NETWORK REACHABILITY: QUANTIFICATION, VERIFICATION, TROUBLESHOOTING, AND OPTIMIZATION

By

Amir Reza Khakpour

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

Computer Science and Engineering

2012

ABSTRACT

NETWORK REACHABILITY: QUANTIFICATION, VERIFICATION, TROUBLESHOOTING, AND OPTIMIZATION

By

Amir Reza Khakpour

Quantifying, verifying, troubleshooting, and optimizing the network reachability is essential for network management and network security monitoring as well as various aspects of network auditing, maintenance, and design. Although attempts to model network reachability have been made, feasible solutions for computing, maintaining and optimally designing network reachability have remained unknown. Network reachability control is very critical because, on one hand, reachability errors can cause network security breaches or service outages, leading to millions of dollars of revenue loss for an enterprise network. On the other hand, network operators suffer from lack of tools that thoroughly examine network access control configurations and audit them to avoid such errors. Besides, finding reachability errors is by no means easy. The access control rules, by which network reachability is restricted, are often very complex and manually troubleshooting them is extremely difficult. Hence, having a tool that finds the reachability errors and fix them automatically can be very useful. Furthermore, flawed network reachability design and deployment can degrade the network performance significantly. Thus, it is crucial to have a tool that designs the network configurations such that they have the least performance impact on the enterprise network.

In this dissertation, we first present a network reachability model that considers connectionless and connection-oriented transport protocols, stateless and stateful routers/firewalls, static and dynamic NAT, PAT, IP tunneling, etc. We then propose a suite of algorithms for

quantifying reachability based on network configurations (mainly access control lists (ACLs)) as well as solutions for querying network reachability. We further extend our algorithms and data structures for detecting reachability errors, pinpointing faulty access control lists, and fixing them automatically and efficiently. Finally, we propose algorithms to place rules on network devices optimally so that they satisfy the networks central access policies. To this end, we define correctness and performance criteria for rule placement and in turn propose cost-based algorithms with adjustable parameters (for the network operators) to place rules such that the correctness and performance criteria are satisfied.

We implemented the algorithms in our network reachability tool called Quarnet and conducted experiments on a university network. Experimental results show that the offline computation of reachability matrices takes a few hours and the online processing of a reachability query takes 75 milliseconds on average. We also examine our reachability error detection and correction algorithms on a few real-life networks to examine their performance and ensure that Quarnet is efficient enough to be practically useful. The results indicate that we can find reachability errors in order of minutes and fix them in order of seconds depending on the size of network and number of ACLs. Finally, we added the rule placement suite of algorithms to Quarnet, which can design a network ACL in based on the network central policies in order of tens of minutes for an enterprise network. We compare it with Purdue ACL placement, the state-of-the-art access policy design technique, and explain its pros and cons.

I dedicate this dissertation to by far the most valuable person in my life, my beloved mother,

Dr. Zohreh Oloomi

for her lifetime dedication to her sons, her compassion, and her unconditional love and support.

ACKNOWLEDGEMENTS

Foremost, I would like to express my sincere gratitude to my advisor Dr. Alex Liu for his continuous support of my Ph.D study and research, for his patience, motivation, and enthusiasm. I want to thank him for his generous two years support of Research Assistantship, which helped me immensely to focus on my research and finish my dissertation in a timely manner. Besides my advisor, I would like to thank the rest of my thesis committee Prof. Heydar Radha, Dr. Abdol-Hossein Esfehanian, and Dr. Li Xiao, for their encouragement and insightful comments during my Qualifier and Comprehensive exams.

I am pleased to thanks the Michigan State University, and more specifically Department of Computer Science and Engineering for providng me with a great opportunity of learning and doing research. I immensely appreciate their generous fellowship offer and two years of Teaching Assistantship, which not only supported me fully financially, but also was a priceless source of learning and gaining teaching experience for which I cannot be more grateful.

Many faculty members of the MSU CSE department assisted and encouraged me in various ways during my course of studies. I am especially grateful to Dr. Eric Torng, CSE graduate advisor, Dr. Pang-ning Tan, Dr. Rong Jin, and Dr. Abdol-Hossein Esfahanian for all that they have taught me. I was also greatly inspired by Prof. Philip McKinley, Dr. Li Xiao, Dr. Guoliang Xing, and Dr. Sandeep Kulkarni for whom I was a Teaching Assistant for two years, and I want to thank the students whom I was privileged to teach and from whom I also learned much. This is also a great opportunity to express my respect to Ms. Linda Moore for knowing the best answer to all my questions and always helping me promptly.

Many thanks to my friends in SENS lab, my second home for four years. In particular, I would like to thanks Dr. Chad Meiners, for insightful technical discussions, Dr. Borzoo Bonakdarpour, for his thoughtful tips on graduate school and academia, and the rest of our group for all their support and very profound discussions we often had in our group meetings. In addition, I would like to thank Mahmoud Taghizadeh, Dr. Hamed Valizadegan, Mehrdad Mahdavi, Joshua Hulst, and Seth Morash for helping and working with me on different projects and papers.

I must say that I owe my great time in East Lansing to all of my fabulous friends from all over the world. In particular, I want to thank all my Iranian friends and MSU Persian Student Association (PSA) board members, Rouhollah Jafari, Maryam Sayadi, Dr. Sohrab Soltani, Mehdi Aghagolzadeh, Ali Mohebi, and Fatemeh Noohi, for their friendship and support. Moreover, I would like to thank some other amazing friends, Candra O'Connor, Shiva Thompson, Melodi Litkouhi, Yasaman Back, Dr. Mehdi Sarmady, and Dr. Allison Brown, who brought joy to my life, and I am always grateful for that. Indeed, by having them around, all these cold years in Michigan are the warmest years of my life. I would never forget them and the memorable time we spent together.

During my studies, I worked with scholars, researcher, and entrepreneurs outside of MSU. I would like to thank AT&T Research center, more specifically, Dr. Jia Wang, Dr. Dan Pei, and Dr. Zihui Ge for mentoring me on Firewall Fingerprinting project. I also want to thank Alex Kazerani, the CEO of EdgeCast Networks, for offering me an internship position for six months, and taught me how one can be successful in industry and how good work pays off. I learned a lot from him, and I am sure I will learn more after my graduation.

Finally, I do not know how I can thank my family enough: my beautiful mother, Dr. Zohreh Oloomi, to whom I dedicate this dissertation, my great grandmother, Parvin

Kazemian, from whom I realized that kindness and devotion is endless, my brother, Hamid R. Khakpour, who always supports me no matter what, my uncle, Dr. Mehdi Oloomi, whom I always look up to, and the rest of the family members who were always supportive of my studies.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

1.1 Motivation

Although computer networks are created to provide reachability between end hosts, various sophisticated mechanisms, such as router Access Control Lists (ACLs) and firewalls, have been deployed to limit such reachability for security or privacy purposes.As a critical infrastructure of national importance, the Internet faces unprecedented challenges for reliable reachability. Correctly implementing the exact reachability that is needed for a large interconnected network is crucial. While more reachability than necessary may open doors to unwanted and even malicious traffic causing sometimes irreparable damages, less reachability than necessary may disrupt normal businesses causing huge revenue losses.

Unfortunately, in reality, network reachability management is often a mess for most networks due to its high complexity and the lack of advanced reachability debugging tools. First, network configurations have become far more complex than even a skilled operator can correctly manage. The complexity of modern networks has been rapidly increasing due to the explosive growth of Internet connectivity expanding from end-hosts to pervasive devices and

network supported applications of various scales. Second, due to the lack of advanced reachability debugging tools, the current common practice for reachability management is still "trial and error", which of course results in a plethora of reachability errors. Configuration errors have been observed to be the largest cause of failure for Internet services [OGP03]. A recent Yankee Group report has shown that more than 62% of network downtime is due to human configuration errors and more than 80% of IT budgets are allocated towards maintaining just the status quo [Ker04]. Network operators face tremendous pressure to fix problems quickly because operational networks often support critical business applications and important communications; thus, the loss caused by network outrages becomes increasingly acute. For example, the estimated revenue losses per hour of downtime for the industry of media, banking, and brokerage are 1.2, 2.6, and 4.5 million dollars, respectively [Ker04]. Quantitative studies have shown that most firewalls are misconfigured [Woo04]. The reality could be worse than these published staggering numbers as the errors of more reachability than needed are often undetected, while less reachability than needed is a common source of complaints to operators. Therefore, a scientific approach, instead of "trial and error", is needed to manage network reachability, which will continue to grow more complex as networking technologies evolve.

As organizations continually transform their network infrastructure to maintain their competitive edge by deploying new servers, installing new software and services, expanding connectivity, etc, operators constantly need to manually modify their network configurations. Ideally, such manual modifications should exactly implement the desired changes; however, practically, they often come with undesirable, yet unnoticed, side effects on reachability. Therefore, an automated tool that monitors the all-to-all reachability of a large interconnected network in real-time is crucial to its successful management. Moreover, querying and

verifying reachability is a routine, yet error-prone, task for operators. Hence, an automated reachability query and verification tool as well as an automated reachability troubleshooting tool that includes error detection and correction can be very critical for a network operator to find errors proactively and fix them accordingly.

As network reachability deployment can have a significant impact on the performance of the network, optimal design of the ACLs and their deployment is very essential for the network operators, yet since it is too difficult to do so, it is often ignored. An ideal reachability deployment requires early discarding of unwanted traffic in the path to minimize the wasted resources, and using the network devices capabilities efficiently to have the minimum end-to-end delay between hosts in source and destination subnetworks. Early discarding of unwanted packets reduces the number of unnecessary packet forwardings and probable network congestions. Moreover, network devices have limited resources and capabilities; thus, they should be used wisely. For instance, NetScreen-100 only allows 733 rules [LTM08], which is an important fact that must be considered when the ACLs are designed. Another example includes software firewalls (*e.g.* IPTables), which often scans through the ACL (or the firewall policy) to find the first match for a given packet. For such firewalls, smaller number of rules can lead to lower average packet processing time. Hence, it is highly advised to keep the ACL size small so that the average packet processing time is minimized.

To enforce the right amount of network reachability, no more and no less, for a large interconnected enterprise network to achieve access control, security, and privacy, in this dissertation, we investigate four aspects of network reachability: quantification, querying, troubleshooting, and optimization, and present a reachability management tool, which we call Quarnet. Quantifying the reachability between any two subnets means to compute the set of packets that can traverse from one subnet to another based on network configura-

tions. Querying reachability means to ask questions like "who can access what?". Network reachability troubleshooting enables the network operator to find errors and fix them automatically, which potentially avoid network outages and security breaches. Finally, reachability optimization helps the operator to design the network ACLs automatically, correctly, and optimally in terms of performance. Such functionalities are useful in many aspects of network management:

(1) *Network Security Monitoring and Auditing*: Verifying that the deployed ACLs satisfy certain security specifications is an integral part of network security monitoring and auditing. The current practice is to send probing packets. However, this approach has drawbacks. First, it is infeasible to generate all possible probing packets. Second, as routing tables change over time, the auditing result is valid only for a specific time. Combining the current active probing approach with quantitative network reachability analysis, we can have comprehensive security auditing for large enterprise networks.

(2) *Network Troubleshooting*: An important task for network operators is to troubleshoot reachability problems when two hosts fail to communicate or there is unauthorized traffic passing through a series of ACLs. For large complex networks, troubleshooting reachability problems is extremely difficult. To check the reachability of a path, the current practice is to check the reachability for every hop (*i.e.*, routers or firwalls) in the path by actively sending probing packets. This approach is disruptive, time consuming, and sometimes infeasible when isolating some hops is impossible. In contrast, when the reachability of every path has been pre-computed, troubleshooting reachability problems and identifying faulty ACLs is easy. Quarnet can help to detect reachability errors, pinpoint faulty ACLs and automatically fix them.

(3) *Network Security Design, and Maintenance*: Designing ACLs based on certain se-

curity policies is a critical part of network design. If ACLs are designed manually, which is often the case, it is important to first verify reachability before deployment. This helps to avoid security breaches and service outbreaks caused by misconfigured ACLs. Second, the access policy must be deployed on network devices such as routers and firewalls. As mentioned, the correct deployment of access policy is not easy because of the complexity of access policy rules. Furthermore, the access policies can have an impact on the network performance. Thus, they should be deployed very carefully and of course correctly. In addition, networks change over time with the evolving of topology and connected servers/hosts. Often network changes require corresponding changes on ACL rules. Due to the interconnected nature of networks, a modification on one ACL may have unnoticed side effects on network reachability such as causing two servers to fail to communicate or opening security holes by enabling unauthorized accesses. Thus, network reachability analysis helps to detect and resolve potential problems before committing any change to ACLs.

As an alternative approach, it will be important to design ACLs and deploy them on the network routers and firewalls automatically based on the network access policy. Such ACL design must fully address the correctness and performance concerns.

1.2 Problem Statement

1.2.1 Reachability Quantification

Given a network topology and all the ACLs deployed on the network routers and firewalls, what is the set of packets that can traverse the network from one subnet to another subnet?

1.2.2 Reachability Verification

Given a network topology and all the ACLs deployed on the network routers, how to verify certain properties to ensure that the access control configuration are correct and sound?

1.2.3 Reachability Troubleshooting

The problem of reachability troubleshooting can be stated as two subproblems: (1) Given an error and all the ACLs on the current path between a source and destination subnet, how to pinpoint faulty rule(s) and ACL(s) and how to fix it? (2) Given a network topology and all the ACLs deployed on the network routers, how to proactively find reachability errors, pinpoint faulty rule(s) and ACL(s) and fix them?

1.2.4 Reachability Optimization

Given a network topology, its central access policy, and its performance criteria, what is the ACL for each individual classifier such that the ACLs all together enforce the correct network reachability that satisfies the not only the network central access policy, but also the network performance criteria?

1.3 Technical Challenges

Quantifying and verifying network reachability are hard. First, from the reachability perspective, the interaction among the rules in one ACL is already complex due to the multi-dimensionality of ACL rules, but the interaction of multiple ACLs in interconnected networks is even more complex. Second, routing and network topology have complex impact on reach-

ability. There is typically more than one path between a source and a destination, and a packet may only be able to traverse from the source to the destination via some, but not all, available paths. Third, middleboxes often have complex impact on reachability. For example, packet transforming middleboxes (such as NAT and PAT) and IP tunneling complicate reachability calculation because they modify packets headers when they are traveling in a network. Fourth, transport layer protocols also complicate reachability calculation because, for connection-oriented protocols, the reachability of both data path and signalling path should be taken into account. It is even more challenging while there are some stateful middleboxes in the path. Last but not least, the problem space is huge as the ACL rules are typically specified over the standard 5-tuple with 104 bits in total.

In addition, for network troubleshooting we need to deal with a new set of challenges. First, once the faulty ACLs that cause a reachability error are detected, we may have a choice to fix one of the ACLs and resolve the error; however, finding the "best" ACL to fix is not straight-forward. Second, proactively finding reachability errors, pinpointing faulty rules and ACLs and fixing them is complicated. Third, since different paths between a pair of subnets mingle together, their reachabilities are tightly dependent. Thus, to fix a reachability error we may have a choice to find the best set of faulty ACLs to fix and resolve an error; however, finding the best set of faulty ACL is a non-trivial problem.

For network reachability optimization, placing complex rules on paths that are not disjoint is not a trivial task. As rules often overlap, misplacement of a rule can result in wrong reachability, which in turn causes network service inaccessibility or a security breach. Moreover, addressing multiple performance criteria at the same time to place rules optimally is a challenging task. While we want to place rules close to the source subnetwork, a classifier may have limited capacity for packet processing and accommodating rules. Since a network

often includes hardware and software classifiers with different capabilities, the optimal rule placement is even more difficult. In practice, hardware classifiers allow a limited number of rules, yet their packet processing time is constant, whereas in software classifiers more number of rules leads to greater packet processing delay.

1.4 Our Approach and Solution

In this dissertation, we present Quarnet, a tool that comprises a suite of concrete algorithms for quantifying, querying, troubleshooting, and optimizing network reachability. For reachability quantification, Quarnet takes a network topology and the ACLs deployed on middleboxes as its input, and outputs reachability matrices that represent the lower-bound reachability (*i.e.*, the minimal set of packets that can traverse from a source to a destination at any time), instantaneous reachability (*i.e.*, the set of packets that can traverse from a source to a destination at a particular time), and upper-bound reachability (*i.e.*, the maximal set of packets that can traverse from a source to a destination at some time) for every pair of source and destination subnets. For reachability querying, Quarnet allows network operators to ask questions about the reachability of the subnets in their network, *e.g.*, "which hosts in subnet A can access the mail server in subnet B?". We proposed a language for formally specifying reachability queries. To efficiently process queries, we use decision diagrams as the data structure for representing reachability matrices. For reachability troubleshooting, Quarnet enables network operators to pinpoint faulty ACLs and fix them automatically. Fixing errors can be done reactively upon the request of network operator, or proactively by comparing reachability lower-bounds and upper-bounds. For reachability optimization, Quarnet takes the network topology and network central access policy (either directly from

the network operator or compute it using the deployed ACLs) and outputs the optimal ACL for all classifiers in the network.

Quarnet can be deployed on a server as shown in Figure 1.1. Initially, this server first collects the configuration file from each middlebox (via SNMP for example) and the network topology from the operator and then computes reachability matrices offline. Afterwards, network operators can perform reachability queries through a GUI interface, where each query is then formulated by an SQL-like query language and processed by the Quarnet query module. Fine-grained access control to reachability matrices can be enforced so that different operators can perform different reachability queries. Each time the network operator changes the configuration of a middlebox or the topology of the network, the Quarnet server needs to be notified, and the reachability matrices need to be updated accordingly. If some reachability errors are reported either network users or network security appliances, an operator can use Quarnet to pinpoint the faulty ACLs and fix them automatically. Quarnet can report reachability inconsistencies caused by configuration errors on multiple paths from two subnets automatically and fix them accordingly.

Quarnet can be also used in network reachability design. If the network operator provides the Quarnet with the network central access policy, it can find the correct ACL design that also addresses the network performance criteria. If the ACLs of a network have been already deployed on the classifiers, the Quarnet server can collect them all, calculate the central access policy, come up with correct and optimal ACL design, and accordingly push the new ACLs to the network classifiers.

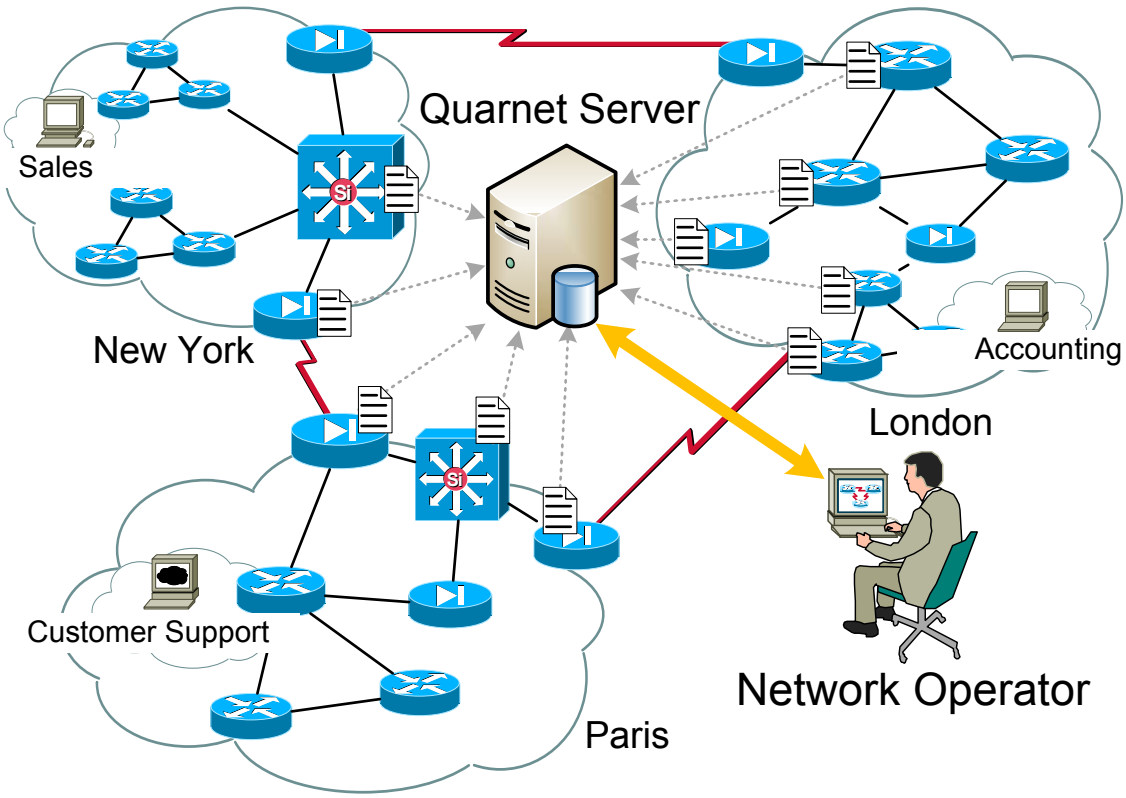


Figure 1.1: Quarnet architecture (For interpretation of the references to color in this and all other figures, the reader is referred to the electronic version of this dissertation)

1.5 Key Contributions

Our key contributions in this dissertation are as follows.

1. We propose a comprehensive reachability modeling and formulation that encompasses dynamic NAT, PAT, IP tunneling, connection orientation of transport protocols, and statefulness of middleboxes.

2. We propose efficient algorithms for computing network reachability using Firewall Decision Diagrams (FDDs).

3. We propose a probabilistic network reachability calculation technique to calculate reachability in a more effective and realistic way.

4. We propose a language as well as solutions for querying network reachability.

5. We formulate two types of network reachability errors.

6. We extend FDD data structure to support "rule and ACL tracking feature", which facilitates pinpointing faulty rules and ACLs.

7. We propose a suite of algorithms to detect reachability errors, pinpoint faulty rules and ACLs, and fix them automatically considering network performance metrics.

8. We introduce a suite of algorithms to compute the optimal access policy deployment, which satisfies the network cental access policy and complies with the network performance criteria.

9. We implemented Quarnet in C++ and experimented on real-life enterprise networks to evaluate the performance of proposed algorithms in terms of execution time and memory usage.

1.6 Organization

The rest of the dissertation proceeds as follows. We first present our network reachability model in chapter 2 and algorithms for computing reachability for networks without NAT/PAT in chapter 3. We present algorithms for computing reachability for networks with NAT/PAT in chapter 4. Querying network reachability is presented in chapter 5. We define reachability errors and inconsistencies and propose algorithms to detect them in chapter 6. We then explain how these errors can be fixed automatically and efficiently in chapter 7. Network reachability optimization algorithms are explained in chapter 8. We dedicate

chapter 9 to discussion. We show and explain the experimental results in chapter 10. We review related work later in the dissertation in chapter 11 for ease of understanding. Finally, we summarize the work and explore future work in chapter 12.

The academic papers for this dissertation are in submission state; however, the major part of this dissertation has been published in [KL10].

Chapter 2

Reachability Modeling and Formulation

In this chapter, we first introduce a network model, based on which we formulate three network reachability metrics: *lower-bound reachability*, *instantaneous reachability*, and *upper-bound reachability*. We also differentiate network reachability formulations based on transport protocol types and the presence of network address translation.

2.1 Network Modeling

In this dissertation, we model a network as a non-simple directed bipartite graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{F})$, where $\mathcal{V}$ is a set of vertices, $\mathcal{E}$ is a set of arcs over $\mathcal{V}$, and $\mathcal{F}$ is a set of ACLs. Each vertex in $\mathcal{V}$ represents a subnet or a middlebox. We use the term "subnet" to represent a set of adjacent subnetworks (*i.e.*, local area networks (LANs) or VLANs, where either they have the same reachability (*i.e.*, there is no ACL deployed between any two subnetworks in the set) or the reachability among the subnetworks is not a concern. For example, given an

enterprise network, we represent the outside Internet as a subnet. The term "middlebox" refers to any networking device that can forward packets from one subnet to another, such as a network router, a firewall, a traffic shaper, or a L3 switch. Let $\mathcal{N}$ be the set of subnets in $\mathcal{V}$ and $\mathcal{R}$ be the set of middleboxes in $\mathcal{V}$. Each arc in $\mathcal{E}$ represents a unidirectional physical link between a subset in $\mathcal{N}$ and a middlebox in $\mathcal{R}$. Each ACL in $\mathcal{F}$ filters the packets traveling on an arc in $\mathcal{E}$.

We model network links as unidirectional arcs because every ACL is associated with a unidirectional physical link and each bidirectional link can be modeled as two unidirectional arcs. Note that some physical links, such as satellite links, are physically unidirectional. We model a network as a bipartite graph because between two adjacent subnets there is at least one middlebox and between any two middleboxes there exists at least one subnet. We model a network as a non-simple graph because between a subnet and a middlebox there may exist multiple physical links for backup.

Given a network with m middleboxes, where the maximum number of unidirectional interfaces on a middlebox is denoted u, we represent the network by an $m \times u$ matrix $\mathcal{I}$ called the *Network Incident Matrix*. Here $\mathcal{I}[i, j] = N$ if and only if subnet N connects to middlebox i on its interface j, and $\mathcal{I}[i, j] = 0$ if and only if no subnet connects to middlebox i on its interface j. For simplicity, incoming interfaces are represented by even numbers and outgoing interfaces are represented by odd numbers. Similarly, we represent the ACLs deployed on the network by an $m \times u$ matrix $\mathcal{A}$ called the *ACL Matrix*. We use $\mathcal{A}[i, j]$ to denote the ACL deployed on the j-th interface of middlebox i.

Figure 2.1(a) shows a network with three middleboxes and four subnetworks. Two VLANs (S1 and S2) are connected to an L3 switch (SW). One subnetwork (S3) and a DMZ (S4) are connected to firewall (FW). SW and FW are connected to the Internet through a gateway

router (GW). Figure 2.1(b) shows graph representing the topology. Note that we assume there is an ACL on each interface of the middleboxes. The graph consists of 11 vertices representing the 8 subnets S1, ..., S8 and the 3 middleboxes SW, FW, and GW. Note that S1, ..., S4 denote the four subnetworks LAN1, LAN2, LAN3, and DMZ, S5 denotes the outside Internet, and S6, ..., S8 denote the subnetworks that connects two adjacent middleboxes. The network incident matrix for this network with $m = 3$ and $u = 8$ is as follows:

$$\mathcal{I} = \begin{pmatrix} S1 & S1 & S2 & S2 & S6 & S6 & S7 & S7 \\ S3 & S3 & S4 & S4 & S7 & S7 & S8 & S8 \\ S5 & S5 & S6 & S6 & S8 & S8 & 0 & 0 \end{pmatrix}$$

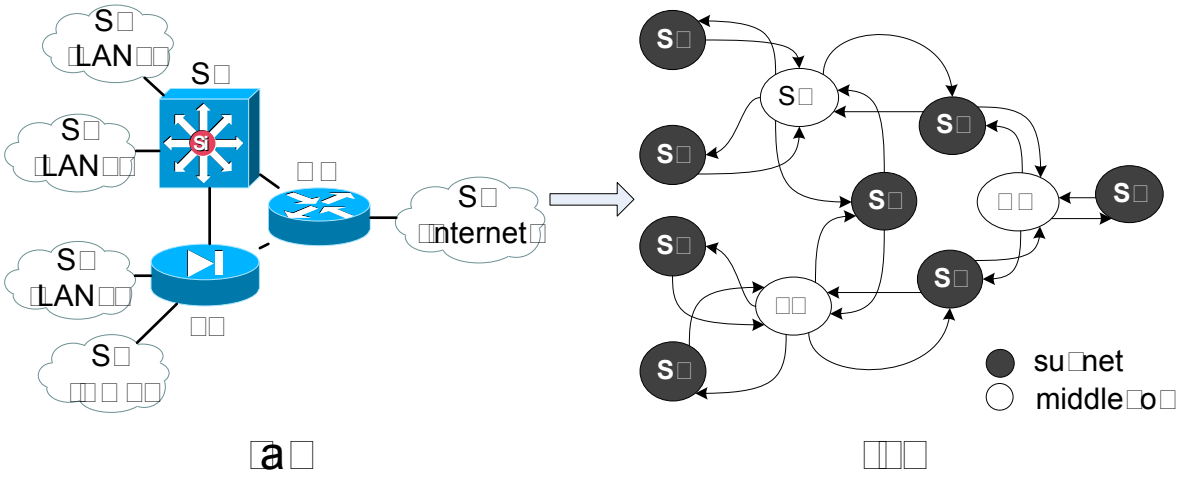


Figure 2.1: Example network topology

Between any two directly connected middleboxes, our model assumes there is a subnetwork because the middlebox interfaces may have a distinct IP address and an ACL guarding that interface. The reachability of such a subnetwork is important because of several reasons. First, there are often some management services on a middlebox (such as SNMP, Telnet, and

SSH) that are accessible through each interface. Such services are intended to be used only by network operators. Therefore, there are often some rules in the ACL deployed on each interface to restrict the access to this subnetwork. Second, if a middlebox is compromised, the source of the subsequent attacks is the IP address of an interface of the middle box; thus, the reachability from that subnetwork to other subnetworks is critical. Indeed, if the interfaces are not assigned IP addresses, the subnet is modeled with an empty address set; henceforth, reachability to and from the subnetwork is empty.

An ACL consists of a list of rules, where each rule has a predicate over some packet header fields and a decision (*i.e.*, action) to be taken for the packets that match the predicate. The decision of a rule is typically *accept* (*i.e.*, *permit*) or *discard* (*i.e.*, *deny*). As a packet may match two rules in an ACL and the two rules may have different decisions, the decision for a packet is the decision of the first (i.e., highest priority) rule that the packet matches. Table 2.1 shows an example ACL.

Rule	Src IP	Dst IP	Src Port	Dst Port	Proto.	Action
r_1	35.9.1.0/24	192.168.0.1	*	80	TCP	accept
r_2	35.9.1.0/24	192.168.0.10	*	8080	TCP	discard
r_3	35.9.1.0/24	192.168.0.0/24	*	*	*	accept
r_4	*	*	*	*	*	discard

Table 2.1: An example ACL

2.2 Reachability Formulation

Network reachability depends on not only some static factors, *i.e.*, topology and ACL configurations, but also some dynamic factors, *i.e.*, routing states, where each is defined as a snapshot of all the routing tables of the middleboxes in the network, and the one-to-one

mapping tables of dynamic NATs and PATs. We formulate three types of network reachability: lower-bound reachability, instantaneous reachability, and upper-bound reachability for a given network topology and the ACL configurations. We define the *lower-bound reachability* from subnet N_i to N_j as the set of packets that can go from N_i to N_j *at any time*. We define the *instantaneous reachability* from subnet N_i to N_j as the set of packets that can go from N_i to N_j *at a particular time*. We define the *upper-bound reachability* from subnet N_i to N_j as the maximal set of packets that can go from N_i to N_j *at some time*.

Below we formulate instantaneous, upper-bound, and lower-bound reachability. In our notations, we use "I", "U", and "L" to denote instantaneous, upper-bound, and lower-bound reachability, respectively; we further use "CL" and "CO" to denote connectionless and connection-oriented protocols.

In formulating network reachability, we differentiate connectionless transport protocols (*e.g.*, UDP, ICMP) and connection-oriented transport protocols (*e.g.*, TCP). In connectionless transport protocols, only the destination needs to be reachable from the source. However, in connection-oriented transport protocols, both the source and destination need to be reachable from each other to ensure the transmission of acknowledgment messages between them. For connection-oriented transport protocols, we further differentiate the following three cases based on the statefulness of the routers on the path from a source to a destination: (1) all routers are stateless, (2) all routers are stateful, and (3) some routers are stateless and some are stateful. For stateful routers, because they monitor connection states, they allow signaling packets associated with an existing connection to pass through and drop signaling packets that are not associated with any connection. For stateless routers, they match all signaling packets against their ACLs. Therefore, in calculating the reachability of stateless routers, we need to consider the impact of their ACLs on signaling packets.

2.2.1 Instantaneous Reachability

Given a network routing state s, which is a snapshot of all the routing tables of the middleboxes in the network, let $P_{i,j}(s)$ denote the path from N_i to N_j at state s, and M denote the number of hops/middleboxes on path $P_{i,j}(s)$. For the k-th middlebox, we use C_{2k-1} to denote the ACL on the incoming middlebox interface and C_{2k} to denote the ACL on the outgoing middlebox interface.

For connectionless protocols (mainly the UDP protocol), the instantaneous reachability from N_i to N_j is the intersection of the set of UDP packets accepted by every ACL on the path from N_i to N_j. Thus, we calculated instantaneous reachability as follows:

$$R^I_{CL}(i,j,s) = \bigcap_{k=1}^{2M} A_{UDP}(C_k) \tag{2.1}$$

where $A_{UDP}(C_k)$ is the set of UDP packets accepted by C_k.

For connection-oriented protocols (mainly the TCP protocol), the instantaneous reachability from N_i to N_j also depends on the reachability of the acknowledgment (ACK) messages from N_j to N_i. To incorporate the signaling path reachability of data path $P_{i,j}(s)$, we distinguish the statefulness of the intermediate middleboxes according to the following three cases: all middleboxes in $P_{i,j}(s)$ are stateful, all middleboxes in $P_{i,j}(s)$ are stateless, and $P_{i,j}(s)$ contains both stateful middleboxes and stateless middleboxes.

All middleboxes in $\mathbf{P_{i,j}(s)}$ are stateful: In any stateful middlebox on path $P_{i,j}(s)$, the state of every TCP session is stored in a state table to ensure that the corresponding signaling messages can traverse back from N_j to N_i. Such messages are not checked against any ACL of the middleboxes on path $P_{j,i}(s)$. When a signaling message does not match any entry in the state table of a stateful middlebox, the message is dropped and the connection

will fail. Here we assume that the network is designed such that we have *path-coupled* signaling on stateful firewalls and NAT, which means that the forward data path and the backward signaling path contain the same set of middleboxes. The path-coupled property holds for most existing networks [STA05, KSK+05]. Thus, when all middleboxes in $P_{i,j}(s)$ are stateful, the instantaneous reachability from N_i to N_j is the intersection of the set of TCP packets accepted by every ACL on the path from N_i to N_j. Therefore, we calculated instantaneous reachability for this case as follows, where $A_{TCP}(C_k)$ represents the set of TCP packets accepted by ACL C_k.

$$R^I_{CO}(i,j,s) = \bigcap_{k=1}^{2M} A_{TCP}(C_k) \tag{2.2}$$

All middleboxes in $\mathbf{P_{i,j}(s)}$ are stateless: If all the intermediate middleboxes are stateless, we not only need to consider the data path from N_i to N_j, but also the signaling path from N_j to N_i. Let $\tilde{A}$ represent the set of accepted packets where in each packet the values of source and destination IP address fields are swapped, and the values of source and destination port number fields are also swapped. Field swapping is needed because in one TCP session each data packet and its corresponding signaling packet have their IP addresses and port numbers in the opposite order. Note that when all middleboxes in $P_{i,j}(s)$ are stateless, we do not need path-coupled assumption. Thus, the instantaneous reachability for the connection-oriented and reliable protocols is the intersection of the set of accepted TCP packets in the data path and the set of accepted TCP packets in the signaling path. Therefore, we calculated instantaneous reachability for this case as follows, where the ACLs on path $P_{j,i}$ consists of $C'_1, C'_2, \ldots, C'_{2M'}$.

$$R^{I}_{CO}(i,j,s) = \bigcap_{k=1}^{2M} A_{TCP}(C_k) \bigcap_{k=1}^{2M'} \tilde{A}_{TCP}(C'_k) \tag{2.3}$$

(Here $\bigcap_{k=1}^{2M} A_{TCP}(C_k) \bigcap_{k=1}^{2M'} A_{TCP}(C'_k)$ denotes $A_{TCP}(C_1) \cap \cdots \cap A_{TCP}(C_{2M}) \cap A_{TCP}(C'_1)1 \cap \cdots \cap A_{TCP}(C'_{2M'})$.)

$\mathbf{P_{i,j}(s)}$ contains both stateful middleboxes and stateless middleboxes: For stateful middleboxes, we again need assumption of path-coupled signaling. For stateless middleboxes, we do not need this assumption. Thus, the instantaneous reachability on $P_{i,j}(s)$ is the intersection of the set of accepted packets of stateful middleboxes calculated by formula (2.2) and the set of accepted packets of stateless routers calculated by formula (2.3).

2.2.2 The Reachability Bounds

The *Reachability Lower-bound* from N_i to N_j, $R^L(i,j)$, denotes the set of packets that can traverse from N_i to N_j in *all* routing states. The *Reachability Upper-bound* from N_i to N_j, $R^U(i,j)$, denotes the maximal set of packets that can traverse from N_i to N_j in *some* routing states. Let $\mathcal{S}$ denote the set of all routing states of a network. The reachability lower-bound and upper-bound from N_i to N_j are calculated below:

Based on above definitions, the reachability upper-bound from N_i to N_j is the union of the instantaneous reachability of all paths from N_i to N_j and the corresponding lower-bound is the intersection of the instantaneous reachability of all paths from N_i to N_j. Thus, for connectionless protocols, the reachability bounds are calculated as follows:

$$R^{U}_{CL}(i,j) = \bigcup_{s \in \mathcal{S}} R^{I}_{CL}(i,j,s) \tag{2.4}$$

$$R^L_{CL}(i,j) = \bigcap_{s \in \mathcal{S}} R^I_{CL}(i,j,s) \tag{2.5}$$

Similar to the reachability bounds for connectionless protocols, the reachability bounds of the connection-oriented protocols using formulas (2.2) and (2.3) are calculated below:

$$R^U_{CO}(i,j) = \bigcup_{s \in \mathcal{S}} R^I_{CO}(i,j,s) \tag{2.6}$$

$$R^L_{CO}(i,j) = \bigcap_{s \in \mathcal{S}} R^I_{CO}(i,j,s) \tag{2.7}$$

Computing reachability lower-bound and upper-bound is very useful. For example, lower-bound reachability can be used to ensure that the available services on a subnet are reachable regardless of routing states, and upper-bound reachability can be used to ensure that the access to some services is restricted. Furthermore, the reachability upper-bound and lower-bound are useful in verifying the correctness of ACLs. Ideally, the reachability upper-bound and lower-bound from N_i to N_j should be the same (*i.e.*, $\Delta R(i,j) = R^U(i,j) - R^L(i,j)$ should be $\emptyset$). Otherwise, the ACLs have inconsistent decisions for the packets in $\Delta R(i,j)$: sometimes they are allowed to traverse from N_i to N_j, and sometimes they are not. For a packet $\hbar \in \Delta R(i,j)$, if $\hbar$ should be constantly allowed to traverse from N_i to N_j, then blocking $\hbar$ at some routing states may disrupt legitimate services; if $\hbar$ should be constantly disallowed to traverse from N_i to N_j, then accepting $\hbar$ at some states may cause security breaches.

2.3 Reachability Formulation with NAT

Thus far, network reachability calculations are based on the assumption that packet header fields are not changed in the traversal from a source subnet to a destination subnet. Actually, there may be some packet transformers, such as Network Address Translation (NAT) and Port Address Translation (PAT), on the intermediate middleboxes that modify packet headers. A NAT transformer on a middlebox may change the source address field of a packet from x to x' and keep a record of this transformation in a table, which is used to change the destination field of the corresponding signaling packet from x' to x. A PAT transformer works similarly for port fields. Here, the path-coupled signaling assumption is necessary for paths that contain packet transforming filters. Next, we present reachability formulations for static and dynamic NAT.

2.3.1 Static NAT

Typically, a middlebox (such as a Cisco router [cis05] and IPtables [And06]) applies NAT to a packet after it passes the ACL on the incoming interface and before it is sent to the ACL on the outgoing interface. Let middlebox γ be the one on path $P_{i,j}(s)$ that uses a packet transformation (for source address or port number fields) function $T_S : N_i \mapsto N_i{}'$, where $N_i{}'$ is the virtual subnet to which N_i is mapped. We use T_S^{-1} to denote the reverse function. The instantaneous reachability for connectionless protocols is calculated using formula (2.1) as follows:

$$R^I_{CL}(i,j,s,T_S) = \bigcap_{k=1}^{2\gamma-1} A_{UDP}(C_k) \bigcap T_S^{-1}\Big(\bigcap_{k=2\gamma}^{2M} A_{UDP}(C_k)\Big) \tag{2.8}$$

Note that applying function T_S^{-1} to $\bigcap_{k=2\gamma}^{2M} A_{UDP}(C_k)$ means changing the source fields of every packet in $\bigcap_{k=2\gamma}^{2M} A_{UDP}(C_k)$ from $N_i{}'$ to N_i.

The reachability bounds for connectionless protocols are calculated using formulas (2.4) and (2.5) as follows:

$$R_{CL}^{U}(i,j,T_S) = \bigcup_{s\in\mathcal{S}} R_{CL}^{I}(i,j,s,T_S) \tag{2.9}$$

$$R_{CL}^{L}(i,j,T_S) = \bigcap_{s\in\mathcal{S}} R_{CL}^{I}(i,j,s,T_S) \tag{2.10}$$

For connection-oriented protocols, the middlebox γ in the data path is the middlebox γ' in the signaling path (based on the path-coupled assumption). The instantaneous reachability formulation for data paths $R_{CO}^{\rightarrow}(i,j,s,T_S)$ is as follows:

$$R_{CO}^{\rightarrow}(i,j,s,T_S) = \bigcap_{k=1}^{2\gamma-1} A_{TCP}(C_k) \bigcap T_S^{-1}\Big(\bigcap_{k=2\gamma}^{2M} A_{TCP}(C_k)\Big) \tag{2.11}$$

Similarly, the instantaneous reachability formulation for signaling path $R_{CO}^{\leftarrow}(j,i,s,T_D)$ is as follows:

$$R_{CO}^{\leftarrow}(j,i,s,T_D) = T_D^{-1}\Big(\bigcap_{k=1}^{2\gamma'-1} A_{TCP}(C_k')\Big) \bigcap_{k=2\gamma'}^{2M'} A_{TCP}(C_k') \tag{2.12}$$

where T_D transforms the destination addresses of signaling packets from N_i to N_i'.

Using formulas (2.11), (2.12), and (2.3), we formulate instantaneous reachability for connection-oriented protocols as:

$$R^{I}_{CO}(i,j,s,T_S,T_D) = R^{\rightarrow}_{CO}(i,j,s,T_S) \cap \tilde{R}^{\leftarrow}_{CO}(j,i,s,T_D) \tag{2.13}$$

Note that formula (2.13) can be easily generalized to handle the paths that have multiple packet transformers.

The reachability bounds for connection-oriented protocols are formulated based on equations (2.6) and (2.7) as follows:

$$R^{U}_{CO}(i,j,T_S,T_D) = \bigcup_{s\in\mathcal{S}} R^{I}_{CO}(i,j,s,T_S,T_D) = \bigcup_{s\in\mathcal{S}} R^{\rightarrow}_{CO}(i,j,s,T_S) \bigcap \bigcup_{s\in\mathcal{S}} \tilde{R}^{\leftarrow}_{CO}(j,i,s,T_D) \tag{2.14}$$

$$R^{L}_{CO}(i,j,T_S,T_D) = \bigcap_{s\in\mathcal{S}} R^{I}_{CO}(i,j,s,T_S,T_D) = \bigcap_{s\in\mathcal{S}} R^{\rightarrow}_{CO}(i,j,s,T_S) \bigcap_{s\in\mathcal{S}} \tilde{R}^{\leftarrow}_{CO}(j,i,s,T_D) \tag{2.15}$$

2.3.2 Dynamic NAT

Unlike in static NAT, an address x in N_i is dynamically mapped to an address x' in N'_i only when x initiates a connection. Thus, the T_S function in dynamic NAT is not well defined. When a path contains a dynamic NAT transformer, computing the instantaneous reachability of the path is infeasible. Furthermore, the reachability bounds need to be formulated over both routing states and dynamic network address transformation. Therefore, the reachability upper-bound from N_i to N_j in a network with dynamic NAT is defined as the set of packets that can traverse from N_i to N_j under *a* viable routing state and *a* NAT transformation.

The reachability lower-bound from N_i to N_j in a network with dynamic NAT is defined as the set of packets that can traverse from N_i to N_j under *all* routing states and *all* feasible NAT transformations.

Given the routing state s and the transformation function T_S, the instantaneous reachability for connectionless protocols is similar to static NAT and is calculated using formula (2.8). The reachability bounds for connectionless protocols however must include all possible NAT transformations. Thus, using formulas (2.4), (2.5), and (2.8), the reachability bounds for connectionless protocols are calculated as follows:

$$R^U_{CL}(i,j) = \bigcup_{s\in\mathcal{S}} \bigcup_{T_S\in\mathcal{T}} R^I_{CL}(i,j,s,T_S) \tag{2.16}$$

$$R^L_{CL}(i,j) = \bigcap_{s\in\mathcal{S}} \bigcap_{T_S\in\mathcal{T}} R^I_{CL}(i,j,s,T_S) \tag{2.17}$$

where $\mathcal{T}$ denotes all possible transformations on source fields.

For connection-oriented protocols, the instantaneous reachability is calculated similar to static NAT using formula (2.13) for a given s, T_S and T_D. The reachability bounds for connection-oriented protocols are calculated based on formulas (2.6), (2.7), (2.15), and (2.15) as follows:

$$R^U_{CO}(i,j) = \bigcup_{s\in\mathcal{S}} \bigcup_{T_S,T_D\in\mathcal{T}} R^I_{CO}(i,j,s,T_S,T_D) \tag{2.18}$$

$$R^L_{CO}(i,j) = \bigcap_{s\in\mathcal{S}} \bigcap_{T_S,T_D\in\mathcal{T}} R^I_{CO}(i,j,s,T_S,T_D) \tag{2.19}$$

2.4 Reachability Formulation with IP Tunneling

IP tunneling is achieved by encapsulating one IP packet inside another. It is commonly used in enterprise networks. For example, IPv6 packets can be carried in IPv4 tunnels in a network. In practice, IP tunnels are mostly used to send encrypted traffic from a private network to another over public networks. Popular tunneling protocols include Generic Routing Encapsulation (GRE) [FLHDM00], a stateless IP tunneling protocol with transport protocol number 47, and Layer 2 Transport Protocol (L2TP), which operates on UDP protocol on port 1701 [TVR+99].

Next, we formulate network reachability on paths with IP tunnels. Suppose we have an IP tunnel between middleboxes γ and γ' on path $P_{i,j}(s)$. As packets from N_i to N_j are encapsulated in IP packets with fixed source and destination addresses in the path from γ to γ', if the tunnel from γ to γ' can be established, then all packets from N_i to N_j can traverse from γ to γ' inside the tunnel; otherwise, no packets from N_i to N_j can traverse from γ to γ' inside the tunnel. We denote the tunnel reachability from γ to γ' by $\delta(\hbar)$, where $\hbar$ is the packet header for this IP tunnel. Thus, $\delta(\hbar)$ is the complete set of packet Σ if the tunnel can be established and $\delta(\hbar)$ is empty if the tunnel cannot be established.

$$\delta^I(\hbar, s) = \begin{cases} \Sigma & \text{if } \hbar \in \bigcap_{k=2\gamma}^{2\gamma'-1} A(C_k) \\ \emptyset & \text{otherwise} \end{cases} \tag{2.20}$$

Using instantaneous tunnel reachability, the instantaneous reachability for connectionless protocols is calculated using formula (2.1) as follows:

$$R^{I}_{CL}(i,j,s,\hbar) = \left(\bigcap_{k=1}^{2\gamma-1} A_{UDP}(C_k)\right) \cap \delta^{I}(\hbar,s) \cap \left(\bigcap_{k=2\gamma'}^{2M} A_{UDP}(C_k)\right) \tag{2.21}$$

Similar to formulas (2.9) and (2.10), the reachability bounds for connectionless protocols are calculated as follows:

$$R^{U}_{CL}(i,j,\hbar) = \bigcup_{s\in\mathcal{S}} R^{I}_{CL}(i,j,s,\hbar) \tag{2.22}$$

$$R^{L}_{CL}(i,j,\hbar) = \bigcap_{s\in\mathcal{S}} R^{I}_{CL}(i,j,s,\hbar) \tag{2.23}$$

For connection-oriented protocols, suppose the tunnel identified by $\hbar_1$ is between middleboxes γ_1 and γ_1' in the data path and the signaling path tunnel identified by $\hbar_2$ is between middleboxes γ_2 and γ_1 and γ_2'. Note that in IP tunneling the data paths and signaling paths can be decoupled. The instantaneous reachability formulation for data paths $R^{\rightarrow}_{CO}(i,j,s,\hbar_1)$ and signaling paths $R^{\leftarrow}_{CO}(j,i,s,\hbar_2)$ are as follows:

$$R^{\rightarrow}_{CO}(i,j,s,\hbar_1) = \left(\bigcap_{k=1}^{2\gamma_1-1} A_{TCP}(C_k)\right) \cap \delta^{I}(\hbar_1,s) \cap \left(\bigcap_{k=2\gamma_1'}^{2M} A_{TCP}(C_k)\right)$$

$$R^{\leftarrow}_{CO}(j,i,s,\hbar_2) = \left(\bigcap_{k=1}^{2\gamma_2-1} A_{TCP}(C_k)\right) \cap \delta^{I}(\hbar_2,s) \cap \left(\bigcap_{k=2\gamma_2'}^{2M} A_{TCP}(C_k)\right)$$

Using above formulas and formula(2.3), we formulate instantaneous reachability for

connection-oriented protocols as:

$$R^{I}_{CO}(i,j,s,\hbar_1,\hbar_2) = R^{\rightarrow}_{CO}(i,j,s,\hbar_1) \cap \tilde{R}^{\leftarrow}_{CO}(j,i,s,\hbar_2) \tag{2.24}$$

Note that formula (2.24) can be easily generalized to handle the paths that have multiple IP packet tunnels.

Similar to formulas (2.14) and (2.15), the reachability bounds for connection-oriented protocols are formulated as follows:

$$R^{U}_{CO}(i,j,\hbar_1,\hbar_2) = \bigcup_{s\in\mathcal{S}} R^{I}_{CO}(i,j,s,\hbar_1,\hbar_2) = \bigcup_{s\in\mathcal{S}} R^{\rightarrow}_{CO}(i,j,s,\hbar_1) \bigcap \bigcup_{s\in\mathcal{S}} \tilde{R}^{\leftarrow}_{CO}(j,i,s,\hbar_2) \tag{2.25}$$

$$R^{L}_{CO}(i,j,\hbar_1,\hbar_2) = \bigcap_{s\in\mathcal{S}} R^{I}_{CO}(i,j,s,\hbar_1,\hbar_2) = \bigcap_{s\in\mathcal{S}} R^{\rightarrow}_{CO}(i,j,s,\hbar_1) \bigcap_{s\in\mathcal{S}} \tilde{R}^{\leftarrow}_{CO}(j,i,s,\hbar_2) \tag{2.26}$$

2.5 Probabilistic Reachability Computation

In a large-scale network, there could be exponential number of loopless paths between each pair of subnets. However, because of network routing, only a small subset of these paths is actually chosen to forward packets from a source to a destination. Thus, considering all the possible paths to calculate the reachability bounds is unnecessary and can be avoided. As routing protocols mostly choose the shortest path to the destination (*i.e.* the path with minimum sum of link's weights), we define *effective paths*, the subset of paths that are more probable to be chosen between a source and a destination. Note that if an enterprise net-

work uses policy based routing or any other types of routing mechanisms to find the best path between a source and a destination, the effective paths can be computed accordingly. Next, we compute the *effective reachability bounds* using effective paths rather than theoretical reachability bounds where all the possible paths are considered. Figure 2.2 shows the theoretical reachability bounds and effective reachability bounds for a two fields (F_1 and F_2) packet header. It is clear that the effective upper-bound reachability is a subset of theoretical upper-bound reachability and the effective lower-bound reachability is a superset of theoretical lower-bound reachability.

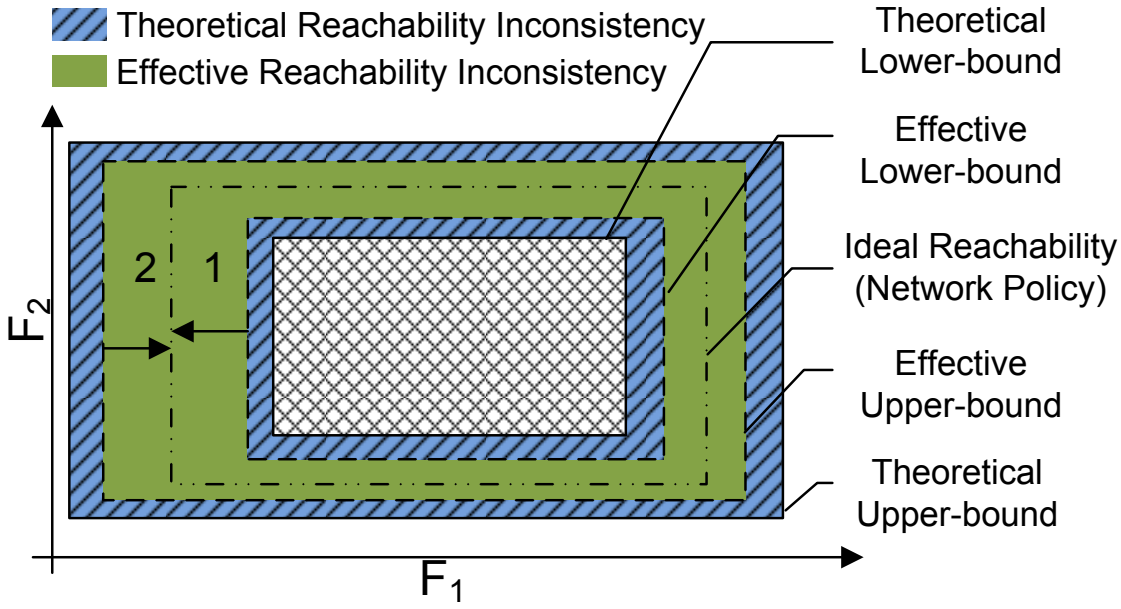


Figure 2.2: Reachability bounds and reachability inconsistencies

To calculate the set of effective paths, we first compute the probability for each path to be chosen by routing daemon as the shortest path (given a weight for each link and the probability that the link is up and working). We then remove the paths that are unlikely be chosen using parameter $\epsilon_{i,j}$, which is the network reachability uptime between N_i and N_j. Network reachability uptime is the percentage of time that the calculated reachability

is applicable. Note that $\epsilon_{i,j}$ is bounded by network uptime for between N_i and N_j, denoted $\Upsilon_{i,j}$ ($\epsilon_{i,j} \leq \Upsilon_{i,j}$), which indicates the percentage of time that subnets N_i and N_j are connected. For instance, $\Upsilon_{i,j} = 99.999\%$ and $\epsilon_{i,j} = 99.99\%$ indicate that N_i and N_j are connected in 99.999% of the time and the calculated reachability between them is applicable in 99.99% of the time.

Given $P_{i,j}$, let L_a be the set of links, denoted by ℓ_b, that construct path $p_a \in P_{i,j}$. Let w_a be the weight for path p_a. Thus, $w_a = \sum_{\ell_b \in L_a} w_{\ell_b}$ where w_{ℓ_b} is the link weight for ℓ_b. Let $\bar{P}_{i,j}$ be the sequence of all the paths between two given subnets sorted ascendingly by their weights as follows:

$$\bar{P}_{i,j} = \{\langle p_1, \ldots, p_a, \ldots, p_b, \ldots, p_{|P_{i,j}|}\rangle | a < b, w_a \leq w_b\}$$

Let π_a denotes the probability that path p_a be used by the network to forward packets from the source subnet to the destination subnet. The effective paths, denoted by $\bar{P}^e_{i,j}$, are the subsequence of the paths whose sum of probabilities to be used is less than $\epsilon_{i,j}$. We formally define $\bar{P}^e_{i,j} = \langle p_1, p_2, \ldots, p_k\rangle$ where

$$\max_k \sum_{a=1}^{k} \pi_a \leq \epsilon_{i,j}$$

Note that the π_as are independent and $\sum_{a=1}^{|P_{i,j}|} \pi_a = \Upsilon_{i,j}$.

As routing protocols choose the shortest path whose links are all up and available, path $p_a \in \bar{P}_{i,j}$ will be chosen if $p_1, \ldots, p_{a-1}$ are down and path p_a is up. Path p_a is up, if and only if all the links in L_a are up and available. Knowing that, to calculate π_a, we use the reliability theory [Ros07] such that we calculate the probability that $p_1, \ldots, p_{a-1}$ are down

and p_a is up. To formulize π_a, let $\mathbf{x} = (x_1, \ldots, x_h)$ be the state vector for all the h links in the network where x_b is an indicator variable and defined as follows:

$$x_b = \begin{cases} 1, & \text{if link } \ell_b \text{ is up} \\ 0, & \text{if link } \ell_b \text{ is down} \end{cases}$$

We define the structure function $\varphi^a(\mathbf{x})$ as follows

$$\varphi^a(\mathbf{x}) = \begin{cases} 1, & \text{if path } p_a \text{ is up} \\ 0, & \text{if path } p_a \text{ is down} \end{cases}$$

Using the definition, the structure function is given by

$$\varphi^a(\mathbf{x}) = \bigwedge_{\ell_b \in p_a} x_b \tag{2.27}$$

We then define a new function $\psi^a(\mathbf{x})$ that indicates if p_a is chosen by the routing daemon. Therefore, it is clear that $\psi^a(\mathbf{x}) = 1$ if last $a-1$ paths are down and path p_a is up. Thus, $\psi^a(\mathbf{x})$ is calculated as follows:

$$\psi^a(\mathbf{x}) := \bigwedge_{k=1}^{a-1} \neg\varphi^k(\mathbf{x}) \wedge \left(\bigwedge_{\ell_b \in p_a} x_b \right) \quad , \quad (a > 1) \tag{2.28}$$

Note that $\varphi^0(\mathbf{x}) = 0$ and $\psi^1(\mathbf{x}) = \varphi^1(\mathbf{x})$.

Using formula (2.28), $\pi_a = Prob\{\psi^a(\mathbf{x}) = 1\}$. Thus, to calculate π_a, we need to find assignments that make $\psi^a(\mathbf{x}) = 1$. Toward this end, we calculate minterms of $\psi^a(\mathbf{x})$ to declare it in disjunctive normal form (DNF). Let π_{ℓ_b} be the probability that link ℓ_b is up and working. $Prob\{\psi^a(\mathbf{x}) = 1\}$ is the sum of the product of the π_{ℓ_b} associated with element

$x_b \in \mathbf{x}$.

Network monitoring devices can measure π_{ℓ_b} using link uptime based on router logs and network events. In our calculation, for sake of simplicity, we assume that the links uptime are independent from each other. However, in practice, when some network device fails more than one link may be down. Hence, we either need to calculate an upper bound for $Prob\{\psi^a(\mathbf{x}) = 1\}$ or calculate $Prob\{\psi^a(\mathbf{x}) = 1\}$ using the probability of π_a retrieved directly from empirical measurement by monitoring devices.

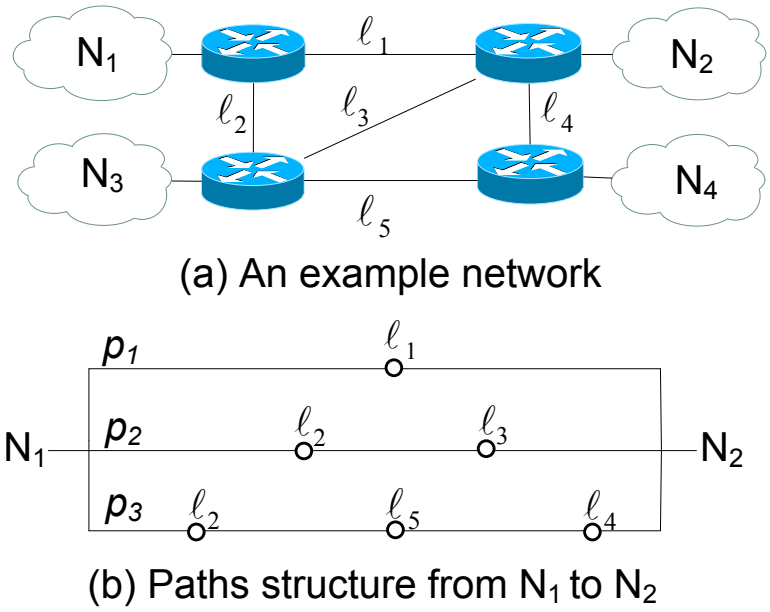


Figure 2.3: Example network

Figure 2.3(a) shows an example network with four routers and four subnets. Figure 2.3(b) illustrates the structure model of paths between N_1 and N_2. Considering each link has a weight equal to 1, The paths are going to be sorted as $\bar{P}_{1,2} = \langle p_1, p_2, p_3 \rangle$ where $L_1 = \{\ell_1\}, L_2 = \{\ell_2, \ell_3\}$ and $L_3 = \{\ell_2, \ell_4, \ell_5\}$.

$\psi^1(\mathbf{x}) := \mathbf{1} \wedge (x_1)$

$\psi^2(\mathbf{x}) := (\neg(x_1)) \wedge (x_2 \wedge x_3)$

$\psi^3(\mathbf{x}) := (\neg(x_1)) \wedge (\neg(x_2 \wedge x_3)) \wedge (x_2 \wedge x_4 \wedge x_5)$

Given $\pi_{\ell_1}, \cdots, \pi_{\ell_5}$ as 0.95, 0.9, 0.99, 0.8, and 0.98, respectively, the minterms for ψ^a and subsequently, π_a are as follows:

$\psi^1(\mathbf{x})$: minterms= `1----` $\Rightarrow \pi_1 = \pi_{\ell_1} = 0.95$

$\psi^2(\mathbf{x})$: minterms= `011--` $\Rightarrow \pi_2 = (1 - \pi_{\ell_1})\pi_{\ell_2}\pi_{\ell_3} = 0.04455$

$\psi^3(\mathbf{x})$: minterms= `01011` $\Rightarrow \pi_3 = (1 - \pi_{\ell_1})\pi_{\ell_2}(1 - \pi_{\ell_3})\pi_{\ell_4}\pi_{\ell_5} = 3.528 \times 10^{-3}$

where `-` denotes "don't care".

In this example, we can calculate $\Upsilon_{1,2} \simeq 99.49\%$. And, supposing $\epsilon_{1,2} = 99.46\%$, the effective paths are $\bar{P}^e_{1,2} = \langle p_1, p_2 \rangle$.

Note that in practice if there are paths with same weights, the routing daemon usually picks one of the paths randomly. In such a case, we have multiple path sequences and the effective paths are the union of effective paths for each path sequence. For instance, if the weights for p_1 and p_2 are the same, there are two possible path sequences, $P^1_{1,2} = \langle p_1, p_2, p_3 \rangle$ and $P^2_{1,2} = \langle p_1, p_3, p_2 \rangle$ and accordingly, $P^{e1}_{1,2} = \langle p_1, p_2 \rangle$ and $P^{e2}_{1,2} = \langle p_1, p_3, p_2 \rangle$, thus, $P^e_{1,2} = P^{e1}_{1,2} \cup P^{e2}_{1,2} = \langle p_1, p_2, p_3 \rangle$.

For the connection-oriented protocols, we have to consider both data paths and signaling paths. Let $\overrightarrow{P}_{i,j}$ and $\overleftarrow{P}_{i,j}$ be the sorted paths sequence for data path and signaling paths. Accordingly, the structure function for path $p_\alpha \in \overrightarrow{P}_{i,j}$ and $p_\beta \in \overleftarrow{P}_{i,j}$, denoted by $\psi^{\alpha,\beta}(\mathbf{x})$, is calculated as follows:

$$\psi^{\alpha,\beta}(\mathbf{x}) = \psi^{\alpha}(\mathbf{x}) \wedge \psi^{\beta}(\mathbf{x})$$

Accordingly, we calculate $\pi_{\alpha,\beta} = Prob\{\psi^{\alpha,\beta}(\mathbf{x}) = 1\}$ same as connectionless protocol. Note that as the link set L_α in data path p_α may not be disjoint with the link set L_β in signaling path p_β, there may not be any assignment for $\psi^{\alpha,\beta}(\mathbf{x}) = 1$. In other words, some of the combination of data paths and signaling paths are not possible in practice. An example

of which can be the case if data path is p_1 and the signaling path is p_2, then $\psi^{1,2}(\mathbf{x}) = (x_1) \wedge ((\neg(x_1)) \wedge (x_2 \wedge x_3))$ has no minterms, which means $\pi_{1,2} = Prob\{\psi^{1,2}(\mathbf{x}) = 1\} = 0$.

By calculating the sequence of effective paths ($P^e_{i,j}$) using network reachability uptime ($\epsilon_{i,j}$) and link weights for paths between N_i and N_j, we can trim the instantaneous reachability matrix from the routing states whose paths are not in $P^e_{i,j}$. This is to avoid unnecessary reachability computations for the paths that are unlikely to be used in the network. As the number of effective paths could be significantly less than the total number of paths between each pair of subnets, we can save considerable amount of time and memory for reachability computation. This computation saving is not only for instantaneous reachability matrices, but also for reachability bounds matrices. Note that by using probabilistic reachability matrices the reachability troubleshooting process is more efficient and practical as fewer paths should be checked and fixed.

Given the sequence of effective paths ($P^e_{i,j}$), the effective reachability bounds ($\hat{R}^U_{CL}(i,j)$) between N_i and N_j using formulas (2.4), (2.5), (2.6), and (2.7) are as follows:

$$\hat{R}^U_{CL}(i,j) = \bigcup_{s \in \mathcal{S}^e} R^I_{CL}(i,j,s) \quad , \quad \hat{R}^L_{CL}(i,j) = \bigcap_{s \in \mathcal{S}^e} R^I_{CL}(i,j,s) \tag{2.29}$$

$$\hat{R}^U_{CO}(i,j) = \bigcup_{s \in \mathcal{S}^e} R^I_{CO}(i,j,s) \quad , \quad \hat{R}^L_{CO}(i,j) = \bigcap_{s \in \mathcal{S}^e} R^I_{CO}(i,j,s) \tag{2.30}$$

where S^e is the set of routing states that associate with effective paths.

Chapter 3

Algorithms for Computing Reachability Matrices

In this section, we present algorithms for computing reachability for networks with no packet transformation filters.

3.1 Reachability Matrices

We represent network reachability as the six matrices shown below. We use n to denote the number of subnets, and z to denote the maximum number of paths between any pair of subnets.

1. $A^I_{CL}[1..n, 1..n, 1..z]$ where each element $A^I_{CL}[i, j, k]$ is the set of packets representing the instantaneous reachability from N_i to N_j on the k-th path for connectionless protocols.

2. $A^I_{CO}[1..n, 1..n, 1..z, 1..z]$ where each element $A^I_{CO}[i, j, k, k']$ is the set of packets representing the instantaneous reachability from N_i to N_j on the k-th data path and the

k'-th signaling path for connection-oriented protocols.

3. $A^L_{CL}[1..n, 1..n]$ where each element $A^L_{CL}[i, j]$ is the set of packets representing the lower-bound reachability from N_i to N_j for connectionless protocols.

4. $A^L_{CO}[1..n, 1..n]$ where each element $A^L_{CO}[i, j]$ is the set of packets representing the lower-bound reachability from N_i to N_j for connection-oriented protocols.

5. $A^U_{CL}[1..n, 1..n]$ where each element $A^U_{CL}[i, j]$ is the set of packets representing the upper-bound reachability from N_i to N_j for connectionless protocols.

6. $A^U_{CO}[1..n, 1..n]$ where each element $A^U_{CO}[i, j]$ is the set of packets representing the upper-bound reachability from N_i to N_j for connection-oriented protocols.

3.2 Basic Data Structures and Algorithms

For any ACL l, we define the *accept set* of l, denoted $accept(l)$, to be the set of packets that can be accepted by l. In this section, we first consider the following core problem in computing network reachability matrices: given two ACLs l_1 and l_2, how can we compute $accept(l_1) \cap accept(l_2)$ and $accept(l_1) \cup accept(l_2)$? Our algorithm for this computation consists of three steps: FDD construction, FDD shaping, and FDD logical operations.

3.2.1 Step I - FDD Construction

In this step, we convert each ACL to an equivalent Firewall Decision Diagram (FDD). FDD was introduced by Gouda and Liu in [GL07] as a data structure for representing access control lists. In nutshell, an FDD is a decision tree where nodes are rule fields (or packet

fields) and edges are labeled by a set of intervals. The leaves of the tree are the actions that have to be taken for the matched packet. As a formal definition, a *firewall decision diagram* with a decision set DS and over fields $F_1, \cdots, F_d$ is an acyclic and directed graph that has the following five properties: A *firewall decision diagram* with a decision set DS and over fields $F_1, \cdots, F_d$ is an acyclic and directed graph that has the following five properties: (1) There is exactly one node that has no incoming edges. This node is called the *root*. The nodes that have no outgoing edges are called *terminal* nodes. (2) Each node v has a label, denoted $F(v)$, such that $F(v) \in \{F_1, \cdots, F_d\}$ if v is a nonterminal node and $F(v) \in DS$ if v is a terminal node. (3) Each edge $e{:}u \rightarrow v$ is labeled with a nonempty set of integers, denoted $I(e)$, where $I(e)$ is a subset of the domain of u's label (*i.e.*, $I(e) \subseteq D(F(u))$). (4) A directed path from the root to a terminal node is called a *decision path*. No two nodes on a decision path have the same label. (5) The set of all outgoing edges of a node v, denoted $E(v)$, satisfies the following two conditions: (i) *Consistency*: $I(e) \cap I(e') = \emptyset$ for any two distinct edges e and e' in $E(v)$. (ii) *Completeness*: $\bigcup_{e \in E(v)} I(e) = D(F(v))$.

decisions are $\{1, 0\}$ where 1 represents *accept* and 0 represents *discard*. We call such FDDs "Binary FDDs". In converting an ACL to an equivalent binary FDD, we replace all flavors of *accept*, such as *accept* and *accept with logging*, by 1, and replace all flavors of *discard*, such as *discard*, *reject*, and *discard/reject with logging*, by 0. We further define a *full-length ordered FDD* as an FDD where in each decision path, all fields appear exactly once and in the same order. For ease of presentation, in the rest of this dissertation, we use the term "FDD" to mean "binary full-length ordered FDD" if not otherwise specified. Figure 3.1(b) shows the two FDDs constructed from the two ACLs in 3.1(a).

An FDD construction algorithm, which converts a sequence of range rules to an equivalent full-length ordered FDD, is described in [LG08]. For computing reachability matrices, we

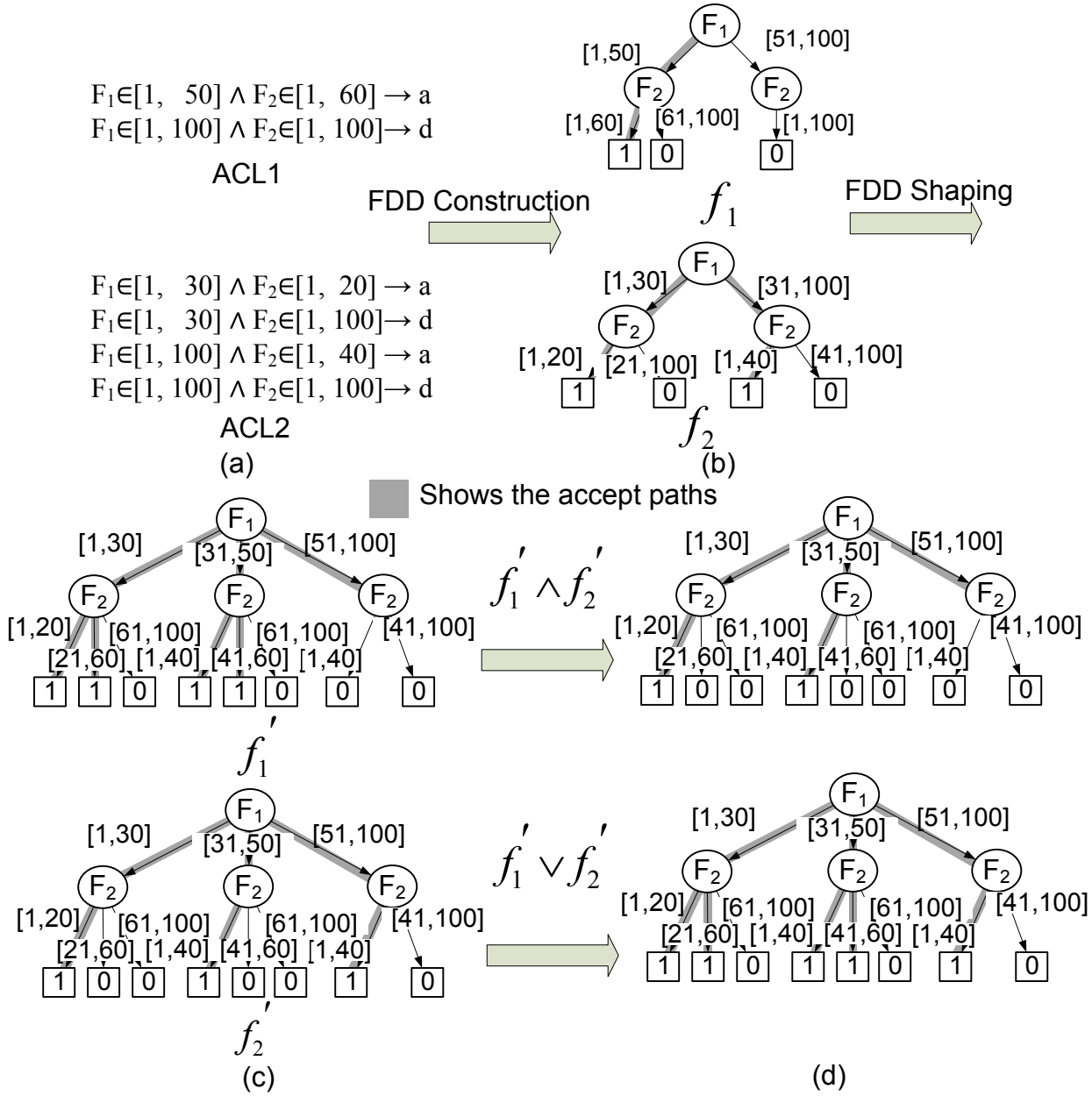


Figure 3.1: (a) Two ACLs (b) Two FDDs before shaping (c) Two FDDs after shaping (d) FDD logical AND/OR

choose the protocol type field as the label of the root node.

We call the decision paths whose terminal nodes are labeled 1 *accept paths*. Similarly, we call the decision paths whose terminal nodes are labeled 0 *discard paths*. Given an ACL l, after we convert it to an equivalent FDD f, the accept paths of f represent the set $accept(l)$.

3.2.2 STEP II - FDD Shaping

In the previous step, we convert the two given ACLs l_1 and l_2 to two FDDs f_1 and f_2 such that l_1 is equivalent to f_1 and l_2 is equivalent to f_2. In this step, we further convert the two FDDs f_1 and f_2 to another two FDDs ${f_1}'$ and ${f_2}'$ such that the following three conditions hold: (1) f_1 is equivalent to ${f_1}'$, (2) f_2 is equivalent to ${f_2}'$, and (3) ${f_1}'$ and ${f_2}'$ are semi-isomorphic. Two FDDs are *semi-isomorphic* if and only if they are exactly the same except the labels of their terminal nodes [LG08]. The algorithm for equivalently converting two FDDs to two semi-isomorphic FDDs is described in [LG08]. Fig. 3.1(c) shows the two semi-isomorphic FDDs converted from the two FDDs in Fig. 3.1(b).

3.2.3 STEP III - FDD Logical AND/OR Operations

In previous steps, we equivalently convert two given ACLs l_1 and l_2 to two semi-isomorphic FDDs ${f_1}'$ and ${f_2}'$. In this step, we compute $accept(l_1) \cap accept(l_2)$ and $accept(l_1) \cup accept(l_2)$ using ${f_1}'$ and ${f_2}'$.

For any two semi-isomorphic FDDs ${f_1}'$ and ${f_2}'$, we define ${f_1}' \wedge {f_2}'$ as a new FDD f such that f is semi-isomorphic to ${f_1}'$ (and ${f_2}'$) and a terminal node in f is labeled 1 if and only if the two corresponding nodes in ${f_1}'$ and ${f_2}'$ are both labeled 1 (otherwise is labeled 0). This implies that the accept paths of ${f_1}' \wedge {f_2}'$ are the intersection of the set of accept paths in ${f_1}'$ and that in ${f_2}'$. Therefore, we can calculate $accept(l_1) \cap accept(l_2)$ by calculating the accept paths in ${f_1}' \wedge {f_2}'$.

Similarly, for any two semi-isomorphic FDDs ${f_1}'$ and ${f_2}'$, we define ${f_1}' \vee {f_2}'$ as a new FDD f such that f is semi-isomorphic to ${f_1}'$ (and ${f_2}'$) and a terminal node in f is labeled 0 if and only if the two corresponding nodes in ${f_1}'$ and ${f_2}'$ are both labeled 0 (otherwise

is labeled 1). This implies that the accept paths of ${f_1}' \vee {f_2}'$ are the union of the set of accept paths in ${f_1}'$ and that in ${f_2}'$. Therefore, we can calculate $accept(l_1) \cup accept(l_2)$ by calculating the accept paths in ${f_1}' \vee {f_2}'$.

Note that after each FDD AND/OR operation, it is important to perform FDD reduction in order to bring down the FDD size expansion caused by the shaping procedure. An FDD is *reduced* if and only if it satisfies the following two conditions: (1) no two nodes are isomorphic; (2) no two nodes have more than one edge between them. Two nodes v and v' in an FDD are *isomorphic* if and only if v and v' satisfy one of the following two conditions: (1) both v and v' are terminal nodes with identical labels; (2) both v and v' are nonterminal nodes and there is a one-to-one correspondence between the outgoing edges of v and the outgoing edges of v' such that every pair of corresponding edges has identical labels and both edges point to the same node. An efficient FDD reduction algorithm that processes the nodes level by level from the terminal nodes to the root node using signatures to speed up comparisons is in [LMTar]. We use an efficient FDD reduction algorithm that processes the nodes level by level from the terminal nodes to the root node using signatures to speed up comparisons. In our experiments, we have found that applying FDD reduction is critical in reducing memory usage for computing network reachability.

3.3 Computing Path and FDD Matrices

We next discuss the computing of two matrices, called path matrix and FDD matrix, which will be used in computing reachability matrices. The path matrix P is an $n \times n$ matrix where each element $P[i, j]$ is the *set* of one-way paths from N_i to N_j. For each path, letting $l_1, \cdots, l_h$ be the ACLs along the path, we compute the FDD that represents $accept(l_1) \cap$

$\cdots \cap accept(l_h)$. The resulting FDDs are stored in the FDD matrix F, which is also an $n \times n$ matrix.

First, we initialize matrices P and F as follows. For any $1 \leq i, j \leq n$, if there is path from N_i to N_j via a middlebox, then $P[i, j]$ consists of this path (which is composed of two links: the link from N_i to the middlebox and the link from the middlebox to N_j) and $F[i, j]$ consists of the FDD that represent the intersection of the accept sets of the two ACLs associated with the two links; otherwise, $P[i, j]$ and $F[i, j]$ are both empty.

Second, we complete matrices P and F based on formulas (3.1) and (3.2) using dynamic programming. We use $P[i, k] \circ P[k, j]$ to denote the set of paths where each path is a concatenation of a path in $P[i, k]$ and a path in $P[k, j]$. Similarly, we use $F[i, k] \wedge F[k, j]$ to denote the set of FDDs where each FDD is the logical AND of an FDD in $F[i, k]$ and an FDD in $F[k, j]$. Note that we remove all paths with cycles because cycles are typically prevented by routing protocols.

$$P[i, j] = \bigcup_{k \in \mathcal{N}} P[i, k] \circ P[k, j] \tag{3.1}$$

$$F[i, j] = \bigcup_{k \in \mathcal{N}} F[i, k] \wedge F[k, j] \tag{3.2}$$

Third, for each FDD in $F[i, j]$, we reduce the domain of the source IP address field to the set of IP addresses used in subnet N_i and the domain of the destination IP address field to the set of IP addresses used in N_j. We use $src(N_i)$ to denote the set of packets whose source IP address is in N_i and $dst(N_j)$ to denote the set of packets whose destination IP address is in N_j. We use $fdd(src(N_i))$ to denote the FDD that represents $src(N_i)$ and $fdd(src(N_j))$ to denote the FDD that represents $dst(N_j)$. Therefore, in this step, we replace each FDD f

in $F[i, j]$ by $fdd(src(N_i)) \wedge f \wedge fdd(dst(N_j)$.

3.4 Computing Reachability Matrices

We are now ready to compute the 6 reachability matrices.

3.4.1 Reachability for Connectionless Protocols

For any $1 \leq k \leq |F[i, j]|$, we use $F[i, j]_k$ to denote the k-th FDD in $F[i, j]$ and $P[i, j]_k$ to denote the k-th path in $P[i, j]$. We use $Sub_{CL}(F[i, j]_k)$ to denote the UDP subtree of FDD $F[i, j]_k$. Recall that in computing reachability we choose protocol type to be the label of the root node. Therefore, the instantaneous reachability of the path $P[i, j]_k$ is:

$$A^I_{CL}[i, j, k] = Sub_{CL}(F[i, j]_k) \tag{3.3}$$

Accordingly, based on formulas (2.4) and (2.5), the reachability upper-bound and lower-bound from N_i to N_j are calculated as follows:

$$A^U_{CL}[i, j] = Sub_{CL}(\bigvee_{k=1}^{|F[i,j]|} F[i, j]_k) \tag{3.4}$$

$$A^L_{CL}[i, j] = Sub_{CL}(\bigwedge_{k=1}^{|F[i,j]|} F[i, j]_k) \tag{3.5}$$

3.4.2 Reachability for Connection-oriented Protocols

We first consider the case that all middleboxes on paths from N_i to N_j are stateful. We use $Sub_{CO}(F[i, j]_k)$ to denote the TCP/ICMP subtree. The instantaneous, upper-bound,

and lower-bound reachability matrices are calculated using formulas (2.2), (2.6) and (2.7), as follows:

$$A^{I}_{CO}[i,j,k] = Sub_{CO}(F[i,j]_k) \tag{3.6}$$

$$A^{U}_{CO}[i,j] = Sub_{CO}(\bigvee_{k=1}^{|F[i,j]|} F[i,j]_k) \tag{3.7}$$

$$A^{L}_{CO}[i,j] = Sub_{CO}(\bigwedge_{k=1}^{|F[i,j]|} F[i,j]_k) \tag{3.8}$$

Second, we consider the case that all middleboxes on paths from N_i to N_j are stateless. As discussed in chapter 2.2.1, for the instantaneous reachability, we need to look at the reachability of each data path $P[i,j]_k$ and the corresponding signaling path $P[j,i]_{k'}$. The swapping operator is implemented by function $Swap_{SD}$. For an FDD f, the function $Swap_{SD}(f)$ basically swaps the labels of source fields and destination fields. The instantaneous, upper-bound, and lower-bound reachability matrices are calculated using formulas (2.3), (2.6), and (2.7), as follows:

$$A^{I}_{CO}[i,j,k,k'] = Sub_{CO}(F[i,j]_k \wedge Swap_{SD}(F[j,i]_{k'})) \tag{3.9}$$

$$A^{U}_{CO}[i,j] = \bigvee_{k=1}^{|F[i,j]|} \bigvee_{k'=1}^{|F[j,i]|} A^{I}_{CO}[i,j,k,k'] \tag{3.10}$$

$$A^{L}_{CO}[i,j] = \bigwedge_{k=1}^{|F[i,j]|} \bigwedge_{k'=1}^{|F[j,i]|} A^{I}_{CO}[i,j,k,k'] \tag{3.11}$$

For the case that the paths from N_i to N_j contain both stateful and stateless middleboxes,

we use the formulas (3.6), (3.7), and (3.8) to handle the stateful middleboxes and formulas (3.9), (3.10), and (3.11) to handle stateless middleboxes.

To compute the reachability for paths with IP tunneling based on formulas (2.21) and (2.24), we need to use an FDD to represent $\delta^I(\hbar, s)$ according to formula (2.20). Thus, we create two FDDs namely "all-accept" FDD and "all-discard" FDD to represent Σ and $\emptyset$, respectively. The "all-accept" and "all-discard" FDDs contains 5 nodes (for 5-tuple packet) and 5 edges that are all labeled by node's complete domain ($D(F(v))$). The decision for 'all-accept" FDD is 1 and for "all-discard" FDD is 0.

3.4.3 Probabilistic Reachability Computation

Given the sequence of effective paths ($P^e_{i,j}$), we first compute probabilistic reachability matrices $\hat{A}^I_{CL}$ and $\hat{A}^I_{CO}$ similar to A^I_{CL} and A^I_{CO} using formulas (3.3) and (3.9) but only for effective paths. We then compute the effective reachability bounds based on formulas (2.29) and (2.30) as follows:

$$\hat{A}^U_{CL}[i,j] = \bigvee_{p \in P^e_{i,j}} \hat{A}^I_{CL}[i,j,p] \quad , \quad \hat{A}^L_{CL}[i,j] = \bigwedge_{p \in P^e_{i,j}} \hat{A}^I_{CL}[i,j,p] \tag{3.12}$$

$$\hat{A}^U_{CO}[i,j] = \bigvee_{p \in P^e_{i,j}} \hat{A}^I_{CO}[i,j,p] \quad , \quad \hat{A}^L_{CO}[i,j] = \bigwedge_{p \in P^e_{i,j}} \hat{A}^I_{CO}[i,j,p] \tag{3.13}$$

Chapter 4

Handling Packet Transformers

In this section, we present algorithms for computing reachability for networks that have packet transformation filters. There are two types of packet transformers, Network Address Translation (NAT) and Port Address Translation (PAT). We use the terms *private subnet* and *private addresses* to refer to respectively, the source subnet and its address pool, which is behind NAT middlebox. Similarly, we use the terms *public subnet* and *public addresses* to refer the virtual subnet and its address pool after NAT transformer, respectively. There are two types of NAT transformation: *Static NAT* and *Dynamic NAT*. A static NAT uses a static one-to-one mapping from private addresses to public addresses. This mapping table is configured by network administrators. A dynamic NAT maps private addresses to public addresses on-the-fly randomly. In Port Address Translation (PAT), all private addresses are mapped to a single public IP address but with different source port numbers, where each port number is designated for a connection initiated from the private subnet. Port address assignment is random.

In presenting our solution for handling packet transformers, we assume that all paths from one subnet to another need to go through the same packet transformer. This assumption

is purely for the ease of presentation, and it imposes no limitation on our solutions for general cases. First, for paths that contain no packet transformers, we can easily compute reachability by combining the algorithms in this section and those in Section 3. Second, our algorithms below can be easily extended to paths that contain more than one packet transformers by dividing each path into multiple chunks where each chunk contains only one packet transformer.

4.1 Reachability for Connectionless Protocols

We discuss reachability computation based on packet transformer types: static NAT, dynamic NAT, and PAT.

4.1.1 Static NAT

To handle static NAT, we only need to change the algorithm for computing an FDD matrix as follows. Given the k-th path from N_i to N_j, which contains a static NAT that maps addresses in N_i to N_i', we call the path from N_i to the NAT middlebox a *private subpath* and the path from the NAT middlebox to N_j a *public subpath*. Let fr and fu be the FDDs that represent the set of packets that can be accepted by the ACLs on the private subpath and those on the public subpath respectively. We still use T_S to denote the packet transformation function of the static NAT. Therefore, $src(N_i) \wedge fr \wedge dst(N_j)$ represents the packets that can traverse on the private subpath with source addresses in N_i, and $src(N_i') \wedge fu \wedge dst(N_j)$ represents the packets that can traverse on the public subpath with source addresses in N_i'. Based on formula (2.8), the set of packets that can traverse from N_i to N_j passing the NAT can be calculated as follows:

$$F[i,j]_k = (src(N_i) \wedge fr \wedge dst(N_j)) \wedge T_S^{-1}(src(N_i') \wedge fu \wedge dst(N_j)) \tag{4.1}$$

After this, reachability matrices can be computed using formulas (3.3), (3.4), and (3.5). Note that the operation T_S^{-1} $(src(N_i') \wedge fu \wedge dst(N_j))$ basically replaces every source IP address x' in the domain of N_i' to x in the domain of N_i for FDD $src(N_i') \wedge fu \wedge dst(N_j)$.

4.1.2 Dynamic NAT

Consider all the paths from N_i to N_j, which all pass through a dynamic NAT. Let FDDs fr_k and fu_k represent all the accepted packets by the k-th private and public subpaths respectively, and let a and b be the total number of private and public subpaths respectively. The upper-bound reachability of the private subpaths and public subpaths are calculated as follows:

$$fr^U = \bigvee_{k=1}^{a} (src(N_i) \wedge fr_k \wedge dst(N_j))$$

$$fu^U = \bigvee_{k=1}^{b} (src(N_i') \wedge fu_k \wedge dst(N_j))$$

Similarly, lower-bound reachability of the private subpaths and public subpaths are calculated as follows:

$$fr^L = \bigwedge_{k=1}^{a} (src(N_i) \wedge fr_k \wedge dst(N_j))$$

$$fu^L = \bigwedge_{k=1}^{b} (src(N_i') \wedge fu_k \wedge dst(N_j))$$

Next, we compute the reachability bounds from N_i to N_j using fr^U, fr^U, fr^L, fu^L. First, we need to reorder the fields of FDDs fu^U and fu^L such that the label of the root is the source IP address. Because a dynamic NAT may map an address in N_i to any address in N_i', to compute the reachability upper-bound from N_i to N_j, we need to disregard the source IP address filed in fu^U by applying the logical OR operation on all the subtrees of the root. Thus, the upper-bound reachability for paths that contains a dynamic NAT transformer is calculated using formula (2.16) as follows:

$$A_{CL}^U[i,j] = Sub_{CL}(fr^U \wedge \bigvee(\text{all subtrees of the root of } fu^U)) \tag{4.2}$$

Similarly, to compute the reachability lower-bound from N_i to N_j, we need to disregard the source IP address filed in fu^L by applying the logical AND operation on all the subtrees of the root. Thus, the lower-bound reachability for paths that contains a dynamic NAT transformer is calculated using formula (2.17) below:

$$A_{CL}^L[i,j] = Sub_{CL}(fr^L \wedge \bigwedge(\text{all subtrees of the root of } fu^L)) \tag{4.3}$$

Note that reordering the packet fields of an FDD f can be done in two steps. First, we generate an equivalent ACL l from f using the algorithms in [GL07]. Second, we construct an equivalent FDD f' from l using the new packet field order.

For simplicity, we assume that a dynamic NAT transformer only changes the source IP address of a packet and does not change the source port field. However, even if it is the case,

we can easily adapt our solution by excluding both fields from the FDDs fu^U and fu^L.

4.1.3 PAT

In port address translation, the public addresses ($N_{i'}$) consists of a single IP address. Similar to dynamic NAT, PAT dynamically assign port numbers to new connections. Therefore, for paths with PAT, the instantaneous reachability cannot be computed because the transformation function is not well-defined. However, we can define and compute reachability bounds for PAT similar to those for dynamic NAT. Formulas (4.2) and (4.3) are still valid except that we reorder the packet fields of the FDDs fu^U and fu^L such that the source port field is the label of the roots.

4.2 Reachability for Connection-oriented Protocols

As we stated previously, for simplicity, we present solutions to the case that all middleboxes on a path are stateful and the case that all middleboxes on a path are stateless. When all middleboxes on the path from N_i to N_j are stateful, for static NAT, we can use the FDD matrix calculated by formula (4.1) and use formulas (3.6), (3.7) and (3.8) to calculate the reachability matrices; for dynamic NAT, we can use formulas (4.2) and (4.3) to calculate the reachability matrices except that the Sub_{CL} function should be replaced by Sub_{CO}. Next, we present solutions for the case that all middleboxes on a path are stateless based on the three types of packet transformers.

4.2.1 Static NAT

Considering the communication between N_i and N_j where all paths between them pass through a static NAT t, let $\overrightarrow{fr}$, $\overrightarrow{fu}$, $\overleftarrow{fu}$, $\overleftarrow{fr}$ be the FDDs that represent the packets that can traverse on the paths from N_i to t, t to N_j, N_j to t, and t to N_i, respectively. According to formula (4.1), the packets that can traverse on the data path is represented by $\overrightarrow{F[i,j]_k} = (src(N_i) \wedge \overrightarrow{fr} \wedge dst(N_j)) \wedge T_S^{-1}(src(N_i') \wedge \overrightarrow{fu} \wedge dst(N_j))$. Similarly, the packets that can traverse on the signaling path are represented by $\overleftarrow{F[j,i]_k} = (src(N_j) \wedge \overleftarrow{fr} \wedge dst(N_i)) \wedge T_D^{-1}(src(N_j) \wedge \overleftarrow{fu} \wedge dst(N_i'))$. Therefore, according to formula (3.9), we can compute the instantaneous reachability between N_i and N_j as formula (4.4). The reachability bounds are calculated according to formulas (3.10) and (3.11).

$$A_{CO}^{I}[i,j,k,k'] = Sub_{CO}(\overrightarrow{F[i,j]_k} \wedge Swap_{SD}(\overleftarrow{F[j,i]_{k'}})) \tag{4.4}$$

4.2.2 Dynamic NAT

As discussed above, we use the new definitions for computing reachability bounds for networks with dynamic NAT. Let a and b be the total number of private and public paths respectively. For the reachability upper-bound of the data path from N_i to N_j, we first compute the reachability upper-bound for the private and public paths as follows:

$$\overrightarrow{fr^U} = \bigvee_{k=1}^{a} (src(N_i) \wedge \overrightarrow{fr_k} \wedge dst(N_j))$$

$$\overrightarrow{fu^U} = \bigvee_{k=1}^{b} (src(N_i') \wedge \overrightarrow{fu_k} \wedge dst(N_j))$$

$$\overleftarrow{fr^U} = \bigvee_{k=1}^{a} (src(N_j) \wedge \overleftarrow{fr_k} \wedge dst(N_i))$$

$$\overleftarrow{fu^U} = \bigvee_{k=1}^{b} (src(N_j) \wedge \overleftarrow{fu_k} \wedge dst(N'_i))$$

Second, we reorder the packet fields for the FDDs $\overrightarrow{fu^U}$ and $\overleftarrow{fu^U}$ such that the root of $\overrightarrow{fu^U}$ is labeled as the source IP address and the root of $\overleftarrow{fu^U}$ is labeled as the destination IP address. Third, we compute the reachability upper-bound as:

$$A^U_{CO}[i,j] = Sub_{CO}(\overrightarrow{fr^U} \wedge \bigvee(\text{all subtrees of the root of } \overrightarrow{fu^U})$$
$$\wedge Swap_{SD}(\overleftarrow{fr^U} \wedge \bigvee(\text{all subtrees of the root of } \overleftarrow{fu^U}))) \quad (4.5)$$

For reachability lower bound, we first compute $\overrightarrow{fr^L}$, $\overrightarrow{fu^L}$, $\overleftarrow{fr^L}$, and $\overleftarrow{fu^L}$ as above by replacing all logical OR with Logical AND. Then, we compute the lower-bound as follows:

$$A^L_{CO}[i,j] = Sub_{CO}(\overrightarrow{fr^L} \wedge \bigwedge(\text{all subtrees of the root of } \overrightarrow{fu^L})$$
$$\wedge Swap_{SD}(\overleftarrow{fr^L} \wedge \bigwedge(\text{all subtreesof the root of } \overleftarrow{fu^L}))) \quad (4.6)$$

4.2.3 PAT

We define and compute reachability bounds for PAT similar to those for dynamic NAT. Formulas (4.5) and (4.6) are still valid except that we reorder the packet fields of the FDDs $\overrightarrow{fu^U}$ and $\overrightarrow{fu^L}$ such that the root is labeled the source port field and reorder the packet fields in $\overleftarrow{fu^U}$ and $\overleftarrow{fu^L}$ such that the root is labeled the destination port field.

Chapter 5

Online Reachability Queries

After reachability matrices are calculated, we can use them as the engine for efficiently processing network reachability queries. In this section, we discuss languages for specifying reachability queries, ways of using such queries for network and security management, and algorithms for processing these queries. Based on the nature of queries, Quarnet supports three types of queries: upper-bound, lower-bound, and instantaneous. Upper-bound/lower-bound reachability queries are useful in verifying whether the ACLs on middleboxes satisfy certain security policies. Instantaneous reachability queries are useful for real-time security monitoring as the network operator identifies which paths are used at the time of querying. Such queries are also useful to verify whether the changes on the ACLs on some middleboxes have undesired impact on reachability. Based on the answer of queries, Quarnet supports two types of queries: *closed* and *open*. A closed query demands an answer of *yes/no*. For instance, considering the network in Figure 2.1, can all hosts in S1 communicate with Mail Server in S4 on TCP port 25 via any path? An open query demands an answer in terms of a *set*. For example, which hosts in S1 can access the Mail Server in S4 on TCP port 25 via any path from S1 to S4? As another example, what set of paths may let all hosts in S1

access the Mail Server in S4 on TCP port 25?

5.1 Reachability Query Language

SRQL Syntax: We define an SQL-like language called *Structured Reachability Query Language* (SRQL) for specifying reachability queries. SRQL has the following format:

```
reachability_type 𝒯
connection_type 𝒪
select ℱ
where (F_1 ∈ S_1) ∧ ··· ∧ (F_d ∈ S_d) ∧ (𝒫 ∈ S_P)
```

The reachability type $\mathcal{T}$ denotes the type of reachability, namely instantaneous (I), upper-bound (U), or lower-bound (L). The connection type $\mathcal{O}$ denotes the connection orientation of transport protocols, namely connection-oriented (CO) or connectionless (CL). When the reachability type $\mathcal{T}$ is upper-bound or lower-bound, the `select` clause $\mathcal{F}$ is a subset of packet fields $\{F_1, F_2, \cdots, F_d\}$; when $\mathcal{T}$ is instantaneous, $\mathcal{F}$ is a subset of fields $\{F_1, F_2, \cdots, F_d, \mathcal{P}\}$ where $\mathcal{P}$ denotes the attribute of "path". In the `where` clause, the predicate $(F_1 \in S_1) \wedge \cdots \wedge (F_d \in S_d)$ specifies the set of packets that this query is concerned with and $(\mathcal{P} \in S_P)$ specifies the set of paths that this query concerns. For example, SRQL query for the question "Through what paths the mail server in S4 on TCP port 25 is accessible from S1?" is the following:

```
type I
protocol CO
select 𝒫
where (S ∈ S1) ∧ (D ∈ MailServer) ∧ (DP ∈ 25) ∧ (PT ∈ TCP)
```

Note that we do not expect network operators to specify queries using SRQL directly. Instead, we expect a full-fledged implementation of Quarnet to provide a GUI interface for inputting queries and specifying paths. The SRQL will be used to formally represent a query under the hood.

SRQL Semantics: The result of an upper-bound reachability query, where $\mathcal{F} = \{\mathcal{F}_1, \cdots, \mathcal{F}_h\}$ and $\mathcal{F}_i \in \{F_1, F_2, \cdots, F_d\}$ for every $1 \leq i \leq h$, is defined as follows:

$\{(\hbar_{\mathcal{F}_1}, \cdots, \hbar_{\mathcal{F}_h}) | (\hbar_1 \in S_1) \wedge \cdots \wedge (\hbar_d \in S_d)$ and packet $(\hbar_1, \cdots, \hbar_d)$ can traverse from its source to its destination at some time$\}$

The result of a lower-bound query is defined similarly except that "at some time" is replaced by "at any time".

The result of an instantaneous reachability query, where $\mathcal{F} = \{\mathcal{F}_1, \cdots, \mathcal{F}_h, \mathcal{P}\}$ and $\mathcal{F}_i \in \{F_1, F_2, \cdots, F_d\}$ for every $1 \leq i \leq h$, is defined as follows:

$\{(\hbar_{\mathcal{F}_1}, \cdots, \hbar_{\mathcal{F}_h}, \pi) | (\hbar_1 \in S_1) \wedge \cdots \wedge (\hbar_d \in S_d)$ and packet $(\hbar_1, \cdots, \hbar_d)$ can traverse from its source to its destination through path π where $\pi \in S_P.\}$

SRQL Example 1: We next give some query examples, where we use the shorthand S for source IP, D for destination IP, SP for source port, DP for destination port, and PT for protocol type. The question "Can all hosts in S1 communicate with the mail server in S4 on TCP port 25?" can be formulated as the following query:

```
type L
protocol CO
select S
where (S ∈ S1) ∧ (D ∈ MailServer) ∧ (DP ∈ 25) ∧ (PT ∈ TCP)
```

If the query result is all the IP addresses in S1, then the answer is "yes"; otherwise the answer is "no".

SRQL Example 2: SRQL query for the question "Through what paths the mail server in S4 on TCP port 25 is accessible from S1?" is the following:

```
type I
protocol CO
select P
where (S ∈ S1) ∧ (D ∈ MailServer) ∧ (DP ∈ 25) ∧ (PT ∈ TCP)
```

SRQL Example 3: The answer to some questions may be the union or intersection of multiple SRQL query results. For example, the answer for the question "Which hosts in S1 can access the Mail Server in S4 on both UDP and TCP port 25 via any path from S1 to S4?" is the intersection of the results of the following two SRQL queries:

```
type L
protocol CO
select S
where (S ∈ S1) ∧ (D ∈ MailServer) ∧ (DP ∈ 25) ∧ (PT ∈ TCP)
```

```
type L
protocol CL
select S
where (S ∈ S1) ∧ (D ∈ MailServer) ∧ (DP ∈ 25) ∧ (PT ∈ UDP)
```

5.2 Reachability Query Engine Construction

Our reachability query engine consists of six FDDs representing the six reachability matrices. We compute the six FDDs as follows. For each of the four upper-bound/lower-bound

reachability matrices, we apply the logical OR operation to all matrix elements, where each element is an FDD representing the reachability between two specific subnets. The resultant FDD over d fields represents the upper-bound/lower-bound reachability between any two subnets. For each of the two instantaneous reachability matrices, we compute the two corresponding FDDs as follows. First, we reduce the two instantaneous reachability matrices to 2-dimensional matrices by combining the FDDs for the various paths from a source to a destination into one FDD. To achieve this, we first add a new node labeled with a new attribute "*path*" to each FDD as the root whose outgoing edge is labeled with path IDs, and then apply the logical OR operation to all FDDs regarding the reachability from one subnet to another. It is trivial to label every path with a unique ID. Second, for each of the two resultant 2-dimensional matrices, we apply the logical OR operation to all elements and get an FDD over d+1 fields. The six FDDs will be used to process SRQL queries.

5.3 Online Reachability Query Processing

Reachability queries can be quickly processed by traversing one of the six FDDs computed above. The algorithm is essentially the same as the one described in [LG09] for querying one firewall policy.

Chapter 6

Reachability Diagnosis

In this section, we first formally define the reachability errors. We then show how we find the faulty ACLs and rules that cause instantaneous reachability errors and reachability inconsistencies.

6.1 Reachability Errors

There are two types of reachability errors. (1) There may be some services, hosts, or subnets that are supposed to be reachable from a set of hosts, but they are not (aka service/host/subnet inaccessibility). Such reachability errors are usually reported by network end-users. (2) There may be some services, hosts, or subnets that are *not* supposed to be reachable from specific hosts/subnets, but they are (aka security breach). Such reachability errors are usually detected by monitoring and surveillance devices such as Intrusion Detection Systems (IDSs).

Reachability errors in this dissertation are formulated in two categories: *instantaneous reachability errors* and *network reachability inconsistencies*.

Instantaneous reachability errors are usually given by customers or network surveillance devices. Such errors correspond to dynamic factors in network reachability such as the path that connects source and destination subnets at the time the errors occurred.

Reachability inconsistencies, on the other hand, are called to reachability difference of the paths that connect two subnets together. In other words, reachability inconsistencies between two subnets is the difference between subnets reachability upper-bound and lower-bound. Hence, effective reachability inconsistencies are the reachability differences between effective reachability bounds. Having the reachability inconsistencies, a network operator not only can proactively detect reachability errors and fix them accordingly but also can come up with the ideal reachability that corresponds to the network central access policy. Using such ideal reachability for two given subnets, we can divide the reachability inconsistencies into two categories (shown in Figure 2.2 by region 1 and 2). The first category, which is the difference of ideal and lower-bound reachability, shows a set of type 1 reachability errors. This set of packets must be accepted in all of the effective paths from a source subnet to a destination subnet. The second category, which is the difference of upper-bound and ideal reachability, shows the set of type 2 reachability errors. This set of packets must be discarded in all of the effective paths from a source subnet to a destination subnet.

6.2 Instantaneous Reachability Diagnosis

To detect instantaneous reachability errors, we need to compute the instantaneous reachability matrix using FDDs with complementary ACL and rule tracking feature (ARTF). An FDD with ARTF represents reachability with additional information such as ACL and rule indices. The ARTF extends the FDD properties such that the label for a terminal node

(under each decision path) also contains *the ACL index, the rule number*, and *rule's corresponding decision* that has the highest priority in the ACL and matches the packets in the

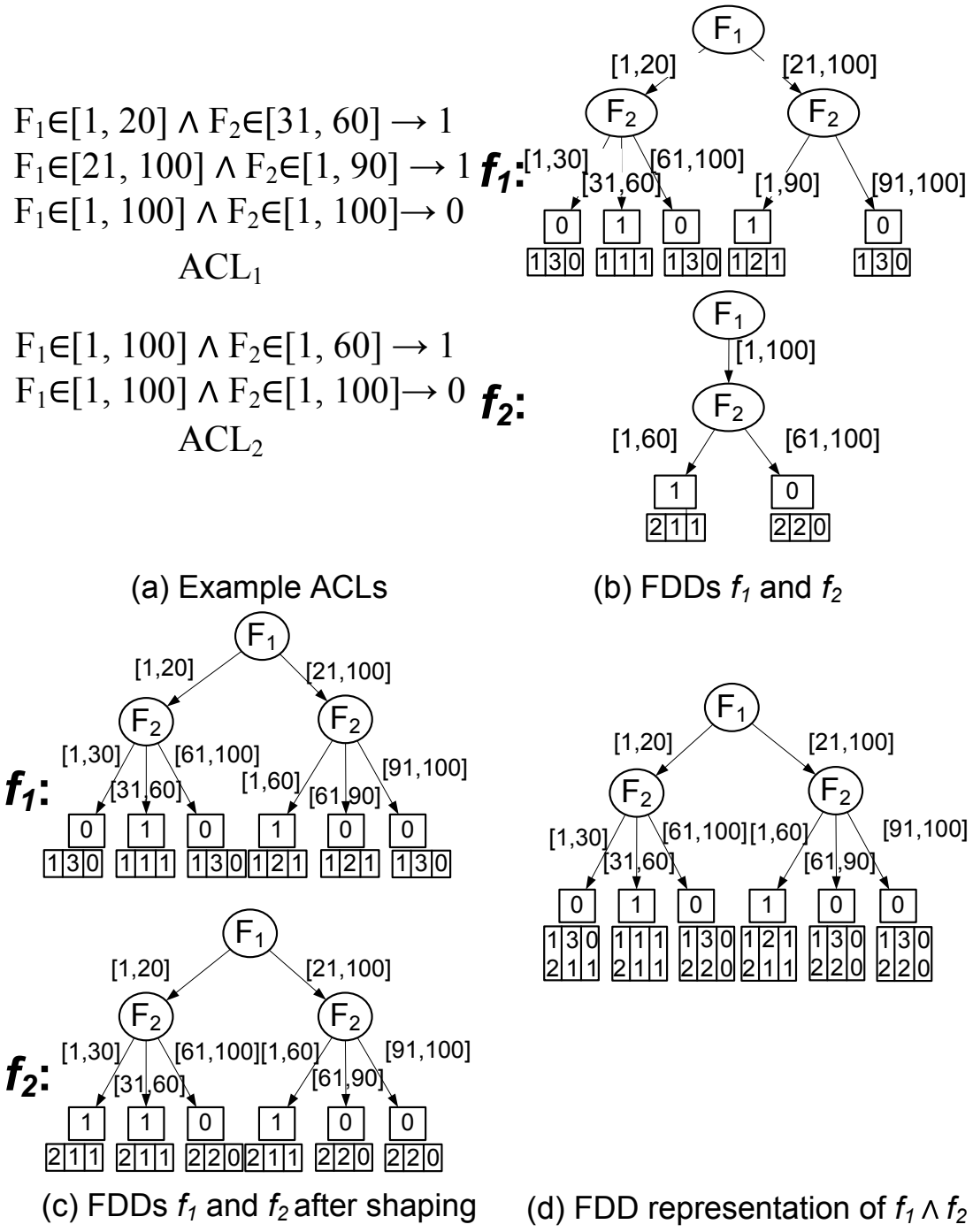


Figure 6.1: FDD shaping and logical AND operation example

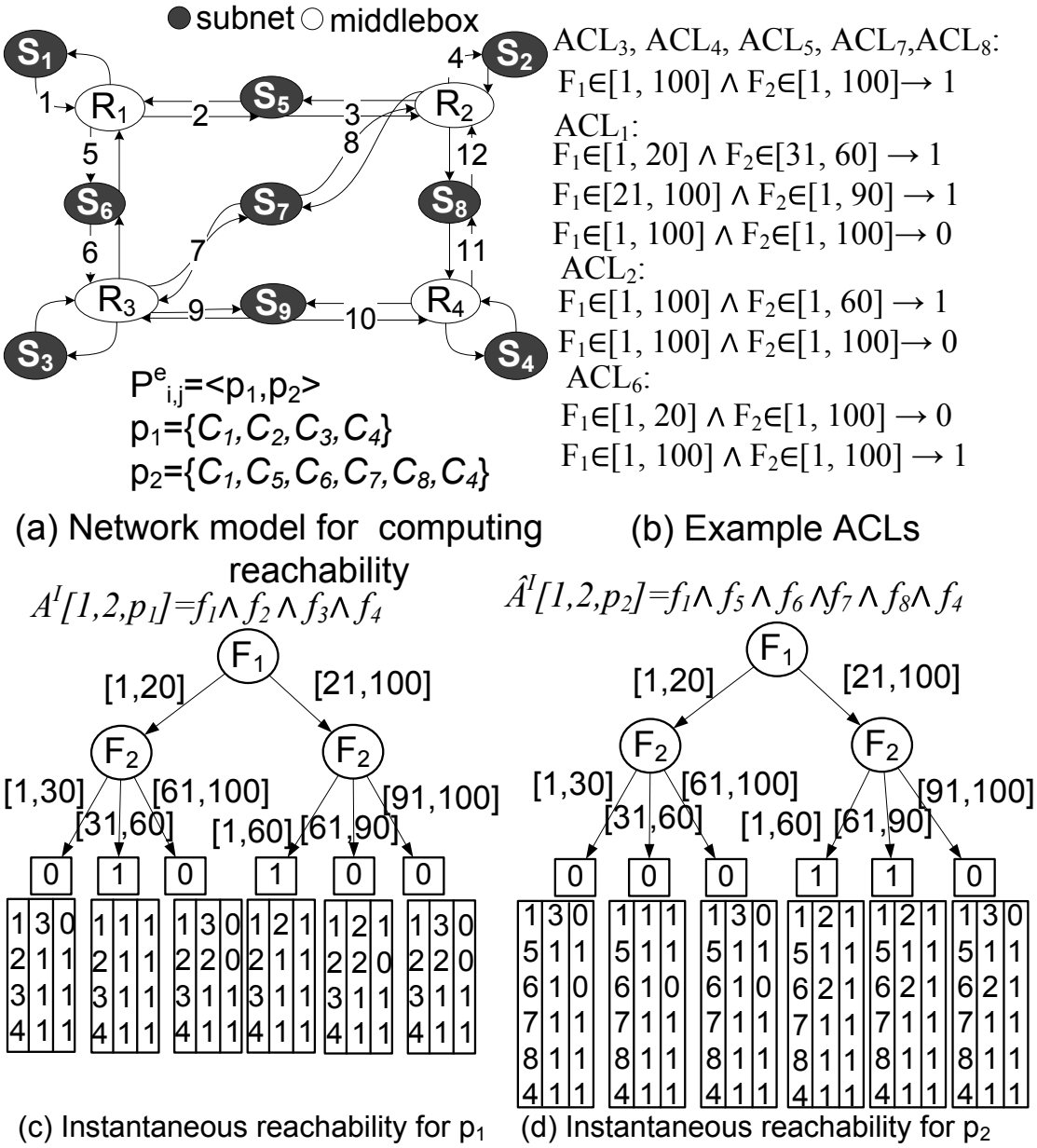


Figure 6.2: Reachability modeling and detecting instantaneous reachability error

decision path.

Figure 6.1 demonstrates the steps of an FDD with ARTF construction. Figure 6.1(a) shows two example ACLs (ACL1 and ACL2) of two packet header fields F_1 and F_2. Figure 6.1(b) shows how these ACLs are converted to their corresponding FDDs with ARTF. Figure 6.1(c) illustrates the FDDs after shaping operation.

In order to enable reachability troubleshooting, we first need to compute a path's instantaneous reachability using formulas (3.3) and (3.9) with FDD with ARTF. We then detect reachability errors as follows.

Let $\mathcal{E}$ be the set of packets whose reachability is wrong (*i.e.* reachability errors). Let $\Phi(x, f)$ denote the set of decision paths ϕ_i in FDD f with decision $x \in DS$. Note that decisions "discard" (*i.e.* drop) and "reject" are signified by 0 and decision "accept" (*i.e.* permit) is signified by 1. Let $\mathcal{E}_1(i, j, p)$ denote type 1 reachability errors (*i.e.* service inaccessibilities) between N_i and N_j using path p. To detect these errors, we need to first identify all the decision paths with decision 0 that intersects with the reachability error set. The set of faulty decision paths, denoted by $\Phi_{\mathcal{E}_1}(i, j, p)$, is computed as follows:

$$\Phi_{\mathcal{E}_1}(i, j, p) = \{\forall \phi_i \in \Phi(0, \hat{A}^I[i, j, p]) | \phi_i \cap \mathcal{E}_1 \neq \emptyset\}$$

Then, all the ACLs with decision 0 under selected decision paths are faulty.

Let $\mathcal{E}_2(i, j, p)$ denote type 2 reachability errors (*i.e.* security breaches) between N_i and N_j using path p. To detect such errors, we need to first identify all the decision paths with decision 1 that intersects with the reachability error set. The set of faulty decision paths, denoted by $\mathcal{E}_2(i, j, p)$, is computed as follows:

$$\Phi_{\mathcal{E}_2}(i, j, p) = \{\forall \phi_i \in \Phi(1, \hat{A}^I[i, j, p])) | \phi_i \cap \mathcal{E}_2 \neq \emptyset\}$$

Then, at least one of the ACLs with decision 1 under selected decision paths ($\Phi_{\mathcal{E}_2}(i, j, p)$) is faulty.

Figure 6.2 shows the instantaneous error diagnosis in the example network shown in

Figure 2.3(a). We first model the example network in Figure 6.2(a). Figure 6.2(b) shows 8 ACLs ($ACL_1, \ldots, ACL_8$), which are associated to the links in the paths between N_1 and N_2. Having $P^e_{1,2} = \langle p_1, p_2 \rangle$, we compute the FDDs that represent the instantaneous reachability for p_1 and p_2 in Figure 6.2(c) and 6.2(d), respectively. For type 1 reachability errors, suppose packets in error set $\mathcal{E}_1 = \{(25, 95)\}$ sent from N_1 cannot be received by N_2 using path p_1. Considering $\hat{A}^I[1, 2, p_1]$ in Figure 6.2(c), decision path ϕ_6 intersects with the error set and its decision is 0. Thus, $\Phi_{\mathcal{E}_2} = \{\phi_6\}$. Among all the ACLs under ϕ_6, the reachability error is caused by rule 3 in ACL_1, because its decision is 0. Similarly for type 2 reachability errors, suppose packets in error set $\mathcal{E}_1 = \{(25, 55)\}$ sent form N_1 to N_2 are considered ineligible packets but they can be received by the destination through path p_2. Considering $\hat{A}^I[1, 2, p_2]$ in Figure 6.2(d), decision path ϕ_4 intersects with the error set and its decision is 1. Thus, $\Phi_{\mathcal{E}_1} = \{\phi_4\}$. At least one ACL under ϕ_4 with decision 1 is faulty.

6.3 Reachability Inconsistencies Diagnosis

To proactively find reachability errors, we need to first detect reachability inconsistencies and then compare them with the network central policy to pinpoint faulty ACLs and fix them accordingly. To identify reachability inconsistencies, we compute the effective reachability bounds matrices using the reachability uptime associated with each pair of subnets based on formula (3.12) and (3.13). Next, we calculate the *effective reachability bounds difference* for packets sent from N_i and received by N_j, denoted by $\Delta\hat{R}$, as follows:

$$\Delta\hat{R}(i, j) = \hat{R}^U(i, j) - \hat{R}^L(i, j)$$

Let $\Delta\hat{A}[i,j]$ represent the FDD for $\Delta\hat{R}(i,j)$. We can compute effective reachability bounds difference as follows:

$$\Delta\hat{A}[i,j] = \hat{A}^U[i,j] \wedge \neg\hat{A}^L[i,j]$$

where $\neg f$ is FDD f where decisions 1 are changed to 0 and vice versa. The decision paths with decision 1 in $\Delta\hat{A}$ show the set of packets that has inconsistent decisions on different paths. Hence, the set of decision paths that causes reachability inconsistencies are $\Phi_{\mathcal{E}}(i,j) = \Phi(1, \Delta\hat{A}[i,j])$.

Figure 6.3(a) and 6.3(b) show the effective reachability upper-bound and lower-bound FDDs with ARTF, respectively. The effective reachability inconsistencies are shown in Figure 6.3(c). As it is shown, the reachability inconsistences are in decision paths ($\phi_2 = F_1 \in [1,20] \wedge F_2 \in [31,60]$) and ($\phi_5 = F_1 \in [21,100] \wedge F_2 \in [61,90]$). However, based on the network access policies these decision paths can be considered as type 1 or type 2 reachability inconsistencies.

To identify type 1 and type 2 reachability errors, let $\mathbb{F}(i,j)$ be the FDD that represents the network central access policy for packets sent by N_i and received by N_j. As in practice the central access policy may not be available in all the enterprise networks, $\mathbb{F}(i,j)$ can be provided by the network operator by going through all the reachability inconsistencies and identifying the correct decision for each inconsistency. Accordingly, based on Figure 2.2, the set of decision paths that identifies type 1 and type 2 reachability errors is computed as $\Phi_{\mathcal{E}_1}(i,j) = \Phi(1, \mathbb{F}(i,j) \wedge \neg\hat{A}^L[i,j])$ and $\Phi_{\mathcal{E}_2}(i,j) = \Phi(1, \hat{A}^U[i,j] \wedge \neg\mathbb{F}(i,j))$, respectively. The network operator can calculate the $\Delta\hat{R}$ when he makes modifications in the ACLs to find inconsistencies and identify reachability errors in the network.

Figure 6.3(c) shows an example for identifying type 1 and type 2 reachability errors.

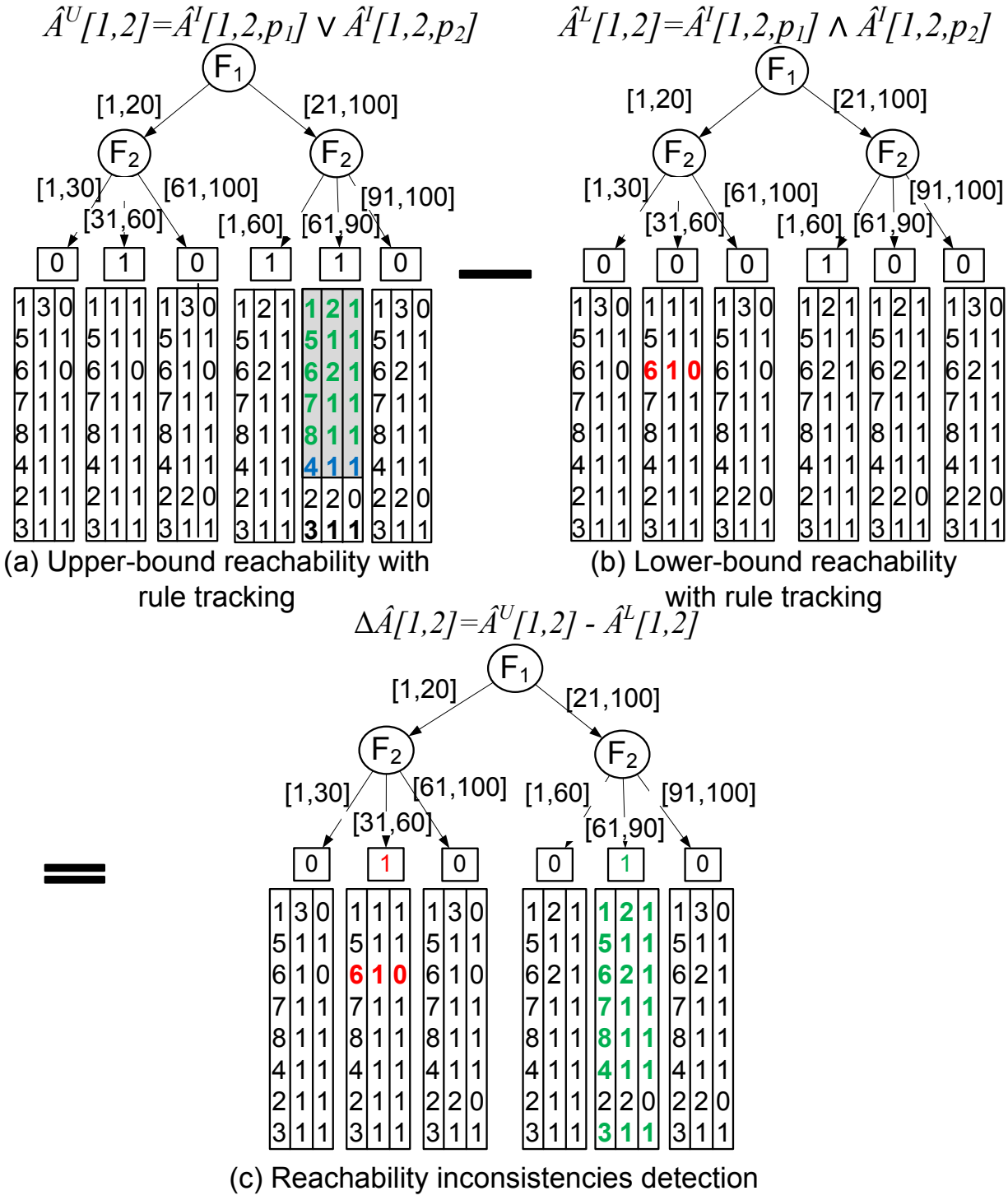


Figure 6.3: Detecting reachability inconsistencies

Suppose the network central access policy accepts packets in ϕ_2 and discards packets in ϕ_5 between N_i and N_j. Thus, the faulty decision paths are $\Phi_{\mathcal{E}_1}(i,j) = \{\phi_2\}$ and $\Phi_{\mathcal{E}_2}(i,j) = \{\phi_5\}$. As highlighted in Figure 6.3, all the ACLs under $\Phi_{\mathcal{E}_1}(i,j)$ with decision 0, and at least one ACL under $\Phi_{\mathcal{E}_2}(i,j)$ with decision 1 are faulty and must be fixed.

Chapter 7

Reachability Troubleshooting

The basic idea to troubleshoot reachability errors automatically is to add one rule on the top of faulty ACLs for each reachability error. This rule, called fixing rule, includes the predicate of the reachability error associated with the correct decision. As ACLs are processed based on the first-match semantic, the fixing rule overlaps with the faulty rule, yet because the fixing rule is prior to the faulty rule in the ACL, it fixes the reachability error. Note that the fixing rule predicate should be chosen carefully to avoid introducing other reachability errors. In the following, we explain troubleshooting algorithms for type 1 and type 2 reachability errors in detail.

7.1 Type 1 reachability errors

To troubleshoot type 1 reachability errors, for each error we need to find a fixing rule and add it to all the faulty ACLs that cause the error. As each reachability error is identified under a decision path, the decision path can be stated as a fixing rule with decision 1 (*i.e.* accept). The rule must be added to *all* the ACLs under that decision path with decision 0.

This solution is common for instantaneous reachability errors and reachability inconsistency errors. For instance, the second decision path in the FDD in Figure 6.2(d) indicates a type 1 reachability error. A fixing rule is then the second decision path ($F_1 \in [1, 20] \wedge F_2 \in [31, 60] \rightarrow 1$) that must be added to faulty ACL_6. The corrected ACL with its corresponding FDD with ARTF is shown in Figure 7.1. As the fixing rule is added to the beginning of the ACL, the rest of the FDD remains intact. This indicates that the fixing rule does not have any impact on the rest of the reachability. Note that the new FDD is the same as the old FDD, except that the decision for the faulty decision path is changed to 1. Also, to avoid recalculating the FDD with ARTF and keeping the tracking records consistent, we first remove the faulty ACL record under the faulty decision path and add a new record with faulty ACL index, rule index and decision equals to 1. Next, we increment the rule index of the faulty ACL tracking records of all other decision paths in the FDD. This algorithm is repeated for all the type 1 reachability errors individually until all of them are resolved. Note that after the error troubleshooting we can use ACL compression techniques [LTM08]to reduce the number of the rules

7.2 Type 2 reachability errors

To troubleshoot type 2 reachability errors, for each error we need to find a fixing rule and add it to at least one of the faulty ACLs that causes the error. As each reachability error is identified by a decision path, the decision path that includes the reachability error can be stated as a fixing rule with decision 0 (*i.e.* discard).

Troubleshooting type 2 reachability errors are different for instantaneous reachability errors and reachability inconsistency errors. For instantaneous reachability errors, the rule

must be added to at least one of the faulty ACLs under that decision path because if one of the ACLs along the path discards the packet, it will not reach the destination. For instance, the fifth decision path in the FDD in Figure 6.2(d) and 7.1(a) indicate a type 2 reachability error and faulty ACLs. A fixing rule is then the fifth decision path ($F_1 \in [21, 100] \wedge F_2 \in [61, 90] \rightarrow 0$) and it must be added to at least one of the faulty ACLs $ACL_1, ACL_4..ACL_8$. However, for reachability inconsistency errors, as paths mingle, we can have a set of fixing rules to fix an error. In the following we explain the troubleshooting algorithms separately for instantaneous reachability errors and reachability inconsistency errors.

7.2.1 Instantaneous reachability errors

To troubleshoot the instantaneous reachability errors, on one hand, we want to add the rule to the first faulty ACL in a given path to minimize the number of unwanted packet forwardings, which saves bandwidth for links along the rest of the path. On the other hand, we also want to add the rule to the ACLs that have fewer rules, as greater number of the rules degrades the router or firewall performance and throughput especially if they exceeds some thresholds [rul]. To address this tradeoff, we introduce a performance cost function C for fixing each faulty ACL, and we add the fixing rule to the ACL with the lowest fixing cost. Let $C(p, k)$ be the fixing cost of ACL_k in path p, $d(k, p)$ be the number of hops to reach ACL_k in path p, $|p|$ be the length of the path, $|ACL_k|$ be the number of rules for ACL_k, and $\mathcal{R}$ be the maximum number of rules an ACL can have. We use two weight coefficients w_d and w_r for number of unwanted packet forwardings and number of the rules, respectively. We then define $\mathbb{C}(k, p)$ as the normalized weighted average of the number of unwanted packet forwardings and the number of rules as follows:

$$\mathbb{C}(k,p) = w_d \frac{d(k,p)}{|p|} + w_r \frac{|ACL_k|}{\mathcal{R}} \tag{7.1}$$

For example, as shown in Figure 7.1, to fix the fifth decision path, we calculate the fixing cost for all the faulty ACLs. Suppose we can have only 10 rules on each ACL ($\mathcal{R} = 10$), and we have same weight for the number of unwanted packet forwardings and number of the rules ($w_d = w_r = 0.5$). The fixing costs for $ACL_1, ACL_4..ACL_8$ are in turn 0.15, 0.383, 0.05, 0.266, 0.216, 0.383. Hence, ACL_5 with the lowest fixing cost is chosen to be fixed and accommodate the fixing rule.

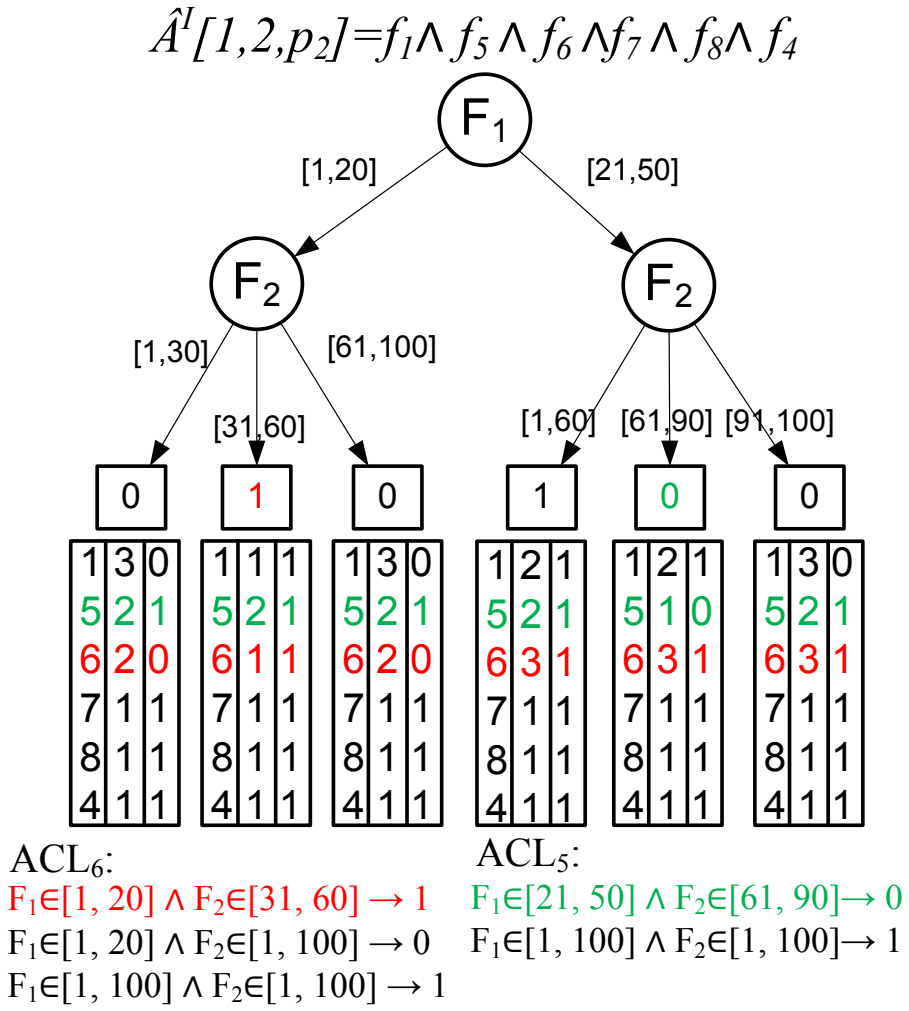


Figure 7.1: Troubleshooting reachability errors

7.2.2 Reachability inconsistency error

For each reachability error, we need to place fixing rules in the faulty ACLs of faulty paths that allow illegitimate traffic to reach the destination. The steps to efficiently resolve each reachability error are as follows. (1) We find all the faulty paths between a pair of subnets. (2) As faulty paths mingle, we find all of the possible minimal faulty ACL sets that can fix all faulty paths (*i.e.* we find a minimum edge cut for all faulty paths). (3) We calculate the total fixing cost for each set of faulty ACLs. (4) We choose the set with the minimum total fixing cost to fix and resolve the error.

The detailed algorithm for each step is as follows. For step (1), we find all the paths between a pair of subnets whose ACLs accept the illegitimate traffic identified by the reachability error. To this end, we single out the paths whose ACLs are all found under the decision paths associated with the error.

For step (2), let $P_{i,j}^{F}$ be the set of faulty paths between N_i and N_j. For a given set of unwanted packets that cause the reachability error, let $\mathbf{y} = (y_1, \ldots, y_K)$ be the state vector for all the K ACLs in the network where y_k is an indicator variable and defined as follows:

$$y_k = \begin{cases} 1, & \text{if } ACL_k \text{ accpets the set of unwanted packets} \\ 0, & \text{if } ACL_k \text{ discards the set of unwanted packets} \end{cases}$$

We define the structure function $\lambda^a(\mathbf{y})$ that indicates if path p_a is faulty or not.

$$\lambda^a(\mathbf{y}) = \begin{cases} 1, & \text{if path } p_a \text{ accepts the set of unwanted packets} \\ 0, & \text{if path } p_a \text{ discards the set of unwanted packets} \end{cases}$$

Using the definition, the structure function is given by

$$\lambda^a(\mathbf{y}) = \bigwedge_{ACL_k \in pa} y_k \tag{7.2}$$

Next, we define a new function $\gamma(\mathbf{y})$, which indicates the set of minimal faulty ACLs that cannot reach the destination. Hence, $\gamma(\mathbf{y})$ can be formulated as follows:

$$\gamma(\mathbf{y}) := \neg \left(\bigvee_{pa \in P^F_{i,j}} \lambda^a(\mathbf{y}) \right) \tag{7.3}$$

The set of minterms for $\gamma(\mathbf{y}) = 1$ is the set of all of the possible minimal faulty ACL sets that must be fixed to resolve the reachability error.

For step (3), as paths mingle and their ACLs can be jointly used in a set of faulty paths, on one hand, we intend to place the fixing rule close to the source subnet, especially on the paths that are more likely to be chosen by routing daemon. On the other hand, we want to avoid adding rules to the faulty ACLs that have large number of rules. To address this tradeoff, the fixing cost for adding the fixing rule depends on the number of the rules that the ACL has, the distance of the ACL to the source subnet in the set of faulty paths, and the probability of faulty paths to be chosen. Thus, similar to formula (7.1) the fixing cost of ACL_k in a set of faulty paths between N_i and N_j, denoted by $P^F_{i,j}$, is calculated as follows:

$$\mathbb{C}(k, P^F_{i,j}) = w_d \sum_{pa \in P^F_{i,j}} \pi_a \frac{d(k, pa)}{|pa|} + w_r \frac{|ACL_k|}{\mathcal{R}} \tag{7.4}$$

Note that if ACL_k is not in the path p, then $d(k, p) = 0$. Let A_g be the g-th minimal faulty

ACL set. Using formula (7.4), the cost for all the faulty ACLs in A_g is calculated as follows:

$$\mathbb{C}(A_g, P^F_{i,j}) = \sum_{ACL_k \in A_g} \mathbb{C}(k, P^F_{i,j}) \tag{7.5}$$

We calculate this cost for all possible values of A_g between N_i and N_j. We then add the fixing rule to the set of faulty ACLs that has the lowest fixing cost. Following the example for reachability inconsistencies in Figure 6.3(c), to troubleshoot the type 2 reachability error on fifth decision path (ϕ_5), we first identify the faulty paths. As second rule in ACL_2 discards the unwanted packets in ϕ_5, the only faulty path is p_2 (*i.e.* $P^F_{1,2} = \{p_2\}$). Thus, the problem is reduced to instantaneous reachability error where adding rule to ACL_5 has the lowest fixing cost.

Chapter 8

Reachability Optimization

One of the very basic challenges that network operators often face when they design a large-scale enterprise network is how to design ACLs and rules and place them on the network classifiers so that they satisfy network access control requirements. This task is usually done manually and it is very tiring, complex, and error-prone. Moreover, manual ACL design is often not optimized and can deteriorate the network performance. In this chapter, we present our approach of optimal ACL design and rule placement on packet classifiers in respect to network reachability correctness and network performance criteria. Given a network central access policy, which indicates the reachability between each pair of subnets in the network, the ACL design can be automated such that it satisfies the central policy and it satisfies the network and its classifiers' performance criteria. Therefore, we define two main ACL design criteria: (1) correctness criterion, in which we ensure that the ACL design enforces the central access policy correctly and (2) performance criterion, in which we ensure that the ACL design is optimal for different network and routers performance criteria.

In the following, we first explain the design criteria in more details. We then shed light on our approach namely, rule placement and the techniques that we use to satisfy the design

criteria. We compare our approach with prior art and we explain pros and cons.

8.1 Problem Formal Definition

8.1.1 Input and Output

The main two input elements are a network topology and the network central access policy. Based on our network reachability model, the network topology is identified using the network incident matrix ($\mathcal{A}$) described in chapter 2. The network central access policy identified as a reachability matrix called central policy reachability denoted $\mathbb{R}$, where each element $\mathbb{R}(i, j)$ describes the set of reachable packets from N_i to N_j. A network central access policy is often extracted from the enterprise security and access protocols and policies. However in practice, the network central access policy may not be available at the time of the design as it can be built up upon requests and emerging new services. In such cases, the input is the ACLs deployed on the network classifiers from which we can compute the network reachability bounds and use them as the network central access policy. Of coarse, if there are reachability inconsistencies as the result of differences between network reachability lower-bound and upper-bound, they should be resolved to have a solid network central access policy.

There are also some complementary parameters that can be identified by the operator for better adaptation of the output results to actual limitations in the network. For instance, some middleboxes may not have packet classifiers to be used for ACLs. On the other hand, some middleboxes such as firewalls can accept a large number of rules, which are preferred by operators to accommodate most of the network access policies. Other network parameters

such as network link-up probability, stateful routers, etc, can also be useful for better and more efficient ACL design.

The output is a set of ACLs for all classifiers across the network. Note that based on some of the design parameters or the position of the classifier in the network, some classifiers may have no rules.

8.1.2 Optimization Criteria

8.1.2.1 Correctness

Given M routers in a path when the network is in routing state $s \in \mathcal{S}$, the correctness criterion is formally stated as follows:

$$\{\forall s \in S : R^I(i, j, s) = \mathbb{R}(i, j)\} \tag{8.1}$$

which indicates that the reachability of each path between a given pair of subnets are the same as the central reachability for that pair of subnets. In case of effective reachability optimization, the routing state $\mathcal{S}$ in formula (8.1) is changed to $\mathcal{S}^e$.

8.1.2.2 Performance

In terms of network performance, unwanted packets need to be discarded as early as possible in the path to the destination. Moreover, we need to minimize the packet processing time for each classifier in the path. To minimize the packet processing time we need to follow different strategies for hardware and software middleboxes. The hardware middleboxes (especially the recent ones) such as hardware firewalls and routers usually use Ternary Content Access Memory (TCAMs), which has constant classification delay, whereas software based

middleboxes such as Linux routers with IPTables [And06] traverse the ACL to find the first match for the packet. Thus, the average classification delay is different based on the size of the ACL. Furthermore, the hardware middleboxes has limited capacity to accept rules, whereas the software middleboxes usually do not have strict capacity limits. Note that because the TCAMs are power intensive, it is advised to either use all of their capacity, or turn them off by assigning no rules to them.

Considering such limitations, the performance criteria is stated in three properties. Given a network with h links, where each link k has an ACL_k, we can formally define these properties as follows:

(1) Minimizing the total number rules ($\min \sum_{k=1}^{h} |ACL_k|$)

(2) Minimizing the average number of forwardings for unwanted packets

(3.1) For software middleboxes: Minimizing the maximum number of the rules per ACL ($\min \max |ACL_k|$)

(3.2) For hardware middleboxes: Minimizing the number of the middleboxes that have at least one rule ($\max_k \{|ACL_k| == 0\}$)

8.2 Our Approach

8.2.1 Rule Placement vs. ACL Placement

There are two main approaches to solve this problem: (1) ACL placement and (2) rule placement.

ACL placement is introduced by Sung *et al.* in [SRXM08]. The basic idea in this approach is deploy the central access policy for a given pair of subnets on all the paths that connects

these two subnets together. They have reduced the bin packing problem to this problem with ACL placement approach and show that using this approach the problem is NP-hard. They later explain heuristics based on "first fit decreasing" strategy to distributed ACLs across all middleboxes in the network.

The advantages of ACL placement are the fact that it is relatively fast and straightforward. All the rules in the central policy for a given source A and destination B are defined to be restricted to subnet A and B, so that the access policies of different pairs of subnets never overlap. As the access policies are terminated by a rule that discards all packets with source addresses in A and destination addresses in B (*i.e.* discard default policy), they can be placed only on one middlebox in a path which in turn helps to have minimum number of rules in the ACLs. However, this approach has a few disadvantages. First, because the ACL obtained by the central access policy may have redundant rules, the outcome of stacking a few ACLs together in a new ACL may not result in an ACL with minimum number of the rules. Second, the proposed heuristics are satisfying one performance criteria at a time and use others as tie breakers. Thus, finding a heuristics to address all of the performance criteria seems not to be trivial.

In this dissertation, we proposed to solve this problem following rule placement approach rather than ACL placement approach. In rule placement approach, the access policies are first converted to a set of non-overlapping discard rules. These rules are then placed on the network middleboxes optimally using a cost function that creates a weighted balance between all performance criteria. Using this approach all rules that belong to the central access policy for a given source and destination can be individually distributed on all middleboxes along the paths that connect the source and the destination.

8.2.2 Rule Placement Algorithm

To explain the rule placement algorithm, we first explain each step of the algorithm. We then complete our explanation with running the algorithm on an example network with a central policy.

In this algorithm, we first convert the central access policy to non-overlapping discard rules. Second, we place rules on the classifiers using a cost function. Third, we use the topological information to aggregate and simplify the places rules. Forth, we exactly tune our rule placement by moving rules from a classifier to other classifiers. Finally, we use compression techniques to finalize the rule placement.

8.2.2.1 Central Policy Conversion to Non-overlapping Discard Rules

As the first step, we ensure that the access policy for a given pair of subnets is only limited to that pair of subnets and does not have any overlap with subnets. To this end, we make sure that the source address range (N_i) and destination address range (N_j) in all rules of access policy ($\mathbb{R}_{i,j}$) is limited to its associated source and destination subnet address ranges. We also needs to make sure that each access policy for a given pair of subnets is complete. This means that the access policies should cover the entire address space that belongs to the source and destination. Otherwise, the incompleteness of access policies can potentially causes reachability inconsistencies and errors. Second, we put all these access policies together to create the central access policy that includes all possible packets in the network. Third, we add a default policy rule with `accept` decision to the end of the central policy rule list to be able to construct an FDD for that. Forth, we construct an FDD for the cental access policy. Using the FDD, we can convert the central access policy to a set of

non-overlapping discard rules. In addition, converting cental access policy to an FDD helps to remove all types of rule shadowing and conflicts, which in turn helps the network operator to review the central access policy for possible errors and mistakes.

8.2.2.2 Cost-based Rule Placement

In order to meet the correctness and performance criteria, we need to place rules associated with a source and a destination on *at least one* classifier on all the paths between them in an efficient way. One classifier is enough because they are discarding rules and we only place a rule on one classifier in a path to minimize the number of total rules on the classifiers. As we assign a cost for adding a rule to each classifier, we want to find the set of classifiers that has the least cost to accommodate a rule. This problem is similar to find the minimum edge cut of the network graph, where the weights for the edges are the cost for adding non-overlapping discard rules.

There are many algorithms to find the minimum edge cuts of a graph. But in terms of implementation, we found it easier to use Quarnet libraries to compute the minimum edge cut as follows. As the first step, we group the non-overlapping rules for each pair of source and destination subnet. Second, for each pair of source and destination subnets, we calculate all possible edge cuts, as each classifier is assigned to an edge in our network model. The edge cuts are calculated similar to formula (7.3); however, this time we consider all the effective paths rather than faulty paths. The modified version of the formula (7.3) is as follows:

$$\gamma(\mathbf{y}) := \neg \left(\bigvee_{pa \in P^{e}_{i,j}} \lambda^{a}(\mathbf{y}) \right) \tag{8.2}$$

The set of minterms for $\gamma(\mathbf{y}) = 1$ is the set of all of the possible edge cuts. Note that the

edge cuts that includes the classifiers that are exempted from accommodating rules by the network operator are ignored. Third, we calculate a cost based on the performance criteria for each edge cut. The theory and the basic idea behind designing a cost function here is similar to ones mentioned for formula (7.4) and Formula (7.5). However, we modify these formulas to incorporate both software and hardware middleboxes. Let q_i be an indicator of the type of the middlebox. Hence, if C_k is hardware then $q_k = 0$, and if C_k is software then $q_k = 1$. Thus, similar to formula (7.4) and formula (7.5) the rule placement cost of ACL_k in a set of effective paths between N_i and N_j, denoted by $P^e_{i,j}$, is calculated as follows:

$$\mathbb{C}(k, P^e_{i,j}) = w_d \sum_{pa \in P^e_{i,j}} \pi_a \frac{d(k, pa)}{|pa|} + w_r \left(q_k \frac{|C_k|}{\mathcal{R}} + (1 - q_k) \cdot \Omega_k \right) \tag{8.3}$$

where Ω_k is the hardware cost function indicator for classifier C_k and it is defined as follows:

$$\Omega_k = \begin{cases} \omega, & \text{if } C_k \text{ has no rules} \\ 0, & \text{if the number of rules in } C_k \text{ does not exceed the classifier capacity} \\ 1, & \text{if the number of rules in } C_k \text{ exceeds the classifier capacity} \end{cases}$$

This function returns the cost of 0, if the classifier has at least one rule. It return 1, if the classifier cannot accept any more rules. It returns ω ($\in [0, 1]$), if the classifier has no rules. Note that ω should be chosen relatively based on w_d and w_r by the network operator considering the middlebox specifications and desired performance criteria.

Let A_g be the g-th edge cut. Using formula (8.3), the cost for all the classifiers in A_g is calculated as follows:

$$\mathbb{C}(A_g, P^e_{i,j}) = \sum_{C_k \in A_g} \mathbb{C}(k, P^e_{i,j}) \tag{8.4}$$

Once the placement cost for all edge cuts in A_g is calculated, we place rules on the classifiers in the edge cut with *minimum cost*. As the rules are non-overlapping, the order of the rule in the ACL is not important, thus, it can be inserted anywhere in the ACL. Finally, the rulesets on the classifiers are terminated by accept-default-policy rule.

Note that the cost calculation procedure iterates for each non-overlapping discard rule on paths between all subnets that intersects with the rule's source address and destination address range. However, if the network operator collects information regarding the percentage of traffic that matches against each rule, we can sort non-overlapping rules descending such that the rules with high traffic percentage be placed earlier than rules with low traffic percentage. This is because of the fact that, for the very first rules, the cost of rule placement is mainly determined by their distance to the source. Thus, it is advised to have rules with high traffic percentage to be placed earlier than others.

8.2.2.3 Topological Rule Aggregation

The goal in topological rule aggregation (TRA) is to use topological information for generalizing and aggregating non-overlapping rules. The topological information includes the range of the addresses for subnets whose traffic is routed through network classifier. Using this information the source and destination fields of a rule can be generalized to the domain size, which can be later helpful for rule aggregation. In rule aggregation, if there are similar rules with different source address fields on a classifier, but their source address ranges includes all range of addresses for subnets that sends traffic to the classifier, all these rules can be

aggregated with source field labeled as the domain range. Similarly, if there are similar rules with different destination address fields on a classifier, but their destination address ranges includes all range of addresses for subnets that receives traffic from the classifier, all these rules can be aggregated with destination field labeled as the domain range. For instance, if classifier C_1 only forwards traffic from networks N_1 and N_2, if there are two similar rules where one rule's source address range includes N_1 and the other rule's source address range includes N_2, they can be aggregated with source address `all` or `*`.

In topological rule aggregation, given all effective paths, for each classifier, we create (1) a list of all subnets that can send traffic trough the classifier denoted by L_S and (2) a list of all subnets that can receive traffic for the classifier denoted by L_D. Second, we create an FDD for each classifier using the inserted non-overlapping discard rules. Third, for each FDD, we go through the labels of the all outgoing edges of the source address nodes and relabel them with the domain range if they includes L_S. Forth, similarly for each FDD, we go through the labels of the all outgoing edges of the destination address nodes and relabel them with the domain range if they includes L_D. Finally, we extract the discard rules again from each FDD and terminate them with the accept-default-policy rule.

Note that the topological rule aggregation can be done after the central policy FDD creation, where L_D and L_S contains all the subnets in the network.

8.2.2.4 Rule Expulsion

Rule expulsion is a complementary feature than enables operators to move rules along the paths. Rule expulsion, in practice, can be manually done by the operator or automatically done by another algorithm to meet some custom performance criteria that each network may have based on its design, or some specific services they have. For example, if the number

of rules on a classifier is higher than the classifier's capacity, the rules can be push around to other classifiers along *all* the path that the home classifier is a part of. We believe this functionality is necessary in practice as there are always custom preferences that must be taken into account.

The rule expulsion needs to be done correctly while the correctness criterion holds. Hence, if a rule in one classifier in edge cut A_1 is decided to be expelled to another classifier in edge cut A_2, all the copies of the rule on the classifiers in A_1 are expelled to the classifiers in A_2. Note that the rules that are relabeled in TRA algorithm, may not be expelled as they are associated with classifier position in the network.

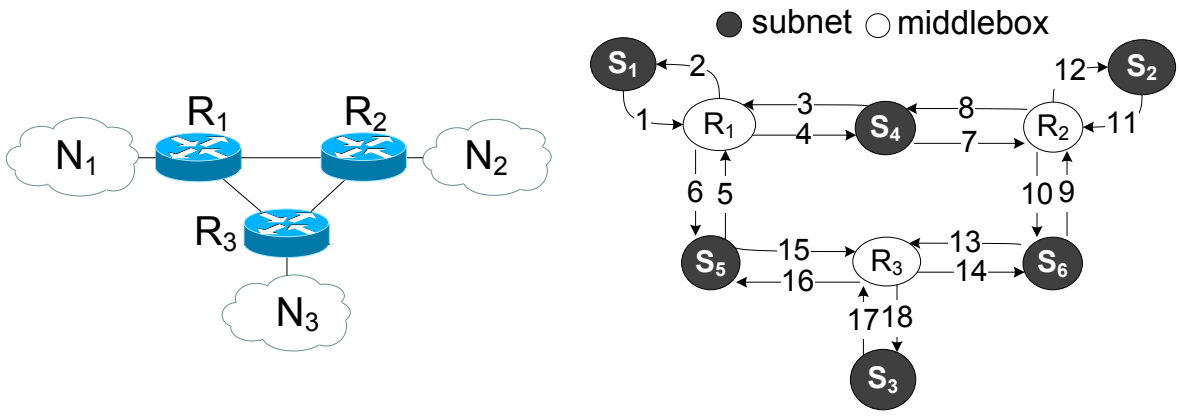
8.2.2.5 Finalization

To minimize the number of rules on each ACL, we use firewall compression techniques on the ACLs on the middleboxes. For this section, we use the techniques that are introduced in [MLT07,LTM08,LTM11]. If the final ACL sizes exceed the classifier limitations on number of rules or does not meet the operator custom performance criteria, we can rearrange the rule placement using rule expulsion to reach the desired rule placement in the network.

8.2.3 Rule Placement Example

To explain the rule placement algorithm more clearly, we run it on an example network with an example central access policy shown in Figure 8.1(a) and Figure 8.2, respectively. The example network has three routers that connects three subnets together. This network is modeled in Figure 8.1(b) with 6 subnets, three routers, and 18 links. As each links can have one classifier, we have 18 classifier to place rules to satisfy the central access policy.

Figure 8.2 shows the central access policy between three subnets of N_1, N_2, and N_3. In



(a) Example network topology (b) Example network reachability model

Figure 8.1: Example network topology and its network model

this example, we consider two dimensional rules, where the first dimension is considered as the source address field and the second dimension is considered as the destination address field. We also suppose that N_1, N_2, and N_3 has address ranges of [1, 20], [21, 40], and [41, 60], respectively. Looking at the access policies we can ensure that they only include packets that belong to their corresponding source and destination subnets. Also, the last rules for each central policy is the default policy rules that ensure the access policy is complete and cover all the address space.

$N_1 \rightarrow N_2$

F_1	F_2	
[1, 20]	[21, 30]	d
[1, 5]	[35, 35]	a
[1, 15]	[25, 30]	a
[1, 20]	[21, 40]	d

$N_1 \rightarrow N_3$

F_1	F_2	
[1, 20]	[51, 60]	a
[1, 20]	[41, 55]	d
[1, 20]	[41, 60]	a

$N_2 \rightarrow N_1$

F_1	F_2	
[31, 40]	[1, 20]	d
[21, 30]	[11, 15]	d
[21, 40]	[1, 20]	a

$N_2 \rightarrow N_3$

F_1	F_2	
[21, 30]	[41, 50]	a
[21, 40]	[41, 60]	d

$N_3 \rightarrow N_1$

F_1	F_2	
[41, 60]	[1, 20]	d

$N_3 \rightarrow N_2$

F_1	F_2	
[41, 60]	[21, 40]	a

Figure 8.2: Example central access policy between N_1, N_2, and N_3

We then put all those access policies together to build the central access policy ruleset shown in Figure 8.3(a). The last rule in the ruleset is the accept-default-policy rule to ensure the access list is complete. The central policy FDD is shown in figure 8.3(b). After running

the TRA algorithm on the FDD, the outgoing edge of node F_2 with labels of [1, 20] and [41, 60] is relabeled to [1, 100].

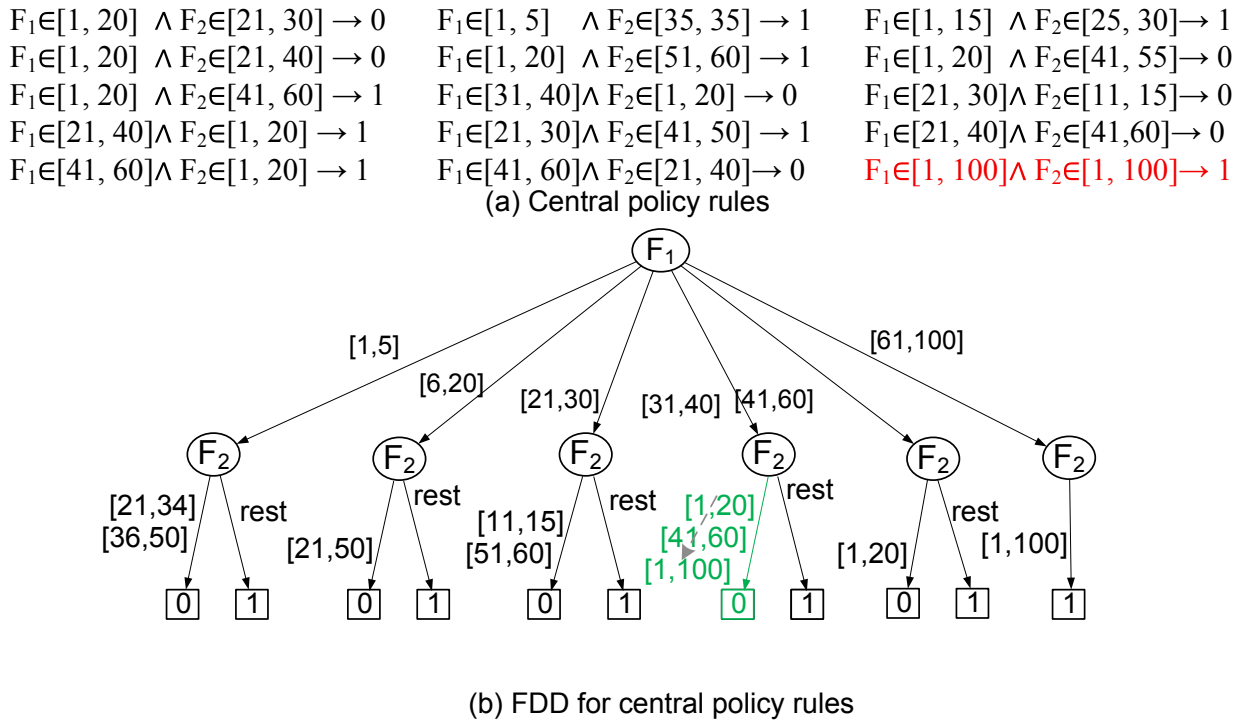


Figure 8.3: Example central access policy and its corresponding FDD

The central access policy FDD is converted to a set of non-overlapping rules in Figure 8.4(a). The next step is to calculate the effective paths to deploy these rule on their classifiers. The effective paths presented in sets of sequences of classifiers are shown in Figure 8.4(b). As mentioned, a rule is assigned to a set of paths if its source and destination range intersects with paths source address range and destination address range. For each rule and each set of path, we calculate all the edge cuts and compute the cost for each edge cut. We finally use the edge cut with minimum cost to place rule on its classifiers.

Figure 8.5 shows the rule placement for all classifiers in the network. For this example, we supposed that $w_r = w_d = 0.5$ and all middleboxes are software. Using the placed rules, we can recalculate the FDD to run topological rule aggregation algorithm. Although, in

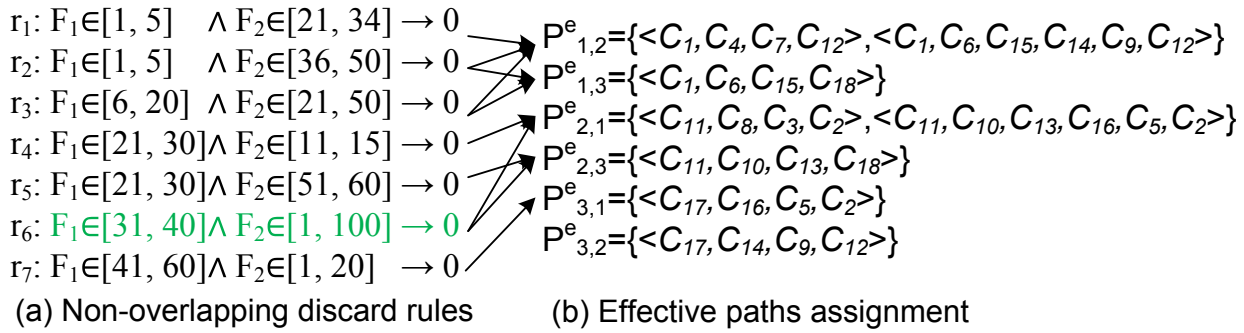


Figure 8.4: Non-overlapping discard rule assignment to the effective paths

this step of the algorithm we cannot aggregate any rules using TRA, we can simply rules so that they can be more effective in rule compression. For example, for C_4, which is only connected to N_1, the first rule's source field ([6, 20]) can be relabeled to domain size ([1, 100]). Similarly, for C_{17}, the the first rule's source field of [41, 100] can be relabeled to [1, 100]. In this example we did not have any rule expulsion, but it is feasible for r_1 to be expelled from C_1 to C_4 and C_6, or C_7 and C_{15}.

8.2.4 Handling Connection-oriented Protocols

The most simple and the most straight-forward way to place rules for connection-oriented protocols is to treat them as connectionless protocols and only consider data path and leave no rule for signalling path. This simplifies the rule placement and under this technique both stateful and stateless middleboxes can be considered as stateful where we only cares about the data path. However, if one intents to place rules on signaling path, all pairs of data path and signaling path must be considered independent paths. Also, the rules for the source and destination fields of the rules placed on the classifiers in the signalling paths must be swapped.

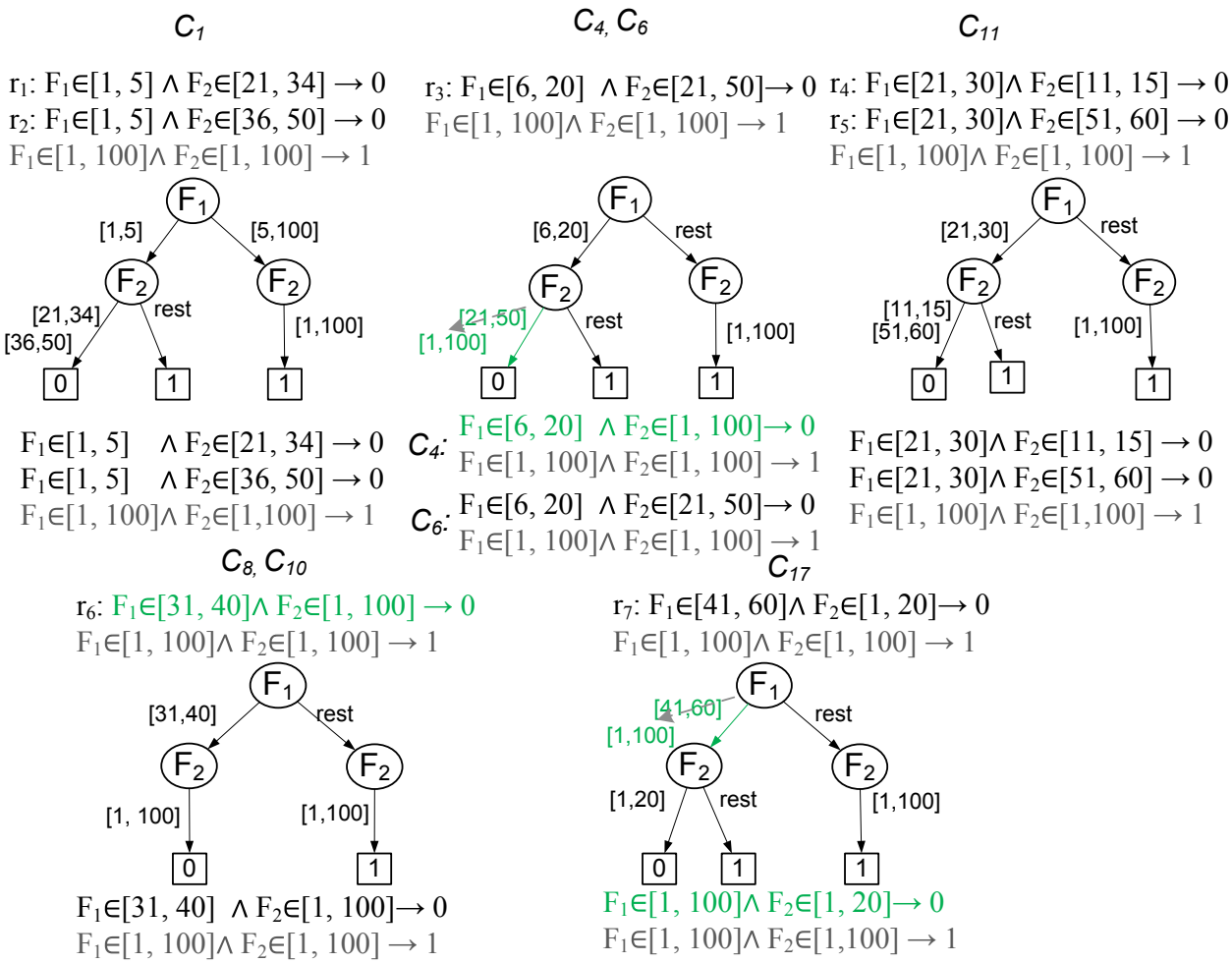


Figure 8.5: Final optimized access policy placement results (before compression) using rule placement

8.2.5 Rule Placement with IP Transformation

If there is an IP transformation middlebox such as NAT/PAT and IP tunneling is in the network, the rule placement needs to adjust the rules based on the position of the classifier to such transformation middleboxes. In case of static NAT, the source fields of the rules after the middlebox in the path from a source to destination should be transformed based on the NAT transformation function. If the middleboxes are configured for dynamic NAT, the source fields' intervals for the rules after the middlebox in a path must be divided to the

block of transforming IPs and be replaced by the transformed IP block. Note that the source field intervals must be dividable to blocks of transforming IP addresses, otherwise, it is a misconfiguration that should be notified to the network operator. In case of IP tunneling, it is highly recommended not to place rules on the classifiers in the tunnel. If access is required to be restricted, the rules is better be placed outside of the tunnel, preferably before tunnel, or after the tunnel. Otherwise, it is better to disable the tunnel for that pair of subnets.

8.2.6 Interval-based vs Prefix-based Classifiers

The interval-based classifiers contains the IP addresses in form of intervals, while the prefix-based classifiers contains the IP addresses in form of IP prefixes. Usually software middleboxes can accept both interval-based and prefix-based IP ranges, whereas the hardware classifiers works based on prefix-based classifiers.

Rule placement is designed for interval-based classifiers rather than prefix-based classifiers. The reason is the non-overlapping discard rules are interval-based. For a prefix-based middlebox, we need to convert an interval to a set of prefixes. Yet, a non-overlapping discard rule can be converted to a set of rules which theoretically can be very large [Liuar]. Hence, for such middleboxes, we either need to use the ACL placement algorithms for access policy design, or use the compression techniques introduced for hardware middleboxes [MLT07, LMZ08].

Chapter 9

Discussion

9.1 Handling Interface Rules

Some middleboxes (such as Cisco PIX firewalls [PIX09] and Sidewinder Firewalls [Sid09]) allow network operators to specify rules to accept or discard all packets coming in from one interface and forwarded to another interface. For example, the `security-level` command in Cisco PIX firewalls implement such rules. Such rules, unlike ACL rules specified over packet header fields, can be easily handled by Quarnet. If ACL rules have higher priority over interface rules, which is the case for PIX firewalls, then the interface rules can be modeled as the default rule in the two ACLs. For example, an interface rule discarding all packets from incoming interface `int1` to outgoing interface `int2` essentially specifies that the default rules in the ACLs guarding `int1` and `int2` are all `discard`. If interface rules have higher priority over ACL rules, such interface rules can be easily handled by Quarnet in the FDD matrix initialization phase described in chapter 3.3. Recall that in this phase, we first calculate the FDDs for one-hop paths. For a one-hop path connected by one middlebox and two links, one connected to an incoming interface of the middlebox and one connected to an outgoing

interface of the middlebox, if there is an `accept` interface rule for the two interfaces, we simply model the FDD as one accepting all packets; if there is a `discard` interface rule for the two interfaces, we simply model the FDD to discard all packets.

9.2 Complexity Analysis

9.2.1 Quantification

For a given network, let n be the number of subnets, τ be the length of the longest path, p be the number of the paths in the network, g be the maximum of number of rules in an ACL, and d be the number of fields in each rule. Note that d is typically a constant, which is 4 or 5 for IP networks. The complexity of constructing the equivalent FDD from an ACL with g d-dimensional rules is $O(g^d)$ [LG08]. The complexity of shaping the two FDDs constructed from two ACLs is $O((2g)^d) = O(g^d)$. Therefore, the complexity of computing reachability matrices is $O(\tau \cdot p \cdot g^d)$. As the length of the paths cannot exceeds a constant (*i.e.* 30), the complexity turns into $O(p \cdot g^d)$.

In theory, the total number of paths p is exponential in terms of the number of subnets and middleboxes in the network. However, in practice, p is much smaller than its theoretical upper-bound because networks are typically designed following the hierarchical network design model [cis]. Using this model, a network is designed in three layers, namely a core layer, a distribution layer, and an access layer. Security policies are mostly applied on the distribution layer, and the core layer is mainly used to facilitate routing between subnets. For networks designed by this model, the number of paths between two subnets is typically small (often one), and the length of a path is typically short including a few middleboxes.

9.2.2 Querying

The complexity for querying is O(1), as we only need to find the FDD (it also could be given based on the query) and traverse the target FDD to return the result for the query.

9.2.3 Diagnosis and Troubleshooting

Having the FDDs, the complexity for diagnosis for $|C|$ number of classifiers that has maximum rule number of g is $O(|C|g)$ per instantaneous reachability problem. The algorithm to resolve type 1 errors has the complexity of $O(|C|)$ per error. However, for the network model graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{F})$ presented in chapter 2, the algorithm to resolve type 2 errors is $O(|\mathcal{V}||\mathcal{E}|log(\frac{|\mathcal{V}|^2}{|\mathcal{E}|}))$, if we compute the minimum edge cut using Hao and Orlin algorithm [hao]. It seems that is the best algorithm with the lowest complexity for computing the minimum edge cut. Yet, our implementation is based on the BDD and reliability theory, in which we calculate all edge cuts once, and for each non-overlapping rule, we recalculate the total cost for the each edge cut and we choose the edge cut with minimum cost. In theory, the complexity of this technique is exponential in terms of number of edges; however in practice, it can be even faster than computing minimum edge cut individually.

9.2.4 Optimization

Let υ be the number of rules in the network central access policy. The complexity for constructing the central access policy FDD is $O(\upsilon^d)$. Let the total number of non-overlapping discard rules be $\bar{\upsilon}$. For the network model graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{F})$ presented in chapter 2, similar to the troubleshooting, the complexity of cost-based rule placement for each rule is $O(|\mathcal{V}||\mathcal{E}|log(\frac{|\mathcal{V}|^2}{|\mathcal{E}|}))$. Hence, the total complexity for basic rule placement (excluding TRA

and compression) is $O(v^d + \bar{v}|\mathcal{V}||\mathcal{E}|log(\frac{|\mathcal{V}|^2}{|\mathcal{E}|}))$.

9.3 Incremental Changes

As a network evolves over time, we need to update reachability matrices accordingly. There are three types of network changes that will affect reachability matrices: topology changes, configuration changes, and ACL changes. We explain how each of these incremental changes is handled in different parts of Quarnet.

9.3.1 Quantification, Verification, and Troubleshooting

9.3.1.1 Network Topology Changes

Any modification to the physical structure of a network, such as adding/removing a link, leads to network topology change. When such changes happen, we need to recompute the path matrix. For any new path in the new path matrix, we need to recompute the corresponding element in the FDD matrix F. The reachability matrices need to be updated accordingly.

9.3.1.2 Subnet Configuration Changes

Network configuration changes refer to the changes in the subnet addresses or subnet masks. For such changes, the third step in computing the FDD matrix needs to be performed for the paths whose beginning or ending subnets are changed.

9.3.1.3 Middlebox ACL Changes

ACL changes refer to the changes on ACL rules. When an ACL on a link changes, for all the paths that contains the link, we need to recompute the

9.3.2 Optimization

9.3.2.1 Network Topology Changes

The optimization algorithm is based on all the effective paths between a pair of subnets. If the network topology change adds new paths to the current paths, the rule placement for the new set of paths should be done accordingly. In this case, the new set of paths are considered as effective paths and the algorithm must be re-executed on new set of paths. If the network topology change removes some paths, no new computation is required. If the network topology changes such that some paths are removed and some new paths created, we need to remove the rules added based on central access policy of all pairs of subnets whose set of paths are changed. We then need to consider the new paths and recompute the rules for each classifier in the path. Indeed, major topology changes that change all the paths will lead to recalculation of rule placement for all classifiers.

If the network change adds or removes subnets, accordingly the central policy should be changed. In this case for each new central policy we need to re-execute the algorithm. If some central policy is removed, we can remove its associated rules from the all the classifiers.

9.3.2.2 Central Access Policy Changes

If central access policies are changed, we need to recalculate the central (or partial) FDD and regenerate all the non-overlapping discard rules. If some non-overlapping discard rules

are removed, then those rules should be eliminated from the classifiers across the network. If some new rules are generated, then they should be independently placed on the classifiers.

Note that Quarnet keeps all edge cuts and non-overlapping discard rules (before compression) so that it can easily handle incremental changes.

9.4 Integrating Network Routing for Real-Time Reachability

To effectively monitor real-time network reachability, the network routing can be integrated with the instantaneous reachability matrix. If the Quarnet server collects routing states from the middleboxes once they change, it can identify the real-time reachability between all pairs of subnets. Moreover, by corresponding time to the network reachability the network operator can better understand the dynamics of temporal network outages or security breaches. He can also generate real-time queries to troubleshoot ongoing reachability problems.

9.5 Practical Issues of Quarnet

In practice, the computed network reachability due to undocumented and neglected middleboxes (*e.g.*, personal hardware or software firewalls, a filtering bridge), links (*e.g.*, temporal test links), or services (*e.g.* Port Knocking [por]). Moreover, a misbehaving device can cause problems, which are untraceable using Quarnet. Hence, to minimize such difference and have accurate network reachability database, it is advised that network operators document and add all filtering devices to the Quarnet server and make sure that it is fed with all possible information that concerns network reachability.

Chapter 10

Experimental Results

10.1 Implementation and Evaluation Testbeds

We implemented Quarnet in C++ and evaluated it on two testbeds: (1) a university campus network, (2) real-life based networks with large variety of configurations.

The university campus consists of 48 subnets interconnected in a hierarchical topology. We model the core campus network as one subnet as most ACLs are deployed on edge routers/firewalls. We disregard the ACLs deployed on the core routers in the network because we have no access to them. Furthermore, their potential impact on the reachability calculation is expected to be small because the ACLs on core routers are typically very small and they mostly specify a few rules allowing only administrators to access the middleboxes for management purposes [cis]. One subnet may contain multiple Virtual Local Area Networks (VLANs) and there are no ACLs among VLANs. For example, a VLAN could be the network that consists of all the printers in the subnet of a department. Note that we model the outside Internet as one subnet. This network does not have NATs/PATs or IP tunnels. Further, all ACLs that we obtained are in use on stateful firewalls. In this network

model, there are 49 subnets that are connected by 192 links through 2401 paths. Also, the total number of VLANs is 399 that are protected by 98 ACLs, where each ACL contains 573 rules in average (total number of rules: 56189). Among the 98 ACLs, 14 of them are original and the rest are generated based on the statistical features of the original ones and the subnet addresses because we do not have access to all the ACLs in deployment. We used university campus topology to evaluate Quarnet in terms of execution time and memory usage for reachability computation and querying.

However, since the university campus topology has one paths between each two subnets, we used another another real-life topologies to be able to properly evaluate probabilistic reachability and reachability troubleshooting. Hence, we create four network topologies by adding and removing some links and subnets to a real network topology with 8 routers and original 32 links. These networks has different number of paths ranges from 122 to 4010. For the ACLs on the interfaces of the routers, we provide a pool of real-life rules and we put 20±5 rules on routers interfaces. More details of the network topologies are shown in Table 10.1.

bb	No. of paths	No. of subnets	No. of routers	No. of links
Test Case 1	122	9	8	32
Test Case 2	1036	8	8	39
Test Case 3	2019	10	8	39
Test Case 4	4010	8	8	43

Table 10.1: Created networks statistics

10.2 Reachability Computation

We compute reachability for a university campus network on a desktop computer with a Dual Core AMD64 CPU 2.4GHz and 16GB of RAM. This is a public machine running processes from other users as well. Our experimental results are shown in Table 10.2. Note that the total amount of memory used by Quarnet is 4.7GB, not the sum of the memory used in each step because some memory is released after each intermediate step.

	# of FDDs	Time (mins)	RAM (MB)
FDD construction	192	2.1	1276
One-hop path calculation	96	0.3	106
FDD matrix calculation	2352	11	1505
Reachability calculation for CL protocols	2352	661	2400
Reachability calculation for CO protocols	2352	1	800
Total execution	7344	675	4700

Table 10.2: Results on computing reachability matrices

We gained two insights from our experiments. First, the running time of our algorithm goes up as the number of VLANs increases. This is because more number of VLANs means more intervals on the outgoing edges of nodes labeled with source or destination fields, which further means more edge splitting and subtree copying in performing FDD shaping. Second, the time for computing reachability matrices for connection-oriented protocols goes down dramatically as the number of stateful firewalls increases. This is because based on formulas (3.6), (3.8), and (3.7), we can use the FDDs calculated in the connectionless matrices and simply change the subtree functions from Sub_{CL} to Sub_{CO}.

10.3 Experimental Results Validation

To validate the reachability matrices computed by Quarnet, we designed an ACL simulator that makes the decision for each packet by sequentially comparing the packet with every rule in an ACL. We generated a large number of packets for every path including all corner case packets based on the bounding values in the ACLs. We compared the decisions made by the ACL simulator and those made by the reachability matrices computed by Quarnet. The results are all positive.

10.4 Performance of Core Operations

The core operations in reachability computation are FDD logical AND and OR operations, the performance of which is the same as they all come down to FDD shaping. We are interested in evaluating their performance because it helps us to estimate the running time and memory usage of reachability matrix calculation. To evaluate the performance of the core operations, we created synthetic rules generated based on the statistical features of the real rules that we have obtained, such as the probability of unique IP addresses and port numbers, the decision bias of the rules, and the probability of fields being `any`. We focused on measuring the time and memory for calculating the reachability of paths with different lengths.

Figure 10.1 (a) and (b) show the running time and memory usage of the core operations over different path lengths for a network with 100 subnets. The average number of rules per ACL ranges from 100 to 400 with 10% STD based on normal distribution. Note that based on the dynamic programming in Section 3.3, the FDD for each path is calculated using a

single logical operation. For instance, the FDD for a path of length 8 is calculated by an AND operation of either two FDDs where each is for a path of length 4 or two FDDs, one for a path of length 6 and one for a path of length 2. For any path length, we calculate the average cost of all possible permutations by which the FDD of the path may be calculated.

The interesting observation on Figure 10.1 is that in general the running time and memory usage decrease as the path length increases. This makes sense because the rules in different ACLs from one network often share common fields. The source and destination IP address fields of many rules in the ACLs from one network are drawn from the IP prefixes of the subnets in the network. Similarly, many of the port number and protocol fields are drawn from the set of services provided by the network. When we shape two FDDs each corresponds to a long path, because each of the two FDDs has gone through many edge splitting, the amount of new edge splitting tends to be small, which leads to reduced cost.

Knowing the performance of Quarnet core operations allows us to estimate the time and memory that it used to compute the reachability matrices for a given network. Let $\mathcal{C}(x)$

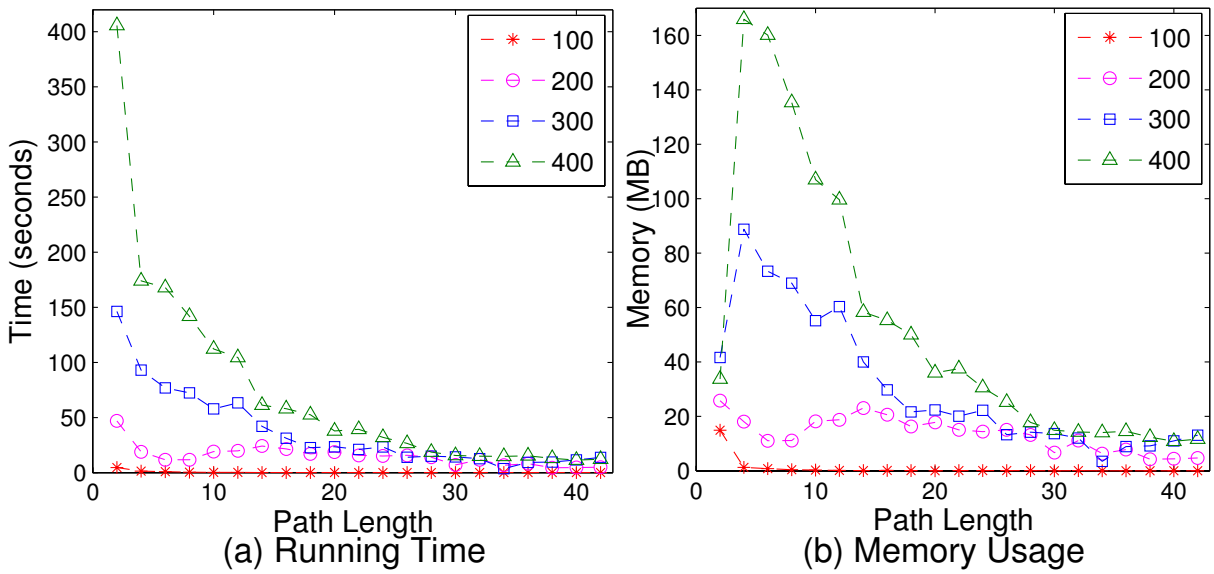


Figure 10.1: Performance of Quarnet core operations

be the cost function for calculating the FDD for a path with length x as shown in figure 10.1. Let $\mathcal{L}(P[i,j]_k)$ be the length of the path $P[i,j]_k$. We use $\mathfrak{C}^I_{CL}$, $\mathfrak{C}^I_{CO}$, $\mathfrak{C}^B_{CL}$, and $\mathfrak{C}^B_{CO}$ to denote the estimated cost (*i.e.*, running time or memory usage) for computing the matrices of instantaneous reachability for connectionless protocols, instantaneous reachability for connection-oriented protocols, reachability bounds for connectionless protocols, and reachability bounds for connection-oriented protocols, respectively. They can be calculated as follows:

$$\mathfrak{C}^I_{CL} = \sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{k=1}^{|P[i,j]|}\mathcal{C}(\mathcal{L}(P[i,j]_k))$$

$$\mathfrak{C}^I_{CO} = \sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{k=1}^{|P[i,j]|}\sum_{k'=1}^{|P[j,i]|}\mathcal{C}(\mathcal{L}(P[i,j]_k) + \mathcal{L}(P[j,i]_{k'})))$$

$$\mathfrak{C}^B_{CL} = \sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{k=2}^{|P[i,j]|}\mathcal{C}(\sum_{k'=1}^{k}\mathcal{L}(P[i,j]_{k'}))$$

$$\mathfrak{C}^B_{CO} = \sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{k=1}^{|P[i,j]|}\sum_{l=1}^{|P[j,i]|}\mathcal{C}(\sum_{k'=1}^{k}\mathcal{L}(P[i,j]_{k'}) + \sum_{l'=1}^{l}\mathcal{L}(P[j,i]_{l'}))$$

10.5 Quarnet Scalability Analysis

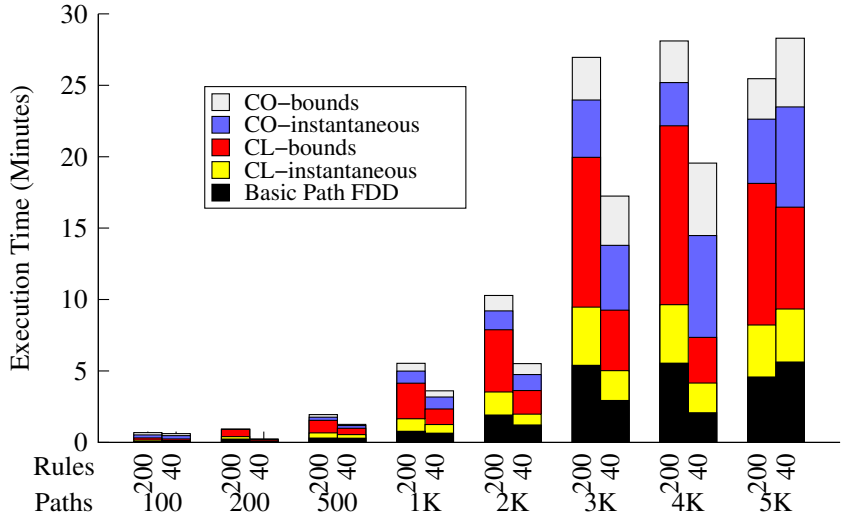
In this section, we evaluate the scalability of Quarnet by increasing the number of paths and rules on synthetic network topologies and synthetic ACL rules generated. We trained our rule generation model using real network topologies and configurations to generate close-to-

reality networks. We use a large-scale enterprise network with 8 middleboxes (*i.e.* routers with ACLs on their interfaces), which divide the network into 10 subnets. We change the network connectivity by adding and removing edges to change the number of paths from 100 to 5,000. We perform scalability tests twice for each network topology, where in each test we use 20 ACLs on random router interfaces with 40±10 and 200±50 rules. The results for reachability computation execution time and memory are shown in Figure 10.5(a) and (b), respectively.

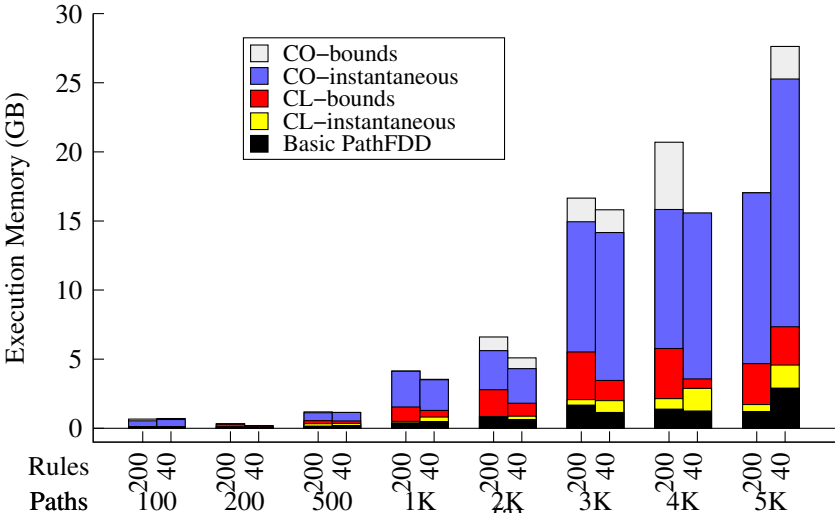
The results confirms the observations we had in previous sections. First, in most cases the execution time and memory usage increases linearly when the number of rules and paths increases. But at some point, the number of paths and rules does not have much impact because the FDDs for different paths are very similar, which in turn means that less number of edge splitting and subtree copying is required. Second, the execution time and memory can vary for different Quernet operations. While the computation of CL matrices takes more time, the computation of CO matrices takes more memory.

Note that by comparing the results for synthetic networks and real-life network, we can observe a very large difference between computation time of real-life networks (*i.e.* approximately 11 hours) and the computation time of synthetic networks (*i.e.* approximately 30 minutes). As mentioned, the reason is because the number of VLANs (*i.e.* prefixes) is considerably larger (399 vs. 10) the diversity of rules is much higher, which leads to larger FDDs with more edge splitting and subtree copying in FDD shaping.

We also examined NAT/PAT on synthetic networks. The results shows a negligible difference with reachability computation with no NAT/PAT. The reason is for instantaneous reachability, the only overhead in computation is relabeling edges and for bounds, because FDD's subtrees are very similar the overhead of FDD subtree shaping is trivial.



(a) Running Time



(b) Memory Usage

Figure 10.2: Performance of Quarnet for synthetic networks

10.6 Performance of Online Querying

As our reachability query engine uses FDDs as its core data structures, we evaluated the performance of online query processing by performing randomly generated queries over large FDDs with millions of nodes. Our experimental results in Table 10.3 show that our online reachability query engine is very efficient. For example, over an FDD with 2 million nodes, which is similar to the size of the FDDs uses in the online query engine built for the university

campus network that we experimented, the average processing time for a query is 0.075 seconds, although some queries (less than 1%) take 2-3 seconds. The existence of some outliers is because some randomly generated queries may cause the engine to search through a large portion of the FDD.

FDD Size (# nodes)	Average Query Processing Time	Outliers
0.5 million	0.032s	1% of queries: 0.5s $\sim$ 1s
1 million	0.049s	0.8% of queries: 0.8s $\sim$ 1.5s
2 million	0.075s	1% of queries: 2s $\sim$ 3s

Table 10.3: Performance of online query processing

10.7 Probabilistic Reachability Computation

Using the second testbed, we focused the measurement on the algorithm execution time and memory usage for probabilistic reachability computation, instantaneous reachability error diagnosis and troubleshooting and reachability inconsistencies diagnosis and troubleshooting. We conducted experiments on a UNIX machine with Intel(R) Xeon(R) Quad-cores CPU running at 2.66GHz and 16GB of RAM. We concluded that the proposed algorithms are sufficiently efficient to be used in practice for online reachability troubleshooting.

Figure 10.3(a) and (b) show the number of effective paths and the percentage of effective paths out of all the paths, respectively, for 5 different reachability uptime values of $\epsilon = 0.9, 0.99, 0.999, 0.9999$ and 0.99999. The results indicate that in large-scale and highly connected networks (*e.g.* test case 3 and 4) where there are too many paths, up to 52.42% of the paths can be ignored as their reachability is not a concern in 0.99999 of the time. This impact, however, is less for small scale and weakly-connected networks. Another observation is that large scale networks are more sensitive to ϵ rather than small networks, which

indeed suggests that probabilistic reachability computation can be very effective for large scale networks.

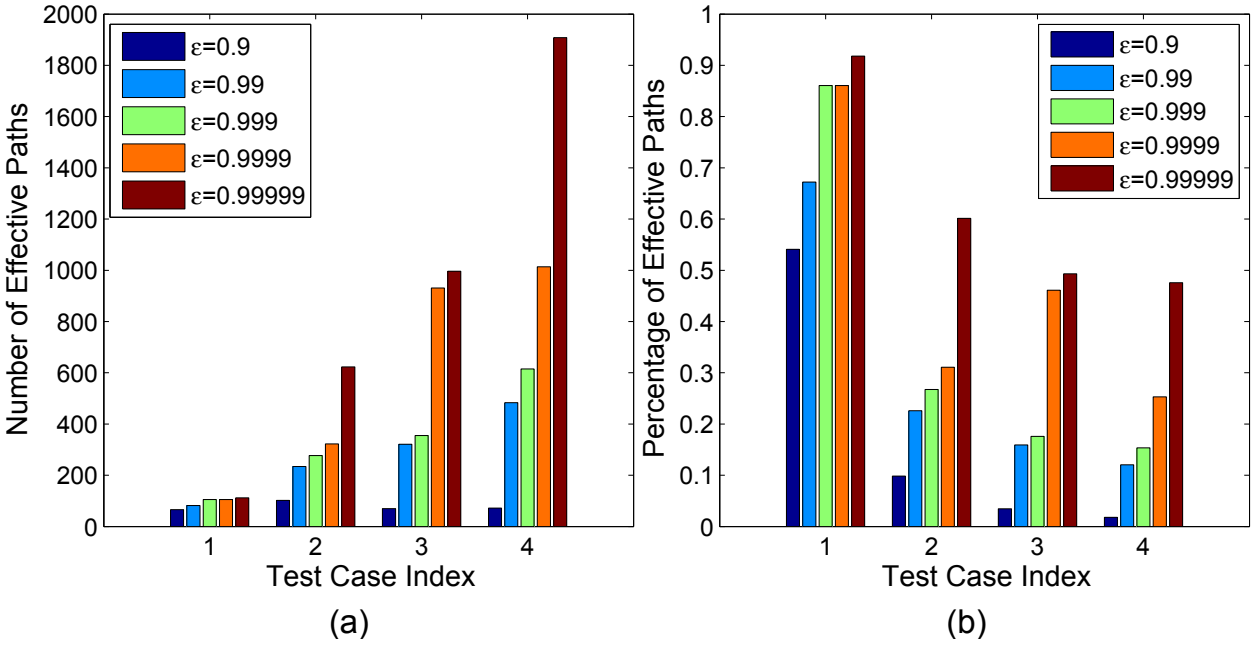


Figure 10.3: Number of effective paths for different reachability uptimes

Figure 10.4(a) and (b) show the time and memory for calculating effective instantaneous reachability, respectively. The results indicate that in most cases as ϵ increases, the probabilistic reachability calculation time increases exponentially. The reason is larger values of ϵ leads to larger number of effective paths, which are mostly longer (*i.e.* have more ACLs); thus, their reachability calculation takes even more time and memory. For example, for a large scale network (*i.e.* test case 4), the calculation time varies from 5 seconds to 31 hours, and calculation memory varies from 88MB to 11.38GB for different values of ϵ. Thus, finding an appropriate ϵ can drastically change the reachability calculation time and memory.

Figure 10.5(a) and 10.5(b) show the calculation time and memory for calculating effective reachability bounds, respectively. The results indicate that although calculating of reachability bounds is not as resource-intensive as calculating instantaneous reachability, the

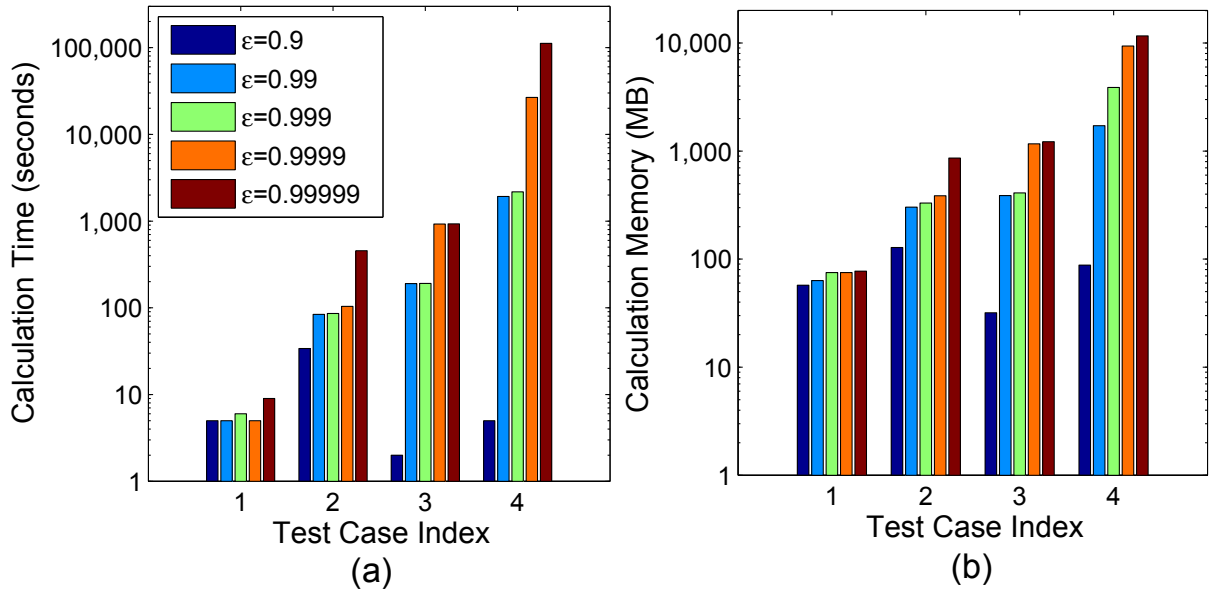


Figure 10.4: Effective instantaneous reachability calculation time and memory

execution time and memory usage both increase as reachability uptime increases. The reason is that the reachability bound calculation is mostly dependent on the number of the paths and not the length of the paths. Thus, when reachability uptime value increases, its impact on required time and memory is less significant. For instance, for a large scale network (*i.e.* test case 4), the calculation time can vary from 3 seconds to 56.6 minutes, and calculation memory can vary from 60MB to 697MB for different values of ϵ.

In summary, The results indicate that for large scale networks, we can save significant amount of time and memory, if we calculate probabilistic reachability and disregard the paths that are unlikely to be used in a network.

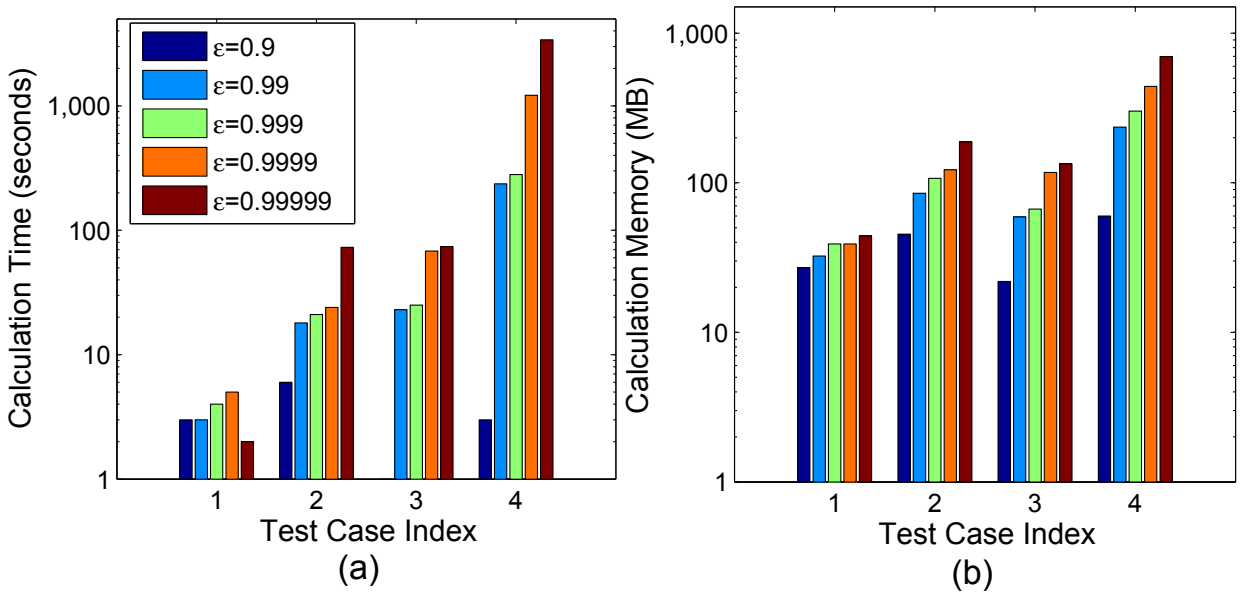


Figure 10.5: The effective reachability bounds calculation time and memory

10.8 Instantaneous Reachability Error Troubleshooting

To measure the instantaneous reachability error troubleshooting time and memory usage, we first setup a linear network, which consists of 5 routers (*i.e.* 10 ACLs) lined up in a row, and we consider the paths from a subnet connected to router 1 to a subnet connected to router 5. Second, we consider four test cases where in each test the average number of rules on the ACLs are in turn $10 \pm 2, 100 \pm 20, 1000 \pm 200$, and 10000 ± 2000. Note that the rules are chosen from a pool of real rules. Finally, we measure the instantaneous reachability error detection and troubleshooting time and memory for different number of errors varying from 10 to 5000 errors. To create reachability errors, we generate a random set of packet header predicates to represent the set of errors. For each error predicate, if the decision for the error is 0 based on the instantaneous reachability, it is considered as type 1 error and we fix the ACLs to change the error reachability to 1. Similarly, if the decision for the error is 1 based

on the instantaneous reachability, it is considered as type 2 error and we fix the ACLs to change the error reachability to 0.

Figure 10.6 shows the time for instantaneous reachability error diagnosis and troubleshooting. The results indicate that the error diagnosis time has a nearly constant trend for different number of the errors. The reason is our error query algorithm queries all the reachability errors at the same time. Thus, the query time may change based on the size of instantaneous reachability FDD rather than the number of errors. However, the troubleshooting time increases linearly as number of errors increases. From the results we can deduce that test case 3 has the largest instantaneous reachability FDD and its average detection time is 791ms for different number of errors. Test case 4, however, has the largest troubleshooting time that varies from 106ms to 75s for different number of errors.

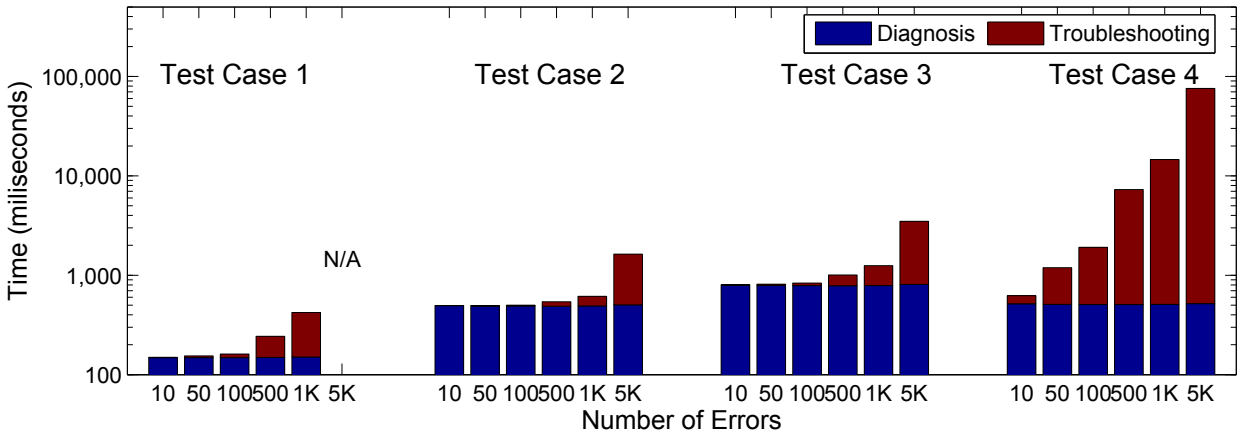


Figure 10.6: Instantaneous reachability faulty ACL detection and troubleshooting time

Figure 10.7 shows the memory for instantaneous reachability error diagnosis and troubleshooting. Similar to the error diagnosis time, the memory remains almost constant when the number of errors increases. Test case 3, which has largest instantaneous reachability FDD among all the test cases, consumes the most amount of memory up to 32MB. This figure also shows that the reachability error troubleshooting consumes trivial amount of memory,

which was expected as the troubleshooting algorithm is not memory-intensive.

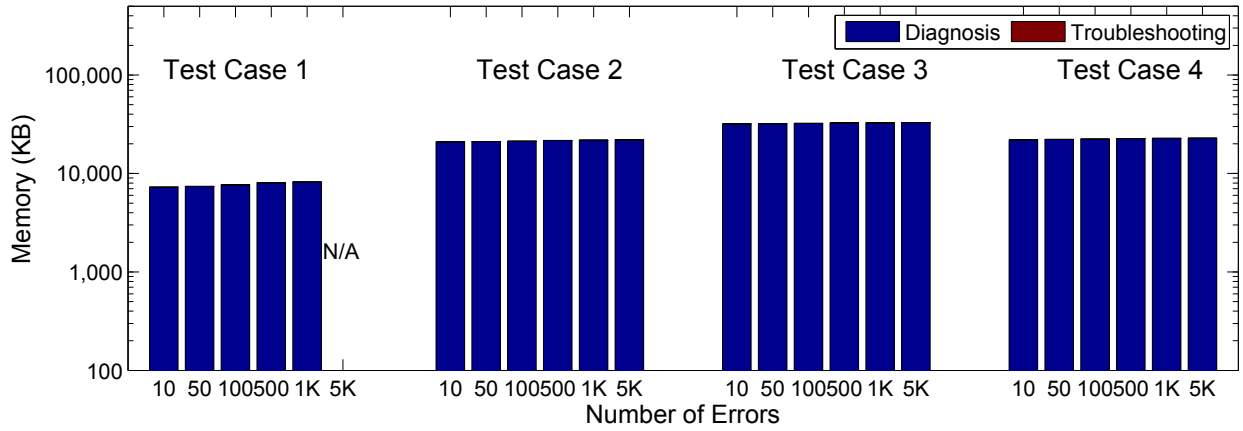


Figure 10.7: Instantaneous reachability faulty ACL detection and troubleshooting memory

Both figures suggest that if we calculate instantaneous reachability off-line, we can perform error diagnosis and troubleshooting online in a timely and memory efficient fashion.

10.9 Reachability Inconsistencies Troubleshooting

To measure the reachability inconsistencies detection and troubleshooting time and memory, we first use a network with 805 paths where only 596 of them are effective. The number of paths between randomly selected pairs of subnets are sorted and shown in Figure 10.8(a). We then generate four test cases where in each test case the routers have in turn up to 10, 50, 100, and 500 real-life rules. The rules on the routers on parallel paths are chosen to be somehow relative to avoid unpractical situations where there is a big difference between the reachabilities of parallel paths. To create our central policies, we randomly choose the correct decisions for the calculated reachability inconsistencies and accordingly we identify type 1 and type 2 reachability errors. The numbers of rules to fix type 1 and type 2 reachability errors for each selected pair of subnets are shown in Figure 10.8(b) and (c), respectively.

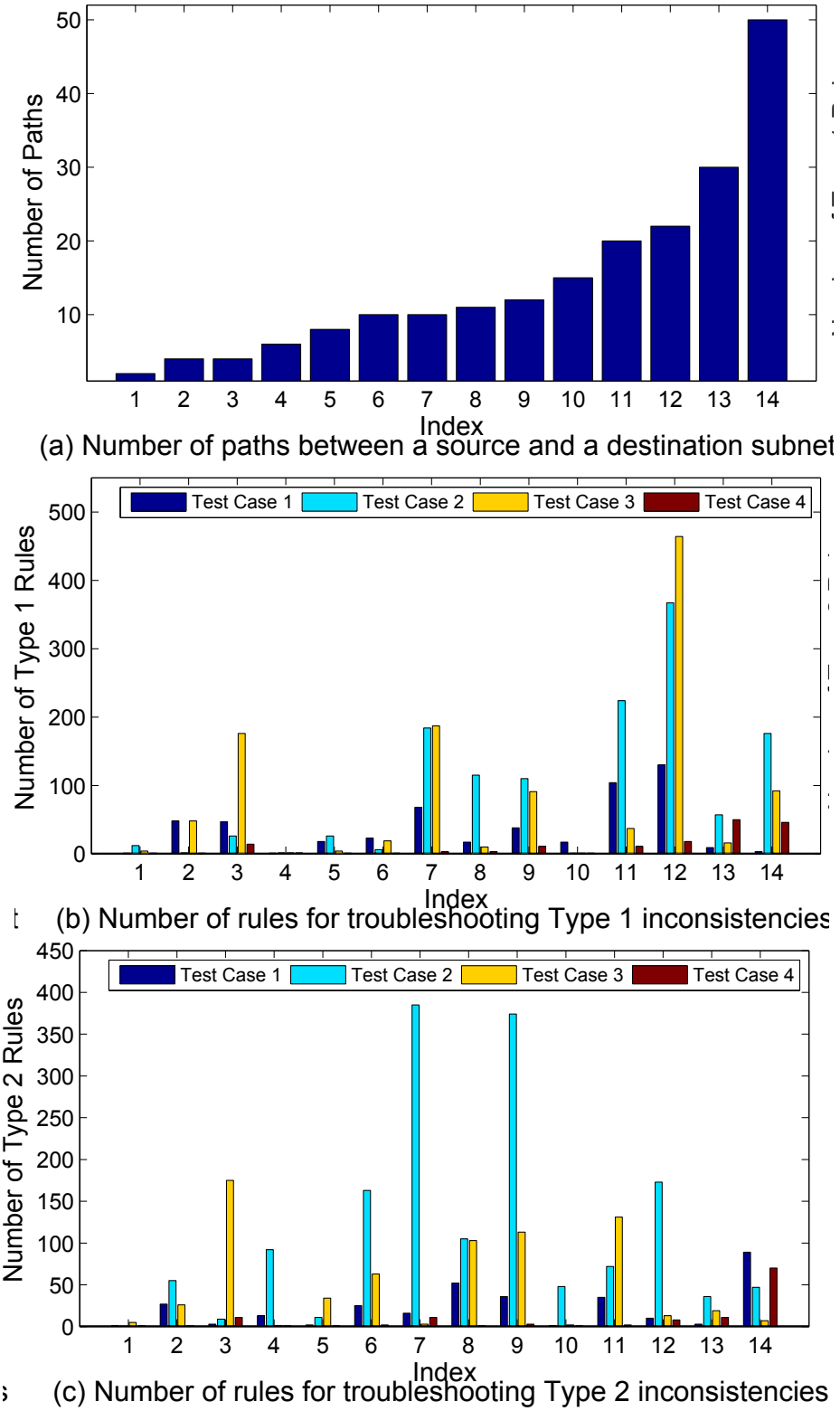


Figure 10.8: Reachability inconsistencies detection and troubleshooting time

The results show that the number of fixing rules is more dependent on the ACL rules and their interactions rather than the number of paths.

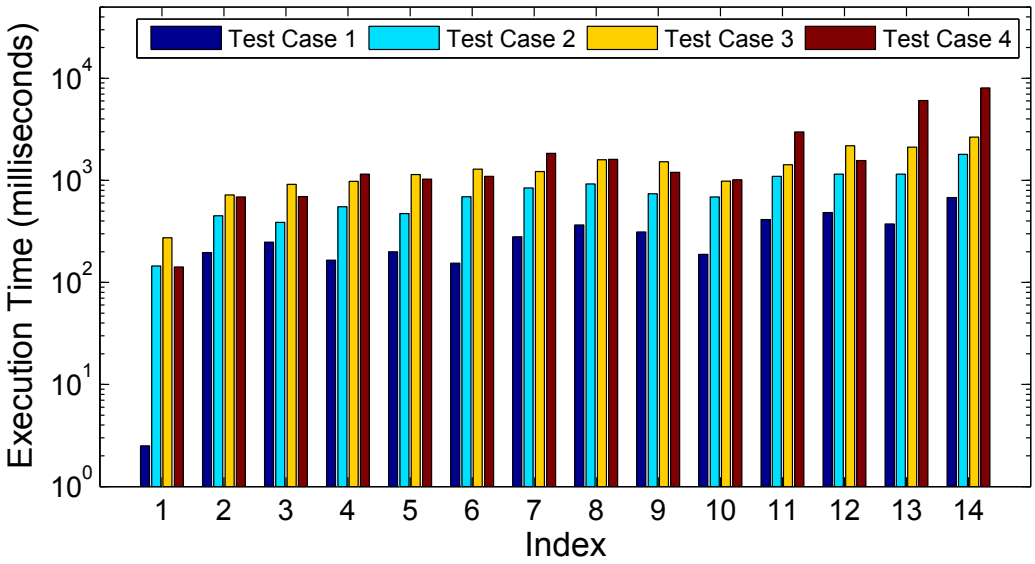


Figure 10.9: Reachability inconsistency detection time

Figure 10.9 shows that reachability inconsistencies can be detected in less than a few seconds. It also shows that the detection time is only dependent on the number of rules and the number of paths, and not dependent on the number of errors.

Figures 10.10 and 10.11 show the troubleshooting time for type 1 and type 2 errors. The results indicate that when the number of errors increases, the troubleshooting time increases as well. By comparing these two figures, we can observe that fixing type 1 errors takes less time than fixing type 2 errors. The reason is that type 2 errors troubleshooting has a time overhead due to fixing cost computations. The results show that the type 1 error troubleshooting takes less than a second whereas type 2 error troubleshooting takes tens of minutes when the number of errors exceeds 350.

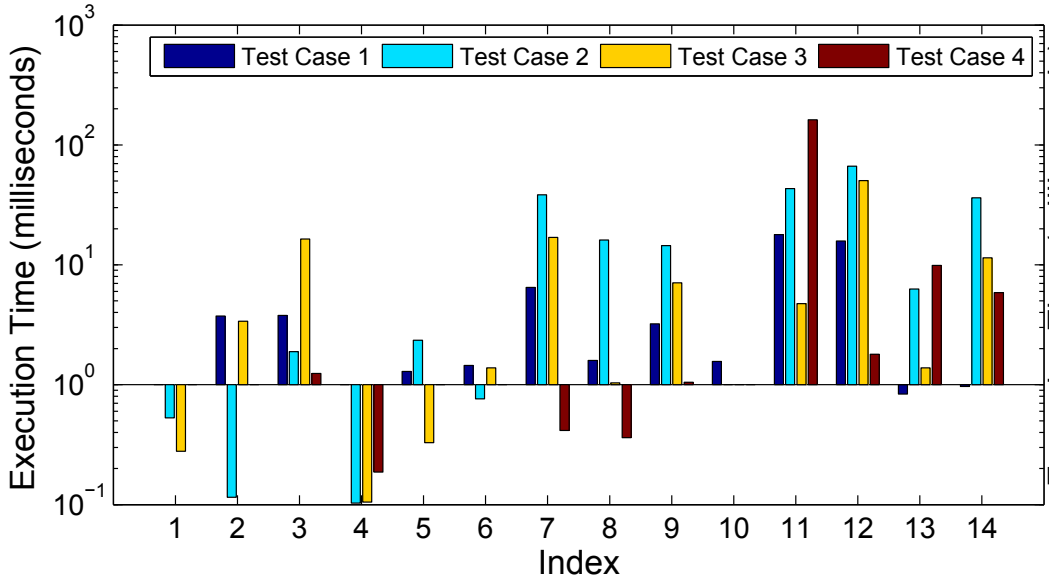


Figure 10.10: Reachability inconsistency type 1 troubleshooting time

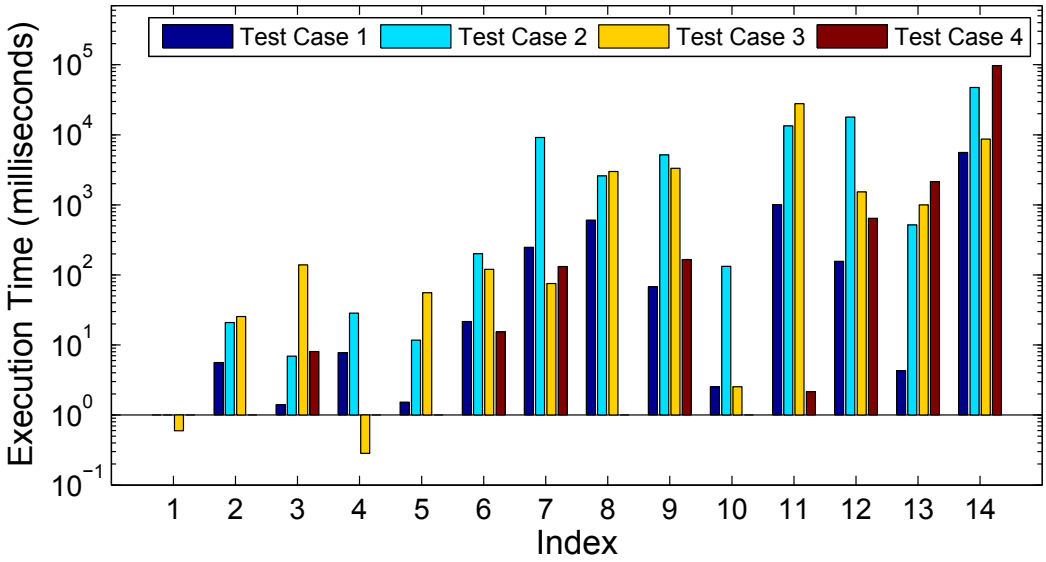


Figure 10.11: Reachability inconsistency type 2 troubleshooting time

10.10 Reachability Optimization

To evaluate the performance of the rule placement algorithms, we implemented them as a part of Quarnet tool. The input for the algorithm can be either the network central access policy, or the central policy calculated by the rules that have been already placed on the

classifiers and will be redesigned based on the algorithm output. We also implemented the ACL placement algorithm based on Sung *et al.*'s work in [SRXM08] to compare the results for rule placement and ACL placement. Comparing rule placement and ACL placement is difficult because, as mentioned before, the ACL placement can only satisfies one strategy at a time; hence, comparing it to rule placement that creates a tradeoff between different requirements may not seem fair. Nonetheless, to have a reference point to evaluate the performance of the rule placement, we compare ACL placement with one strategy and rule placement with three configurations.

For this experiments, we consider three network topologies shown in Figure 10.12. Figure 10.12(a) shows the test case 1 network topology, which is a small network with 4 subnets connected in a ring-like topology. Such topologies are often used for networks with high availability and reliability. Figure 10.12(b) shows the test case 2 network topology. This is a mid-size network where two networks are connected through a core router together. In this network, there are 6 main subnets whose access to each other is limited based on the central access policy. Figure 10.12(c) shows a network with hierarchical architecture where the core layer (hilighted with orange background) interconnects the access routers (highlighted with yellow background) and their 12 subnets.

The experiment is conducted where the central policy is created synthetically using a model, which is trained by a set of real-life access policies collected from a university campus network. The central access policies between each pair of subnets is designed to be around 100 rules. The decision bias of the rules is 95% for `discard` decision, but the default access policy is `accept`. We will later explain how the decision bias can be important in the final results. The ACL placement algorithm is designed based on "Minimum Rules (MIN)" strategy, in which the minimum number of rules are placed on the classifier. Unfortunately,

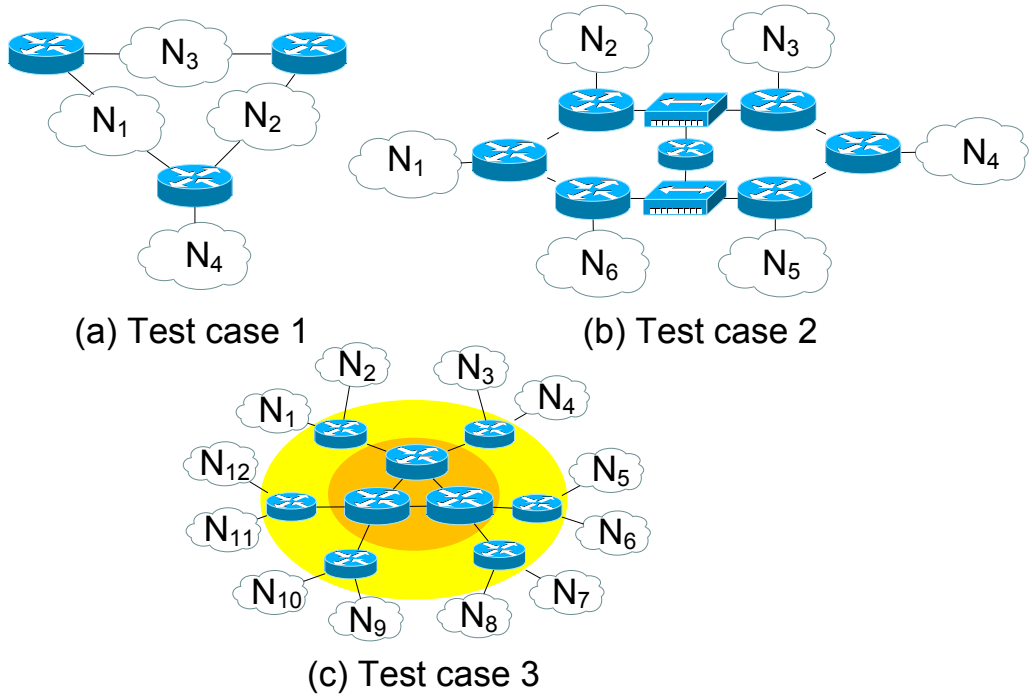
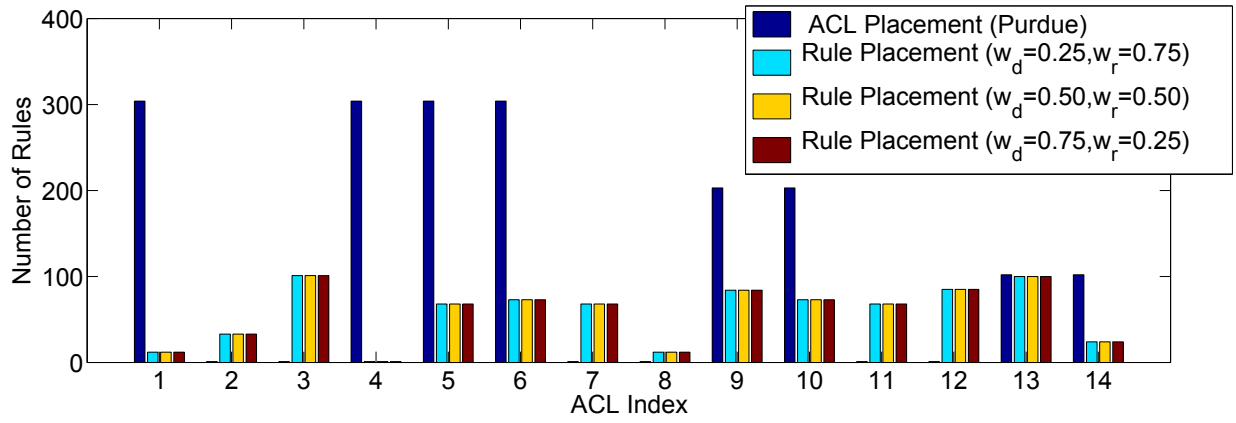


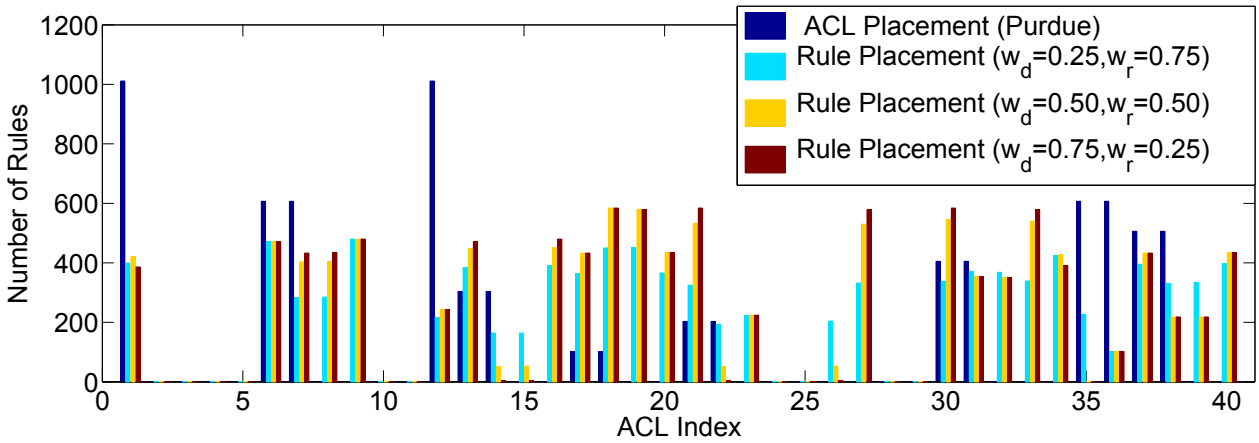
Figure 10.12: Three network topology for three test cases

ACL placement does not support multiple strategy as the time. Hence, we chose MIN, the first strategy they have introduced in [SRXM08]. The experimental results is shown in figure 10.13. In this figure we show the distribution of rules by sketching the ACL size for each classifier in figures 10.13(a), (b), and (c) for test case 1, 2, and 3, respectively. For each test case we compare the rule placement with three different configuration set up for weights together along with the Purdue ACL replacement with MIN strategy. The three configurations for rule placement first put more weights on load balancing rather than early packet discard by having $w_d = 0.25$ and $w_r = 0.75$. Second, we have same weights for load balancing and early packet discard ($w_d = 0.75$ and $w_r = 0.25$) Third, on the contrary to the first configuration, we put weights on early packet discard rather than load balancing by having $w_d = 0.75$ and $w_r = 0.25$.

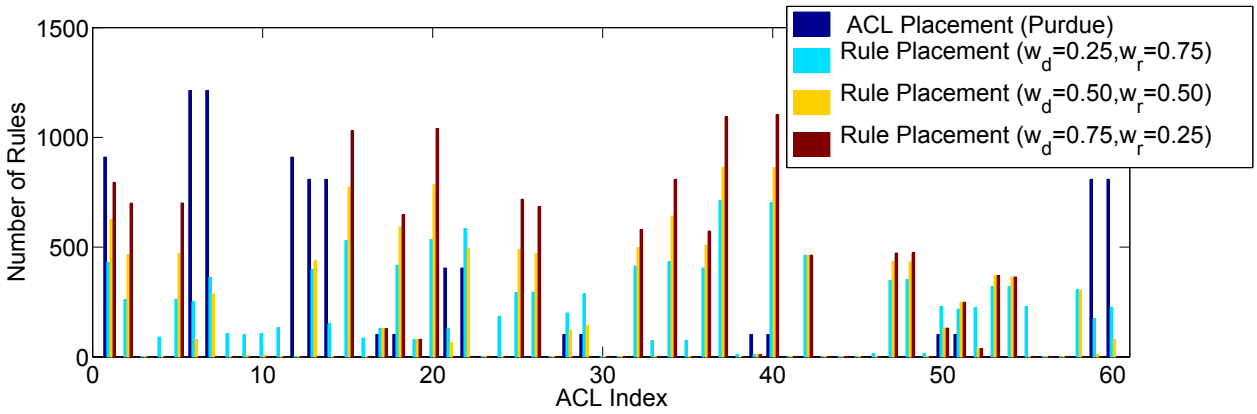
The results for Test case 1 shown in figure 10.13(a) indicates that out of 14 classifiers for



(a) Test Case 1



(b) Test Case 2



(c) Test Case 3

Figure 10.13: The rule distribution on the three test cases

access policy placement, 13 classifiers are used for rule placement using our rule placement algorithm, whereas only 7 classifiers is used for ACL placement algorithm. However, the

maximum and the minimum number of rules in rule placement (with all three configuration) is 100 and 12 rules, respectively, while for the ACL placement it is 304 and 102 rules.

The results for Test case 2 shown in figure 10.13(b) indicates that out of 40 classifiers for access policy placement, 30/29/29 classifiers are used for rule placement using our rule placement algorithm (for three different configurations), whereas only 16 classifiers is used for ACL placement algorithm. The maximum and the minimum number of rules in rule placement is 480/584/584 and 102/51/4 rules, for three different configurations respectively, while for ACL placement it is 1011 and 102 rules.

Finally, the results for Test case 3 shown in figure 10.13(c) indicates that out of 60 classifiers for access policy placement, 48/36/24 classifiers are used for rule placement using our rule placement algorithm (for three different configurations), whereas only 18 classifiers is used for ACL placement algorithm. The maximum and the minimum number of rules in rule placement is 711/848/1,103 and 11/8/11 rules, for three different configurations respectively, while for ACL placement it is 1,213 and 102 rules.

Looking at the results we can draw following conclusions. (1) Using rule placement, we can have a closer minimum and maximum values, which indicates that we use our resources more evenly and efficiently if we use rule placement. Table 10.4 shows the standard deviation of the number of the rules on the classifiers for the test cases of ACL placement and rule placement with three different configuration. The standard deviation results supports our initial observation and indicates that using rule placement the classifiers can share the load of access policy enforcement among the classifier more evenly. Moreover, comparing the results for different configurations of the rule placement, the more weight we have for w_r the rules, the less standard deviation and the closer the minimum and the maximum number of rules per classifier. This is expected as w_r is used to leverage load balancing for rule placement.

	Test Case 1	Test Case 2	Test Case 3
ACL Placement (Purdue Algorithm)	133.925	287.996	325.024
Rule Placement ($w_d = 0.25$,$w_r = 0.75$)	33.994	166.174	188.916
Rule Placement ($w_d = 0.50$,$w_r = 0.50$)	33.994	217.921	265.307
Rule Placement ($w_d = 0.75$,$w_r = 0.25$)	33.994	230.178	344.032

Table 10.4: Standard Deviation of the number of the rules

(2) On average, rule placement uses more classifiers to place rules than ACL placement. Table 10.5 shows the number of classifiers, which were used for access policy enforcement using rule and ACL placement algorithms. The results clearly show that using ACL placement, less number of classifier are used for enforcing access policy. It also indicates that the larger the value for w_r, the more number of classifier are used. This corresponds with the intent of using w_r for load balancing between classifiers.

	Test Case 1	Test Case 2	Test Case 3
ACL Placement (Purdue Algorithm)	57.143%	40%	30%
Rule Placement ($w_d = 0.25$,$w_r = 0.75$)	92.857%	75%	80%
Rule Placement ($w_d = 0.50$,$w_r = 0.50$)	92.857%	72.5%	60%
Rule Placement ($w_d = 0.75$,$w_r = 0.25$)	92.857%	72.5%	40%

Table 10.5: Percentage of the classifiers used (*i.e.* their ACLs have more then 1 rule)

(3) Rule placement induces less number of forwardings for unwanted traffic. Table 10.6 shows the average number of forwardings for unwanted traffic. The results for the rule placement (with all configuration) is better than ACL placement. Moreover, the larger the value for w_d is, the less average number of forwardings for unwanted traffic. Again, this is expected because we use w_d to leverage early unwanted packet discarding.

(4) Rule placement often creates more number of rules than ACL placement. Table 10.7 shows the sum of the number of rules for ACL placement and rule placement. Except test case 1, other test cases shows that rule placement creates more number of rules. The

	Test Case 1	Test Case 2	Test Case 3
ACL Placement (Purdue Algorithm)	1	3.48551	2.45625
Rule Placement ($w_d = 0.25$,$w_r = 0.75$)	0.568584	2.87783	1.10333
Rule Placement ($w_d = 0.50$,$w_r = 0.50$)	0.568584	2.5528	0.74691
Rule Placement ($w_d = 0.75$,$w_r = 0.25$)	0.568584	2.51476	0.529298

Table 10.6: Average number of unwanted packet forwardings

reason is twofold. First, the ACL heuristic is with MIN strategy; thus, it is designed to have least possible number of rules. Second, the number of non-overlapping rules that represent a network access policies can be more than overlapping rules that represent same access policy. Even after compression, because a set of non-overlapping rules can be hardly compressed, the total number of rules is often larger for rule placement.

	Test Case 1	Test Case 2	Test Case 3
ACL Placement (Purdue Algorithm)	1,832	7,514	9,150
Rule Placement ($w_d = 0.25$,$w_r = 0.75$)	802	9,778	12,693
Rule Placement ($w_d = 0.50$,$w_r = 0.50$)	802	10,478	12,799
Rule Placement ($w_d = 0.75$,$w_r = 0.25$)	802	10,504	13,290

Table 10.7: Sum of the total number of rules

10.11 Rule Placement Practical Pros and Cons

Evaluation results show that the rule placement can be effectively used in practice and can create a tradeoff between all design requirements and it seems to be able to distribute rule across classifiers optimally. However, rule placement for some central access policy may not result better than ACL placement. This is because that in rule placement the cental policies are represented in non-overlapping discard rules. Hence, in theory, depending on the access policy, the number of non-overlapping discard rules can be very large. The large number of discard rules not only causes a long algorithm execution time (comparing to ACL placement),

but also increases total number of rules that must be placed on the classifiers and makes the ultimate results worse than ACL placement results. In practice, however, we believe rule placement can be used because network operators typically defined the set of access restrictions and the rest of the traffic will be accepted. In such cases the rule placement can be very helpful, as the set of restrictions limited to reasonable number of non-overlapping discard rules and can be efficiently placed on the classifiers.

Finally, we compare the execution time for ACL placement and rule placement in table 10.8. The results indicate that the ACL placement takes significantly less time for execution as it was expected. The rule placement execution time linearly increases as the number of non-overlapping discard rules increases. However, this operation is going to be done once and it is in the order of seconds and minutes, which certainly satisfies network operator requirements in practice.

	Test Case 1	Test Case 2	Test Case 3
ACL Placement (Purdue Algorithm)	0.035s	0.638s	1.173s
Rule Placement	2.472s	43.339s	63.427s

Table 10.8: Execution time

Chapter 11

Related Work and State-of-the-Art

11.1 Active Probing

Active probing tools, which actively test network reachability by sending probing packets and analyzing the response packets, are commonly used by network operators. Such tools include `ping` and `traceroute`, which use ICMP [Pos81] echo request/reply or ICMP time-exceed packets, to test whether a host on a target network is reachable. The use of such tools is limited because they cannot verify the reachability of UDP or TCP packets. There are tools such as NMAP [Fay] and NESSUS [nes] that can test the reachability of UDP or TCP packets; however, such tools have significant limitations. First, they cannot perform comprehensive testing due to the amount of packets that have to be generated. Second, the test results of such tools are valid only for the routing state at the time that the testing is performed and may not hold afterwards due to the change of routing states over time. Third, such tools only show the open ports on which a server daemon is listening and does not reveal the open ports with no server listening on them at the time of testing. In comparison, our Quarnet is non-intrusive and comprehensive. But of course, active probing tools have some

benefits that Quarnet does not offer. For example, they may identify reachability faults, such as errors in routing software, which are not caused by ACL misconfigurations. Nevertheless, our tool is complementary to such tools.

11.2 Network Reachability Analysis

Little work has been done on network reachability analysis. Xie *et al.* presented a model of network reachability in their seminal work [XZM+05]; however, they give no algorithms for computing reachability (and of course no experimental results). Xie *et al.*'s network reachability model does not address IP tunneling, dynamic NAT and PAT, and does not take into account whether transport layer protocols are connectionless or connection-oriented. Furthermore, Xie *et al.*'s model is limited to describing the networks where each subnet connects to only one router because they model a network as a graph of routers. We significantly go beyond Xie *et al.*'s work along three dimensions: (1) reachability modeling and formulation, (2) algorithms for computing reachability, (3) solutions for reachability queries. Our model differs from Xie *et al.*'s work in the following aspects. First, we model a network as a graph over both routers and subnets, while, as mentioned, Xie *et al.* model a network as a graph over only routers. Thus, Xie *et al.*'s model is limited to describing the networks where each subnet connects to only one router, while our model does not have this limitation. Furthermore, we calculate reachability between two subnets and they calculate reachability between two middleboxes. Second, we distinguish network reachability formulations based on both the properties of transport layer protocols (namely connectionless and connection-oriented protocols) and the statefulness of routers/firewalls on every path, while Xie *et al.* did not. Third, our model addresses IP tunneling and three types of packet transformations: static

NAT, dynamic NAT, and PAT, while Xie *et al.*'s model addresses only static NAT. Xie *et al.* gave no implementable algorithms for computing reachability and no solutions for reachability queries. Recently, in a poster paper [ZNW08], Zhang *et al.* proposed monitoring and verifying reachability in real-time by computing instantaneous reachability. However, they provided no algorithm for computing the reachability from a source to a destination along a given path and no experimental results. Furthermore, they have the other limitations of Xie *et al.*'s work mentioned above.

Some effort has been made to use Binary Decision Diagrams (BDDs) for reachability analysis [ASMEAE09, SLL+09]; however, such work can only answer host-to-host queries and cannot answer subnet-to-subnet queries as we do in this dissertation. We choose FDDs instead of BDDs, as the basic data structure in this dissertation for several reasons. First, it is extremely difficult, if not infeasible, to swap the values of packet fields or rearrange packet fields in a BDD calculated over multiple ACLs. These operations are critical for computing reachability for connection-oriented protocols and networks with NAT/PAT. Note that our methods for performing these operations on FDDs, which requires generating rules from FDDs and reconstructing FDDs, cannot be applied to BDDs because generating rules from a BDD could easily lead to millions of rules as reported in [LG08]. Second, a reachability query engine built with BDDs can only process closed queries that demand a *yes*/*no* answer. In contrast, our reachability query engine built with FDDs can process both closed queries and open queries. Third, the reachability calculated by BDDs is not human readable. In contrast, every element in our reachability matrix is an FDD, which can be visualized for examination for humans to examine.

11.3 Complementary Work on Network Reachability

In [MWZ00], Mayer *et al.* proposed Fang, a firewall analysis engine. Fang supports limited queries (over 3-tuples) and each query is compared with every rule in every ACL along every path from a source to a destination, which is very inefficient.

There is some work, which is orthogonal to ours, on detecting reachability problems caused by routing faults (*e.g.*, the faults identified in [Pax97]), instead of ACL misconfiguration (*e.g.*, [MWA02, SG04, ABKM01, DTDD07, MBGR06]). In [ILP06], Ingols *et al.* proposed algorithms for creating attack graphs for a network; however, their focus is not on reachability computation. Our proposed work is complementary to [ILP06].

11.4 Reachability Troubleshooting

To the best of our knowledge, there is no prior work on network reachability error troubleshooting. Yet, the closest thread of related work is on firewalls and network ACLs error detection and removal. Such errors include rule redundancy and shadowing, among rules in a single firewall policy or among ACLs of an interconnected network. As a very first work in the firewall error detection thread of research, Hari *et al.* discussed overlapping filters (*i.e.* rules) in a firewall policy and its possible conflicts that can cause security holes in the firewall policy. Considering classifiers with n rules, they also provide trie-based algorithms with time complexity of $O(n^2)$ for general classifiers to detect and remove such conflicts [HSP00]. Eppstein and Muthukrishnan proposed rectangle geometry based algorithms to improve rule conflict detection and removal in time complexity of $O(n^{3/2})$ [EM01]. As the proposed algorithms were not fast enough for 5-tuple packet classifiers, Baboescu and Varghese proposed a

faster trie-based algorithm for conflict detection and removal [BV02]. A couple of years later, Gouda and Liu proposed FDD and related algorithms to design a firewall such that it satisfies certain properties that includes no rule conflicts, no rule redundancy and shadowing [GL04]. More recent work though focuses on rule conflicts on ACLs in a network. Al-Shaer *et al.* use state diagrams for inter-firewall anomaly detection and removal. They can also detect and remove rule redundancies and shadowing among different firewalls in a network [ASHBH05]. Similarly, Alfaro *et al.* proposed a network model and based on that they presented some algorithms to detect and remove rule conflicts in a network [GACC06]. Fireman [YCM$^+$06] is another similar tool that uses symbolic model checking to find firewalls misconfigurations and inconsistencies.

11.5 Reachability Optimization

The most related work on network ACL design is introduced by Sung *et al.* in [SRXM08], in which they followed a "top-down" approach for systematic design of enterprise networks. Their work includs a suite of algorithms for access policy placement on network classifiers so that the reachability is limited to the network central access policy, while it satisfies network performance criteria. The basic idea in this work is to interpret the reachability between two given subnets as an ACL and then place this ACL on all the paths that connect these two subnets together. They recognized the performance criteria as "strategies" in ACL design, and they proved that for each strategy, the ACL placement problem is NP-hard. Accordingly, they proposed greedy heuristics based on "first fit deceasing" technique. The heuristic for all strategies are the same, however for each strategy, they changed the order of ACL placement so that the final results complies with the strategy. Although their heuristics are straight-

forward, they have not shown how multiple (or all) strategies can be taken into account simultaneously. As mentioned in section 8.2.1, Quarnet follows a different approach (*i.e.* rule placement) to solve this problem. Using this approach, we can consider all the network performance criteria at the same time and create a balance between them by introducing a weight for each criterion.

Sun *et al.* in [SR11] focused on network *redesign* and accordingly they considered its associated costs. In their work, the authors aimed to resolve network dependencies problems that can occur when a network is redesigned. They presented a model for VLAN redesign costs such as access policies reconfiguration induced by changes in VLAN design. They later presented algorithms for two basic functions `move()` and `merge()`, using which the VLAN redesign is feasible. Although Quarnet can apply algorithms for incremental changes, this work can be used in parallel with Quarnet to handle the configuration inconsistencies caused by VLAN configuration changes.

Another genre of related work focused on the networks with centralized control using OpenFlow [MAB+08] and flow switches. In such networks, the routing and switching is centralized by OpenFlow controllers. The OpenFlow controllers control all the switches and routers (so-called flow switches) in the network to forward packets from a source to a destination. Access policy enforcement in OpenFlow networks is easy as the classifiers can be installed on the controllers and decide whether a packet can be forwarded from a source to a destination. The problem with such frameworks is the scalability. Such controllers cannot scale well when the number of flows in the network is large. Yu *et al.* presented DIFANE [YRFW10], in which they addressed this issue. Their approach was to move the central access policy enforcement from the control plane to data plane using widely distributed *authority switches*. In this framework, rules are distributed among authority

switches, so that they are enforced on the traffic. When a packet is received by a flow switch, it checks if it has any rule that match the packet, if it does not have any rule, it forwards the rule to the authority switches. The authority switches find a match for the packet and send the cached rule to the flow switch. Accordingly, the rest of the packets can be classified based on the cached rule in the data plane. Their experiments indicated lower delay, higher throughput, and better scalability comparing to directing packets through a separate controller.

Another close work for OpenFlow networks is the Ethane controller introduced by Casado *et al.* in [CFP+09]. In this controller, flow-level rules are installed reactively for the first packet of each TCP and UDP flow. While the OpenFlow networks received a great deal of attention in last few years, OpenFlow networks are still in the research phase and has not been widely deployed in practice. They are only partially deployed on a few university campus networks. Hence, it will take a long time for such networks to be regularly used in practice.

Chapter 12

Conclusions

12.1 Summary and Contributions

In this dissertation, we first model and formulate network reachability considering the differences in connectionless and connection-oriented transport protocols, stateless and stateful middleboxes, as well as the presence and absence of various packet transformers. We also define the concept of probabilistic reachability calculation, in which we only calculate reachability for the paths that are likely to be chosen by the routing daemon to dramatically save time and memory for reachability calculation. Second, we present algorithms for computing network reachability matrices using Firewall Decision Diagrams (FDDs). Third, we give solutions for expressing and processing reachability queries in a SQL-like language called SRQL. Forth, for network reachability troubleshooting, we formulate two types of network reachability errors. Type 1 errors disrupt legitimate access to network resources and type 2 errors facilitate unauthorized access to network resources. Given a reachability error, we propose a suite of algorithms for detecting faulty ACLs and fixing them in an efficient way. We also use network reachability inconsistencies to find reachability errors proactively and

fix them before they cause any network reachability problem. Fifth, we propose a suite of algorithms for optimized ACL placement in the network so that it satisfies the network central access policy and complies with the network performance criteria. The network central access policy can be calculated from the current network configurations and the network ACLs can be redesigned error free and optimized (based on network performance criteria) and be pushed out to all the classifiers on the middleboxes.

We implemented our algorithms and conducted experiments on a campus network and other real-life network topology. Results show that our offline reachability computation is practical and online query processing is very efficient. Moreover, our experimental results on reachability troubleshooting show that our algorithms can detect and fix faulty ACLs in the order of milliseconds. Yet, the time for error detection and troubleshooting increases if the network ACL average number of rules in the increases. Similarly, the experimental results indicate that network reachability inconsistencies can be detected in the order of minutes, but it may increase if the number of network paths increases. We also implemented our rule placement algorithm and compared it with ACL placement algorithm [SRXM08]. We show that on some generic network test cases with synthesized central access policies, our algorithm yields better and more even rule placement on the classifiers.

12.2 Future Work

12.2.1 Parallel Reachability Computation

Different elements of reachability matrix computation can be parallelized to improve the matrix computation time. Several pieces of work has been proposed to parallelize dynamic

programming [TSG, SSdlB+10]. As in dynamic programming, a problem is recursively divided into subproblems, and optimal solutions are assembled from optimal subsolutions, we can fork the solver for each subsolution so that each subsolution can be computed in parallel. Yet, on each step of recursion, we need to cache the subsolutions, and lock on them to protect the individual solutions from recomputing.

Recently, Graphics Processing Units (GPUs) received significant attention for parallelizing algorithms and make them faster. As a future work, the reachability computation algorithm can be redesigned such that the FDD operations are computed in parallel for each element of the reachability matrix.

12.2.2 Rule Cache and Forward

One way to further minimize the number of forwardings, we can adapt the rule placement to the actual traffic dynamically by keep tracking the number of packet match for each rule and cache the rule close to the packet source. Caching can be proffered over rule expulsion because of its simplicity, although the total number of rules may not be optimal any more and it is probable that we can have similar rules along a path. One possible application of such rule caching is when the enterprise network is connected to the Internet, and one of its subnets are under attack. In such situations, the rule that matches the traffic can be placed on the network gateway firewall to protect the subnet from the attack traffic on the gateway. This avoids the attack's collateral damages to the rest of the subnets in the network. To have such rule caching mechanism, the pattern of the unwanted packet must be studied so that we can evaluate the effectiveness of this technique and use this study to design a smart cache replacement policy. Assuming the unwanted packets has a certain

pattern (excluding the time that the network is under Denial of Service (DoS) attacks), rule caching can trigger rule expulsion predicably so that we keep the optimality of the number of rules and the minimum unwanted packet forwardings. Note that the caching rule is not necessarily a non-overlapping discard rule that is generated by the FDD. This rule can be generated based on unwanted traffic pattern and exported along the path so that it is placed on a classifier closer to the source.

12.2.3 Optimization with Compression

The current cost-based rule placement algorithm is independent from the ACL compression. This makes the ACL compression techniques less effective. However, if we adapt the rule placement and edge cut selection to the ACL compression such that we can have a better compression rates, we have one more step further to minimize the number of rules per ACL. Moreover, for prefix-based classifiers, we can further change the algorithm so that the final set of rules has the minimum number of prefix rules. Indeed, such techniques can be different for software and hardware classifiers.

APPENDIX

APPENDIX A: Summary of Important Notations

$\mathcal{N}$ ..... set of subnets

n ..... number of subnets

$\mathcal{N}$ ..... set of subnets

n ..... number of subnets

$\mathcal{R}$ ..... set of middleboxes

m ..... number of middleboxes

u ..... max number of unidirectional interfaces on a middlebox

$\mathcal{I}$ ..... network incident matrix

$\mathcal{A}[i,j]$ ..... ACL of j-th interface of middlebox i

p ..... number of the paths in a network

s ..... routing state

$\mathcal{S}$ ..... set of routing states

$\mathcal{S}^e$ ..... set of effective routing states

$P_{i,j}(s)$ ..... path from N_i to N_j at routing state s

M ..... number of hops on a path

C_k ..... k-th classifier in a path from N_i to N_j

$A(C_k)$ ..... packets accepted by ACL C_k

z ..... max number of paths between any two subnets

R ..... reachability set

$\overrightarrow{R}$ ..... reachability set for data path

$\overleftarrow{R}$ ..... reachability set for signaling path

l ..... ACL

ℓ ..... link

F_i ..... i-th field in FDD

F ..... FDD matrix

f ..... Firewall Decision Diagram (FDD)

P ..... path matrix

A ..... network computed reachability FDD matrix

$\hat{A}$ ..... network computed probabilistic reachability FDD matrix

fr ... FDD for the set of packets that can be accepted by the ACLs on the private subpath

fu ... FDD for the set of packets that can be accepted by the ACLs on the public subpath

T_S ..... transformation function on source fields

T_D ..... transformation function on destination fields

g ..... max number of rules in an ACL

N_i ..... subnet i

d ..... number of fields in each rule

$\hat{R}$ ..... effective reachability

$P^e_{i,j}$ ..... set of effective paths from N_i to N_j

$\bar{P}$ ..... sorted sequence of path

h ..... number of links in the network

Υ ..... network uptime

M ..... number of hops on a path

ϵ ..... reachability uptime

N_i ..... subnet i

$P_{i,j}$ ..... paths from N_i to N_j

F_i ..... i-th field in FDD

C_k ..... k-th classifier in a path

$A(C_k)$ ..... packets accepted by classifier C_k

f_k ..... FDD representing $A(C_k)$

p_a ..... a-th path

La ..... set of link in a-th path

w_a ..... total weight for a-th path

ℓ_b ..... b-th link

π_a ..... the probability for a-th path

ϕ_i ..... i-th decision path in an FDD

$\mathbb{F}(i,j)$ ..... FDD for the network central access policy between N_i and N_j

$\mathcal{E}_1$ ..... type 1 errors

$\mathcal{E}_2$ ..... type 2 errors

$\Phi_{\mathcal{E}}$ ..... set of decision paths that match error $\mathcal{E}$

$\mathbb{C}$ ..... fixing and rule placement cost function

BIBLIOGRAPHY

BIBLIOGRAPHY

[ABKM01] David Andersen, Hari Balakrishnan, Frans Kaashoek, and Robert Morris. Resilient overlay networks. In *Proceedings of 18th ACM Symposium on Operating System Principles (SOSP)*, 2001.

[And06] Oskar Andreasson. IPtables tutorial 1.2.2. *http://iptables-tutorial.frozentux.net/iptables-tutorial.html*, 2006.

[ASHBH05] Ehab Al-Shaer, Hazem Hamed, Raouf Boutaba, and Masum Hasan. Conflict classification and analysis of distributed firewall policies. *IEEE Journal on Selected Areas in Communications (JSAC)*, 23(10):2069–2084, 2005.

[ASMEAE09] Ehab Al-Shaer, Will Marrero, Adel El-Atawy, and Khalid ElBadawi. Network configuration in a box: Towards end-to-end verification of network reachability and security. In *Proceedings of IEEE International Conference of of Network Protocols (ICNP)*, 2009.

[BV02] Florin Baboescu and George Varghese. Fast and scalable conflict detection for packet classifiers. In *Proceedings of 10th IEEE International Conference of on Network Protocols (ICNP)*, 2002.

[CFP+09] Martin Casado, Michael J. Freedman, Justin Pettit, Jianying Luo, Natasha Gude, Nick McKeown, and Scott Shenker. Rethinking enterprise network control. *IEEE/ACM Transactions on Networking (ToN)*, 17:1270–1283, 2009.

[cis] Internetworking design basics. *http://www.cisco.com/en/US/docs/internet working/design/guide/nd2002.html*.

[cis05] NAT order of operation. *http://www.cisco.com/warp/public/556/5.html*, 2005.

[DTDD07] Amogh Dhamdhere, Renata Teixeira, Constantine Dovrolis, and Christophe Diot. NetDiagnoser: troubleshooting network unreachabilities using end-to-end probes and routing data. In *Proceedings of ACM International Conference of on Emerging Networking EXperiments and Technologies (CoNEXT)*, 2007.

[EM01] David Eppstein and S. Muthukrishnan. Internet packet filter management and rectangle geometry. In *Symp. on Discrete Algorithms*, pages 827–835, 2001.

[Fay] Faydor. Nmap: Free network security scanner. http://nmap.org.

[FLHDM00] D. Farinacci, T. Li, S. Hanks, and and P. Traina D. Meyer. Generic routing encapsulation (GRE). *RFC 2784*, 2000.

[GACC06] Joaquin Garcia-Alfaro, Frederic Cuppens, and Nora Cuppens. Analysis of policy anomalies on distributed network security setups. In *Proceedings of 11th European Symposium On Research In Computer Security (ESORICS)*, September 2006.

[GL04] Mohamed G. Gouda and Alex X. Liu. Firewall design: consistency, completeness and compactness. In *Proceedings of 24th IEEE International Conference of on Distributed Computing Systems (ICDCS)*, pages 320–327, March 2004.

[GL07] Mohamed G. Gouda and Alex X. Liu. Structured firewall design. *Computer Networks Journal (Elsevier)*, 51(4):1106–1120, March 2007.

[hao]

[HSP00] Adiseshu Hari, Subhash Suri, and Guru M. Parulkar. Detecting and resolving packet filter conflicts. In *Proceedings of IEEE INFOCOM*, pages 1203–1212, 2000.

[ILP06] Kyle Ingols, Richard Lippmann, and Keith Piwowarski. Practical attack graph generation for network defense. In *Proceedings of 22nd IEEE Annual Computer Security Applications Conference (ACSAC)*, pages 121–130, 2006.

[Ker04] Zeus Kerravala. As the value of enterprise networks escalates, so does the need for configuration management. Enterprise Computing & Networking, The Yankee Group Report, January 2004.

[KL10] Amir R. Khakpour and Alex X. Liu. Quantifying and querying network reachability. In *Proceedings of International Conference of on Distributed Computing Systems (ICDCS)*, 2010.

[KSK+05] Andreas Klenk, Philipp Schlicker, Ralph Kuhne, Ali Fessi, Changpeng Fan, Falko Dressler, and Georg Carle. Path coupled accounting mechanisms for all IP networks. In *6th IEE International Conference of on 3G & Beyond (3G 2005)*, 2005.

[LG08] Alex X. Liu and Mohamed G. Gouda. Diverse firewall design. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 19(8), 2008.

[LG09] Alex X. Liu and Mohamed G.Gouda. Firewall policy queries. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 20(6):766–777, 2009.

[Liuar] Alex X. Liu. Firewall policy verification and troubleshooting. *Computer Networks*, to appear.

[LMTar] Alex X. Liu, Chad R. Meiners, and Eric Torng. Tcam razor: A systematic approach towards minimizing packet classifiers in tcams. *IEEE/ACM Transactions on Networking*, to appear.

[LMZ08] Alex X. Liu, Chad R. Meiners, and Yun Zhou. All-match based complete redundancy removal for packet classifiers in TCAMs. In *Proceedings of 27th Annual IEEE Conference of on Computer Communications (Infocom)*, April 2008.

[LTM08] Alex X. Liu, Eric Torng, and Chad Meiners. Firewall compressor: An algorithm for minimizing firewall policies. In *Proceedings of INFOCOM*, Phoenix, Arizona, April 2008.

[LTM11] Alex X. Liu, Eric Torng, and Chad R. Meiners. Compressing network access control lists. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 22:1969–1977, 2011.

[MAB+08] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computter Communication Review*, 38:69–74, 2008.

[MBGR06] Z. Morley Mao, Randy Bush, Timothy G. Griffin, and Matthew Roughan. BGP beacons. In *Proceedings of 3rd ACM SIGCOMM conference on Internet measurement (IMC)*, 2006.

[MLT07] Chad R. Meiners, Alex X. Liu, and Eric Torng. TCAM Razor: A systematic approach towards minimizing packet classifiers in TCAMs. In *Proceedings of 15th IEEE Conference of on Network Protocols (ICNP)*, pages 266–275, October 2007.

[MWA02] Ratul Mahajan, David Wetherall, and Tom Anderson. Understanding BGP misconfiguration. In *Proceedings of SIGCOMM*, 2002.

[MWZ00] Alain Mayer, Avishai Wool, and Elisha Ziskind. Fang: A firewall analysis engine. In *Proceedings of IEEE Symposium on Security and Privacy*, pages 177–187, 2000.

[nes] Nessus vulnerability scanner. http://www.nessus.org/nessus/.

[OGP03] David Oppenheimer, Archana Ganapathi, and David A. Patterson. Why do internet services fail, and what can be done about it? In *Proceedings of 4th USENIX Symposium on Internet Technologies and Systems (USITS)*, March 2003.

[Pax97] Vern Paxson. End-to-end routing behavior in the internet. *IEEE/ACM Transactions on Networking*, 5:601–615, 1997.

[PIX09] Configuring the PIX firewall. http://www.cisco.com/en/US/docs/security/pix/pix42/configuration/guide/Pix42cfg.pdf, 2009.

[por] Port knocking: A stealthy system for network authentication across closed ports. http://portknocking.org/.

[Pos81] J. Postel. Internet control message protocol. *RFC 792*, 1981.

[Ros07] Shekdon M. Ross. *Introduction to Probability Models*. Elsevier, 9 edition, 2007.

[rul] Firewall throughput test. http://www.hipac.org/performance_tests/results.html.

[SG04] Aman Shaikh and Albert Greenberg. Ospf monitoring: Architecture, design, and deployment experience. In *Proceedings of USENIX Networked Systems Design and Implementation (NSDI)*, 2004.

[Sid09] Sidewinder g2 startup guide. http://www.securecomputing.com/techpubs_download.cfm?id=1642, 2009.

[SLL+09] Yu-Wei Eric Sung, Carsten Lund, Mark Lyn, Sanjay G. Rao, and Subhabrata Sen. Modeling and understanding end-to-end class of service policies in operational networks. In *ACM SIGCOMM*, 2009.

[SR11] Xin Sun and Sanjay Rao. A cost-benefit framework for judicious enterprise network redesign. In *Proceedings of IEEE INFOCOM (Mini-Conference)*, 2011.

[SRXM08] Yu-Wei Eric Sung, Sanjay Rao, Geoffrey Xie, and David Maltz. Towards systematic design of enterprise networks. In *Proceedings of ACM CoNEXT*, 2008.

[SSdlB+10] Alex Stivala, Peter J. Stuckey, Maria Garcia de la Banda, Manuel Hermenegildo, and Anthony Wirth. Lock-free parallel dynamic programming. *Journal of Parallel and Distributed Computing*, 70:839 – 848, 2010.

[STA05] M. Stiemerling, H. Tschofenig, and C. Aoun. Nat/firewall nsis signaling layer protocol (nslp). *draft-ietf-nsis-nslp-natfw-05*, 2005.

[TSG] Guangming Tan, Ninghui Sun, and Guang R. Gao. A parallel dynamic programming algorithm on a multi-core architecture. In *Proceedings of ACM Symposium on Parallel Algorithms and Architectures (SPAA)) year = 2007, owner = Amir, timestamp = 2011.11.13.*

[TVR+99] W. Townsley, A. Valencia, A. Rubens, G. Pall, G. Zorn, and B. Palter. Layer Two Tunneling Protocol (L2TP). *RFC 2661*, 1999.

[Woo04] Avishai Wool. A quantitative study of firewall configuration errors. *IEEE Computer*, 37(6):62–67, 2004.

[XZM+05] G.G. Xie, Jibin Zhan, D.A. Maltz, Hui Zhang, A. Greenberg, G. Hjalmtysson, and J. Rexford. On static reachability analysis of IP networks. *Proceedings of IEEE INFOCOM*, 3:2170–2183, March 2005.

[YCM+06] Lihua Yuan, Hao Chen, Jianning Mai, Chen-Nee Chuah, Zhendong Su, and Prasant Mohapatra. Fireman: a toolkit for firewall modeling and analysis. In *IEEE Symposium on Security and Privacy*, May 2006.

[YRFW10] Minlan Yu, Jennifer Rexford, Michael J. Freedman, and Jia Wang. Scalable flow-based networking with DIFANE. In *Proceedings of of ACM SIGCOMM*, 2010.

[ZNW08] Bo Zhang, T. S. Eugene Ng, and Guohui Wang. Reachability monitoring and verification in enterprise networks. In *Proceedings of SIGCOMM*, 2008.