa,z_edge,z_internet),ANY,Access(Permit)).
```

Similarly, enterprise design can follow web security guidelines to block access to sites with non-business related content. Enforcing related policies require to route web traffic through Web Security Appliance (WSA) where URL filtering, reputation-based filtering, and malware detection is performed. Also, it may control peer-to-peer file sharing and selected internet applications, e.g., IM, BitTorrent, or Skype. WSA must inspect all outbound HTTP traffic. To enforce this, the policy rule only permits HTTP flows if going through WSA.

```
webOut is Flow(z_enterprise,z_internet,HTTP).
FlowRule(webOut, Waypoint(s_wsa),ANY,Access(Restrict)).
```

4.3.2 Access Control

According Cisco SAFE architecture guidelines, router providing Internet connectivity also represents the first line of defense against unauthorized access. Security appliance device within Internet Edge Zone (see Figure 4.11) implements other protecting service. The security appliance is configured to enforce following access rules:

- Deny any connection attempts originating from the Internet to internal resources and subnets.

```
FlowRule(Flow(z_internet, z_access,ANY), ANY, ANY, Access(Deny)).
FlowRule(Flow(z_internet, z_date,ANY), ANY, ANY, Access(Deny)).
FlowRule(Flow(z_internet, z_core,ANY), ANY, ANY, Access(Deny)).
```

- Allow outbound Internet access for users residing at any of the enterprise locations and for the protocols permitted by the organization's policies, e.g., HTTP and HTTPS.

```
FlowRule(Flow(z_access,z_internet,HTTP),ANY,ANY,Access(Permit)).
FlowRule(Flow(z_access,z_internet,HTTPS),ANY,ANY,Access(Permit)).
```

- Allow outbound Internet SSL access for administrative updates, SensorBase, IPS signature updates, etc.

```
FlowRule(Flow(z_access,z_internet,SSH),ANY,ANY,Access(Permit)).
FlowRule(Flow(z_data,z_internet,SSH),ANY,ANY,Access(Permit)).
```

- Allow users access to DMZ services such as company's website, E-mail, and domain name resolution (HTTP, SMTP, POP, IMAP, and DNS).

```
FlowRule(Flow(z_access, z_public,HTTP), ANY, ANY, Access(Permit)).
FlowRule(Flow(z_access, z_public,SMTP), ANY, ANY, Access(Permit)).
FlowRule(Flow(z_access, z_public,POP), ANY, ANY, Access(Permit)).
FlowRule(Flow(z_access, z_public,IMAP), ANY, ANY, Access(Permit)).
FlowRule(Flow(z_access, z_public,DNS), ANY, ANY, Access(Permit)).
```

- Restrict inbound Internet access to the DMZ for the necessary protocols and servers (HTTP to Web server, SMTP to the mail transfer agent, DNS to DNS server, etc.).

```
FlowRule(Flow(z_internet,z_public,HTTP),ANY,ANY,Access(Permit)).
FlowRule(Flow(z_internet,z_public,HTTPS),ANY,ANY,Access(Permit)).
FlowRule(Flow(z_internet,z_public,SMTP),ANY,ANY,Access(Permit)).
FlowRule(Flow(z_internet,z_public,DNS),ANY,ANY,Access(Permit)).
```

- Implement Network Address Translation (NAT) and Port Address Translation (PAT) to shield the internal address space from the Internet.

```
s_nat is Service("NAT").
NatService(s_nat, DYNAMIC, z_access, ip_public).
ServiceRule(s_nat, z_edge, ANY).
```

  The last rule is a service rule that defines Network Address Translation parameters. Here, it is used as access control rule because NAT also effectively blocks incoming traffic.

4.3.3 Quality of Service

Quality of service expresses required resource allocation to guarantee performance properties of network traffic. Quality of network communication is given in terms of bounds on delay, jitter, drop rate and bandwidth. To express these properties, NPSL offers `Qos(delay,jitter,bandwidth)` policy action. The meaning of this thing is to ensure that the flow is processed with the provided parameters. For example, enabling VoIP traffic from customers to the Internet and requiring QoS parameters to each of such flow is specified by the following policy:

```
gw   is Service("VoiceGateway").
voip is Flow(access, internet,RTP).
FlowRule(voip, Scope(Waypoint(gw), ANY, Qos(100,5,3000)).
```

In addition to these settings, this policy also requires that VoIP traffic pass through VoiceGateway device.

4.3.4 Security

A part of recommended security policies contains rules for device hardening. Device hardening means to disable all service that are not part of intended network functionality and can represent a potential threat.

- Only HTTPS and SSH can be used for management device access. Only devices with predefined IP addresses can be utilized for administrative access. It is related to the rule that requires the provision of a management interface for each network appliance. Thus, connections are only possible within management (virtual) zone.

```
f_mgmt1 is Flow(z_management, z_management, SSH).
f_mgmt2 is Flow(z_management, z_management, HTTPS).
FlowRule(f_mgmt1,Scope(Path(z_management)), ANY,Access(REQUIRE)).
FlowRule(f_mgmt2,Scope(Path(z_management)), ANY,Access(REQUIRE)).
```

- AAA service for role-based access control and logging should be employed. Use a local fallback account in case AAA server is unreachable.

```
s_radius  is Service("AAA").
s_local is Service("Local").
s_auth is Service("Authentication").
ServiceRule(s_auth, z_enterprise, ANY).
Authentication(s_auth).
AuthService(s_auth,RadiusAuth(s_radius)).
AuthFallback(s_auth,LocalAuth(s_local)).
```

- Use NTP to synchronize the time. The following rule specifies NTP service using "hierarchical mode" for whole network. The NTP master server is the Internet Border Router, which itself synchronizes with predefined public servers.

```
s_ntp   is Service("NTP").
Ntp(s_ntp, NTP_HIERARCHY).
NtpPublicServer(s_ntp, "0.europe.pool.ntp.org").
NtpPublicServer(s_ntp, "1.europe.pool.ntp.org").
NtpMasterServer(s_ntp, r_ibr).
ServiceRule(s_ntp, z_enterprise, ANY).
```

4.4 Properties of NPSL

Semantics of NPSL is first developed for flow-based policies and service-based policies separately and then linked together assuming relations between flow and service objects. Based on the presented semantics, consistency of policies can be defined, and a method for consistency analysis is designed. Network Policy Specification Language is defined as a collection of FORMULA domains that includes flow-based policy and service-based policy domains.

4.4.1 The Semantics

In previous sections, the informal semantics of NPSL was introduced using couple of examples. In this section, the formal semantics is developed. Because NPSL is embedded in a language of FORMULA, it is possible to provide the formal semantics using FORMULA as a metalanguage and defining a semantic mapping by conditional statements (FORMULA rules). Formal semantics of a

metalanguage, as well as the translation, inherits the semantics of Formula language. Intended meaning of the policy is to classify a traffic or service either as wanted or unwanted. If the traffic is permitted or denied depends on additional conditions, which includes constraints on a location, time or network state. Therefore, the meaning of a network policy can be interpreted with respect to network reachability. Under this interpretation, a policy represents a set of flows or services that are enabled in time, scope and state. Taking formal representation of the policy domain and a semantic domain represented by network reachability domain, semantic mapping can be defined. This mapping is built employing transformation feature of FORMULA system. A path-based reachability semantics for policy P with respect to network N is expressed as a function $r[P][N]$ defined as

$$r[P][N] : \mathcal{F} \rightarrow 2^{\mathcal{P}}$$

where

- $\mathcal{F}$ is a set of flows, and
- $\mathcal{P}$ is a set of all possible paths in network N.

If a flow is not permitted in a network then, an empty path set is assigned. The flow is defined as:

```
Flow ::= new (srcIp:IP,dstIp:IP,app:App,qos:Cos).
```

that is, a set of tuples consisting of source and destination addresses, application name and *class of service (Cos)* that can define either flow priority or express a required quality of service.

Interpretation is captured by the reachability domain. Because policy language enables to define a scope for flow employing path constraints, the meaning of policy rules is given by assigning filters to paths rather than to individual edges.

```
domain Reachability includes Types, NetworkBasedApplications {
    IP ::= UI32.
    PT ::= Application.
    Priority ::= {0..7}.
    anyPrio ::= new(val:Priority).
    anyPrio(x) :- x = 0; x = 1; x = 2; x = 3; x = 4; x = 5; x = 6; x = 7.
    anyClass ::= new(cls:{ BE,EF,AF,CS}).
    anyClass(c) :- c = BE; c = EF; c = AF; c = CS.

    Dscp ::= new (class:{ BE, EF, AF, CS}, value:Priority).

    Flow ::= new (srcIp:IP,dstIp:IP,pt:PT,dscp:Dscp).

    PathFilter ::= new (path:Path, flow:Flow).
    Interface ::= new (ip:IP).
    Target ::= new (net:UI32R).
    Location ::= Interface + Target.
    Idge ::= new(dev:ID,ids:Interface,idd:Interface).
    Edge ::= new(eds:Location,edd:Location).
    % Path represented as a list of links:
    Path ::= new (Edge+Idge, any Path + {NIL}).
    % Computes a path from source to target
```

```
    px ::= (source:Edge,target:Edge, path: Path).
}
```

Semantic mapping defines a set or rules as transformations from the input domain to target domain. The main output relation is `PathFilter`. This relation is not computed directly from original policy rules. It is because, policy rules may be in conflict. To resolve conflicts, two auxiliary reachability relations are determined first. To compute forbidden flows along any path deny rule is defined in transformation module:

```
deny ::= (path:output.Path, flow:Flow).
deny(path,Flow(srcIp,dstIp,pt,dscp)) :-
    policy.Rule(Conversation(cc,cs,cp),scope,Access(DENY)),
    policy.application(cp,pt),
    addr(cc,srcIp), addr(cs,dstIp),
    admit(scope,path),
    anyDscp(dscp).
```

This rule marks any flow denied by a policy rule with matching conditions and scope. Similarly, rules for computing permitted flows are defined. Permit rules comprise except Permit policy rules also Priority and Qos policy rules. For instance, permit flow rule that is instantiated for a policy rule with set priority action is given as follows:

```
permit(path,Flow(srcIp,dstIp,pt,dscp)) :-
    policy.Rule(Conversation(cc,cs,cp),scope,Priority(pr)),
    policy.application(cp,pt),
    addr(cc,srcIp), addr(cs,dstIp),
    admit(scope,path),
    prioDscp(pr, dscp).
```

This statement represents the meaning of a policy rule with Priority action. Value of DSCP field is constrained to priority class CS that is itself expressed by definition:

```
prioDscp ::= (priority:Integer, dscp:Dscp).
prioDscp(x,Dscp(CS,x)) :- anyPrio(x).
```

It can happen that object v is in both access groups. This conflict is resolved by stating which of these groups has higher priority within the policy. Implicitly, deny group has the higher priority, thus, the transformation contains the following definition for a path filter:

```
PathFilter(p,f) :- permit(p,f), no deny(p,f).
```

This relation represents a set of filters that corresponds to a single policy specification. For flow to be permitted in a network, it must be explicitly enabled by some policy rule. For a flows not allowed in a network, it is either denied by a policy rule or not captured by any policy rule. Flows that are not explicitly addressed in any policy rule are called unclassified flows. A set of such flows can be identified by the following rule

```
unclass(flow) :- p is output.path,
                 no permit(p,flow), no deny(p,flow).
```

Difference in denied group, and unclassified group of flows has meaning when considering policy composition. Policies can be combined to form more complex policy. Possible compositions and their definitions are:

- "Serial" composition of policy A and B. Policy A has greater priority and thus only unclassified localized objects by policy A are classified by policy B. Policy $A \odot B$ classifies a flow according to definition of $\odot$ composition operator defined as follows:

$A \odot B$	*deny*	*unclass*	*permit*
deny	*deny*	*deny*	*deny*
unclass	*deny*	*unclass*	*permit*
permit	*permit*	*permit*	*permit*

- "Parallel-OR" composition of policy A and B. A flow is classified by policy $A \oplus B$ according to definition of $\oplus$ composition operator defined as follows:

$A \oplus B$	*deny*	*unclass*	*permit*
deny	*deny*	*deny*	*permit*
unclass	*deny*	*unclass*	*permit*
permit	*permit*	*permit*	*permit*

- "Parallel-AND" composition of policy A and B. A flow is classified by policy $A \otimes B$ according to definition of $\otimes$ composition operator defined as follows:

$A \otimes B$	*deny*	*unclass*	*permit*
deny	*deny*	*deny*	*deny*
unclass	*deny*	*unclass*	*permit*
permit	*deny*	*permit*	*permit*

Policy interpretation based on reachability model enables to analyze the consistency of any policy and also to check some other properties. It is possible to check if a policy is a subsumption of some other policy or to test equivalence of two policy systems.

4.4.2 Conformance Checking

A set of policy rules can contain conflicts. A conflict between a pair of rules is when different action is applied to the same target object. The purpose of conformance checking is to detect and report these rules to a user. Because NPSL employs space and time scope constraint model, another form of problems may be found when policy is to be analyzed with respect to network topology. It may be possible that space scope constraint cannot be satisfied within a network topology. In this case, the tool should report this issue to a user as enforcing unsatisfiable constraints is probably not the

intention of any reasonable policy specification. Sometimes, the requirement that a policy rule must be applied in the network is stated explicitly (see example of policy that requires the existence of redundant paths).

Network policy can be validated to meet high-level requirements. There are best practices that govern what services should be enabled or disabled and how particular network traffic should be handled. Having specified network policy in the form of a network reachability model it is possible to analyze this model with respect to rules from predefined collection of best practices.

4.5 Chapter Summary

Existing network policy languages were presented in this chapter together with proposal and evaluation of a new policy language intended for formalizing network policies that can be represented as constraints on network flows or services. Proposed network policy language can be applied in the following cases:

- *Network design* contains an informal description of functional and security requirements. The benefit of formalizing network requirements is for assessing network design with respect to stated requirements and security standards.
- *Real-time control* of OpenFlow software-defined networks utilizes operations on flows to implement network requirements. In these networks, it is possible to fix paths for individual flows, which offers a fine-grained level of control. Policy decision engine is a part of SDN controller. Hinrichs et al. tested effectiveness of a logic-based decision engine for SDN in [34] with promising results.
- *Network configuration validation* can take network policy specification as input requirements. As network configuration is also interpretable in reachability model, the configuration validation stands for identifying if configuration's reachability model subsumes the reachability model computed from policy specification.

The network policy language introduced in this chapter stems from Path-based policy language proposed by Stone and flow-based language developed by Hinrichs et al. Because of this, it shares many properties with these two languages. Expressiveness of the proposed language enables to describe many flow-based features of network policies. In addition, service-based features can also be specified. Reasoning about accessibility of services and related security properties is possible in generated reachability model. In comparison to other policy languages, security properties that can be expressed by NPSL are limited to access control.

Although many properties can be directly represented in the proposed network policy language, much work remains to make this system practical for defining network policies of current enterprise networks. Completed case studies suggest that proposed language is quite flexible to formalize many properties and concepts. Possibility to arrange specifications into domains and employ mechanisms for domain inclusion enable developing a near-complete system of network policy concepts. The presented language provides the foundation for the complete environment formalizing and reasoning about network policies in practical network design.

Chapter 5
Firewall Verification

Firewalls are network devices deployed to implement network access security policy. Each firewall consists of a set of rules where every rule specifies an action to be executed on matching packets. Firewall verification stands for checking whether a given set of firewall rules conforms to the intended security policy. Another problem is to verify that there are not conflicts between rules of the firewall that can signalize the error done by the author of this regulation. Checking conflicts does not require knowing the intended network policy. Similarly as in program analysis where particular sequence of statements identify error independently to program specification, in case of firewall rules a certain combination of rules signalizes the wrong implementation regardless the network policy. Al-Shaer, Hamed, Boutaba and Hasan [4] provide the most extensive classification of conflicts between rules. Some conflicts do not represent wrong policy implementation but can have an effect on firewall performance. Several methods were presented on finding and removing such conflicts. Conflict resolution methods are not only useful for the firewall efficiency but also for finding a conflict free firewall rule base, which is often a source for verification methods based on constraint solving.

This chapter first presents a description of the functionality of a typical firewall device, which helps to build an idea of the problem of implementing network policy through configuration of firewalls. Next, a unified formal model of a firewall device defining the semantics of packet matching algorithm is introduced. In section 5.3, the conflict classification framework is presented. This structure is employed in the following two sections where methods for conflict checking and resolution are defined and explained. Section 5.4 contains a review of existing methods for the complete firewall verification against specified network policy. The last section of this chapter provides design, implementation and evaluation of the new conflict checking algorithm.

5.1 Firewalls

A firewall controls the flow of packets and selectively drop packets depending on the implemented network security policy. Usually, a firewall is driven by an access control rules forming Access Control List (ACL). An ACL consists of ordered list of rules. Each rule has an action and attributes that define matching condition. To find an action for processed packet, rules are checked from the top of the list. The first rule that matches the packet is selected, and its action is executed. Ordinary firewalls perform matching procedure by inspecting only packet's header fields. Overlapping rules

```
1: tcp,  140.192.37.20, any,       *.*.*.*,  80,   deny
2: tcp,   140.192.37.*, any,       *.*.*.*,  80, accept
3: tcp,        *.*.*.*, any, 161.120.33.40,  80, accept
4: tcp,   140.192.37.*, any, 161.120.33.40,  80,   deny
5: tcp,  140.192.37.30, any,       *.*.*.*,  21,   deny
6: tcp,   140.192.37.*, any,       *.*.*.*,  21, accept
7: tcp,   140.192.37.*, any, 161.120.33.40,  21, accept
8: tcp,        *.*.*.*, any,       *.*.*.*, any,   deny
9: udp,   140.192.37.*, any, 161.120.33.40,  53, accept
10:udp,        *.*.*.*, any, 161.120.33.40,  53, accept
11:udp,   140.192.38.*, any,  161.120.35.*, any, accept
12:udp,        *.*.*.*, any,       *.*.*.*, any,   deny
```

Fig. 5.1 An example of firewall configuration

Firewall configuration consists of lines of rules. Each rule has 7 columns with that represent: (1) rule order (priority), (2) packet protocol, (3) source address range, (4) source port number, (5) destination address range, (6) destination port number (application), and (7) action associated with the rule.

are common in practice as they often make ACL shorter and better express the intention, but in several cases it signalizes the problem in ACL correctness.

5.1.1 First Match

In first-match semantics, the filter is processed sequentially in the order specified by the priority of its rules. Rules in the filter have priorities that determine the order in which these rules are tested to match the packet. If the rule matches the packet the rule's corresponding action is executed. Otherwise, a next rule is taken. This process continues until matching is found, or the end of the filter is reached. In case, no match is found during the evaluation of the filter a default action is applied to the packet. The design of this process is shown in Fig.5.2. The implementation of the first match algorithm can employ hash tables or tree representation to make the processing more efficient.

5.1.2 Longest Prefix Match

Longest prefix match is often used for inspecting routing tables than filtering. The entries in the table have form of prefixes. The algorithm tries to find the longest prefix that matches the given packet. It is straightforward for routing where the matching problem needs to be solved only for a single dimension, which represents the destination address. Because the longest prefix match has better complexity compared to the first match, several methods were proposed to use longest prefix match also for filtering. The filter represented in the form of a list of rules can be used as the source for creating prefix trees that allow for implementing optimized query operation.

5.2 The Firewall Model

A access control list admits or denies the packets that cross the interface to which the access control list assigned. It is thus a function that for each packet determines an action that should be applied. The firewall implements a filtering policy. Consider a set of *Packets* and a set of possible *Actions*, we can define firewall to be a filter function defined as follows:

$$Filters \subseteq Packets \rightarrow Actions$$

In the following, a packet model and rule representation is developed. Based on this, the possible representation of filtering function of a firewall is defined. Simple description of firewalls are considered, which means that the rules can match only fields from the header of a packet. Current firewall can be more complicated allowing for implementing advanced matching of other properties in packets using techniques such as Deep Packet Inspection [1]. In principle, the similar approach can be applied but deeper examination is needed to find efficient way of modeling and analyzing these kinds of firewalls.

5.2.1 Packet Model

To create a simple but sufficient packet model we start with an observation of properties and behavior of firewall rules. The rules match packets according to a particular set of fields in packet's header. Although techniques such as Deep Packet Inspection enables to classify a packet based on other information than header fields, we stick to a fixed set of fields for most of the content in this chapter.

A packet p is modeled as the d-dimensional vector from the d-dimensional domain, formally:

$$p \in D_1 \times D_2 \times \ldots \times D_n.$$

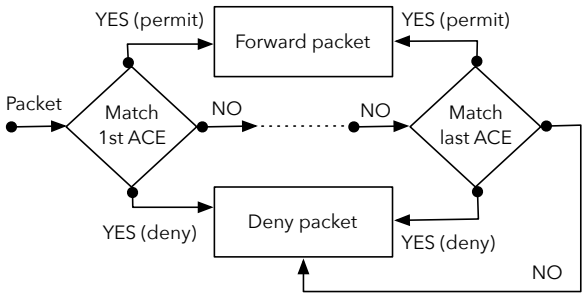


Fig. 5.2 ACL Matching Algorithm

First match algorithm requires to iterate through all rules until the rule that matches inspected packet is found. Implicit deny rule is applied if no rule matches the packet.

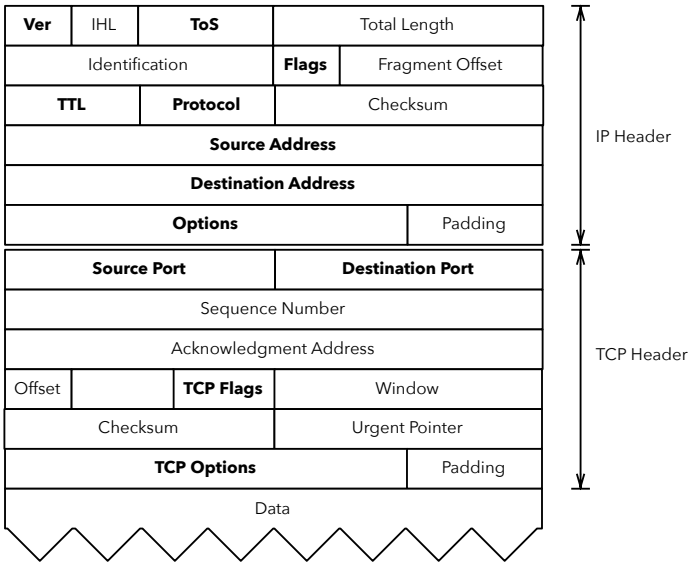


Fig. 5.3 IP and Tcp header format

Fields that are commonly used for packet matching have bold labels. However, modern hardware enables to create rules that can analyze any combination of fields of a header or even any byte within an inspected packet.

Usually, and for the scope of this section, we consider the following 5-dimensional domain:

- D_0 is range $0 \leq d_0 \leq 2^{32} - 1$ corresponding to packet's *Source Address*,
- D_1 is range $0 \leq d_1 \leq 2^{32} - 1$ corresponding to packet's *Destination Address*,
- D_2 is range $0 \leq d_2 \leq 2^{16} - 1$ corresponding to packet's *Source Port* number,
- D_3 is range $0 \leq d_3 \leq 2^{16} - 1$ corresponding to packet's *Destination Port* number,
- D_4 is range $0 \leq d_4 \leq 2^{8} - 1$ corresponding to packet's *Protocol* number.

Domains for IP addresses, port numbers and protocol types are introduced as follows:

```
IP ::= (left:UInt32, right:UInt32).
PN ::= (left:UInt16, right:UInt16).
PT ::= (left:UInt16, right:UInt16).
```

The presented 5-dimensional model is sufficient for most firewall configurations as in the vast majority the firewall rules match a subset of these fields.

```
Flow ::= (pt: PT, srcIp : IP, srcPn : PN, dstIp: IP, dstPn : PN).
```

Figure 5.2.1 shows a structure of relevant protocol headers for TCP packets.

The presented packet model dictates the further development of the model for access rules. An access rule consists of matching part, which can contain matching condition for each field identified in the packet model. Each matching field represents a new dimension. For filter processing, adding a new field means to process an additional dimension by the algorithm.

5.2.2 *Ranges, Prefixes and Protocol Field*

Presented packet encoding considers that any of the header fields can be encoded in a single (integer) domain.

Firewall configuration language provides some flexibility in representation of the matching patterns for each field. For instance, it is possible to use wildcards when specifying addresses in ACL rules. For port fields, intervals can be used. A protocol field represents refers to protocols of network and transport layers. For instance, by specifying that rule matches IP protocol it matches all IP packets regardless on transport layer protocol. Specifying TCP protocol means that the rule matches only IP packets that also contain TCP segments. For the purpose of representation of different fields in a uniform manner, we need to be able to represent the specifications as mentioned above as ranges. Also, for certain algorithms or firewall representations it is suitable to use prefix representation instead of ranges.

Description of addresses and intervals of addresses is straightforward for IPv4 addresses that represent 32-bit integers. IPv6 addresses represent 128-bit integer values. Finally, port numbers represent 16-bit integers.

Representing protocols as ranges is defined in table bellow. Mapping protocols to ranges, it is possible to employ same operations as for other dimensions in the most of filter manipulation algorithms.

Protocol	Range	Interval
ip	[0.0.0.0,255.255.255.255]	0 - 4294967295
icmp	[1.0.0.0,1.255.255.255]	16777216 - 33554431
igmp	[2.0.0.0,2.255.255.255]	33554432 - 50331647
tcp	[6.0.0.0,6.255.255.255]	100663296 - 117440511
udp	[17.0.0.0,17.255.255.255]	285212672 - 301989887

In firewalls, it is usually possible to specify other information on individual protocols, e.g., matching only SYN packets of TCP communication or particular types of ICMP protocol. Proposed encoding of protocols into ranges can be extended to accommodate it. This is shown and explained in the case of TCP protocol in Fig. 5.4. For more complicated cases, it would be necessary to implement a different method for handling the information, e.g., considering set representation instead of ranges. Nevertheless, for most of the rules found in ACL configuration, the presented encoding suffices.

Protocol dimension is used for encoding additional information on flows. For instance, TCP flows can be filtered using the following rule:

```
access-list 199 permit tcp any any established
```

The established keyword is used to specify that the packet needs to be from established TCP connection. It means that matching TCP segments have Acknowledgment (ACK) or Reset (RST) bit set, which indicates that packets are from responder to originator direction.

```
Pt ::= fun (pt:Protocol => PT).
Pt(IPV4, PT(0, 4294967295)).
Pt(ICMP, PT(16777216, 33554431)).
```

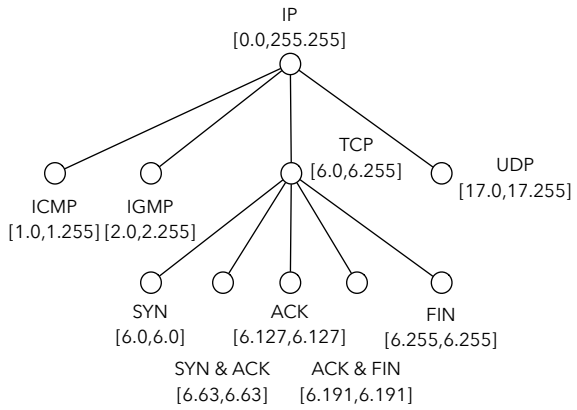


Fig. 5.4 An example of TCP/IP protocol encoding

Encoding is extended with other 8 bits to have total width 16bits. The dot notation is used for presentation purposes. Encoding values and enumerations that are disjoint leads directly to non-intersecting ranges. For flags the range representation is possible if only subset of all combinations is considered. This is illustrated for TCP flags encoding. The TCP packet with SYN and ACK set contains value 6.63 in the protocol field. It can be matched by rule that matches SYN packets ([6.0, 6.63]) or ACK packets ([6.63, 6.191]).

```
Pt(ICMP_ECHO, PT(16777216, 16777216)).
Pt(ICMP_ECHOREPLY, PT(17301504, 17301504)).
Pt(IGMP,PT(33554432,50331647)).
Pt(UDP,PT(285122672,301989887)).
Pt(TCP,PT(100663296,117440511)).
Pt(TCP_ESTABLISHED,PT(109051904,117440511)).
```

The present approach considers a uniform representation of all dimension using intervals. To obtain the uniform representation, some computation would be necessary. Protocol is usually specified by its symbolical name (*tcp*) that has to be converted to a range value (6.0.0.0 – 6.255.255.255). Ports are represented by integer intervals (0 – 1024), which can be taken without modifications. Addresses are often given using prefixes (172.16.10.0/24) or using wildcards. A conversion from the prefix format to the interval format is straightforward. Prefix representation of ranges involves splitting ranges in multiple subranges. It introduces additional rules that can have an impact on the run-time of algorithms. For example, the interval $[2, 8]$ can be converted to prefixes $001*, 01*, 1000$. Instead of a single rule, three rules needs to be analyzed.

5.2.3 A Model for Access Rules with Priorities

In this section, the model for ordered access rules is presented. Each access rule consists of matching part, action, and priority. The model can be further extended by adding new fields as needed. The matching component is uniformly represented as the multidimensional space where each dimension is discrete and finite.

An access-list rule is a record consisting of

- d-dimensional ranges that define matching criteria,
- action that is to be executed on matching packet
- and priority to resolve possible multiple matching.

Access-list rules are defined using the following function.

```
Rule ::= fun (filter:Filter, prio:Integer, pt: Protocol,
              srcIp : IP, srcPn : PN, dstIp : IP, dstPn : PN
           => action : Action).
```

Each rule is assigned to a filter object and comprise of matching constraints and an action. Each `Rule` statement contains same fields as `Flow` and adds filter reference and action specification.

Definition 3 (Packet Matching). A rule $r \in$ `Rule` matches flow $f \in$ `Flow` if in each of its dimensions, rule r entirely contains flow f:

```
match ::= (r:Rule,f:Flow).
match(r,f) :- contains(r.pt,f.pt),
              contains(r.srcIp,f.srcIp), contains(r.srcPn,f.srcPn),
              contains(r.dstIp,f.dstIp), contains(r.dstPn,f.dstPn).
```

Based on packet matching definition it is possible to express a condition for finding a rule that matches a packet. As the filter can contain an arbitrary set of rules, it is necessary to resolve possible multiple matching and find a single rule that matches the given packet in a deterministic way. One of the possible approaches is to use priority for resolving multiple matches.

Definition 4 (Classification with Priority Resolution). A packet $p \in$ *Packets* is classified by a rule $r \in f$, if and only if the packet is matched by the rule r and no other rules r' with lower priority than rule r matches packet p, formally:

```
classify ::= (filter:Filter,rule:Rule,flow:Flow,action:Action).
classify(filter,rule,flow,action) :-
    rule is Rule(filter,_,_,_,_,_,action),
    match(rule,packet),
    no { q : Rule | q.Filter = filter, match(q,packet),
                    q.priority < rule.priority }.
```

Prioritization of rules is one of the possible approaches to creating filters where the potential matching conflict can be resolved, and the matching process is deterministic.

Lemma 1 (Deterministic classification). *Given a filter $\in$ Filter, then for any given flow $\in$ Flow there is at most one rule $\in$* `Rule`$(filter, \_\_\_\_, action)$ *that satisfies*

$$\texttt{classify}(filter, rule, flow, action).$$

Proof. By definition, for every packet and filter, `classify` *predicate is satisfied for every rule that matches the packet, and there is not another rule with a lower priority that also matches the packet. From the assumption of the lemma, the priority of rules are always ordered, and no two rules have the same priority. Thus if at least*

some rule satisfies `classify` *predicate it is always that with the least priority and no other rules can satisfy this predicate at the same time.*

A priority is a way to solve possible conflicts among the classification rules. In general, a conflict between rules arises if they overlap in all their dimensions. In the next section, the classification of these anomalies is provided.

5.3 Conflict Classification

This section provides an overview of a conflict classification problem. A conflict in access control list occurs if at least two rules of the list match the same packet. There are various kinds of conflicts, some of them representing configuration error while others may be intentional. First, basic relations among individual rules are examined and later a classification framework that provide a complete information on conflicts found is presented.

Classification of anomalies of firewall rules presented in this section is due Al-Shaer [4] and Qian et al.[70].

5.3.1 Basic Relations

Two rules are either disjoint or in an intersection. First, a group of disjoint relations is examined. Disjoint relations can be further refined by examining individual dimensions. If the rules do not intersect in any of its dimension, then they are called completely disjoint. If they intersect in some of their dimensions, they are partially disjoint. There is also a particular case, when two rules are adjacent. This case is interesting as it enables merging these two rules in some situation as discussed later in section 5.5.

In the following definitions, operators from `Range` domain are used. This domain is specified as follows:

```
domain Range {
    R ::= (left:Integer, right:Integer).

    eq ::= (r1:R,r2:R).
    eq(r1,r2) :- r1 is R, r2 is R,
                 r1.left = r2.left,
                 r2.right = r2.right.

    inter ::= (r1:R,r2:R).
    inter(r1,r2) :- r1 is R, r2 is R,
                    r1.left <= r2.left, r1.right >= r2.left.
    inter(r1,r2) :- r1 is R, r2 is R, inter(r2,r1).
}
```

Predicate `eq` tests equality on two ranges. Predicate `inter` evaluates to true if the given intervals intersect.

Further, a helper function `dim` that provides access to individual domains can be defined as follows:

```
D ::= { PROTO, SRCIP, SRCPN, DSTIP, DSTPN }.
dim ::= (rule:Rule, d:D, range:R).
dim(rule,PROTO,range) :- rule is Rule, range is Range,
    range.left = rule.proto.left, range.right = rule.proto.right.

dim(rule,SRCIP,range) :- rule is Rule, range is Range,
    range.left = rule.srcIp.left, range.right = rule.srcIp.right.

dim(rule,SRCPN,range) :- rule is Rule, range is Range,
    range.left = rule.srcPn.left, range.right = rule.srcPn.right.


dim(rule,DSTIP,range) :- rule is Rule, range is Range,
    range.left = rule.dstIp.left, range.right = rule.dstIp.right.

dim(rule,DSTPN,range) :- rule is Rule, range is Range,
    range.left = rule.dstPn.left, range.right = rule.dstPn.right.
```

Disjoint rules represent rules that do not have conflicts as they do not match any common packets.

Definition 5 (Disjoint Rules). Two rules are disjoint if they do not intersect at least in one of their dimensions:

```
disjoint(p,q) :- d is D, dim(p,d,x), dim(q,d,y), no Range.inter(x,y).
```

Disjoint rules can be further classified according to intersection in each of domains.

Definition 6 (Completely Disjoint Rules). Two rules are *completely disjoint* if they are disjoint in any of their dimensions.

```
completeDisjoint(p,q) :-
    count { d : D | d is D, dim(p,d,x), dim(q,d,y),
                    no Range.inter(x,y)} =
    count { d : D | d is D}.
```

A special case of disjoint relation is adjacent relation. Identification of this relation may be useful as it enables us to merge adjacent rules.

Definition 7 (Adjacent Disjoint Rules). Let interval x is adjacent to interval y,

$$Adjacent(x,y) := \begin{matrix} \vee Math.Max(x)+1 = Math.Min(y) \\ \vee Math.Max(y)+1 = Math.Min(x) \end{matrix}$$

Two rules are *adjacent disjoint* if they are adjacent in a single dimension and they are equal in all other dimensions.

```
adjacentDisjoint(p,q) :-
    count { d : D | d is D, dim(p,d,x), dim(q,d,y),
                    Range.eq(x,y) } = D.count - 1.
    count { d : D | d is D, dim(p,d,x), dim(q,d,y), Range.adj(x,y)} = 1.
```

A group of intersection relations define various relations for two rules that intersect in all of their dimensions. If two rules are equal in all of their dimensions, the intersection relation is denoted as exactly matching. If one rule is contained in another rule then this case is denoted as inclusive matching. Again, there is a particular relation for the case if two rules intersect in a single dimension and are equal in other dimensions. This instance denotes correlated matching. In [70], this relation is called as overlapping.

Definition 8 (Intersection of Rules). Two rules are in intersection relation, if:

```
intersects(p,q) :-
    count { d : D | d is D, dim(p,d,x), dim(q,d,y),
                    Range.inter(x,y) } = D.count.
```

Disjoint relations represents a situation when two rules match different packets; even though they can intersect in some of their dimensions.

Matching relations stand for cases when two rules match the same subset of packets. Depending on relations between individual dimensions we recognize three kinds of matching relations, namely, exact, inclusive, and overlapping matching.

Definition 9 (Exactly Matching). Two rules are *exactly matching* if they define the same ranges in all of their dimensions.

```
exact(p,q) :-
    count { d : D | d is D, dim(p,d,x), dim(q,d,y),
                    Range.eq(x,y) } = D.count.
```

Definition 10 (Inclusive Matching). Rules are inclusively matching if rule p matches all packets matched by rule q .

```
includes(p,q) :-
    count { d : D | d is D, dim(p,d,x), dim(q,d,y),
                    Range.includes(x,y) } = D.count.
```

Definition 11 (Overlapping Matching). Rules P and R are overlapping if they share same subset of packets, but they are not in Exact nor Includes relation.

```
    overlaps(p,q) :-
        intersects(p,q), no exact(p,q), no includes(p,q).
```

Previously defined relations provide an exhaustive classification for any pair of rule as stated in the following proposition.

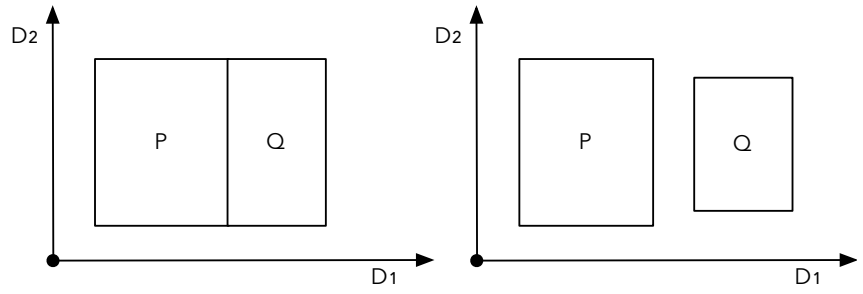


Fig. 5.5 An example of different inter-rule relations

Left-hand side picture shows rules in adjacent disjoint relation. Right-hand side picture shows rules in partial disjoint relation. In both cases, rules P and Q overlaps in dimension D_2.

Theorem 1. *Let F∈Filters is an arbitrary filter. Any pair of rules of this filter is related by at least one defined relations.*

$$\begin{aligned}\forall R_1, R_2 \in F : &\vee \texttt{disjoint}(R_1, R_2)\\ &\vee \texttt{exact}(R_1, R_2)\\ &\vee \texttt{includes}(R_1, R_2)\\ &\vee \texttt{overlaps}(R_1, R_2)\end{aligned}$$

Proof. Each dimension is represented as a range. There are four possible combination of two ranges, namely, disjoint ($r_1 \cap r_2 = \emptyset$), intersection ($r_1 \cap r_2 \neq \emptyset$), subset ($r_1 \subset r2$) and equality ($r1 = r2$). We need to show that for all combinations in each dimension there is a defined relation. We start with two dimensions only and then extend it to n-dimensions using induction.

5.3.2 Conflict Classes

An anomaly between rules is classified into four classes. In general, the anomaly is a conflict between two (or more) rules such that the rules match at least one packet in common. Not all conflicts represent an error in the access list. Some of the conflicts are intentional as they either provide more intuitive or more compact representation. Conflict classes definitions are built from previously defined rule relations.

Definition 12 (Shadowing). A rule R is shadowed by previous rule P if rule P matches all packets that are matched by rule R. Because rule P precedes rule R in a priority list then rule R will never be fired. Formally, P shadows R if:

$$\begin{aligned}&\wedge \quad P[priority] < R[priority]\\ \mathit{Shadows}(P, R) := &\wedge \quad \texttt{includes}(P, R) \vee \texttt{exact}(P, R)\\ &\wedge \quad P[action] \neq R[action]\end{aligned}$$

Because rules have different action, shadowing is considered as a serious conflict as rule R is never fired and thus the policy it specifies has not effect in the filter. The problem may arise when

administrator wants to drop certain traffic, and this is expressed in the shadowed rule. Then this traffic will be permitted causing security issue.

Definition 13 (Correlation). Two rules are in correlation anomaly if they differ in action and they matches some common subsets of packets. Rule P correlates with rule R and vice versa if, formally,

$$Correlates(P,R) := \begin{array}{ll} \wedge & \texttt{overlaps}(P,R) \\ \wedge & R[action] \neq P[action] \end{array}$$

Correlation is not considered to be a fatal configuration error. Correlation complicates understanding of access control list. An administrator has to be aware that some packets matched by the lower-priority rule will be processed according to the action of a higher priority rule. On the other hand, removing correlation conflicts can introduce additional rules making control list longer.

Definition 14 (Generalization). Rule R generalizes the preceding rule P if it matches all its packet and the actions of the rules are different, formally,

$$Generalizes(R,P) := \begin{array}{ll} \wedge & P[priority] < R[priority] \\ \wedge & \texttt{includes}(R,P) \\ \wedge & P[action] \neq R[action] \end{array}$$

Generalization is often intentionally introduced in firewall configurations for implementing special handling to a subset of packets that are handled in opposite way by a more general rule. Using generalization, it is possible to reduce the length of filter [50].

Definition 15 (Redundancy). Redundant rule matches same packet as other rule with lower priority and has the same action as this priority rule. Formally, rule R is redundant to rule P:

$$Redundant(R,P) := \begin{array}{lll} & \wedge P[priority] < R[priority] & \\ & \wedge R[action] = P[action] & \\ & \quad \vee \texttt{exact}(P,R) & \\ & \quad \vee \texttt{includes}(P,R) & \\ \wedge & & \wedge P[priority] < Q[priority] < Q[priority] \\ & \quad \vee \texttt{includes}(R,P) \wedge \neg \exists Q : & \wedge P[action] \neq Q[action] \\ & & \wedge \texttt{includes}(Q,P) \vee \texttt{overlaps}(P,Q) \end{array}$$

If redundant rule is removed, the policy remains the same. Redundancy is not dangerous for security but redundant rules add unnecessary overhead. Figure 5.6 visually explain the role of rule Q in the definition.

In this section, four classes of rule anomalies were presented. This classification describes the anomaly class between pairs of rules. Finding rule anomalies can be complicated as a single rule may be involved in more than one anomaly relation. In next section, anomaly detection and resolution algorithms are presented.

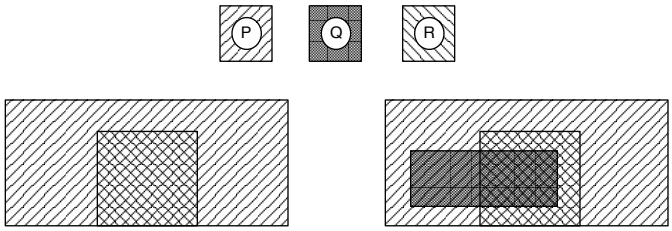


Fig. 5.6 An example of redundant and non-redundant rules

Considering $R[priority] < Q[priority] < P[priority]$. Left-hand picture shows the case where rules R and P are redundant. Right-hand picture shows the case where rules R and P are not redundant because of rule Q. Rule Q matches a subset of packets of rule P and rule R which has greater priority than rule Q performs the same action as rule P on a specific subset of packets that would otherwise be matched by rule Q.

5.3.3 Anomaly Classifier

An anomaly classification algorithm that for a pair of rules decides the class of the anomaly is presented in this section. This algorithm is defined according the previously given classification. An input to this algorithm is a pair of rules and the output is either a class of conflict or indication that rules are conflict free.

Because this classifier works only with a couple of rules P, R the redundancy cannot be fully identified. Instead, potential redundancy is identified, and further checking is needed to determine real redundancy conflict.

The classifier has form of finite automate that process dimensions of input rules. The individual classes of anomaly between a pair of rules recognized by the classifier are as follows:

- Shadowing ($sha(P, R)$): A rule R is shadowed by previous rule P if rule P matches all packets that are matched by rule R.
- Redundancy ($red(P, R)$): A rule R is redundant to rule P if rule P matches all packets matched by rule R and both rules have the same action.
- Potential Redundancy ($pred(P, R)$): A rule P is potentially redundant to rule R if rule R matches all packets matched by rule P and they have the same action.
- Generalization ($gen(P, R)$): A rule R generalizes rule P if rule R matches all packets matched by rule P and they have different actions.
- Correlation ($cor(P, R)$): A rule R correlates with rule P if they matches the common subset of packets and differ in their actions.
- Disjoint ($dis(P, R)$): A rule R is disjoint to rule P if they do not match any single packet.

Al-Shaer et al. [3] presented conflict classifier for three dimensions. Here, a finite state machine that will detect conflicts for n-dimensions is presented.

Definition 16. Conflict classifier is a finite state machine defined as:

$$Classifier := (\Sigma, S, s_{ini}, \delta, F)$$

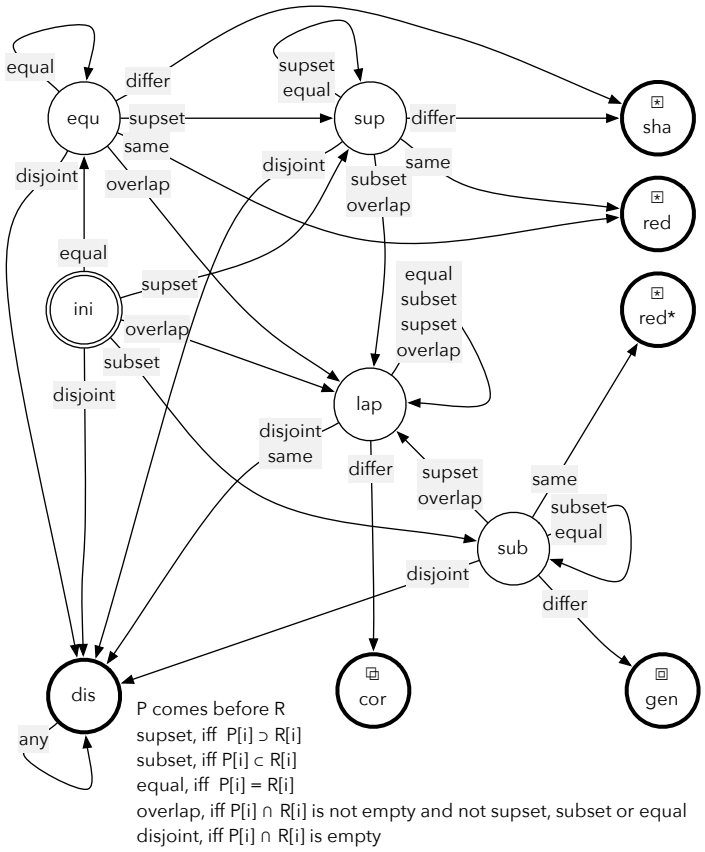


Fig. 5.7 A conflict classification state machine

where

- $\Sigma = \{equal, disjoint, supset, subset, overlap, same, differ\}$ is the input alphabet
- $S = \{s_{ini}, s_{equ}, s_{esb}, s_{sup}, s_{lab}, s_{sub}, s_{dis}, s_{sha}, s_{red}, s_{red*}, s_{gen}, s_{cor}\}$ is a set of states,
- s_{ini} is an initial state,
- $\delta : S \times \Sigma \rightarrow S$ is a transition function, and
- $F = \{s_{red}, s_{red*}, s_{sha}, s_{gen}, s_{cor}, s_{dis}\}$ is a set of final states.

Transition function is defined as shown in Fig 5.7.

Classifier compares rules in each dimension. Results of comparison are inputs for the state machine of the classifier. Example 1 illustrates the functionality of the classifier on a simple case.

Example 1. This example shows how the classifier determines the class of anomaly. Input of the classifier will be a sequence of symbols each representing a result of the comparison of two corresponding fields from the pair of entry of rules. The classifier functionality is demonstrated on two rules R_1 and R_2 defined bellow:

$$R_1 := [match = \begin{bmatrix} 0 \mapsto \{tcp\} \\ 1 \mapsto \{140.192.37.20\} \\ 2 \mapsto * \\ 3 \mapsto *.*.*.* \\ 4 \mapsto \{80\} \end{bmatrix}, action = deny, priority = 1]$$

$$R_2 := [match = \begin{bmatrix} 0 \mapsto \{tcp\} \\ 1 \mapsto 140.192.37.* \\ 2 \mapsto * \\ 3 \mapsto *.*.*.* \\ 4 \mapsto \{80\} \end{bmatrix}, action = accept, priority = 2]$$

Conflict classifier will be executed on the stream of symbols $\langle i_0, i_1, i_2, i_3, i_4, i_5 \rangle$ computed as follows:

$$\begin{aligned} &i_0 = equal \text{ because } R_1[match][0] = R_2[match][0] \\ &i_1 = subset \text{ because } R_1[match][1] \subset R_2[match][1] \\ &i_2 = equal \text{ because } R_1[match][2] = R_2[match][2] \\ &i_3 = equal \text{ because } R_1[match][3] = R_2[match][3] \\ &i_4 = equal \text{ because } R_1[match][4] = R_2[match][4] \\ &i_5 = differ \text{ because } R_1[match][5] \neq R_2[match][5] \end{aligned}$$

The sequence of classifier state machine configurations is following:

$$\begin{aligned} &(ini, \langle equal, subset, equal, equal, equal, differ \rangle) \mapsto \\ &(equ, \langle subset, equal, equal, equal, differ \rangle) \mapsto \\ &\qquad \vdots \\ &(esb, \langle differ \rangle) \mapsto (gen, \langle\rangle) \end{aligned}$$

The result of classification is that rules R_1 and R_2 are in generalization anomaly, in particular, R_2 generalizes R_1. This anomaly is considered only as warning and is often used in real firewalls [3].

5.4 Firewall Representation

In this section, a number of different structures for firewall representation are discussed. The purpose of different representations is to provide an underlying data structure offering efficient implementation of analytical methods. The order list of rules yields to conflict checking algorithms with time complexity $O(n^2)$. The C implementation that performs the pairwise comparison of rules implemented as a single outer loop, and a single inner loop can typically process a firewall containing 10^4 rules in 10 seconds. From the practical viewpoint and considering currently available computation power this can be sufficient for smaller filters. Nevertheless, if the conflict checking algorithm is a part of a complex verification system or online detection tool, more efficient solution is desirable.

Optimized algorithms of conflict detection require a suitable data structure that provides an efficient implementation of specific essential operations. Tree-based structures are prevailing for this purpose. They offer efficient data insertion and searching, which is suitable for conflict classification algorithms.

This section surveys several representations. It begins with the description called policy trees. Policy trees speed up the procedure of conflict checking significantly by providing for each rule a set of potential conflicting rules.

Next representation is a multidimensional interval tree. Although the basic operations are more complicated than in policy trees, interval trees provide more compact representation for firewall rules.

Another tree based method is based on the Trie representation [39]. This representation also employs bit vectors for efficient storage of index information about rules in each branch [7].

There are also other methods, which are out of the scope of this thesis. For instance, firewall rules represented using geometry. The conflict or matching checking problem is transformed to the problem of finding intersection of hyper-ranges in d-dimensional space. Eppstein and Muthukrishnan [20] describe a method for fast packet matching using rectangle geometry. Part of their work is focused on conflict detection, which amounts to represent rules in d-dimensional space and finding intersection among the hypercubes that represent these rules. They showed how to use kD-tree for this purpose.

5.4.1 Policy Tree

A policy tree [2] is a simple tree representation of filtering rules. Each node in a tree represents a header field. Each outgoing edge represents a possible value that the header field can have. In this representation, each rule is encoded in a form of the path from the root node to a leaf node. Rules that have the same values in some fields can share a part of their paths. For this representation, authors developed conflict classification algorithm that can analyze a single firewall [3] as well as a set of firewalls [4]. An example of the policy tree is shown in Fig.5.8.

Leaf nodes contain associated rules. In addition to rule's identification, there is also a set of rules that are in conflict with the primary rule. Policy tree representation treats individual domains different to other tree-based representations. Referring to Fig. 5.8, rule no.1, and rule no.5 have separate paths from other rules although their source addresses are part of 140.192.37.* range. This representation simplifies the insert operation. If there is the exact match with some branch, then the rule is inserted in this branch; otherwise a new branch is created. However for finding conflicting rules each rule has to be also checked against also subset and superset branches. For instance, when inserting rule no.6 the branch of rule 5 is a subset branch as all fields of rule no.5 are subsets of fields of rule no.6. Thus, the algorithm identifies that rule no.5 is a subset of rule no.6 that is interpreted as generalization conflict between rule no.6 and rule no.5. In conflict classification, the policy tree provides an efficient structure for determining if there is a conflict. It is done by checking if the path of newly added rule conicides with paths of existing rules in the tree. If the rule's path does not coincide with any other path, we can be sure that there is no conflict.

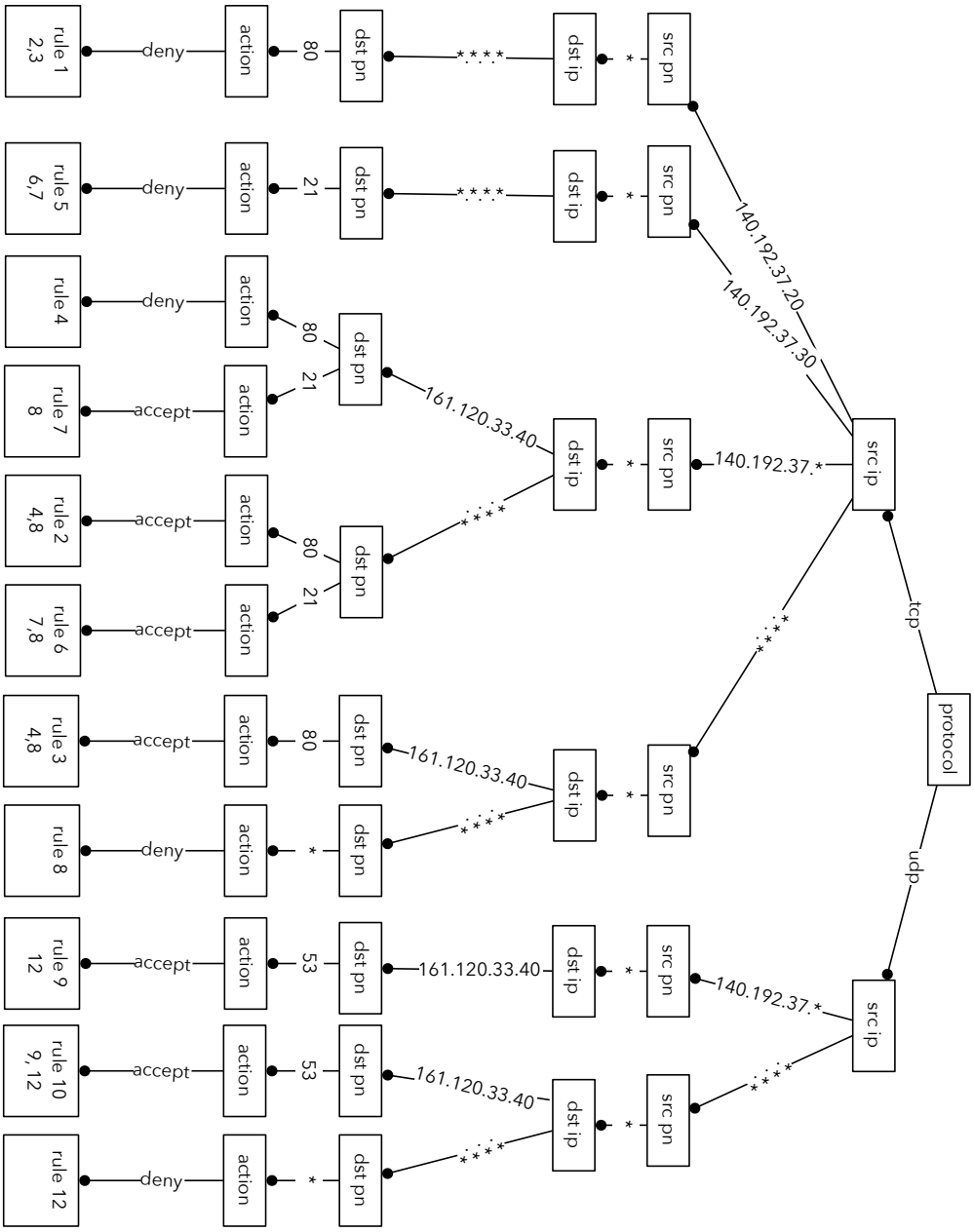


Fig. 5.8 A policy tree for example ACL in Fig.5.1(taken from [2]).
Edges correspond to possible values of the source fields. Note that values labeling these edges do not have to be disjoint. This distinguishes the policy tree from other tree-based representations.

5.4.2 Firewall Decision Diagrams

Liu and Gouda introduced a Firewall Decision Diagram in [24]. A *firewall decision diagram* (FDD) is an acyclic and directed graph $G = (N, E)$. It has exactly one root node. Non-terminal nodes $v_1, \ldots, v_n$ are labeled with field names, $F(v_i) \in \{F_1, \ldots, F_d\}$. Terminal nodes $v_1^t, \ldots, v_m^t$ are labeled with action, $F(v_i^t) \in \{permit, deny\}$. An outgoing edge e of node v is labeled with the non-empty set of integers, denoted $I(e)$. Edge labels are drawn from domains of associated fields. For node v, it label is given as $I(e) \subseteq D(F(v))$, e.g., numbers from 16-bit unsigned integer for domain of transport layer port identifier. A directed path from the root ending at a terminal node is called a decision path. All nodes on the decision path must be labeled with different fields. In addition, each FDD has to satisfy the consistency and completeness property:

- Consistency: $\forall\, e, f \in E : e \neq f \implies I(e) \cap I(f) = \emptyset$
- Completeness: $\bigcup_{e \in E(v)} I(e) = D(F(v))$

FDD can be used to find an action (rule) for a packet by testing the packet fields from the root to the terminal node. Each non-terminal represents a testing of a particular packet field. The edge containing value that the packet has in its field tested by the node is followed by the next test. A decision path in an FDD, $p = v_1 e_1 \ldots v_k e_k v_{k+1}$, represents the following rule:

$$F_1 \in S_1 \wedge \ldots \wedge F_d \in S_d \rightarrow \langle decision \rangle$$

where

$$S_i = \begin{cases} I(e_j) & \exists\, v_j \in p : L(v_j) = F_i \\ D(F_i) \text{ otherwise.} \end{cases}$$

It means that it is possible to extract a firewall rule directly in a form of predicate by composing constraints along the decision path. All rules thus can be obtained by enumerating all paths in FDD. Note that not all dimension needs to be specified for every rule in FDD. In this case, the interval S_i is equal to the whole interval of the corresponding dimension.

Because of consistency and completeness condition, there is one and only one path for each packet. This property is formalized in the following theorem.

Theorem 2. *FDD unique path for packet matching Let Σ denotes the set of all packets. Denote $permit(f) \subseteq \Sigma$ to be a set of packet classified as accepted by FDD f and $deny(f) \subseteq \Sigma$ to be a set of packet classified as dropped by FDD f. Then, for any FDD f, the following holds:*

$$permit(f) \cap deny(f) = \emptyset \wedge permit(f) \cup deny(f) = \Sigma.$$

Two FDDs are equivalent if they have the same permit and deny sets.

$$\forall\, f, f' \in FDD : f = f' \Leftrightarrow permit(f) = permit(f') \wedge deny(f) = deny(f').$$

Firewall decision diagrams are similar to interval decision diagrams (IDD) employed in the verification of timed systems. However, there are two significant differences:

- Each edge is labeled with a finite set of values from the finite domain.
- The intervals can be used as labels of edges but contrary to IDD the labels in FDD can contain more than a single interval.

The reason for using a particular structure rather than employing existing one, e.g.. IDDs or binary decision diagrams or decision trees, is to provide refined representation for firewall rules that optimizes the representation in terms of a number of simple rules.

In the rest of this section, we focus on the two operations that compute a compact representation of firewall rules suitable for further analysis.

FDD Reduction

Reduction algorithm aims at minimizing number of decision paths in FDD. The minimization is based on removing nodes and edges that are redundant in FDD. To remove redundant information from FDD, we first need to identify which information is not necessary. For this purpose, the isomorphism between nodes is determined.

Definition 17 (Isomorphic nodes). Two nodes v and v' in an FDD f are isomorphic iff:

- Both v and v' are terminal nodes with identical labels.
- Both v and v' are non-terminal nodes and they have all of their edges are identical in terms of their numbers, labels and destination of individual edges.

Reduced FDD does not contain isomorphic nodes. Also, nodes that have only a single outgoing edge are useless because they do not include testing point in the decision procedure. Further, edges between the same pair of nodes can be merged as FDD deals with sets of intervals as edge labels.

Definition 18 (Reduced FDD). An FDD f is reduced iff it satisfies following conditions:

- No node in f has only one outgoing edge.
- No two nodes in f are isomorphic.
- No two nodes have more than one edge between them.

The reduction procedure, which takes an arbitrary valid FDD and produces the reduced FDD is given in Algorithm 1.

FDD Marking

An FDD by definition enables to assign non-consecutive intervals to edges. However, the most firewalls can accept only rules that are simple according the definition bellow.

Definition 19 (Simple rules). A firewall rule for a corresponding path in FDD,

$$F_1 \in S_1 \wedge \ldots \wedge F_d \in S_d \rightarrow \langle decision \rangle$$

input : An FDD f_{input}
output: A reduced FDD $f_{reduced}$ that is equivalent to F_{input}.

for $v \in f[nodes]$ **do**
  if $Count(v[out]) = 1$ **then**
    $e \leftarrow v[out][0]$;
    $v' \leftarrow e[to]$;
    for $e' \in v[in]$ **do** $e'[to] \leftarrow v'$;
    $Remove(f[nodes], v)$
  end
end
for $v, v' \in f[nodes] : v \neq v'$ **do**
  if $Isomorphic(v, v')$ **then**
    for $e \in v'[in]$ **do** $e[to] \leftarrow v$;
    for $e \in v'[out]$ **do** $e[from] \leftarrow v$;
    $Remove(f[nodes], v')$
  end
end
for $e, e' \in f[edges] : e \neq e'$ **do**
  if $e[in] = e'[in] \wedge e[out] = e'[out]$ **then**
    $e[label] \leftarrow e[label] \cup e'[label]$;
    $Remove(f[edges], e')$
  end
end

Algorithm 1: FDD reduction algorithm

is called *simple*, iff $\forall\, i \in (1..d) : S_i$, S_i forms an interval of consecutive integers.

An FDD path that contains non-consecutive labeling generates several simple rules. Rules in a firewall are evaluated using the first-match semantics. Therefore it is possible to reduce the number of rules by using wildcard matching. Thus, we generate more specific rules first followed by rules equipped with wildcards. To decide which rules are suitable for using wildcards, we attempt to evaluate the complexity of FDD with respect to the number of possibly generated rules called *load*.

The idea is to mark exactly one outgoing edge for all non-terminal nodes v labeled with F_i with attribute *all*. Marking the edge with attribute *all* denotes that all values are matched by this edge that can be represented as expression $F_i \in D(F_i)$ in the generated rule. Computing load value helps us to find the most suitable position for *all* attribute.

Definition 20 (Load function). Let $load(s)$ of a set s is the smallest number of intervals that completely cover set s. The load of an edge e, denoted as $load(e)$ is computed as follows:

$$load(e) = \begin{cases} 1 & \text{if } e \text{ is marked all} \\ load(e[label]) & \text{otherwise} \end{cases}$$

Load of FDD is computed recursively starting from a root node as follows:

$$load(v) = \begin{cases} 1 & \text{if } v \text{ is terminal} \\ \Sigma_{i=1}^{k}(load(e_i) \cdot load(v_i)) & \text{if } v \text{ is non-terminal:} \end{cases}$$

An algorithm that computes marked FDD employs load function to determine a suitable outgoing edge from each node. It holds that a number of simple rules generated for FDD f is less that a number of simple rules generated from FDD f' is $load(f) > load(f')$. A procedure for construction of Marked FDD can be found in[24].

Non-overlapping firewall generation

A firewall with non-overlapping rules is easier to analyze than an arbitrary firewall. Thus, the issue of converting an arbitrary firewall to firewall with non-overlapping rules was also studied for FDD representation [56]. The procedure for obtaining non-overlapping firewall is as follows. First, an input possibly overlapping firewall is represented as FDD. Next, the FDD is compacted using the reduction algorithm from [55]. Finally, from the reduced FDD, the non-overlapping firewall is obtained by generating a rule for each decision path.

5.4.3 Multidimensional Interval Tree

Qian, Hinrichs and Nahrstedt Storing propose to represent filtering rules in multidimensional interval tree in [70]. An interval tree is a binary tree which labels its nodes with intervals. The structure of the tree is determined by selecting a center point and branching the intervals with respect to this center point to intervals smaller than the center point forming the left branch and intervals bigger than the selected center point forming the right branch, respectively. Often the median of the boundary points of intervals is used for the root's center point. The tree is constructed recursively until all boundary points occur in the tree.

Each *node* of the interval tree contains the following information:

- value of the center point *node.value*,
- a pointer *node.prev* to node representing a root node of the subtree which contains intervals that are preceding the center point,
- a pointer *node.next* to node representing a root node of the subtree which contains intervals that are following the center point,
- all overlapping intervals *node.lints* sorted by their beginning point, and
- all overlapping intervals *node.rints* sorted by their ending point.

Searching in one-dimensional tree

Interval tree provides an efficient way to find all intervals that contain some specified *point*. The search starts with the root node. The value of this node is compared with the value of the *point.value*. There are three possibilities:

- If *value of the point is less than the value of the node* then the list of intervals sorted by their beginning point is examined, and all ranges $int \in node.lints$ whose starting point is smaller than point's value $int.left \leq point.value$ are gather to the result set.
- The *value of the point is equal to the value of the node*. In this case the search ends and all intervals associated with the node is taken to the result set.
- If *value of the point is greater than the value of the node* then the list of intervals sorted by their ending points is examined, and all ranges $int \in node.rints$ whose ending points is greater than point's value $int.right \geq point.value$ are gathered to the result set.

Searching in multi-dimensional tree

A search method for the multidimensional tree is a direct extension of the search process for one-dimensional trees. The process starts from the first dimension, and if it terminates it uses an edge connecting the terminal node to a tree of the next dimension. Search algorithm in each dimension is the same as the algorithm for one-dimensional tree.

For one-dimensional tree, search, insert and delete operations take $O(log\, n)$ time where n is a number of interval's endpoints. For m intervals there are up to $n = 2m$ endpoints. Tree construction time is $O(n \cdot log\, n)$. The structure occupies $O(n)$ space. The extension to d-dimensions means that search, insert and delete operations take $O(n \cdot log^d\, n)$ time.

Example 2 (Multidimensional Interval Tree). Consider again example of ACL presented in Fig.5.2. The example shows only 2-dimensional interval tree that encodes source and destination addresses. Intervals for the dimension representing source addresses are defined as:

interval		associated rules
I_{srcAdr} :	s_1 = (140.192.37.20..140.192.37.20)	$\{r_1\}$
	s_2 = (140.192.37.30..140.192.37.30)	$\{r_5\}$
	s_3 = (140.192.37.0..140.192.37.255)	$\{r_2, r_4, r_6, r_7, r_9\}$
	s_4 = (140.192.38.0..140.192.38.255)	$\{r_{11}\}$
	s_5 = (0.0.0.0..255.255.255.255)	$\{r_3, r_8, r_{10}, r_{12}\}$

All endpoints are thus defined as:

$$P_{srcAdr}\text{:=}\{0.0.0.0, 140.192.37.0, 140.192.37.20, 140.192.37.30, 140.192.37.255,$$
$$140.192.38.0, 140.192.38.255, 255.255.255.255\}$$

For these endpoints, the interval tree is created as shown bellow. For each mode in this interval tree, the secondary interval tree is computed. Here, the subtree of only one node, named as 140.197.37.20, is presented. This node contains intervals $\{s1, s3, s5\}$, which in turns stand for intervals in rules $\{r_1, r_2, r_4, r_6, r_7, r_9, r_3, r_8, r_{10}, r_{12}\}$. These rules altogether defines following intervals for destination address:

interval	associated rules
I_{dstAdr} : t_1 = (161.120.33.40..161.120.33.40)	$\{r_3, r_4, r_7, r_9, r_{10}\}$
t_2 = (161.120.35.0..161.120.35.255)	$\{r_{11}\}$
t_3 = (0.0.0.0..255.255.255.255)	$\{r_1, r_2, r_5, r_6, r_8, r_{12}\}$

an ordered set of all endpoints is defined as:

$$P_{dstAdr} := \{0.0.0.0, 161.120.33.40, 161.120.35.0, 161.120.35.255, 255.255.255.255\}.$$

The complete interval tree is shown in Fig.5.9. The tree is the compact representation of the example ACL.

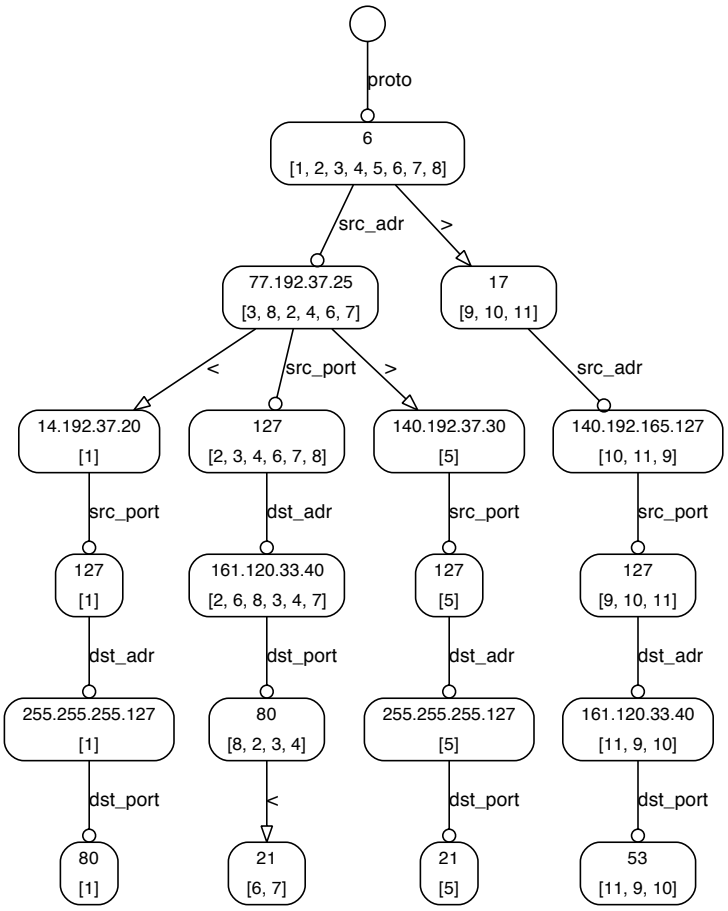


Fig. 5.9 A Multidimensional Interval Tree for example ACL

There are different kinds of edges. Edges ending with arrows are links within the same dimension. Edges ending with discs connect trees in adjacent dimensions. Edges are labeled with relation (less or greater) and dimension names. Node labels contain middle value of the interval in the first line and a set of associated rules that intersect the middle interval in the second line.

5.5 Filter Normalization

In this section, several algorithms for ACL manipulation are presented. These algorithms can be used for further analysis during preprocessing phase, in order to provide a normalized alternative to the original filter. Normalization is understood rather informal as there is not single definition of normalized filter in the literature. In the scope of the present work, we consider that normalized filter represents a filter where all rules are disjoint. To compute the filter with all rules disjoint the algorithms presented in this section can be utilized.

The optimization algorithm removes particular kind of conflicts from the input filter and produces an output filter that is free from redundancy and generalization conflicts. The other presented algorithm computes an action uniform filter from the provided filter and action. This algorithm is appropriate for cases when the filter is used as a predicate, and thus the action of all rules has to be same. It can be shown that such filter have the same matching semantics as the original filter.

5.5.1 Filter Optimization

Filter optimization algorithm by Qian [70] merges adjacent and overlapping rules and removes redundancies. The resulting filter provides the same classification but has a simpler structure. The algorithm computes interval trees for permit and deny actions. The new optimized rule list can be generated by traversing both interval trees simultaneously and generating corresponding rules preserving the rule order. The running time of the algorithm is in the worst-case $O(n^2)$. This worst-case scenario occurs if all rules are pairwise conflicting. In common real-world scenarios, the algorithm behaves much better having complexity $O(n \cdot log^d n)$ considering that a size of rule's conflicting set is constant and rather small [70].

The code of the optimization procedure is presented as Algorithm 2.

The algorithm is capable of removing certain kinds of conflicts from the input filter. In particular, the algorithm removes redundancy and inconsistency (lines 11 and 12), removing generalization (lines 15-18). It also performs merging overlapping and adjacent rules (lines 19-22). In case of generalization and redundancy we need to check the condition that there is not rule z that supports the existence of rule x.

5.5.2 Action-uniform Filters

Certain firewall and configuration analysis method can benefit from the action-uniform representation of firewalls. The algorithm due to Qian et al. [70] that computes from an arbitrary filter a new filter where all rules have the same action is defined as Algorithm 3.

Their algorithm processes all rules that have the required action and checks all intersections with higher priority rules that have a different action. If such rule is found, then the actual rule needs to be modified. The d-dimensional rectangle subtraction is applied to compute a collection of new non-intersecting rules. This set of new rules can be added to the resulting filter preserving matching semantics. Further, the filter optimization algorithm can be executed to compress the set of rules.

input : an input filter f_{in}
output: an output optimized filter f_{out}

```
1  f_permit ← [] ;
2  f_deny ← [] ;
3  for p ∈ f_in do
4      a ← p[action] ;
5      intsec ← fun(x) → Rule.Intersect(p, x) ;
6      contained ← fun(x) → Rule.Includes(x, p) ;
7      s_permit ← Seq.Filter(intsec, f_permit) ;
8      s_deny ← Seq.Filter(intsec, f_deny) ;
9      if a = permit then
10         (f_pos, f_neg) ← (f_permit, f_deny) ;
11         (s_pos, s_neg) ← (s_permit, s_deny)
12     else
13         (f_pos, f_neg) ← (f_deny, f_permit);
14         (s_pos, s_neg) ← (s_deny, s_permit)
15     end
16     if Seq.Exists(contained, s_pos) ∨ Seq.Exists(contained, s_neg) then
17         continue
18     else
19         y ← p ;
20         for x ∈ f_pos ∧ ∄z ∈ f_neg : x[priority] < z.[priority] do
21             if Rule.Includes(p, x) then
22                 remove all x from f_pos
23             end
24             if Rule.Adjacent(y, x) ∨ Rule.Overlap(y, x) then
25                 remove all x from f_pos ;
26                 y ← Rule.Merge(x, y)
27             end
28         end
29         f_pos ← List.Add(f_pos, y)
30     end
31 end
```

Algorithm 2: An algorithm for computing optimized version of a filter (from [70])

Lines 1-4 prepares the environment for running the main loop of the algorithm. A queue for processing rules which intersects with higher priority rules from a filter with opposite action is used to track necessary modifications. The two aliases on input filters are defined. A positive filter f_{pos} equals to the filter which action is the same as the input action. A negative filter f_{neg} is the alias for the other action. Line 4 defines a function that computes an intersection of a given rule with rules from the negative filter. It helps to find all intersecting rules with the higher priority from the negative filter. Lines 5-19 represent the main loop. This code iterates through all rules in positive filter and for each such rule it determines its intersection with the negative filter. The rule is processed in block of lines 8-18, where the rule's intersections are analyzed. In the case of no intersection, it is safe to add the rule to the resulting filter. In other case, all intersections must be

input : a pair of filters (f_{permit}, f_{deny}) and action a
output: a filter f_{out} consisting only of rules with action a

$f_{out} \leftarrow [] ;$
$q \leftarrow [] ;$
$(f_{pos}, f_{neg}) \leftarrow$ **if** $a = permit$ **then** (f_{permit}, f_{deny}) **else** $(f_{deny}, f_{permit}) ;$
$intsec(y) \leftarrow$ **fun**$(x) \rightarrow Rules.Intersect(y, x) \wedge x[priority] < y[priority] ;$
for $y \in f_{pos}$ **do**
  $s_{neg} \leftarrow Seq.Filter(intsec(y), f_{neg}) ;$
  $q \leftarrow Queue.Enqueue(q, y);$
  while not$Queue.IsEmpty(q)$ **do**
    $(q, t) \leftarrow Queue.Dequeue(q) ;$
    $s_t \leftarrow Seq.Filter(intsec(t), s_{neg}) ;$
    if $Seq.IsEmpty(s_t)$ **then**
      $f_{out} \leftarrow List.Add(f_{out}, t)$
    else
      for $v \in Rules.SubtractAll(t, s_t)$ **do**
        $q \leftarrow Queue.Enqueue(q, v)$
      end
    end
  end
end

Algorithm 3: Computing an action uniform filter (from [70])

eliminated using subtraction algorithm. The queue is necessary because rule y can intersect with more rules from s_{neg}. The running time of this algorithm is $O(n \cdot log^d n)$ for n being number of rules in f_{pos} and d being rules' dimension.

5.6 Direct Conflict Checking Algorithm

Several techniques presented in this chapters were employed in our novel conflict checking algorithm implemented by Hozza [36]. In particular, the conflict checking algorithm[1] is based on ideas presented by Baboescu and Varghese [7] and Al-Shaer and Hamed [2]. This algorithm performs efficient conflict detection and reports these conflicts to a user. An algorithm for conflict detection and classification has two phases:

- Detection of potentially conflicting rules. Baboescu and Varghese [7] proposed this method. Rules are inserting in prefix trees constructed for all rule dimensions. In the end, it is possible to extract conflicting vectors that represent potential conflicts between pairs of rules.
- For potentially conflicting rules a conflict classification algorithm is executed. This algorithm classifies conflicts into several categories. We adapt the classification proposed by Al-Shaer and Hamed [2].

[1] Implementation is available from https://code.google.com/p/acl-check/.

In prefix trees, each node contains a vector accumulating the scope of ACL rules. The size of this vector corresponds to the number of rules in the ruleset. Thus, an efficient bit compression scheme for reducing memory necessary to store of these vectors should be implemented.

For reducing memory requirements of prefix tree conflict detection algorithm, bit vectors are compressed. In particular, we use the Word-Aligned Hybrid Bit Vector as developed by Wu and Otoo [89]. The WAH bit vector scheme belongs to the group of run-length encoding methods. It offers efficient implementation of basic logical operations and excellent scalability. It encodes groups of bits aligned to words of the same size. In the encoded vector, there are two kinds of words, distinguished by the most significant bit:

- *literal word*, which represents a sequence of uncompressed bits, and
- *fill word*, which compresses a successive sequence of bits. The second most significant bit denotes value of bits in a compressed sequence.

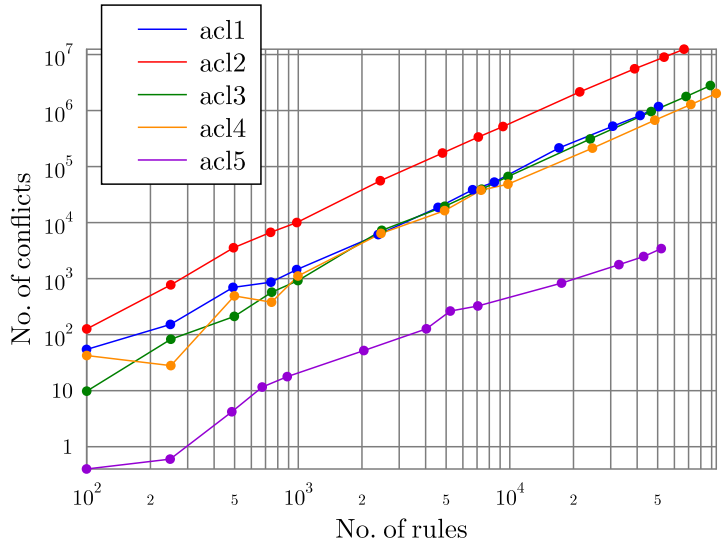
The difference between another similar encoding scheme is that compressed bits should be aligned with words of the size corresponding to the used CPU architecture to improve the overall performance.

5.6.1 Implementation

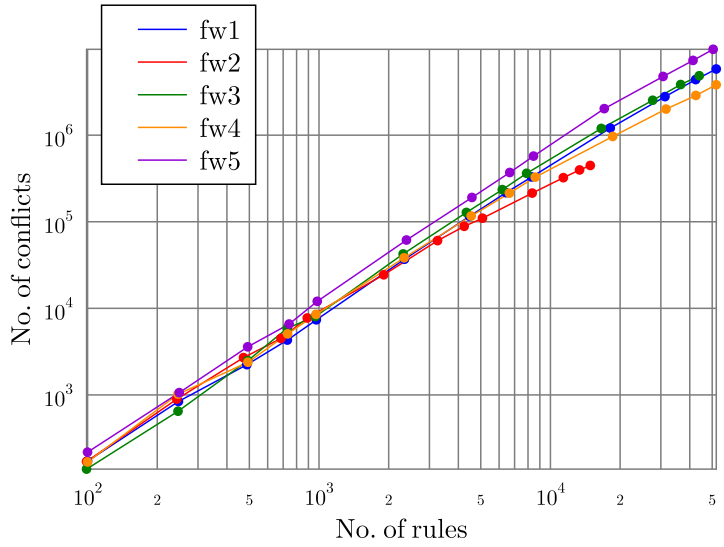
We run experiments to measure performance of implemented tool. The source data for experiments were generated ACLs. The ACL conflict detection tool is implemented using C/C++ language and Boost libraries. The tool implementation is based on previously described algorithm using prefix trees for detecting conflicts among access control list rules. Conflict pairs are determined using conflicts bit vector and classified using conflict classification state machine. The goal of the tool is to implement a practical method for checking ACLs with large number of rules. To speed up the computation and save memory during ACL analysis the Word-Aligned Hybrid (WAH) bitmap compression scheme is utilized. This scheme provides memory-efficient representation while allowing fast bitwise logical operations (AND, OR, ...). A textual configuration file in Cisco, HP or Juniper syntax is accepted by the tool. Output of the tool contains information about detected conflicts in the form of XML file to allow further processing. Input and output blocks have been implemented modularly with standard interfaces therefore it is possible to extend a number of supported vendors and output formats in the future if needed.

5.6.2 Test set

The testing set of filtering rules consists of filters generated using the tool called ClassBench [82]. This generator is equipped with templates of filtering rules derived from a collection of real firewall configurations. The tool creates ACLs of different sizes and parameters. For the testing purpose, we made filters representing different ACL configurations. In particular, we use ClassBench's templates *acl* and *fw*. Among other differences, the *acl* and *fw* templates differ by the ratio of conflicting rules. For every template, a range of filters of various size was generated. For each template and size, we generated 5 different ACLs. Fig. 5.10 shows approximate number of conflicts for firewalls of

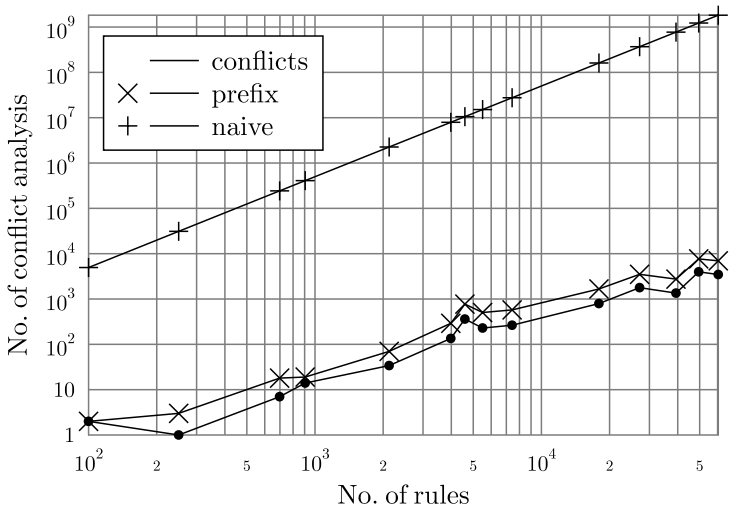


(a) Conflicting rules in rule sets acl1- acl5

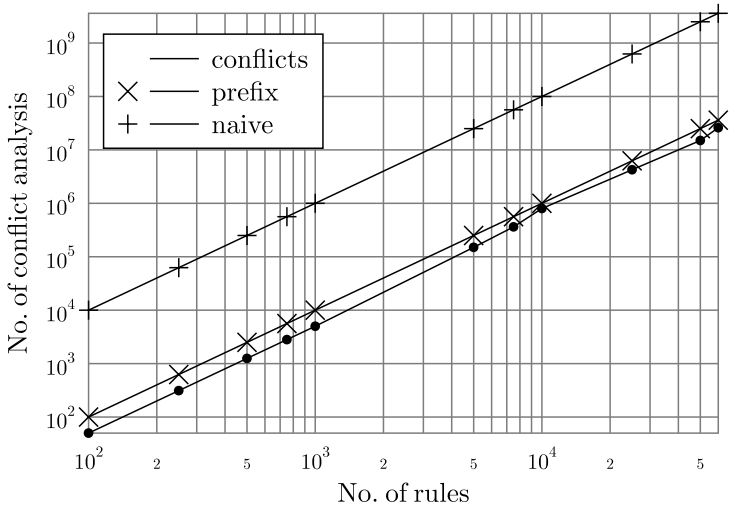


(b) Conflicting rules in rule sets fw1 - fw5

Fig. 5.10 Number of conflicting rules in generated firewalls

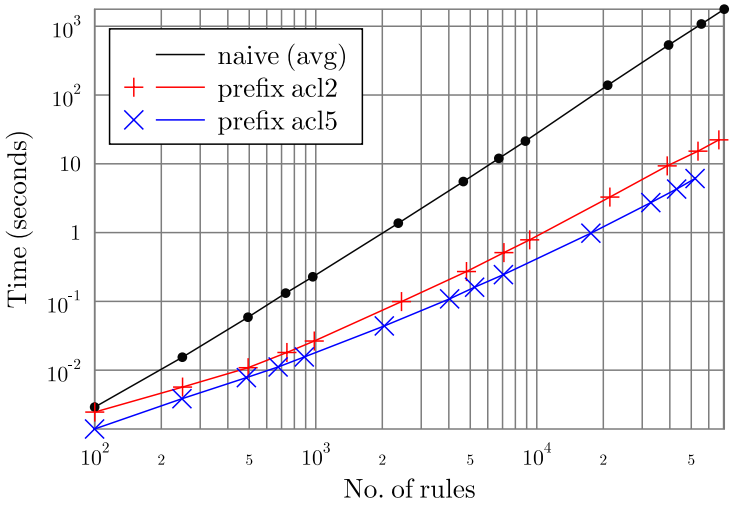


(a) Reduction of conflict analysis for rule set acl5

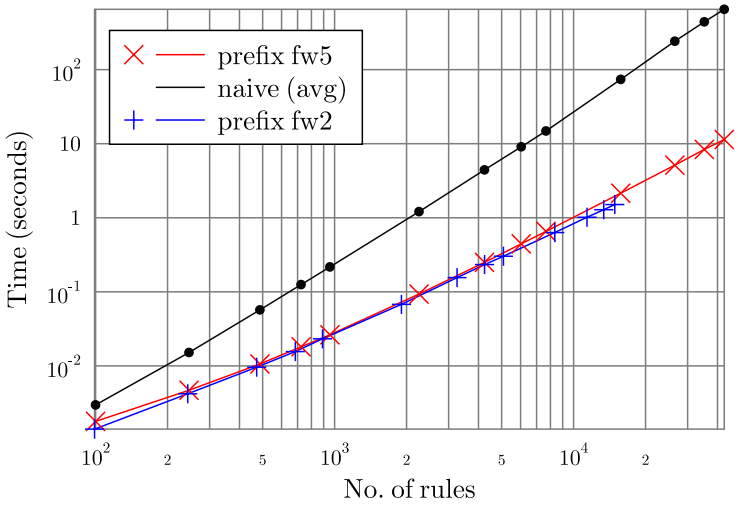


(b) Reduction of conflict analysis for rule set fw5

Fig. 5.11 Reduction of conflict analysis operations



(a) Time required analysis for rule sets acl



(b) Time required for rule sets fw

Fig. 5.12 Time required for conflict analysis

various sizes and the source template. Totally, we created a test set consisting of 960 different ACLs, organized into *acl* and *fw* groups.

5.6.3 Results

Experiments were performed on a 2Ghz dual core machine with 2 GB of RAM with operating system Ubuntu 10.04. The running time was measured only for execution of analysis procedure not including time required for input parsing and XML output generating.

Fig. 5.11 shows the reduction in number of executions of conflict classification algorithm for optimized prefix-based conflict detection algorithm. It shows that there is the necessity to perform a slightly higher number of conflict classifications then there are real conflicts in ACLs. This is because of prefix approximation used for representing ACL dimensions in prefix trees. However, in comparison to the naive algorithm there is a significant difference. These graphs are for acl5 and fw5, which contains, the least and the greatest number of conflicts among generated ACLs. It can be seen that for rules sets containing 0.0002% conflicts the optimized algorithm performs $100,000$ times less comparison than the naive version. As optimized algorithm is sensitive to the number of conflicts, with increasing number of conflicts this ratio decreases. For ACLs containing about 2.7% conflicts the optimized algorithm requires 100 times fewer executions of conflict analysis procedure.

Fig. 5.12 shows comparison of time performance for the naive algorithm and the prefix-based optimized algorithm. Results for the optimized algorithm are displayed only for two selected firewalls. The rest falls within the boundaries given by the curves of these firewalls.

The algorithm evaluation shows that proposed optimization based on the indexing rules using WAH bit vectors provides more efficient way of detecting firewall conflicts. The implementation scales to checking 10,000 rules within a second.

5.7 Chapter Summary

Filtering rules are widely employed mechanisms for implementing network security. This chapter presented various methods for analysis of firewalls, in particular, conflicts between firewall rules. The survey of the current state of conflict checking methods and the firewall rule reorganization was presented. Firewall representation and conflict checking problem was formalized using Formula system. Also, a novel algorithm for efficient detection of conflicts in access control rules was described. Evaluation of this algorithm shows that complete conflict analysis of a rule base containing 10^5 rules does not exceed 10s. Except analysis of consistency of firewall rules, the goal of this chapter was to overview of methods that can be used to preprocess ACL rules in order to obtain a suitable representation suitable for network configuration analysis. Logic-based methods for configuration analysis often require that there be no conflicts in ACL as conflict resolution mechanisms would make analysis more complicated.

Chapter 6
Network Configuration Analysis

In this chapter, the approach based on constrained relations for modeling and analysis of network reachability is presented. A constraint model is created for the analyzed system such that constraints represent packet filters as well as packet transformations. An example of packet transformation is a network address translation or packet encapsulation. The situation is a bit complicated as translations frequently have session flow semantics. A session consists of flow of packets that are in inbound and outbound directions. For instance of an ICMP echo request query, the session includes the request outbound packet flow and the reply inbound packet flow. The direction of a session is determined by the first packet sent in the session.

Flow-based description of network communication presents benefits for specification of network functionality. For instance, flow-based approach is central to the software defined network paradigm. In SDN, a flow is the fundamental component of network communication that can be controlled. Language Procera [85] introduces flow constraint functions to represent the behavior of network nodes. This language is intended for describing a system policy for OpenFlow networks.

Narain, Talpade and Levin [61, 64, 65] developed a method for validation of network configurations against end-to-end requirements into the configuration verification system called ConfigAssure. Validation of requirements is based on application of logical programming and constraint solving. The benefit of configuration validation is illustrated on a system with decentralized administration, which is inherently vulnerable to configuration errors. Properties that can be analyzed includes GRE and IPsec tunnels, consistent addressing scheme, OSPF and BGP routing configuration, and MTU settings. ConfigAssure consists of a configuration acquisition subsystem for extracting configurations from components, requirement library that collects typical end-to-end requirements, specification language used for writing requirements specifications and the evaluation component that checks the configuration against the active requirements. ANTLR parser performs extracting information from a configuration file. It skips parts of the configuration that has no meaning for the analysis. The data are inserted into the database. ConfigAssure can handle several category of requirements. Requirements on integrity of logical structure, that are checked by the tool span IPSec and GRE configuration, HSRP configuration, iBGP peering, OSPF areas and MPLS tunnels. Connectivity requirements evaluated by the tool contains IP, VLAN, GRE, IPSec, BGP and MPLS. ComfigAssure can check reliability requirements by verifying an absence of single points of failure in IP network, existence of multiple OSPF area-border-routers and replication of IPSec tunnels. The tool verifies an implementation of IPSec configuration and ACLs as a part of security requirements. The tool also evaluates performance requirements by checking that all

DiffServ policies on all routers are identical. To validate configuration, the user can specify the required properties by instantiating a predefined template requirement. Evaluation system employs graph algorithms for the most of the operations. For checking firewall configurations, a symbolic method is applied. Extended configuration validation employs constraint representation over finite domains and can be automatically solved by a constraint solver, e.g. Kodkod. For requirements checking Prolog program is executed. Furthermore, ConfigAssure offers to suggest the configuration corrections if checking fails. This task is implemented with the help of the constraint solver. Alternatively, MuVal can provide similar functionality by pointing at problematic configuration statements that contribute to requirements violation. While more than 100 requirement templates are implemented in a system, it is declared that further development is necessary before network practitioners may accept the system. Suggested future research consists of increasing robustness of configuration acquisition systems, formalizing requirements in a Requirement Library, adding new classes of requirements, which needs to be supported by extending capabilities of a requirement specification language.

Al-Shaer, Marrero, El-Atawy and ElBadawy [5, 6] describe a tool called *ConfigChecker* that allows for reachability and security property verification. The ConfigChecker maintains a description of a network dynamics by modeling the forwarding process as a state machine. Each state is defined by occurrences of traced packets at specific network locations. Employing symbolic representation, this approach is practically feasible. Reachability requirements are expressed in CTL and evaluated by a symbolic model checker. The finite state machine of a network describes locations of packets. A state of the network machine is encoded as $\sigma : packet \times location \rightarrow \{true, false\}$, where *packet* is an abstract specification of a packet. The behavior of devices are defined in terms of packet transformation, e.g., NAT changes information in the packet header and its location in the network. This change is characterized by a Boolean formula. Using this approach it is possible to uniformly denote usual operations of network devices as (possibly large) Boolean formulae. By exposing more information on packets, one may encode other operations. For instance, encapsulation transformation carried by IPSec takes an incoming packet and put it into a new IPSec packet. Suitable packet model has to include outer, and inner header fields. The method is implemented by using BDD for encoding Boolean formulae that represent transformation relations. A standard CTL model checking algorithm can evaluate the required end-to-end reachability properties. Direct CTL queries verify properties, such as the possibility that the packet p reaches location l, expressed as $EF(location = l)$. A security analysis aims at verification of the access control requirements. The input is a set of all allowed flows between two locations, u and v. A configuration expressed by the network state machine is evaluated against the set of requirements that represent a set of authorized paths. The configuration is correct if all possible paths from u and v are subset of authorized paths. Thanks to expressiveness of CTL, it is possible to verify other non-trivial properties, such as

- the absence of routing loops,
- shadow or bogus routing entries, which stands for routing decisions that will never be applied as it is impossible for traffic that would fire these rules to reach the router,
- integrity of IPSec tunnels, including nested or cascaded tunnels, and
- the absence of backdoors or broken flows that may happen after route changes.

Authors implemented a prototype that was tested on more than 90 networks, and its performance was evaluated. They claim that results prove the feasibility of the method for the practical applica-

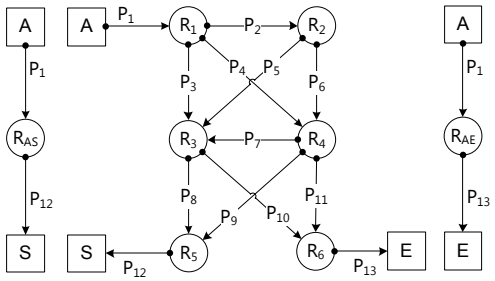


Fig. 6.1 An example of network for configuration analysis

tions. Building a model of a network with thousands of nodes requires tens of seconds and grows linearly.

Following sections provide development of a unified system for configuration modeling by constrained relations. From many possible configuration features, access control lists, network address translation, type of service marking, tunneling and routing are examined. Configuration verification consists of two parts:

- checking that the configuration obeys rules for configured features, e.g., setting the same crypto parameters at both sides of IPSec channel. This part is performed by proving conformance requirements. The conformance requirements are provided by the domain modeling the configuration feature.
- validating that the configuration satisfies reachability requirements. This part is evaluated by interpreting the configuration in reachability model and performing computations as presented in Chapter 3.

6.1 Access Control Lists

Access control lists (ACL) contains rules that classify flows. ACL can be used for traffic filtering (as utilized in Chapter 3 for illustrating filter relations) or in various other networking mechanisms. An ACL selects the traffic to which some other operation is applied, e.g., selecting source traffic to be tunneled. For configuration modeling, inconsistent ACLs represent the challenge. Thus, all ACLs should be preprocessed to represent only consistent ACLs (for instance, method due to Liu [56] can be utilized). As an example, we consider the following ACL.

```
permit icmp any any echo-reply
permit icmp any any echo
deny ip any 10.10.10.0 0.0.0.255
deny ip any 10.10.11.0 0.0.0.255
permit ip any any
```

This ACL configuration permits clients to use ping for troubleshooting and denies the access to network infrastructure and management area.

Constraints can be inferred from ACL rules by considering individual fields as numeric intervals. The previous ACL is represented as four rows in constraint table P:

P				
id	pt	dst.ip	dst.pn	constraint
P_1	icmp	x	p	$p = 0x800$
P_1	icmp	x	p	$p = 0x0$
P_1	ip	x	p	$x < 10.10.10.0$
P_1	ip	x	p	$10.10.11.255 < x$

Deny rules split the network address space into intervals, which are specified by row constraints. In our model, we assign a set of permitted packets P to each edge in a graph of network topology. Considering the example network from Fig.6.1, set P_3 is computed using the previously described approach from an ACL, which is the concatenation of an outbound $\mathsf{ACL}_{\mathsf{OUT}}$ applied on the interface of R_1 and an inbound $\mathsf{ACL}_{\mathsf{IN}}$ applied on the interface of R_3.

ACL matching process has the first match semantics, which means that rules are ordered, and packet is examined for rules in that order until the match is found. It is possible and often the case in practice to write ACL that are not consistent. An ACL is inconsistent if more that one rule can match the same packet. However, to represent ACLs as constraints we wish to have a consistent set of rules. A weak consistency, which means that a packet cannot be matched by permit and deny rules at the same time, is sufficient. To obtain more suitable ACL, the normalization is computed. The resulting set of predicates that was obtained from ACL normalization consists of four items:

```
[action=permit,pt=IP,
 srcIp=[0.0.0.0-255.255.255.255],
 dstIp=[0.0.0.0-10.10.9.255],
 srcPn=[0-65535],dstPn=[0-65535]]

[action=permit,pt=ICMP,
 srcIp=[0.0.0.0-255.255.255.255],
 dstIp=[10.10.12.0-255.255.255.255],
 srcPn=[0-65535],dstPn=[0-65535]]

[action=permit,pt=ICMP,
 srcIp=[0.0.0.0-255.255.255.255],
 dstIp=[0.0.0.0-255.255.255.255],
 srcPn=[0-65535],dstPn=[0-0]]

[action=permit,pt=ICMP,
 srcIp=[0.0.0.0-255.255.255.255],
 dstIp=[0.0.0.0-255.255.255.255],
 srcPn=[0-65535],dstPn=[2048-2048]]
```

First two items were computed from deny rules by excluding their destination address intervals. The last two items are associated to rules permitting ICMP flows. A set of deny predicates is computed by the same method but add operation is executed for denying rules and subtract operation is executed for permit rules.

```
[action=deny,pt=IP,
```

```
 srcIp=[0.0.0.0-255.255.255.255],
 dstIp=[10.10.10.0-10.10.10.255],
 srcPn=[0-65535],dstPn=[0-65535]]

[action=deny,pt=IP,
 srcIp=[0.0.0.0-255.255.255.255],
 dstIp=[10.10.11.0-10.10.11.255],
 srcPn=[0-65535],dstPn=[0-65535]]
```

Denying rules represent the complement of permit rules, thus checking `acl(A,F)` for an arbitrary flow `F` should yields true. An arbitrary rule of a normalized ACL

$$[action = a, pt = p, src.ip = s, dst.ip = d, src.pn = r, dst.pn = q]$$

is represented as a clause of predicate `AccessList`. Based on this, constraint `acl/3` is defined to match specified flows:

```
AccessList ::= new (id:Id, action:Action, proto:Protocol, srcIp:IpRange, srcPn:
    PortRange, dstIp:IpRange, dstPn:PortRange).

acl ::= (list:Id, action:Action, f:Flow).
acl(list,action,flow) :-
    AccessList(list,action,proto,srcIp,srcPn,dstIp,dstPn),
    protocol_match(proto,flow.pt),
    IpRange.in(flow.srcIp,srcIp),
    IpRange.in(flow.dstIp,dstIp),
    PortRange.in(flow.srcPn,srcPn),
    PortRange.in(flow.dstPn,dstPn).
```

Predicate `protocol_match(proto1:Protocol,proto2:Protocol)` is true when protocol `proto1` carries as its payload protocol `proto2`. For instance, the following is valid:

```
protocol_match(IP,ICMP).
protocol_match(IP,GRE).
protocol_match(IP,TCP).
protocol_match(TCP,HTTP).
...
```

It can be seen that the limitation of the presented approach is in the representation of addresses and port values in ACL rules as intervals. However, this is sufficient in most cases as many ACL configurations use wildcard masks specifying only intervals. To enable arbitrary wildcard masks, as set of intervals are computed to cover all possible combinations. The extension of this method to cope with arbitrary wildcard mask for address definitions in ACL rules is straightforward.

6.2 Network address translation

Network address translation (NAT) translates traffic coming into and leaving the private network. There are several types of NAT. Static and dynamic NATs map private addresses to public addresses

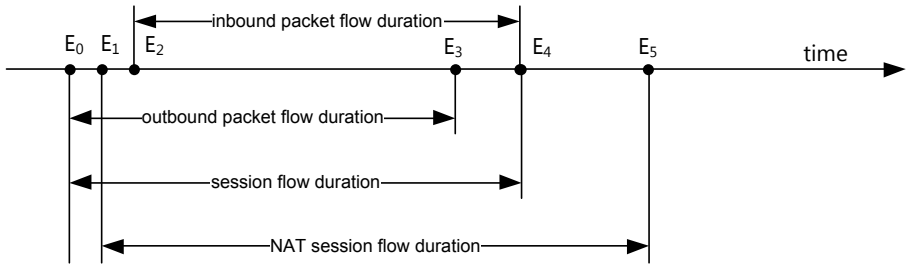


Fig. 6.2 A sequence of events in NAT session lifetime

The following events can be identified:

- E_0 - the first outbound packet leaves its originating node, this is the beginning of session flow,
- E_1 - a NAT session is established for a new session flow,
- E_2 - the first inbound packet is created and sent by target node. Since this time the session flow consists of bidirectional packet flow,
- E_3 - the outbound flow is terminated, the session flow is only unidirectional if $E_3 \prec E_4$. In other case the session terminates.
- E_4 - the inbound flow is terminated, and
- E_5 - the NAT session is terminated, most often because of timer expiration as there are not flows belonging to the session that causes creating the NAT session.

on a one-to-one basis. Overloaded NAT maps multiple private addresses to a single address by using different ports. NAT interpretation was used in Chapter 3 for illustrating transform relations. Here, the more advanced model is developed showing how three different NAT mechanisms can be uniformly represented.

First, an informal description of NAT semantics according RFC 2663 is given. The NAT semantics is interpreted in terms of session model. The session of NAT is not necessary equal to application-specific session, e.g., HTTP session can consist of multiple TCP connections and NAT session model deals with each of this TCP connection separately. Moreover, a NAT session can timeout while application session is still active.

A NAT session begins with the occurrence of the first outbound packet. A NAT device then must be able to provide forward and reverse address translation. The detection of the end of a NAT session is not trivial. It requires the use of timers and/or heuristics based on the analysis of application protocols behaviors. Figure 6.2 shows the timeline with possible occurrences of NAT related events.

We say that a NAT is well-behaved if the session is available for a specified period of the corresponding session flow. Formally, a *well-behaved NAT* have to satisfy these properties:

- $E_0 \prec E_1$ - a NAT session is not created before the first outbound packet reaches the NAT device,
- $E_1 \prec E_2$ - the NAT session has to be already established before first inbound packet reaches the NAT device, which is important for the reverse translation, and
- $E_4 \prec E_5$ and $E_3 \prec E_5$, the NAT session can be terminated after the corresponding session flow ends.

In the rest of this section, only well-behaved NATs are considered. This assumption simplifies the reasoning about the configuration. First static NAT is examined, which represents the simplest case. Then, dynamic NAT is investigated. Finally, a model for dynamic NAT with overloading is provided.

6.2.1 Static NAT

A static NAT requires that there be one-to-one static address mapping. The following listing represents an example of a typical static NAT configuration: R_2:

```
ip nat inside source static 172.16.200.48 12.34.56.150
```

It will translate any packet with private source address *172.16.200.48* to a packet with public source address *12.16.47.150*. This rule is represented by the following constraint relation:

R			
id	in.src.ip	out.src.ip	constraint
R_2	10.20.22.13	12.34.56.150	
R_2	x	x	$x \neq 10.20.22.13$

The constraint relation for R_2 consists of two rows. The first row defines intended translation, the second row matches all other packets that are not translated by NAT.

Flow-based analysis, which is sufficient in case of the static NAT, can only verify that a packet from a private network can reach the public network either translated by NAT or unchanged if its source address does not match in.srp.ip of the NAT translation. Similarly, for packets in the opposite direction it is tested if their source address matches out.src.ip of the NAT translation.

The encoding of static NAT translation from the previous example is given as follows:

```
1  nat_translation ::= (dev:Device,kind:NatKind,dir:Direction,input:Flow,output:
       Flow).
2
3  nat_translation("R2",STATIC,FORWARD,inflow,outflow) :-
4    inflow.srcIp =  10.20.22.13, outflow.srcIp =  12.34.56.150,
5    flow_eq([PT,DSTIP,SRCPN,DSTPN,TOS],inflow,outflow).
6
7  nat_translation("R2",STATIC,FORWARD,flow,flow) :-
8    flow.srcIp != 10.20.22.13.
9
10 nat_translation("R2",STATIC,REVERSE,inflow,outflow) :-
11   outflow.dstIp = 10.20.22.13,
12   inflow.dstIp = 12.34.56.150,
13   flow_eq([PT,SRCIP,SRCPN,DSTPN,TOS],inflow,outflow).
14
15 nat_translation("R2",static,REVERSE,flow,flow) :-
16   flow.dstIp != 12.34.56.150.
```

There are four clauses that express forward and reverse transformations. Two of them capture the case when transformation is in effect. The other two define that flows with different source address

are not translated. The next section deals with the dynamic address translation where exact pairs of private and public addresses of each transformation are not known.

6.2.2 *Dynamic NAT*

A dynamic one-to-one NAT maps private address to an available public address from a pool of NAT addresses. In a dynamic mapping, the public addresses can be used for different sessions as the address is recycled after the session ends. The exact behavior of this mechanism is implementation dependent. Therefore, only assumption that NAT is well-behaved can be made.

Address binding associates the local address of a host with an assigned external address. The binding is done if the first packet of a session reaches the NAT device. If there are other sessions originating from or to the local host, then these sessions will use the same binding. There can be many simultaneous sessions originating from the same host that will use a single address binding, which obeys the principle of one-to-one dynamic mapping.

If a NAT device make a decision that none of the sessions from a local host is active, it can perform an address unbinding which frees the external address for another binding with possibly different local hosts.

The following configuration snippet contains a definition of the pool of public addresses, dynamic NAT setting, and an access control list that together represent dynamic NAT configuration.

```
ip nat pool NAT_POOL 12.34.56.1 12.34.56.250
ip nat inside source list NAT-ACL pool NAT_POOL
access-list NAT-ACL permit 10.20.0.0 0.0.0.255
access-list NAT-ACL permit 10.22.0.0 0.0.0.255
```

The constrained relation, which represents dynamic NAT consists of five rows. Every ACL rule is represented by a single row in the constraint relation. The last three rows match other packets that are from other source addresses.

R		
in.src.ip	out.src.ip	constraint
x	y	$10.20.0.0 < x < 10.20.0.255$ $12.34.56.1 \leq y \leq 12.34.56.250$
x	y	$10.22.0.0 \leq x \leq 10.22.0.255$ $12.34.56.1 \leq y \leq 12.34.56.250$
x	x	$x < 10.20.0.0$
x	x	$10.20.0.255 < x < 10.22.0.0$
x	x	$10.22.0.255 < x$

Admittedly, the representation does not precisely capture “dynamics" of the NAT. Instead, it expresses that the source address of a packet from one of the specified networks will be translated to one of the addresses from NAT pool. This representation approximates the NAT behavior at the level provided by the reachability model.

To check if communication is possible between devices behind NAT and public service, one needs to verify the reachability in the opposite direction. For packets outbound from the private network, the source IP address is translated. For the corresponding inbound packets, the destination

IP address is translated. The session consists of packets that are part of the same communication. These packets travel in two directions. The outbound packets are sent from the client to the server. The inbound packets are sent from the server to the client. Any NAT translation respecting a session model has to satisfy the following correctness property:

$$\forall x_o, y_o, y_i \in Flow : nat_{fwd}(x_o, y_o) \wedge session(y_o, y_i) \implies \exists x_i \in Flow : nat_{rev}(y_i, x_i) \wedge session(x_o, x_i) \quad (6.1)$$

where

- $nat_{fwd}(x_o, y_o)$ is a forward NAT transformation that maps flow x_o to flow y_o,
- $nat_{rev}(y_i, x_i)$ is a reverse NAT transformation that maps flow y_i to flow x_i,
- $session(x_o, x_i)$ is true when x_o and x_i belongs to the same session, and
- $session(y_o, y_i)$ is true when y_o and y_i belongs to the same session.

This property requires that the translation if exists, is consistently defined for both flows that compound a session. The consistency means that the transformation of forward translation makes from outbound flow x_o an outbound flow y_o. If y_i is an inbound flow for some outbound flow y_o, then reverse translation has to transform it to a flow x_i, such that it is an inbound flow that forms with x_o a single session.

For consistent sessions, it is important that both outbound and inbound flows traverse the same NAT device. This can be verified by checking the existence of a path from a target network to the NAT's outside interface. As packets of the inbound flow targets NAT's public interface this verification amounts to check routing configuration consistency. This situation is similar to the problem of keep state rules in firewalls as discussed in [8].

The representation of a dynamic NAT translation from the previous example is as follows:

```
nat_translation('DynamicNAT',FORWARD,x,y) :-
    acl('NAT-ACL',PERMIT,x),
    flow_src_ip(y,ip),
    nat_pool('NAT-POOL',INCLUDES,ip),
    x.pt = y.pt,
    x.dstIp = y.dstIp,
    x.srcPn = y.srcPn,
    x.dstPn = y.dstPn.
```

The ACL is encoded following the pattern defined in the previous subsection. The address pool is represented by two clauses of `nat_pool(Name,Type,Ip)`:

```
nat_pool('NAT-POOL',INCLUDES,x) :- x >= 12.34.56.1, x <= 12.34.56.250.
nat_pool('NAT-POOL',EXCLUDES,x) :- x < 12.34.56.1, x > 12.34.56.250.
```

The first clause assigns to x addresses from the NAT's address pool, while the second clause assigns to x their complements. The pool of excluded addresses is used in `nat_translation/4` that expresses flows untouched by the translation. First, clause for forward translation is defined. Packets are not transformed by forward translation if they are matched by the complement of `NAT-ACL`, which has a direct representation:

```
nat_translation('DynamicNAT',FORWARD,flow,flow) :-
```

```
    acl('NAT-ACL',DENY,flow).
```

A reverse translation matches inbound flows, whose destination addresses belong to the address pool. These flows are translated to inbound flows for which the NAT's ACL should match the corresponding outbound flows as implied by the NAT correctness property (6.1). Thus, the reverse translation can be represented in terms of forward translation:

```
nat_translation('DynamicNat',REVERSE,x,y) :-
    session(y,yr), session(x,xr),
    nat_translation('CEInternet',FORWARD,xr,yr).
```

Finally, the case when inbound flows are not translated occurs. These flows can be identified by checking destination addresses, which should not be in the range of a NAT's address pool.

```
nat_translation('DynamicNat',REVERSE,flow,flow) :-
    flow_dst_ip(flow,dstIp), nat_pool('NAT-POOL',EXCLUDES,dstIp).
```

These four clauses of `nat_translation/4` completely define mapping of inbound and outbound flows as performed by the dynamic NAT for the example. In the next section, a network address port translation (NAPT) is elaborated. NAPT device maps multiple private addresses to a single public address by modifying source IP address and if necessary also source port.

6.2.3 Network Address Port Translation

Network Address Port Translation (NAPT), also called NAT overloading, allows mapping numerous private addresses to a single external address. In the case of a clash, the transport identifiers are also translated, e.g., TCP and UDP port numbers and ICMP query identifiers.

For an outbound packet flow, the NAPT translates the source IP address, the source transport identifier and other related fields such as header checksums. For inbound packets, the NAPT translates the destination IP address, the destination transport identifiers, and other affected fields.

A configuration bellow is used to demonstrate the constraint representation of NAPT mechanisms:

```
ip nat pool NAT_POOL 12.34.56.251 12.34.56.254
ip nat inside source 7 pool NAT_POOL overload
access-list 8 permit 10.21.0.0 0.0.255.255
```

In the case of NAPT, a constraint relation is extended with an attribute specifying the source port number. It is possible to specify additional constraints on translation, e.g., selection of port numbers based on appropriate port groups (0-511, 512-1023, or 1024-65535).

R				
in.src		out.src		
ip	pn	ip	pn	constraint
x	p	y	q	$10.21.0.0 \leq x \leq 10.21.255.255$ $12.34.56.251 \leq y \leq 12.34.56.254$
x	p	x	p	$x < 10.21.0.0$
x	p	x	p	$10.21.255.255 < x$

The representation of the NAPT is similar to previously described dynamic NAT. The only difference is in the representation of the forward translation relation. The exception is that source port can differ in the translated packet. Constraints on source port numbers only impose requirements that port numbers should be in the same port groups, which is the standard behavior of NAPT implementations.

```
nat_translation('NAPT',FORWARD,x,y) :-
    acl('NAT-ACL',PERMIT,x),
    flow_src_ip(y,ip),
    nat_pool('NAT-POOL',INCLUDES,ip),
    flow_eq([pt,dst_ip,dst_pn],x,y),
    flow_src_pn(x,g1),flow_src_pn(y,g2),
    ( g1 in 0..511, g2 in 0..511
    ; g1 in 512..1023, g2 in 512..1023
    ; g1 in 1024..65535, g2 in 1024..65535).
```

Three kinds of network address translation mechanism were defined in the form of predicate `nat_translation/4`. Predicate `transform/5` defines an overall functionality of the router represented as the packet flow transformation between input and output interfaces of a router. To "deploy" NAT configuration, `transform` clause is specified. It builds the transformation relation from a NAT configuration.

The network address translation is implemented on a router by specifying a group of inside interfaces and a group of outside interfaces as shown in the following configuration snippet:

```
interface Serial0
 ip address 172.16.47.161 255.255.255.0
 ip nat inside
interface Serial1
 ip address 172.16.48.161 255.255.255.0
 ip nat inside
interface Serial2
 ip address 172.16.49.146 255.255.255.0
 ip nat outside
```

In this example, `Serial0` and `Serial1` are inside interfaces and `Serial2` is a single outside interface of the NAT. If a flow enters the NAT inside interface and is routed to the NAT outside interface, then the NAT ACL is checked whether to perform translation or not. If the input flow matches ACL, it is translated according the kind and rules of the NAT configuration.

For each inside interface `nat_interface(Device,Interface,inside)` is defined in the model. For each outside interface corresponding `nat_interface(Device,Interface,outside)` is defined in the model. These statements denote interfaces that only can participate in NAT transformation.

Considering NAT the only transformation offered by devices allows to define `transform/5` predicate as follows:

```
transform ::= (d:Device,l:Interface,r:Interface,p:Flow,q:Flow)
transform(d,l,r,p,q) :-
    idge(d,l,r),
    nat_interface(d,l,INSIDE),
    nat_interface(d,r,OUTSIDE),
```

```
6     nat_translation(d,FORWARD,p,q).
7
8  transform(d,l,r,p,q) :-
9     idge(d,l,r),
10    nat_interface(d,l,OUTSIDE),
11    nat_interface(d,r,INSIDE),
12    nat_translation(d,REVERSE,p,q).
13
14 transform(d,l,r,p,q) :-
15    idge(d,l,r),
16    (no nat_interface(D,L,_) ; no nat_interface(D,R,_)).
```

This section presented a formalization of various NAT mechanisms in the form of transform relation. The presented approach made some assumptions. For instance, only static properties were addressed. Also, routing has not been considered in the model. The relation of routing and NAT is clearly described in [15]:

> When a packet traverses from inside to outside, a NAT router checks its routing table for a route to the outside address before it continues to translate the packet. Therefore, it is important that the NAT router has a valid route for the outside network. The route to the destination network must be known through an interface that is defined as NAT outside in the router configuration. The return packets are translated before they are routed. Therefore, the NAT router must also have a valid route for the inside local address in its routing table.

The presented model is extended with routing at the end of this chapter. The definition of the transform rule is adapted accordingly to provide correct order of NAT and routing operations.

6.3 Constraint Queries

This section gives a set of queries that utilizes constraint reachability model for configuration analysis. The analysis method is similar to methods defined for the Flow-based Management Language (FML) [34] and the ConfigChecker [6].

For the evaluation of queries, a simple routing assumption is considered. Packets cannot pass the same device multiple times, and routing provides end-to-end reachability. The presented approach allows us to answer reachability queries considering different available paths. If necessary, it is possible to restrict the set of analyzed paths by specifying additional assumptions.

The following operations defined in Chapter 3 are used in queries:

- $\textsc{reach}(\pi, p, q)$ is an input to output packet flow relation of path π, and
- $\textsc{path}(m, n)$ is a set of all paths from location m to location n.

Also, all queries consider that reachability model M is provided. This model can be computed in Formula as transformation from *Configuration* domain to *Reachability* domain. It is possible to evaluate the following queries:

- Will a packet p, originating from a node m, reach a destination node n? Formally written as:

$$q_1(p) := \exists \pi \in \text{PATH}(m, n) : \exists q \in P : \text{REACH}(\pi, p, q)$$

The question ask whether there exists at least a single path that permits delivering packet flow p. As we admit packet transform, which can modify packet fields when packet is forwarding, a packet delivered to the target location, denoted as q, can be different to the original packet. In FORMULA , the question can be encoded as following:

```
query M path(source,target,path), reach(path,p,q).
```

Here `Path` is implicitly existentially instantiated and FORMULA is able to find all paths that can carry packet flow p. Also it may be possible to rewrite the predicate so that `q` would be visible.

- Can some packet be delivered along a path π? It means, isn't this path of kind deny-all? Formally written as:

$$q_2(\pi) := \exists p \in P, \text{REACH}(\pi, p, p)$$

Again FORMULA can not only check if the path is not void, but can also come up with an enumeration of permitted packets:

```
query M reach(path,packet,packet).
proof
...
proof tree
...
```

Usually, it is more relevant to ask whether certain packets can reach the target network considering the provided path:

- Can packets with property Q be delivered along the path π? Formally written as:

$$q_3(\pi) := \forall p \in P, Q(p) \implies R(\pi, p, p)$$

This query is analyzed by specifying a set of flows and finding a path delivering all flows from this set. The *count* aggregation operation is employed to represent this query in FORMULA . All packets satisfying Q is enumerated using set comprehension. This enumeration is then used for testing if the path π can deliver all packets.
For instance, it can be check if there is a path delivering all ICMP packets, by the following query:

```
query count{flow | flow_pt(flow,icmp)} = count({ flow | flow_pt(flow,icmp,
  flow_reach(path,flow,flow)}.
```

The query evaluates to true when both sets have the same size, that is, for every packet, there exists a path in a network. Formula $count(\{x|p(x)\}) = count(\{x|p(x) \wedge q(x)\})$ is equivalent to Prolog's predicate `foreach(Generate,Goal)` that is true if the conjunction of results is true.

Similarly to definitions in [6], it can be check whether the configuration is sound and complete:

- A *configuration is sound* if possible reachability paths are subset of authorized paths.

- A *configuration is complete* if authorized paths are subset of reachable paths computed from the configuration.

All the cases above consider reachability in the form of $\text{REACH}(\pi, p, p)$, which is nevertheless too restrictive, in general. Regarding network address translation, the resulting packet q will not be exactly the same as originating packet p. Generalized shape of queries will have the form of:

$$\exists \pi \in \text{PATH}(m, n), \forall p \in P, A(p) \wedge G(q) \implies \text{REACH}(\pi, p, q).$$

where $A(p)$ is a predicate stating *assumed properties* on a packet p and $G(q)$ is a predicate stating *asserted properties* on packet q.

This statement is satisfied if there exists a path, which transfers assumed packets to the destination location yielding asserted packets. Following formula checks that every possible path can be used for data transfer:

$$\forall \pi \in \text{PATH}(m, n), \forall p \in P, A(p) \wedge G(q) \implies R[\pi](p, q).$$

Nevertheless, in practice an interesting checking could be whether a service is guaranteed under some assumption on connectivity. It means not only to define properties for packets, but also to consider properties restricting available paths. Consider S is a predicate that selects such paths. The formal definition is as follows:

$$\forall \pi \in \text{PATH}(m, n), \forall p \in P, S(\pi) \wedge A(p) \wedge G(q) \implies R(p, q).$$

This expression can be directly represented as FORMULA query. The query enumerates all paths together with input and output flows that satisfy the path constraints *pathConstraint*, assumption predicate *flowAssume* on inbound flows and assertion predicate *flowAssert* on outbound flows.

```
query M path(source,target,path), reach(path,flowIn,flowOut),
        pathConstraint(path),
        flowAssume(flowIn),
        flowAssert(flowOut).
```

The presented queries are used in the next section to illustrate reachability analysis of GRE configuration. Queries are refined by specifying which routers should be included in forwarding paths (waypoint property) and which should be excluded (forbidden property). Finally, we discuss validation of QoS properties based on rate limitation.

6.3.1 Tunnel Configuration Validation

Generic Routing Encapsulation (GRE) configuration enables creating virtual connections between intermediate devices by encapsulating IP traffic in GRE flows. In addition to a correct configuration of encapsulation and security policy, the connectivity has to be ensured between GRE endpoints for proper GRE functionality.

We consider a typical GRE configuration, implemented on router R_1:

```
interface Tunnel0
 ip address 10.12.1.1 255.255.255.0
 tunnel source Ethernet0/0
 tunnel destination 10.11.11.6
interface Ethernet0/0
 ip address 10.11.2.1 255.255.255.0
```

Router R_6 has the destination address of GRE tunnel configured on one of its interfaces. The GRE implementation is validated by showing that GRE flows originating at R_1 can reach R_6. It is done by computing relation $\hat{R}_{1,6}$ and testing if it contains GRE flow.

$$\hat{R}^{\cup}_{1,6}(\ in.src.ip = 10.11.2.1, in.dst.ip = 10.11.11.6,\\ in.pt = gre, out.src.ip = 10.11.2.1\\ out.dst.ip = 10.11.11.6, out.pt = gre)$$

This query tests if there is at least a single path between R_1 and R_6 that transmits GRE traffic. The corresponding Formula query is generated:

```
path('R1','R6',P), assume(F), assert(G), flow_reach(P,F,G),
flow_src_ip(F,10.11.2.1),flow_dst_ip(F,10.11.11.6),flow_pt(F,gre),
flow_src_ip(G,10.11.2.1),flow_dst_ip(G,10.11.11.6),flow_pt(G,gre).
```

6.3.2 Waypoints and Forbidden Paths

It is a straightforward to extended reachability relation $\hat{R}^{S}_{s,d}$ with a path descriptor attribute S to obtain only a set of paths satisfying some additional constraints. In the previous example, four paths are found to meet the query. These paths can be then checked on occurrences of waypoints and forbidden points.

The other way is to specify waypoints and forbidden points as the part of the query. This method is useful for analysis of failover configurations, which amounts to checking reachability considering link and device failures. The path object is represented as a list of nodes. The waypoints and forbidden point constraints can be expressed by testing the membership of these points in a path:

```
path('R1','R6',P), member('R2',P), no member('R3',P).
```

The previous constraint makes R_2 a waypoint and R_3 a forbidden point.

6.3.3 Rate Limitation

For networks implementing QoS, it is important to limit how much bandwidth each class may use to prevent network congestion and to provide enough network resources to high priority applications. Policy map configurations specify bandwidth limits for traffic classes. In principle, it is possible to extend reachability analysis with bandwidth parameters. Thus, it would be possible to verify that

all paths provide guaranteed minimum bandwidth for configured classes and compute network states that satisfy the requirements of high priority applications. Alternatively, it is possible to reveal network segments with insufficient rate limitations that potentially could harm QoS policy of the network.

6.4 Analysis of Routing

In this section, a definition of network model is extended to include routing information. A path-based model determines a set of paths that meets defined criteria for each destination network. These paths are found in the graph of a network topology that defines physical interconnections among network devices and connected networks. Each protocol maintains its knowledge in a structure called Routing Information Base (RIB). Routers exchanges information from individual RIBs to compute Forwarding Information Base (FIB), which controls the forwarding of packets.

This work is complementary to the work on packet filter analysis as carried out by, e.g., Guttman [25], Liu [51, 53, 54], Bera, Dasgupta and Ghosh [10, 12, 11]. The work presents an alternative to the approach introduced by Xie, Zhan, Maltz and Zhang in [90] and, in particular, by Maltz, Xie, Zhan and Greenberg [58]. Contrary to their work, current method computes the global view using standard graph algorithms without the need to simulate behaviors of routing protocols. On the other hand, it can become difficult to represent a route modification, e.g., tagging used to prevent routing loops. While the theoretical complexity of the method was determined, the future work is required to assess its practical contribution on real world examples. In the rest of this chapter, following terms are used:

- *Local RIB* is stored in routing process address space running on a router. Each process has its own RIB, e.g. RIP maintains RIP database. Similarly, *local FIB* is a single datatable, which is used by a router to decide where to forward incoming packets.
- *Network RIB/FIB* is a network wide view of routing information. This represents a shared routing knowledge of forwarding devices. Similarly, network FIB represents a global view on routing information that governs forwarding packets in the entire network.
- *Forwarding device* is a network device that actively decides where to forward packets based on its local routing information stored in FIB.
- *Redistribution* stands for copying route information to a target protocol instance, which is done in the scope of a single router.
- *Routing Protocol* defines rules of routing information exchange and routing information synthesis at local router. Commonly, Routing Information Protocol (RIP) , Open Shortest Path First (OSPF), Interior Gateway Routing Protocol (IGRP), and Enhanced IGRP (EIGRP) are employed in local networks.
- *Routing Instance* also called routing process is a process that runs the implementation of routing protocol within router's boundaries. It interacts with routing instances running at neighboring routers.

The model network topology for the purpose of routing information analysis is given as a hypergraph $G_{\mathsf{NET}} = \langle V_{\mathsf{NET}}, E_{\mathsf{NET}}, C \rangle$, where V_{NET} is a set of forwarding devices (routers) and $E_{\mathsf{NET}} \subseteq$

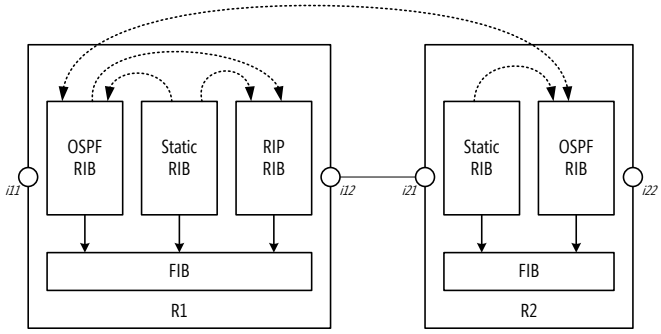


Fig. 6.3 The Model of Forwarding Device

$2^{V_{\mathsf{NET}}}$ is a set of physical links [1], and C is a set of configurations that govern behaviors of forwarding devices. For any $v \in V_{\mathsf{NET}}$ there is a configuration $C_v \in C$. This model stems from the model introduced in [59] and further developed in [81].

Similarly, a graph for each routing instance in a network can be defined. A model of forwarding devices explains an abstract internal structure of a router with respect to active routing instances.

6.4.1 The Model of Forwarding Device

A packet forwarding device (router) is modeled as a collection of routing instances each maintaining its routing information base (RIB). Router forwards packets using information from forwarding information base (FIB). The FIB is populated from local RIBs according to the specified procedure, which is usually proprietary to each device vendor. Cisco devices are taken as the reference platform. On this platform, each routing protocol is assigned by administrative distance that specifies a priority of the information stored in RIB with respect to router's FIB. Routing protocols with lower administrative distances are believed to maintain more accurate routing information and hence their information is equipped by a higher priority than information of routing protocols with higher administrative distances.

Figure 6.3 depicts the model of a router. This model is close to the model defined by Maltz et al. in [58]. The key features of this model include different graphs of routing information flows denoted as G_{RIB} . In Figure 6.3, there are three routing processes, denoted as Static [2], OSPF, and RIP. The arrows represent the following information flows:

- A flow from RIB to FIB describes the process of populating FIB with selected items from RIB according to the defined rules, e.g. based on administrative distance.

[1] Hypergraph is used because it can describe topologies that include n-to-n connections.

[2] The Static routing process maintains static information configured on a router. It consists of directly connected networks and static routes inserted by an administrator.

- A flow between RIBs represents a *redistribution* of information between different routing protocols or different instances of the same routing protocol running on the same router.
- A flow between RIBs on different devices represents information sharing (or exchanging) between routers that are using the same routing protocol.

Based on the previous information which all can be gathered from configuration files we can define a graph G_{RIB}. Routing information flows form a graph $G_{\mathsf{RIB}} = \langle V_{\mathsf{RIB}}, E_{\mathsf{RIB}}, \mathcal{P} \rangle$, where V_{RIB} is the set of RIBs in the network and E_{RIB} is the set of adjacencies between RIBs over which routing information can flow. Set $\mathcal{P}$, contains properties that can be assigned to edges E_{RIB}.

6.4.2 Representing Routing Information

A router's FIB stores routing information in the form of a record consisting of identification of a destination network, a next hop router, which is either determined by specifying an outgoing interface or by its IP address, and a cost. In the path-based model, a global view of a network is provided. Thus, the network FIB is represented by the FIB matrix that contains best paths to all destinations. Each cell of this matrix includes information in the form of $v_1 \rightarrow^{c_1} \rightarrow \ldots v_{n-1} \rightarrow^{c_{n-1}} v_n$, where $v_i \in V_{\mathsf{NET}}$. Notation, $\langle \pi \rangle^c$, where $\pi = v_1, \ldots, v_n$ represents a path and c is an aggregate cost is used. A cost of the path is always interpreted with respect to a routing protocol that advertises it, e.g. RIP uses hop count while OSPF uses values proportional to bandwidth along the path. Thus, meaning of every costs is provided, i.e., $c_i = \langle p, v \rangle$, where p denotes an interpretation for cost v.

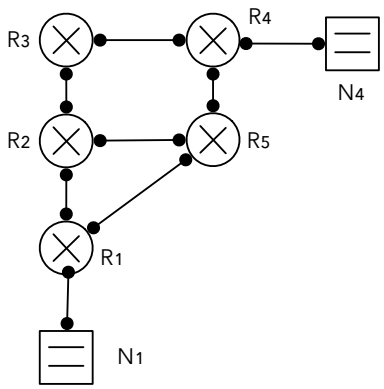
A *cost-path* is $\langle \pi \rangle^c$, where π denotes a path and c is an aggregate cost to the destination. A path may contain subpaths, for instance, $\langle \langle (v_1, v_2, v_3)^2 \rangle, v_6, v_5, v_8) \rangle^5$. This description allows to model a path that was observed by employing multiple protocols using redistribution. The aim is to express a global view on the network routing information; hence, instead of modeling local RIBs and FIBs, a network-wide RIBs, and the network-wide FIB are built. There exist numerous network RIBs, depending on the number of routing instances in the network. There is always at least one network RIB that represents the static routing. There is always a single network FIB, which contains a complete information on routing in the current network state. In the following sections, the computation of network RIBs and network FIB from static routing configurations is discussed.

6.4.3 Static RIBs

Static routing information base (RIB) for every router is computed from the network routing configuration. For instance, in the case of Cisco devices the static information occurring in the routing table consists of directly connected networks and static routes.

6.4.4 Directly connected networks

Directly connected networks are automatically placed in local RIBs. There are two methods to define directly connected route:



(a) Network topology

```
R2# ip route N1 R1
R3# ip route N1 R2
R4# ip route N1 R3
R5# ip route N1 R1

R1# ip route N4 R5
R2# ip route N4 R3
R3# ip route N4 R4
R5# ip route N4 R2
```

(b) Routing Configuration

N_1	R_1	R_2	R_3	R_4	R_5
R_1	0	×	×	×	×
R_2	1	×	×	×	×
R_3	×	1	×	×	×
R_4	×	×	1	×	×
R_5	1	×	×	×	×

N_1	R_1	R_2	R_3	R_4	R_5
R_1	×	×	×	×	×
R_2	$(2,1)^1$	×	×	×	×
R_3	$(3,2,1)^2$	$(3,2)^1$	×	×	×
R_4	$(4,3,2,1)^3$	$(4,3,2)^2$	$(4,3)^1$	×	×
R_5	$(5,1)^1$	×	×	×	×

(c) Adjacency table and path matrix for destination N_1

N_4	R_1	R_2	R_3	R_4	R_5
R_1	×	×	×	×	1
R_2	×	×	1	×	×
R_3	×	1	×	1	×
R_4	×	×	1	0	×
R_5	×	1	×	×	×

N_4	R_1	R_2	R_3	R_4	R_5
R_1	×	$(1,5,2)^2$	$(1,2,3,4)^3$	$(1,5,2,3,4)^4$	$(1,5)^1$
R_2	×	×	$(2,3)^1$	$(2,3,4)^2$	×
R_3	×	×	×	$(3,4)^1$	×
R_4	×	×	×	×	×
R_5	×	$(5,2)^1$	$(5,2,3)^2$	$(5,2,3,4)^3$	×

(d) Adjacency table and path matrix for destination N_4

Fig. 6.4 An example of static network configuration and network RIBs.
This figure presents adjacency matrices and path matrices for destinations N_1 and N_4. Path information in a path matrix has form of $(path)^{cost}$. The resulting routing information for network N_1 and N_4, respectively, is obtained by taking column R_1 and R_4.

- An interface is configured with a valid IP address and mask. Such configuration is implicitly considered as a directly connected route that will be installed in the routing table.
- A static route is configured without defining a next hop Ip address. It means only an outgoing interface is defined.

6.4.5 Static routes

Static routes govern packet switching on a local router. Implicitly, they have assigned a small administrative distance, which stands for their high priority in the forwarding process. It means that static routes will replace dynamic routes in the router's forwarding information base (FIB). For instance, using RIP dynamic routing protocol, router `R1` knows about the destination network `10.151.14.0/8` via interface `s0/1` with cost 8. However, the static route for this destination suggests using interface `s0/2`. The static route has priority over the dynamic route and, thus, the router will send packets for `10.151.14.0/8` out the interface `s0/2`.

6.4.6 Representing Static Network RIB

Figure 6.4 depicts an example of static routing configuration and the resulting RIBs for networks N_1 and N_4. The computation starts with adjacency matrices that capture an effect of `ip route` configuration commands. Then, using a standard graph algorithm, e.g., Floyd-Warshall, the path matrices are computed. Because network N_1 is directly connected to router R_1 the column R_1 of the path matrix for N_1 defines all paths from any network destination to network N_1. From the example, it can be concluded that

- it is possible to use standard graph approach to determine paths taken by packets routed under the static routing configuration, and
- for each network, it is necessary to perform independent computation unless two or more networks share the same adjacency matrix.

Computing all pair shortest path using Floyd-Warshall algorithm has time complexity of $|V|^3$. As stated above we need to perform up to $|N|$ executions of this algorithm, which is the number of destinations to be analyzed, to determine the reachability of the network. Hence, the overall complexity is $|N| \cdot |V|^3$.

6.4.7 Dynamic Routing

Routing processes running on network devices execute a distributed algorithm to collect all relevant routing data. These data are stored in a local database of routing protocol, which is private to the routing process. Every instance of routing protocol has its individual database that maintains this kind of information. The following is a content of a RIP database captured on a Cisco router:

```
Router#show ip rip database
11.0.0.0/8  directly connected, FastEthernet0/0
12.0.0.0/8
    [1] via 192.168.1.2, 00:00:23, Serial2/0
13.0.0.0/8
    [2] via 192.168.1.2, 00:00:23, Serial2/0
192.168.1.0/24    directly connected, Serial2/0
192.168.2.0/24
```

```
[1] via 192.168.1.2, 00:00:23, Serial2/0
```

Using defined criteria, the router selects from local RIBs of all running routing protocols the information about available routes and installs them in the routing table. The content of a routing table determines paths, which the traffic takes in the network.

Maltz et al. [58] developed a model for understanding routing contribution to a network dynamics. In this section, their routing process graph is utilized to define an abstraction for network wide-routing information dissemination. The goal is to compute an approximation of routing base information for the given system state without simulating (distributed) routing protocol algorithms.

For elementary cases, it is easy to determine routing information. Based on the routing graph one knows the flow restrictions of routing information and, by application of a standard graph traversal algorithm, it is possible to find the best paths with respect to the cost models. However, it is possible that routing information be modified as it is disseminated among routers. In the next section, routing configuration includes access control lists to control dissemination of routing updates.

6.4.8 Filtering Routing Updates

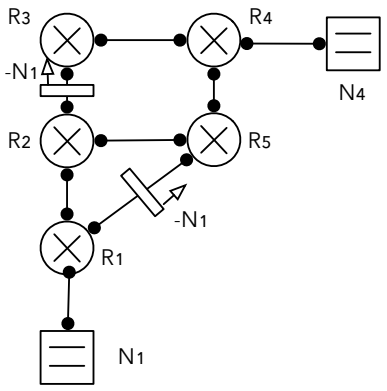
Route filtering is provided by regulating the route advertisements sent to neighboring routers and by filtering routes advertised by other routers before they are added to or updated in the local routing protocol database. Route filters have only an effect on distance vector protocols, e.g. Routing Information Protocol (RIP), Interior Gateway Routing Protocol (IGRP), and Enhanced IGRP (EIGRP).

It is possible to block routing updates sent through the interface or to control the processing and advertising of routes in routing updates. The first option stands for completely denying updates usually sent by the router to its neighbors through the connecting interface. The second option represents applying filters that delete some routes from the routing update sent to the neighbor routers.

Depending on the device vendor, some form of access control lists is used to decide, which routes will be filtered. It is possible to process i) incoming routing updates to control, which routes are added to a local database, or ii) outgoing routing updates to control, which routes are sent to neighbor routers. Bellow is an example of two routing filters:

```
access-list 1 permit 1.0.0.0 0.255.255.255
access-list 2 permit 1.2.3.0 0.0.0.255
router rip
  distribute-list 2 in ehternet 0
  distrubute-list 1 out
```

The routing filter implemented by access control list 1 affects all outgoing routing updates and allows to send only information about destination `1.0.0.0/8`. The routing filter implemented by access control list 2 accepts from all updates received on Ethernet interface only information about destination `1.2.3.0/24`.



(a) Network topology

```
R1# router rip
R1# network N1
R2# router rip
R3# router rip
R4# router rip
R4# network N4
R5# router rip
```

(b) Routing Configuration

N_1	R_1	R_2	R_3	R_4	R_5
R_1	0	1	×	×	1
R_2	1	×	1	×	1
R_3	×	1	×	1	×
R_4	×	×	1	×	1
R_5	×	1	×	1	×

N_1	R_1	R_2	R_3	R_4	R_5
R_1	×	$(1,2)^1$	$(1,2,3)^2$	$(1,2,4)^2$	$(1,5)^1$
R_2	$(2,1)^1$	×	$(2,3)^1$	$(2,3,4)^2$	$(2,5)^1$
R_3	$(3,4,5,2,1)^4$	$(3,4,5,2)^3$	×	$(3,4)^1$	$(3,4,5)^2$
R_4	$(4,5,2,1)^3$	$(4,5,2)^2$	$(4,3)^1$	×	$(4,5)^1$
R_5	$(5,2,1)^2$	$(5,2)^1$	$(5,2,3)^2$	$(5,4)^1$	×

(c) Routing adjacency table and path matrix for destination N_1

N_4	R_1	R_2	R_3	R_4	R_5
R_1	×	1	×	×	1
R_2	1	×	1	×	1
R_3	×	1	×	1	×
R_4	×	×	1	×	1
R_5	1	1	×	1	×

N_4	R_1	R_2	R_3	R_4	R_5
R_1	×	$(1,2)^1$	$(1,2,3)^2$	$(1,5,4)^2$	$(1,5)^1$
R_2	$(2,1)^1$	×	$(2,3)^1$	$(2,3,4)^2$	$(2,5)^1$
R_3	$(3,2,1)^2$	$(3,2)^1$	×	$(3,4)^1$	$(3,2,5)^2$
R_4	$(4,5,1)^2$	$(4,3,2)^2$	$(4,3)^1$	×	$(4,5)^1$
R_5	$(5,1)^1$	$(5,2)^1$	$(5,2,3)^2$	$(5,4)^1$	×

(d) Routing adjacency table and path matrix for destination N_4

Fig. 6.5 An example of route filtering and computation of network RIBs:
For simplicity there are only two route filters applied on links $\langle R_1, R_5\rangle$ and $\langle R_2, R_3\rangle$, respectively. These filters deny to send information on network N_1 to routers R_5 and R_3, which is captured in adjacency table for network N_1 by deleting adjacencies at $Adj_{N_1}[R_5, R_1]$ and $Adj_{N_1}[R_3, R_2]$. For comparison, Adj_{N_4} enjoys full adjacency as no filters for N_4 are configured in the network.

6.4.9 Computing the Effects of Filtering Routing Updates

First, the case without route filters is considered. It is possible to compute a network RIB for the routing protocol instance by defining an adjacency table that exactly follows the neighborship relations in a network among the same routing protocol instances. Then, again by adopting the standard graph algorithms, the path matrix is computed. It would be possible to calculate a single RIB that defines reachability and path information for all networks.

Then, the case, in which route filters could remove some destinations from routing updates is presented. In this case, it is necessary to calculate the path matrices for individual networks. If there is route filtering for network N on a link between router R_s and R_t the adjacency between these two nodes from the adjacency table must be removed. It involves computing a path matrix for each network, which leads to $|V|^3 \cdot |N|$ time complexity, be the same as in the case of computing static network RIB. However, in real-world scenarios, not all networks are filtered, or a single filter affects multiple networks. A term *slice* denotes a collection of the equally treated networks.

A *slice* describes a collection of destinations that are refined by the same filters. A collection of slices can be obtained by analyzing configuration of routing update filters. A routing update filter is a set of networks removed from routing updates. For instance, filter $f = \{n1, n2, n5, n7\}$ that deletes information about networks n_1, n_2, n_5 and n_7 from an update. As described in the previous subsection, there are various kinds of filters applicable in various way. However, all these filters can be appropriately represented in a graph model by associating the filtering sets to edges.

Definition 21 (Slicing). Given F to be a set of filtering, we define the slicing S to be a set partitioning such that:

- $\forall f \in F, s \in S : s \cap f \neq \emptyset \implies s \subseteq f$,
- $\bigcup_{s \in S} s = N$
- $\forall s_1, s_2 \in S : s_1 \cap s_2 = \emptyset$.

The perfect slicing stands for such slicing that provides minimal $|S|$ with respect to F among a set of possible slicing sets. The problem is solved in the following two steps:

- estimate a set of filters for each network, and
- collect networks according to their sets of filters; networks that have the same filters belong to the same slice.

Figure 6.5 depicts a demonstration of route filtering for two destination networks. For simplicity, we do not consider the slice-based approach. The algorithm starts by initiating cost matrices using information from configurations. Cost adjacency matrices for networks N_1 and N_4 describe the effect of routing update filters on a RIP databases. The basic idea is to classify edges between RIP vertices into two categories. If an edge has no associated filter that prevents an information about a network to be sent from a source vertex to a destination vertex, then the edge has associated its cost; hence, this costs is stored in the adjacency matrix. On the other hand, the cost denoted as `x` expresses that information is filtered on that edge. The path matrix is computed using the standard graph algorithm to obtain N_1-RIB and N_4-RIB associated with RIP instance.

6.5 Redistribution

The situation when routing protocols advertize routes learned by some other means is called redistribution. While the most evident and desirable use of Interior Gateway Protocols (IGP) is to employ a single routing protocol for the entire domain, there are situations, which are quite common in practice: multiple IGP in a single area. In these situations some form of redistribution is necessary.

```
1 for r ∈ V do
2 |   if RIB_s[v, r] ∈ FIB then
3 |   |   RIB_t[v, r] ← c_min(m_S(RIB_s[v, r]), RIB_t[v, r])
4 |   end
5 end
```

Algorithm 4: Redistribution algorithm

When redistributing, it is important to define a correct initial (seed) metric for each redistributing route as each protocol uses a different cost scheme. A configuration snippet bellow shows redistributing static routes and OSPF routes to RIP. A value after `metric` keyword specifies the seed metric used for redistributed routes in RIP.

```
router rip
redistribute static metric 1
redistribute ospf metric 1
```

A router uses an administrative distance to select the best route for each destination. This route is installed in the router's forwarding information base (routing table). Using redistribution in a wrong way can lead to problems in forming routing loops, convergence problems or inefficient routing [48].

The redistribution mechanism is vendor dependent, but most platforms obey two additional rules when doing redistribution:

RR1: The route can only be redistributed if it is installed in router's FIB.

RR2: Even if a route is redistributed in the routing process with lower AD, this new route is not installed into router's FIB.

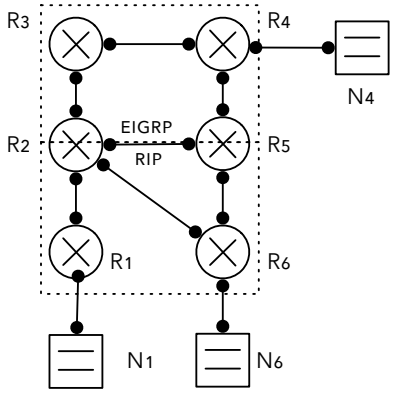
These two rules cause that redistribution is not transitive as pointed out by Le, Xie and Zhang in [48]. This observation makes our computation more complicated.

6.5.1 Computing Redistribution

To demonstrate the approach to redistribution the example shown in Figure 6.7. Redistribution is done within router's boundary. Redistribution matrix is similar to a matrix that denotes adjacency for distance vector routing protocols. This matrix expresses how the route redistribution is configured on a router.

Redistribution configured on router R_v, which redistributes routing information from RIB_s to RIB_t requires to insert paths at row v of matrix RIB_s to corresponding items in the matrix RIB_t if these paths have lower costs and can be found in FIB. Redistribution process is defined in Algorithm 4. In the algorithm, operator c_{min} selects a path with lower cost and function m_S assigns a metric seed to redistributed information.

The impact of redistribution rules RR1 and RR2 is best observable in the process of route selection. The route selection finds the most appropriate information from local RIBs and puts it in the router's FIB. Redistribution then must check whether the information that is redistributed was correctly selected, i.e., it conforms to Rule RR1. Also, the route selection must not choose redistributed



(a) Network topology

R_2	STATIC	RIP	EIGRP
STATIC	×	×	×
RIP	×	×	1
EIGRP	×	×	×

R_5	STATIC	RIP	EIGRP
STATIC	×	×	×
RIP	×	×	×
EIGRP	×	1	×

(b) Redistribution Tables

EIGRP	R_1	R_2	R_3	R_4	R_5	R_6
R_1	×	×	×	×	×	×
R_2	×	×	1	×	×	×
R_3	×	1	×	1	×	×
R_4	×	×	1	×	1	×
R_5	×	×	×	1	×	×
R_6	×	×	×	×	×	×

(c) Routing Adjacency for EIGRP

RIP	R_1	R_2	R_3	R_4	R_5	R_6
R_1	×	1	×	×	×	×
R_2	1	×	×	×	×	1
R_3	×	×	×	×	×	×
R_4	×	×	×	×	×	×
R_5	×	×	×	×	×	1
R_6	×	1	×	×	1	×

(d) Routing Adjacency for RIP

Fig. 6.6 An example of route redistribution configuration

Network topology presented in (a) consists of two routing domains. Routers R_1 and R_6 are only in RIP routing domain and routers R_3 and R_4 are only in EIGRP routing domain. Router R_2 is in both routing domain and performs RIP to EIGRP redistribution. Router R_5 is also in both domains and makes redistribution from EIGRP domain to RIP domain. Redistribution is expressed in redistribution tables in (b). Each routing domain has its private routing adjacency matrix shown in (c) and (d). Using these matrices, separate RIBs for both routing domains are computed.

network into the FIB to satisfy Rule RR2. The next section describes the route selection process in detail.

6.5.2 Route Selection

The purpose of the routing processes running on each router is to maintain routing protocol specific information. Every process manages its (topological) database that allows to determine the best path to the destination as viewed by the routing protocol. The router needs to select a single (or a collection of alternate paths for load balancing) route to its FIB. This process is vendor dependent, but most often the routing information is prioritized by using administrative distance measure. This section provides a direct algorithm that computes a content of the network FIB from a collection of network RIBs. For simplicity, all routers follow the same route selection rules and no router has modified default administrative distance for any routing protocol nor any single route. This situation corresponds to the vast majority of configurations used in enterprise networks. The case

N^*	R_1	R_2	R_3	R_4	R_5	R_6
R_1	×	×	×	×	×	×
R_2	×	×	$(2,3)^1$	$(2,3,4)^2$	$(2,3,4,5)^3$	×
R_3	×	$(3,2)^1$	×	$(3,4)^1$	$(3,4,5)^2$	×
R_4	×	$(4,3,2)^2$	$(4,3)^1$	×	$(4,5)^1$	×
R_5	×	$(5,4,3,2)^3$	$(5,4,3)^2$	$(5,4)^1$	×	×
R_6	×	×	×	×	×	×

(a) EIGRP RIB

N^*	R_1	R_2	R_3	R_4	R_5	R_6
R_1	×	$(1,2)^1$	×	×	$(1,2,6,5)^3$	$(1,2,6)^2$
R_2	$(2,1)^1$	×	×	×	$(2,6,5)^2$	$(2,6)^1$
R_3	×	×	×	×	×	×
R_4	×	×	×	×	×	×
R_5	$(5,6,2,1)^3$	$(5,6,2)^2$	×	×	×	$(5,6)^1$
R_6	$(6,2,1)^2$	$(6,2)^1$	×	×	$(6,5)^1$	×

(b) RIP RIB

N^*	R_1	R_2	R_3	R_4	R_5	R_6
R_1	×	$(1,2)^1$	×	×	$(1,2,6,5)^3$	$(1,2,6)^2$
R_2	$(2,1)^1$	×	$(2,3)^1$	$(2,3,4)^2$	$(2,3,4,5)^3$	$(2,6)^1$
R_3	×	$(3,2)^1$	×	$(3,4)^1$	$(3,4,5)^2$	×
R_4	×	$(4,3,4)^2$	$(4,3)^1$	×	$(4,5)^1$	×
R_5	$(5,6,2,1)^3$	$(5,4,3,2)^3$	$(5,4,3)^2$	$(5,4)^1$	×	$(5,6)^1$
R_6	$(6,2,1)^2$	$(6,2)^1$	×	×	$(6,5)^1$	×

(c) Network FIB

Fig. 6.7 Initial content of RIBs and FIB for example from Fig.6.6

Initially, network RIBs for two routing instances are computed yielding to EIGRP RIB and RIP RIB tables as shown in (a) and (b). Network FIB is obtained by application of Algorithm 5 taking EIGRP RIB and RIP RIB as inputs. At this stage, Network FIB represents a network routing state before redistribution is applied. This means that R_4 and R_3 are not accessible from R_1, yet. The result of applying redistribution is shown in Fig.6.8.

when administrative distances are redefined is left for future work. The route selection method is defined in Algorithm 5.

The algorithm has time complexity $|N| \cdot |R| \cdot |V^2|$, where N is a number of destination networks, R is a number of RIBs and V is a number of routers. Informally, the computation proceeds as follows:

1. Take the lowest priority network RIB and copy all information to a network FIB. This step will initialize the network FIB.
2. Take the RIB with immediately higher priority and replace paths in FIB with existing paths in this RIB. This step corresponds to the selection of route information with less administrative distance. If there is not any path in this RIB then the path from a lower priority RIB remains in the FIB.
3. For each path in the RIB, check if this path can replace a suffix in an existing path in the FIB. This means, that if the FIB contains a path $\langle r_1, r_4, r_2, r_5, r_3 \rangle$ and the RIB contains a path $\langle r_4, r_7, r_8, r_9, r_3 \rangle$,

N^*	R_1	R_2	R_3	R_4	R_5	R_6
R_1	×	×	×	×	×	×
R_2	$(2,1)^1$	×	$(2,3)^1$	$(2,3,4)^2$	$(2,3,4,5)^3$	$(2,6)^1$
R_3	×	$(3,2)^1$	×	$(3,4)^1$	$(3,4,5)^2$	×
R_4	×	$(4,3,2)^2$	$(4,3)^1$	×	$(4,5)^1$	×
R_5	×	$(5,4,3,2)^3$	$(5,4,3)^2$	$(5,4)^1$	×	×
R_6	×	×	×	×	×	×

(a) EIGRP RIB

N^*	R_1	R_2	R_3	R_4	R_5	R_6
R_1	×	$(1,2)^1$	×	×	$(1,2,6,5)^3$	$(1,2,6)^2$
R_2	$(2,1)^1$	×	×	×	$(2,6,5)^2$	$(2,6)^1$
R_3	×	×	×	×	×	×
R_4	×	×	×	×	×	×
R_5	$(5,6,2,1)^3$	$(5,4,3,2)^1$	$(5,4,3)^1$	$(5,4)^1$	×	$(5,6)^1$
R_6	$(6,2,1)^2$	$(6,2)^1$	×	×	$(6,5)^1$	×

(b) RIP RIB

Fig. 6.8 Network RIBs after redistribution was applied

Tables (a) and (b) represent EIGRP RIB and RIP RIB after redistribution was applied by executing Algorithm 4. Redistribution propagates new information into RIBs. The RIBs are updated by disseminating routes within routing domain (see Fig.6.9).

```
1  for n ∈ N do
2      for R ∈ RIB do
3          for r ∈ V do
4              if ∃p ∈ R[r, n] then
5                  FIB[r, n] ← p for s ∈ V : ⟨q_0, ..., r, ..., q_n⟩ = FIB[s, n] do
6                      FIB[s, n] = ⟨q_0, ..., r⟩ + p
7                  end
8              end
9          end
10     end
11 end
```

Algorithm 5: Route selection algorithm

the FIB's path should be replaced with $\langle r_1, r_2, r_4, r_7, r_8, r_9, r_3 \rangle$. This replacement corresponds to installation of a route with lower AD.

4. Repeat from step 2 until all RIBs are processed.

Information stored in the network FIB can be used to determine paths to all destinations. In the example presented in Figure 6.5, the table $N^\star$-FIB for all networks are computed as static routing nor route filtering are configured. If network configuration combines static routing, dynamic routing with route filtering, and redistribution, one will need to compute more network FIBs depending on the number of networks.

N^*	R_1	R_2	R_3	R_4	R_5	R_6
R_1	×	×	×	×	×	×
R_2	$(2,1)^1$	×	$(2,3)^1$	$(2,3,4)^2$	$(2,3,4,5)^3$	$(2,6)^1$
R_3	$(3,2,1)^2$	$(3,2)^1$	×	$(3,4)^1$	$(3,4,5)^2$	$(3,2,6)^2$
R_4	$(4,3,2,1)^3$	$(4,3,2)^2$	$(4,3)^1$	×	$(4,5)^1$	$(4,3,2,6)^3$
R_5	$(5,4,3,2,1)^4$	$(5,4,3,2)^3$	$(5,4,3)^2$	$(5,4)^1$	×	$(5,4,3,2,6)^4$
R_6	×	×	×	×	×	×

(a) EIGRP RIB

N^*	R_1	R_2	R_3	R_4	R_5	R_6
R_1	×	$(1,2)^1$	$(1,2,6,5,4,3)^4$	$(1,2,6,5,4)^4$	$(1,2,6,5)^3$	$(1,2,6)^2$
R_2	$(2,1)^1$	×	$(2,6,5,4,3)^3$	$(2,6,5,4)^3$	$(2,6,5)^2$	$(2,6)^1$
R_3	×	×	×	×	×	×
R_4	×	×	×	×	×	×
R_5	$(5,4,3,2,1)^2$	$(5,4,3,2)^1$	$(5,4,3)^1$	$(5,4)^1$	×	$(5,6)^1$
R_6	$(6,2,1)^2$	$(6,2)^1$	$(6,5,4,3)^2$	$(6,5,4)^2$	$(6,5)^1$	×

(b) RIP RIB

N^*	R_1	R_2	R_3	R_4	R_5	R_6
R_1	×	$(1,2)^1$	$(1,2,6,5,4,3)^4$	$(1,2,6,5,4)^4$	$(1,2,6,5)^3$	$(1,2,6)^2$
R_2	$(2,1)^1$	×	$(2,3)^1$	$(2,3,4)^2$	$(2,3,4,5)^3$	$(2,6)^1$
R_3	$(3,2,1)^2$	$(3,2)^1$	×	$(3,4)^1$	$(3,4,5)^2$	$(3,2,6)^2$
R_4	$(4,3,2,1)^3$	$(4,3,4)^2$	$(4,3)^1$	×	$(4,5)^1$	$(4,3,2,6)^3$
R_5	$(5,4,3,2,1)^4$	$(5,4,3,2)^3$	$(5,4,3)^2$	$(5,4)^1$	×	$(5,4,3,2,6)^4$
R_6	$(6,2,1)^2$	$(6,2)^1$	$(6,5,4,3)^2$	$(6,5,4)^2$	$(6,5)^1$	×

(c) Network FIB

Fig. 6.9 Final RIBs and network FIB tables

Tables in (a), (b) represents the final content of routing information bases for both routing domains. Table (c) shows the converged Network FIB. All redistributed routes were propagated. Note that route $R_1 \mapsto R_5$ goes through EIGRP routing domain because of better cost associated.

6.6 Chapter Summary

The method presented in this chapter aims at validating network configuration against the absence of errors and security flaws. The network configuration model allows describing effects of static and dynamic routing, access control lists, network address translation and other related features of network devices. The verification technique is based on representing configurations as domain models and applying automatized proving techniques to validate the configuration and query the reachability model. Discussed procedure can be also viewed as a refinement of models used by Narain [62, 65] to include routing effects, which would allow network designers to get an insight on the issue of interweaving routing and filtering, and their impact on network security properties. Further work is aimed at refining the method to experimental implementation and performing experiments to evaluate its performance and scalability.

The routing domain can be used to compute network paths for network targets. Results of routing model can be incorporated in network reachability model to further constraint paths.

Routing model information can be turn to filters associated with network locations in a reachability model. Alternatively, routing model can be precomputed, and reachability model can be inspected only for best paths.

A presented method is suitable for computing the network-wide view of the forward information base (FIB), which allows ones to predict paths for traffic in the analyzed network. This work is complementary to the work on packet filter analysis as carried out by, e.g., Guttman [25], Liu [87], Bera, Dasgupta and Ghosh [12]. The work presents an alternative method to the approach introduced by Xie, Zhan, Maltz and Zhang in [58] and, in particular, by Maltz, Xie, Zhan and Greenberg [90]. Contrary to their work, the global view using standard graph algorithms is computed without simulating routing protocols. On the other hand, it can be difficult to represent various modifications in routing algorithms, e.g., using tagging to prevent routing loops.

Chapter 7
Conclusions

Current network configurations are involved because they have to satisfy many different requirements. Network devices are configured using high-level declarative languages that control devices behavior forming the overall system functionality. Because network parameters depend on the composition of these individual configurations, they should be consistent and meet expected properties. From this perspective, providing correct device configurations for enterprise network is a difficult task requiring advanced knowledge of various technologies comprising routing, security, access control, high-availability, quality of service and monitoring. A method aiding to deliver correct and network-wide consistent configurations would significantly improve the current situation. The research lead to design of methods for selected problems, e.g. firewall configuration, service security control, policy languages as well as to development of systems for complex configuration analysis and synthesis, e.g., Config Assure, MulVal, FAME, FireCrocodile or Fireman among others.

In this thesis, a logic-based framework supporting formalization and analysis of network policy, firewalls and network configurations is considered. The framework enables revealing misconfigurations or security issues at design-time by analysis of configuration files. The novel results comprise of:

- Definition of a simple domain for network reachability analysis. This domain is capable of interpreting meaning of many network mechanisms, e.g., routing, forwarding, access control, network address translation, quality of service, and traffic tunneling. The simplicity of the models expressed by constraints in *Reachability* domain simplifies reasoning about reachability properties. The reasoning can be performed directly in Formula environment. *The primary contribution of this part is in providing simple but expressive model for interpreting network policy rules and network configuration semantic in terms of reachability calculation.*
- Definition of formal language for capturing and analysis of network policy. A network policy specification is an aggregation of policy rules, each of which consists of a condition and action. The purpose of a security policy is to control network communication to meet security requirements. Network policy thus governs availability and states conditions of the use of services provided by servers to clients. Security policies defined at the service level classify the flows into different service groups. The proposed policy language and its formalization allow to express many network policies and provide automated analysis of their properties. In addition, service-based part of the new policy language also allow to specify and reason about accessibility of network services and related security features. *The primary contribution of this part consists of the network*

policy domain definition that can be further developed and refined. Possible extensions depend on intended application areas, e.g., policy-based network design, policy-based network control or network configuration verification.

- Definition of a formal framework for specification of firewall rules and analysis of firewall consistency. In addition to formalization of firewall domain, a novel algorithm for efficient checking of firewall consistency was developed. Conflict checking methods are not only useful for detecting errors in firewall configurations but also for improving the firewall efficiency by eliminating redundant rules. A conflict free firewall rule base for an arbitrary firewall also serves as a more suitable input for configuration analysis methods based on constraint solving techniques. *Contribution of this part is in formalization of firewall domain and identifying methods that can be used for firewall manipulation to obtain the conflict-free representation. In addition, a novel algorithm for conflict detection was presented, and its efficiency was evaluated.*
- Development of a domain for capturing the semantics of network configuration and providing a method for design-time analysis of network configuration by interpreting configurations in the reachability domain. This result stands for the primary achievement of the presented work. The presented work stems from the contemporary research and in many aspects follows or alternates existing research directions. For instance, Narain [63] introduced ConfigAssure environment for configuration verification and synthesis. ConfigAssure represents a tool that can verify existing network configuration and in some cases also perform configuration synthesis. ConfigAssure employs requirement solver for finding values of configuration variables that would satisfy system requirements. The requirement solver is implemented using a combination of Kodkod model finder and Prolog. The approach presented in this thesis is in comparison to ConfigAssure more focused on formalization of information from network configuration files and interpreting it with respect to network functionality. The idea is to provide a system configuration context in a form of hierarchy of Formula domains to provide the foundation for implementing practical tools for network configuration management. *Contribution to the area of network configuration analysis consists of proposing a formal framework that unifies network specification languages. The formal framework can express network requirements, firewalls, and configurations. It also provides techniques for finding semantic errors in configurations of various kinds, e.g., policy, firewalls, devices.*

Presented framework is limited to features that can be expressed in terms of constraints on finite domains. It seems that many practical requirements can be expressed in this way. The proposed method can lead to implementation of practical tools assisting in network configuration management [1]. Promoting the presented framework into a full-fledged tool requires to extend it with other configuration features found in enterprise networks. Currently, covered features consist of routing, access control, quality of service, network address translation and tunneling. Features such as high-availability, policy-based routing, virtual private networks and MPLS were not discussed in the presented work. Experts in networking may not have skills to use a logical system for formalization of domain assumptions. To overcome this problem, the framework presented in this work is designed as a collection of reusable domains that can be assembled or altered in new development.

Future work is oriented towards extending proposed context with other domains capturing more network configuration features. The selection of new features is based on configurations of

[1] Currently, an experimental version with limited functionality is implemented and can be found at `https://github.com/rysavy-ondrej/Netcow`

typical real enterprise networks. Practical considerations drive the other direction of future work. To provide a tool that can assist in network configuration management, one needs to implement functions that allow for the automatic configuration acquisition and provide a rich reporting system offering human readable information explaining verification results. The tool in its current state provides only limited capabilities with this respect.

References

1. AbuHmed, T., Mohaisen, A., Nyang, D.: A Survey on Deep Packet Inspection for Intrusion Detection Systems. Information Security **24**, 10 (2008). URL http://arxiv.org/abs/0803.0037
2. Al-Shaer, E., Hamed, H.: Discovery of policy anomalies in distributed firewalls. In: Ieee Infocom 2004, pp. 2605–2616. Ieee (2004). DOI 10.1109/INFCOM.2004.1354680. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1354680
3. Al-Shaer, E., Hamed, H.: Modeling and Management of Firewall Policies. IEEE Transactions on Network and Service Management **1**(1), 2–10 (2004). DOI 10.1109/TNSM.2004.4623689. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4623689
4. Al-Shaer, E., Hamed, H., Boutaba, R.: Conflict classification and analysis of distributed firewall policies. IEEE Journal on Selected Areas in Communications **23**(10) (2005). URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1514536
5. Al-Shaer, E., Marrero, W., El-Atawy, A., ElBadawi, K.: Towards global verification and analysis of network access control configuration. DePaul University, Chicago, IL, USA, Tech. Rep (2008). URL http://via.library.depaul.edu/cgi/viewcontent.cgi?article=1007&context=tr
6. Al-Shaer, E., Marrero, W., El-Atawy, A., ElBadawi, K.: Network configuration in a box: towards end-to-end verification of network reachability and security. 2009 17th IEEE International Conference on Network Protocols pp. 123–132 (2009). DOI 10.1109/ICNP.2009.5339690. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5339690
7. Baboescu, F., Varghese, G.: Fast and scalable conflict detection for packet classifiers. Computer Networks **42**(6), 717–735 (2003). DOI 10.1016/S1389-1286(03)00213-5. URL http://linkinghub.elsevier.com/retrieve/pii/S1389128603002135
8. Bandhakavi, S., Bhatt, S., Okita, C., Rao, P.: Analyzing end-to-end network reachability. 2009 IFIP/IEEE International Symposium on Integrated Network Management pp. 585–590 (2009). DOI 10.1109/INM.2009.5188865. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5188865
9. Bartal, Y., Mayer, A., Nissim, K., Wool, A.: Firmato: a novel firewall management toolkit. In: Proceedings of the 1999 IEEE Symposium on Security and Privacy, pp. 17–31. IEEE Comput. Soc (1999). DOI 10.1109/SECPRI.1999.766714. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=766714
10. Bera, P., Ghosh, S., Dasgupta, P.: Formal Verification of Security Policy Implementations in Enterprise Networks. Information Systems Security pp. 117–131 (2009). URL http://www.springerlink.com/index/2528M87085N32371.pdf
11. Bera, P., Maity, S., Ghosh, S.: Generating policy based security implementation in enterprise network: a formal framework. In: Proceedings of the 3rd ACM workshop on Assurable and usable security configuration, pp. 1–8. ACM (2010). URL http://dl.acm.org/citation.cfm?id=1866900
12. Bera, P., Maity, S., Ghosh, S., Dasgupta, P.: A Query based Formal Security Analysis Framework for Enterprise LAN. In: 2010 10th IEEE International Conference on Computer and Information Technology,

Cit, pp. 407–414. Ieee (2010). DOI 10.1109/CIT.2010.96. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5578175

13. Buchmann, D.: Verified Network Configuration: Improving Network Reliability. Ph.D. thesis (2008). URL http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Verified+Network+Configuration:+Improving+Network+Reliability#5
14. Cheswick, W.R., Bellovin, S.M., Rubin, A.D.: Firewalls and Internet Security; Repelling the Wily Hacker, second edn. Addison-Wesley, Reading, MA (2003). URL http://www.wilyhacker.com/
15. Cisco: IP Addressing Services - NAT Order of Operation (2015). URL http://www.cisco.com/c/en/us/support/docs/ip/network-address-translation-nat/6209-5.html
16. Clark, D.: Policy Routing in Internet Protocols. Tech. rep., IETF Network Working Group (1989)
17. Cuppens, F., Saurel, C.: Specifying a security policy: a case study. In: Proceedings 9th IEEE Computer Security Foundations Workshop, pp. 123–134. IEEE Comput. Soc. Press (1996). DOI 10.1109/CSFW.1996.503697. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=503697
18. Damianou, N., Dulay, N., Lupu, E., Sloman, M.: The Ponder Policy Specification Language. Policy **1995**, 18–38 (2001). DOI 10.1007/3-540-44569-2_2. URL http://www.springerlink.com/index/1R0VN5HFXK6DXEBB.pdf
19. Enck, W., Moyer, T., McDaniel, P., Sen, S.: Configuration management at massive scale: system design and experience. Selected Areas in **27**(3), 323–335 (2009). DOI 10.1109/JSAC.2009.090408. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4808476http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4808476
20. Eppstein, D., Muthukrishnan, S.: Internet packet filter management and rectangle geometry. In: Proceeding SODA '01 Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms, pp. 1–9 (2001). URL http://portal.acm.org/citation.cfm?id=365791
21. Frühwirth, T.: Theory and practice of constraint handling rules (1998). DOI 10.1016/S0743-1066(98)10005-5
22. Gan, Q., Helvik, B.: Dependability modelling and analysis of networks as taking routing and traffic into account. In: 2nd Conference on Next Generation Internet Design and Engineering, 2006. NGI '06. 2006, pp. 1–8 (2006). URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1678219
23. Gomes, C., Selman, B.: The Science of Constraints. Constraint Programming Letters **1**(1), 15–20 (2007). URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.157.5887&rep=rep1&type=pdf
24. Gouda, M.G., Liu, A.X.: Structured firewall design. Computer Networks **51**(4), 1106–1120 (2007). DOI 10.1016/j.comnet.2006.06.015. URL http://linkinghub.elsevier.com/retrieve/pii/S1389128606001988
25. Guttman, J.: Filtering postures: Local enforcement for global policies. In: IEEE Symposium on Security and Privacy, pp. 120–129. IEEE Comput. Soc. Press (1997). DOI 10.1109/SECPRI.1997.601327. URL http://www.computer.org/portal/web/csdl/doi/10.1109/SECPRI.1997.601327
26. Guttman, J., Herzog, A., Thayer, F.: Authentication and Confidentiality via IP sec. Computer Security-ESORICS 2000 **1895**(June), 255–272 (2000). URL http://www.springerlink.com/index/vk033107x422xh36.pdf
27. Guttman, J.D., Herzog, A.L.: Rigorous automated network security management. International Journal of Information Security **4**(1-2), 29–48 (2004). DOI 10.1007/s10207-004-0052-x. URL http://www.springerlink.com/index/10.1007/s10207-004-0052-x
28. Hamed, H., Al-Shaer, E., Marrero, W.: Modeling and verification of IPSec and VPN security policies. In: Network Protocols, 2005. ICNP 2005. 13th IEEE International Conference on, p. 10. IEEE (2005). URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1544626
29. Han, W., Lei, C.: A survey on policy languages in network and security management. Computer Networks **56**(1), 477–489 (2012). DOI 10.1016/j.comnet.2011.09.014. URL http://dx.doi.org/10.1016/j.comnet.2011.09.014
30. Hari, a., Suri, S., Parulkar, G.: Detecting and resolving packet filter conflicts. In: INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE, vol. 3, pp. 1203–1212. IEEE (2000). DOI 10.1109/INFCOM.2000.

832496. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=832496http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=832496

31. Hidalgo, S.P., Ceballos, R., Gasca, R.M.: Fast Algorithms for Consistency-Based Diagnosis of Firewall Rule Sets. 2008 Third International Conference on Availability, Reliability and Security pp. 229–236 (2008). DOI 10.1109/ARES.2008.42. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4529342
32. Hinrichs, S.: Policy-based management: bridging the gap. Proceedings 15th Annual Computer Security Applications Conference (ACSAC'99) pp. 209–218 (1999). DOI 10.1109/CSAC.1999.816030. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=816030
33. Hinrichs, T., Gude, N., Casado, M., Mitchell, J., Shenker, S.: Expressing and enforcing flow-based network security policies. Tech. rep., University of Chicago (2009). URL http://people.cs.uchicago.edu/~thinrich/papers/hinrichs2008design.pdf
34. Hinrichs, T.L., Gude, N.S., Casado, M., Mitchell, J.C., Shenker, S.: Practical declarative network management. Proceedings of the 1st ACM workshop on Research on enterprise networking - WREN '09 p. 1 (2009). DOI 10.1145/1592681.1592683. URL http://portal.acm.org/citation.cfm?doid=1592681.1592683
35. Hong, J.: XML-based configuration management for IP network devices. IEEE Communications Magazine **42**(7), 84–91 (2004). DOI 10.1109/MCOM.2004.1316538. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1316538
36. Hozza, T.: STATIC ANALYSIS OF COMPUTER NETWORKS. Ph.D. thesis, Brno Universitz of Technology (2012)
37. Huang, S., Green, T., Loo, B.: Datalog and emerging applications: an interactive tutorial. Proceedings of the 2011 ACM SIGMOD . . . (2011). URL http://dl.acm.org/citation.cfm?id=1989456
38. Huang, Y., Feamster, N., Lakhina, A., Xu, J.J.: Diagnosing network disruptions with network-wide analysis. Proceedings of the 2007 ACM SIGMETRICS international conference on Measurement and modeling of computer systems - SIGMETRICS '07 p. 61 (2007). DOI 10.1145/1254882.1254890. URL http://portal.acm.org/citation.cfm?doid=1254882.1254890
39. Hyesook, L., Mun, J.H.: An efficient IP address lookup algorithm using a priority trie. In: GLOBECOM - IEEE Global Telecommunications Conference (2006). DOI 10.1109/GLOCOM.2006.347
40. Iannaccone, G., Chuah, C.n., Mortier, R., Bhattacharyya, S., Diot, C.: Analysis of link failures in an IP backbone. Proceedings of the second ACM SIGCOMM Workshop on Internet measurment workshop - IMW '02 p. 237 (2002). DOI 10.1145/637235.637238. URL http://portal.acm.org/citation.cfm?doid=637201.637238
41. Jackson, E.K., Schulte, W., Nikolaj Bjørne: Open-World Logic Programs: A New Foundation for Formal Specifications. Tech. rep., Microsoft Research (2013)
42. Jackson, E.K., Seifert, D., Dahlweid, M., Santen, T., Bjø rner, N., Schulte, W.: Specifying and composing non-functional requirements in model-based development. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) **5634 LNCS**, 72–89 (2009). DOI 10.1007/978-3-642-02655-3_7
43. Jaffar, J., Maher, M.J.: Constraint logic programming: a survey (1994). DOI 10.1016/0743-1066(94)90033-7
44. Jeffrey, A., Samak, T.: Model checking firewall policy configurations. In: Proceedings - 2009 IEEE International Symposium on Policies for Distributed Systems and Networks, POLICY 2009, pp. 60–67 (2009). DOI 10.1109/POLICY.2009.32
45. Kamara, S., Fahmy, S., Schultz, E., Kerschbaum, F., Frantzen, M.: Analysis of vulnerabilities in internet firewalls. Computers and Security **22**, 214–232 (2003). DOI 10.1016/S0167-4048(03)00310-9
46. Khan, B., Mahmud, M., Khan, M.K., Alghathbar, K.S.: Security analysis of firewall rule sets in computer networks. In: Proceedings - 4th International Conference on Emerging Security Information, Systems and Technologies, SECURWARE 2010, pp. 51–56 (2010). DOI 10.1109/SECURWARE.2010.16
47. Laborde, R., Barrere, F., Benzekri, A.: A formal framework for network security mechanisms configuration. In: IEEE International Symposium on Network Computing and Applications, Fourth, pp. 223–227 (2005)

48. Le, F., Xie, G.G., Zhang, H.: Understanding Route Redistribution. 2007 IEEE International Conference on Network Protocols pp. 81–92 (2007). DOI 10.1109/ICNP.2007.4375839. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4375839
49. Li, N., Mitchell, J.C.: Datalog with Constraints: A Foundation for Trust Management Languages. In: PADL, vol. 2562, pp. 58–73 (2002). URL http://www.springerlink.com/index/10.1007/3-540-36388-2_6$\delimiter"026E30F$npapers2://publication/doi/10.1007/3-540-36388-2_6
50. Liu, A., Torng, E., C.R., M.: Compressing Network Access Control Lists. IEEE Transactions on Parallel and Distributed Systems **22**(12), 1969 – 1977 (2011). URL http://ieeexplore.ieee.org/iel5/71/4359390/05740875.pdf?arnumber=5740875
51. Liu, A.X.: Change-impact analysis of firewall policies. Computer Security–ESORICS 2007 pp. 155–170 (2007). DOI 10.1007/978-3-540-74835-9_11. URL http://link.springer.com/chapter/10.1007/978-3-540-74835-9_11
52. Liu, A.X.: Formal verification of firewall policies. In: IEEE International Conference on Communications, pp. 1494–1498 (2008). DOI 10.1109/ICC.2008.289
53. Liu, A.X.: Firewall policy verification and troubleshooting. Computer Networks **53**(16), 2800–2809 (2009). DOI 10.1016/j.comnet.2009.07.003. URL http://linkinghub.elsevier.com/retrieve/pii/S1389128609002199
54. Liu, A.X.: Firewall policy change-impact analysis (2012). DOI 10.1145/2109211.2109212
55. Liu, A.X., Gouda, M.G.: Complete Redundancy Detection in Firewalls. Ifip International Federation For Information Processing pp. 196–209 (2005)
56. Liu, A.X., Gouda, M.G.: Firewall Policy Queries. IEEE Transactions on Parallel and Distributed Systems **20**(6), 766–777 (2009). DOI 10.1109/TPDS.2008.263. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4731249
57. Lobo, J., Bhatia, R., Naqvi, S.: A Policy Description Language. In: AAAI, pp. 291–298 (1999). URL http://aaaipress.org/Papers/AAAI/1999/AAAI99-043.pdf
58. Maltz, D., Xie, G., Zhan, J., Zhang, H., Hjálmt\'ysson, G., Greenberg, A.: Routing design in operational networks: A look from the inside. In: Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications, pp. 27–40. ACM (2004). URL http://portal.acm.org/citation.cfm?id=1015467.1015472
59. Matoušek, P., Ráb, J., Ryšavy, O., Sveda, M.: A Formal Model for Network-Wide Security Analysis. 15th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ecbs 2011) pp. 171–181 (2008). DOI 10.1109/ECBS.2008.13. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4492398
60. Menth, M., Duelli, M., Martin, R.: Resilience analysis of packet-switched communication networks. ACM Transactions on Networking **17**(6), 1950–1963 (2010). URL http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Resilience+Analysis+of+Packet-Switched+Communication+Networks#0
61. Narain, S.: Network configuration management via model finding. In: Proceedings of the 19th conference on Large Installation System Administration Conference-Volume 19, p. 15. USENIX Association (2005). URL http://portal.acm.org/citation.cfm?id=1251165
62. Narain, S.: Applying Formal Methods to Configuration Synthesis and Debugging (2009). URL http://dimacs.rutgers.edu/Workshops/DesigningNetworks/slides/sanjai-narain.ppt
63. Narain, S.: ConfigAssure: A Science of Configuration (2013). DOI 10.1109/MILCOM.2013.252. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6735834
64. Narain, S., Levin, G., Malik, S.: Declarative Infrastructure Configuration Synthesis and Debugging. Journal of Network and Systems pp. 1–26 (2008). URL http://www.springerlink.com/index/J786661856261 00L.pdf
65. Narain, S., Talpade, R., Levin, G.: Network Configuration Validation. Guide to Reliable Internet Services and Applications pp. 277–316 (2010). URL http://www.springerlink.com/index/TV31541071671040.pdf

66. Oppenheimer, D., Ganapathi, A., Patterson, D.: Why do Internet services fail, and what can be done about it? In: Proceedings of the 4th conference on USENIX Symposium on Internet Technologies and Systems-Volume 4, p. 1. USENIX Association (2003). URL http://portal.acm.org/citation.cfm?id=1251461
67. Ou, X., Govindavajhala, S., Appel, A.: MulVAL: A logic-based network security analyzer. 14th USENIX Security ... p. 8 (2005). URL http://portal.acm.org/citation.cfm?id=1251398.1251406$\delimiter"026E30F$nhttp://static.usenix.org/publications/library/proceedings/sec05/tech/full_papers/ou/ou_html/
68. Ou, X., Govindavajhala, S., Appel, A.: MulVAL: A logic-based network security analyzer. In: Proceedings of the 14th conference on USENIX Security Symposium-Volume 14, pp. 8–8. USENIX Association (2005). URL http://portal.acm.org/citation.cfm?id=1251406
69. Pranothi, N., Hemavathy, R.: A Survey of Network Device Configuration Audit Tools. International Journal of Advanced Networking and Applications **2**(2), 532–538 (2010). URL http://www.ijana.in/papers/sp1.9.pdf
70. Qian, J., Hinrichs, S., Nahrstedt, K.: ACLA: A framework for access control list (ACL) analysis and optimization. In: Communications and Multimeida Security, pp. 1–15 (2001). URL http://books.google.com/books?hl=en&lr=&id=NBN0IQVRa4oC&oi=fnd&pg=PA197&dq=ACLA+:+A+Framework+for+Access+Control+List+(+ACL+)+Analysis+and+Optimization&ots=-W0DDb5cDL&sig=GM6VLMY_kQbDN-Vw_D7h-RHd6IQ
71. Revesz, P.: Safe Datalog Queries with Linear Constraints. Principles and Practice of Constraint Programming — CP98 **1520**, 355–369 (1998). URL http://www.springerlink.com/index/Q02CKGKJ9XMJRGCE.pdf
72. Revesz, P.Z.: Safe query languages for constraint databases. ACM Transactions on Database Systems **23**(1), 58–99 (1998). DOI 10.1145/288086.288088. URL http://portal.acm.org/citation.cfm?doid=288086.288088
73. Samak, T.: DISCOVERY , GENERATION AND ANALYSIS OF NETWORK POLICY CONFIGURATIONS. Ph.D. thesis, DePaul University of Chicago (2010)
74. Sanders, W.H., Meyer, J.F.: Stochastic Activity Networks : Formal Definitions and Concepts. Lectures on formal methods and performance analysis **315-343**(9975019), 315–343 (2002). DOI 10.1007/3-540-44667-2_9. URL http://dx.doi.org/10.1007/3-540-44667-2_9
75. Schüpbach, A., Baumann, A., Roscoe, T., Peter, S.: A Declarative Language Approach to Device Configuration. ACM Transactions on Computer Systems **30**(1), 1–35 (2012). DOI 10.1145/2110356.2110361. URL http://dl.acm.org/citation.cfm?doid=2110356.2110361
76. Srinivasan, V., Varghese, G., Suri, S., Waldvogel, M.: Fast and scalable layer four switching. ACM SIGCOMM Computer Communication Review **28**(4), 191–202 (1998). URL http://portal.acm.org/citation.cfm?id=285282
77. Stone, G., Lundy, B., Xie, G.: Network policy languages: a survey and a new approach. IEEE Network **15**(1), 10–21 (2001). DOI 10.1109/65.898818. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=898818
78. Stone, G.N.: A Path-based Network Policy Language. Ph.D. thesis, Naval Postgraduate School (2000)
79. Sung, Y.W.E., Lund, C., Lyn, M., Rao, S.G., Sen, S.: Modeling and understanding end-to-end class of service policies in operational networks. Proceedings of the ACM SIGCOMM 2009 conference on Data communication - SIGCOMM '09 p. 219 (2009). DOI 10.1145/1592568.1592595. URL http://portal.acm.org/citation.cfm?doid=1592568.1592595
80. Sung, Y.W.E., Sun, X., Rao, S.G., Xie, G.G., Maltz, D.A.: Towards Systematic Design of Enterprise Networks. IEEE/ACM Transactions on Networking **19**, 695–708 (2011). DOI 10.1109/TNET.2010.2089640
81. Sveda, M., Rysavy, O., De Silva, G., Matousek, P., Rab, J.: Reachability analysis in dynamically routed networks. In: Proceedings - 18th IEEE International Conference and Workshops on Engineering of Computer-Based Systems, ECBS 2011, pp. 197–205 (2011). DOI 10.1109/ECBS.2011.24
82. Taylor, D.E.: ClassBench: A Packet Classification Benchmark. IEEE/ACM Transactions on Networking **15**(3), 135–511 (2007). DOI 10.1109/TNET.2007.893156
83. Tidwell, T., Larson, R., Fitch, K., Hale, J.: Modeling Internet Attacks. Network **1**, 5–6 (2001). URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.108.9040&rep=rep1&type=pdf

84. Vanbever, L.: Towards validated network configurations with NCGuard. Internet Network . . . pp. 1–6 (2008). DOI 10.1109/INETMW.2008.4660329. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4660329http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4660329
85. Voellmy, A., Kim, H., Feamster, N.: Procera. In: Proceedings of the first workshop on Hot topics in software defined networks - HotSDN '12, p. 43. ACM Press, New York, New York, USA (2012). DOI 10.1145/2342441.2342451. URL http://dl.acm.org/citation.cfm?doid=2342441.2342451
86. Wang, A., Basu, P., Loo, B., Sokolsky, O.: Declarative network verification. Practical Aspects of Declarative Languages pp. 61–75 (2009). URL http://www.springerlink.com/index/A62813T6267H0JH1.pdf
87. Wang, A., Jia, L., Liu, C., Loo, B., Sokolsky, O., Basu, P.: Formally verifiable networking. In: 8th Workshop on Hot Topics in Networks (ACM SIGCOMM HotNets-VIII) (2009). URL http://www.seas.upenn.edu/~liminjia/research/papers/fvn_hotnets.pdf
88. Wool, A.: Trends in Firewall Configuration Errors: Measuring the Holes in Swiss Cheese. IEEE Internet Computing **14**(4), 58–65 (2010). DOI 10.1109/MIC.2010.29. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5440153
89. Wu, K., Otoo, E.: Optimizing bitmap indices with efficient compression. ACM Transactions on Database Systems (**31**(1), 1–38 (2006). DOI 10.1145/1132863.1132864. URL http://portal.acm.org/citation.cfm?doid=1132863.1132864http://dl.acm.org/citation.cfm?id=1132864
90. Xie, G., Maltz, D., Greenberg, A., Hjalmtysson, G., Rexford, J.: On static reachability analysis of IP networks. Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies. pp. 2170–2183 (2005). DOI 10.1109/INFCOM.2005.1498492. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1498492
91. Yuan, L., Chen, H., Mai, J., Chuah, C.N., Su, Z., Mohapatra, P.: FIREMAN: A toolkit for firewall modeling and analysis. In: Proceedings - IEEE Symposium on Security and Privacy, vol. 2006, pp. 199–213 (2006). DOI 10.1109/SP.2006.16
92. Zakeri, R., Jalili, R., Abolhassani, H., Shahriari, H.R.: Using description logics for network vulnerability analysis. In: Proceedings of the International Conference on Networking, International Conference on Systems and International Conference on Mobile Communications and Learning Technologies,ICN/ICONS/MCL'06, vol. 2006 (2006). DOI 10.1109/ICNICONSMCL.2006.222
93. Zegura, E.W., Catlett, C., Clark, F., Dahlin, M., Feigenbaum, J., Forrest, S., Kearns, M., Lazowska, E., Nissenbaum, H., Peterson, L., Rexford, J., Shenker, S., Wroclawski, J.: Network Science and Engineering: Research Agenda. Tech. Rep. September, NetSE Counsil (2009)