Security Policy Definition and Enforcement in Distributed Systems

Hang Zhao

Submitted in partial fulfillment of the
requirements for the degree
of Doctor of Philosophy
in the Graduate School of Arts and Sciences

COLUMBIA UNIVERSITY

2012

ABSTRACT

Security Policy Definition and Enforcement in Distributed Systems

Hang Zhao

Security in computer systems is concerned with protecting resources from unauthorized access while ensuring legitimate requests can be satisfied all the time. The recent growth of computer systems both in scale and complexity poses tremendous management challenges. Policy-based systems management is a very promising solution in this scenario. It allows the separation of the rules that govern the behavior choices of a system from the provided functionality, and can be adapted to handle a large number of system elements. In the past two decades there have been many advances in the field of policy research. Although existing solutions in centralized systems are well-established, they do not work nearly as well in distributed environments because of scalability, network partitions, and the heterogeneity of the endpoints.

This dissertation contributes to this endeavor by proposing three novel techniques to address the problem of security policy definition and enforcement in large-scale distributed systems. To correctly enforce service and security requirements from users who have no intimate knowledge of the underlying systems, we introduce the first *distributed policy refinement* solution that translates high-level policies into low-level implementable rules, for which the syntax and semantics can be fully interpreted by individual enforcement points. Taking advantage of both the centralized and end-to-end enforcement approaches, we propose a novel *policy algebra framework* for policy delegation, composition and analysis. As a concrete instantiation of policy delegation enabled by the algebraic framework, we invent a novel firewall system, called ROFL (ROuting as the firewall layer), that implements packet filtering using the underlying routing techniques. ROFL implements a form of *ubiquitous enforcement*, and is able to drop malicious packets closer to their origins to save transmission bandwidth and battery power, especially for resource-limited devices in mobile ad hoc networks (MANET). The correctness and consistency of ROFL can be verified using policy algebra. It provides formalisms to address the complexity of distributed environments, increase assurance and show how to tune tradeoffs and improve security with ubiquitous enforcement.

To demonstrate the effectiveness and efficiency of ROFL as a high-performance firewall mechanism, we analyze its performance quantitatively and conduct experiments in a simulated environment with two ad-hoc routing protocols. Empirical study shows that the increase in traffic for handling ROFL routing messages is more than outweighed by the savings by early drops of unwanted traffic.

Table of Contents

1 Introduction **1**

1.1 Problem Description 1

1.2 Network Access Control 3

1.2.1 Firewalls 4

1.3 Requirements 5

1.4 Hypothesis 7

1.5 Contributions 7

1.5.1 What is Not Addressed in this Dissertation 8

1.6 Dissertation Organization 9

I Background **10**

2 Background and Related Work **11**

2.1 Access Control Models 11

2.2 Network Access Control 13

2.3 Policy Based Management 15

2.3.1 Policy Specification 16

2.3.2 Policy Composition and Analysis 17

2.3.3 Policy Refinement 18

2.4 Policy Based Management in Distributed Systems 19

2.4.1 Policy Enforcement in MANETs 20

2.5 Discussion 21

II Solution 22

3 An Algebraic Framework for Distributed Policy Enforcement 23

3.1 An Algebraic Framework for Policy Composition and Delegation 24

3.1.1 Policy Algebra 24

3.1.2 Delegation of Enforcement 25

3.1.3 Cost and Risk Functions 26

3.2 Policy Instantiations 27

3.2.1 An Algebra for Hybrid Firewall Policy 28

3.2.2 An Algebra for Ponder2 Policy 32

3.3 Discussion 39

4 Policy Refinement 40

4.1 Centralized Refinement for Network Service Policies 41

4.1.1 System Overview 42

4.1.2 A Logic-based Abstract Policy Language 47

4.1.3 Centralized Policy Refinement Scheme 51

4.1.4 Policy Updates 58

4.2 Distributed Refinement for Access Control Policies 60

4.2.1 Revisiting Access Control Policies 61

4.2.2 System Overview 62

4.2.3 Distributed Policy Refinement Scheme 70

4.2.4 Policy Conflict Analysis 76

4.3 Discussion 80

III Application 82

5 Distributed Enforcement of Firewall Policies 83

5.1 Firewalls as A Form of Routing 83

5.1.1 Basic ROFL Scheme 84

5.1.2 Client Routes 85

5.1.3 Routing Metric 86
5.1.4 Extensions of ROFL 88
5.2 Policy Routing with ROFL 89
5.2.1 ROFL with Source Prefix Filtering 89
5.2.2 The Transit Node Problem 91
5.2.3 Aggregation Points 91
5.2.4 Refinement of ROFL Policies 93
5.3 Algorithms 96
5.3.1 Route Propagation 97
5.3.2 Packet Forwarding 98
5.3.3 Correctness 99
5.4 Performance Evaluation 102
5.4.1 Analytical Results 102
5.4.2 Protocol Design 104
5.4.3 Simulation Setup 106
5.4.4 Performance of Routing Protocols with RPGM 108
5.4.5 The Impact of Network Density 114
5.5 Discussion 117

6 A Unification of Routing-based Distributed Policy Enforcement 119
6.1 Routing Algebra 120
6.2 Unifying ROFL with Policy Algebra 121
6.3 Routing Properties 122
6.3.1 Definitions and Formalism 123
6.3.2 Protocol Convergence 126
6.3.3 Path Optimality 127
6.4 Security Properties 128
6.4.1 An Algorithm for Calculating Filtering Effect 128
6.4.2 Examples 131
6.5 Discussion 131

IV Conclusions 133

7 Conclusions 134

7.1 Results Summary 134

7.2 Limitations and Future Work 136

V Bibliography 138

Bibliography 139

List of Figures

1.1 Traditional firewalls v.s. distributed firewalls. 4

2.1 Policy based management stack. 15

4.1 A centralized policy server for each policy domain. 43
4.2 A UML class diagram for a subset of the US policy domain. 45
4.3 Grammar for a logic-based abstract policy language. 48
4.4 Quantification for association. 49
4.5 Knowledge database implementation in Prolog. 58
4.6 Refinement rule set implementation in Prolog. 59
4.7 Policy distribution and partial refinement in a distributed policy scenario. 65
4.8 A scenario of distributed enterprise network. 67

5.1 Two subnets connected by both a transit node and a transit net. 92
5.2 Encoding scheme for label value 95
5.3 Algorithms for propagating ROFL service specific advertisements. 97
5.4 Algorithm for packet forwarding. 99
5.5 Performance of pure-AODV and pure-OLSR under RPGM mobility model (no malicious traffic). 109
5.6 Reduction of data packets at destinations using ROFL with AODV. 109
5.7 Reduction of data packet transmission using ROFL with AODV. 110
5.8 Reduction of control packet transmission using ROFL with AODV. 110
5.9 Reduction of data packets at destinations using ROFL with OLSR. 112
5.10 Reduction of data packet transmission using ROFL with OLSR. 112

5.11 Performance of pure-AODV under different network densities. 115

5.12 Performance of AODV with ROFL under different network densities. 116

6.1 A network interpretation for addition (+) and conjunction (&). 121

6.2 Propagation of ROFL policy R along a simple path of three nodes. 122

6.3 Mapping function $\oplus_A$ for programmatic label L_A and signature set Σ_A in the packet filter algebra A. 125

6.4 An example of path selection. 127

6.5 An algorithm for calculating distributed filtering effect. 129

6.6 Examples of calculating distributed filtering effect for ROFL policy R_1, R_2 and R_3 announced by node d under two different network topologies. 130

List of Tables

3.1 Policy combination matrices for operators $+$, &, $-$, $\neg$. 30

3.2 Policy combination matrix for operators $\oplus, \otimes, \ominus, \neg_a, \neg_o$ 34

3.3 Sample policies for a conference reviewing system. 38

4.1 Update of knowledge database upon operations 60

5.1 The number of services running out-of-the-box on default installations of assorted desktop operating systems. The numbers are approximate, since different vendors will ship slightly different configurations. In addition, some ports (i.e., DHCP) are never used off-LAN, and hence would not occupy space in the enterprise routing table. In our tests, about half of the Vista ports were in the Vista client port range ($[2^{14} + 2^{15}, 2^{16} - 1]$); they may thus be covered by our client port scheme. 102

5.2 Simulation setup. 107

5.3 Reduction of data packet transmission using ROFL with AODV, given small amount of malicious traffic. 113

5.4 Reduction of control packet transmission using ROFL with AODV, given small amount of malicious traffic. 113

5.5 Reduction of data packet transmission using ROFL with OLSR, given small amount of malicious traffic. 113

6.1 Precedence, associativity and definition of six algebraic operators. 123

Acknowledgments

The work presented in this thesis would not have been possible without the help and support of many people. First, I would like to express my greatest gratitude to my advisor, Professor Steven Bellovin, for his advice and guidance over the past few years. I would not have been able to complete this work without his support, encouragement and inspiration.

I owe special thanks to Professors Angelos Keromytis, Salvatore Stolfo and Tal Malkin from security research group at Columbia University, for their valuable suggestions and useful discussion on projects I have worked on and presentations I have given at conferences and workshops.

I would like to thank my thesis committees Professors Salvatore Stoflo, Dan Rubenstein, Dr. Jorge Lobo from IBM Thomas J. Watson Research Center and Professor Jonathan Smith from University of Pennsylvania, for their precious time and many value suggestions and comments to improve this thesis.

I also want to thank my fellow students from the Computer Science Department at Columbia University for their support, help and making the PhD study such an enjoyable experience.

Finally, I must thank my parents Ji Han and Jianhua Zhao, and my husband Zhiyang for unconditional support and encouragement throughout this course. I also want to thank my son, baby Anlan, for making me stronger than I ever was.

Chapter 1

Introduction

The growth of computer systems, both in scale and complexity, poses tremendous management challenges. These systems are often interconnected, resulting in a distributed environment with a large number of devices and users, vast amounts of data and resources, and a variety of applications, protocols, and mechanisms. Policy-based systems management is a very promising solution in this scenario. A general *policy* definition adopted in [Sloman, 1994] considers policies as rules governing the behavior choices of a system. The policy-driven approach facilitates the dynamic change of behavior of the distributed management system, while avoiding the burden of recoding system functionality upon changes.

1.1 Problem Description

A very important aspect for any policy-based systems management is to protect managed data and resources against unauthorized access, while ensuring their availability to legitimate users. This process is called *access control* [Samarati and de Capitani di Vimercati, 2001]. Access control is a crucial aspect of a system's security, and provides the basis for all the other mechanisms and procedures the system may utilize. The development of any access control system requires the following two concepts: an *access control policy* that defines high-level rules according to which access control must be regulated, and an *enforcement mechanism* that implements the controls imposed by the policy using software and/or hardware solutions. For example, assume a system administrator decides to permit web service on a machine. This high-level regulation (i.e., the access control policy) is translated into a low-level firewall rule that allows incoming traffic to connect to its port 80, which is then enforced using the packet filtering function provided by a firewall (i.e., the enforcement mechanism).

Given the large number of system elements managed in a distributed environment, the access control mechanism employed must be scalable. The traditional way of dealing with scalability at the human level has been decentralization of management and delegation of authority. Thus it is impossible to maintain a central policy agent for managing all the system devices, which implies the need for integrating and analyzing policies issued by multiple policy authors to ensure that they are always consistent and compliant with the global security requirements. For example, maintaining security policies in dynamic environments, such as mobile networks or virtual environments, where devices in the network or entities in the virtual organization wish to share resources in a controlled way, requires mechanisms to compose and integrate security policies. In another example, coalitions are formed by multiple organizations to carry out joint missions in a military scenario. When information is shared for collaboration, it becomes necessary to combine, compare and contrast the underlying security policies that govern the access to these information. However, the questions of how to understand the interactions between these policies, and how to integrate and compose policies to enforce consistency in the distributed systems, have not yet been adequately investigated.

The aforementioned firewall example demonstrates the importance of a third concept, sometimes optional, the refinement of high-level service and security requirements into low-level implementations. This process is often referred as *policy refinement* [Moffett and Sloman, 1993], which automatically translates high-level goals into low-level rules that can be directly implemented by individual enforcement points. The goal of policy refinement is to generate low-level rules such that their syntax and semantics can be interpreted by the chosen enforcement mechanism. Given a large number of system elements managed in a distributed scenario, it is efficient and scalable to issue global service and security requirements in terms of high-level policies rather than mechanism rules. On the other hand, these high-level requirements are mostly specified by policy makers without an intimate knowledge of the underlying system. Given the diverse applications, services, and numerous objects and devices managed in large-scale distributed systems, it is important to maximize the automation of the refinement process as much as possible. Therefore policy refinement fills the gap between policy specification and enforcement. While the latter two techniques have been studied intensively, only limited work has addressed policy refinement, especially for distributed policy scenarios. It is essential to verify that the lower level rules actually meet the requirements specified by the high-level policy by ensuring that at each stage the translation is correct and consistent. In addition, policy refinement should be interwoven with *conflict analysis* to guarantee that the refined rules are always consistent with existing one.

1.2 Network Access Control

Network access control is concerned with regulating access to protected resource in a communications network that complies with defined security policies. Generally speaking, network access control deals with two levels of protection [Cheswick *et al.*, 2003]:

- *Host-based security* protects the safety of a single host that is connected to a network. Unfortunately, most commercial systems are not bug-free, and some have serious security holes that can be exploited by the attacker. In addition, hosts within the same administrative zone tend to trust each other such that one weak link can compromise the whole cluster of systems.

- *Perimeter security* protects a cluster of hosts using two components: a layer of defense built up around the cluster, called *the wall*, and *the gate* that allows legitimate traffic to pass through while blocking malicious one. This approach often assumes every host behind the wall is trusted.

Generally speaking, a network access control solution unifies a number of mechanisms, including, but not limited to, the following techniques:

- Endpoint security techniques such as antivirus software to prevent, detect and remove malware such as computer virus, worms, Torjan horses, etc.; host-based intrusion detection and prevention systems that monitor system activities to report malicious behavior and policy violation.

- User or system authentication methods such as passwords (something you know), secure devices (something you own), and biometric (something you have).

- Network security enforcement such as firewalls to protect local system from network-based threat through traffic filtering, IPsec protocols to provide end-to-end or end-to-gateway encryption and authentication, etc.

Moreover, there is a growing trend of enforcing access control based on the end-to-end design principle in distributed systems, similar to the IP structure in the communications network. This approach implements access control in a distributed manner by removing potential performance bottleneck to corporate rapidly growing networks, and hence yields better scalability.

1.2.1 Firewalls

Firewalls are an effective means of protecting a local system or network of systems from network-based security threats by examining the traffic passing by. They can be categorized into packet filters, circuit-level gateways, application-level filters and so on [Cheswick *et al.*, 2003]. In this dissertation work, we restrict our discussion to packet filters. Packet filters make forwarding decisions based on the source/destination address and port number of a packet. No traffic state is maintained and thus decisions are made solely on the current packet. Filtering can be performed at the incoming interface of a router, the outgoing interface, or both. Suppose we want to allow inbound mail traffic to our gateway machine, but block mails from a particular untrusted side. We can implement the following two rules[1]:

action	source	port	destination	port	comment
block	*	*	SPIGOT	*	*we don't trust these people*
allow	OUR-GW	25	*	*	*connection to our SMTP port*

Note that the order of packet filter rules matters as they are evaluated from top to bottom.

To demonstrate the concept of end-to-end enforcement, let us examine two different approaches of implementing firewalls in communications network as depicted in Figure 1.1. Traditional firewalls rely on the notion of restricted topology and control entry points to the network. All traffic from inside to outside, and vice versa, must pass through the firewall. The traditional approach relies on the assumption that every host behind the firewall is to be trusted, which no longer holds in today's networks. On the other hand, telecommuters, who use the Internet to connect, must be

[1]This example is taken from [Cheswick *et al.*, 2003].

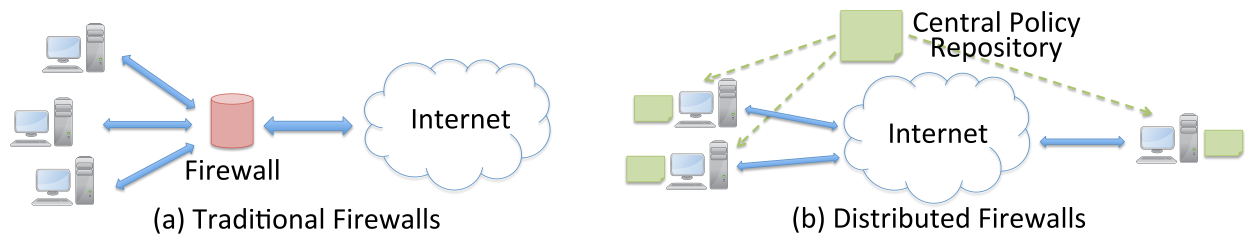

Firewall
Internet
(a) Traditional Firewalls
Central Policy Repository
Internet
(b) Distributed Firewalls

Figure 1.1: Traditional firewalls v.s. distributed firewalls.

protected if not using secure tunnels. In addition, end-to-end encryption is another problem, as the relevant packet fields cannot be examined without the necessary keys [Bellovin, 1999].

To address all the aforementioned shortcomings, Bellovin [Bellovin, 1999] proposed a distributed solution to address the shortcomings. In distributed firewalls, security policies are still centrally defined, but the enforcement is pushed to individual end hosts. A security administrator defines security policy in terms of host identifiers. The resulting policy is then shipped out to each individual host that participates in the distributed firewall. The policy file is consulted when processing incoming or outgoing messages to verify their compliance.

The most obvious advantage of the distributed firewall approach is that there is no longer a single choke point in terms of performance. Throughput is not limited by the speed of firewall processing. There is no longer a single point of failure between the inside and the outside network. A traditional firewall does not have the knowledge of the intentions from each individual host. On the contrary, distributed firewalls rely on the hosts to make appropriate and thus more secure decisions. Most importantly, distributed firewalls can protect hosts, such as those used by telecommuters, that are not within the topological boundary, regardless whether or not a tunnel has been set up.

Distributed firewalls have their own disadvantages: unwanted traffic flows all the way to destinations before being discarded. They introduce additional problem for end hosts with limited bandwidth supply and battery power, especially for wireless devices in mobile ad hoc networks (MANETs).

1.3 Requirements

The implementation of network access control solutions rely on security policies specified by system administrators. In the past two decades there have been many advances in the field of policy research. Although existing solutions in centralized systems are well-established, they do not work nearly as well in large-scale networks because of the increased complexity, i.e., scalability, network partitions, and the heterogeneity of the endpoints. The aforementioned firewall example demonstrates several key requirements in any security policy solution for large-scale distributed networks.

Firstly, the policy solution must include a refinement process that automatically translates high-level security and service requirements into low-level implementable rules. This problem is as difficult as software implementation from requirement specifications. High-level requirements are written by users who have no intimate knowledge of the managed system. In addition, users are often facilitated with a policy authoring tool that provides pre-defined templates supporting natural language like structure. Therefore they cannot be enforced directly as their syntax and semantics cannot be interpreted by individual enforcement points. Moreover, given the variety and vast number of devices managed in a large network, this refinement process must be highly automated for easy management and better performance.

Secondly, the policy solution must define a mechanism to reason about policy languages. With this mechanism, one is able to manipulate policies, move them around, combine rules, and understand the semantics of the policy languages; meanwhile ensure enforcement correctness and consistency. An essential application of distributed systems is to enable the sharing of protected data and resources among multiple parties. Policy integration facilitates the sharing in a controlled way, such that it allows policies specified by more than one policy authors to be combined to verify their compliance with the global requirements. Moreover, it is not sufficient to merely provide a policy authoring tool as policies can interact with each other, often with undesired effects. Therefore policy analysis must be performed to ensure the enforced rules are always conflict-free, not redundant and cover the entire problem space.

Thirdly, the policy solution must include a hybrid enforcement mechanism for improved system security and performance. Since both the centralized and end-to-end enforcement have their own advantages and shortcomings, a hybrid approach is more efficient that allows access control enforcement to take place selectively at intermediate nodes as well as the end hosts. Therefore malicious traffic can be dropped very early before it reaches final destination to save transmission bandwidth, especially for resource-constrained environment like a MANET. Moreover, it provides multiple policy arrangements and enforcements scenarios for the managed system to optimize the overall performance.

Last but not least, the policy solution must also include a formalism for addressing the tradeoffs and proving the correctness and consistency of distributed enforcement. The hybrid approach creates new tradeoffs in a distributed scenario. For example, implementing packet filtering at intermediate nodes allows malicious traffic to be dropped earlier for a improved performance. However this imposes trust on selected policy delegates. The compromise of any node between a delegate and the end host introduces vulnerability to the system. In addition, different policy arrangements and enforcements must always implement the global requirement correctly in a consistent manner. Therefore a policy framework is required to handle this situation, provide formalisms to increase assurance, and show how to tune tradeoffs and improve security with ubiquitous enforcement.

1.4 Hypothesis

My thesis investigates and evaluates the following hypothesis: a policy mechanism containing the following essential elements will provide a more flexible, more efficient, and more secure access control solution for distributed systems:

1. A *distributed policy refinement scheme* automating the translation from high-level security and service requirements into low-level implementable rules as inputs to the enforcement mechanism;

2. A *policy algebra framework* providing a formalism for policy delegation, composition, and analysis in distributed networks, and defining mechanisms to reason about policy languages;

3. A *ubiquitous enforcement mechanism* implementing policy delegation, whose correctness and consistency can be verified using the policy algebra.

1.5 Contributions

This dissertation contributes to the field of policy research by focusing on some of the fundamental challenges that still limit the realism and effectiveness of security policy management for large-scale distributed systems. The principle contributions are the following:

- The *first* policy algebra formalism that enables policy delegation for improved overall system performance in terms of the cost and risk of enforcing security policies.
- Two concrete instantiations of the algebraic framework to support firewall policies and Ponder2 policies respectively.
- The *first* distributed policy refinement solution for access control policies, following the successful design and development of a centralized refinement solution for network services.
- A *novel* distributed firewall mechanism, named ROFL (ROuting as the Firewall Layer), as a concrete application for policy delegation enabled by the algebraic framework. ROFL provides a new security paradigm by implementing packet filtering using the underlying routing techniques.
- An implementation of the ROFL scheme in a simulated environment with two ad-hoc routing protocols: ad-hoc on demand distance vector protocol (AODV) and optimized link state routing protocol (OLSR). The performance overhead of ROFL in terms of routing able maintenance has been analyzed to prove the feasibility of adapting ROFL by existing routing protocols. The simulation experiments demonstrate the efficiency and effectiveness of ROFL as a high performance firewall mechanism, specially for MANETs.
- A unification of policy algebra, routing algebra [Sobrinho, 2003] and the ROFL scheme to demonstrate the correctness and consistency of enforcing ROFL rules.

1.5.1 What is Not Addressed in this Dissertation

The area of security research has been active over forty years. Most of the effort made by researchers is in trying to answer the following three questions, also called the Lampson's gold standard for security [Lampson, 2004]:

1. *Authentication*: answering the question like "who said that", or "who is getting that information". It could be a user, a machine, or a program.

2. *Authorization*: answering the question "who can do which operations on this object".

3. *Audit*: figuring out what happened and why.

The process of authorization is guided by access control policies, and these two terms are often used interchangeably in the context of security policy management. While authorization is concerned with specifying permissions and prohibitions, *obligation* (or refrain policies) specify management actions that must or must not be performed. For instance, in a conference review system, the assigned reviewer is obliged to submit a paper review by a certain deadline. A positive obligation implies that the corresponding authorization must be granted. In the aforementioned example, the reviewer must be authorized to submit her review to the system. However, a negative obligation says nothing about the permissions granted to the policy subject.

This dissertation only addresses security policies with respect to authorizations (i.e., access control), with the only exception in Chapter 3 when discussing policy algebra extension for Ponder2 language that supports both authorization and obligation.

1.6 Dissertation Organization

The rest of this dissertation is organized as follows. Chapter 2 discusses the background of policy-based policy management and reviews related work. We describe the policy algebra framework with two concrete extensions for firewall policies and Ponder2 policies in Chapter 3. Chapter 4 presents two policy refinement solutions as another essential element of policy-based security management solution. Chapter 5 describes a concrete application of policy delegation in the context of firewall policies, a system we call ROFL. The performance overhead, efficiency, and and benefits of ROFL are also carefully analyzed. Following that, in Chapter 6 we provide a unification of the three mechanisms: policy algebra, routing algebra and ROFL to demonstrate the correctness and consistency of ROFL. In addition we show that ROFL paths converge. Finally, we draw conclusions from this research and point out promising future directions in Chapter 7.

Part I

Background

Chapter 2

Background and Related Work

This chapter presents a literature review of the work related to policy based management in distributed systems. We also present a short introduction to several technical areas that will be helpful in understanding the remainder of this dissertation. First, we introduce the concept of access control, summarize existing access control models in the literature, and discuss the role of security policy in these models. Second, we provide a discussion of network access control mechanisms. Third, we discuss policy based management from the perspective of the logic flow of policies, highlighting essential components such as policy specification, policy composition and analysis, and policy refinement. Then, policy based management is described and reviewed from the perspective of its use in large-scale distributed systems, specially for enforcing security policies in mobile ad hoc network (MANETs). We conclude the chapter with a discussion on the limitations of existing proposals, that lays a foundation for the solution presented in this dissertation.

2.1 Access Control Models

Access control serves as the basis for many security related services and applications. It refers to the process of regulating access to protected data and resources based on pre-defined security policies. Security policy governs the behavior choice of the managed system without the need for a reimplementation. A *security model* provides a formal representation of the security policy and its working [Samarati and de Capitani di Vimercati, 2001]. There are three classic access control models: *mandatory access control* (MAC), *discretionary access control* (DAC) and *role-based access* *control* (RBAC). These models make a clear separation between authentication and authorization. While authentication is the process of verifying the identify of a subject; authorization refers to the process of granting privilege to that subject.

Mandatory access control grants access based on the regulations mandated by a central authority. It considers the processes operated on behalf of users as subjects, thus controlling the indirect access of information by execution of processes. The most common model of mandatory access control is the multilevel security policy, where each subject and object is assigned to an access class and partial order is defined among those access classes. Authorization policies can be categorized into secrecy-based mandatory policies and integrity-based mandatory policies [Bell and LaPadula, 1973; Biba, 1977]. These two models could be applied in a dual manner to achieve the protection of both confidentiality and integrity on information. Since mandatory policies enforce access control based on classification, assigning access classes to subjects and objects at the most appropriate granularity is not always an easy task. On the other hand, mandatory policies still remain vulnerable to covert channels, where information can be inferred through abnormal communication.

Discretionary access control grants access based on the identity of requesters. It also gives users the ability to pass their privileges to other users, where the granting and revocation of privileges is regulated by administrative policies. The access control matrix provides a basic framework for describing DAC. It was first proposed by Lampson [Lampson, 1974] for resource protection within operation systems, and then was further refined and formalized in [Graham and Denning, 1972; Harrison *et al.*, 1975]. The access matrix model considers a set of subject S, a set of objects O and a set of actions A. Authorization is represented as a $|S| \times |O|$ matrix, where each entry of the matrix specifies a list of actions can be granted to subject s on object o. Typical mechanisms of enforcing DAC include: Authorization Table, Access Control List (ACL) and Capability, such that each of them interprets and implements the access matrix in a different way. Discretionary access control suffers from its vulnerabilities to Torjan horses, as the definitions of subjects do not distinguish users from processes originated by users. Therefore, it does not enforce any control on the flow of information.

Role-based access control was proposed particularly for enterprise and corporate environment, where authorizations are granted to roles instead of individual users. This maps naturally to an organization's structure. A *role* represents a set of privileges that a user playing the role is granted to. Roles can be hierarchically organized to allow the propagation of access control privilege along the hierarchy. The concept of grouping privileges was initially proposed by Baldwin [Baldwin, 1990], and different types of RBAC model have been proposed in the literature [Ferraiolo *et al.*, 2001; Sandhu *et al.*, 1996]. With RBAC, the problem of granting access privileges to individual users are mitigated to assigning roles to them. Thus the organizational responsibilities of a user, rather than the user's identity, determines the set of privileges. Defining a good role taxonomy is not an obvious task, and there are always exceptions. Besides, organizations are dynamic and the access rights associated with roles must be adjusted accordingly. Situations, such as reorganization, merge, acquisition, and so on, pose additional challenges to systems implementing RBAC.

In addition to the three classic models, credential-based access control has been proposed recently to address access control in open and dynamic scenarios, where clients and servers may not be known to each other in advance. The traditional separation between authentication and authorization cannot be applied anymore. Access control solutions need to answer which client can be granted access to the protected resource, as well as which server is qualified to provide this resource [Jajodia and Wijesekera, 2002]. Trust management was proposed as a solution for addressing access control in open systems. Access decisions are made based on policies containing credentials that effectively combine authentication and authorization. The first trust management solutions are PolicyMaker [Blaze *et al.*, 1996] and KeyNote [Blaze *et al.*, 1999]. The key idea of their proposals is to bind public keys to authorizations in order to achieve delegation of trust among keys. Based on that, STRONGMAN was proposed as the first fully automated access control system implementing the concept of trust management [Keromytis *et al.*, 2003]. It introduced three approaches to enforce local policy complying with global security policies in an efficient way.

2.2 Network Access Control

As we discussed in Chapter 1, network access control include host-based and network-based security enforcement. For the purpose of this dissertation work, we focus on the latter techniques.

Firewalls are effective means of enforcing network-based security. In some of the early work, [Mogul, 1989] describes one of the first packet filters, where a simple and flexible datagram access control mechanism was proposed and implemented in a Unix-based gateway. Packet screening can be performed to protect restricted hosts and applications within an organization from inter-organization access. Similarly, [Estrin *et al.*, 1988] presented a cryptographic technique for authenticating and authorizing a flow of datagrams, called visas. Two visa protocols were evaluated: one requires distributed state information in gateways, and one uses additional encryption operations instead. Traditional firewalls make forwarding decisions based on source and destination address as well as port numbers. They rely on the fixed topology and trust every host behind the gateway router [Cheswick *et al.*, 2003]. Distributed firewalls make no such assumption and pushes policy enforcement to each end host [Bellovin, 1999], which has been implemented using the KeyNote trust management system [Ioannidis *et al.*, 2000].

Lots of effort has been made to study firewall configuration effects from the viewpoint of network administrators. Centralized firewall policy computation can be extremely difficult [Wool, 2004], due to the large, complex rule sets to configure. [Mayer *et al.*, 2000] designed and implemented a firewall analysis tool to allow the administrator to easily discover and test the global firewall policy. [Wool, 2001] improved the firewall analyzer engine by solving the usability problem in the original paper [Mayer *et al.*, 2000]. [Bandara *et al.*, 2006] proposed another approach to firewall policy specification and analysis that uses a formal framework for argumentation based preference reasoning.

Routing provides the basic functionality for a communications network. Policy-based routing utilizes routing techniques to regulate network traffic and access to network resources (e.g., bandwidth, buffers, etc.). There are three models for policy based routing: policy based distribution of routing information, policy based packet filtering/forwarding and policy based dynamic allocation of network resources [Braun, 1989]. Each model considers the power of policy based routing in a different way. Basically, they allow administrators to control who can or who cannot use specific network resources and how network resources can be allocated based on needs and policies (e.g., traffic allocation based on different types of traffic, such as Telnet, FTP, etc.). [Yu and Braun, 1989] provides a case study of the implementation of routing between inter-peer networks: NSFNET and DDN components (the MILNET and the ARPANET). It discusses the changes made in designing inbound and outbound routing announcements using attached Mailbridges compared the old architecture for significant performance and reliability improvements.

Protecting today's enterprise networks require a combination of routing policies along with various mechanisms such as ACLs, packet filters, etc. To eliminate the need for complex ad-hoc

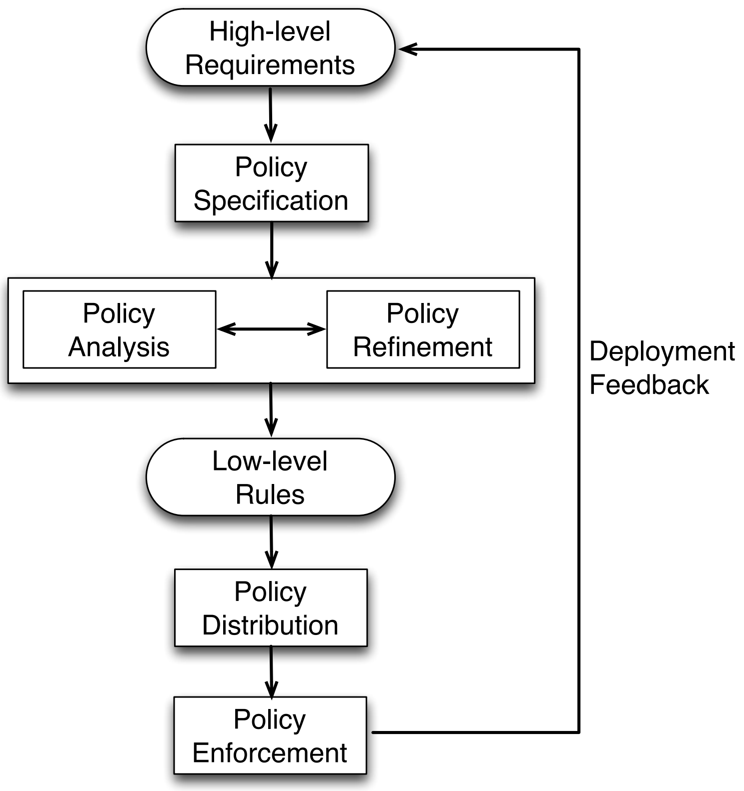


Figure 2.1: Policy based management stack.

mechanisms, SANE [Casado *et al.*, 2006], a protection architecture, was proposed to provide a single layer solution. In SANE, all the routing and accessing decisions are made by a logically-centralized policy server. Each host contact the policy server to acquire capabilities (encrypted source routes) for accessing services, according to fine-grained policies. Ethane [Casado *et al.*, 2007] extends the work of SANE to support network management in a broader approach, and provide incremental deployability and significant deployment experience.

2.3 Policy Based Management

Figure 2.1 depicts the logic flow of policies in the policy based management stack.[1] Initially, business and service requirements are expressed using high-level languages through the *policy specification*

[1]This graph is a modified version from [Bandara, 2005].

step; those high-level policies are further analyzed and refined to produce low-level mechanism rules that are then disseminated through *policy distributed* and directly implemented by individual devices at the *policy enforcement* phase. As we discussed previously, *policy analysis* must be interwoven with *policy refinement* to ensure correctness and consistency at the semantic level. In addition, enforcement outputs can be gathered as deployment feedback to policy authors to adjust their initial requirements and implement iterative enforcement approach if necessary, as it is extremely difficult to get everything right through one iteration, especially for large-scale distributed systems. This graph highlights the key components in any policy based management solution, and serves as the guideline for my research.

2.3.1 Policy Specification

To configure access control mechanisms, a number of specification languages have been defined that assist users to specify policies. The Chinese Wall [Brewer and Nash, 1989] policy combines commercial discretion with legally enforceable mandatory controls. Unlike Bell-LaPadula-like policies, a user's permitted accesses are constrained by the history of his previous accesses. Several attempts have been made towards a single policy framework, that is able to investigate and enforce multiple security policies. Woo and Lam [Woo and Lam, 1993] investigates logic-languages for the specification of authorizations for distributed systems. Their proposal abstracts from low level authorization triples and adopts a high level specification language to achieve the need of expressiveness and flexibility. Starting from that, Jajodia et al. [Jajodia *et al.*, 2001] proposed a logic-based language aiming at balance flexibility and expressiveness, as well as easy management and performance. The KeyNote system proposed by Blaze [Blaze *et al.*, 1999] specifies access control policies by effectively combining public keys with authorizations for open systems.

A variety of high-level policy specification languages have been proposed. The *security policy langauge* (SPL) [Ribeiro *et al.*, 1999] is an event-driven policy language that supports authorization, history-based and obligation-based policies. *Ponder2* [Damianou *et al.*, 2001] supports two types of policies: authorization policies and obligation policies. It specifies policies in a subject-action-target (SAT) format, in additional to optional fields such as constraints, triggers etc, and maps them onto various access control implementation mechanisms for firewalls, operating systems, databases etc. *XACML* [OASIS, 2012] stands for eXtensible Access Control Markup Language, and is initially defined by the Organization for the Advancement of Structured Information Standards (OASIS). It is a declarative access control policy language that expresses policies in an XML specification for information control over the Internet.

2.3.2 Policy Composition and Analysis

Policy composition facilitates the sharing of protected data and resources among multiple parties in a controlled way. It allows policies specified by more than one policy authors to be integrated to verify their compliance with the global requirements. The composition of access control policies have been studied in the literature. [Bonatti *et al.*, 2002] described an algebra for composing access control policies. They formulate access control policies as a set of ground terms over an alphabet for subject object action terms. Hence the policy algebra was modeled as a composition language. [Wijesekera and Jajodia, 2003] presented a propositional policy algebra for access control. They took another approach by modeling policies as nondeterministic relations over a collection of subjects, objects and action terms, and defining policy operators such as union, intersection and sequential composition etc. [Backes *et al.*, 2004] proposed an algebra for composing enterprise privacy policies, which facilitates the compliance with different privacy policies when several parts of an organization or different enterprises cooperate. The algebra provided various type of operators for composing and restricting enterprise privacy policies like conjunction, disjunction and scoping together with its formal semantics. [Pincus and Wing, 2005] models security policies as access right matrices in terms of principals, typed objects and rights. They define operations like Add, Or and Minus for combining and changing security policies. [Rao *et al.*, 2009] proposed an algebra for fine-grained integration of XACML policies. The algebra supports complex integration requirements of authorization policies using the defined algebraic operations. The authors proved the completeness and minimality of the algebra, and also described a methodology for generating integrated XACML rules using Multi-Terminal Binary Decision Diagrams.

It has long been recognized that it is not sufficient to merely provide a policy editing tool to ensure correct syntax, as policies can interact with each other, often with undesired effects. Therefore, policies need to go through a process called *ratification*, by which a new policy is approved before committed in a system [Agrawal *et al.*, 2005]. There are the following tasks specified in the ratification process:

- *Dominance check* studies the effect of adding one policy to a group of existing ones. We say that a policy A is dominated by policy group G if the adding of A does not affect the behavior of the system governed by G. Thus it helps to detect redundancy at the semantic level.
- *Coverage check* studies whether the specified policies have covered a certain range of input parameters.
- *Conflict check* detects conflict between two policies when they cannot be satisfied simultaneously.
- *Consistent priority assignment* prioritizes policies by assigning an integer value to each policy. It is considered the primary method of resolving conflicts.

Among these tasks for policy analysis, lots of effort has been devoted to studying conflict detection and resolution techniques. [Jajodia *et al.*, 1997] proposed a logical language for the specification of authorizations. This language allows users to specify different kinds of security requirements, according to which access control decisions are to be made. The logic representation also helps to perform conflict resolution and constraint checking. [Lupu and Sloman, 1997] reviews conflicts that may arise in a large-scale distributed system with role based management. Since management policies are specified in terms of domains, conflicts arise when there are overlapping between domains. This type of modality conflicts can be resolved by assigning precedence between conflicting rules. While modality conflicts can be detected purely by examining the syntax of the policies, application specific conflicts arise from the semantics of the policy, such as conflict of resources, multiple management, self management, etc. Application specific conflicts can be resolved using meta-policies. [Al-Shaer *et al.*, 2005] proposed a set of techniques to automatically discover policy anomalies and conflicts in centralized and distributed firewalls. Five different relationships were defined exclusively between two firewall rules, such that policy tree and state diagrams can be constructed to discover intra-firewall/inter-firewall anomalies and to determine the proper rule placement and ordering.

2.3.3 Policy Refinement

In policy-based security management, high-level security requirements need to be translated to low-level rules, for which the syntax and semantics can be interpreted and implemented by individual enforcement points in order to make an appropriate and consistent decision upon receiving an access request. This process is often referred to *policy refinement*, that remains one of the most ambitious goals in policy-based system management. It fills the gap between policy specification and enforcement. While the latter two have been studied intensively in the literature, only limited progress has been made in the area of policy refinement, especially for distributed systems.

Policy refinement remains one of the most ambitious goals in policy based management. Most of the existing work on policy refinement assumes the existence of a central policy server, where all the system information for refinement is available at one single place. Early research [Abadi and Lamport, 1988] performs theoretical work on refinement mappings to prove that a lower-level specification correctly implements a higher-level one. Recent studies [Bandara *et al.*, 2004; Bandara *et al.*, 2005; Beigi *et al.*, 2004; Campbell and Turner, 2008; Rubio-loyola *et al.*, 2006] address subsets of the problem, such as taking the goal-oriented approach for goal decomposition using Event Calculus, mapping policy objectives to specific configuration details using transformation algorithms, etc. In [Craven *et al.*, 2009], some initial work on policy transformation is presented that applies syntactic and algorithmic ideas adapted from the concepts of data integration. In [Craven *et al.*, 2010b], the same group proposes action decomposition techniques towards a framework for automated distributed refinement of both authorization and obligation policies. Other related work includes [Davy, 2008], where harnessing knowledge embodied in information models and ontologies is used to represent relationships between policy components that could indicate potential conflicts between policies.

2.4 Policy Based Management in Distributed Systems

Policy based management solution provides the basis for dealing with automated management of large-scale distributed systems. Policies are specified to regulate access to protected data, services, as well as network resources.

[Sloman, 1994] addressed the concept of a passive policy object to model both authorization and obligation policies in a management system. To deal with large number of devices in the system, subject and target objects are grouped into domains to which particular policies apply. System elements join or leave the domain without the need for modifying policies. Domain manipulation can be performed using set operations. These concepts can be implemented in any object based environment.

Centralized access control and resource management rely on a combination of mandatory access control (MAC) and discretionary access control (DAC), which introduces several difficulties in distributed environment. [Greenwald, 1996] described a new version of Distributed Compartment Model (DCM), that includes handles, a method for role based access control, and compartments that allows resources being managed in distributed systems across administrative domain boundaries.

[Minsky and Ungureanu, 1998] addressed the problem of having different subsystems governed by different security policies in a heterogeneous environment. A unified security mechanism was proposed to efficiently support a wide range of models and policies including MAC, DAC and other sophisticated models.

[Thompson *et al.*, 1999] implemented an access control mechanism using digital certificate to define and enforce access policies for managing distributed resources. Stakeholders specify their access requirements in use-condition certificates and users are identified by X.509 identity certificate. A policy engine then collects all the relevant certificates to make an access decision upon user's request.

2.4.1 Policy Enforcement in MANETs

A mobile ad hoc network (MANET) is a self-configuring infrastructure consisting of mobile devices connected by wireless links. Mobile devices in a MANET are free to move and hence form an arbitrary topology. Those devices are often resource-constrained with limited battery power and computational capacity.

Research on mobile ad hoc networks has been primarily focused on routing issues. [Hubaux *et al.*, 2001] provided an overview of security problems in MANET and described a fully decentralized solution to protect the routing mechanisms and security mechanisms. Their solution is based on a self-organized public key infrastructure, such that certificates are stored and distributed by users. They claim that this solution can also be applied to peer-to-peer applications.

[Chadha *et al.*, 2004] described a policy-based mobile ad hoc network management system for self-forming, self-configuring, and self-healing capabilities in the network. Network requirements are specified as high-level policies and then implemented in the network by intelligent agents. The main functionality in the system is to facilitate monitoring and implementation of network configuration.

[Chiang *et al.*, 2006; Ritu Chadha, 2008] proposed a policy-based system, called DRAMA, for network management in MANETs. In DRAMA, network management functions are designed to perform in a distributed fashion without human intervention. To reduce bandwidth usage and increase the utility of management messages, policies are specified to control the frequency and content of message exchange. The performance of DRAMA was studied using simulation, and the results showed that it can scale up to 500 nodes.

[Alicherry and Keromytis, 2010] introduced a deny-by-default security policy enforcement architecture, called DIPLOMA, based on the concept of network capabilities. It allows compromised nodes to access only authorized services, and restricts their accessibility to local communication radius. Policy enforcement is done in a hop-by-hop distributed manner. Testbed implementation demonstrated minimal overhead in throughput and latency, and protection of network bandwidth and end hosts from attackers.

2.5 Discussion

In the past two decades there have been many advances in the field of policy research. Although existing solutions in centralized systems are well-established, they do not work nearly as well in distributed environments because of scalability, network partitions, and the heterogeneity of the endpoints.

Although a variety of policy specification languages have been proposed each with specific syntax, access control policies can be rephrased using the following generic template [Johnson *et al.*, 2010]:

{Subject} can (or cannot) perform {Action} on {Target} if {Condition}.

It states that the subject is allowed or prohibited to perform an action on the target if certain condition is satisfied. Depending on the specific enforcement mechanism, subject (target) can be a simple identifier, a domain scope expression, a public key, etc. The specified action field can be a high-level goal or a low-level operation, which emphasizes the needs for translating high-level policies into low-level mechanism rules for enforcement.

Existing work on policy composition focuses on the integration of policies using algebraic operations to produce compound rules. However, policy algebra goes beyond the semantic level when tied with policy distribution and enforcement. It allows policies to be rearranged in the network and studies the enforcement effect of compound policies using algebraic operations.

Sometimes, policy analysis tasks cannot be applied directly to high-level policies as they carry less-detailed domain knowledge of the managed system. Therefore policy analysis must be interwoven with policy refinement to achieve desired results. For example, interweaving policy refinement with conflict analysis ensures that the refined rules are always consistent with existing ones.

The development of policy-based solutions falls far behind the growth of decentralized computer systems. Previous proposals mostly focus on a single task from the policy stack. In this dissertation, we consider the flow of policies as an integrated process that consisting of individual components addressed in the following chapters.[2] Such a unified approach ensures the correctness and consistency of policy enforcement to provide a more flexible, more efficient and more secure policy solution for distributed systems.

[2]Policy specification has been studied intensively in the literature, hence is out of the scope of this dissertation work.

Part II

Solution

Chapter 3

An Algebraic Framework for Distributed Policy Enforcement

In this chapter we propose an algebraic framework for policy composition, analysis and delegation, the first step toward a distributed policy definition and enforcement solution. The algebra provides five basic operations: *addition*, *conjunction*, *subtraction*, *negation* and *projection*. With these operations, we are able to integrate policies and perform analytical tasks to ensure completeness and correctness in a rigorous way.

More importantly, the algebra facilitates a decentralized policy solution by providing many possible arrangements and enforcements of policies through delegation to achieve optimized overall system performance in terms of cost and risk. For example, consider a facility network consisting of light bulbs, light switches and the circuit breaker box, which are possible policy enforcement points. However, the first two types of devices have limited processing power and user interface. If we outsource policy for a whole building of bulbs and switches to the breaker box, we can denote the resources necessarily. Policy algebra can let us know if this is a safe thing to do. This becomes more important in situations where network topology and enforcement vary with time.

This generic algebra framework defines the high-level semantics of the algebraic operations, and hides their low-level implementation details which are highly language-specific. As concrete instantiations, we further extend the framework to support firewall policies and Ponder2 policies respectively.

3.1 An Algebraic Framework for Policy Composition and Delegation

Security policy research largely focuses on the specification and management of access control requirements. The questions of how to understand the interactions between access control policies and how to enforce consistency in a policy-based system have not yet been adequately investigated. Moreover, existing policy composition and analysis solutions are mostly concerned with merging individual policy authors' security requirements in a controlled way. However, policy integration has another important but often neglected implication – that is to enable enforcement delegation in heterogenous environments, where each device may incur different expense in terms of cost and risk for enforcing the same security policy. Therefore, we proposed an algebraic framework for policy composition, analysis and delegation, the first step towards a distributed policy management solution. Our algebra defines mechanisms to reason about policy languages. It takes sets of policy rules as input and output, manipulates them, move them around, and combine them to understand the semantics of the policy language.

3.1.1 Policy Algebra

We observe that the algebraic approach of composing access control policies can occur at different levels of abstraction. Most existing policy languages are flexible enough to obtain the desired policy combination behavior. Thus the composition can be applied directly at the level of their language specification, such as the algebra for XACML policies [Rao *et al.*, 2009]. On the other hand, for composing policies initiated for different applications or from different policy authors, a much lower level composition scheme on access control primitives is necessary as there is no single adopted high level policy language. The work by Bonatti *et al.* [Bonatti *et al.*, 2002] and Wijesekera *et al.* [Wijesekera and Jajodia, 2003] took the latter approach. They model an access control policy as a set of ground authorization terms, that are triples of the form $\langle subject,\ object,\ action\rangle$.

While each of the above proposals has its own advantages and drawbacks, we believe an unified approach, that defines algebra operations with their semantics at high level, and implements the detailed and language-specific operations at low level, provides the most flexible and promising solution. Thus we proposed a policy algebra for integrating access control policies by utilizing the following set of algebraic operators whose semantics are formally defined within the policy context [Zhao and Bellovin, 2007].

- *Addition* (+): $P = P_1 + P_2$ results in an integrated policy P equivalent to the union of the two. Note that the addition operation could be non-commutative if the two policies are correlated and the order of applying them matters. For instance, the security effect of applying a firewall policy P_1 followed by P_2 is not necessarily the same as applying P_2 first.

- *Conjunction* (&): $P = P_1 \& P_2$ results in an integrated policy P equivalent to the intersection of the two. If the two policies share no security effect in common, then we have $P_1 \& P_2 = \emptyset$.

- *Subtraction* ($-$): $P = P_1 - P_2$ eliminates all access rules in P_2 from P_1 if any.

- *Negation* ($\neg$): $P = \neg P_1$ returns a policy P that has the opposite access decision for each request applied to P_1.

- *Projection* (π): $P = \pi^c P_1$ extracts a portion of a given policy P_1 based on certain criteria c. For example, $\pi^t P_1$ represents a portion of P_1 that applies to a specific target t only.

Compound policies can be computed and expressed using the defined operations over existing ones. The algebra also provides a solid foundation for evaluating policies and performing analytical tasks, such as coverage check, conflict detection and resolution, etc. We will elaborate more on policy analysis using concrete instantiations discussed in subsequent sections.

3.1.2 Delegation of Enforcement

Depending on a host's capability, such as battery power, computation capacity, etc., it may decide to outsource some policies to a carefully selected set of delegates that are more capable to perform the enforcement on behalf of itself. Integrating policies using the aforementioned algebraic operations enables many different arrangements and enforcements for the same set of security policies in a distributed heterogeneous environment. For a given set of initial policies, we wish to find one enforcement delegation that has the following properties:

1. It verifiably has the same security semantics as the original;

2. It minimizes total system cost for policy enforcement;

3. It keeps the risk within acceptable bounds.

This algebra is the very first formalism that provides multiple policy arrangements and enforcements through delegation to optimize overall system performance. Consider a heterogeneous network of N nodes. Each node n_i enforces its own set of policies $\mathcal{P}_i$, and together they achieve the desired global policy requirement $\mathcal{P}^{\mathcal{G}}$, such that

$$\mathcal{P}^{\mathcal{G}} = \sum_{i=1}^{N} \mathcal{P}_i = \mathcal{P}_1 + \mathcal{P}_2 + \cdots + \mathcal{P}_N, \tag{3.1}$$

With delegation of policy enforcement, each node decides to outsource policy $\mathcal{P}_i^O$ to selected delegates and also accepts foreign policy $\mathcal{P}_i^I$ from others,[1] which results in a different arrangement of policy enforcement $\mathcal{P}^{\mathcal{G}'}$, such that:

$$\mathcal{P}^{\mathcal{G}'} = \sum_{i=1}^{N} (\mathcal{P}_i - \mathcal{P}_i^O + \mathcal{P}_i^I) \tag{3.2}$$

The *global compliance check* requires that semantically $\mathcal{P}^{\mathcal{G}} = \mathcal{P}^{\mathcal{G}'}$. That is outsourcing must not affect the correctness and consistency of policy enforcement in order to preserve property 1.

3.1.3 Cost and Risk Functions

To the best of our knowledge, we are the first to introduce the concept of the cost and the risk metrics associated with policy enforcement in a distributed environment. Policy enforcement is costly as it requires additional software and hardware effort to implement the enforcement mechanism. Thus, we define a cost function $\mathcal{C}(n_i, \mathcal{P}_i)$ associated with the enforcement of policy $\mathcal{P}_i$ at each node n_i. Policy delegation enables a single policy $\mathcal{P}$ to be enforced at different locations in the network with, consequently, different costs, i.e., $\mathcal{C}(n_i, \mathcal{P}) \neq \mathcal{C}(n_j, \mathcal{P})$. For example, intuitively it is less expensive for a wired host to enforce an access control policy that requires the verification via public-key cryptography than on a mobile device with limited battery power and computation capability.

On the other hand, policy delegation is risky. A host becomes vulnerable if its security policy delegates are compromised. Different risk levels could be incurred if the host outsources its security policies to varying locations in the network, e.g., the likelihood of a delegate being captured by the

[1] Note that $\mathcal{P}_i^O = \emptyset$ indicates no outsource of enforcement is initiated by this node, and $\mathcal{P}_i^I = \emptyset$ implies this node is not selected as a delegate by others.

attacker may vary. A host may also be willing to reduce the risk by choosing multiple delegates to avoid single point failure. Hence we use $\mathcal{R}(n_i, \mathcal{P}_i)$ to represent the risk of enforcing policy $\mathcal{P}_i$ at host n_i. Intuitively, it is often the case that $\mathcal{R}(n_i, \mathcal{P}_i)$ is minimized if host n_i decides to implement $\mathcal{P}_i$ by itself (e.g., the distributed firewall approach described in [Bellovin, 1999]).

Given the formalism of policy algebra and the cost and the risk functions, we define the global cost of policy enforcement for the entire network as

$$\mathcal{C}^{\mathcal{G}} = \sum_{i=1}^{N} C(n_i,\ (\mathcal{P}_i - \mathcal{P}_i^O + \mathcal{P}_i^I)) \tag{3.3}$$

Similarly, the global risk of policy enforcement for the entire network can be represented as

$$\mathcal{R}^{\mathcal{G}} = \sum_{i=1}^{N} R(n_i,\ (\mathcal{P}_i - \mathcal{P}_i^O + \mathcal{P}_i^I)) \tag{3.4}$$

Thus, minimization on the global cost is achieved by an optimal enforcement configuration that minimizes $\mathcal{C}^{\mathcal{G}}$. A similar minimization on global risk can be performed on $\mathcal{R}^{\mathcal{G}}$ as well. Usually, one specific enforcement configuration can only arrive at the optimization of one metric but not both. Policy enforcement with lower cost may result in a higher risk level. As a concrete example, suppose host A decides to outsource policy $\mathcal{P}_A$ to its upstream neighbor B with more computation power. In this case, lower cost $\mathcal{C}(B, \mathcal{P}_A)$ is incurred but with higher risk $\mathcal{R}(B, \mathcal{P}_A)$ since A becomes vulnerable if any host along the path from A to B gets comprised. Under most circumstances, we are more interested in minimizing the cost function $\mathcal{C}^{\mathcal{G}}$. When such a trade-off indeed occurs, it is the system administrator's choice to balance these two. One possibility, as we suggest, is to consider only enforcement arrangements with risk metric below certain threshold value λ_R. Thus the global cost minimization only considers $\mathcal{C}^{\mathcal{G}}$ as a candidate enforcement configuration, if for every $\mathcal{P}_i$, $\mathcal{R}(n_i, \mathcal{P}_i) \leq \lambda_R$ is satisfied.

3.2 Policy Instantiations

As discussed in previous sections, our policy algebra framework can be applied to many types of security policies as we do not make assumption on the policy specification languages used. The algebra defines high-level semantics of the operations, and hides their low-level implementation details which are highly language-specific. As concrete instantiations, we extended it to support firewall policies [Zhao and Bellovin, 2007], traditional low-level access control policies specified at the network level, and Ponder policies [Zhao *et al.*, 2008b], a well-known high-level declarative policy specification language. We aim at demonstrating different aspects of the algebraic framework in these two instantiations: the algebra for firewall policies focuses on enforcement delegation to improve overall system performance; whereas the Ponder2 algebra illustrates the benefits of policy composition and analysis using the provided algebraic operators. In addition, the Ponder2 instantiation can be easily adapted to support a large variety of policy specification languages following the general *Subject-Action-Target* (SAT) format.

3.2.1 An Algebra for Hybrid Firewall Policy

Firewalls can be categorized into packet filters, circuit-level gateways, application-level filters, and so on [Cheswick *et al.*, 2003]. In this thesis, we choose packet filter rules for a concrete instantiation primarily because it is the most basic type of firewall installed in almost any gateway router. However, due to the increasing complexity of network topology and security requirements, the set of packet filter rules can become extremely large and hence is very difficult to correctly configure and maintain [Wool, 2004]. Moreover, packet filters become the network bottleneck when the increased filtering latency forces an interface buffer queue to drop further packets. With the support of the algebraic framework, we can apply enforcement delegation on packet filter rules to improve the overall system performance.

3.2.1.1 Refinement of Algebraic Operators

To better understand the semantics of these algebra operations in the context of firewalls, syntactically we define a packet filter rule P as a 5-tuple, such that

$$P = (Act,\ IP_s,\ Port_s,\ IP_d,\ Port_d)$$

where the action field $Act = \{allow,\ deny\}$, IP_s (or IP_d) represents the source (or destination) IP address prefix, and $Port_s$ (or $Port_d$) denotes a set of one or more source (or destination) port numbers. Given two packet filter rules $P_1 = (Act_1, IP_{s_1}, Port_{s_1}, IP_{d_1}, Port_{d_1})$ and $P_2 = (Act_2, IP_{s_2}, Port_{s_2}, IP_{d_2}, Port_{d_2})$, we now refine the aforementioned algebraic operators as follows:

- *Addition* (+): $P = P_1 + P_2$ we have the following two cases:

1. if $Act_1 = Act_2$ and three out of the four remaining fields are the same, except for field X, say X represents IP_s in this case, then we have $P = P_1 + P_2 = (Act_1, IP_{s_1} \cup IP_{s_2}, Port_{s_1}, IP_{d_1}, Port_{d_1})$. [2] The same pattern can be applied if X represents $Port_s$, IP_d and $Port_d$ respectively.
2. otherwise, $P = \{P_1, P_2\}$, i.e., the addition of two packet filter rules produces a set of two rules of which the order does matter.

- *Conjunction* (&): $P = P_1 \& P_2$ we have the following two cases:

  1. if $Act_1 = Act_2$, then we have $P = P_1 \& P_2 = (Act_1, IP_{s_1} \cap IP_{s_2}, Port_{s_1} \cap Port_{s_2}, IP_{d_1} \cap IP_{d_2}, Port_{d_1} \cap Port_{d_2})$.
  2. otherwise, $P = \emptyset$.

- *Subtraction* ($-$): $P = P_1 - P_2$ similar to the addition operation, we have the following two cases:

  1. if $Act_1 = Act_2$ and three out of the four remaining fields are the same, except for field X, say X represents IP_s in this case, then we have $P = P_1 + P_2 = (Act_1, IP_{s_1} \backslash IP_{s_2}, Port_{s_1}, IP_{d_1}, Port_{d_1})$. The same pattern can be applied if X represents $Port_s$, IP_d and $Port_d$ respectively.
  2. otherwise, $P = P_1$, since the filtering effect specified in P_2 does not affect the filtering decision in P_1 at all.

- *Negation* ($\neg$): $P = \neg P_1$ we have $P = \neg P_1 = (\neg Act_1, IP_{s_1}, Port_{s_1}, IP_{d_1}, Port_{d_1})$, such that $\neg allow = deny$ and $\neg deny = allow$.

- *Projection* (π): $P = \pi^c P_1$ based on certain filtering criteria c. For example, $P = \pi^{(IP,\ Port)} P_1$ extracts the filtering effect of P_1 based on a specific pair of IP address and port number such that $P = \pi^{(IP,\ Port)} P_1 = (Act_1, IP_{s_1} \cap IP, Port_{s_1} \cap Port, IP_{d_1}, Port_{d_1})$.

We illustrate the filtering effect of combining two packet filter rules using operators $+$, &, $-$, $\neg$ in Table 3.1. In the sub-matrix for each binary operator, the first column represents the filtering

[2]Here we treat IP address prefixes and port numbers as sets. So we can apply set operations such as union $\cup$, intersection $\cap$ and difference $\backslash$.

effect of P_1 with respect to an incoming packet and the first row represents the filtering effect of P_2 with respect to the same packet. The last sub-matrix depicts the filtering effect of applying the unary operation $\neg$ on P_1. Since we are examining packet filter rules, we have four possible choices: Y means the packet is allowed to pass through; N means the packet is going to be dropped; NA means no deterministic decision can be made based on the current input (assuming no default rule is applied); and finally C represents a potential conflict between the two rules unless a specific order is enforced.

$P_1 + P_2$	Y	N	NA	$P_1 \& P_2$	Y	N	NA	$P_1 - P_2$	Y	N	NA	P_1	$\neg P_1$
Y	Y	C	Y	Y	Y	NA	NA	Y	N	Y	Y	Y	N
N	C	N	N	N	NA	N	NA	N	N	Y	N	N	Y
NA	Y	N	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA

Table 3.1: Policy combination matrices for operators $+$, $\&$, $-$, $\neg$.

3.2.1.2 Conflict Resolution

Unlike other access control policies, firewall enforcement introduces extra difficulties as the order of packet filter rules may alter the security semantics. For example, when combining two rules with opposite filtering effects using the addition (+) operation, conflict may arise (indicated as C in Table 3.1) as justified in Addition case (2). It is the network administrator's decision how to resolve conflicts when applying algebraic operations. Our algebra cannot prevent conflicts; it can, however, reveal their existence.

One way to resolve conflict is to specify the order between rules such that the preceding rule is always given high priory over the next one. Another approach is to perform *decorrelation* on packet filter rules to remove any ambiguity [Hari *et al.*, 2000]. Two rules are considered to be correlated if there is a non-nil intersection between the values of each of their attributes except for the action field. Correlated rules introduce potential conflicts as an incoming packet would match two rules with potentially opposite filtering decisions. Once the rules are decorrelated, there is no ordering requirement on the filtering rules, since at most one rule will match a packet. Negative expressions can be introduced to improve the efficiency of decorrelation algorithm, by generating a relatively small set of decorrelated rules. However, the design issue becomes how to define and allow a simple, yet adequate, form of negative expressions in security policy specification [Bartal *et al.*, 1999].

3.2.1.3 Enforcement Delegation in Hybrid Firewalls

As motivated in Chapter 1, traditional firewalls and distributed firewalls each have their own advantages and shortcomings. The former rely on a fixed topology and trust every single host behind the defense; whereas the latter allow malicious traffic flow all the way to destination and hence waste transmission bandwidth and battery power (e.g., for wireless devices). To take advantage of both schemes, policy algebra further expands the idea of distributed firewalls by utilizing the property of enforcement delegation to construct a hybrid firewall scheme. It allows each end host select a set of delegates to enforce firewall policies on behalf of itself. Hence policy enforcement becomes quite flexible, such that they can be enforced at individual end hosts in a purely distributed manner, or at some intermediate nodes which act as gateways to protect systems sitting behind. Multiple arrangements and enforcements of firewall policies can be examined to optimize overall system performance and security. Therefore, hybrid firewalls take advantages of both traditional and distributed firewall approaches; meanwhile, they enable early dropping of unwanted traffic closer to their origins, and hence reduce potential collisions among packet transmissions in MANETs, which often requires additional MAC layer efforts and collaborations to resolve.

There are several attempts to define proper cost and risk metrics. In reality, the size of a firewall rule set limits how well a packet filter can achieve its goal. A rule set is a linear list of individual rules, which are evaluated from top to bottom upon each packet arrival. Packets have to pass through the filtering device, adding a certain amount of latency between the time a packet is received and the time it is forwarded. Any device can only process a finite number of packets per second. When packets arrive at a higher rate than the device can forward them, packets are lost. The input processor passes packets to the packet filter in a sequential manner. Thus an obvious choice for the cost function associated with each firewall is the size of rule set. As the rule set gets larger, it takes more time and computational resources to perform the filtering task. Although several rule set optimization mechanisms have been proposed, it remains the bottleneck for packet filter performance. Moreover, by delegating packet filter rules to upstream neighbors, the filtered packets will not consume any transmission power and bandwidth capacity on the downstream links. Intuitively, devices with more battery power are better choices for policy delegation. Thus the cost metric is also subject to the node's energy level. On the other hand, the number of hops between the delegate and its protected host could be a reasonable risk metric since any single compromised host or link along the path between those two exposes security hole to the attacker. Hybrid firewalls pose a trade-off between cost and risk: the further from the node security policies are enforced, the less transmission and filtering cost there; on the other hand, the risk of compromise is greater.

3.2.2 An Algebra for Ponder2 Policy

We further extend our policy algebraic framework to support Ponder2 policies [Lupu *et al.*, 2007], a high-level declarative policy specification language. It provides a common means of specifying security policies that map onto various access control mechanisms for firewalls, operating systems, databases, etc. It primarily supports two types of policies: *authorization* that defines which actions are permitted under given circumstances, and *obligation* that defines which actions should be performed in response to an event occurring if certain condition is satisfied. Ponder2 specifies policies in a general *subject-action-target* (SAT) format, in additional to optional fields such as constraints, event triggers, etc. We propose a simple, yet powerful, notation of policy semantics to capture both authorization and obligation requirements to study their composition and interaction. Hence this algebra instantiation can be easily adapted to support a large variety of policy languages in the SAT format.

3.2.2.1 Policy Semantics

Definition 3.2.1 *A security policy is defined as an evaluation function $\mathcal{P} : \mathcal{ST} \times \mathcal{A} \rightarrow \mathcal{D}$, where $\mathcal{ST}$ is a set of system states, $\mathcal{A}$ represents a finite set of actions and $\mathcal{D}$ denotes the set of decision tuples $\{\langle d_a, d_o \rangle\} = \{\langle Y, Y \rangle, \langle Y, NA \rangle, \langle N, NA \rangle, \langle NA, NA \rangle\}$. The function $\mathcal{P}$ takes a system state $st \in \mathcal{ST}$ and an action $a \in \mathcal{A}$ as input, and returns a decision tuple $\langle d_a, d_o \rangle$ on whether action a is authorized and obliged to execute in system state st.*

The above definition supports three-valued policies when evaluating an authorization request, and two-valued policies when evaluating an obligation request (i.e., Y for obliged and NA for not applicable).[3] Note that we exclude $\langle N, Y \rangle$ and $\langle NA, Y \rangle$ from the set of decision tuples to ensure

[3]We do not consider negative obligations in our model, because negative obligations can be easily transformed into access control requirements like refrain policies, and thus can be enforced directly.

policy consistency as an obligation decision intuitively implies the existence of a corresponding positive authorization in the system.

Definition 3.2.2 *Let $\mathcal{S}$ be the set of subjects, $\mathcal{T}$ be the set of targets, $\mathcal{E}$ be the set of event triggers and $\mathcal{C}$ be the set of constraints. We now define system states as $\mathcal{ST} = \mathcal{E} \times \mathcal{C} \times \mathcal{S} \times \mathcal{T}$. This definition allows a system state to be described as $st = \langle e, c, s, t \rangle$ consisting of an event trigger $e \in \mathcal{E}$, a constraint $c \in \mathcal{C}$, a subject $s \in \mathcal{S}$ and a target $t \in \mathcal{T}$ in Ponder2.*

Definition 3.2.3 *We also define the following two policy constants in the algebra $P^+ : \mathcal{ST} \times \mathcal{A} \to \langle Y, NA \rangle$ and $P^- : \mathcal{ST} \times \mathcal{A} \to \langle N, NA \rangle$. More precisely, P^+ specifies that every authorization request will be allowed in any state and no obligation is required in the system; whereas P^- says that any authorization request is denied therefore no obligation is required.*

It is quite common for a policy-based management system to adopt a default authorization policy: everything is allowed or everything is forbidden. For instance, Ponder2 allows the system administrator to specify a default authorization policy: `ALL+` (P^+) or `ALL-` (P^-). Firewall policy is another example, where a default rule "*allow* $* * * *$" (or "*deny* $* * * *$") can be specified to accept (or drop) every incoming packet.

3.2.2.2 Refinement of Algebraic Operators

The algebra consists of the two constant policies P^+ and P^- and six basic algebraic operators: addition ($+$), intersection ($\&$), subtraction ($-$), projection (π), negation for authorization ($\neg_a$), and negation for obligation ($\neg_o$). Let P_1 and P_2 be two policies to be combined, and P_I be the result from combination. To assist our discussion, we introduce three binary operators $\oplus, \otimes, \ominus$ in Table 3.2. The first column of each matrix are the evaluation results of P_1 with respect to an access request (st, a) and the first row are the results of P_2 with respect to the same input. Each entry denotes the effect of integrating P_1 and P_2 using $\oplus, \otimes, \ominus$ respectively. The effect of two negation operators $\neg_a$ and $\neg_o$ are also illustrated in this table.

Now we are ready to refine the algebraic operators in the context of Ponder2 policies as follows:

- *Addition* ($+$): Addition of P_1 and P_2 results in a policy P_I equivalent to the union of them.

$P_1(st,a) \oplus P_2(st,a)$	$\langle Y,Y\rangle$	$\langle Y,NA\rangle$	$\langle N,NA\rangle$	$\langle NA,NA\rangle$
$\langle Y,Y\rangle$	$\langle Y,Y\rangle$	$\langle Y,Y\rangle$	$\langle NA,NA\rangle$	$\langle Y,Y\rangle$
$\langle Y,NA\rangle$	$\langle Y,Y\rangle$	$\langle Y,NA\rangle$	$\langle NA,NA\rangle$	$\langle Y,NA\rangle$
$\langle N,NA\rangle$	$\langle NA,NA\rangle$	$\langle NA,NA\rangle$	$\langle N,NA\rangle$	$\langle N,NA\rangle$
$\langle NA,NA\rangle$	$\langle Y,Y\rangle$	$\langle Y,NA\rangle$	$\langle N,NA\rangle$	$\langle NA,NA\rangle$

$P_1(st,a) \otimes P_2(st,a)$	$\langle Y,Y\rangle$	$\langle Y,NA\rangle$	$\langle N,NA\rangle$	$\langle NA,NA\rangle$
$\langle Y,Y\rangle$	$\langle Y,Y\rangle$	$\langle Y,NA\rangle$	$\langle NA,NA\rangle$	$\langle NA,NA\rangle$
$\langle Y,NA\rangle$	$\langle Y,NA\rangle$	$\langle Y,NA\rangle$	$\langle NA,NA\rangle$	$\langle NA,NA\rangle$
$\langle N,NA\rangle$	$\langle NA,NA\rangle$	$\langle NA,NA\rangle$	$\langle N,NA\rangle$	$\langle NA,NA\rangle$
$\langle NA,NA\rangle$	$\langle NA,NA\rangle$	$\langle NA,NA\rangle$	$\langle NA,NA\rangle$	$\langle NA,NA\rangle$

$P_1(st,a) \ominus P_2(st,a)$	$\langle Y,Y\rangle$	$\langle Y,NA\rangle$	$\langle N,NA\rangle$	$\langle NA,NA\rangle$
$\langle Y,Y\rangle$	$\langle NA,NA\rangle$	$\langle NA,NA\rangle$	$\langle NA,NA\rangle$	$\langle Y,Y\rangle$
$\langle Y,NA\rangle$	$\langle NA,NA\rangle$	$\langle NA,NA\rangle$	$\langle NA,NA\rangle$	$\langle Y,NA\rangle$
$\langle N,NA\rangle$	$\langle NA,NA\rangle$	$\langle NA,NA\rangle$	$\langle NA,NA\rangle$	$\langle N,NA\rangle$
$\langle NA,NA\rangle$	$\langle NA,NA\rangle$	$\langle NA,NA\rangle$	$\langle NA,NA\rangle$	$\langle NA,NA\rangle$

$P(st,a)$	$\langle Y,Y\rangle$	$\langle Y,NA\rangle$	$\langle N,NA\rangle$	$\langle NA,NA\rangle$
$\neg_a P(st,a)$	$\langle N,NA\rangle$	$\langle N,NA\rangle$	$\langle Y,NA\rangle$	$\langle NA,NA\rangle$
$\neg_o P(st,a)$	$\langle Y,NA\rangle$	$\langle Y,Y\rangle$	$\langle N,NA\rangle$	$\langle NA,NA\rangle$

Table 3.2: Policy combination matrix for operators $\oplus, \otimes, \ominus, \neg_a, \neg_o$

$$\begin{aligned} P_I &= P_1 + P_2, \text{ such that} \\ P_I(st,a) &= P_1 + P_2(st,a) \\ &= P_1(st,a) \oplus P_2(st,a) \end{aligned}$$

- *Conjunction* (&): Conjunction of P_1 and P_2 results in a policy P_I equivalent to their set intersection.

$$\begin{aligned} P_I &= P_1 \& P_2, \text{ such that} \\ P_I(st,a) &= P_1 \& P_2(st,a) \\ &= P_1(st,a) \otimes P_2(st,a) \end{aligned}$$

- *Negation* ($\neg_a, \neg_o$): Negation of a policy P returns a policy P_I that permits (denies) all requests denied (permitted) by P. The operator $\neg_a$ is used to negate the result of evaluating an authorization request. It does not modify the obligation part with the only exception that $\neg_a P_I(st,a) = \langle N, NA\rangle$ if $P(st,a) = \langle Y,Y\rangle$ since $\langle N,Y\rangle$ is an invalid decision tuple. Similarly, the operator $\neg_o$ is applied to negate the evaluation result of an obligation request without changing the authorization part.

$$\begin{aligned} P_I &= \neg P\text{, such that} \\ P_I(st,a) &= \neg P(st,a)\text{, where } \neg \in \{\neg_a, \neg_o\} \end{aligned}$$

- *Subtraction* ($-$): Subtracting policy P_2 from P_1 results in a policy P_I which applies only to those requests that P_2 does not apply to. The subtraction operation can be expressed in terms of $\{+, \&, \neg_a, \neg_o\}$.

$$\begin{aligned} P_I &= P_1 - P_2\text{, such that} \\ P_I(st,a) &= P_1 - P_2(st,a) \\ &= P_1(st,a) \ominus P_2(st,a) \\ &= (P_1 + \neg_a P_2) \& (P_1 + \neg_o P_2)(st,a) \end{aligned}$$

- *Projection* (Π): The projection operation is used to extract a portion of a given policy P. Let $c(\mathcal{ST} \times \mathcal{A})$ be a computable subset of $\mathcal{ST} \times \mathcal{A}$. The projection operation restricts the applicability of policy P based on the evaluation result of request $(st,a) \in c(\mathcal{ST} \times \mathcal{A})$. The optional super index $\langle d_a, d_o \rangle$ is required only if there is further restriction on the evaluation results of input (st,a).

$$\begin{aligned} P_I &= \Pi^{\langle d_a,d_o \rangle}_{c(\mathcal{ST}\times\mathcal{A})} P\text{, such that} \\ P_I(st,a) &= \Pi^{\langle d_a,d_o \rangle}_{c(\mathcal{ST}\times\mathcal{A})} P(st,a) \\ &= \begin{cases} \langle d_a, d_o \rangle & \text{if } (st,a) \in c(\mathcal{ST} \times \mathcal{A}) \text{ and } P(st,a) = \langle d_a, d_o \rangle \\ \langle NA, NA \rangle & \text{otherwise} \end{cases} \end{aligned}$$

3.2.2.3 Properties and Expressiveness of the Algebra

The aforementioned algebra instantiation in the context of Ponder2 policies has all the algebraic properties listed in Theorem 1 in [Rao *et al.*, 2009] except for those involving negation operations. Instead, we have the following properties for negations:

1. *Commutativity:* $P_1 + P_2 = P_2 + P_1$; $P_1 \ \& \ P_2 = P_2 \ \& \ P_1$;

2. *Associativity:* $(P_1 + P_2) + P_3 = P_1 + (P_2 + P_3)$; $(P_1 \ \& \ P_2) \ \& \ P_3 = P_1 \ \& \ (P_2 \ \& \ P_3)$;

3. *Adsorption:* $P_1 + (P_1 \ \& \ P_2) = P_1$; $P_1 \ \& \ (P_1 + P_2) = P_1$;

4. *Distributivity:* $P_1+(P_2 \,\&\, P_3) = (P_1+P_2) \,\&\, (P_1+P_3)$; $P_1 \,\&\, (P_2+P_3) = (P_1 \,\&\, P_2)+(P_1 \,\&\, P_3)$; $\Pi(P_1 + P_2) = (\Pi P_1) + (\Pi P_2)$; $\Pi(P_1 \,\&\, P_2) = (\Pi P_1) \,\&\, (\Pi P_2)$; $\neg_a(P_1 + P_2) = \neg_a P_1 + \neg_a P_2$; $\neg_a(P_1 \,\&\, P_2) = \neg_a P_1 \,\&\, \neg_a P_2$;

5. *Complements for authorization::* $P_+ = \neg_a P_-$, $P_- = \neg_a P_+$;

6. *Idempotence:* $P_1 + P_1 = P_1$; $P_1 \,\&\, P_1 = P_1$;

7. *Involution for authorization:* $\neg_a(\neg_a P_1) = P_1$.

We introduced two constant policies P^+ and P^- in Definition 3.2.3. The other two constant policies $P^Y : \mathcal{ST} \times \mathcal{A} \rightarrow \langle Y, Y\rangle$ and $P^{NA} : \mathcal{ST} \times \mathcal{A} \rightarrow \langle NA, NA\rangle$ can be produced using P^+ and P^- such that $P^Y = \neg_o P^+$ and $P^{NA} = P^+ + P^- = P^+ \& P^-$.

Corollary 3.2.4 [4] *Following Theorem 1 in [Rao* et al.*, 2009], given any n-dimensional policy combination matrix M, let $M^*(P_1, P_2, \ldots, P_n)$ be the result of integrating policies $P_1, P_2, \ldots, P_n$ using matrix M. We can always find an algebra expression $\mathcal{E}$ such that $\mathcal{E}(P_1, P_2, \ldots, P_n) = M^*(P_1, P_2, \ldots, P_n)$, where $\mathcal{E}$ consists of $\{P^+, P^-, +, -, \&, \neg_a, \neg_o, \pi\}$. Therefore, our algebra is complete.*

3.2.2.4 Policy Integration and Analysis

Integration of policies $P_1, P_2, ..., P_n$ can be easily expressed as $P_I = P_1 + P_2 + ... + P_n$. The result of policy integration P_I will permit (deny) requests that are permitted (denied) by any policy P_i. If a request is permitted by one policy but denied by another, P_I does not make any decision and simply returns $\langle NA, NA\rangle$ as the result. Sometimes, we may also want to extract desired parts of individual policies and combine them into an integrated one. For example, policy P_I combines the allows from P_1 and the denise from P_2 using $P_I = \pi^{\langle Y,\ d_{o_1}\rangle} P_1 + \pi^{\langle N,\ d_{o_2}\rangle} P_2$. Moreover, it is quite common for a policy-based management system to adopt a default policy P_d, such that any request evaluated to $\langle NA, NA\rangle$ by the integrated policy P_I will take the default result specified by P_d. Thus we have $P'_I = P_I + (P_d - P_I)$, where $P_d \in \{P^+,\ P^-,\ P^Y\}$ depending on the system requirement.

[4] The detailed proof of this corollary is omitted as it follows the proof of completeness by induction for Theorem 1 in [Rao *et al.*, 2009].

It has been long recognized that merely providing a policy editing tool to ensure correct policy syntax is not sufficient. Policies can interact with each other, often with undesirable effects [Agrawal *et al.*, 2005]. With the support of policy algebra, we can perform the following policy analysis tasks.

1. *Dominance Check*: It is important to ensure that any ineffective policy must not be deployed so that computer resources are not wasted on evaluating it. We say that a policy P is dominated by another policy P' if P is ineffective because of the existence of P', i.e. there is no access request in the system whose evaluation decision can be affected by P. For example, a policy that permits the creation of usernames that must be at least 4-digit long (i.e., "username $\geq$ 4") is dominated by another policy saying that "4 $\leq$ username $\leq$ 10". Given two policies P and P', we say that P is dominated by P' if and only if $P + P' = P'$ or $P \& P' = P$. This definition also applies to integrated policies when $P = P_1 + P_2 + ... + P_m$ and $P' = P'_1 + P'_2 + ... + P'_n$.

2. *Coverage Check*: When specifying policies for management systems, the system administrator may want to know if enough rules have been defined for a certain range of input parameters, e.g., a set of system state $\mathcal{ST}$ and an action set $\mathcal{A}$. That is, given a computable subset $c(\mathcal{ST} \times \mathcal{A})$ of $\mathcal{ST} \times \mathcal{A}$, we want to make sure that $\pi_{c(\mathcal{ST} \times \mathcal{A})} P(st, a) \neq \langle NA, NA \rangle$ for any input request $(st, a) \in c(\mathcal{ST} \times \mathcal{A})$. Similarly, coverage check can also be performed for an integrated policy P_I.

3. *Conflict Detection and Resolution*: Policy conflicts can arise due to omissions, errors or conflicting requirement when specifying policies. For example, there may be two authorization policies which permit and forbid the same activity; or an obligation policy may define an activity which is forbidden by a negative authorization. We say that two policies are in conflict if they cannot be enforced simultaneously [Lupu and Sloman, 1999]. Given two policies P_1 and P_2, we know that they conflict with each other if and only if $P_1(st, a) \neq P_2(st, a)$ for some $(st, a) \in \mathcal{ST} \times \mathcal{A}$. The key point of *conflict resolution* is to assign precedence between two conflicting rules. Generally, there are the following three possibilities:

   - $P_{I_1} = P_1 + (P_2 - P_1)$ assigns higher precedence to P_1. That is, $P_I(st, a) = P_1(st, a)$ for all (st, a) such that $P_1(st, a) \neq P_2(st, a)$.

- $P_{I_2} = P_2 + (P_1 - P_2)$ assigns higher precedence to P_2. Similarly, $P_I(st, a) = P_2(st, a)$ for all (st, a) such that $P_1(st, a) \neq P_2(st, a)$.
- $P_{I_d} = (P_1 + P_2) + (P_d - (P_1 + P_2))$ solves the conflicts by applying the default policy to requests that receive conflicting evaluation results from P_1 and P_2.

3.2.2.5 Examples

Table 3.3 illustrates sample policies taken for a conference reviewing system. Policies $\{P_1, P_2, \ldots, P_5\}$ are written in Ponder2 language including four authorization rules and one obligation rule, which are then transformed into $\{P'_1, P'_2, \ldots, P'_5\}$ using our defined policy semantics.

P_1	**auth**+ a=/author→p=/paper /paper.read() **when** submit(a,p)=T
P_2	**auth**+ r=/reviewer→p=/paper /paper.read() **when** assign(r,p)=T
P_3	**auth**+ r=/reviewer→p=/paper /paper.review() **when** assign(r,p)=T
P_4	**auth**− a=/author→p=/paper /paper.read(), /paper.review() **when** submit(a,p)=T
P_5	**on** assign(r,p)=T r=/reviewer p=/paper **do** /reviewer→/paper.review()
P'_1	P'_1(⟨submit(a,p)=T, a=/author, p=/paper⟩ , read()) = $\langle Y, NA\rangle$
P'_2	P'_2(⟨assign(r,p)=T, r=/reviewer, p=/paper⟩ , read()) = $\langle Y, NA\rangle$
P'_3	P'_3(⟨assign(r,p)=T, r=/reviewer, p=/paper⟩ , review()) = $\langle Y, NA\rangle$
P'_4	P'_4(⟨submit(a,p)=T, a=/author, p=/paper⟩ , {read(), review()}) = $\langle N, NA\rangle$
P'_5	P'_5(⟨assign(r,p)=T, r=/reviewer, p=/paper⟩ , review()) = $\langle Y, Y\rangle$

Table 3.3: Sample policies for a conference reviewing system.

Example 1: We can perform $P_I = P'_1 + P'_2$ to obtain an integrated policy P_I, such that for any request (st, a), we have $P_I(st, a) = \langle Y, NA\rangle$ if and only if $st \in$ {⟨submit(a,p)=true, a=/author, p=/paper⟩, ⟨assign(r,p)=T, r=/reviewer, p=/paper⟩} and a = read(); otherwise, $P_I(st, a) = \langle NA, NA\rangle$. That is the integrated policy P_I allows both the author and the assigned reviewer to read the paper. Similarly, $P'_I = P'_2 + P'_3$ and the resulting policy P'_I allows the reviewer to read and review the assigned paper.

Example 2: Given the proposed policy semantics, the enforcement of an obligation policy automatically implies a corresponding authorization requirement to ensure system consistency. For example, $P_I = P'_3 + P'_5 = P'_5$, i.e., the assigned reviewer must be authorized to review that paper.

EXAMPLE 3: Policy P_1' and P_4' are conflicting with each other, since $P_1'(st, a) = \langle Y, NA \rangle$ but $P_4'(st, a) = \langle N, NA \rangle$ when $(st, a) = (\langle$submit(a,p)=T, a=/author, p=/paper$\rangle$, read()). We can solve this conflict by assigning precedence to P_1', i.e., $P_I = P_1' + (P_2' - P_1')$, such that $P_I(st, read()) = \langle Y, NA \rangle$ and $P_I(st, review()) = \langle N, NA \rangle$. Hence, we allow the author to read but not to review his own paper.

3.3 Discussion

In this chapter, as the first step toward a distributed policy enforcement solution, we described an algebraic framework for policy composition, analysis and delegation. Policy algebra serves as the theoretical foundation for enforcement delegation by providing multiple arrangements and enforcements of policies in a distributed policy scenario. Chapter 5 describes an instantiation of policy delegation in the context of firewall policies. Using the proposed algebraic operations, we understand the effect of the composition of multiple policies at a single enforcement point as the result of policy outsourcing. In addition, the algebra helps to prove the correctness and consistence of distributed enforcement schemes (See Chapter 6 for a concrete demonstration for firewall application). These operations also facilitate policy analysis tasks, such as conflict detection and resolution, coverage check, dominance check, etc.

During the development of this algebra, we identified an essential element in any policy composition model: with the increasing needs of integrating and analyzing policies enforced by different system elements to achieve overall business and service requirements, it is necessary to translate policies written by different policy authors and/or in multiple high-level languages (e.g., Ponder2, XACML, SAML) into low-level access control primitives before any compositional and analytical tasks can be applied. This observation re-emphasizes the needs of a well-defined policy refinement scheme and inspires our research addressed in Chapter 4.

Chapter 4

Policy Refinement

Policy refinement translates high-level service and security requirements to low-level mechanism rules that can be directly implemented by enforcement points. There are three main objectives of policy refinement identified in a pioneer work [Moffett and Sloman, 1993]:

1. Determine the resources required to satisfy the requirements of the policy by mapping abstract entities to concrete objects and devices of the underlying system;

2. Translate high-level policies into rules in terms of operations that are supported by the underlying system;

3. Verify that the lower level policies actually meet the requirements specified by the high-level policy by ensuring that at each stage the decomposition is correct and consistent.

The refinement process must make use of domain specific information in order to meet these objectives. This implies the refinement solution must include the construction of detailed system knowledge specified by domain experts. On the other hand, these high-level requirements are mostly specified by policy authors without an intimate knowledge of the underlying system. Therefore the refinement process cannot be fully automated. However, given the diverse applications and services, and numerous objects and devices managed in large-scale distributed systems, it is important to maximize the automation of the refinement process as much as possible.

High-level requirements and goals – the inputs to a refinement process – specified by policy authors often follow certain pre-defined syntax, called policy template. It provides useful guidelines to the refinement process as the policy template naturally reflects the structure of a policy, and the syntactical notation provides semantic implications. In this chapter, we first describe a *centralized refinement scheme* focusing on *network service policies* written in the following format:

{Subject} can (or cannot) {access Service provided by} {Target} if {Condition}.

It states that a *subject* is allowed (or prohibited) to access a *service* provided by a *target* if certain *condition* holds. This kind of policy is widely used for security management in communications networks, such as the access control list implementation for regulating client requests, firewall policies, etc. Following the design and implementation of the centralized scheme, we further identify challenges and requirements of the *distributed refinement approach* for *access control policies* in general:

{Subject} can (or cannot) perform {Action} on {Target} if {Condition}.

such that the service provision field is generalized to an arbitrary *action* performed on the *target*. The outputs of the refinement process are low-level rules that can be directly interpreted and implemented by individual enforcement points. Therefore, the syntax and semantics of these rules are determined by the enforcement mechanism of the system devices.

Note that high-level policies and low-level rules are relative concepts. The distinction between them may depend on the context. For example, in one situation the “disk backup” requirement is defined as a standard low-level operation, while in other situations it might be a high-level goal such that the system administrator has to perform several low-level operations (e.g., copy data to a backup disk, encrypt the disk, and put it in a protected storage place) in order to achieve this goal.

4.1 Centralized Refinement for Network Service Policies

Our first attempt to solve the policy refinement problem is a centralized refinement scheme for network service policies specifically. In contrast to the distributed approach discussed later in this chapter, here the refinement process takes place in a central policy server that has all the information of the underlying system and the policies available at one place. Low-level rules, as the results of refinement, are then shipped to individual enforcement points for implementation.[1] One contribution of our approach is a hierarchical representation of the system structure using UML class diagram description. Thus policies can be specified at different levels of the structures and further refined to low-level rules that can be fully interpreted by system mechanisms. Our approach meets the essential objectives of policy hierarchies discussed in the pioneer work [Moffett and Sloman, 1993].

4.1.1 System Overview

Before describing the centralized policy refinement scheme in detail, we provide an overview of our solution from the system perspective.

4.1.1.1 Policy Server

We assume the refinement process takes place in a centralized policy server, where the policies are composed, refined and then disseminated to individual enforcement points. The policy server consists of three main components as depicted in Figure 4.1:

- A *Knowledge Database* that records system-specific information on a policy domain;
- A *Refinement Rule Set* that defines the refinement procedure in terms of rules;
- A *Policy Repository* that stores policies written at different levels of abstraction.

The *Knowledge Database* captures two categories of knowledge: intra-domain knowledge of confidential information disclosed within an organization, and inter-domain knowledge of coalition participants outside the organization. Inter-domain knowledge is limited by how much information an organization is willing to share with other coalition participants. For example, the UK force knows the existence of a sensor fabric in the US domain as part of inter-domain knowledge, but may not have further details of the structure and components of the US sensor fabric. The distinction between intra-domain knowledge and inter-domain knowledge indicates another important application of policy refinement. That is, a high-level policy specified by one party must be refined to low-level rules by another party for enforcement purposes, because the former has no detailed

[1]The problem of policy dissemination is out of the scope of this dissertation work.

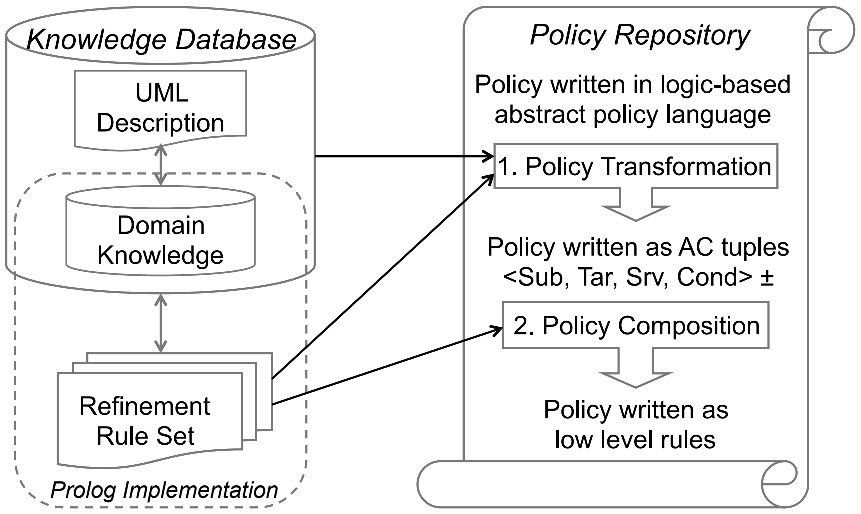


Figure 4.1: A centralized policy server for each policy domain.

information on the intra-domain knowledge of the latter, so that the low-level mechanism rules cannot be specified directly by the former. There are different representations to model a knowledge database. In the centralized approach, we describe our database using UML description and logic representation.

The *Refinement Rule Set* defines two types of rules: transformation rules that transform policies written in a logic-based abstract language into access control (AC) tuples $\langle Sub, Tar, Srv, Cond\rangle\pm$; composition rules that generate low-level enforceable rules from those tuples. Our policy refinement scheme is domain-independent, such that modifications on a policy domain do not affect refinement rules. Transformation rules are language-independent, taking policies in our abstract language as input; composition rules are highly language-specific, following exactly the syntax and semantics of low-level rules implementable by the enforcement mechanism.

The *Policy Repository* stores policies written at different levels of abstraction. For instance, it stores policies written in a structured natural language, access control lists (ACLs) implemented by individual enforcement points, as well as any intermediate results generated during the refinement process.

4.1.1.2 Policy Scenario

To assist our discussion by providing concrete examples, we will work with a policy scenario introduced in [Craven *et al.*, 2010a]. Suppose a coalition X is formed of three organizations, namely US forces, UK forces, and the Red Cross. Each organization owns various devices, communications networks, command centers, sensors and other equipment. In addition, each organization has its own *policy server*, which stores policies and performs policy analysis and refinement tasks. Each party keeps its organization-specific domain knowledge private. Without loss of generality, we address the US domain, focusing specifically on the following two sample policies written in natural language:

[US devices are permitted to access location information from one US location server with high quality, if communication is encrypted and both sides are from the same quad.] (4.1)

[Devices belonging to non-US organizations for coalition X are prohibited from querying sensor data from any US sensor fabric with high quality between 9am and 5pm.] (4.2)

4.1.1.3 System Representation

Similar to the knowledge representation approach employed in [Craven *et al.*, 2010b], we use UML representation together with first-order logic to describe domain specific information to assist the refinement process. However, unlike previous work on policy refinement [Bandara *et al.*, 2004; Craven *et al.*, 2010b], in our proposed approach, UML representation provides a hierarchical structure of the entire managed systems. The UML diagram starts with a high-level view of system components, and more detailed information on the managed individual objects are specified downwards. Therefore, by utilizing the system hierarchy, policies can be specified at different levels of the structure. On the other hand, different sub-diagrams can be presented to assist policy specification for different policy agents, who are only allowed to have partial view of the managed systems. The specified policies can be fully transformed to low-level rules when the entire UML diagram is presented at the refinement point.

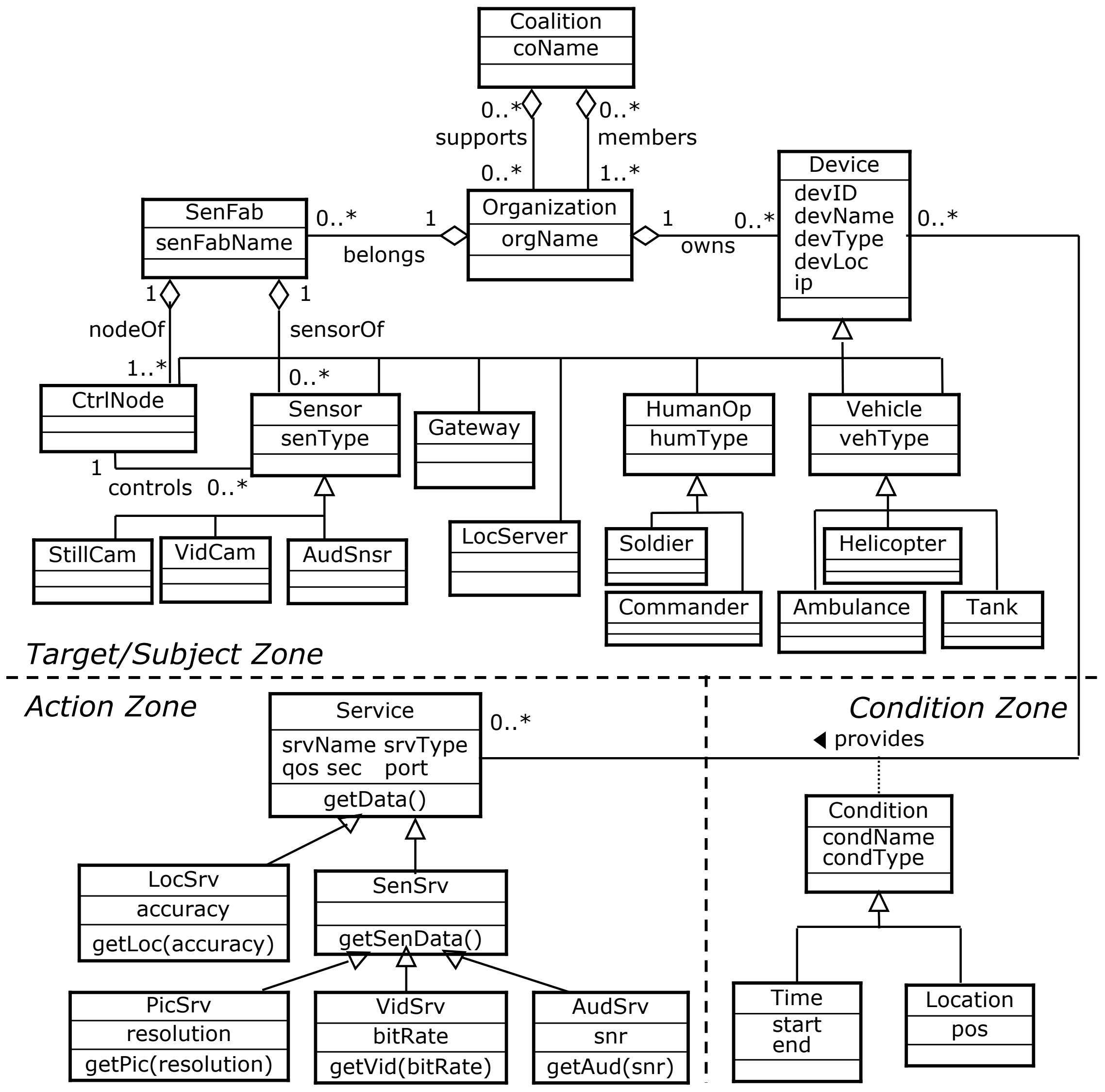


Figure 4.2: A UML class diagram for a subset of the US policy domain.

Figure 4.2 illustrates a UML diagram for a subset of the US policy domain. Each class in the UML diagram represents a group of objects stored in the knowledge database. A class also has properties, such as attributes and methods. For example, the *Device* class has five attributes named *devID*, *devName*, *devType*, *devLoc* and *ip* automatically inherited by all its child classes. Child classes can also define their own attributes and methods.

A line connecting two classes defines an association between them. Three types of associations are supported in our policy scenario: *generalization* (i.e., IS-A relationship) describing relationship between parent and child classes; *aggregation* (or *composition*) representing relationship between aggregate (or whole) class and part class; and regular associations not falling into the first two categories. Each association, except for generalization, is named uniquely, and can have their own attributes to form an class. For example, association class *Condition* describes additional information on service provision (i.e., association named *provides*) between class *Device* and class *Service* with two attributes *condName* and *condType*.

To facilitate policy representation, we group classes into different zones. Instances from the *target zone* define targets in such policies; those from the *subject zone* represent subjects. These two zones coincide in our policy scenario as illustrated in Figure 4.2. The *action zone* describes actions that a target can take. Finally, the association class *Condition* captures additional constraints on service provision.

We construct domain knowledge using logic representation from the UML class diagram defined previously. Six types of definitions are maintained. Each class in the UML diagram is defined using predicate *class(C)*, where C is a class name.

$$class(device). \quad class(service). \quad class(condition).$$

Instances of individual classes are described by predicate *obj(O, C)*, where O is an object name and C is the name of class that O belongs to.

$$obj(sc1, device). \quad obj(pic1, service). \quad obj(time1, condition).$$

Predicate *att(O, X, Y)* represents an attribute named X with value Y for instance O.

$$att(sc1, devName, sc1). \quad att(sc1, devType, stillCam).$$

Predicate *assType(X, C_1, A, C_2)* defines an association of type X (*agg* for aggregation, *comp* for composition and *reg* for regular association), identifiable by an unique name A between two classes C_1 and C_2. Service provision (i.e., *provides*) is a regular association between *device* class and *service* class.

$$assType(agg, organization, members, coalition).$$
$$assType(reg, device, provides, service).$$

The actual relationship between two instances O_1 and O_2 is described using predicate *ass(X, O_1,* *A, O_2)* that takes instances rather than classes as arguments.

$$ass(agg, us, members, ita). \quad ass(agg, uk, members, ita).$$
$$ass(agg, sc1, owns, us). \quad ass(reg, sc1, provides, pic1).$$

Unlike other associations, a generalization is defined as *isa(C_1, C_2)* between a pair of child class C_1 and parent class C_2:

$$isa(sensor, device). \quad isa(stillCam, sensor).$$
$$isa(senSrv, service). \quad isa(picSrv, senSrv).$$

In addition, attribute *port* (i.e. port number) from class *service* is often determined by other attributes of the same class as follows:

$$\mathcal{F}(att_1,\ \ldots,\ att_n) = port$$

For example, port number for web service is determined by its security feature such that regular http traffic goes through port 80 and encrypted https traffic goes through port 443.

4.1.2 A Logic-based Abstract Policy Language

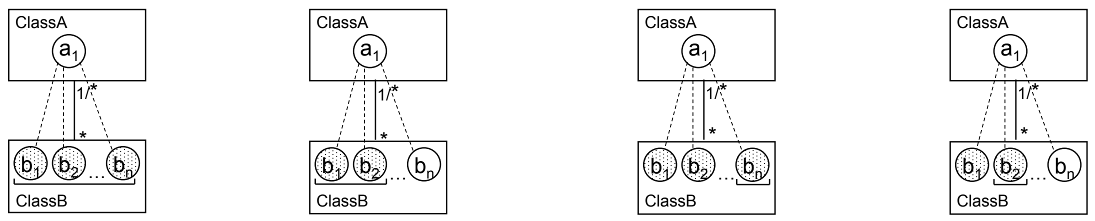
To assist policy refinement, we introduce a logic-based abstract language, intended to serve as a generic formal language which multiple policy languages can be translated into and out of during the refinement process. Therefore a network service policy written in an arbitrary policy specification language can be translated into the logic-based abstract form and refined into low-level rules using our refinement scheme.[2]

The language grammar written in Backus–Naur Form (BNF) is summarized in Figure 4.3. Meta-symbol | specifies multiple choices. Optional items are enclosed in [and]. Repetitive items (zero or more times) are enclosed in { and }, and ; is the termination symbol. Double quotes surrounding terminal characters (i.e., $\langle\ \rangle$ { } ( ) , . Tar′ Srv′ Sub′) are omitted for better readability. Comparison operators ($=$, $\neq$, $>$, $\geq$, $<$, $\leq$), logical connectives ($\neg$, $\wedge$, $\vee$, $\rightarrow$) and quantifiers ($\forall$, $\exists$) are integrated into our language to provide expressiveness. Terms in italic are ground *Instance*, *Class*, *Attribute*, *Association* names and association types from the UML description and the knowledge database.

[2]A language parser is required for each high-level policy specification language to enable the translation, which is out of the scope of this dissertation and can be left for future work.

```
 policy  ::=  auth sign ;
   auth  ::=  ⟨ Q Sub , Q Perm [ , Cond ] ⟩ ;
   Perm  ::=  ⟨ Q Tar , Q Srv ⟩ ;
    Sub  ::=  Sub′ ∈ { O | Exp } ;
    Tar  ::=  Tar′ ∈ { O | Exp } ;
    Srv  ::=  Srv′ ∈ { O | Exp } ;
   Cond  ::=  C_d | C_cf | Cond ∧ Cond ;
      O  ::=  Instance ;
      C  ::=  Class ;
      X  ::=  AssType ;
    Att  ::=  Attribute ;
    Ass  ::=  Association ;
      Q  ::=  ∀ | ∃ ;
   sign  ::=  + | − ;
     op  ::=  = | ≠ | > | ≥ | < | ≤ ;
      s  ::=  O . Att | a ;
      c  ::=  s op s ;
  C_att  ::=  c | ¬ C_att | C_att ∧ C_att | C_att ∨ C_att ;
      F  ::=  Tar′ | Srv′ | Sub′ ;
      e  ::=  F . Att ;
      f  ::=  e op e ;
   C_cf  ::=  f | ¬ C_cf | C_cf ∧ C_cf | C_cf ∨ C_cf ;
      d  ::=  obj ( O , C ) ∧ C_att ;
    C_d  ::=  d | C_d ∧ C_d | C_d ∨ C_d ;
    l_c  ::=  Q O ( ass ( X , O , Ass , O ) [ ∧ C_att ] → C_att ) ;
    l_o  ::=  Q O ( ass ( X , O , Ass , O ) [ ∧ C_att ] → [ C_att ∧ ] ;
      L  ::=  l_c | ¬ L | L ∧ L | L ∨ L ;
  C_ass  ::=  L | l_o C_ass ) | ¬ C_ass | C_ass ∧ C_ass | C_ass ∨ C_ass ;
    Exp  ::=  obj ( O , C ) [ ∧ C_att ] [ ∧ C_ass ] ;
```

Figure 4.3: Grammar for a logic-based abstract policy language.



$(a)\ \forall O' ass(O,A,O') \rightarrow \cdots$ $(b)\ \forall O'(ass(O,A,O') \wedge C\_att) \rightarrow \cdots$ $(c)\ \exists O' ass(O,A,O') \rightarrow \cdots$ $(d)\ \exists O'(ass(O,A,O') \wedge C\_att) \rightarrow \cdots$

Figure 4.4: Quantification for association.

A *policy* consists of an authorization rule *auth*, a sign to indicate positive authorization $+$ (permitting a service) and negative authorization $-$ (prohibiting a service). Each *auth* rule is a tuple $\langle Sub,\ Perm,\ Cond\rangle$ of three fields: Sub is the subject of an authorization policy; $Perm$ is further defined as a tuple $\langle Tar,\ Srv\rangle$ that represents a service Srv provided by target Tar; $Cond$ denotes an optional condition field. More specifically, Sub defines that a refined subject Sub' is an object O satisfying predicate Exp, where O belongs to class C (i.e. $obj(O,\ C)$) from *subject zone* satisfying both attribute constraints (C_att) and association constraints (C_ass). Tar and Srv are defined in a similar way respectively. Quantifier Q appears preceding with Sub, $Perm$, Tar and Srv for expressiveness. $Cond$ is a logic expression on condition element d and cross-field attribute constraints C_cf.

Our language supports compound constraints by defining C_att as an arbitrary propositional composition of constraint element c, including negation ($\neg$) given that op is closed under negation. Each constraint element c compares two sub-expressions of form s, where s is either an instance attribute $O.Att$ or a constant a. We also support cross-field attribute constraints C_cf that compares attributes of different fields (i.e., Sub', Tar' and Srv'). The condition C_att is limited to the object and class in the field where they appear.

C_ass defines the association constraints held for an object O. Expression l_c, the basic building block for any compound association constraints, describes a closed relationship between two objects that traverse exactly one link (association). Quantifiers are required for one-to-many or many-to-many associations with scope of the entire expression l_c. Since association name Ass uniquely defines the two end classes, the class definition for O' is omitted. Notice that l_c contains two optional attribute constraints: C_att appearing before "$\rightarrow$" further restricts the selection of O'; whereas C_att' after "$\rightarrow$" specifies properties held for selected O'. Thus l_c states that for all objects (or exists one object) O' associated with object O through Ass with property C_att held, C_att'

must also hold for those O'. Figure 4.4 depicts the four cases. Shaded nodes represent instances that satisfy $ass(X, O, Ass, O')$ with optional attribute constraints C_att on O'. Underlined nodes are instances further selected by the quantifier. The four cases are able to describe any subset of objects O' associated with O. L defines an arbitrary propositional composition of l_c, as O can be associated with objects through multiple associations. Consider the following C_ass associate constraint *[an organization that is a member organization of a coalition named ita with all its sensor fabric located at west quad, or it is a supporting organization of the same coalition]*.

$$\begin{aligned} C\_ass \equiv\ & (\exists O'\ (ass(agg,\ O,\ members,\ O') \rightarrow (O'.coName = ita)) \wedge \\ & \forall O''\ (ass(agg,\ O,\ belongs,\ O'') \rightarrow (O''.strLoc = westquad))) \vee \\ & (\exists O'\ (ass(agg,\ O,\ supports,\ O') \rightarrow (O'.coName = ita))) \end{aligned}$$

Unlike l_c, l_o is an open link between O and O' that allows O' to be further associated with other objects in a recursive manner. The last element on a path of consecutive association constraints must be L for C_ass to terminate properly.

$Cond$ is an arbitrary propositional composition of condition element d, excluding negation. Each condition element d is of the form $obj(O,\ C) \wedge C\_att$, where O is an instance of class C from the *condition zone* satisfying attribute constraints C_att.

Authorization policies written in natural language with a constrained lexicon and syntax designed for policy expression can be translated into our language. We assume an automated process that accomplishes the translation. Therefore, our policy refinement scheme starts from an authorization policy already written in our language. We translate initial policies (4.1) and (4.2) into our abstract language as follows:

[Each US device is permitted to access location information from one US location server with high quality, if communication is encrypted and both sides are from the same quad.]

$$\begin{aligned} & \Downarrow \\ & policy \equiv \langle \forall Sub,\ \forall\langle \exists Tar,\ \forall Srv\rangle,\ Cond\rangle + such\ that, \\ & Sub \equiv Sub' \in \{O \mid obj(O, device) \wedge (\forall O'\ (ass(agg, O, owns, O') \rightarrow O'.orgName = us))\} \\ & Tar \equiv Tar' \in \{O \mid obj(O, locServer) \wedge (\forall O'\ (ass(agg, O, belongs, O') \rightarrow O'.orgName = us))\} \\ & Srv \equiv Srv' \in \{O \mid obj(O, locSrv) \wedge (O.qos = high \wedge O.sec = crypto)\} \\ & Cond \equiv Tar'.devLoc = Sub'.devLoc \end{aligned} \tag{4.3}$$

[Devices belonging to non-US organizations for coalition X are prohibited from querying sensor data from any US sensor fabric with high quality between 9am and 5pm.]

$$
\begin{aligned}
&\Downarrow\\
&policy \equiv \langle \forall Sub,\ \forall \langle \forall Tar,\ \forall Srv \rangle,\ Cond \rangle - such\ that,\\
&Sub \equiv Sub' \in \{O \mid obj(O, device) \wedge (\forall O'\ (ass(agg, O, owns, O') \rightarrow ((O'.orgName \neq us) \wedge\\
&\qquad\qquad (\exists O''\ (ass(agg, O', members, O'') \rightarrow O''.coName = ita) \vee\\
&\qquad\qquad \exists O'''\ (ass(agg, O', supports, O''') \rightarrow O'''.coName = ita))))\}\\
&Tar \equiv Tar' \in \{O \mid obj(O, senFab) \wedge (\forall O'\ (ass(agg, O, belongs, O') \rightarrow O'.orgName = us))\}\\
&Srv \equiv Srv' \in \{O \mid obj(O, senSrv) \wedge (O.qos = high)\\
&Cond \equiv obj(O, time) \wedge (O.start = 9am \wedge O.end = 5pm)
\end{aligned}
\tag{4.4}
$$

4.1.3 Centralized Policy Refinement Scheme

We now describe the centralized policy refinement scheme in detail. The inputs of the refinement process are authorization policies written in our logic-based abstract language, and the outputs are low-level enforceable rules.

Definition 4.1.1 *We define the following transitive closure on generalization (IS-A relationship):*

$$
\begin{aligned}
&isa\_trans(C, C') \leftarrow isa(C, C')\\
&isa\_trans(C, C'') \leftarrow isa(C, C'),\ isa\_trans(C', C'')
\end{aligned}
\tag{4.5}
$$

Similarly, we define transitive closure on predicate $obj(O, C)$:

$$
\begin{aligned}
&obj\_trans(O, C) \leftarrow obj(O, C)\\
&obj\_trans(O, C') \leftarrow obj(O, C),\ isa\_trans(C, C')
\end{aligned}
\tag{4.6}
$$

In UML representation, the difference between *aggregation* and *composition* is subtle. Aggregation is more like a *has-a* relationship (also known as weak-aggregation); composition is more like a *part-of* relationship (also known as strong-aggregation).

Definition 4.1.2 *We define the following transitive closure on aggregation and composition asso-*

ciations:

$$\begin{aligned}
&ass\_trans(ac, O, Ass, O') \leftarrow ass(agg, O, Ass, O').\\
&ass\_trans(ac, O, Ass, O') \leftarrow ass(comp, O, Ass, O').\\
&ass\_trans(ac, O, Ass + Ass', O'') \leftarrow ass(agg/comp, O, Ass, O'),\\
&\qquad\qquad\qquad\qquad\qquad\qquad ass\_trans(ac, O', Ass', O'').
\end{aligned} \tag{4.7}$$

where $Ass + Ass'$ indicates that object O is associated with object O'' that traverses an aggregation (weak or strong) link Ass and a path Ass' in sequence.

Definition 4.1.3 *Association "provides" describing service provision is treated as a regular association between services and their providers. We define transitive closure on predicate ass for provides as follows:*

$$\begin{aligned}
&ass\_trans(reg, T, provides, V) \leftarrow ass(reg, T, provides, V).\\
&ass\_trans(reg, T', provides, V) \leftarrow ass(reg, T, provides, V),\\
&\qquad\qquad\qquad\qquad\qquad\qquad ass\_trans(ac, T, Ass, T').\\
&ass\_trans(reg, T, provides, V') \leftarrow ass(reg, T, provides, V),\\
&\qquad\qquad\qquad\qquad\qquad\qquad ass\_trans(ac, V', Ass, V).
\end{aligned} \tag{4.8}$$

The above definition states that target object T' transitively provides service V if T' is an aggregate of object T and T provides V. Similarly, target object T transitively provides service V' if V' is a part of service object V provided by T.

Definition 4.1.4 *Let policy be an authorization policy written in the logic-based abstract policy language. A refinement rule is an expression:*

$$\begin{aligned}
&policy \equiv \langle Q_S Sub,\ Q_P \langle Q_T Tar,\ Q_V Srv\rangle,\ Cond\rangle \pm\\
&\Downarrow ref\\
&policy' \equiv \langle Sub',\ Tar',\ Srv',\ Cond'\rangle \pm
\end{aligned} \tag{4.9}$$

that translates this higher-level policy into a rule policy′ at the lowest level through a gradual refinement. Positive and negative authorization signs are preserved automatically. There are many argument values of policy′ to satisfy the refinement rule (4.9). *The selection of policy′ is determined by quantifiers $Q_S \ldots Q_V$ in the policy expression, and will be discussed in detail during the refinement process.*

The centralized refinement process consists of two successive phases (see Figure 4.1):

1. A *policy transformation phase*, that transforms policies written in the logic-based abstract language to access control tuples by querying the pre-constructed knowledge database using refinement rule (4.9);

2. A *policy composition phase*, that composes policy rules at the lowest level from access control tuples obtained previously.

4.1.3.1 Policy Transformation

The implementation of refinement rule (4.9) consists of six successive steps. It starts with policies written in our abstract language so one can easily adapt it to another policy domain.

Permission Refinement A *permission*, i.e., $perm(T, V)$, defines a service V provided by a target T. Given the target (Tar) and service (Srv) expressions specified in our logic-based abstract language, we get a set of permissions using rule (4.10):

$$\begin{aligned}
&refPerm(\overbrace{obj(T, C_T), C\_att_T, C\_ass_T}^{Tar}, \overbrace{obj(V, C_V), C\_att_V, C\_ass_V}^{Srv}, C\_cf, perm(T, V)) \leftarrow \\
&\quad obj\_trans(T, C_T), \\
&\quad checkAttConst(T, C\_att_T), \\
&\quad checkAssConst(T, C\_ass_T), \\
&\quad obj\_trans(V, C_V), \\
&\quad checkAttConst(V, C\_att_V), \\
&\quad checkAssConst(V, C\_ass_V), \\
&\quad ass\_trans(reg, T, provides, V), \\
&\quad checkCFConst(C\_cf, perm(T, V)).
\end{aligned} \tag{4.10}$$

such that, T is an object transitively belonging to target class C_T, satisfying attribute constraints C_att_T and association constraints C_ass_T; V is an object transitively belonging to service class C_att_V, satisfying attribute constraints C_att_V and association constraints C_ass_V. Besides, target T transitively provides service V. Each resulting permission also satisfies any cross-field constraint specified in C_cf comparing attributes from Tar' and Srv'. The resulting set of permissions are further selected using Eq. (4.13) based on quantifiers Q_T, Q_V and their relative order.

Subject Refinement Subject refinement finds all the subjects S that transitively belong to class C_S and satisfy attribute constraints C_att_S and association constraints C_ass_S based on rule (4.11).

$$\begin{aligned}
&refSub(\overbrace{obj(S,C_S), C\_att_S, C\_ass_S}^{Sub}, sub(S)) \leftarrow \\
&\quad obj\_trans(S,C_S), \\
&\quad checkAttConst(S, C\_att_S), \\
&\quad checkAssConst(S, C\_ass_S).
\end{aligned} \tag{4.11}$$

Access Refinement Access refinement generates a set of access predicates $acc(S, perm(T,V))$ by computing the Cartesian product of permission set $\{perm(T,V)\}$ and subject set $\{sub(S)\}$ using rule (4.12). The resulting access predicates must also satisfy corresponding cross-field constraints specified in C_cf. Similarly, the set of accesses are further selected based on the value of Q_S, Q_P and their relative order using Eq. (4.13).

$$\begin{aligned}
&refAcc(C\_cf, acc(S, perm(T,V))) \leftarrow \\
&\quad perm(T,V), \\
&\quad sub(S), \\
&\quad checkCFConst(C\_cf, acc(S, perm(T,V))).
\end{aligned} \tag{4.12}$$

Quantification Refinement Since the two quantifications $\forall$ and $\exists$ are not commutative, we have all together six different cases. We define $P = \{(x,y)$: value pairs that make predicate $p(X,Y)$ true by assigning variables $X = x$ and $Y = y\}$. Quantification refinement selects elements from P based on the value of quantifiers Q_1 and Q_2 for X and Y respectively. The order of quantifiers also matters. In Eq. (4.13), $\mathcal{R}$ is a non-deterministic function that returns a maximal set of refined value pairs P' for the initial set of value pairs P given the combination of quantifiers Q_1 and Q_2.[3]

[3]The non-determinism can be restricted by other semantic considerations that select, for example, the most appropriate target to perform a service in the current situation.

Notation $p.X$ $(p.Y)$ returns the X (Y) value of an element p.

$$\mathcal{R}(Q_1, Q_2, P),\ where\ P = \{(x,y)\}$$
$$= \begin{cases} P' = P & \text{if } \forall X \forall Y; \\ P' \subseteq P \wedge |P'| = 1 & \text{if } \exists X \exists Y; \\ P' \subseteq P \wedge \forall i \forall j (i \neq j \wedge p_i, p_j \in P' \Rightarrow p_i.X \neq p_j.X) & \text{if } \forall X \exists Y; \\ P' \subseteq P \wedge \forall i \forall j (i \neq j \wedge p_i, p_j \in P' \Rightarrow p_i.Y \neq p_j.Y) & \text{if } \forall Y \exists X; \\ P' \subseteq P \wedge \forall i \forall j (i \neq j \wedge p_i, p_j \in P' \Rightarrow p_i.X = p_j.X \wedge p_i.Y \neq p_j.Y) & \text{if } \exists X \forall Y; \\ P' \subseteq P \wedge \forall i \forall j (i \neq j \wedge p_i, p_j \in P' \Rightarrow p_i.Y = p_j.Y \wedge p_i.X \neq p_j.X) & \text{if } \exists Y \forall X; \end{cases} \tag{4.13}$$

Granularity Refinement Given a set of access predicates after quantification refinement, the goal of granularity refinement is to traverse all the aggregation and composition associations for each field and produce the actual low-level objects that appear in the enforceable rules. Note that here we want the low level target Tar' and low level service Srv' that are directly associated through predicate ass on service provision.

$$\begin{aligned} & refGran(acc(Sub', perm(Tar', Srv'))) \leftarrow \\ & \quad acc(S, perm(T, V)), \\ & \quad ass(reg, Tar', provides, Srv'), \\ & \quad ass\_trans(ac, Tar', \_, T), \\ & \quad ass\_trans(ac, Srv', \_, V), \\ & \quad ass\_trans(ac, Sub', \_, S). \end{aligned} \tag{4.14}$$

Condition Refinement Since $Cond$ is an arbitrary propositional composition of condition element d, in rule (4.15), each $d \equiv obj(O, C) \wedge C\_att$ is refined to a list of condition instances connected using disjunctions, that belong to condition class C and satisfy attribute constraints C_att. Logic connectives among condition elements are automatically preserved. Notice that cross-field constraints C_cf have already been refined in previous steps.

$$\begin{aligned} & Cond \equiv d \mid C\_d \wedge C\_d \mid C\_d \vee C\_d \\ & \Downarrow \text{refCond} \\ & Cond' \equiv \vee_i^{i \geq 1} O_i \mid C\_d' \wedge C\_d' \mid C\_d' \vee C\_d' \end{aligned} \tag{4.15}$$

Examples Following our previous policy examples, we apply refinement rule (4.9) on policies (4.1) and (4.2) to query the pre-constructed knowledge database and produce the following results:

$$\begin{aligned} &\mathit{policy} \equiv \langle \forall Sub,\ \forall\langle \exists Tar,\ \forall Srv\rangle,\ Cond\rangle+ \\ &\Downarrow \text{ref} \\ &\mathit{policy}' \equiv \langle sc1,\ ls1,\ loc1,\ \emptyset\rangle+ \end{aligned} \tag{4.16}$$

$$\begin{aligned} &\mathit{policy} \equiv \langle \forall Sub,\ \forall\langle \forall Tar,\ \forall Srv\rangle,\ Cond\rangle- \\ &\Downarrow \text{ref} \\ &\mathit{policy}' \equiv \langle ls3,\ sc1,\ pic3,\ time1\rangle- \end{aligned} \tag{4.17}$$

Rule (4.16) refines policy (4.1) into a low-level rule *policy′* saying that US still camera $sc1$ is allowed to access location service $loc3$ (high accuracy with mandatory encryption) provided by US location server $ls1$. There are many other *policy′* satisfying rule (4.16) but rule (4.13) ensures that each subject is allowed to access one location server, i.e., $\langle sc1,\ ls2,\ loc1,\ \emptyset\rangle+$ is not a valid refinement if $\langle sc1,\ ls1,\ loc1,\ \emptyset\rangle+$ already exists. Similarly, *policy′* in rule (4.17) is one possible refinement of policy (4.2), saying that UK location server $ls3$ is prohibited to access picture service $pic3$ (high resolution with mandatory encryption) provided by US still camera $sc1$ at given time $time1$ (i.e., $9am - 5pm$).

4.1.3.2 Policy Composition

It is often the case that access control tuples generated from the *policy transformation* phase cannot be directly implemented because their syntax may not be understood by low-level devices. Thus the goal of *policy composition* is to produce low-level policies from these tuples. This step is highly language-dependent, because the final output is a set of low-level rules written in a policy language specification determined by the choice of underlying enforcement mechanism. As a concrete example, we demonstrate how to compose access control lists (ACLs) from the results of policy transformation using rule (4.18):

$$\begin{aligned} &\mathit{policy}' \equiv \langle Tar',\ Srv',\ Sub',\ Cond'\rangle\pm \\ &\Downarrow \text{ACL} \\ &acl(Tar') = \langle Sub',\ \pm Act',\ Cond'\rangle \end{aligned} \tag{4.18}$$

where $policy'$ is a refined policy produced by transformation rule (4.9), $acl(Tar')$ denotes an access control list on object Tar'. The operation field Act' is defined as a method provided by the object class C of Tar'. This method may take zero or more attributes of Tar' as parameters, such that $Act' = C.method(Tar'.Att_1,\ \ldots,\ Tar'.Att_n)$, where $n \geq 0$ and $obj(Tar', C)$. The positive or negative authorization sign is placed in front of Act' to indicate whether certain operation is allowed to performed or not. Alternatively, only positive authorization rules are maintained in ACLs. Hence any operation that is not explicitly granted is prohibited. Finally, the condition field $Cond'$ is required only if the implementation support complex ACLs with additional constraints. For example, we generate ACLs for refined rules (4.16) and (4.17) respectively.

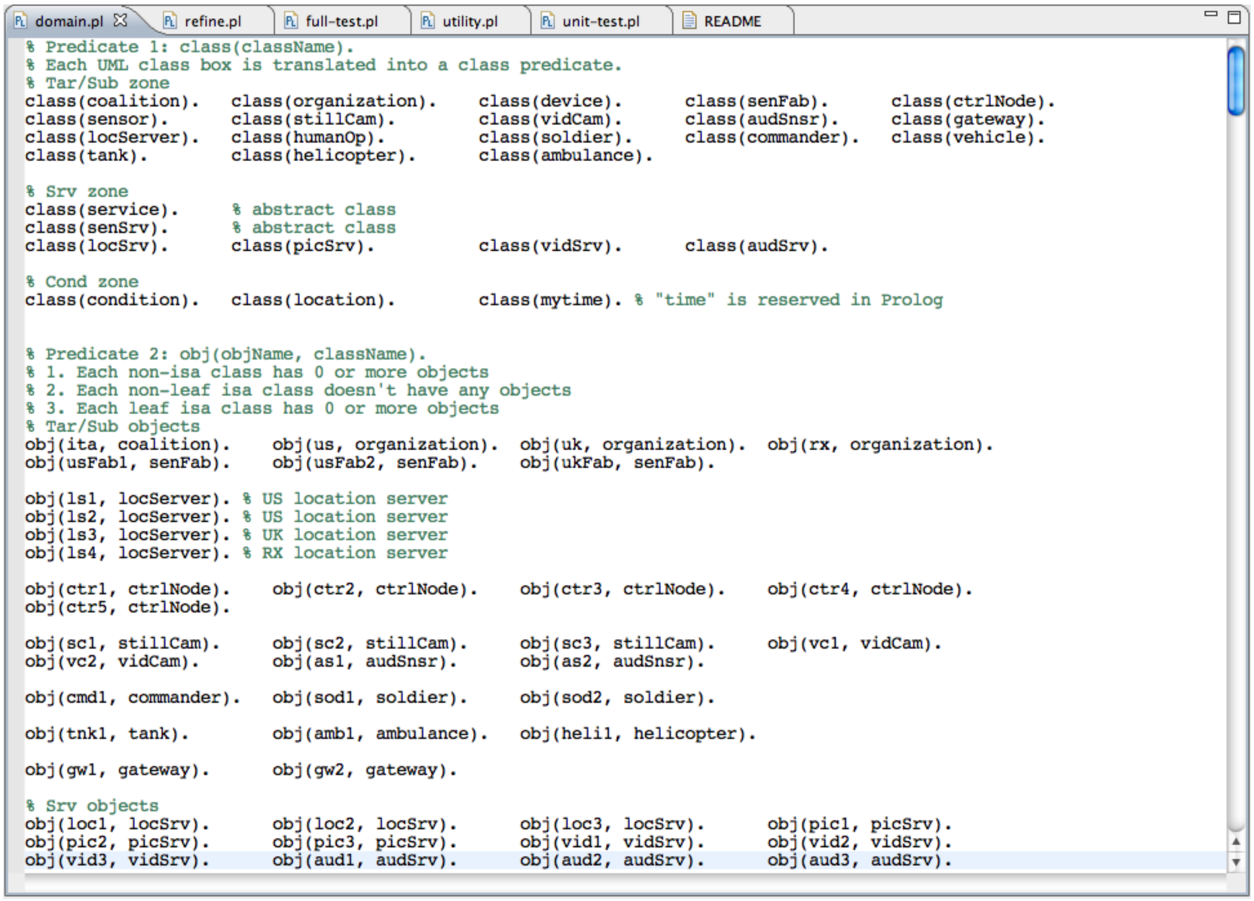
$$\begin{aligned} & policy' \equiv \langle sc1,\ ls1,\ loc1,\ \emptyset\rangle + \\ & \Downarrow \text{ACL} \\ & acl(ls1) \equiv \langle sc1,\ +getLoc(loc1.accuracy),\ \emptyset\rangle \end{aligned} \tag{4.19}$$

$$\begin{aligned} & policy' \equiv \langle ls3,\ sc1,\ pic3,\ time1\rangle - \\ & \Downarrow \text{ACL} \\ & acl(sc1) \equiv \langle ls3,\ -getPic(pic3.resolution),\ time1\rangle \end{aligned} \tag{4.20}$$

4.1.3.3 Implementation

We choose Prolog [Clocksin and Mellish, 1984], a popular general purpose logic programming language, for a reference implementation of the policy server in Figure 4.1. We use an open source implementation called SWI-Prolog [swi, 2011] together with its plugin ProDT [pro, 2012] developed for Eclipse [ecl, 2012].

Domain knowledge stored in the knowledge database is represented using Prolog facts (see Figure 4.5 for an implementation screenshot), and the refinement rule set is implemented using Prolog rules (see Figure 4.6 for an implementation screenshot). Policy transformation process (i.e., permission refinement, subject refinement, access refinement, quantification refinement, granularity refinement and condition refinement) is implemented following closely the definition of rule 4.10 - 4.15.. We also demonstrate two policy composition mechanisms for ACLs and ROFL firewall policies (see Chapter 5 for details on ROFL firewalls) respectively. The refinement results are



```
% Predicate 1: class(className).
% Each UML class box is translated into a class predicate.
% Tar/Sub zone
class(coalition).   class(organization).    class(device).      class(senFab).      class(ctrlNode).
class(sensor).      class(stillCam).        class(vidCam).      class(audSnsr).     class(gateway).
class(locServer).   class(humanOp).         class(soldier).     class(commander).   class(vehicle).
class(tank).        class(helicopter).      class(ambulance).

% Srv zone
class(service).     % abstract class
class(senSrv).      % abstract class
class(locSrv).      class(picSrv).          class(vidSrv).      class(audSrv).

% Cond zone
class(condition).   class(location).        class(mytime). % "time" is reserved in Prolog


% Predicate 2: obj(objName, className).
% 1. Each non-isa class has 0 or more objects
% 2. Each non-leaf isa class doesn't have any objects
% 3. Each leaf isa class has 0 or more objects
% Tar/Sub objects
obj(ita, coalition).    obj(us, organization).  obj(uk, organization).  obj(rx, organization).
obj(usFab1, senFab).    obj(usFab2, senFab).    obj(ukFab, senFab).

obj(ls1, locServer). % US location server
obj(ls2, locServer). % US location server
obj(ls3, locServer). % UK location server
obj(ls4, locServer). % RX location server

obj(ctr1, ctrlNode).    obj(ctr2, ctrlNode).    obj(ctr3, ctrlNode).    obj(ctr4, ctrlNode).
obj(ctr5, ctrlNode).

obj(sc1, stillCam).     obj(sc2, stillCam).     obj(sc3, stillCam).     obj(vc1, vidCam).
obj(vc2, vidCam).       obj(as1, audSnsr).      obj(as2, audSnsr).

obj(cmd1, commander).   obj(sod1, soldier).     obj(sod2, soldier).

obj(tnk1, tank).        obj(amb1, ambulance).   obj(heli1, helicopter).

obj(gw1, gateway).      obj(gw2, gateway).

% Srv objects
obj(loc1, locSrv).      obj(loc2, locSrv).      obj(loc3, locSrv).      obj(pic1, picSrv).
obj(pic2, picSrv).      obj(pic3, picSrv).      obj(vid1, vidSrv).      obj(vid2, vidSrv).
obj(vid3, vidSrv).      obj(aud1, audSrv).      obj(aud2, audSrv).      obj(aud3, audSrv).
```

Figure 4.5: Knowledge database implementation in Prolog.

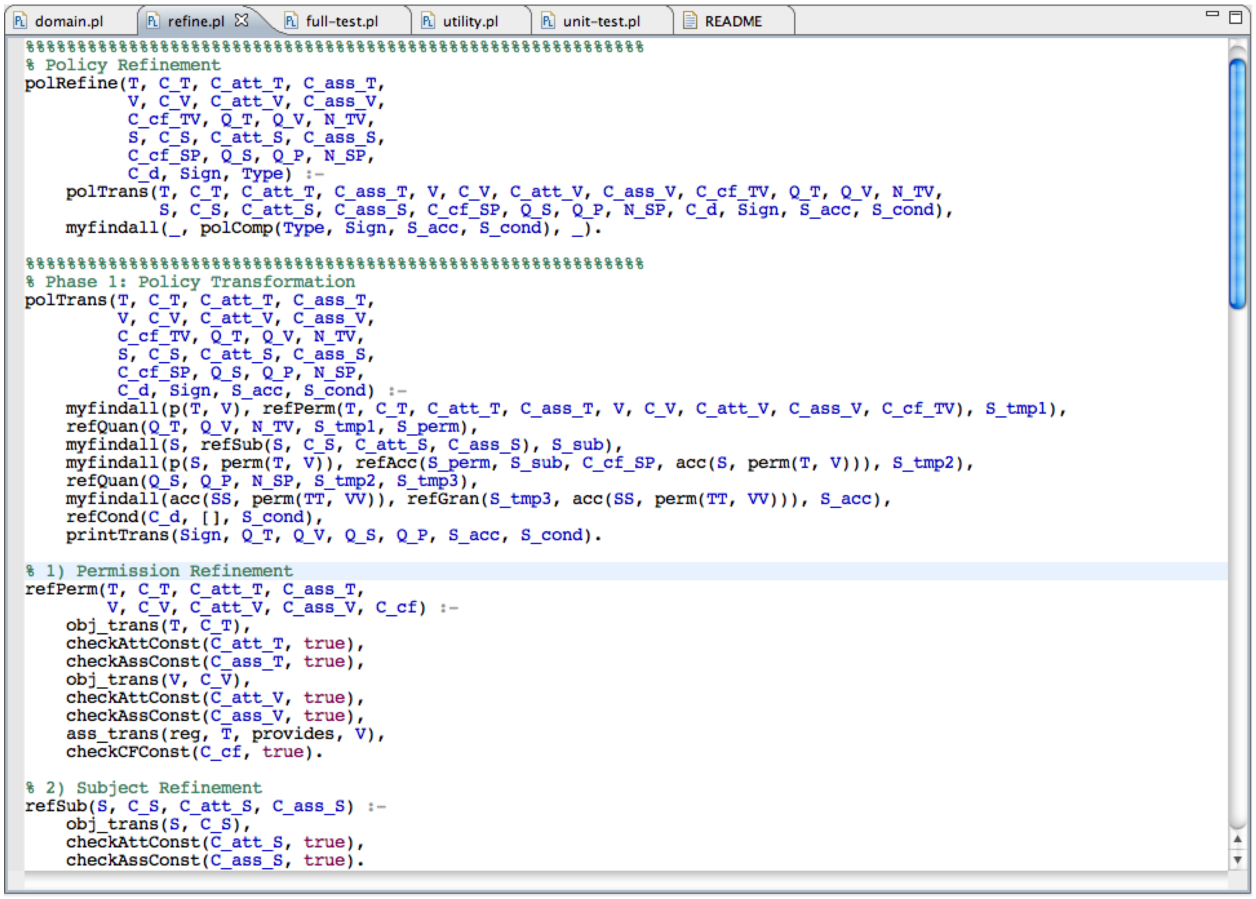
illustrated in the aforementioned examples. A comprehensive study on performance analysis is left for future work.

4.1.4 Policy Updates

The proposed policy refinement solution must cope with policy updates when the knowledge database changes as system devices and objects may join or leave the policy domain.[4] In order to generate consistent policies, knowledge database $\mathcal{D}$ must be conflict-free with the following requirements enforced:

1. The definition of class and associations among classes is self-contained, such that: if $isa(C, C') \in$

[4] We will not discuss the situations when new policies are added into the repository. This kind of incremental refinement is left for future work.



```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Policy Refinement
polRefine(T, C_T, C_att_T, C_ass_T,
          V, C_V, C_att_V, C_ass_V,
          C_cf_TV, Q_T, Q_V, N_TV,
          S, C_S, C_att_S, C_ass_S,
          C_cf_SP, Q_S, Q_P, N_SP,
          C_d, Sign, Type) :-
    polTrans(T, C_T, C_att_T, C_ass_T, V, C_V, C_att_V, C_ass_V, C_cf_TV, Q_T, Q_V, N_TV,
             S, C_S, C_att_S, C_ass_S, C_cf_SP, Q_S, Q_P, N_SP, C_d, Sign, S_acc, S_cond),
    myfindall(_, polComp(Type, Sign, S_acc, S_cond), _).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Phase 1: Policy Transformation
polTrans(T, C_T, C_att_T, C_ass_T,
         V, C_V, C_att_V, C_ass_V,
         C_cf_TV, Q_T, Q_V, N_TV,
         S, C_S, C_att_S, C_ass_S,
         C_cf_SP, Q_S, Q_P, N_SP,
         C_d, Sign, S_acc, S_cond) :-
    myfindall(p(T, V), refPerm(T, C_T, C_att_T, C_ass_T, V, C_V, C_att_V, C_ass_V, C_cf_TV), S_tmp1),
    refQuan(Q_T, Q_V, N_TV, S_tmp1, S_perm),
    myfindall(S, refSub(S, C_S, C_att_S, C_ass_S), S_sub),
    myfindall(p(S, perm(T, V)), refAcc(S_perm, S_sub, C_cf_SP, acc(S, perm(T, V))), S_tmp2),
    refQuan(Q_S, Q_P, N_SP, S_tmp2, S_tmp3),
    myfindall(acc(SS, perm(TT, VV)), refGran(S_tmp3, acc(SS, perm(TT, VV))), S_acc),
    refCond(C_d, [], S_cond),
    printTrans(Sign, Q_T, Q_V, Q_S, Q_P, S_acc, S_cond).

% 1) Permission Refinement
refPerm(T, C_T, C_att_T, C_ass_T,
        V, C_V, C_att_V, C_ass_V, C_cf) :-
    obj_trans(T, C_T),
    checkAttConst(C_att_T, true),
    checkAssConst(C_ass_T, true),
    obj_trans(V, C_V),
    checkAttConst(C_att_V, true),
    checkAssConst(C_ass_V, true),
    ass_trans(reg, T, provides, V),
    checkCFConst(C_cf, true).

% 2) Subject Refinement
refSub(S, C_S, C_att_S, C_ass_S) :-
    obj_trans(S, C_S),
    checkAttConst(C_att_S, true),
    checkAssConst(C_ass_S, true).
```

Figure 4.6: Refinement rule set implementation in Prolog.

$\mathcal{D}$, then $class(C), class(C') \in \mathcal{D}$; if $assType(X, C, A, C') \in \mathcal{D}$, then $class(C)$, $class(C') \in \mathcal{D}$.

2. The definition of instances and associations among them is self-contained, such that: if $obj(O, C) \in \mathcal{D}$, then $class(C) \in \mathcal{D}$, $att(O, Att_i, V_i) \in \mathcal{D}$, for all Att_i of $class(C)$; if $ass(X, O, A, O')$ $\in \mathcal{D}$, then $class(C)$, $class(C')$, $assType(X, C, A, C')$, $obj(O, C)$, $obj(O', C') \in \mathcal{D}$.

3. The definition of service instances cannot stay alone without their service providers: if $obj(O, C) \in \mathcal{D}$ and $class(C)$ is from the *service zone*, then $obj(O', C') \in \mathcal{D}$ such that $class(C')$ is from the *target zone*, and $ass(reg, O', Ass, O) \in \mathcal{D}$, where Ass is an association describing service provision.

Standard techniques like [Kowalski *et al.*, 1987; Grefen and Widom, 1996] for constraint verification and integrity checking on knowledge database can be applied there. We implement those techniques using logic programming.

From the system point of view, adding new objects (or classes) or removing existing ones only affects knowledge database not the rest of the policy server (Figure 4.1). Table 4.1 summarizes the changes one needs to perform for operations in the first column, where Y means the modification is mandatory, − implies optional, and N means not required.

Operation	***class***	***isa***	***obj***	***att***	***assType***	***ass***
Add/Remove Tar/Sub O	N	N	Y	Y	N	–
Add/Remove Tar/Sub C	Y	–	N	N	–	N
Add/Remove Srv O	N	N	Y	Y	N	Y
Add/Remove Srv C	Y	–	N	N	Y	N
Add/Remove Cond O	N	N	Y	Y	N	–
Add/Remove Cond C	Y	–	N	N	–	N

Table 4.1: Update of knowledge database upon operations

Clearly, adding or removing target (or subject) objects only affects predicate *obj* and *att*. It might affect associations, such as aggregation, composition, etc. On the other hand, adding or removing target (or subject) classes only affects the class definition *class*, and definition on associations among classes (i.e. *isa* and *assType*) may be updated as well. Update on service objects (or classes) is handled in a similar way except that predicates *assType* and *ass* must be updated to enforce requirement 3) discussed in previous subsection. Updates on condition objects (or classes) is handled the same as target objects (or classes).

4.2 Distributed Refinement for Access Control Policies

The centralized refinement scheme makes an assumption that the refinement process takes place in a single place, i.e., the central policy server, with all the information on the managed systems available as a whole. This assumption no longer holds in a distributed environment where a centralized knowledge database is not available, and each sub-system maintain its specific security needs while being compliant with the global security requirement. More specifically, distributed policy refinement poses the following additional challenges:

1. Because a global knowledge database to assist policy refinement is no longer available, the construction of information models for managed systems as well as the refinement process itself must be distributed and made locally.

2. The decentralized scheme may yield conflicting policies. Thus policy refinement must be interwoven with conflict analysis to ensure that the refined rules are always consistent with existing ones.

4.2.1 Revisiting Access Control Policies

A general policy definition adopted in [Sloman, 1994] considers policies as rules governing the behavior choices of a system. The term policy in the context of security management refers to rules that specify permissions and prohibitions in managed systems. These rules are called authorization policies, which regulate the access control to protected data and resources.[5] An authorization policy states that a *subject* is permitted (or prohibited) to perform an *action* on a *target* if certain *condition* holds. They are considered *target based* such that there is an access handler associated with the target object which enforces the policy and decides whether an action should be permitted or not.[6]

Unlike the centralized refinement approach, where high-level policies are transformed into low-level access control tuples of ground terms, given the large number of objects managed in distributed systems, it is impractical to specify policies for individual objects, i.e., ID-based access control rules are not sufficient. Two alternatives have been proposed to address this issue. *Management domain* approach [Sloman, 1994] that specifies policies for a collection of managed objects which have been explicitly grouped together for the purpose of management, and *Role-based access control* (RBAC) model [Sandhu *et al.*, 1996; Ferraiolo *et al.*, 2001] that groups authorizations to roles and assign roles to individual subjects. In this work, we adopt the more general management domain approach, where subjects and targets are specified in terms of Domain Scope Expressions. It naturally reflects the organizational or geographical distribution of the managed systems.

[5]These two terms – authorization and access control – are often used interchangeably in the context of security policy management.

[6]Subject is not trusted to regulate access in this case.

Authorization policies are intended to have immediate applicability to allow or disallow certain action. The target and action fields can be combined to form a *permission* that the subject is allowed (or prohibited) to access, as presented in our centralized policy refinement solution [Zhao *et al.*, 2011]. This combination could be specified in terms of high-level access privilege, which is then further refined to a set of ordered or unordered low-level operations performed on target objects. However, high-level access privilege and low-level operations are relative concepts. The distinction between them may depend on the context. For example, in one situation the "disk backup" action is defined as a standard low-level operation, while in other situations it might be a high-level access privilege such that the system administrator has to perform several operations (e.g., copy data to a backup disk, encrypt the disk, and put it in a protected storage place), and would need appropriate permissions for each such action in order to achieve this goal.

The condition field specifies constraints to restrict the applicability of an authorization policy. It is defined as a boolean expression referring to global attributes, such as time and action, or object attributes for subject and target. Constraint expressions on global attributes are usually evaluated at run time; while those on object attributes can be applied to select a set of subjects and targets satisfying the specified attribute constraints.

4.2.2 System Overview

Policy-based management of distributed systems cannot be centralized in a single policy center, but must be distributed to reflect the organizational or geographical distribution of the systems being managed [Sloman, 1994]. The key strategy to a policy refinement solution is the decomposition of policy components, including subject decomposition, target decomposition and action decomposition respectively. Subject/target decomposition has been studied specifically in the centralized refinement scheme. Action decomposition was proposed in [Craven *et al.*, 2010b]. In a fully distributed refinement scenario, partially-refined policies can propagate from one policy center to another, where local domain expertise and refinement patterns can be applied to further decompose them into low-level mechanism rules. On the other hand, separation of management is the key to scale better in large-scale distributed systems. By distributing refinement responsibilities to local policy centers, we take advantage of the decentralized structure to avoid any administration and performance bottleneck. We thus define the following two types of policy servers in a distributed policy scenario:

- *Global Policy Server (GPS)* that maintains some degree of central knowledge for the entire distributed environment, and stores policies on overall security requirements that can be disseminated to other policy servers and further refined to low-level enforceable rules.
- *Local Policy Server (LPS)* that keeps track of local domain expertise and stores policies that are applicable to the local environment.

GPS aims at keeping track of similar needs for the entire system. It may have some high-level view of the managed systems in terms of shared domain knowledge from LPSs, but individual needs tend to be very different as specified by each LPS. Moreover, GPS and LPS are also relative concepts. One policy server that plays the role of a LPS with respect to policy servers at higher level of the organization hierarchies may be considered as a GPS for those at relatively lower level of the hierarchies. A typical distributed refinement scenario may contain one global policy server and multiple local policy servers organized in certain structure, that could reflect physical network connectivity, distributed application structure or hierarchical management structure (see Figure 4.8 for an example). Policy severs are able to communicate with each other so that global policies and partially-refined rules can propagate from one policy server to another.[7] Similar to our centralized approach, each policy server consists of three main components:

1. A *Knowledge Database* that maintains an information model of the managed systems;
2. A *Refinement Rule Set* that defines the refinement procedure and patterns in terms of rules;
3. A *Policy Repository* that stores policies written at different levels of abstraction.

4.2.2.1 Centralized vs. Distributed Refinement

Policy reasoning methodologies, such as policy refinement and policy analysis, are closely tied to a formal representation of the managed systems. Such an information model on domain expertise is maintained in the knowledge database of a policy server. The process of policy refinement, taking places in the policy repository, accepts a high-level policy P^{H} as input and outputs a low-level policy

[7]Connectivity patterns and communication methodologies among policy servers are out of the scope of this work.

P^{L} by applying the domain-specific knowledge and refinement rules and patterns. High-level and low-level policies are relative concepts in a distributed refinement scenario. The level of detail of the constructed information model directly determines how far a high-level policy can be refined toward low-level enforceable rules. For example, a partially-refined low-level policy disseminated from the global policy server may be considered a high-level policy from the point of view of a local policy server, such that it needs to be further refined to enforceable rules by applying more detailed local domain-specific knowledge, which is not available in the global policy server.

In a centralized refinement scheme, a central policy server maintains a comprehensive view of the entire managed systems in its knowledge database. Hence the complete policy refinement process can be carried out in a single place. On the other hand, in a distributed policy scenario, we identify the following two mechanisms for a distributed refinement solution to achieve better performance in terms of scalability and complexity:

- *Partial refinement* refers to the process of decomposing a relatively high-level policy into some intermediate rules that semantically carry more enforcement information but still cannot be directly implemented by low-level devices.

- *Policy distribution* refers to the process of disseminating policies from one policy server to another for further refinement. The distributed policies could be high-level ones or partially-refined rules.[8]

These two techniques combined would allow global policies (with respect to local policies) be specified at a single point in the distributed systems (e.g., a GPS) and get distributed to multiple locations depending on the connectivity patterns among policy servers. Partially refined rules, that reflect common security needs, can propagate from one GPS to many LPSs as the recipient domains do not have higher level knowledge necessary to refine global security goals.

In a distributed refinement scenario, to enable partial refinement, it is necessary that one policy server (i.e., the local domain) maintains at least some knowledge of the other domain (i.e., the foreign domain) in order to perform policy decomposition locally to some extent. If the local policy server knows nothing about the foreign domain, no refinement can be performed; at the other

[8]Note the difference between policy distribution and deployment, where the latter refers to dispatching fully refined low-level rules to individual enforcement points.

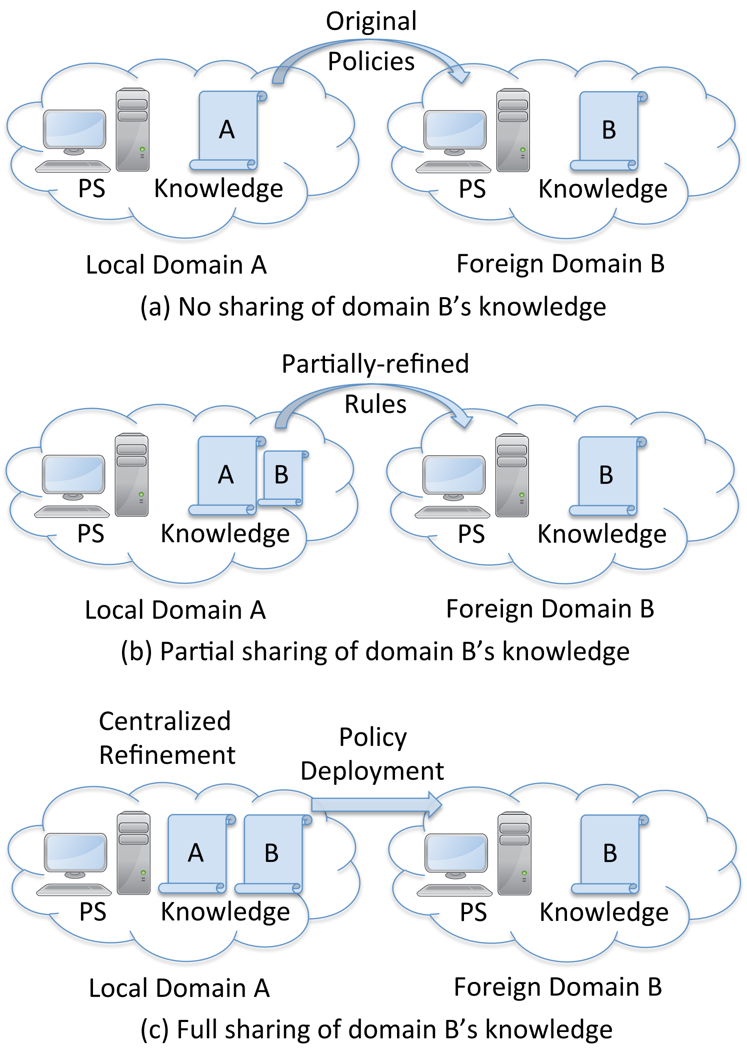


Figure 4.7: Policy distribution and partial refinement in a distributed policy scenario.

extreme, if it has a complete knowledge of the foreign domain, refinement can be accomplished locally, which is equivalent to centralized refinement. Hence the amount of shared knowledge of a foreign domain determines the applicability of partial refinement that can be performed locally. As illustrated in Figure 4.7, we identify the following three cases of knowledge sharing in a distributed policy scenario. Given a local domain A and a foreign domain B, if domain B shares no knowledge with domain A, high-level policies specified in A but also applicable to domain B will be distributed from A to B without any refinement (case a); if domain A maintains some partial knowledge of domain B, partially-refined policies based on shared knowledge can then be distributed to B for further refinement (case b); finally if full sharing of B's knowledge is present, a centralized refinement can be performed for policies applicable to B at local domain A and fully refined low-level rules are deployed to individual devices in domain B as discussed earlier in this chapter (case c).

How to draw a boundary for the degree knowledge sharing is highly system and application specific. It may also depend on corporate style and the type of policies. In some case, central headquarters knows comparatively little about the subsidiaries; in other cases, the division is between business units – all of the locations in a given business unit may have similar-enough needs that there's detailed central knowledge, but other units may be varied. For example, manufacturing units may need central policies for multiple locations, while research division has very different ones in different locations. Moreover, geographic legal issues may also play a role in determining the boundary of knowledge sharing. For instance, the EU policies prevent data from being sent to countries (e.g., the US) which do not have acceptable privacy rules.

4.2.2.2 A Distributed Policy Scenario

To assist our discussion on the distributed refinement scheme, we describe a simplified distributed policy scenario of an enterprise network in Figure 4.8. The head office maintains a global policy server (GPS), that is connected to three local policy servers (LPS) at individual regional offices. The LPS at Regional Office 1 further connects two lower-level LPSs for Branch Office A and B respectively. Policy servers are organized in hierarchies, and the arrows indicate the directions of policy distribution for this specific example.[9] Without loss of generality, we assume that Head Office has a partial view of the managed systems in all regional offices (i.e., case (b) in Figure 4.7), which reflects the commonalities among regional offices, such that partially-refined policies for specifying global security requirements can be distributed from GPS to LPSs for further refinement. On the other hand, Branch Office A and B share no knowledge with Regional Office 1 since each of them has their special needs (i.e., case (a) in Figure 4.7). Suppose a policy author from the Head Office specifies the following global access control policy written in natural language that needs to be

[9]Regular network communication links between office networks and devices are omitted for readability.

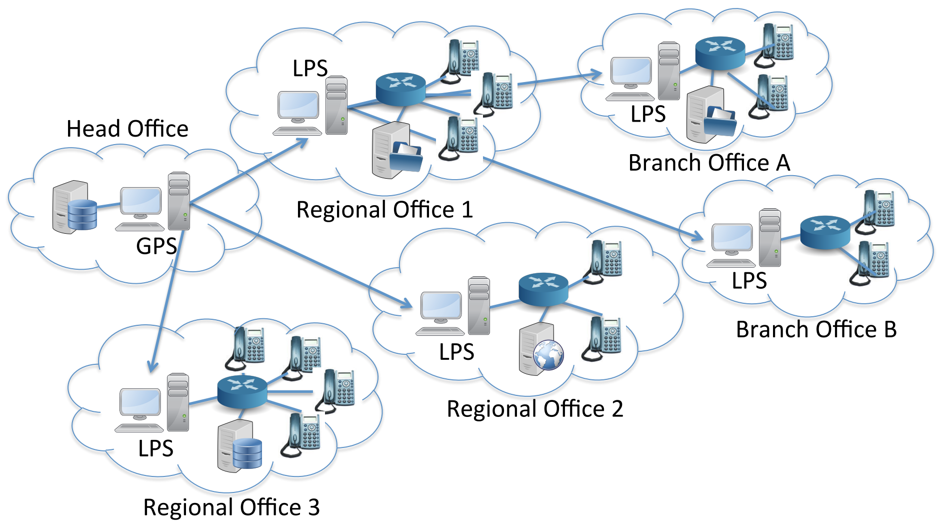


Figure 4.8: A scenario of distributed enterprise network.

translated into low-level enforceable rules using a distributed refinement scheme:

[Only local system administrators are allowed to access server machines for maintenance during scheduled hours between 5pm to 7pm everyday.] (4.21)

4.2.2.3 Formal System Representation

To keep track of the domain-specific knowledge stored in the knowledge database in each policy server, we need a formal representation for the managed system per domain. We simply use first-order logic predicate to represent static features of the system model, that can be translated from a UML class diagram constructed for the managed system. Predicate $\mathsf{class}(\mathsf{CName})$ represents a class with name CName and $\mathsf{obj}(\mathsf{OName}, \mathsf{CName})$ denotes a object OName belongs to class CName. Following the distributed policy scenario illustrated in Figure 4.8, the following predicates state that Regional Office 1 maintains a backup server named $\mathsf{bsRO1}$ belonging to class $\mathsf{backupSrv}$.

$$\mathsf{class}(\mathsf{backupSrv}),\ \mathsf{object}(\mathsf{bsRO1}, \mathsf{backupSrv})$$

Since system objects are organized in management hierarchies, also called management domains, we use predicate $\mathsf{domain}(\mathsf{DName})$ to represent a domain with name DName. For example, we have the following two domains for servers in general and backup servers more specifically:

$$\mathsf{domain}(/\mathsf{obj}/\mathsf{server}/),\ \mathsf{domain}(/\mathsf{obj}/\mathsf{server}/\mathsf{backupSrv}/)$$

Domain membership is described using predicate $\mathsf{isMember}(\mathsf{OName}, \mathsf{DName})$ stating that object OName is a direct or indirect object member of domain DName. For instance,

$$\mathsf{isMember}(\mathsf{bsRO1}, \mathsf{/obj/server/backupSrv/})$$

Since domains can be nested, we use the predicate $\mathsf{subdomain}(\mathsf{DName}_1, \mathsf{DName}_2)$ to represent the relationship between a child domain DName_1 and its parent domain DName_2. For example, backup server domain $\mathsf{obj/server/backupSrv}$ is a subdomain of server domain $\mathsf{/obj/server/}$.

$$\mathsf{subdomain}(\mathsf{/obj/server/backupSrv/}, \mathsf{/obj/server/})$$

Predicate $\mathsf{attr}(\mathsf{CName}, \mathsf{AName}, \mathsf{Type})$ describes an attribute AName of type Type for objects belonging to class CName. Object attribute can be represented using predicate $\mathsf{state}(\mathsf{OName}, \mathsf{AName}, \mathsf{Value})$ to reflect the object's current state. For example, each backup server has an attribute named quota of type int in Gigabyte, such that the backup server $\mathsf{bsRO1}$ from regional office 1 has a disk quota of $\mathsf{560}$ GB.

$$\mathsf{attr}(\mathsf{backupSrv}, \mathsf{quota}, \mathsf{int}),\ \ \mathsf{state}(\mathsf{bsRO1}, \mathsf{quota}, 560)$$

Since low-level policies are enforced by implementing operations defined for individual objects, we use the predicate $\mathsf{method}(\mathsf{CName}, \mathsf{MName}, \mathsf{Parms})$ to represent a method MName defined for class CName with a list of parameters enclosed in Parms. Objects belonging to class CName automatically inherit all the methods defined for this class. For example, objects belong to class $\mathsf{backupSrv}$ inherit the following three methods defined for this class:

$$\mathsf{method}(\mathsf{backupSrv}, \mathsf{copy}, [\mathsf{src}, \mathsf{dest}]), \mathsf{method}(\mathsf{backupSrv}, \mathsf{encrypt}, [\mathsf{alg}, \mathsf{key}])$$
$$\mathsf{method}(\mathsf{backupSrv}, \mathsf{store}, [\mathsf{room}])$$

System facts existing at time T can be represented in Event Calculus using base predicate $\mathsf{holdsAt}(\mathsf{F}, \mathsf{T})$, where F is a fluent describing properties that can vary over lifetime of a system. A fluent can be a predicate including a parameter specifying time, or a function dropping the time argument. For example, at time T_1, the following state of object $\mathsf{bsRO1}$ holds:

$$\mathsf{holdsAt}(\mathsf{state}(\mathsf{bsRO1}, \mathsf{quota}, 560), \mathsf{T}_1)$$

4.2.2.4 Policy Representation

Authorization policies can be further categorized into positive authorizations, which allows a subject to perform certain action on a target if condition satisfied, and negative authorizations, which prohibits the subject from performing that action. We use predicate $\mathsf{perm}(\mathsf{Tar}, \mathsf{Act})$ to represent an access permission, such that a subject Sub is allowed (or not allowed) to perform an action Act on a target Tar. The action field can be further refined to a list of operations, i.e., $\mathsf{op}(\mathsf{OpName}, \mathsf{Parm})$, with name OpName and parameters Parms (for purposes of function overloading). Grouping of the target and action fields into a permission implies their close relationship because authorization policies are target based. Sub and Tar are specified using Domain Scope Expressions of form $\mathsf{/DName_1/DName_2/.../[OName]}$ to denote all the direct and indirect members of a nested domain, and the last object name is required only if it refers to a specific member of that domain (e.g., $\mathsf{/obj/server/backupSrv/bsRO1}$). Access permission perm can be either high-level or low-level depending on the context. The action of performing authorized operations at time T may change the system state at time $\mathsf{T'}$ ($\mathsf{T'} > \mathsf{T}$).

Policy constraints are often specified as Boolean expressions to restrict the applicability of authorization rules. It can be defined as comparisons referring to global attributes, such as time, or attributes of managed objects (e.g., subject, target) as part of the selection expression within certain domain. In this work, we support a set of six basic comparison operators $\{=, \neq, >, \geq, <, \leq\}$. For example, to specify constraints saying that "backup server $\mathsf{bsRO1}$ has a larger capacity than server $\mathsf{bsRO2}$ at time T", we can write the following expression:

$$\begin{aligned}&\mathsf{holdsAt}(\mathsf{state}(\mathsf{bsRO1}, \mathsf{quota}, \mathsf{Q_1}), \mathsf{T}),\\&\mathsf{holdsAt}(\mathsf{state}(\mathsf{bsRO2}, \mathsf{quota}, \mathsf{Q_2}), \mathsf{T}),\\&\mathsf{Q_1} > \mathsf{Q_2}.\end{aligned}$$

Now, we can define positive (allow) and negative (deny) authorization policies as follows:

$$\mathsf{allow}(\mathsf{Sub}, \mathsf{perm}(\mathsf{Tar}, \mathsf{Act})) \leftarrow \mathsf{C_1}, \ldots, \mathsf{C_n}. \tag{4.22}$$

$$\mathsf{deny}(\mathsf{Sub}, \mathsf{perm}(\mathsf{Tar}, \mathsf{Act})) \leftarrow \mathsf{C_1}, \ldots, \mathsf{C_n}. \tag{4.23}$$

such that a subject Sub is granted (or denied) a permission perm to perform action Act on target Tar if a list of constraints $\mathsf{C_i}$ are satisfied. For example, policy 4.21 introduced in the enterprise scenario can be formally expressed as follows[10]:

$$\begin{aligned}
&\mathsf{allow}(/\mathsf{user}/\mathsf{sysAdmin}/, \mathsf{perm}(/\mathsf{obj}/\mathsf{server}/, \mathsf{maintain})) \leftarrow \\
&\quad \mathsf{isMember}(\mathsf{U}, /\mathsf{user}/\mathsf{sysAdmin}/), \\
&\quad \mathsf{isMember}(\mathsf{O}, /\mathsf{obj}/\mathsf{server}/), \\
&\quad \mathsf{holdsAt}(\mathsf{state}(\mathsf{U}, \mathsf{location}, \mathsf{L}_1), \mathsf{T}), \\
&\quad \mathsf{holdsAt}(\mathsf{state}(\mathsf{O}, \mathsf{location}, \mathsf{L}_2), \mathsf{T}), \\
&\quad \mathsf{L}_1 = \mathsf{L}_2, \\
&\quad \mathsf{5pm} \leq \mathsf{T} \leq \mathsf{7pm}.
\end{aligned} \tag{4.24}$$

4.2.3 Distributed Policy Refinement Scheme

In this section, we describe in detail a distributed refinement scheme consisting of the following four consecutive steps:

1. *Target Refinement* that decomposes the target domain into a group of subdomains based on the domain-specific knowledge stored in the knowledge database;
2. *Action Refinement* that decomposes the high-level action into a list of ordered or unordered low-level operations that can be directly implemented by target objects;
3. *Subject Refinement* that refines the subject domain into subsets if multiple policy enforcement points are available for load-balancing purposes;
4. *Constraints Refinement* that further refines the subject and target scope expressions into a set of ground objects if the early binding approach is preferred.

Unlike step 1, 3 and 4, that implement a form of set decomposition and selection mechanism, we introduce the concept of action decomposition patterns, as generalized in rule 4.25, to assist action refinement in step 2.

$$\mathsf{Tar.Act} \Rightarrow [\mathsf{Tar.MName}_1,\ \ldots,\ \mathsf{Tar.MName}_\mathsf{k}]^{\mathsf{O/U}} \tag{4.25}$$

[10]Here we assume a policy language parser is available to translate a policy written in natural language into our formal representation.

where a high-level action Act for target domain Tar can be decomposed to a list of ordered or unordered operations ($^{\mathsf{O/U}}$), such that each operation must be a valid method defined for the target object class. For example,

$$\begin{aligned} \mathsf{/obj/server/backupSrv/.maintain} \Rightarrow [&\mathsf{/obj/server/backupSrv/.copy}, \\ &\mathsf{/obj/server/backupSrv/.encrypt}, \\ &\mathsf{/obj/server/backupSrv/.store}]^{\mathsf{O}} \end{aligned} \tag{4.26}$$

Target Refinement Given the large number of system elements managed in distributed systems, it is impractical to specify policies in terms of individual objects. Therefore, we adopt the Domain Scope Expression to specify the subject and target fields, and the set of applicable subject and target objects will be determined during policy interpretation. This approach has another advantage such that objects can be added or removed from domains without having to change the affected policies. In addition, scope expressions may contain operators, such as $+$ (set union), $-$ (set difference), $\wedge$ (set intersection), etc., for expressiveness.

Given the formal system representation stored in each policy server (GPS or LPS), we apply domain-specific knowledge to assist target decomposition as described in rule 4.27:

$$\begin{aligned} &\mathsf{allow(Sub, perm(Tar, Act))} \leftarrow \mathsf{C_1, \ldots, C_n}. && (\mathsf{P}) \\ &\Downarrow \mathsf{refTar} \\ &\mathsf{allow(Sub, perm(Tar_1, Act))} \leftarrow \mathsf{C_1, \ldots, C_n}. && (\mathsf{P^T_1}) \\ &\qquad \ldots \\ &\mathsf{allow(Sub, perm(Tar_k, Act))} \leftarrow \mathsf{C_1, \ldots, C_n}. && (\mathsf{P^T_k}) \end{aligned} \tag{4.27}$$

where $\mathsf{Tar_1} + \cdots + \mathsf{Tar_k} = \mathsf{Tar}$, and the decomposition strategy is determined by the hierarchies of the managed objects.

If policy P is only applicable to the local domain (i.e., no policy distribution is required), then we must make sure that for each decomposed subdomain $\mathsf{Tar_i}$, an action decomposition pattern is available for the next step of action refinement; otherwise a missing action decomposition pattern indicates potential errors in system representation and/or policy specification. On the other hand, if P is also applicable to a foreign domain, we decompose Tar to produce partially-refined rules by applying the shared knowledge of the foreign domain. If the shared knowledge does not contain enough information for action refinement, then we distribute sub-policies P_i^T to remote policy servers where target refinement may be performed recursively for each P_i^T until a valid action decomposition pattern can be identified for each decomposed sub-domains.

Action Refinement For each sub-policy P_i^T obtained from the previous step, we further decompose the action field Act into a list of unordered or ordered operations by applying action decomposition patterns.

$$\begin{aligned}
&\mathsf{allow(Sub, perm(Tar, Act))} \leftarrow \mathsf{C}_1, \ldots, \mathsf{C}_n. && (\mathsf{P}_i^\mathsf{T})\\
&\Downarrow \mathsf{refAct}_1\\
&\mathsf{allow(Sub, perm(Tar, [op(OpName_1, Parms_1)]))} \leftarrow \mathsf{C}_1, \ldots, \mathsf{C}_n. && (\mathsf{P}_1^\mathsf{A})\\
&\quad\ldots\\
&\mathsf{allow(Sub, perm(Tar, [op(OpName_k, Parms_k)]))} \leftarrow \mathsf{C}_1, \ldots, \mathsf{C}_n. && (\mathsf{P}_k^\mathsf{A})
\end{aligned} \tag{4.28}$$

Rule 4.28 demonstrates the refinement of action Act into a list of unordered operations $\mathsf{op(OpName_i}$, $\mathsf{Parms_i)}$ expressed in each sub-policy P_i^A respectively, such that each P_i^A specifies a valid access authorization. For example, the action of access a file directory can be refined into a list of unordered operations: read, write and execute respectively, i.e., $\mathsf{perm(/home/zhao, fileAccess)} \Rightarrow \mathsf{op(read,}$ $\mathsf{[/home/zhao])}$, $\mathsf{op(write, [/home/zhao])}$, $\mathsf{op(execute, [/home/zhao])}$.

Alternatively, the action field Act can be decomposed to a list of ordered operations $\mathsf{OpName_1}, \ldots,$ $\mathsf{OpName_k}$ that must be performed in sequence as shown in rule 4.29.

$$\begin{aligned}
&\mathsf{allow(Sub, perm(Tar, Act))} \leftarrow \mathsf{C}_1, \ldots, \mathsf{C}_n. && (\mathsf{P}_i^\mathsf{T})\\
&\Downarrow \mathsf{refAct}_2\\
&\mathsf{allow(Sub, perm(Tar, [op(OpName_1, Parms_1),\ \ldots,\ op(OpName_k, Parms_k)]))} \leftarrow \mathsf{C}_1, \ldots, \mathsf{C}_n. && (\mathsf{P}^\mathsf{A})
\end{aligned} \tag{4.29}$$

Partial refinement only involves target refinement and action refinement. Partially-refined rules resulted from step 2 must be disseminated to corresponding foreign domains, as subject refinement can only be performed locally for the purposes of distributing access requests among multiple enforcement points.

Subject Refinement Similar to specifying target, the subject field is expressed using Domain Scope Expressions. In a distributed environment, a set of subject objects can be further refined into smaller subsets if multiple policy enforcement points (PEPs) are coordinated to distribute access requests for load balancing and traffic shaping purposes. Thus we define the following rules for subject refinement based on previous results:

$$\begin{aligned}
&\mathsf{allow}(\mathsf{Sub}, \mathsf{perm}(\mathsf{Tar}, [\mathsf{op}(\mathsf{OpName}, \mathsf{Parms})])) \leftarrow \mathsf{C}_1, \ldots, \mathsf{C}_\mathsf{n}. \qquad (\mathsf{P}_\mathsf{i}^\mathsf{A})\\
&\Downarrow \mathsf{refSub}_1\\
&\mathsf{allow}(\mathsf{Sub}_1, \mathsf{perm}(\mathsf{Tar}, [\mathsf{op}(\mathsf{OpName}, \mathsf{Parms})])) \leftarrow \mathsf{C}_1, \ldots, \mathsf{C}_\mathsf{n}. \qquad (\mathsf{P}_1^\mathsf{S})\\
&\qquad \ldots\\
&\mathsf{allow}(\mathsf{Sub}_\mathsf{m}, \mathsf{perm}(\mathsf{Tar}, [\mathsf{op}(\mathsf{OpName}, \mathsf{Parms})])) \leftarrow \mathsf{C}_1, \ldots, \mathsf{C}_\mathsf{n}. \qquad (\mathsf{P}_\mathsf{m}^\mathsf{S})
\end{aligned} \tag{4.30}$$

Rule 4.30 takes each sub-policy $\mathsf{P}_\mathsf{i}^\mathsf{A}$ resulted from unordered action refinement as input and decomposes the subject field into a list of subsets, such that $\mathsf{Sub}_1 + \cdots + \mathsf{Sub}_\mathsf{m} = \mathsf{Sub}$.

$$\begin{aligned}
&\mathsf{allow}(\mathsf{Sub}, \mathsf{perm}(\mathsf{Tar}, [\mathsf{op}(\mathsf{OpName}_1, \mathsf{Parms}_1), \ldots, \mathsf{op}(\mathsf{OpName}_\mathsf{k}, \mathsf{Parms}_\mathsf{k})])) \leftarrow \mathsf{C}_1, \ldots, \mathsf{C}_\mathsf{n}. \quad (\mathsf{P}^\mathsf{A})\\
&\Downarrow \mathsf{refSub}_2\\
&\mathsf{allow}(\mathsf{Sub}_1, \mathsf{perm}(\mathsf{Tar}, [\mathsf{op}(\mathsf{OpName}_1, \mathsf{Parms}_1), \ldots, \mathsf{op}(\mathsf{OpName}_\mathsf{k}, \mathsf{Parms}_\mathsf{k})])) \leftarrow \mathsf{C}_1, \ldots, \mathsf{C}_\mathsf{n}. \quad (\mathsf{P}_1^\mathsf{S})\\
&\qquad \ldots\\
&\mathsf{allow}(\mathsf{Sub}_\mathsf{m}, \mathsf{perm}(\mathsf{Tar}, [\mathsf{op}(\mathsf{OpName}_1, \mathsf{Parms}_1), \ldots, \mathsf{op}(\mathsf{OpName}_\mathsf{k}, \mathsf{Parms}_\mathsf{k})])) \leftarrow \mathsf{C}_1, \ldots, \mathsf{C}_\mathsf{n}. \ (\mathsf{P}_\mathsf{m}^\mathsf{S})
\end{aligned} \tag{4.31}$$

Similarly, rule 4.31 takes the sub-policy P^A resulted from ordered action refinement as input and refines the subject field into subsets. The decomposition among multiple subsets is determined by the load balancing policies of the managed systems. It is also possible to have overlapping subsets.

Constraints Refinement Constraints are specified to restrict the applicability of authorization policies in terms of Boolean expressions. Those expressions on global attributes, such as time, must be evaluated at run time upon receiving incoming access requests. Constraints can also be used to specify additional conditions on subject and target set selection. In the centralized refinement approach, we adopted the *early binding* approach that refines the subject and target expressions to ground objects for constructing access control tuples; whereas for a distributed refinement solution, we apply the domain scope expression that takes the alternative *late binding* approach to determine applicable objects at run time, as the domain membership and the state of an object can change dynamically especially in a distributed environment.

Refinement Updates In a distributed environment, objects may join or leave a domain dynamically. Since access control policies are target-based, we only focus on situations when target objects join or leave a domain that triggers domain membership changes. As target objects are specified using domain scope expressions, no additional refinement is necessary if the late binding approach is applied. Otherwise, if the early binding approach is preferred, only policies involving the modified target domains will be considered for re-refinement purposes.

Examples Given the sample policy 4.21 presented previously, we perform the distributed refinement process on policy 4.21 by applying the aforementioned steps.

1. *Target Refinement* The global policy server at the Head Office performs partial refinement to decompose the target domain into a list of subdomains based on shared knowledge from regional offices, such that the target domain /obj/server/ can be decomposed into two subdomains /obj/server/webSrv/ and /obj/server/backupSrv/ respectively.

2. *Action Refinement* Suppose the maintenance task performed on a web server is considered shared common knowledge, hence the refinement of action maintain on target domain /obj/server/webSrv/ can be performed at the GPS based on the available action decomposition patterns. Without loss of generality, let us assume action maintain is decomposed into a low-level standard operation maintainOp defined for class(webSrv). The decomposed rule can be distributed to LPSs at individual regional offices without the need of further refinement.

   However, since the GPS has no idea on how to perform maintenance task on backup servers, partially-refined rules from previous step will be distributed to each LPS for further refinement. Suppose we take Regional Office 1 as an example. High-level action maintain is refined to a list of ordered low-level operations [copy, encrypt, store] on target domain /obj/server/backupSrv/ at the LPS of Regional Office 1 based on the action decomposition pattern rule 4.26.

3. *Subject Refinement* The step of subject refinement can be omitted if there is only one enforcement point available for each enforceable rule.

4. *Constraints Refinement* Suppose the late binding approach is adopted in the enterprise scenario illustrated in Figure 4.8, such that constraint expressions will only be evaluated at run

time.

Therefore, the original high-level policy rule 4.21 specified as a global security requirement at the Head Office will be refined into rule 4.32 and 4.33 for Regional Office 1, such that:

$$
\begin{aligned}
&\mathsf{allow}(\mathsf{/user/sysAdmin/}, \mathsf{perm}(\mathsf{/obj/server/}, \mathsf{maintain})) \leftarrow \\
&\quad \mathsf{isMember}(\mathsf{U}, \mathsf{/user/sysAdmin/}), \\
&\quad \mathsf{isMember}(\mathsf{O}, \mathsf{/obj/server/}), \\
&\quad \mathsf{holdsAt}(\mathsf{state}(\mathsf{U}, \mathsf{location}, \mathsf{L}_1), \mathsf{T}), \\
&\quad \mathsf{holdsAt}(\mathsf{state}(\mathsf{O}, \mathsf{location}, \mathsf{L}_2), \mathsf{T}), \\
&\quad \mathsf{L}_1 = \mathsf{L}_2, \\
&\quad \mathsf{5pm} \leq \mathsf{T} \leq \mathsf{7pm}. \\
&\Downarrow \mathsf{refine} \qquad (4.32) \\
&\mathsf{allow}(\mathsf{/user/sysAdmin/}, \mathsf{perm}(\mathsf{/obj/server/webSrv/}, [\mathsf{op}(\mathsf{maintainOp}, [\mathsf{sName}])])) \leftarrow \\
&\quad \mathsf{isMember}(\mathsf{U}, \mathsf{/user/sysAdmin/}), \\
&\quad \mathsf{isMember}(\mathsf{O}, \mathsf{/obj/server/webSrv/}), \\
&\quad \mathsf{holdsAt}(\mathsf{state}(\mathsf{U}, \mathsf{location}, \mathsf{L}_1), \mathsf{T}), \\
&\quad \mathsf{holdsAt}(\mathsf{state}(\mathsf{O}, \mathsf{location}, \mathsf{L}_2), \mathsf{T}), \\
&\quad \mathsf{L}_1 = \mathsf{L}_2, \\
&\quad \mathsf{5pm} \leq \mathsf{T} \leq \mathsf{7pm}.
\end{aligned}
$$

$$\begin{aligned}
&\mathsf{allow}(\mathsf{/user/sysAdmin/}, \mathsf{perm}(\mathsf{/obj/server/backupSrv/}, [\mathsf{op}(\mathsf{copy}, [\mathsf{src}, \mathsf{dest}]), \\
&\qquad\qquad\qquad\qquad\qquad\qquad \mathsf{op}(\mathsf{encrypt}, [\mathsf{alg}, \mathsf{key}]), \\
&\qquad\qquad\qquad\qquad\qquad\qquad \mathsf{op}(\mathsf{store}, [\mathsf{room}])])) \leftarrow \\
&\quad \mathsf{isMember}(\mathsf{U}, \mathsf{/user/sysAdmin/}), \\
&\quad \mathsf{isMember}(\mathsf{O}, \mathsf{/obj/server/backupSrv/}), \\
&\quad \mathsf{holdsAt}(\mathsf{state}(\mathsf{U}, \mathsf{location}, \mathsf{L}_1), \mathsf{T}), \\
&\quad \mathsf{holdsAt}(\mathsf{state}(\mathsf{O}, \mathsf{location}, \mathsf{L}_2), \mathsf{T}), \\
&\quad \mathsf{L}_1 = \mathsf{L}_2, \\
&\quad \mathsf{5pm} \leq \mathsf{T} \leq \mathsf{7pm}.
\end{aligned} \tag{4.33}$$

4.2.4 Policy Conflict Analysis

There are two major categories of policy conflicts identified in policy-based distributed systems management [Lupu and Sloman, 1999], namely *modality conflicts* that may arise when two or more policies refer to the same subjects, targets and actions but with modalities of opposite sign, and *application-specfic conflicts* that refer to the inconsistency contained in the semantics of policies (also called semantics conflicts), which can be further categorized into the following four cases:

- *Conflict of duties* arises when the same subject is allowed to perform conflicting actions on the same target. For instance, the same manager can authorize and sign payment checks.
- *Conflict of interests* arises when the same subject is allowed to perform conflicting actions on different targets. For example, an investment bank advises one client company on investment decisions while performing consulting tasks for a competitor company.
- *Multiple managers* arises when different subjects perform incompatible operations on the same target. One example is to schedule maintenance task and some other application task on a database server, such that the two tasks cannot be performed simultaneously.
- *Self-management* may arise when the subject of one policy overlap with the target of another policy. For example, a manager is allowed to sign his own expenses.

Conflict detection techniques require overlapping checking on the essential fields of two candidate policies. Subject and target checking can be easily performed using basic set operations since they are specified as domain scope expressions. Constraints checking makes sure that the two conflicting policies could indeed invoke at the same time. However, checking on overlapping, conflicting or incompatible actions is a bit tricky. One operation specified at high-level of abstraction often provides little useful information when compared with another low-level operation. This observation implies that conflict analysis must be interwoven with policy refinement as the latter can translate high-level policies into low-level rules. We introduce the notion of an *atomic operation*, which cannot be further decomposed by applying action refinement technique. Therefore, action checking must be performed between atomic operations.

Without loss of generality, we illustrate the detection method using modality conflicts as an example. Given two policies P_1 and P_2, modality conflicts can be detected using rule 4.34:

$$\begin{aligned}
&\mathsf{conflictDect}(\mathsf{modConflict}, \mathsf{P}_1, \mathsf{P}_2, [\mathsf{Sub}, \mathsf{Tar}, \mathsf{Act}, \mathsf{C}]) \leftarrow \\
&\quad \mathsf{holdsAt}(\mathsf{allow}(\mathsf{Sub}_1, \mathsf{perm}(\mathsf{Tar}_1, \mathsf{Act}_1)), \mathsf{T}), \\
&\quad \mathsf{holdsAt}(\mathsf{deny}(\mathsf{Sub}_2, \mathsf{perm}(\mathsf{Tar}_2, \mathsf{Act}_2)), \mathsf{T}), \\
&\quad \mathsf{isOverlapDomain}(\mathsf{Sub}_1, \mathsf{Sub}_2, \mathsf{Sub}), \\
&\quad \mathsf{isOverlapDomain}(\mathsf{Tar}_1, \mathsf{Tar}_2, \mathsf{Tar}), \\
&\quad \mathsf{isOverlapAction}(\mathsf{Act}_1, \mathsf{Act}_2, \mathsf{Act}), \\
&\quad \mathsf{isOverlapConst}(\mathsf{C}_{\mathsf{P}_1}, \mathsf{C}_{\mathsf{P}_2}, \mathsf{C}).
\end{aligned} \tag{4.34}$$

It states that a positive authorization policy P_1 is conflicting with a negative authorization policy P_2 if they share overlapping subjects Sub, targets Tar and actions Act, and the two policies can be applied simultaneously under certain overlapping constraints C. The last parameter of predicate $\mathsf{conflictDect}$ reconstructs the conflicting part of the two rules.

We use rule 4.35 to define overlapping actions. Action Act_1 and Act_2 are first decomposed into lists of atomic operations using action decomposition patterns defined in the system representation. Then overlapping operations are checked between the two lists OpList_1 and OpList_2, such that any overlapped items are passed into OpList.

$$\begin{aligned}
&\mathsf{isOverlapAction}(\mathsf{Act}_1, \mathsf{Act}_2, \mathsf{OpList}) \leftarrow \\
&\qquad \mathsf{actionDecomp}(\mathsf{Act}_1, \mathsf{Tar}_1, \mathsf{OpList}_1), \\
&\qquad \mathsf{actionDecomp}(\mathsf{Act}_2, \mathsf{Tar}_2, \mathsf{OpList}_2), \\
&\qquad \mathsf{isOverlapOp}(\mathsf{OpList}_1, \mathsf{OpList}_2, \mathsf{OpList}).
\end{aligned} \tag{4.35}$$

Similar to detecting modality conflicts based on overlapping actions, application-specific conflicts can be detected by replacing function $\mathsf{isOverlapOp}$ in rule 4.35 with $\mathsf{isConflictOp}$ and $\mathsf{isIncompatibleOp}$ respectively.

Modality conflicts can be resolved using meta-policies to specify the precedence between two conflicting rules. Here we list a few examples of conflict resolution strategies based on precedence assignment using our policy representation model. We use predicate $\mathsf{conflictReso}(\mathsf{P}_1, \mathsf{P}_2)$ to indicate that policy P_1 is assigned a higher precedence over policy P_2 when a conflict is detected between them.

- *Positive authorization takes precedence* over negative authorization as described in meta-policy 4.36:

$$\begin{aligned}
&\mathsf{conflictReso}(\mathsf{P}_1, \mathsf{P}_2) \leftarrow \\
&\qquad \mathsf{conflictDect}(\mathsf{Type}, \mathsf{P}_1, \mathsf{P}_2, \mathsf{P}), \\
&\qquad \mathsf{holdsAt}(\mathsf{allow}(\mathsf{Sub}_1, \mathsf{perm}(\mathsf{Tar}_1, \mathsf{Act}_1)), \mathsf{T}), \\
&\qquad \mathsf{holdsAt}(\mathsf{deny}(\mathsf{Sub}_2, \mathsf{perm}(\mathsf{Tar}_2, \mathsf{Act}_2)), \mathsf{T}).
\end{aligned} \tag{4.36}$$

- *Negative authorization takes precedence* over positive authorization as described in meta-policy 4.37:

$$\begin{aligned}
&\mathsf{conflictReso}(\mathsf{P}_2, \mathsf{P}_1) \leftarrow \\
&\qquad \mathsf{conflictDect}(\mathsf{Type}, \mathsf{P}_1, \mathsf{P}_2, \mathsf{P}), \\
&\qquad \mathsf{holdsAt}(\mathsf{allow}(\mathsf{Sub}_1, \mathsf{perm}(\mathsf{Tar}_1, \mathsf{Act}_1)), \mathsf{T}), \\
&\qquad \mathsf{holdsAt}(\mathsf{deny}(\mathsf{Sub}_2, \mathsf{perm}(\mathsf{Tar}_2, \mathsf{Act}_2)), \mathsf{T}).
\end{aligned} \tag{4.37}$$

- *Authorization with more specific target expression takes precedence* over less specific one as described in meta-policy 4.38, where predicate $\mathsf{isMoreSpecific}(\mathsf{D}_1, \mathsf{D}_2)$ states that expression

D_1 is more specific than D_2 and the detailed implementation is highly specification dependent. In our adopted domain scope expression scenario, $\mathsf{isMoreSpecific}(\mathsf{D}_1, \mathsf{D}_2)$ is true if and only if predicate $\mathsf{isSubdomain}(\mathsf{D}_1, \mathsf{D}_2)$ holds.

$$\begin{aligned} &\mathsf{conflictReso}(\mathsf{P}_1, \mathsf{P}_2) \leftarrow \\ &\qquad \mathsf{conflictDect}(\mathsf{Type}, \mathsf{P}_1, \mathsf{P}_2, \mathsf{P}), \\ &\qquad \mathsf{isMoreSpecific}(\mathsf{Tar}_1, \mathsf{Tar}_2). \end{aligned} \tag{4.38}$$

- *Authorization with more specific subject expression takes precedence* over less specific one as described in meta-policy 4.39:

$$\begin{aligned} &\mathsf{conflictReso}(\mathsf{P}_1, \mathsf{P}_2) \leftarrow \\ &\qquad \mathsf{conflictDect}(\mathsf{Type}, \mathsf{P}_1, \mathsf{P}_2, \mathsf{P}), \\ &\qquad \mathsf{isMoreSpecific}(\mathsf{Sub}_1, \mathsf{Sub}_2). \end{aligned} \tag{4.39}$$

On the other hand, policy conflict resolution strategies for application-specific conflicts require a system expert to explicitly remove one authorization out of the two rules with conflicting or incompatible actions. For instance, given two conflict of duties rules, a system expert should explicitly eliminate one duty for the subject by removing one authorization rule (e.g., only allows a manager to authorize a payment check without being able to sign it).

Another obvious type of conflicts in a distributed scenario may arise between a global policy specifying overall security needs and a local policy applicable to the local domain. This kind of policy conflicts still falls into the aforementioned categories. Similarly, precedence can be assigned to a global policy if it is conflicting with a local rule to ensure overall security requirement in a distributed environment, and vice versa. For example, a global policy specifies limited Internet access permissions for all divisions in a company. However research groups may need more Internet access permissions, to investigate new services, which is an example of local policy overriding the global one. Conversely, there might be a new data retention policy from corporate headquarters, preventing dat deletions because of a new lawsuit, such that this new global policy should take precedence over existing local rules.

4.3 Discussion

In this Chapter, we described two policy refinement schemes: a centralized refinement solution for network service policies, and a distributed refinement solution for access control policies in general. Each of them focuses on different perspectives of the policy refinement problem. As our first attempt to solve the refinement problem, the centralized approach focuses on a subset of access control policies, i.e., service provision policies through communications networks; whereas the distributed approach considers access control policies in general. The centralized approach elaborates the main components of a refinement scheme from the system perspective: a *knowledge database*, a *refinement rule set* and a *policy repository*. Global Policy Servers (GPS) and Local Policy Servers (LPS) presented in the distributed approach also have a similar structure. In addition, policies written in different specification languages can be translated into and out of the logic-based abstract language to maximize the adaptability of the centralized refinement scheme. In the distributed solution, policy components (i.e., subjects, targets) are specified in terms of domain scope expressions that extends the ID-based access control approach adopted in the centralized solution. Moreover, the distributed solution emphasizes the needs for interweaving policy refinement with conflict analysis to maintain system functionality and integrity by ensuring that the refined rules are always consistent with existing ones.

As the first attempt to solve the distributed refinement problem, we discussed the necessity and benefits of two key techniques in this chapter, namely partial refinement and policy distribution. We have not defined partial refinement from the perspective formal system representation, as it is solely determined by the amount of shared knowledge between system domains. A formal representation for knowledge sharing and partial refinement, as well as an implementation and evaluation of the distributed refinement scheme are left for future work. Moreover, our current work focus on resolution techniques for modality conflicts facilitated by policy refinement. It will be interesting to study methodology for resolving application-specific conflicts that are more complicated and requires system expert intervention.

Although high-level policies and low-level rules are relative concepts, there are situations where a high-level policy cannot be fully refined to rules at the bottom of the system hierarchy. For example, if we don't understand the operations of the low-level mechanisms, high-level actions cannot be refined to implementable operations that the low-level devices can interpret. On the other hand, the correctness of decomposing objects and actions rely on the complete view of the managed systems. The process of decomposition produces subsets from the superset at each refinement step. Hence, if a complete view of the system is not available, the correctness of policy refinement is not guaranteed.

Part III

Application

Chapter 5

Distributed Enforcement of Firewall Policies

As a concrete application of policy delegation and policy refinement discussed in Chapters 3 and 4 respectively, this chapter describes a distributed enforcement mechanism for firewall policies, named ROFL – ROuting as the Firewall Layer. ROFL implements packet filtering by utilizing the underlying routing mechanism. It treats port number as part of the IP address in the routing system, and then turns every router along the path as a firewall. We provide a detailed analysis of the growth of routing tables caused by ROFL, and further evaluate its performance in mobile ad hoc networks (MANETs) using simulation with two ad hoc routing protocols: *Ad hoc On-Demand Distance Vector* (AODV) protocol, a reactive routing protocol using distance vector information, and *Optimized Link Stat Routing* (OLSR) protocol, a proactive routing protocol using link state information. Simulation results demonstrate that ROFL is an effective and efficient firewall mechanism especially for MANETs.

5.1 Firewalls as A Form of Routing

As motivated in Chapter 1, traditional firewalls and distributed firewalls each have their own advantages and shortcomings. Traditional firewalls make assumption on a fixed topology with insiders and outsiders, such that all the insiders are trusted. On the contrary, distributed firewalls push enforcement to every end host to remove any performance bottleneck and rely on the hosts to make appropriate and thus more secure decisions. However, the latter approach is problematic as it allows unwanted traffic flow all the way to destination before being discarded. This is not a problem from host penetration perspective, but it is problematic in bandwidth and in battery-constrained environment like a MANET.

Therefore, a hybrid solution utilizing enforcement delegation is proposed in Chapter 3, which allows unwanted traffic flows being dropped early along the routing path by selected delegates to save transmission power. By doing that, it reduces potential collisions among packet transmissions in MANETs, which often requires additional MAC layer efforts and collaborations to resolve. However, to balance the trade-off between cost and risk associated with policy enforcement, a global optimization algorithm is required to distribute and arrange policies in order to minimize overall cost for policy enforcement while maintaining the risk level below certain threshold. Such a centralized algorithm could be expensive and not scalable when the size of the network grows. To solve this problem, we treat firewalls as a form of routing and propose ROFL – ROuting as the Firewall Layer – to develop a unified approach of those two.

5.1.1 Basic ROFL Scheme

The ROFL (ROuting as the Firewall Layer) scheme is based on a simple notion: services — that are, port numbers — should be treated as part of the IP address in the routing system. If a certain service is not advertised to a particular network, no host on that network can reach it; the routing system will not deliver the packets. We thus use every router along the path as a firewall. There are many benefits to this scheme, especially in MANETs where battery power is limited. If unwanted packets are dropped very early, a lot of power can be saved by not transmitting those packets. A conventional routing advertisement is of the form

$$R = \{d/m, M\}$$

where d is an address prefix, m is a prefix length, and M is a routing metric. In ROFL, we extend the prefix field to include port number s representing a service:

$$R = \{d : s/m, M\}$$

For example, to permit access to an SMTP server and a web server at 192.0.2.42, a node would originate the routes

$$\{192.0.2.42 : 25/48, M_1\}, \{192.0.2.42 : 80/48, M_2\}$$

where M_1 and M_2 are the cost metrics.

Port-specific routing advertisements are handled just like any other routing advertisements: the source node and any intermediate routers do a longest-prefix match on the advertisement. If there is no matching route, the packet is dropped.

Blocking a port is more complex. Suppose we wish to permit references to all ports on 192.0.2.42 *except* port 80. While we could advertise $2^{16} - 1$ routes $\{192.0.2.42 : 0/48, M\}$, $\{192.0.2.42 : 1/48, M\}, \{192.0.2.42 : 79/48, M\}, \{192.0.2.42 : 81/48, M\}, \ldots, \{192.0.2.42 : 65535/48, M\}$, this would be seen as an unfriendly thing to do to a routing table. (This is the fully decorrelated form [Zhao and Bellovin, 2007]; clearly, one could advertise a shorter, correlated form, but it might still require a considerable number of prefix announcements.) Instead, we advertise an *infinity* route:

$$\{192.0.2.42 : 80/48, \infty\}$$

along with $\{192.0.2.42/32, M\}$. Infinity routes are entered into a router's *Forwarding Information Base* (FIB), but they are used as black hole (discard) entries. In effect, we treat the infinity route and the covering route to all services on the host as a correlated specification, compared with the fully decorrelated version given above, and the two forms must be equivalent.

5.1.2 Client Routes

A service-specific route in ROFL is suitable for use by servers. However, virtually all useful protocols require replies from servers to the clients, which in turn implies that there must be a route to client ports. Therefore, we allow client-like computers to advertise their non-server ports as service-specific routes. Current IANA standards[1] say that dynamic ports (i.e., those used for ordinary clients) should be in the range 49152-65535. For host h, then, all clients are in h : 49152/34; services are in h : 32768/34 or h : 0/33. (Note: 49,152 is $2^{15} + 2^{14}$, i.e., $1100\ 0000\ 0000\ 0000_2$.) Advertise an infinity route for those service prefixes, to block access to any services that aren't

[1] http://www.iana.org/assignments/port-numbers

specifically advertised via a /48. Then advertise a normal route to $h : 49152/34$ to permit access to all client ports on the machine. The few servers on higher-numbered ports can be covered by specific infinity routes. Alternatively, the full /48 can be announced, with specific infinity routes to block the few servers running.

A second scheme is for client machines to announce a /48 for client ports as they're used. This avoids the need for unused prefixes to exist when a client port is not in use. The third choice is to piggyback the routing announcement for the client's /48 on the initial SYN packet to the server. More precisely, send a special packet towards the server that contains both the /48 announcement and the initial packet for the server. Routers along the path from the client to the server install the appropriate reverse route, in effect nailing up a reverse circuit. The latter two alternatives are particularly useful in environments that use Network Address Translators (NATs). NATs need to know when to create externally-visible mappings for internal ports. As such, they benefit from explicit announcements from clients about which port numbers are in use.

Note that most machines are either clients or servers; there are comparatively few machines that act as both. The exception is for things like the DNS and infrequent outbound administrative emails from servers. It may, then, be advisable to select one of the above mechanisms based on the normal role of the machine.

5.1.3 Routing Metric

As always, our extended routing protocol requires a metric M that describes the "cost" of a path. Metric construction varies. For example, in BGP [Rekhter *et al.*, 2006], the AS hop count is used; in OSPF [Moy, 1998], the network administrators specify arbitrary metrics for each link.

Our needs are more complex. As discussed in the policy algebra framework in Chapter 3, we require both a cost metric $\mathcal{C}$ and a risk metric $\mathcal{R}$ in the scenario of policy enforcement delegation. Intuitively, "cost" represents the total expense of using a given link. It may reflect bandwidth, power limitations and the difficulty of replacing or recharging batteries, etc. "Risk", by contrast, is related to the safety of a particular transmission. A node may have been captured or otherwise compromised (a significant issue for MANETs operating in hostile territory); if so, it should not be used for the path. The risk metric captures the exposure of a node to such events. We define the cost to be the property of a link; risk is the property of a node. This is a reasonable approach on a MANET. As topologies change, the power consumed by a transmission will vary, thus affecting its cost. Similarly, in a battlefield environment the location of the front line may change, and with it the physical danger to any given node.

Our routing metric must handle both of these concepts. Furthermore, we wish for a scheme that can optimize both simultaneously, with the tradeoff between higher cost and lower risk made by local authorities. This tradeoff may vary over time; furthermore, it may be different for different services. The tradeoff between cost and risk is represented by a value γ, such that $0 \leq \gamma \leq 1$. The actual metric used for route calculations is thus $M = \gamma\mathcal{C} + (1 - \gamma)\mathcal{R}$.

While the precise sources of these values are not crucial, we do offer some plausible suggestions. We do suggest, however, that in general they should change no more frequently than the time required for the routing state to converge, otherwise routing loops could occur.

Link costs are best determined by the transmitting node. If nothing else, it has the best knowledge of its battery state and the power required to send a message. In MANETs, a meaningful approach may be to define link costs as proportional to the geographical distance between two nodes. In wireless communication, it is well-known that short-hop communications can improve the capacity of wireless, because short-range transmissions cause less interference than long-range transmissions. Hence, it is more preferable to choose a long path with many short-hop communications than a short path with a few of long-hop communications. This is best accomplished by using a sublinear relationship between distance and cost. Also, the link costs can be proportional to the number of neighbors of the transmitter; the more neighbors a node has, the more interference the transmitter will inflict on or receive from others.

Risk is more problematic. In combat situations, an enemy may prefer to capture certain military ranks. In such situations, nodes with lower-ranking soldiers may be evaluated as a lower risk. Often, nodes are the best judge of their own risk levels; in other situations, they may be centrally determined. In either case, the actual value used must be known by, and hence transmitted to, all nodes that need to calculate routing metric M for a given hop. Generally, this will be the node's immediate neighbors, as everyone else will simply use the value of M broadcast by the routing protocol. That said, in environments where γ changes considerably more rapidly than $\mathcal{C}$ and $\mathcal{R}$, it may be worthwhile transmitting $\mathcal{C}$ and $\mathcal{R}$ rather than M.

γ has the most intriguing properties. It could be set centrally; alternatively, each node could use its own values. Again, what is important is that all nodes agree on M for each link; this can be done either by only broadcasting M, or by broadcasting $\langle \mathcal{C}, \mathcal{R}, \gamma \rangle$ in each advertisement. Note that in the absence of aggregation points, metrics for a given service $d : s$ are never compared with those for another service: the longest-prefix match is always preferred, regardless of the difference in metric. Therefore, all of these values can be service-specific; in particular, γ could be advertised by the service originator $d : s$, and thus more properly be viewed as $\gamma_{d:s}$.

By folding cost and risk into a single value, the routing system automatically picks the optimum path between any two points. The total routing expense, then, is $\sum_{i,j} M_{d_i:s_j}$, the sum of the individual metrics. The total system expense, though, depends not just on the state of the routing system, but also on the traffic matrix. This is not known (or knowable) locally. Accordingly, we could in fact use $M = \Gamma\mathcal{C} + (1 - \Gamma)\mathcal{R}$, where Γ is a centrally-specified function of γ. By varying it, the overall system expense can be optimized.

5.1.4 Extensions of ROFL

There are a number of extensions of the ROFL scheme that may also prove to be useful. One is to route on IPsec SPIs instead of the port numbers, since port numbers are not visible to intermediate nodes for encrypted traffic. Routing on SPIs permits early drop of many fake packets. It is especially useful if fine-grained SPIs are used, as in distributed firewalls. Note that ROFL and cryptography are orthogonal issues. Cryptographic techniques provide confidentiality and integrity of communications; the goal of ROFL is not to replace cryptography, but rather to drop unwanted packets early to save transmission power.

A second extension is to separate port number from IP address, and instead pass a list of permitted or prohibited port numbers along with a prefix announcement. This would allow a single announcement (and hence FIB entry) to handle many services. More importantly, it would easily handle ROFL announcement for an entire network, such as $\{192.0.2.0/24, \{25, 80\}, M\}$. However, this would require an additional lookup step not part of destination prefix matching; additionally, it might conflict with the desire to route different services differently.

5.2 Policy Routing with ROFL

In common with most firewalls and routing protocols, ROFL makes its decisions based on destination addresses and port numbers. For wired networks, this is quite proper, since it is not possible to trust source addresses coming from beyond the firewall [Cheswick *et al.*, 2003]. In MANETs, where connectivity patterns are constantly changing, the situation is subtly different. While it remains true that the behavior of "untrusted" nodes (i.e., those not protected by the firewall) cannot be relied upon for security purposes, adding source address constraints to MANET rules have a second, and equally important function: they define the boundaries of the *policy region*. That is, source address rules define the boundary between the "inside" — the portion of the network protected by the firewall — and the outside. In a traditional wired network, the firewall itself is the boundary marker, with the network topology determining the inside and outside.

5.2.1 ROFL with Source Prefix Filtering

The basic ROFL scheme performs traffic filtering based on destination IP address and port number only. Therefore, we extend ROFL to include source prefix constraints in route announcement:

$$R = \{d : s/m, S, M\}$$

where S indicates a set of source address prefixes such that data traffic coming from those addresses is allowed to access service s advertised by destination d. More precisely, we define $S = \{p_1, p_2, ..., p_n\}$; each p_i $(0 \leq i \leq n)$ is a source address prefix. Hence no source prefix constraint is specified if $S = \phi$. (Alternately, we could simply set $S = \{0/0\}$, i.e., all addresses are accepted; for clarity in this paper, we prefer to distinguish between no source prefix constraints and one that happens to have no effect.) We remark that in some situations, it is possible to implement S as a Bloom filter [Bloom, 1970] on the set of source addresses or networks of a given prefix length. Bloom filters are a space-efficient data structure that can compress the representation of a set of members in a compact manner, albeit with some chance of false positives. Bloom filters are particularly useful in MANETs, where there is little topological structure and each allowed node may be identified by a flat address.

Source prefix constraint in ROFL controls the propagation of routing advertisements as well as the packet forwarding procedure. A service specific routing announcement will be passed to a node if and only if the node is allowed to access the advertised service. ROFL advertisements in this new scheme are handled similarly to any other routing announcements, except that another layer of checking needs to be performed based on the source address of a packet. The source node and any intermediate routers perform a longest-prefix match on the advertisement. The packet is forwarded if and only if a matching route is found and the source address in the packet header is contained in the source prefix constraints of that matching route; Otherwise, the packet is dropped.

The new scheme is able to implement many filtering functionalities provided by traditional packet filter firewalls. For example, to allow data traffic coming from source address p_1 to reach destination host d on port number s, we simply announce a routing advertisement $R = \{d : s/48, \{p_1\}, M\}$. Blocking certain traffic is a bit trickier. If p_1 is the only source address that is not allowed to access service s on host d, we could announce $R = \{d : s/48, S, M\}$ with $p_1 \notin S$. Otherwise, if d wants to completely block traffic on port s only, it announces $R = \{d : s/48, \infty\}$ together with $R' = \{d/32, M\}$. Recall that traditional firewalls allow the wild card $*$ to appear in any field of source address as well as destination address and port number. In our new ROFL scheme, $*$ in destination port number is equivalent to $d/32$ and $*$ in source address can be described as $S = \phi$. If $*$ appears in the destination address field, we can easily adjust the prefix length m to cover the corresponding subnet.

In the above scheme, route selection is based in part on the source address. Therefore the actual route announcement becomes a form of policy routing. There are two obvious approaches to inserting policy constraints into routing announcements. First, all relevant nodes along the route propagation paths could create or modify the policy statement, in accordance with some central policy. A better approach would be allowing only the node advertising the service — the route originator — to embed a source prefix constraint in routing announcement. Subsequent receivers of this route announcement should not alter the embedded policy statement. We suggest the second approach for a few reasons. First of all, it is the route originator that has the best knowledge of who is or is not allowed to access a certain service. Second, allowing intermediate routers to modify policy statements requires some kind of trust relationship established amongst them. More importantly, the first approach might work for static nodes; but would not be able to cope with dynamically changing topologies in some wireless environments, such as MANETs.

5.2.2 The Transit Node Problem

Consider the network topology shown in Figure 5.1. A node in Net A is advertising a service that a node in Net B wishes to access; however, the transit node T is not an authorized source for this service. That is, the source address policy advertised for this service does not include T; it does, however, include Nets B and C. What should happen?

One option would be to permit traffic from B to transit T; after all, Net B is an authorized source. This in turn would require that the border node in Net A advertise the service to T, which would presumably pass it on to Net B. This option is insecure: when packets for the service arrive at Net A, it is not possible to tell whether they originated from Net B or were forged by transit node T.

We thus adopt the following policies:

1. Routing advertisements are never propagated to a node not authorized for that service, according to the originator's policy.

2. Packets for a service are only accepted from nodes to which routing advertisements were sent. In a wired net, this is generally easy; in a MANET, it will likely require some form of cryptographic authentication of the neighboring node, since in a wireless environment it is generally very difficult to determine the precise source of a packet.

Note that these principles are not related to route aggregation. A shorter prefix may be transmitted to T if and only if such a prefix already existed and T was authorized for it.

5.2.3 Aggregation Points

Although ROFL as defined will work without further enhancement, installing it as part of a larger network could be seen as unfriendly: blasting that many extra routes into, say, the Internet is considered improper. Indeed, the entire rationale for CIDR [Fuller and Li, 2006] is that distant networks need only see a single short prefix that covers many networks; ROFL should not frustrate that scheme. We propose two basic approaches to resolving this: external and internal aggregation.

External aggregation is the approach taken by today's ISPs to deal with overly-long prefixes announced via BGP. In one variant, known as proxy aggregation, an ISP will generate a single short prefix that covers multiple longer prefixes, and re-announce the shorter one rather than the

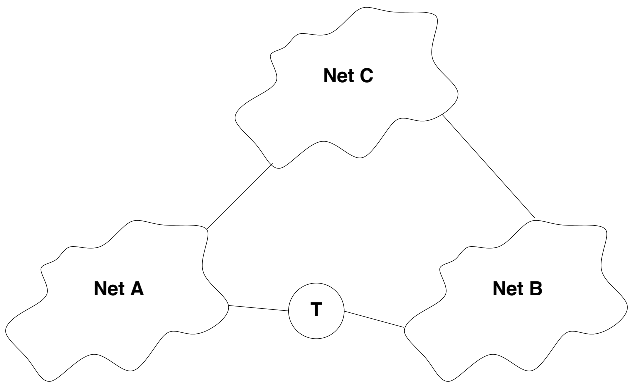


Figure 5.1: Two subnets connected by both a transit node and a transit net.

long ones. In practice, this is very rarely done. The second variant, occasionally known as "satanic philtres" [Rexford *et al.*, 2001], is unilateral: an ISP will drop announcements for too-long prefixes. It assumes that the covered networks will be reachable via some shorter prefix; if they are not, their administrators should arrange for proper CIDR-based addresses.

This latter issue (as well as politeness) impels us to incorporate an aggregation scheme into ROFL. (Other justifications for ROFL aggregation are given in [Zhao *et al.*, 2008a].) We define an *aggregation point* as a node that receives full ROFL announcements but generates fewer re-announcements of shorter prefixes. Typically, this is done at the boundaries of a MANET, though it can be done at other points, both within the MANET and outside it. There are three basic issues: how aggregation points are defined, what prefix should be announced, and what metric should be used.

For the first, we use notation similar to source prefix constraints: we define the ROFL region as a set of prefixes. Any neighbor whose address is not in this set receives only the shorter, covering announcement. Note, though, that the aggregation policy prefix is not the same as the firewall source prefixes; the two are used independently. That is, packets from the wide-area net are dropped if they do not meet any source address constraints for firewall rules; furthermore, firewall rules will frequently exist within the ROFL region.

The aggregation policy is generally a matter of static configuration, and hence is not passed along via a routing protocol. However, since in a MANET any node can be a boundary node, all nodes must have this configuration information. Exactly how this is provided is beyond the scope of this dissertation; we assume that it will be done as part of general node provisioning.

We treat the actual prefix to be re-announced similarly: it is a static policy decision, installed on all nodes.

Metrics are a more interesting problem, since we may wish to preserve some notion of cost even in the wide-area network. Consider: a MANET may consist of several subnets, with multiple attachment points to the wired network. If a given subnet is much closer to one of them, this information should be preserved. It is not obvious how to do this. Even if everyone is using the same values for $\mathcal{C}_i$, $\mathcal{R}_i$, and γ, aggregation points will see many values. Which should be used? The minimum? The maximum? The mean? The median? We suspect that median is correct, but that mean is easier to calculate. This is very similar to routing announcements sent out in Interior Gateway Protocols (IGP) and Exterior Gateway Protocols (EGP) respectively. Routing metrics enclosed in IGP routing advertisements are communicated within an autonomous system (AS); when these announcements reach the boarder of an AS, a different metric will be computed to represent the distance to this AS used by EGP routing. Computation of the new metric may depend on the internal metrics observed at the gateway router. In particular, path selection rules in BGP may consider IGP metrics as the factor. For example, BGP router prefers the path with the lowest IGP metric to the BGP next hop [BGP, 2006].

One more optimization can be introduced. As explained in [Zhao and Bellovin, 2007], local optimizations can be introduced. The policy algebra is used to merge redundant announcements, though in this scheme S must match for two routing advertisements to be merged. As above, treatment of the metric can be a complex process.

5.2.4 Refinement of ROFL Policies

Service-specific route advertisements in ROFL can be translated from network service policies using the centralized policy refinement scheme as the service provider has full control over advertised services and authorized hosts. As discussed in Chapter 4, a network service policy is of the following form:

{Subject} can (or cannot) {access Service provided by} {Target} if {Condition}.

It can be rewritten using our proposed logic-based abstract language as follows:

$$\textit{policy} \equiv \langle Q_S Sub,\ Q_P \langle Q_T Tar,\ Q_V Srv \rangle,\ Cond \rangle\ \pm$$

Then the *policy* is transformed to the following access control tuples by querying the constructed knowledge database using transformation rule 4.9 defined in Chapter 4:

$$\textit{policy}' \equiv \langle Sub',\ Tar',\ Srv',\ Cond' \rangle\ \pm$$

Using the results of policy transformation, low-level implementable rules can be generated from access control tuples through policy composition phase. This step is highly language-specific, and is completely determined by the low-level enforcement mechanism. In the context of ROFL firewalls, service-specific route announcements can be generated using composition rule 5.1:

$$\begin{aligned} &\textit{policy}' \equiv \langle Tar',\ Srv',\ Sub',\ Cond' \rangle \pm \\ &\Downarrow \text{ROFL} \\ &R \equiv \{Tar'.ip : Srv'.port/48,\ Sub'.ip,\ \mathcal{L}(Cond'),\ M\} \end{aligned} \tag{5.1}$$

where $\textit{policy}'$ represents an access control tuple such that a target object Tar' is allowed (+) or prohibited (−) to access service Srv' provided by subject object Sub' under condition $Cond'$.

In rule (5.1), mapping from target or subject to its IP address is straightforward as *ip* is an attribute of Tar' and Sub'. However, it is often the case that a ROFL announcement R propagates to a set of permitted nodes (i.e., the subject). Thus it is more efficient to enclose the whole set of subjects in a single announcement to minimize overhead. Therefore, it is possible to implement it using a Bloom filter [Bloom, 1970] on the set of source addresses or networks of a given prefix length. Bloom filters are a space-efficient data structure that can compress the representation of a set of members in a compact manner, albeit with some chance of false positives. Bloom filters are particularly useful in MANETs, where there is little topological structure and each allowed node may be identified by a flat address.

Mapping from service to port number is obtained by calling $Srv'.port$. Routing metric M is determined by the service provider (i.e., the target). We have $M = \infty$ for negative authorization policy.

Cond' is mapped to labels by calling function $\mathcal{L}(Cond')$:

$$\begin{aligned}\mathcal{L}(Cond') &= \mathcal{L}(\vee_{j=1}^{n} \wedge_{i=1}^{m} (\vee_{k=1}^{p} O_{ij_k}))\\ &= \mathcal{L}(\wedge_{j'=1}^{n'} \vee_{i'=1}^{m'} O_{i'j'})\\ &= l_{\wedge_{j'=1}^{n'}} ({}_{i'=1}^{m'} l_{i'j'})\end{aligned}$$

where it is firstly rewritten into a conjunctive normal form by applying distributive property of logical connectives; then each condition object $O_{i'j'}$ is mapped into a label $l_{i'j'}$. Logical operator $\vee$ is automatically implied between consecutive labels, and operator $\wedge$ is replaced by a special label $l_{\wedge}$.

Now we describe the encoding algorithm from a condition object O to a label l. Each l is a string of 8 consecutive bits, where the first 4 bits denote $O.condType$ and the remaining represent $O.condName$. Together they uniquely identify a condition instance in the knowledge database. Condition type 0000 is reserved, and $l_{\wedge} = 00000000$. Thus our scheme supports 15 types of conditions (although Figure 4.2 shows only two types of conditions in this scenario) and 16 different values for each type. More bits can be added to represent more labels.

Optimization can be made in different ways. The goal is to minimize the length of label list L in a ROFL advertisement, and hence reduce processing time. Wildcard character $*$ represents a bit value of either 0 or 1. Thus 2^n consecutive labels (logical connective $\vee$ implied) different by n bits at fixed positions $b_1, \ldots, b_n$ can be replaced by one label with wildcard $*$ at those positions and the rest remain unchanged. For example, 00010010|00010000 (vertical bar | is for presentation purpose only) can be replaced by $000100*0$.

A more efficient encoding mechanism can also reduce the length of L. We propose an encoding tree (Figure 5.2(a)) to generate labels of the same type, i.e. those have the first 4 bits in common. Root node represents the entire value space for a certain type of label; T_1 denotes 2^{n_1} sub-spaces identified by first n_1 bits; similarly, T_2 further divides each sub-space into 2^{n_2} partitions using the next n_2 bits, and so on until all N bits for label values are used. Thus, only one label is necessary to represent a node or a subtree. Figure 5.2(b) depicts a possible encoding scheme for *Location* labels with $N = 4$. The entire battle field is divided into 4 quads identified by the first 2 bits, and each quad is further split into four sub-quads using the remaining 2 bits. Due to the space limitation, we will not further discuss other optimization schemes for label encoding.

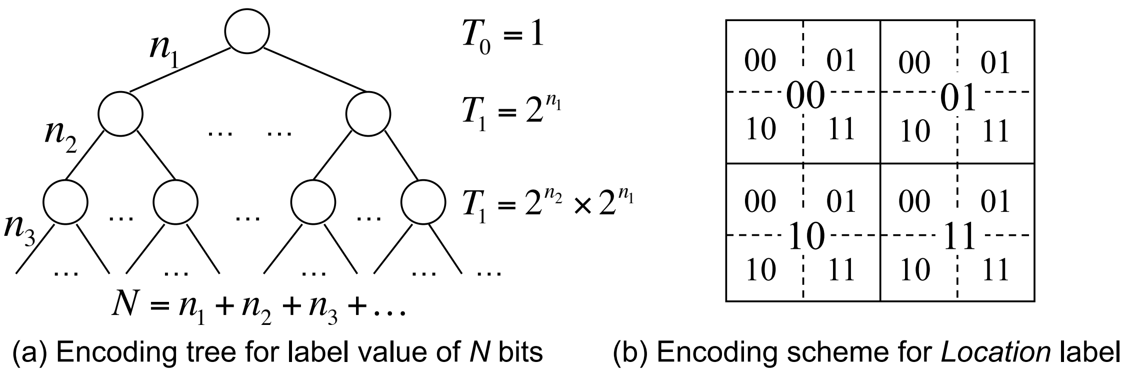

n_1
n_2
n_3
$T_0 = 1$
$T_1 = 2^{n_1}$
$T_1 = 2^{n_2} \times 2^{n_1}$
$N = n_1 + n_2 + n_3 + \dots$
(a) Encoding tree for label value of *N* bits
00 01 00 01
00 01
10 11 10 11
00 01 00 01
10 11
10 11 10 11
(b) Encoding scheme for *Location* label

Figure 5.2: Encoding scheme for label value

As a concrete example, we compose ROFL advertisements from the refinement results obtained previously. The first ROFL announcement says that location server $ls1$ announces location service $loc1$ (high accuracy with mandatory encryption) through secure HTTP service to still camera $sc1$; the second ROFL announcement states that still camera $sc1$ advertises picture service $pic3$ (high resolution with mandatory encryption) to location server $ls3$ through HTTPS from 9am to 5pm (i.e., $time1$).

$$\begin{aligned} & policy' \equiv \langle sc1,\ ls1,\ loc1,\ \emptyset\rangle + \\ & \Downarrow \text{ROFL} \\ & R \equiv \{10.0.0.1 : 443/48,\ 10.0.0.10,\ 00000000,\ M\} \end{aligned} \tag{5.2}$$

$$\begin{aligned} & policy' \equiv \langle ls3,\ sc1,\ pic3,\ time1\rangle - \\ & \Downarrow \text{ROFL} \\ & R \equiv \{10.0.1.1 : 443/48,\ 20.0.0.1,\ 00010001,\ \infty\} \end{aligned} \tag{5.3}$$

5.3 Algorithms

Routers are generally composed of two fundamental mechanisms, the control plane and the data plane. The control plane, sometimes known as route generation, produces a valid path from source to destination by exchanging routing information with other nodes. The data plane, or message forwarding, relays packets from node to node until they reach their final destination, following the selected route. In ROFL, we implement packet filtering by layering it on top of routing. ROFL is

```
RoutePropucast(u)
1  if $u$ provides $s$
2      $R \leftarrow$ genRt($d, s, m, S, M$);
3  else
4      $S \leftarrow$ recvRt($R$);
5      calculateFIB($R$);
6  for each neighbor $h \in N_b$
7      if $S == \phi$ or $h \in S$
8          sendRt($R, h$);
```

```
RoutePropbcast(u)
1  if $u$ provides $s$
2      $R \leftarrow$ genRt($d, s, m, S, M$);
3      bcastRT($R$);
4  else
5      $S \leftarrow$ recvRt($R$);
6      if $S == \phi$ or $u \in S$
7          calculateFIB($R$);
8          bcastRT($R$);
9      else
10         discardRT($R$);
```

Figure 5.3: Algorithms for propagating ROFL service specific advertisements.

agnostic to the specific type of routing protocols used; only slight modifications are needed during the route propagation and the packet forwarding phases.

Because we do not change route calculations or prefix lookups, our new ROFL scheme can cope well with any distance vector or link state algorithms: route announcements in ROFL with source prefix filtering are handled the same way as conventional ones during this phase with minor changes.

5.3.1 Route Propagation

Depending on the dissemination pattern of routing information provided by underlying routing protocol, we propose several variants for route propagation.

Route Propagation via Unicast If point-to-point dissemination of routing announcements is possible, like in a wired network, a service provider has full control over the propagation of service specific routes. In method RouteProp$_{ucast}$ (see Figure 5.3), a service provider (i.e., the route initiator) generates a routing announcement R with appropriate routing information enclosed (Line 1-2), and then propagates R only to neighbors that are allowed to access that service (Line

6-8). Upon receipt of R, a node extracts source prefix constraint information S from R (Line 4), calculates its forwarding information base (FIB) to reflect the update from newly received route (Line 5), and continues forwarding R to its authorized neighbors (Line 6-8).

Route Propagation via Broadcast If routing information is disseminated into a network through broadcast by the underlying routing protocol, such as the flooding phase in many table-driven MANET routing protocols, it is the receiver's responsibility to make a processing and forwarding decision by consulting enclosed source prefix filtering information. In method $\textsc{RouteProp}_{bcast}$ (see Figure 5.3), a service provider u initiates a ROFL announcement R and disseminates it into the network via broadcast (lines 1-3). When a node u receives this announcement, it first retrieves the enclosed SPF constraints S (line 5), and then makes a processing and forwarding decision by consulting S (lines 6-10). If u is an authorized receiver, it shall update its routing table and forwarding information base (FIB) accordingly and then continue broadcasting R (lines 7-8); otherwise, R is silently discarded (line 10).

Route Propagation via Multicast Another alternative for route propagation can be performed via multicast. Suppose there is a separate multicast address for each policy, then ROFL announcements for that policy are sent to that multicast address; only nodes listening to that address will normally receive it. This approach requires a pre-established multicast group for each policy implemented. Moreover, a secure multicast can be performed by assigning group key only to authorized receivers belonging to that group, using available key management mechanism for MANETs.

5.3.2 Packet Forwarding

Once a path is established between a source and destination pair by the underlying routing protocol, every node along the path consults its local routing table T to make forwarding decision. Figure 5.4 describes the packet forwarding algorithm. Upon receiving a data packet K, node u obtains source address p_s, destination address p_d and destination port number s_d from packet header (Line 1). If K arrives at its destination and indeed comes from an authorized source, the packet is passed to upper layer; otherwise K needs to be further routed (Line 2-7). Node u consults its routing table T based on p_d and s_d (Line 9). If a matching route R is found, u extracts the source prefix constraint from R (Line 10). Then K is forwarded towards its destination if it's from an authorized source;

```
    PktForward($u$)
1   $p_s, p_d, s_d \leftarrow$ procPkt($K$);
2   if $u$ is the destination
3     $S \leftarrow$ getSPF($s_d$);
4     if $S == \phi$ or $p_s \in S$
5       sendToUpperLayer($K$);
6     else
7       discardPkt($K$);
8   else
9     if $R \leftarrow$ findRt($T, p_d, s_d$)
10      $S \leftarrow$ extractRt($R$);
11      if $S == \phi$ or $p_s \in S$
12        forwardPkt($K, R$);
13      else
14        discardPkt($K$);
15    else
16      discardPkt($K$);
```

Figure 5.4: Algorithm for packet forwarding.

otherwise, this packet is silently discarded (Line 11-12). Line 3-7 implements a distributed firewall [Bellovin, 1999] at destination host by checking packet source against source prefix constraints for advertised service s_d. This would drop malicious traffic coming from direct neighbors, since there is no intermediate routers in between that can act as firewalls.

5.3.3 Correctness

We claim that our new ROFL scheme with source prefix filtering will not cause any routing mistakes. More precisely, under the assumption that the underlying routing protocols are correct, our algorithm (a) will produce equivalent results for packets not blocked by policy constraints, and (b) will properly drop unwanted packets. We prove the correctness of the following theorems based on the definition of source prefix filtering as well as the routing mechanism provided by the underlying routing protocol. Chapter 6 addresses the correctness and consistency issues from the perspective of policy composition enabled by the algebraic framework discussed in Chapter 3. It also provides an algorithm to verify the correctness of ROFL firewalls by calculating the filtering effect in a given network.

Theorem 5.3.1 Correctness of the Route Propagation Algorithms *The route propagation algorithms ensure that a service specific route advertisement in ROFL only propagates to authorized nodes that are allowed to access the advertised service.*

Proof Suppose a service provider d generates a routing advertisement $R = \{d : s/m, S, M\}$ to announce its service s with source prefix constraint S. Let S be a valid source prefix constraint (i.e. $S \neq \phi$), and p_u represent the address of a node u. We have two different cases when R is propagated into the network before reaching any aggregation point:

1. $p_u \in S$ If the current node is allowed to access service s advertised by d, then u should be able to see the route announcement R originated from d. In the scenario of route propagation via unicast, if u is a direct neighbor of d, it should receive R from d (line 6-8 in method $\textsc{RouteProp}_{ucast}$). If u is multiple hops away from d, it can receive R from its neighbors; otherwise, it implies that none of its neighbors are allowed to access service s. Since transit nodes are not permitted to carry such traffic in our scheme, u cannot establish a valid path to d by any means. In the scenario of route propagation via broadcast, if u is a direct neighbor of d, it should receive R from d through broadcast (line 3 in method $\textsc{RouteProp}_{bcast}$) and process the enclosed information accordingly (line 5-8 in method $\textsc{RouteProp}_{bcast}$). Similarly, if u is multiple hop away from d, it can receive R from an authorized neighbor in absence of the transmit node problem.

2. $p_u \notin S$ If the current node is prohibited from accessing service s provided by d, then u should not see route announcement R originated from d. In the scenario of route propagation via unicast, line 6-8 in method $\textsc{RouteProp}_{ucast}$ guarantees that none of its authorized neighbors will forward the routing advertisement R to u since $p_u \notin S$. In the scenario of route propagation via broadcast, line 9-10 in method $\textsc{RouteProp}_{bcast}$ ensures that a unauthorized node will discard R received from an authorized neighbor via multicast if any.

Therefore, the route propagation algorithms guarantee that only legitimate nodes are allowed to receive and process routing announcement advertised by a service provider. □

Theorem 5.3.2 Correctness of the Packet Forwarding Algorithm *The packet forwarding algorithm ensures that permitted packets will arrive at destination following the established path; whereas non-permitted packets will be dropped early by intermediate routers along the path acting as firewalls.*

Proof The correctness of packet forwarding algorithm relies on the assumption that the underlying routing protocol will generate a valid routing path between each pair of source and destination nodes. Suppose a packet K is originated from a source node with address p_s to access service s advertised by destination node d. Without loss of generality, we assume that the source node is not a direct neighbor of destination d such that every node along the path from source to destination can act as firewalls in ROFL.[2] To demonstrate the correctness of our packet forwarding algorithm, we have the following two cases:

1. $p_s \in S$ Packet K is coming from a legitimate source allowed to access service s. Each router u along the path performs longest-prefix match by consulting its local routing table. Once a matching route R is found, u makes a forwarding decision through a second level of checking against source prefix constraint S embedded in route R. Since $p_s \in S$, u decides to forward this packet according to line 6-10 of the packet forwarding algorithm PktForward. The same process repeats until K reaches its final destination.

2. $p_s \notin S$ Packet K is coming from an illegitimate source prohibited from accessing service s. Each router u along the path performs longest prefix matching by consulting its local routing table. If a matching route R is found, u makes a forwarding decision through a second level of checking against source prefix constraint S embedded in route R. Since $p_s \notin S$, the packet is dropped immediately according to line 11-12 in PktForward, that performs packet filtering; otherwise, K is dropped according to the underlying routing protocol since a matching route is not found (line 15-16).

[2] If source node is a direct neighbor of the destination node, the distributed firewall approach is implemented that allows the destination to make a packet process decision as indicated in line 4-7 in method PktForward. Hence we are not benefiting from the early dropping enabled by ROFL.

Therefore, the packet forwarding algorithm guarantees that permitted packets will arrive at destination following a path constructed by underlying routing protocol; unwanted packets will be dropped by intermediate routers along the path that perform packet filtering functionality based on destination address, port number as well as source address. □

5.4 Performance Evaluation

ROFL will cause growth in local routing tables, and in particular in the FIB. To understand the cost of extending routing protocols to support ROFL firewall mechanism, we analyze the three effects on the local routing table with service specific entries: total storage needed, time to handle routing updates, and time to look up a route. We further evaluate the performance of ROFL in a simulated environment using two ad hoc routing protocols to demonstrate that ROFL is an effective and efficient firewall mechanism for MANETs.

5.4.1 Analytical Results

The total number of routes that must be stored at the aggregation point is the product of the number of hosts within an aggregation region and the average number of services per host. Table 5.1 shows the out-of-the-box configuration for some modern desktop operating systems; in addition, each host will consume one route for client ports, plus an infinity route to block server access. (If the host does not, in fact, run any servers, this latter can be omitted.) We do not consider server machines; in most environments, there are many fewer servers per LAN. It is clear that if very few servers are running, very few service prefixes will be announced for them. We thus assume that on average, each host will announce 13 prefixes. The number of hosts within a region is much more variable, and of course depends on the size of the organization. That said, larger organizations will have larger routers. We can do some rough calculations based on RIR address allocation policies. ARIN requires that organizations use about 80% of their assigned address space before they get more space. They hand out provider-independent address space to organizations that need at least a /20; i.e., to those who have more than about 3300 hosts. If we assume that the number of desktops is a bit larger than the number of employees, we will need to handle about $3300 \times 13 \approx 43,000$ prefixes. This is probably an upper bound; most (though of course not all)

Table 5.1: The number of services running out-of-the-box on default installations of assorted desktop operating systems. The numbers are approximate, since different vendors will ship slightly different configurations. In addition, some ports (i.e., DHCP) are never used off-LAN, and hence would not occupy space in the enterprise routing table. In our tests, about half of the Vista ports were in the Vista client port range ($[2^{14} + 2^{15}, 2^{16} - 1]$); they may thus be covered by our client port scheme.

Operating System	*Services*
Ubuntu 7.10 desktop	3
Windows Vista	13
Mac OS X Leopard	2
NetBSD 4.0	0
FreeBSD 7.0	0
OpenBSD 4.2	3

organizations with more employees than that are split across multiple locations, each of which is a separate aggregation region. 43,000 prefixes is a lot; however, it is less than a fifth of the prefixes in a full Internet feed. Today's enterprise-grade routers are capable of handling this many prefixes. Estimates suggest that only 64M bytes of memory are needed, well below their design limits. It seems, then, that ROFL does not pose an unreasonable memory burden.

Routing table computation time may be more problematic. Let us first approach it analytically. OSPF is based on Dijkstra's Algorithm [Dijkstra, 1959], which is $O(n^2)$ in the number of nodes. However, this bound is for relatively dense graphs. Many enterprise networks and most MANETs are much more sparsely connected; other algorithms can do much better in such cases. In particular, by suitable choice of data structures when implementing Dijkstra's algorithm, the running time can be reduced to $O(e \log_2 n)$ [Johnson, 1977; Aho *et al.*, 1974]. (See Chapter 6 for a detailed algorithm for routing table computation based on Dijkstra's shortest path algorithm for a single source.)

The analytic approach tells us nothing about the constant, so it pays to look at real implementations as well. In [Shaikh and Greenberg, 2001], Shaikh and Greenberg calculated the CPU time needed for OSPF updates on a fully-connected, n-node topology, on a Cisco 7513 router. They do not say which model of Route Switch Processor was used; the time frame suggests that it was likely an RSP1 or RSP2. They found that the CPU time needed was approximately $.00000247n^2$ seconds. For 3,300 nodes, that would come to about 27 seconds, which is a bit on the high side. However, their paper was published in 2001. Furthermore, Cisco has a reputation for using relatively slow CPUs in their routers. Their current top-of-the-line model, the RSP16, has a 400 Mhz CPU; the RSP2 has a 100 Mhz CPU.[3] Accordingly, we suggest that we can scale that figure by a factor of at least 20, which brings the time to about 1.4 seconds, which is clearly acceptable.

There are more important reasons for optimism. Shaikh and Greenberg measured a fully-connected graph. Our topologies are *not* fully connected. There is little published data on enterprise networks; experience suggests, however, that most corporate locations outside of data centers have rather simple and often tree-like topologies. Let us be conservative and assume a fanout of 10 links per node. Using the formula given above, it will take about 2700 operations, compared with the 11,000,000 operations for the fully-connected graph. The CPU time needed should be almost unmeasurably low, even on old, slow CPUs.

There is an implicit assumption in this CPU time analysis that the number of prefixes is irrelevant. If there are no local policies that that cause some services to be routed differently than others, this will in fact be the case; the path between any two nodes will be the same regardless of the port number addressed. If that is not the case, let us approximate differently-routed prefixes as separate nodes. The total cost goes up proportionally to $\log_2$ of the number of nodes; let us continue our assumption that the number of links is 10 per node. We are thus dealing with about 43,000 and 4300 links; the result is only $25\times$ larger than in the simpler case: still very small.

The remaining issue is lookup time. It does not appear to be an issue, either. [Waldvogel, 2000] describes algorithms whose complexity is $O(\log_2 n)$ in prefix length. Doubling the prefix length from a nominal 24 bits on today's networks to 48 bits should cost only one extra memory reference. [Buchsbaum *et al.*, 2003] describes a multilevel lookup algorithm; given the structure of our /48s, this will work extremely well, though their algorithm is not designed for rapid table updates.

From these analyses, we conclude that our scheme is quite feasible, probably on today's plat-

[3] http://www.cisco.com/en/US/products/hw/routers/ps359/products_installation_and_configuration_guide_chapter09186a00801c63a5.html

forms and certainly with minor design changes.

5.4.2 Protocol Design

Analytical results demonstrate the feasibility of adopting ROFL in the current routing system in terms of total storage needed, time to handle routing updates, and time to look up a route. To prove ROFL is an effective and efficient firewall mechanism, we further evaluate the performance of ROFL in a simulated environment. ROFL can be easily adopted by a large number of routing protocols with minor modifications, while being transparent to upper layers. The pattern of disseminating routing information of the underlying routing protocol determines the applicable variant of route propagation algorithm discussed previously. Once the routing table and the forwarding information base (FIB) are successfully updated upon receipt of ROFL announcements, packet forwarding is straight forward following algorithm PktForward discussed previously. Thus, we focus on the propagation of service specific route advertisements in ROFL using MANET routing protocols.

Classification of MANET routing protocols could be done in many ways [Perkins, 2001]. One common taxonomy is to categorize them in terms of routing strategy: reactive routing (on demand), proactive routing (table-driven), and hybrid routing which combines previous two. Another popular classification is based on how routing paths are computing at each node using different routing information. For a comprehensive study, we carefully chose two routing protocols: *Ad hoc On-Demand Distance Vector* (AODV) protocol [Perkins *et al.*, 2003], a reactive routing protocol using distance vector information, and *Optimized Link Stat Routing* (OLSR) protocol [Clausen and Jacquet, 2003], a proactive routing protocol using link state information.

ROFL with AODV AODV is one of the most popular on demand routing protocols for MANETs. It doesn't require routing paths to be established before the presence of active communication. If a route to the destination is not available, a node broadcasts a route request (RREQ) message using an expanding ring search technique. A route reply (RREP) message is then generated by either the destination node or an intermediate node that has a valid route to that destination. Route error (RRER) messages are generated and forwarded to upstream neighbors when a node detects a link break.

To implement ROFL scheme with AODV, we replace conventional routing information embedded in a RREP message with service specific announcement that includes the port number and source prefix constraints. As RREP traverses towards the route requester, ROFL announcement is stored at each intermediate node following the RoutePROP$_{ucast}$ algorithm, because RREP is unicast back to the route initiator along the reverse path that RREQ traversed. We also piggy-back client route information in RREQ messages initiated by the route requestor at the beginning. Therefore, our implementation doesn't require extra control messages to be transmitted comparing to its non-ROFL counterpart (i.e., pure AODV).

ROFL with OLSR OLSR is a commonly deployed table-driven routing protocol for MANETs, that makes use of link state information to compute shortest paths to all destinations in a network. To reduce flooding overhead, only nodes, selected as Multipoint Relays (MPR), are responsible for forwarding control traffic disseminated into the entire network. Hello (HELLO) message contains the link information with the 2-hop neighborhood of a node, and Topology Control (TC) message declares link availability between a node and its MPR selectors. Without loss of generality, we assume that each node has only one interface participating in OLSR routing. Thus multiple Interface Declaration (MID) messages are irrelevant to route discovery. With ROFL, service specific routes with source prefix constraints are embedded in each HELLO message and TC message. Since those messages are broadcast, route propagation algorithm RoutePROP$_{bcast}$ is implemented for OLSR.

Therefore, we implement the ROFL scheme with AODV and OLSR respectively. Most of the implementation effort is dealing with routing table maintenance, and its cost has been analyzed quantitatively.[4] Although the implementation is protocol-specific, ROFL announcements are treated just like any conventional routing advertisements in those protocols.

5.4.3 Simulation Setup

We conduct experiments in a simulated environment to analyze data traffic and control traffic incurred in ROFL with two underlying routing protocols respectively. We implemented ROFL in GloMoSim [glo, 2000], where 100 nodes, with transmission range of 250 m, are randomly deployed

[4]Simulation studies cannot capture the performance overhead in terms of routing table maintenance, which requires a kernel implementation on real systems. This is beyond the scope of this dissertation, and can be left for future work.

in a simulation area of 1500×1500 m^2. Our goal is to demonstrate the high performance of ROFL as an effective and efficient firewall mechanism especially for MANETs. Without presence of unwanted traffic, no firewall mechanism is able to prove its effectiveness. Hence, in our experiments, one key parameter is the amount of malicious traffic injected into the network. We generate 30 constant bit rate (CBR) flows, each consisting of data packets of 512 *Bytes* transmitted between a pair of randomly chosen source and destination nodes at a rate of 4 $pkts/sec$. Thus we observe both intra-group and inter-group communications with respect to the Random Point Group Mobility (RPGM) model. We vary the amount of malicious traffic from 0% to 100% with an increase of 10% each time. Situations with more than 70% malicious traffic, although unlikely to occur in reality, are used to verify the correctness of ROFL. Simulation parameters are specified in Table 5.2.

Simulation Parameters	**Values**
Topology Size	$1500 \times 1500\ m^2$
Node Number	100
Simulation Time	900 s
Transmission Range	250 m
Traffic Type	Constant Bit Rate (CBR)
Packet size	512 *Bytes*
Packet rate	4 $pkts/sec$
Traffic Volume	30 $flows$
Malicious Traffic	$[0, 10, \ldots, 100]$ %
Mobility Model	Reference Point Group Mobility (RPGM)
Max Speed	$[0, 5, 10, 15, 20]\ m/s$
Pause Period	30 s
GloMoSim Version	GloMoSim-2.03
Mobility Generator	BonnMotion-1.4

Table 5.2: Simulation setup.

Individual random mobility patterns, like Random Waypoint (RWP) or Random Walk (RW), are found to have flaws or limitations [Camp *et al.*, 2002]. In a MANET scenario, such as military deployment or disaster recovery, grouped motion behavior is very often. Therefore we import

Reference Point Group Mobility (RPGM) model generated using BonnMotion [bon, 2011]. In RPGM [Hong *et al.*, 1999], nodes are partitioned into groups based on their logical relationship. The movement of an individual node is captured by its group motion plus a random motion from its reference point. To evaluate ROFL's performance as a purely distributed firewall mechanism, we set the averaged number of nodes per group to 10, and vary the maximum speed of node movement from $[0, 5, 10, 15, 20]$ m/s. Notice that 20 m/s corresponds to 72 km/hr, which is a reasonable maximum speed of a vehicle in MANETs, though perhaps too fast for military models.

The performance of ROFL is evaluated against distributed firewalls [Bellovin, 1999], where firewall policies are centrally computed and pushed to individual end hosts for enforcement. Thus filtering of malicious traffic is at destinations only. We are particularly interested in the amount of data traffic and control traffic incurred both with and without ROFL; other metrics, such as throughputs, end-to-end delays, etc, are not our focus, as they are generally considered as performance metrics of underlying routing protocols. Each experiment runs for 900 *sec* with 30 *sec* pause period. Results collected from each experiment are averaged from 20 runs.

5.4.4 Performance of Routing Protocols with RPGM

Prior to conducting experiments with ROFL scheme, we evaluated the performance of AODV and OLSR across the set of RPGM models with various degrees of mobility as shown in Figure 5.5. Clearly, both AODV and OLSR didn't achieve their best performance under RPGM. Previous studies on group mobility models [Bai *et al.*, 2003; Pei *et al.*, 1999] demonstrated that compared with flat routing schemes, such as AODV or DSDV, hierarchical or cluster-based routing protocols exhibits much better scalability. Therefore it is not surprising to observe more fluctuation of these performance curves when parameters, like group size or movement pattern, are further tuned for this purpose. ROFL implements packet filtering by taking advantage of underlying routing mechanisms. Thus its performance cannot be completely isolated from the behavior of underlying routing protocols that are used. Results in Figure 5.5 serves as a baseline evaluation for subsequent experiments.

Scenario 1. ROFL with AODV Figure 5.6 depicts the reduction of data packets observed at destinations using ROFL, given that the percentage of malicious traffic present in the network

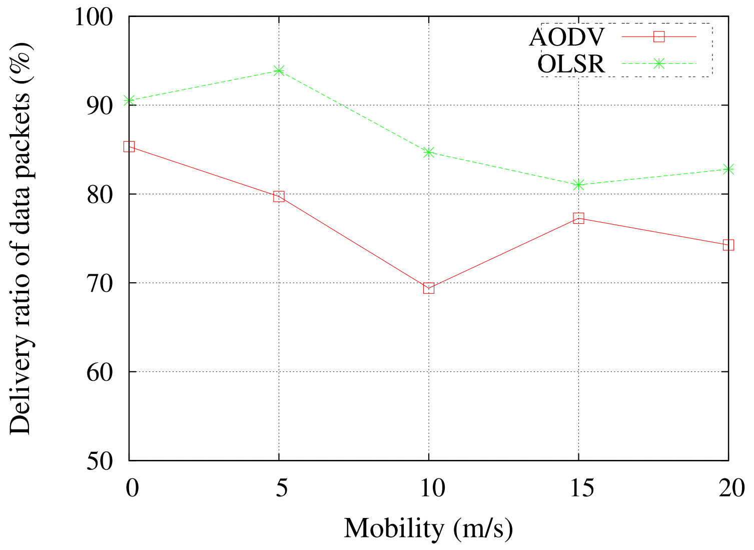


Figure 5.5: Performance of pure-AODV and pure-OLSR under RPGM mobility model (no malicious traffic).

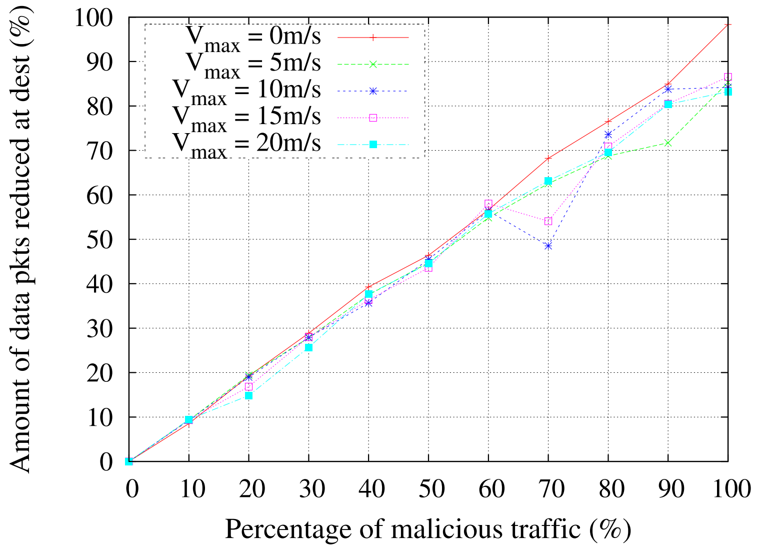


Figure 5.6: Reduction of data packets at destinations using ROFL with AODV.

varies from 0% to 100%. The line with $V_{max} = 0\ m/s$ represents a static ad hoc network; whereas $V_{max} = 20\ m/s$ indicates a highly dynamic MANET. (Data packet loss occurs due to several reasons, such as node mobility, collision at link layer, etc.) Ideally, data packet reduction increases by 10% each time with the growth of malicious traffic. Obviously, with distributed firewalls, malicious

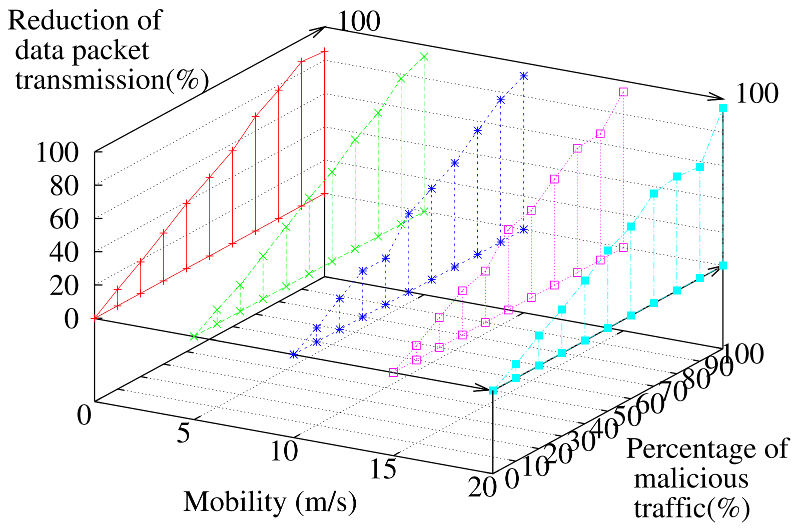


Figure 5.7: Reduction of data packet transmission using ROFL with AODV.

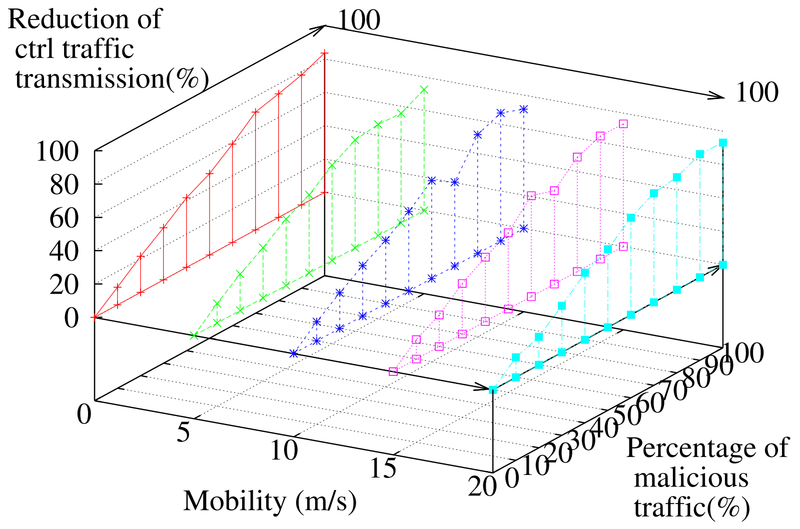


Figure 5.8: Reduction of control packet transmission using ROFL with AODV.

packets are not dropped until reaching their final destinations; hence data packets observed at destinations are not affected by the amount of malicious traffic present.

In ROFL, unwanted packets are dropped further away from destinations depending on how far ROFL announcements can propagate in the network. We observe a significant reduction of data packets at destinations in Figure 5.6, that clearly increases with the growth of malicious traffic. When nodes start moving faster, we notice slight fluctuations caused by rapidly changing topologies. We believe these fluctuations are closely related to the performance of underlying routing protocols rather than ROFL itself (see the baseline scenario in Figure 5.5). With a maximum speed of 20 m/s, we still observe significant packet reduction following the same trend.

As energy conservation is crucial in MANETs, it is beneficial and desirable to save battery power by avoiding unnecessary transmissions. Figure 5.7 and 5.8 illustrate the reduction of packet transmission for data traffic and control traffic respectively. Data packets transmitted in the network counts not only packets generated at their origins but relaying effort made by intermediate nodes. By dropping unwanted data packets as early as possible, we experience significant reduction of data packets with the growth of malicious traffic. In addition, ROFL triggers reduction of control traffic, as route requests (RREQs) from malicious (i.e., unauthorized) nodes are dropped silently by neighbors which have seen the corresponding ROFL announcement before. However, reduction of control traffic flattens when there are significant amount of malicious traffic but very little legitimate flows in the network (at $y = 80\%,\ 90\%,\ 100\%$ in Figure 5.8). Recall that since AODV implements on-demand routing strategy, a service-specific route announcement gets disseminated into the network when there exists a permissive routing request (RREQ). With very little legitimate data traffic, hence very few permissive route requests, we cannot benefit further from ROFL. Moreover, increases of nodes mobility only introduces slight degradation of ROFL's performance.

Scenario 2. ROFL with OLSR Figure 5.9 shows the reduction of data packets received at destinations using ROFL with OLSR as the underlying routing protocol, with malicious traffic varying from 0% to 100%. Obviously, ROFL achieves an incremental reduction of received data packets at destinations along with the growth of malicious traffic, even under a highly dynamic network topology. With OLSR, service-specific route announcements originated from service providers are disseminated into the entire network via the flooding of *Topology Control (TC)* messages. Theoretically, with ROFL, each participant in the network has a complete knowledge of services, that they are allowed to access. Thus malicious packets are expected to be dropped very close to their origins. Figure 5.10 shows the reduction of data packet transmissions. Unlike with AODV, ROFL doesn't affect control traffic transmission in OLSR; because OLSR implements table-driven routing approach with routing information periodically flooded to the entire network.

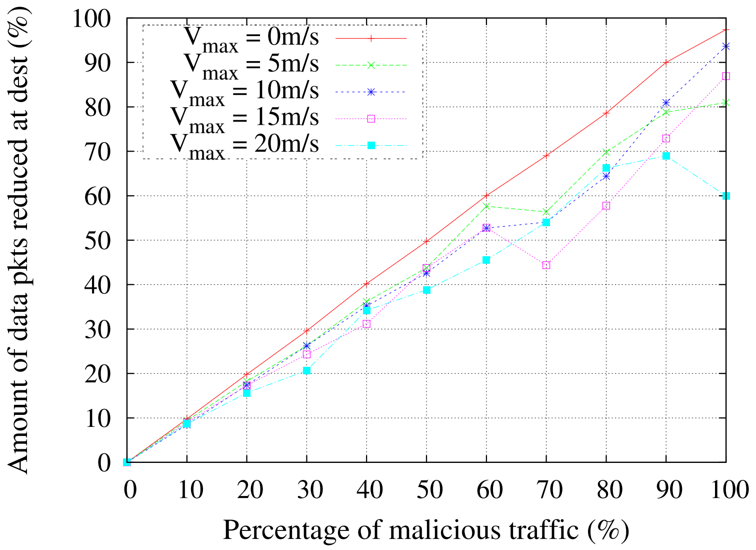


Figure 5.9: Reduction of data packets at destinations using ROFL with OLSR.

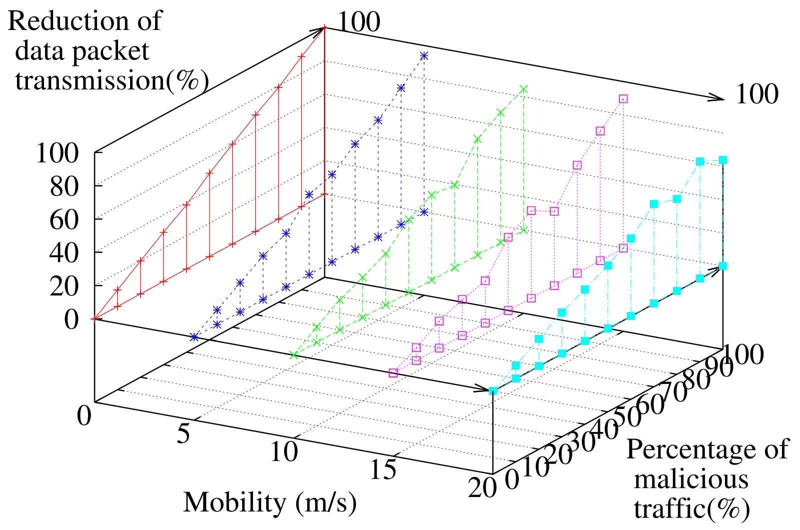


Figure 5.10: Reduction of data packet transmission using ROFL with OLSR.

Summary Experiments with two different MANET routing protocols demonstrate that ROFL outperforms distributed firewalls by effectively reducing the amount of data transmission with various amount of malicious traffic. Implementation overhead of ROFL includes both the extra cost of maintaining routing table with service specific entries and the cost of transmitting control traffic in the network. The former has been discussed analytically. With the growth of nodes mobility, we still observe the same trend. As a recap, we summarize the performance of ROFL with small amount of malicious traffic (as that is often the case in reality) under various degree of mobility in Tables 5.3, 5.4, and 5.5.

	Node Mobility				
Malicious Traffic	0 m/s	5 m/s	10 m/s	15 m/s	20 m/s
10%	9.8%	8.5%	8.4%	8.7%	8.4%
20%	18.9%	15.9%	18.6%	17.9%	17.8%

Table 5.3: Reduction of data packet transmission using ROFL with AODV, given small amount of malicious traffic.

	Node Mobility				
Malicious Traffic	0 m/s	5 m/s	10 m/s	15 m/s	20 m/s
10%	10.7%	11.6%	11.5%	11.7%	11.8%
20%	21.6%	21.8%	21.4%	19.0%	16.7%

Table 5.4: Reduction of control packet transmission using ROFL with AODV, given small amount of malicious traffic.

	Node Mobility				
Malicious Traffic	0 m/s	5 m/s	10 m/s	15 m/s	20 m/s
10%	9.9%	9.4%	8.7%	7.7%	8.0%
20%	19.8%	18.0%	17.5%	16.2%	16.4%

Table 5.5: Reduction of data packet transmission using ROFL with OLSR, given small amount of malicious traffic.

Clearly, ROFL achieves high performance in terms of the reduction of data traffic and control traffic transmission regardless of the network mobility. For instance, for AODV, with nodes moving at 10 m/s and a 10% bad traffic rate, ROFL reduced the number of data packet and control packet transmissions by 8.4% and 11.5% respectively; at 20% bad traffic, the reduction were 18.6% and

21.4%. For OLSR, with nodes moving at the same speed, ROFL reduced the amount of data packet transmission by 8.7% (at 10% bad traffic) and 17.5% (at 20% bad traffic) respectively.

Apart from above results, we also observe some interesting phenomena. In the first scenario with AODV, malicious nodes are induced to generate more routing requests (RREQs), as previous ones were silently dropped by neighbors who have seen the relevant ROFL announcement with embedded source prefix filtering information before. Depending on how aggressive the attacker is, it may generate much more control traffic without being able to establish a valid route. Thus our scheme indeed helps to drain their battery power faster, which can disable the compromised nodes. Possibly, such an abnormal rate of route advertisements can be useful input to an intrusion detection system, though we have not investigated this.

One further aspect is worth mentioning. With group mobility models, if the groups move too far apart ROFL is more likely to result in network partitioning. We do not regard this as a flaw in ROFL; rather, it is a tradeoff between one form of availability (a connected network) and both a second form of availability (battery power) and confidentiality: traffic is not sent through nodes not trusted to handle it. A likely solution would involve some number of trusted, higher-powered transmitters coupled with a hierarchical routing scheme. We have not yet evaluated this alternative. Therefore, we conclude that ROFL is an effective and efficient firewall mechanism for ad hoc networks that are highly dynamic.

5.4.5 The Impact of Network Density

In this subsection, we will study the impact of network density on the performance of ROFL scheme in MANETs using simulations. We chose the Network Simulator NS-2 [ns2, 2012] as a reference implementation for AODV protocol as it remains one of the most popular simulation tools for studying ad-hoc routing protocols. We very the number of nodes N deployed in an area of 1500×1500 m^2 and let N to be $25, 50, 100$ and 150 respectively. The transmission range for each mobile node is set to 250 m by default. CBR traffic generation tool *cbrgen* provided by NS-2 is used to generate 10 constant bit rate connections where each traffic source sends 4 packets of size 512 bytes per second. We use the CMU's node-movement generator *setdest* to create random waypoint node movements for mobile nodes. Each simulations runs for 900 seconds with a pause period set to 30 seconds and maximum speed equal to $\{0, 5, 10\}$ m/s respectively. We set the amount of malicious traffic injected into the network to be 0%, 10%, and 20% respecitvely. In this set of experiments, we are not interested in large amount of malicious traffic and very fast node movement speed as the correctness of ROFL has already been verified in previous studies by varying those two parameters.

Delivery Ratio and Network Density Network partitions occur in sparse networks (e.g., $N = 25, 50$) when nodes are out of each other's transmission range. Therefore, packet delivery ratio heavily depends on the network topology and node movement patterns. Given the fact that the performance of ROFL cannot be completed isolated from the underlying routing protocols, we first study data packet delivery ratio of pure AODV for comparison purposes. Figure 5.11 illustrates the delivery ratio of AODV under different network densities with varying degree of mobility. Each number is obtained by averaging the results from 20 different network topologies and movement patterns.

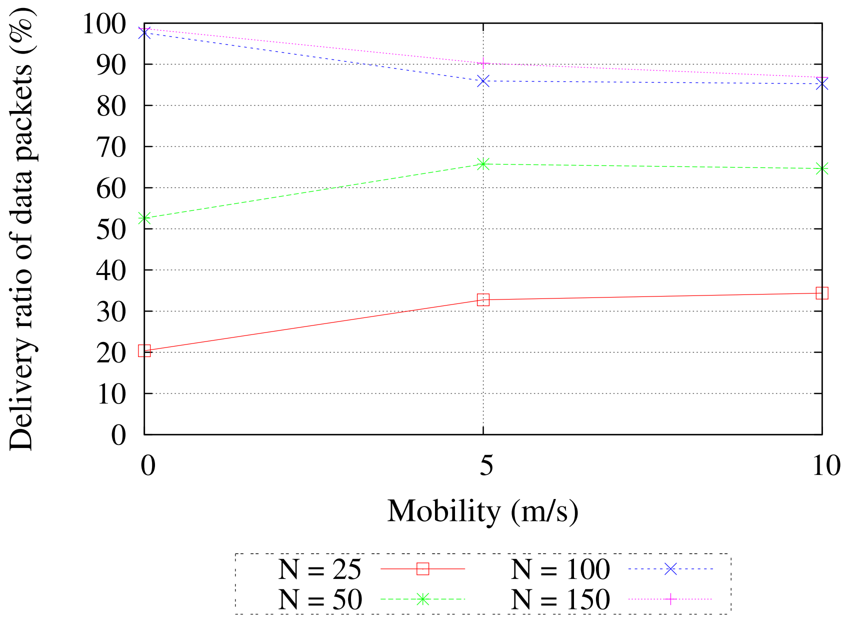


Figure 5.11: Performance of pure-AODV under different network densities.

Figure 5.11 shows that network delivery ratio grows with the increase of network density. AODV protocol experiences the lowest delivery ratio when there are only 25 nodes deployed in an area of $1500 \times 1500m^2$. Generally speaking, the increase of node mobility will decrease the network delivery ratio when the number of deployed nodes is relatively large (e.g., $N = 100,\ 150$). On the other hand, node mobility can have positive impact on delivery ratio in sparse networks (e.g., $N = 25,\ 50$). As nodes start moving around, a valid path could be established between a pair of previously unreachable nodes.

ROFL and Network Density As we have shown previously, ROFL can filter malicious traffic before they reach their destinations, in comparison to distributed firewalls where everything is filtered at the end hosts. In the ideal situation, ROFL can filter all the malicious traffic presented in the network. However, when network partition occurs, both legitimate and malicious traffic gets dropped because of unreachable destinations despite of the packet filtering functionality provided by ROFL. The percentage of malicious traffic that reaches their destination in pure AODV implementation varies given different network topology and movement patterns. For example, our experiments with 100 different network topologies and movement patterns show that a static network of 25 nodes without the presence of malicious traffic has an average delivery ratio of 20.37% with standard deviation equal to 17.55. Given the same set of networks but with 10% (or 20%) malicious traffic, the delivery ratio dropped to 18.87% (or 17.18%) with standard deviation equal to 17.43 (or 16.17). On average, ROFL drops 7.36% (or 15.64%) of malicious traffic among those packets deliverable to destinations with 10% (or 20%) of malicious traffic, although the exact filtering rate differs largely given different network settings.

We observe similar filtering patterns by varying three key parameters: the number of deployed mobile nodes, the degree of node mobility and the amount of malicious traffic injected into the network. The results are illustrated in Figure 5.12, and each number is averaged over 20 runs with different topologies and movement patterns. Figure 5.12 demonstrates that ROFL can effectively drop malicious traffic regardless of the changes in these three parameters. ROFL performs better when the network delivery ratio is relatively high. The intuition behind is that if a malicious packet gets dropped because of network partition, it cannot be filtered by ROFL as ROFL implements filtering as a form of routing.

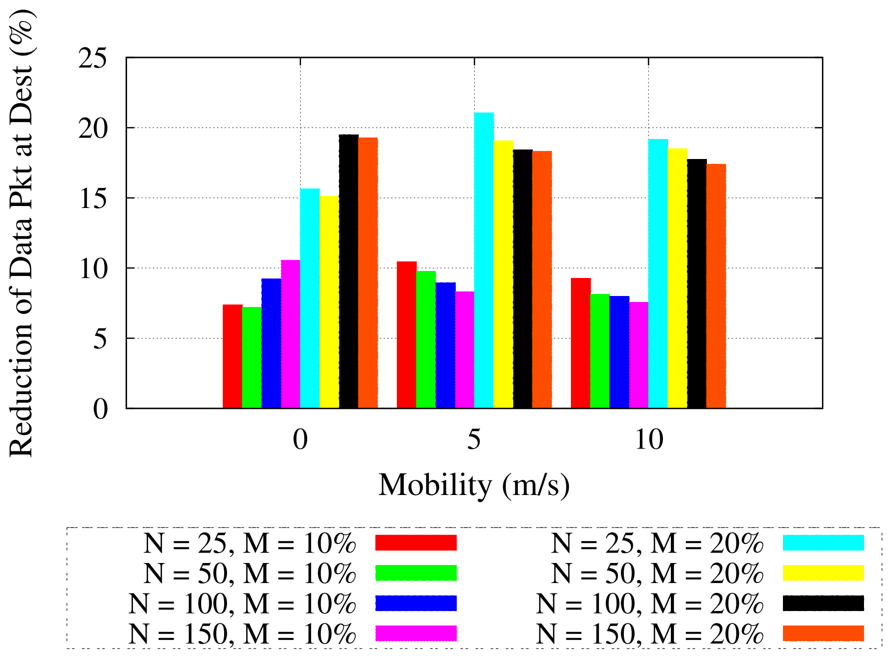


Figure 5.12: Performance of AODV with ROFL under different network densities.

Despite of network density, the presence of transit node problem may also affect network connectivity as discussed in Section 5.2.2. For the purpose of studying the impact of network density on ROFL firewall, we decide to isolate the transit node problem by minimizing the number of unauthorized nodes in our experiments. However, this problem cannot be completely eliminated with the presence of malicious traffic. This is a policy decision that it is better that some packets not be delivered than that they be exposed to compromised nodes. That isn't always the right policy, but sometimes it is. On the other hand, since AODV implements single path routing, with ROFL firewall, a valid path between an authorized source to destination may not be selected if route request message (RREQ) along another path with transit node reaches destination earlier. Because ROFL decreases the number of paths available for forwarding, it effectively decreases the node density or transmission range. Its effects on the likelihood of partition are thus similar. This problem can be solved using the multi-path version of AODV, i.e., ad hoc on-demand multipath distance vector routing (AOMDV). We leave the quantitative study on transit node problem as future work.

5.5 Discussion

In this Chapter we propose a new paradigm of combining routing with packet filtering – the dual effects of the same fundamental question, that is how packets should routed (or dropped if required) to their destinations. ROFL operates at that layer. Today's solutions require either a range of open-to-all ports or a firewall that snoops on control channel traffic to decide what ports should currently be permitted. Both are unsatisfactory; the former can lead to insecurity and the latter stifles innovation. With ROFL, a host knows which ports are needed for which sources, and can emit the proper route advertisements. Nor are routers turned into firewalls, except in effect; they simply listen to routing advertisements and forward packets as usual, albeit with longer addresses. Moreover, ROFL implements a form of policy routing by performing route selection based on source address prefix, in addition to destination address and port number. No extra administration or state is needed, save what is required to hold the larger routing tables; calculations showing that the increase in table size and the cost of routing table computation are acceptable. We also show that the increase in traffic for routing messages is more than outweighed by the savings by early drops of unwanted traffic.

The ROFL firewall mechanism is a concrete application of policy delegation enabled by the algebraic framework as discussed in Chapter 3. Each service provider announces its own policies in terms of routing advertisements and distributes these policies into the communication network using the underlying routing system. Therefore every authorized node along the path is selected as a delegate to enforce the enclosed policy. The algebraic operations help to understand the effect of the composition of packet filter rules at each router, in order to prove the correctness and consistency of policy delegation despite of topology changes (see Chapter 6 for details).

In the proposed framework, cost and risk functions enable the computation of overall system performance associated with policy enforcement. To balance the trade-off between cost and risk, a global optimization algorithm is required to minimize overall cost while maintaining the risk level below certain threshold. However, such a centralized algorithm could be expensive and not scalable when the size of the network grows. In ROFL, the cost of policy distributed is neglected since the dissemination of ROFL rules is embedded in the underlying routing protocol. The cost of routing table maintenance is reasonable as discussed in the analytical results. Empirical study on risk metric in a simulated environment, as well as a kernel implementation of the ROFL scheme to analyze the overhead of routing table maintenance, are left for future work.

Chapter 6

A Unification of Routing-based Distributed Policy Enforcement

Policy enforcement in distributed environments poses a number of challenges; among these are policy distribution, policy consistency, and policy correctness. Policy distribution concerns with disseminating policies to individual enforcement points; whereas policy consistency and correctness ensure that the enforced rules are correct with respect to the security requirements and consistent despite of changes or updates. The ROFL scheme (ROuting as the Firewall Layer) described in Chapter 5 addresses the first concern, by having each node advertise its own policies through the routing system; however, this increases the other two problems: how do we know that policy will be correct and consistent across a changing topology?

ROFL works by carrying security policy statements in ordinary routing protocols. Specifically, port numbers are considered as part of the node's address. Ordinary routing advertisements thus specify what services can be reached on each node. When a node receives an advertisement, it applies its own policies. Understanding the effect of the composition of the two policies is addressed by the policy algebra introduced in Chapter 3. Finally, since the ROFL scheme depends on the routing system, we must ensure that we have not damaged it. We use the routing algebra [Sobrinho, 2003] to show that ROFL networks behave correctly. Hence our goals in this Chapter are to prove that the following three properties hold:

1. *Consistency*: distributed policy enforcement in ROFL correctly implements specified firewall

policies and achieves filtering effects in a consistent manner;

2. *Convergence*: a routing protocol adopting ROFL policies should eventually determine stable forwarding tables that implement loop-free paths between every pair of endpoints;

3. *Optimality*: the resulting path between every pair of endpoints is also an optimal or local-optimal path.

6.1 Routing Algebra

An essential requirement for any routing protocol to function properly is the protocol *convergence*, which states that once all changes have ceased and no more signaling messages are to be found, a routing protocol should eventually determine stable and loop-free paths between every source and destination pair. Another generic requirement for any performance-based routing protocol is property of the paths that a routing protocol converges to, i.e., the path *optimality* requirement. A study on network routing algebra in [Sobrinho, 2003] concludes that *monotonicity* and *isotonicity* are the key properties to guarantee these requirements. Monotonicity means that the weight of a path cannot decrease when extended; isotonicity means that the weight relationship between two paths with the same origin is preserved when both are extended by a common link. The following two definitions correlate those concepts:

Definition 6.1.1 *If the algebra is strict monotone, then the routing protocol always converges.*

Definition 6.1.2 *If the algebra is isotone as well as monotone, then there is an optimal path from node u to node d such that all of its subpaths with destination at d are optimal paths on their own; otherwise, if the algebra is monotone but not isotone, the paths that the routing protocol converges to are local-optimal paths but not necessarily optimal paths.*

A Routing Algebra Meta-Language (RAML) was proposed in [Griffin and Sobrinho, 2005] based on [Sobrinho, 2003], which provides basic building blocks to construct a large family of routing algebras for compound routing protocols. The benefits of metarouting allow a clear separation between *mechanism* and *policy.* More specifically, a basic routing algebra is a tuple:

$$A = \langle \Sigma,\ \preceq,\ L,\ \oplus,\ O \rangle$$

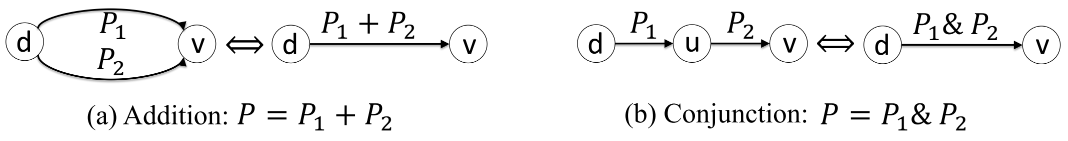


Figure 6.1: A network interpretation for addition (+) and conjunction (&).

where Σ is a set of signatures for describing paths, $\preceq$ is a preference defined over signatures, L is a set of labels for describing links, $\oplus$ is a label application function that maps $L \times \Sigma$ to Σ, and O is an origination set describing signatures associated with originated routes. A complex algebra can be modeled using the basic ones and a set of composition techniques as discussed in [Griffin and Sobrinho, 2005]. lexical product $A \otimes B$, scoped product $A \odot B$, disjunction $A \triangleleft B$ and programmatic labels $\textsc{prog}(A)$.

6.2 Unifying ROFL with Policy Algebra

The policy algebra framework shows how different policies can be composed and rearranged at different nodes in distributed policy enforcement scenarios. As a concrete instantiation, we unify the ROFL scheme with policy algebra in the context of ROFL policies. It provides a basis for constructing routing algebra components and calculating distributed filtering effect in a given network.

In the context of packet filtering implemented by ROFL, addition (+) can be interpreted pictorially as merging firewall policies over parallel paths; conjunction (&) can be viewed as merging firewall policies over serial paths (see Figure 6.1 for details); subtraction ($-$) removes the filtering effect specified in P_2 from P_1; negation ($\neg$) is simply the reverse of filtering action; and finally projection (π) can be used to extract a partial policy $\pi^u P_1$ applied only to traffic initiated from node u. Given a ROFL policy $R = \{d : s/p,\ S,\ M\}$ stating that destination host d announces service s to a set of authorized host S with routing metric M, we then have $\pi^u R = \{d : s/p,\ S \cap \{u\},\ M\}$. If $S \cap \{u\} = \phi$, then $\pi^u R = \phi$, which implies that node u is not authorized to access the advertised service. We introduce another unary operator called Propagation (θ), denoted as $\theta^u P_1$, to describe the propagation of a policy P_1 to an adjacent node u. To ease our discussion, let R^S represent a ROFL advertisement R with source prefix constraints S. Table 6.1 summarizes the precedence,

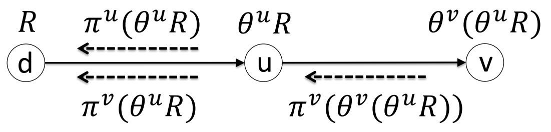


Figure 6.2: Propagation of ROFL policy R along a simple path of three nodes.

associativity and semantic definition of the six algebraic operators.

Figure 6.2 illustrates the propagation of ROFL policy using Projection (π) and Propagation (θ) along a simple path of two links ($d,\ u$) and ($u,\ v$). Solid arrows represent the direction of routing information flow, and dashed arrows show the direction of the actual traffic flow. Route propagation allows node u see policy $\theta^u R$ and node v see policy $\theta^v(\theta^u R)$ in consequence, such that $\theta^u R = R$ iff $u \in S$, and $\theta^v(\theta^u R) = R$ iff $u, v \in S$. Traffic originated from u will be filtered by policy $\pi^u(\theta^u R)$[1], and traffic from v will be filtered by $\pi^v(\theta^v(\theta^u R))$ at v and $\pi^v(\theta^u R)$ at u. Hence, the overall filtering effect of policy R along this path can be computed as $R' = \pi^u(\theta^u R) + \pi^v(\theta^u R) \& \pi^v(\theta^v(\theta^u R)) = \pi^u(\theta^u R) + \pi^v(\theta^u R \& \theta^v(\theta^u R)) = \pi^u(\theta^u R) + \pi^v(\theta^v(\theta^u R)$. This demonstrates a specific property of ROFL policy due to its propagation methodology. In general, suppose node n_0 initiates a policy R propagated along a path of nodes $n_1, n_2, \dots, n_k$, such that node n_i receives policy $R_i = \theta^{n_i}(\theta^{n_{i-1}}(\dots(\theta^{n_1} R)))$ and $R_1 \supseteq R_2 \supseteq \dots \supseteq R_k$. Since the result of conjunction is determined by the most restrictive policy, the filtering effect for traffic originated from node n_i is determined by policy R_i applied at n_i. That is whether traffic initiated from n_i for an advertised service can successfully reach destination is solely determined by the ROFL policy R_i seen by n_i. This observation significantly simplifies the computation of distributed filtering effect.

6.3 Routing Properties

Since ROFL implements packet filtering and policy routing by utilizing the functionalities provided by underlying routing protocols, our primary goal is to study the two essential properties required by the routing algebra theory, namely *monotonicity* and *isotonicity*, in order to draw conclusions on *protocol convergence* and *path optimality* with ROFL.

[1] Filtering at origin implies that only traffic with a destination prefix that can be found in the local routing table will be routed correctly through an outgoing interface.

Prec.	Op.	Description	Associativity	Definition
1	$\neg$	Negation	Right-to-left	$\neg R = \begin{cases} \{d : s/p,\ S,\ \infty\} & \text{if } M \neq \infty \\ \{d : s/p,\ S,\ M' \neq \infty\} & \text{otherwise} \end{cases}$
	π	Projection		$\pi^u R = \begin{cases} \phi & \text{if } S \cap \{u\} = \phi \\ R^{\{u\}}(or\ R^u) & \text{otherwise} \end{cases}$
	θ	Propagation		$\theta^u R = \theta^u \{d : s/p,\ S,\ M\} = \begin{cases} R & \text{if } u \in R \\ \phi & \text{otherwise} \end{cases}$
2	&	Conjunction	Left-to-right	$R^{S_1} \& R^{S_2} = R^{S_1 \cap S_2}$
3	$+$	Addition	Left-to-right	$R^{S_1} + R^{S_2} = R^{S_1 \cup S_2}$
	$-$	Subtraction		$R^{S_1} - R^{S_2} = R^{S_1 \setminus S_2}$

Table 6.1: Precedence, associativity and definition of six algebraic operators.

From the destination-based routing perspective, ROFL extends a destination prefix by including port number to represent a service. Without source prefix constraints S triggering variations of route propagation mechanisms, any routing protocol implementing ROFL is obviously consistent, with the only difference that routers are now simply dealing with longer addresses. However, with policy region and source prefix filtering enabled in ROFL, study on those routing properties remains an open question. Although source address policy routing is an old concept, none of existing works have analyzed that using the routing algebra approach. To the best of our knowledge, we are the first to study routing properties associate with source address routing under a theoretical model.

6.3.1 Definitions and Formalism

Studies on ROFL cannot be isolated from the underlying routing protocol. Without loss of generality, we choose a conventional shortest path routing protocol to implement ROFL scheme, and establish the relationship between them using the metarouting theory. Hence the routing protocol implements both a firewall mechanism, modeled using a packet filter algebra A, and a shortest-path algorithm, modeled using a shortest-path algebra B. The compound routing protocol is then a lexical product of the two, such that:

$$A = \langle \Sigma_A,\ \preceq_A,\ L_A,\ \oplus_A,\ O_A \rangle,\ B = \langle \Sigma_B,\ \preceq_B,\ L_B,\ \oplus_B,\ O_B \rangle,$$

$$A \otimes B = \langle \Sigma,\ \preceq,\ L,\ \oplus,\ O \rangle.$$

Packet Filter Algebra A Each signature $\sigma_A \in \Sigma_A$ is either a tuple $(d : s/p,\ S)$, representing a permitted path for nodes in S to access service s advertised by host d, or a special signature $\phi \in \Sigma_A$, denoting a prohibited path. For any $\sigma_A \in \Sigma_A - \{\phi\}$, we say that σ_A a *usable* signature. $\preceq_A$ is a preference relation over signatures, and intuitively $d : s/p \prec_A \phi$ as the prohibited path is always less preferred. When calculating routing table entries, it is unable to predicate the origin of future incoming packets. Hence, the first component $d : s/p$ dominates the comparison of two signatures $\sigma_1 = (d_1 : s_1/p_1,\ S_1)$ and $\sigma_2 = (d_2 : s_2/p_2,\ S_2)$. The fact that node u already sees the advertised announcements immediately implies $u \in S_1$ and $u \in S_2$. Let $q(d : s,\ p)$ $(0 \leq p \leq 48)$ be a predicate representing the first p bits of the prefix, and $k = min(p_1,\ p_2)$. We define the following rules:

1. $\sigma_1 \sim_A \sigma_2 \Leftrightarrow q(d_1 : s_1,\ p_1) = q(d_2 : s_2,\ p_2)$ and $p_1 = p_2$;
2. $\sigma_1 \preceq_A \sigma_2 \Leftrightarrow q(d_1 : s_1,\ k) = q(d_2 : s_2,\ k)$ and $p_1 \geq p_2$;
3. $\sigma_1 \prec_A \sigma_2 \Leftrightarrow \sigma_1 \preceq_A \sigma_2$ and $\sigma_1 \not\sim_A \sigma_2$;
4. $\sigma_1 \succeq_A \sigma_2 \Leftrightarrow \sigma_2 \preceq_A \sigma_1$;
5. $\sigma_1 \succ_A \sigma_2 \Leftrightarrow \sigma_2 \prec_A \sigma_1$.

Rule 1) says that two signatures are equally preferred if and only if they represent exactly the same destination service prefix. Rule 2) states that one signature is more-or-equally preferred than the other one, if and only if they share a common prefix initial of length $min(p_1,\ p_2)$ and $p_1 \geq p_2$ (i.e., the methodology of longest prefix matching). The rest are defined based on previous rules. $\preceq_A$ is a *partial order* on Σ_A since the property – for every $\sigma_1, \sigma_2 \in \Sigma_A$, either $\sigma_1 \preceq_A \sigma_2$ or $\sigma_2 \preceq_A \sigma_1$ – does not hold. As a counterexample, given $\sigma_1 = (d : 25/48,\ S_1)$ and $\sigma_2 = (d : 80/48,\ S_2)$, we say that σ_1 and σ_2 are not comparable since they represent different services advertised by host d. The two destination service prefixes are never compared during route selection.

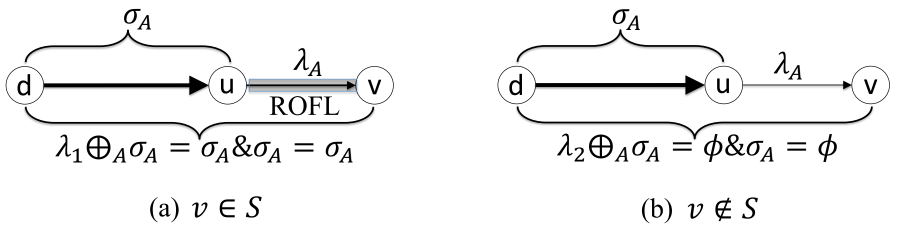

σ_A
λ_A
d
u
v
ROFL
$\lambda_1 \oplus_A \sigma_A = \sigma_A \& \sigma_A = \sigma_A$
(a) $v \in S$
σ_A
λ_A
d
u
v
$\lambda_2 \oplus_A \sigma_A = \phi \& \sigma_A = \phi$
(b) $v \notin S$

Figure 6.3: Mapping function $\oplus_A$ for programmatic label L_A and signature set Σ_A in the packet filter algebra A.

Routing algebra requires a label application function $\oplus_A$ that maps $L_A \times \Sigma_A$ to Σ_A, such that a signature is a collection of labels of its constituent links. Since policy algebra provides mechanism to describe the propagation and enforcement of ROFL policies, we can build $\oplus_A$ using basic algebraic operations. Here each label $\lambda_A \in L_A$ represents the propagation of ROFL policy from node u to adjacent node v, modeled using propagation (θ). O_A contains signature representation of ROFL policies initiated by destination d, and $O_A = \Sigma_A - \{\phi\}$. Thus, the mapping function $\oplus_A$, modeled using conjunction (&), describes the propagation of O_A from d along certain path. We further extend L_A to programmatic labels, based on the propagation rule defined in Section 3. Let p be a predicate over a signature set Σ_A and $\lambda_1, \lambda_2 \in L_A$, then the conditional label $\lambda_A =$ if p then λ_1 else λ_2 is in L_A and if $\lambda_1 = \sigma_A$ and $\lambda_2 = \phi$, such that

$$\lambda_A \oplus_A \sigma_A = \begin{cases} \lambda_1 \oplus_A \sigma_A = \sigma_A \& \sigma_A = \sigma_A & \text{if } p(\sigma_A) \\ \lambda_2 \oplus_A \sigma_A = \phi \ \& \sigma_A = \phi & \text{otherwise} \end{cases}$$

where predicate $p(\sigma_A)$ is a boolean formula, such that it returns *true* if and only if $v \in S$; otherwise it returns *false*. That is, ROFL policy described in signature σ_A flows from u to v iff $\theta^v R = R$, i.e., v is an authorized node to access service s advertised by d; otherwise, $\theta^v R = \phi$, v is not allowed to see this announcement, and the ϕ signature denotes a prohibited path. Figure 6.3 depicts a graphic representation of the mapping function $\oplus_A$. Think arrow denotes a path, thin arrow denotes a link, and the shaded link represents the propagation of ROFL announcement. Again routing information flows from d to v, but the actual network traffic traverses in reverse direction.

Shortest-path Algebra B We choose a simple shortest-path algebra to model the second component of the compound algebra $A \otimes B$. According to [Sobrinho, 2003], a shortest-path algebra can be represented as $B = \langle \mathbb{R}^+ \cup \{+\infty\},\ \leq,\ \mathbb{R}^+,\ +,\ \mathbb{R}\rangle$. Labels and signatures represent lengths: labels are positive real numbers, and signatures are positive real numbers adjoined with the special element $+\infty$; the signature of a path is obtained by simply adding the labels of its constituent links; and the signatures are compared with the less-than-or-equal order.

Lexical Product $A \otimes B$ Given sub-algebras A and B, we now construct a new routing algebra $A \otimes B$ using lexical product, such that $\Sigma = ((\Sigma_A - \{\phi\}) \times (\Sigma_B - \{\phi\})) \cup \{\phi\}$ and $\sigma = \langle \sigma_A,\ \sigma_B\rangle$ for every $\sigma \in \Sigma$. Similarly, $L = L_A \times L_B$ and $\lambda = \langle \lambda_A,\ \lambda_B\rangle$ for every $\lambda \in L$. The preference relation $\preceq$ is defined as follows:

$$\langle \sigma_A,\ \sigma_B\rangle \preceq \langle \beta_A,\ \beta_B\rangle \overset{def}{=} \sigma_A \preceq_A \beta_A \text{ or } (\sigma_A \sim_A \beta_A \text{ and } \sigma_B \preceq_B \sigma_B),$$

$$\langle \sigma_A,\ \sigma_B\rangle \prec \langle \beta_A,\ \beta_B\rangle \overset{def}{=} \sigma_A \prec_A \beta_A \text{ or } (\sigma_A \sim_A \beta_A \text{ and } \sigma_B \prec_B \sigma_B).$$

Thus we have $\langle \sigma_A,\ \sigma_B\rangle \prec \phi$ for all $\langle \sigma_A,\ \sigma_B\rangle \in \Sigma$. The label application function $\oplus$ applies labels to product signatures in a pair-wise way:

$\oplus$	$\langle \sigma_A,\ \sigma_B\rangle$
$\langle \lambda_A,\ \lambda_B\rangle$	$\langle \lambda_A \oplus_A \sigma_A,\ \lambda_B \oplus_B \sigma_B\rangle$

where $\lambda_i \in L_i$ and $\sigma_i \in \Sigma_i$. If either $\lambda_A \oplus_A \sigma_A$ or $\lambda_B \oplus_B \sigma_B$ is ϕ, then $\langle \lambda_A,\ \lambda_B\rangle \oplus \langle \sigma_A,\ \sigma_B\rangle = \phi$. In addition, we always have $\langle \lambda_A,\ \lambda_B\rangle \oplus \phi = \phi$.

The definition of preference relation $\preceq$ indicates that the packet filter algebra A dominates the comparison of two usable signatures. That is, firewall policies embedded in ROFL announcements produce a permitted path to route legitimate traffic; shortest-path algebra breaks the tie by choosing the least expensive one when multiple paths are available. In Figure 6.4, host d announces two ROFL policies: $R_1 = \{d : 80/48, \{b, c, e, f\}, M_1\}$ and $R_2 = \{d/32, \{a, b, c, e, f\},\ M_2\}$. Suppose sub-algebra B simply uses hop count as the routing metric. Although path $p_2 = fad$ appears shorter, HTTP traffic from f to d will take path $p_1 = fecbd$ since $\sigma_1 = \langle (d : 80/48, \{b, c, e, f\}),\ 4\rangle \prec \sigma_2 = \langle (d/32, \{a, b, c, e, f\}),\ 2\rangle$.

6.3.2 Protocol Convergence

Protocol convergence can be guaranteed by an important algebraic property called *strict monotonicity* [Sobrinho, 2003], which states that the weight of a path must increase when extended, i.e.,

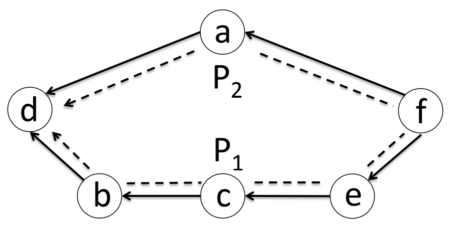


Figure 6.4: An example of path selection.

for all $\sigma \in \Sigma - \{\phi\}$ and $\lambda \in L$, $\sigma \prec \lambda \oplus \sigma$. We have:

$$\begin{aligned}\lambda \oplus \sigma &= \langle \lambda_A,\ \lambda_B\rangle \oplus \langle \sigma_A,\ \sigma_B\rangle = \langle \lambda_A \oplus_A \sigma_A,\ \lambda_B \oplus_B \sigma_B\rangle \\ &= \begin{cases} \langle \sigma_A,\ \lambda_B \oplus_B \sigma_B\rangle & \text{if } p(\sigma) \\ \langle \phi,\ \lambda_B \oplus_B \sigma_B\rangle = \phi & \text{otherwise}\end{cases}\end{aligned}$$

Therefore, if predicate $p(\sigma)$ holds, $\sigma = \langle \sigma_A,\ \sigma_B\rangle \prec \lambda \oplus \sigma = \langle \sigma_A,\ \lambda_B \oplus_B \sigma_B\rangle$, because $\sigma_A \sim \sigma_A$ and $\sigma_B \prec \lambda_B \oplus_B \sigma_B$; otherwise $\sigma \prec \lambda \oplus \sigma = \phi$. For the packet filter algebra A, monotonicity property holds since $\sigma_A \preceq \lambda_A \oplus_A \sigma_A$. The property of monotonicity preservation for the lexical product discussed in [Griffin and Sobrinho, 2005] shows that:

A	B	$A \otimes B$
M	M	M
M	SM	SM

Hence, given a monotonic algebra A, the monotonicity property of lexical product $A \otimes B$ is determined by B. This is also demonstrated by our chosen protocol example. Therefore, by Definition 6.1.1 we conclude that ROFL scheme helps preserve the convergence property of the underlying routing protocol.

6.3.3 Path Optimality

Another useful requirement is to study the property of paths that a routing protocol converges to. [Sobrinho, 2003] shows that a routing protocol always converges to optimal paths if and only if the algebra is *isotonic*. The relationship between two signatures with the same origin is preserved when both are extended to the same node, i.e., for any $\sigma, \beta \in \Sigma$ and $\lambda \in L$, if $\sigma \preceq \beta$ then $\lambda \oplus \sigma \preceq \lambda \oplus \beta$.

In our compound algebra, suppose we have $\sigma = \langle \sigma_A,\ \sigma_B\rangle \preceq \beta = \langle \beta_A,\ \beta_B\rangle$, and both are extended by a label $\langle \lambda_A,\ \lambda_B\rangle \in L$. The goal is to show that whether the following property holds:

$\langle \lambda_A \oplus_A \sigma_A,\ \lambda_B \oplus_B \sigma_B \rangle \preceq \langle \lambda_A \oplus_A \beta_A,\ \lambda_B \oplus_B \beta_B \rangle$. Although both paths are from the same origin, the programmatic label application could yield different results based on the evaluation results of $p(\sigma_A)$ and $p(\beta_A)$. As a counterexample, given two paths p_1 and p_2 with signatures $\sigma_A = (d : s_1/p_1,\ S_1)$ and $\beta_A = (d : s_2/p_2,\ S_2)$ from d to u, such that $\sigma_A \prec \beta_A$. If $v \notin S_1$ and $v \in S_2$, extending both paths by the same link (u, v) yields $\lambda_A \oplus_A \sigma_A = \phi \succ \lambda_A \oplus_A \beta_A = \beta_A$. Hence, the isotonicity property of lexical product $A \otimes B$ does not always hold. Therefore, by Definition 6.1.2 we conclude that the paths that $A \otimes B$ converges to are local-optimal paths.

6.4 Security Properties

Routing properties discussed previously say nothing about whether this distributed enforcement of firewall policies indeed achieves the desired filtering effect by meeting the global policy requirement in a network despite of topology changes. We call it the *security consistency* property of ROFL. Theorem 5.3.1 and 5.3.2 in Chapter 5 state formally that only packets initiated from authorized nodes can reach their destinations in absence of the transit node problem and despite of topology changes. We now describe an algorithm for verifying the correctness of ROFL firewalls by calculating the distributed filtering effect in a given network.

6.4.1 An Algorithm for Calculating Filtering Effect

Suppose destination host d announces a ROFL policy R in a given network $G = (V, E)$, where V is the set of vertices and E is the set of edges. The distributed policy enforcement mechanism consistently implements packet filtering requirement if and only if for each $v \in V$ and $v \in R.S$, $R_e = R$ (R_e is the effective ROFL policy received at node v); for each $v \in V$ but $v \notin R.S$, $R_e = \phi$.

Figure 6.5 shows a variation of Dijkstra's shortest path algorithm [Dijkstra, 1959] for a single source. Instead of using weights of edges as the only routing metric, we introduce a compound metric $d[v] = (r, m)$ for each $v \in V$, where r is the ROFL policy received at node v and m is the weight of a constructed path, such that $d[v].r \in \{R, \phi\}$ ($R < \phi$) and $d[v].m \in \mathbb{R}^+ \cup \{\infty\}$. If $d[v].r = R$, v must be an authorized node to access the advertised service and there exists a path to successfully route traffic initiated from v to destination d; otherwise, such a path cannot be constructed. DIJKSTRA-ROFL takes as input a network G, a set of link weights w, a destination

```
DIJKSTRA-ROFL($G, w, d, R$)
1  INITIALIZE-SINGLE-SOURCE($G, d, R$)
2  $T \leftarrow \emptyset$
3  $Q \leftarrow V$
4  while $Q \neq \emptyset$
5      do $u \leftarrow$ EXTRACT-MIN($Q$)
6          $T \leftarrow T \cup \{u\}$
7          for each vertex $u \in Adj[u]$
8              do RELAX($u, v, w, R$)
```

```
INITIALIZE-SINGLE-SOURCE($G, d, R$)
1  for each vertex $v \in V$
2      do $d[v] \leftarrow (\phi, \infty)$
3          $\pi[v] \leftarrow$ NIL
4  $d[d] \leftarrow (R, 0)$
```

```
EXTRACT-MIN($Q$)
1  $u \leftarrow$ NIL
2  $min \leftarrow (\phi, \infty)$
3  for each vertex $v \in Q$
4      if $d[v].r < min.r$ or
5          ($d[v].r = min.r$ and
            $d[v].m < min.m$)
6          then $u \leftarrow v$
7               $min \leftarrow d[v]$
8  $Q \leftarrow Q \backslash \{u\}$
9  return $u$
```

```
RELAX($u, v, w, R$)
1  $tmp.r \leftarrow \phi$
2  $tmp.m \leftarrow \infty$
3  if $d[u].r = R$ and $v \in R.S$
4      then $tmp.r \leftarrow d[u].r$
5  $tmp.m \leftarrow d[u].m + w(u, v)$
6  if $tmp.r < d[v].r$ or
7     ($tmp.r = d[v].r$ and
       $tmp.m < d[v].m$)
8      then $d[v] \leftarrow tmp$
9           $\pi[v] \leftarrow u$
```

Figure 6.5: An algorithm for calculating distributed filtering effect.

host d and a ROFL policy R announced by d. Distance to destination $d[d]$ is initialized to $(R, 0)$. For every other node $v \in V$, $d[v] = (\phi, \infty)$ indicating a path from v to access the advertised service is not yet available. Initially, the set of selected vertices T is empty and the queue of unresolved nodes Q is equal to V. While the queue of unsolved nodes Q is not empty, each time a vertex u with minimal distance $d[u]$ is selected from Q to join set T. The rule of comparing two distances is described in line 4-7 of method EXTRACT-MIN, such that metric $d[v].r$, describing a ROFL policy,

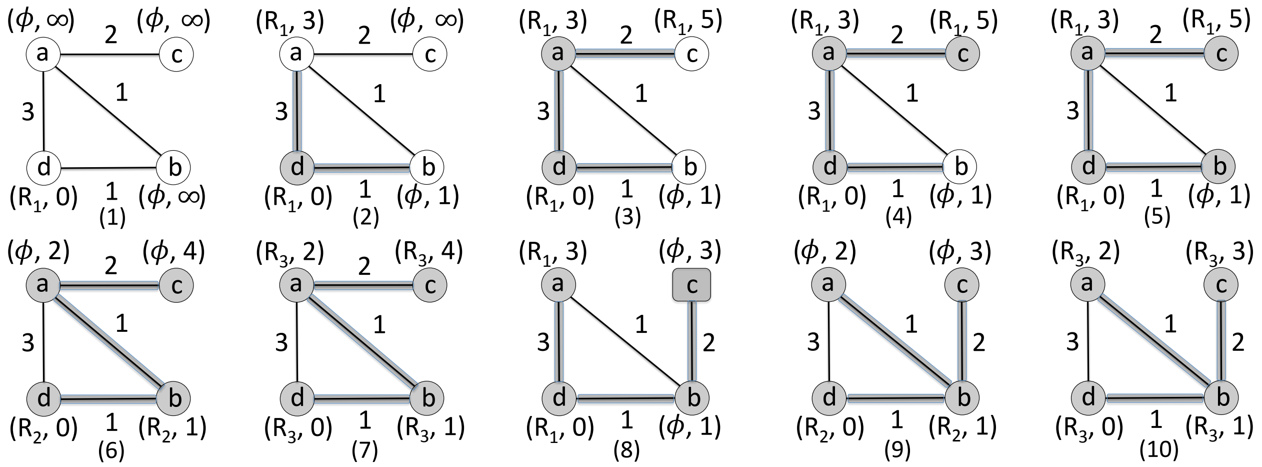


Figure 6.6: Examples of calculating distributed filtering effect for ROFL policy R_1, R_2 and R_3 announced by node d under two different network topologies.

is given higher precedence over conventional routing metric $d[v].m$ to ensure a usable path (e.g., $d[u] = (R, 4) < d[v] = (\phi, 3)$). Once a node u with minimal distance is selected, all its adjacent nodes execute a relaxation method to recalculate shortest path if necessary. If and only if both $d[u].r = R$ and $v \in R.S$, R can propagate from u to v. If $d[u].r \neq R$ but $v \in R.S$, traffic originated from an authorized node v will be dropped by u as it has no knowledge on how to route this packet to d, i.e., a transit node problem occurs as discussed in Chapter 5. The algorithm halts when all vertices are selected to T. Therefore, a consistent distributed enforcement guarantees that for each $v \in V$, if $v \in R.S$ then $d[v].r = R$; otherwise $d[v].r = \phi$ without presence of the transit node problem. If the problem indeed happens, i.e., $v \in R.S$ but $d[v].r = \phi$, a careful inspection on $\pi[v]$ and it predecessors along the path will reveal the exact point where transit node occurs.

The Dijkstra's algorithm has a time complexity of $O(|V|^2)$ for relatively dense graphs. By suitable choice of data structures for implementation, the running time can be reduced to $O(e \log_2 |V|)$ [Johnson, 1977; Aho *et al.*, 1974]. Assume each destination announces k services, the overall complexity of our modified algorithm is $O(k|V| \cdot e \log_2 |V|)$, where k is relatively small (see a detailed analysis on ROFL in Chapter 5).

6.4.2 Examples

We demonstrate the effectiveness of our algorithm using examples shown in Figure 6.6. Destination d announces three ROFL policies:

$$\begin{aligned} R_1 &= \{d : s_1/48,\ \{a, c\}, M_1\}, \\ R_2 &= \{d : s_2/48,\ \{b\},\ M_2\}, \\ R_3 &= \{d/32,\ \{a, b, c\},\ \infty\}. \end{aligned}$$

Subgraphs (1)-(5) illustrate the process of propagating R_1 in the network. For each step, shaded nodes represent those selected in T, and shaded links denote shortest paths calculated so far. The final output is in subgraph (5), where node a is authorized to access service s_1 with a shortest path of length 3, node b is unauthorized, and node c can access service s_1 with a shortest path of length 5 via a. Since we have $a \in R_1.S$ and $d[a].r = R_1$, $b \notin R_1.S$ and $d[b].r = \phi$, $c \in R_1.S$ and $d[c].r = R_1$, we conclude that the distributed filtering effect of R_1 is consistent with d's policy requirement. Similar conclusions can be drawn in subgraphs (6) and (7) for R_2 and R_3 respectively. In distributed environments, like a MANET, links can go up and down that triggers rapid topology changes. Suppose link (a, c) fails and is replaced by link (b, c). Re-running the algorithm yields the results shown in subgraphs (8)-(10). In subgraph (8), propagation of R_1 results in an unpleasant output $d[c] = (\phi, 3)$, such that an authorized node c can no longer access service s_1. That is an alert for the occurrence of the transit node problem. A careful investigation along the path from c to d reveals that $\pi[v] = b$, $d[b].r = \phi$, $\pi[b] = d$ and $d[d].r = R_1$, i.e., unauthorized node b causes the transit node problem.

6.5 Discussion

In this chapter, we combine two proposed mechanisms – ROFL firewall (Chapter 5) and policy algebra (Chapter 3) – with routing algebra in a unified approach. Policy algebra provides the theoretical foundation for policy composition and delegation in a distributed environment; the ROFL scheme is a concrete application for the concept of policy delegation in the context of firewall policies; with the support of routing algebra, we are able to study the routing effect of extending existing routing protocols to implement ROFL mechanism. This unification demonstrate that: ROFL poses no harm to existing routing protocols; it preserves convergency of the underlying routing protocol; the paths to which the protocol converges are also local-optimal. Moreover, policy algebra addresses the effect of policy composition at individual enforcement points, that facilitates the study of policy correctness and consistency – the two fundamental challenges in any distributed enforcement schemes.

To the best of our knowledge, we are the first to study source address policy routing enabled by ROFL using a theoretical approach by extending the routing algebra to a new level. In addition, we describe an algorithm for calculating network filtering effect, and demonstrate that for distributed policy enforcement, each node in the network enforces the correct security policy compliant with the global policy requirements. Moreover, the algorithm can automatically raise an alert when undesired situations, like the transit node problem, occur due to topology changes.

Part IV

Conclusions

Chapter 7

Conclusions

In this dissertation we considered the problem of specifying and enforcing security policies in large-scale distributed systems. We took the policy based management approach by focusing on the essential steps that a high-level policy must go through for implementation at the enforcement point. Initially, a high-level service and security requirement must be refined to low-level mechanism rules such that the syntax and semantics can be interpreted by the end devices. During the refinement process, analytical tasks must be performed to ensure that the enforced rules are conflict-free, not redundant and cover the entire problem space. The refined rules are then distributed to individual enforcement points for implementation. We claim that given the increasing complexity of a distributed environment, policy distribution and enforcement should be an informed decision made by system administrators such that the tradeoffs are tuned carefully and the overall security and system performance are improved.

7.1 Results Summary

More specifically, we answered the following questions in this dissertation:

1. *How to translate a high-level security requirement into low-level implementable rules?* As the first step to answer this question, we proposed a centralized policy refinement scheme for network service policies. It illustrates three key components in any refinement solution from the system perspective: a knowledge database, a set of refinement rules and a policy repository. The refinement solution also demonstrates the necessity of producing lower level access control

primitives through policy transformation in order to generate mechanism-specific rules using policy composition. Following the successful implementation of the centralized scheme, we further identified challenges and requirements in distributed policy scenario, and proposed a decentralized refinement solution for access control policies in general. The distributed approach focused on partial refinement and policy distribution for a more scalable solution in large computer networks. In addition, policy refinement must be interwoven with policy analysis to ensure that the refined rules will not produce any conflicts that can result in an incorrect and unstable system.

2. *Is it possible to take advantage of both centralized and end-to-end enforcement approaches for a more flexible solution to address the increasing complexity of distributed systems?* We proposed a policy algebra framework to handle this complexity. The framework consists of five basic algebraic operations for policy integration and analysis in a distributed policy scenario. More importantly, it enables policy delegation that allows multiple arrangements and enforcements of policies to tune the tradeoffs between the cost and risk of enforcing policies and hence to improve the overall system security and performance. The algebra defines high-level semantics of the operations, and hides their low-level implementation details which are highly language-specific. As concrete instantiations, we extend the framework to support firewall policies and Ponder policies respectively.

3. *Is it possible to utilize policy delegation to implement a more effective and efficient firewall mechanism for resource-constrained environment like a MANET?* We proposed a novel firewall system, called ROFL, that implements packet filtering using the underlying routing techniques. ROFL turns every route along the path to destination to a firewall by delegating firewall policy to these nodes. This form of ubiquitous enforcement is able to drop malicious traffic as early as possible to save transmission bandwidth and battery power, especially for MANETs with limited resources. The improved performance of ROFL has been verified by empirical studies in a simulated environment using two popular ad hoc routing protocols with minor changes in their route propagation and packet forwarding phases.

4. *How do we know that the proposed firewall mechanism does no harm to the underlying routing protocol and always enforces correct and consistent policies despite of topology changes?* We

unified three mechanisms – policy algebra, ROFL, and routing algebra – to study the routing properties and security properties of the ROFL system. Routing algebra demonstrates that protocols with ROFL extension always converge to local-optimal paths by examining the monotonicity and isotonicity properties of ROFL. Policy algebra provides formalisms to study the effect of composing multiple policies at a single place in a routing system, and proves that firewall policies are enforced correctly and consistently despite of topology changes without the presence of transit node problem.

7.2 Limitations and Future Work

The policy algebra framework proposed in this thesis is restricted to a limited semantics. It focuses on access control policies that can be specified using the basic subject, target, action components. We instantiate the algebraic framework to support packet filter rules and XACML policies respectively. However, the instantiations of the framework to many other security policy languages are not trivial. Policy manipulation and management using the proposed framework require careful consideration on the special characteristics of the policy language semantics. For instance, the instantiation for packet filter rules must take account of the ordering and correlation of firewall rules.

As a concrete application of policy delegation introduced in the algebra framework, we designed and implemented the ROFL scheme - a novel firewall mechanism that implements packet filtering using the underlying routing techniques. Although the proposed ROFL scheme concentrates on ingress filtering, it can be easily extended to support egress access control. ROFL routes can be carefully configured on end hosts to restrict their access to outside service providers or proxies. For example, end hosts within an enterprise network are only allowed to access web server and mail server proxies if their local routing tables are configured to store only service specific routes to the proxies and everything else is blocked. Meanwhile, source prefix filtering designed for ingress filtering can also be applied to egress access control to ensure that outgoing traffics are always generated from a legitimate range of source addresses to mitigate address spoofing problem. On the other hand, the current ROFL scheme is aimed at filtering individual packets based upon source/destination addresses and port numbers. Control based upon behavioral/volumetric constraints requires additional examination on the IP packet as well as the connection flow. We consider extending the basic ROFL scheme to support stateful and application level filtering as an interesting and promising future research direction. In addition, this approach is unlikely to scale to the Internet and possibly not to the cloud given the large number of service specific routes managed. In fact, to minimize the impact of maintaining more and longer service specific routes, the concept of aggregation points were introduced in the ROFL scheme, such that longer routes are aggregated and a shorter prefix is announced to the outside world instead.

In this thesis, we focus on several essential tasks from the policy based management stack. It is equally important to study the interactions between them. For example, policy refinement must be interwoven with policy analysis to ensure a conflict-free rule set; policy distribution plays a crucial role in the refinement process to facilitate partial refinement. However, there are missing links from the policy chart. Most of the existing solutions do not include a step of providing deployment feedback to policy authors for adjusting their initial requirements and goals. The urgency of an iterative, interactive and incremental policy based management solution still remains.

Solutions proposed in this thesis are for security policy management specifically. We believe the same requirements and principles apply for other types of policies as well, such as network configuration rules, intrusion detection policies, etc. One promising future research direction is to examine how the proposed solutions can be applied to other policy domains.

Part V

Bibliography

Bibliography

[Abadi and Lamport, 1988] Martn Abadi and Leslie Lamport. The existence of refinement mappings. *Theoretical Computer Science*, 82:253–284, 1988.

[Agrawal *et al.*, 2005] Dakshi Agrawal, James Giles, Kang won Lee, and Jorge Lobo. Policy ratification. In *Proceedings of IEEE Policy 2005*, 2005.

[Aho *et al.*, 1974] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, 1974.

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

[Alicherry and Keromytis, 2010] Mansoor Alicherry and Angelos D. Keromytis. Diploma: Distributed policy enforcement architecture for manets. In *Proceedings of the 2010 Fourth International Conference on Network and System Security*, NSS '10, pages 89–98, Washington, DC, USA, 2010. IEEE Computer Society.

[Backes *et al.*, 2004] M. Backes, M. Duermuth, and R. Steinwandt. An algebra for composing enterprise privacy policies. In *9th European Symposium on Research in Computer Security*, 2004.

[Bai *et al.*, 2003] Fan Bai, Narayanan Sadagopan, and Ahmed Helmy. IMPORTANT: A framework to systematically analyze the Impact of Mobility on Performance of RouTing protocols for Adhoc NeTworks. In *Proc. of Infocom*, 2003.

[Baldwin, 1990] Robert W. Baldwin. Naming and grouping privileges to simplify security management in large databases. *Security and Privacy, IEEE Symposium on*, 0:116, 1990.

[Bandara *et al.*, 2004] Arosha K Bandara, Emil C Lupu, Jonathan Moffett, and Alessandra Russo. A goal-based approach to policy refinement. In *POLICY '04: Proceedings of the 5th IEEE International Workshop on Policies for Distributed Systems and Networks*, page 229. IEEE Computer Society, 2004.

[Bandara *et al.*, 2005] Arosha K. Bandara, Emil C. Lupu, Alessandra Russo, Naranker Dulay, Morris Sloman, Paris Flegkas, Marinos Charalambides, and George Pavlou. Policy refinement for diffserv quality of service management. In *IM 2005: Proceedings of the 9th IFIP/IEEE International Symposium on Integrated Network Management*, 2005.

[Bandara *et al.*, 2006] Arosha Bandara, Tony Kakas, Emil Lupu, and Alessandra Russo. Using argumentation logic for firewall policy specification and analysis. In *Proceedings of 17th IFIP/IEEE Distributed Systems: Operations and Management*, 2006.

[Bandara, 2005] Arosha Bandara. *A Formal Approach to Analysis and Refinement of Policies.* PhD thesis, 2005.

[Bartal *et al.*, 1999] Yair Bartal, Alain Mayer, Kobbi Nissim, and Avishai Wool. Firmato: A novel firewall management toolkit. In *Proc. IEEE Computer Society Symposium on Security and Privacy*, 1999.

[Beigi *et al.*, 2004] Mandis S. Beigi, Seraphin Calo, and Dinesh Verma. Policy transformation techniques in policy-based systems management. In *Proc. 5th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY*, pages 13–22, 2004.

[Bell and LaPadula, 1973] David E. Bell and Leonard J. LaPadula. Secure computer systems: Mathematical foundations and model. Technical Report MTR 2547, MITRE, November 1973. Volume 2.

[Bellovin, 1999] Steven M. Bellovin. Distributed firewalls. *;login:*, pages 39–47, November 1999.

[BGP, 2006] Bgp best path selection algorithm, 2006. `http://http://www.cisco.com/en/US/tech/tk365/technologies_tech_note09186a0080094431.shtml`.

[Biba, 1977] K. J. Biba. Integrity considerations for secure computer systems. Technical report, MITRE Corp., 04 1977.

[Blaze *et al.*, 1996] Matt Blaze, Joan Feigenbaum, and Jack Lacy. Decentralized trust management. In *In Proceedings of the 1996 IEEE Symposium on Security and Privacy*, pages 164–173. IEEE Computer Society Press, 1996.

[Blaze *et al.*, 1999] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. Keromytis. The KeyNote Trust-Management System Version 2. RFC 2704, Internet Engineering Task Force, September 1999.

[Bloom, 1970] B.H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of ACM*, 13(7):422–426, July 1970.

[bon, 2011] Bonnmotion, a mobility scenario generation and analysis tool, 2011.

[Bonatti *et al.*, 2002] Piero Bonatti, Sabrina De Capitani di Vimercati, and Pierangela Samarati. An algebra for composing access control policies. *ACM Trans. Inf. Syst. Secur.*, 5:1–35, February 2002.

[Braun, 1989] H.W. Braun. Models of policy based routing. RFC 1104, Internet Engineering Task Force, June 1989.

[Brewer and Nash, 1989] Dr. David F.C. Brewer and Dr. Micheal J. Nash. The chinese wall security policy. *Security and Privacy, IEEE Symposium on*, 0:206, 1989.

[Buchsbaum *et al.*, 2003] Adam L. Buchsbaum, Glenn S. Fowler, Balachander Krishnamurthy, Kiem-Phong Vo, and Jia Wang. Fast prefix matching of bounded strings. In *Proceedings of ACM SIGACT ALENEX03*, Baltimore, Maryland, January 2003.

[Camp *et al.*, 2002] Tracy Camp, Jeff Boleng, and Vanessa Davies. A Survey of Mobility Models for Ad Hoc Network Research. *Wireless Communication and Mobile Computing: Special issue on Mobile Ad Hoc Networking: Research, Trends and Applications*, 2(5):483–502, Sep. 2002.

[Campbell and Turner, 2008] Gavin A. Campbell and Kenneth J. Turner. Goals and policies for sensor network management. In *SENSORCOMM '08: Proceedings of the 2008 Second International Conference on Sensor Technologies and Applications*, pages 354–359, Washington, DC, USA, 2008. IEEE Computer Society.

[Casado *et al.*, 2006] Martin Casado, Tal Garfinkel, Aditya Akella, Michael J. Freedman, Dan Boneh, Nick Mckeown, and Scott Shenker. Sane: A protection architecture for enterprise networks. In *Usenix Security Symposium*, 2006.

[Casado *et al.*, 2007] Martin Casado, Michael J. Freedman, Justin Pettit, Jianying Luo, Nick McKeown, and Scott Shenker. Ethane: taking control of the enterprise. *SIGCOMM Comput. Commun. Rev.*, 37(4):1–12, 2007.

[Chadha *et al.*, 2004] Ritu Chadha, Hong Cheng, Yuu-Heng Cheng, Jason Chiang, A. Ghetie, Gary Levin, and Harshad Tanna. Policy-based mobile ad hoc network management. *Policies for Distributed Systems and Networks, IEEE International Workshop on*, 0:35, 2004.

[Cheswick *et al.*, 2003] William R. Cheswick, Steven M. Bellovin, and Aviel D. Rubin. *Firewalls and Internet Security; Repelling the Wily Hacker.* Addison-Wesley, Reading, MA, second edition, 2003.

[Chiang *et al.*, 2006] Cho-Yu Jason Chiang, Stephanie Demers, Praveen Gopalakrishnan, Latha Kant, Alex Poylisher, Yuu-Heng Cheng, Ritu Chadha, Gary Levin, Shihwei Li, Yibei Ling, Scott Newman, Lorraine LaVergne, and Richard Lo. Performance analysis of drama: a distributed policy-based system for manet management. In *Proceedings of the 2006 IEEE conference on Military communications*, MILCOM'06, pages 272–279, Piscataway, NJ, USA, 2006. IEEE Press.

[Clausen and Jacquet, 2003] T. Clausen and P. Jacquet. Optimized Link State Routing Protocol (OLSR). RFC 3626, Internet Engineering Task Force, October 2003.

[Clocksin and Mellish, 1984] William F. Clocksin and Christopher S. Mellish. *Programming in Prolog.* 1984.

[Craven *et al.*, 2009] Robert Craven, Jorge Lobo, Emil Lupu, Alessandra Russo, and Morris Sloman. Security policy refinement using data integration: a position paper. In *SafeConfig '09: Proceedings of the 2nd ACM workshop on Assurable and usable security configuration*, pages 25–28, New York, NY, USA, 2009. ACM.

[Craven *et al.*, 2010a] Robert Craven, Jorge Lobo, Emil Lupu, Jiefei Ma, Alessandra Russo, and Morris Sloman. Distributed policy scenario. ITA technical report, 2010.

[Craven *et al.*, 2010b] Robert Craven, Jorge Lobo, Emil Lupu, Alessandra Russo, and Morris Sloman. Decomposition techniques for policy refinement. In *To appear in proc. of the 6th International Conference on Network and Service Management*, 2010.

[Damianou *et al.*, 2001] Nicodemos Damianou, Naranker Dulay, Emil Lupu, and Morris Sloman. The ponder policy specification language. In *Proceedings of IEEE Policy 2001*, 2001.

[Davy, 2008] Steven Davy. *Harnessing Information Models and Ontologies for Policy Conflict Analysis.* PhD thesis, 2008.

[Dijkstra, 1959] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[ecl, 2012] Eclipse, 2012. `http://www.eclipse.org/`.

[Estrin *et al.*, 1988] Deborah Estrin, Jeffrey Mogul, Gene Tsudik, and Kamaljit Anand. Visa protocols for controlling inter-organizational datagram flow. *IEEE Journal on Selected Areas in Communications*, 7, 1988.

[Ferraiolo *et al.*, 2001] David F. Ferraiolo, Ravi Sandhu, Serban Gavrila, D. Richard Kuhn, and Ramaswamy Chandramouli. Proposed nist standard for role-based access control. *ACM Trans. Inf. Syst. Secur.*, 4:224–274, August 2001.

[Fuller and Li, 2006] V. Fuller and T. Li. Classless Inter-domain Routing (CIDR): The Internet Address Assignment and Aggregation Plan. RFC 4632, Internet Engineering Task Force, August 2006.

[glo, 2000] GloMoSim: Global mobile information systems simulation library, 2000.

[Graham and Denning, 1972] G. Scott Graham and Peter J. Denning. Protection - principles and practice. *Managing Requirements Knowledge, International Workshop on*, 0:417, 1972.

[Greenwald, 1996] Steven J. Greenwald. A new security policy for distributed resource management and access control. In *NSPW '96: Proceedings of the 1996 workshop on New security paradigms*, pages 74–86, New York, NY, USA, 1996. ACM.

[Grefen and Widom, 1996] Paul Grefen and Jennifer Widom. Protocols for integrity constraint checking in federated databases. In *Proceedings 1st IFCIS International Conference on Cooperative Information Systems*, pages 38–47, 1996.

[Griffin and Sobrinho, 2005] Timothy G. Griffin and Joäo Luís Sobrinho. Metarouting. *SIGCOMM Comput. Commun. Rev.*, 35(4):1–12, September 2005.

[Hari *et al.*, 2000] A. Hari, S. Suri, and G. Parulkar. Detecting and resolving packet filter conflicts. In *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, March 2000.

[Harrison *et al.*, 1975] Michael A. Harrison, Walter L. Ruzzo, and Jeffrey D. Ullman. On protection in operating systems. *SIGOPS Oper. Syst. Rev.*, 9:14–24, November 1975.

[Hong *et al.*, 1999] Xiaoyan Hong, Mario Gerla, Guangyu Pei, and Ching-Chuan Chiang. A Group Mobility Model for Ad Hoc Wireless Networks. *Proc. of the ACM Int. Workshop on Modelling and Simulation of Wireless and Mobile Systems (MSWiM)*, pages 53–60, 1999.

[Hubaux *et al.*, 2001] Jean-Pierre Hubaux, Levente Buttyán, and Srdan Capkun. The quest for security in mobile ad hoc networks. In *MobiHoc '01: Proceedings of the 2nd ACM international symposium on Mobile ad hoc networking & computing*, pages 146–155, New York, NY, USA, 2001. ACM.

[Ioannidis *et al.*, 2000] Sotiris Ioannidis, Angelos D. Keromytis, Steve M. Bellovin, and Jonathan M. Smith. Implementing a distributed firewall. In *Proceedings of the 7th ACM conference on Computer and communications security*, CCS '00, pages 190–199, New York, NY, USA, 2000. ACM.

[Jajodia and Wijesekera, 2002] Sushil Jajodia and Duminda Wijesekera. Recent advances in access control models. In *Proceedings of the fifteenth annual working conference on Database and application security*, Das'01, pages 3–15, Norwell, MA, USA, 2002. Kluwer Academic Publishers.

[Jajodia *et al.*, 1997] Sushil Jajodia, Pierangela Samarati, and V. S. Subrahmanian. A logical language for expressing authorizations. In *Proceedings of the 1997 IEEE Symposium on Security and Privacy*, SP '97, pages 31–, Washington, DC, USA, 1997. IEEE Computer Society.

[Jajodia *et al.*, 2001] Sushil Jajodia, Pierangela Samarati, Maria Luisa Sapino, and V. S. Subrahmanian. Flexible support for multiple access control policies. *ACM Trans. Database Syst.*, 26:214–260, June 2001.

[Johnson *et al.*, 2010] Maritza Johnson, John Karat, Clare-Marie Karat, and Keith Grueneberg. Usable policy template authoring for iterative policy refinement. In *POLICY '10: Proceedings of the IEEE International Workshop on Policies for Distributed Systems and Networks, POLICY*, 2010.

[Johnson, 1977] Donald B. Johnson. Efficient algorithms for shortest paths in sparse networks. *J. ACM*, 24(1):1–13, 1977.

[Keromytis *et al.*, 2003] Angelos D. Keromytis, Sotiris Ioannidis, Michael B. Greenwald, and Jonathan M. Smith. The strongman architecture. In *3rd DARPA Information Survivability Conference and Exposition (DISCEX-III 2003), 22-24 April 2003, Washington, DC, USA*, pages 178–188. IEEE Computer Society, 2003.

[Kowalski *et al.*, 1987] Robert Kowalski, Fariba Sadri, and Paul Soper. Integrity checking in deductive databases. In *Proceedings of the VLDB International Conference*, pages 61–69. Morgan Kaufmann Publishers, 1987.

[Lampson, 1974] Butler W. Lampson. Protection. *SIGOPS Oper. Syst. Rev.*, 8(1):18–24, 1974.

[Lampson, 2004] Butler W. Lampson. Computer security in the real world. *Computer*, 37:37–46, 2004.

[Lupu and Sloman, 1997] E. Lupu and M. Sloman. Conflict analysis for management policies. In *Proceedings of the fifth IFIP/IEEE international symposium on Integrated network management V : integrated management in a virtual world: integrated management in a virtual world*, pages 430–443, London, UK, UK, 1997. Chapman & Hall, Ltd.

[Lupu and Sloman, 1999] E. Lupu and M. Sloman. Conflicts in policy-based distributed systems management. *IEEE Transactions on Software Engineering*, 25:852–869, 1999.

[Lupu *et al.*, 2007] Emil Lupu, Naranker Dulay, Alberto Schaeffer Filho, Sye Keoh, Morris Sloman, and Kevin Twidle. Amuse: Autonomic management of ubiquitous e-health systems. In *Concurrency and Computation: Practice and Experience, wiley*, 2007.

[Mayer *et al.*, 2000] A. Mayer, A. Wool, and E. Ziskind. Fang: A firewall analysis engine. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 177–187, 2000.

[Minsky and Ungureanu, 1998] Naftaly H. Minsky and Victoria Ungureanu. Unified support for heterogeneous security policies in distributed systems. In *In 7th USENIX Security Symposium*, 1998.

[Moffett and Sloman, 1993] Jonathan D. Moffett and Morris S. Sloman. Policy hierarchies for distributed systems management. *IEEE Journal on Selected Areas in Communications*, 11:1404–1414, 1993.

[Mogul, 1989] Jeffrey C. Mogul. Simple and flexible datagram access controls for unix-based gateways. In *USENIX Conference Proceedings*, pages 203–221, Baltimore, MD, Summer 1989.

[Moy, 1998] J. Moy. OSPF version 2. RFC 2328, Internet Engineering Task Force, April 1998.

[ns2, 2012] The Network Simulator - ns-2, 2012. `http://www.isi.edu/nsnam/ns/`.

[OASIS, 2012] OASIS. eXtensible Access Control Markup Language (XACML) TC, 2012.

[Pei *et al.*, 1999] Guangyu Pei, Mario Gerla, Xiaoyan Hong, and Ching chuan Chiang. Wireless hierarchical routing protocol with group mobility. In *Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC)*, 1999.

[Perkins *et al.*, 2003] C. Perkins, E. Belding-Royer, and S. Das. Ad hoc On-Demand Distance Vector (AODV) Routing. RFC 3561, Internet Engineering Task Force, July 2003.

[Perkins, 2001] Charles Perkins. *Ad hoc Networking*. Addison-Wesley, 2001.

[Pincus and Wing, 2005] J. Pincus and J.M. Wing. Towards an algebra for security policies. In *Proceedings of 26th International Conference on Applications and Theory of Petri Nets*, 2005.

[pro, 2012] ProDT: Prolog development tools, 2012. `http://prodevtools.sourceforge.net/`.

[Rao *et al.*, 2009] Prathima Rao, Dan Lin, Elisa Bertino, Ninghui Li, and Jorge Lobo. An algebra for fine-grained integration of XACML policies. In *Proceedings of the 14th ACM symposium on Access control models and technologies*, SACMAT '09, pages 63–72, New York, NY, USA, 2009. ACM.

[Rekhter *et al.*, 2006] Y. Rekhter, T. Li, and S. Hares. A border gateway protocol 4 (BGP-4). RFC 4271, Internet Engineering Task Force, January 2006.

[Rexford *et al.*, 2001] Jennifer Rexford, Steven M. Bellovin, and Randy Bush. Some initial measurements of prefix length philtres. NANOG talk, May 2001. `http://ran.psg.com/~randy/010521.nanog/index.htm`.

[Ribeiro *et al.*, 1999] Carlos Ribeiro, Andr Zqucte, Paulo Ferreira, and Paulo Guedes. Spl: An access control language for security policies with complex constraints. In *In Proceedings of the Network and Distributed System Security Symposium*, pages 89–107, 1999.

[Ritu Chadha, 2008] Cho-Yu Jason Chiang Ritu Chadha. Drama: Distributed policy management for MANETs. In *POLICY*, pages 235–237, 2008.

[Rubio-loyola *et al.*, 2006] Javier Rubio-loyola, Joan Serrat, Marinos Charalambides, Paris Flegkas, and George Pavlou. A functional solution for goal-oriented policy refinement. In *Proc. 7th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY)*, 2006.

[Samarati and de Capitani di Vimercati, 2001] Pierangela Samarati and Sabrina de Capitani di Vimercati. Access control: Policies, models, and mechanisms. In *FOUNDATIONS OF SECURITY ANALYSIS AND DESIGN (TUTORIAL LECTURES)*, pages 137–196. Springer Verlag, 2001.

[Sandhu *et al.*, 1996] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-based access control models. *Computer*, 29:38–47, 1996.

[Shaikh and Greenberg, 2001] Aman Shaikh and Albert Greenberg. Experience in black-box ospf measurement. In *ACM SIGCOMM Internet Measurement Workshop (IMW)*, November 2001.

[Sloman, 1994] Morris Sloman. Policy driven management for distributed systems. *Journal of Network and Systems Management*, 2:333–360, 1994.

[Sobrinho, 2003] João Luis Sobrinho. Network routing with path vector protocols: theory and applications. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '03, pages 49–60, New York, NY, USA, 2003. ACM.

[swi, 2011] SWI-Prolog, 2011. `http://www.swi-prolog.org/`.

[Thompson *et al.*, 1999] Mary Thompson, William Johnston, Srilekha Mudumbai, Gary Hoo, Keith Jackson, and Abdelilah Essiari. Certificate-based access control for widely distributed resources. In *SSYM'99: Proceedings of the 8th conference on USENIX Security Symposium*, pages 17–17, Berkeley, CA, USA, 1999. USENIX Association.

[Waldvogel, 2000] Marcel Waldvogel. *Fast Longest Prefix Matching: Algorithms, Analysis, and Applications*. PhD thesis, Swiss Federal Institute of Technology, Zurich, 2000.

[Wijesekera and Jajodia, 2003] Duminda Wijesekera and Sushil Jajodia. A propositional policy algebra for access control. *ACM Transactions on Information and System Security (TISSEC)*, 6(2):286–325, May 2003.

[Woo and Lam, 1993] Thomas Y. C. Woo and Simon S. Lam. Authorizations in distributed systems: A new approach. *Journal of Computer Security*, 2(2-3):107–136, 1993.

[Wool, 2001] Avishai Wool. Architecting the lumeta firewall analyzer. In *In 10th USENIX Security Symposium*, 2001.

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

[Yu and Braun, 1989] J.Y. Yu and H.W. Braun. Routing between the NSFNET and the DDN. RFC 1133, Internet Engineering Task Force, November 1989.

[Zhao and Bellovin, 2007] Hang Zhao and Steven M. Bellovin. Policy algebras for hybrid firewalls. Technical Report CUCS-017-07, Department of Computer Science, Columbia University, March 2007. Also presented at the Annual Conference of the ITA, 2007.

[Zhao *et al.*, 2008a] Hang Zhao, Chi-Kin Chau, and Steven M. Bellovin. ROFL: Routing as the firewall layer. In *New Security Paradigms Workshop*, September 2008. A version is available as Technical Report CUCS-026-08.

[Zhao *et al.*, 2008b] Hang Zhao, Jorge Lobo, and Steven M. Bellovin. An algebra for integration and analysis of Ponder2 policies. In *Proceeding of the 9th IEEE Workshop on Policies for Distributed Systems and Networks*, June 2008.

[Zhao *et al.*, 2011] Hang Zhao, Jorge Lobo, Arnab Roy, and Steven M. Bellovin. Policy refinement of network services for MANET. In *The 12th IFIP/IEEE International Symposium on Integrated Network Management (IM'11)*, May 2011.