Minimizing the Maximum Firewall Rule Set in a Network with Multiple Firewalls

MyungKeun Yoon, Shigang Chen, and Zhan Zhang

Abstract—A firewall's complexity is known to increase with the size of its rule set. Empirical studies show that as the rule set grows larger, the number of configuration errors on a firewall increases sharply, while the performance of the firewall degrades. When designing a security-sensitive network, it is critical to construct the network topology and its routing structure carefully in order to reduce the firewall rule sets, which helps lower the chance of security loopholes and prevent performance bottleneck. This paper studies the problems of how to place the firewalls in a topology during network design and how to construct the routing tables during operation such that the maximum firewall rule set can be minimized. These problems have not been studied adequately despite their importance. We have two major contributions. First, we prove that the problems are NP-complete. Second, we propose a heuristic solution and demonstrate the effectiveness of the algorithm by simulations. The results show that the proposed algorithm reduces the maximum firewall rule set by 2-5 times when comparing with other algorithms.

Index Terms—Firewall configuration, access control rules, network security.

1 Introduction

Firewalls are the cornerstones of corporate network security. Once a company acquires firewalls, the most crucial management task is to correctly configure the firewalls with security rules [1], [2]. A firewall's configuration contains a large set of access control rules, each specifying source addresses, destination addresses, source ports, destination ports, one or multiple protocol ids, and an appropriate action. The action is typically "accept" or "deny." Some firewalls can support other types of actions such as sending a log message, applying a proxy, and passing the matched packets into a VPN tunnel [3]. For most firewalls, the rule set is order-sensitive [4]. An incoming packet will be checked against the ordered list of rules. The rule that matches first decides how to process the packet. Other firewalls (such as early versions of Cisco's PIX) use the best-matching rule instead.

Due to the multidimensional nature of the rules (including source/destination addresses and ports), the performance of a firewall degrades as the number of rules increases. Commercially deployed firewalls often carry tens of thousands of rules, creating performance bottlenecks in the network. More importantly, the empirical fact shows that the number of configuration errors on a firewall increases sharply in the size of the rule set [5]. A complex rule set can easily lead to mistakes and mal-configuration. After analyzing the firewall rule sets from many organizations including telecommunication companies and financial institutes, Wool [5] quantified the complexity of a rule set as $R + O + \frac{I \times (I-1)}{2}$, where R is the number of rules in the set, O is the number of network objects referenced by the rules, and I is the number of network interfaces on the firewall. The number of network objects and the number of interfaces are normally much smaller than the number of rules. Therefore, it is very important to keep a firewall's rule set as small as possible in order to lower the chance of security loopholes [5]. In a network with multiple firewalls, reducing the number of rules requires not only local optimization at individual firewalls, but also global optimization across all firewalls. This paper studies how to minimize the maximum rule set among all firewalls in the network, which has not been adequately studied despite its importance in practice.

We investigate a family of related problems. The first one is about how to place the firewalls in a topology during network design. The so-called *firewall placement problem* (FPP) is to find the optimal placement of firewalls that connects a set of domains in such a way that minimizes the maximum number of rules on any firewall. However, allowing the complete freedom in topology construction may be rare in practice. It is more often the case that only limited freedom is available to be exploited for rule-set reduction, while most or the entire network topology is a given input. In the following, we extend FPP to a series of more practical problems.

The second problem, called as *partial FPP*, is to expand an existing topology with new firewalls and domains such that the maximum rule set remains minimized. This problem arises during incremental deployment or in case that a partial network topology has been determined based on more important performance criteria before firewall rule sets are considered. FPP is a special case of partial FPP (with an empty existing topology).

- *M. Yoon is with the Korea Financial Telecommunications and Clearings Institute, 717 YokSam-Dong, KangNam-Gu, Seoul 135-758, South Korea. E-mail: mkyoon@kftc.or.kr.*
- *S. Chen is with the Department of Computer and Information Science and Engineering, University of Florida, Gainesville, FL 32611. E-mail: sgchen@cise.ufl.edu.*
- *Z. Zhang is with Cisco Systems, Mail Stop SJC22/2/3, 821 Alder Drive, Milpitas, CA 95035. E-mail: zhazhang@cisco.com.*

Manuscript received 20 Feb. 2008; revised 17 Oct. 2008; accepted 12 May 2009; published online 3 Nov. 2009.
Recommended for acceptance by S.R. Murthy.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-2008-02-0082.
Digital Object Identifier no. 10.1109/TC.2009.172.

We now move to operational networks whose topologies have already been fully established. Our third problem, called *firewall routing problem* (FRP), is to establish the optimal routing paths on an *existing* network topology such that the maximum number of rules on any firewall is minimized. The fourth problem is called *partial FRP*. It assumes that the routing tables in the network have been *partially* populated based on other performance criteria (such as reliability and bandwidth utilization). For example, if bandwidth is the most important criterion, some routing entries should be selected to optimize the use of bottleneck links, but the choice of other entries may be flexible if alternative paths after bottleneck links are allowed (since end-to-end bandwidth is solely decided by the bottleneck). In this case, we can determine those routing entries by using the secondary criterion of minimizing the maximum firewall rule set.

Our fifth and sixth problems are called *weighted FPP/FRP*. We assign each rule a weight (possibly representing the volume of traffic covered by this rule), and also assign each firewall a weight (possibly representing the capacity of the firewall). The goal is to find the optimal network topology and/or routing paths that minimize the maximum weighted number of rules at any firewall. The solutions to the weighted problems take not only the number of rules, but also traffic distribution, firewall performance, and possibly other factors into consideration.

We have two major contributions. First, by reducing the well-known set-partition problem to the above problems, we prove that they are NP-complete. Second, we propose a heuristic algorithm to solve the FPP problem approximately. Not only does it construct a network topology among domains and firewalls, but also identify routing paths that minimize the maximum firewall rule set. The algorithm can be easily modified to solve partial FPP, FRP, partial FRP, weighted FPP, and weighted FRP. Hence, the algorithm can be used to construct a new topology, complete a topology that has been partially constructed (based on other performance criteria), expand an existing topology, or work on an established topology to build a new routing structure or complete an existing routing structure that has been partially populated (based on other performance criteria). We demonstrate its effectiveness by simulations, which show that the proposed algorithm achieves far better results than two other solutions. The maximum size of all firewall rule sets produced by our algorithm is 2-5 times smaller than those produced by others.

The rest of the paper is organized as follows: Section 2 defines the network model and the problems to be solved. Section 3 proves that the problems are NP-complete. Section 4 proposes a heuristic algorithm. Section 5 presents the simulation results. Section 6 surveys the related work. Section 7 draws the conclusion.

2 Problem Definition

2.1 Network Model

We consider a security-sensitive enterprise network consisting of domains (subnets) that are connected with each other through firewalls. We assume that intradomain security is appropriately enforced. This paper focuses on interdomain access control. We further assume that dynamic routing is turned off on firewalls, while static routes are used to direct interdomain traffic, which is today's common practice in banks or other institutions that have high-level security requirements. In fact, some popular firewalls (such as many Cisco PIX models) do not support dynamic routing protocols. With static routes, robustness is achieved by using dual firewalls, which will be discussed shortly. Using static routes on firewalls is a direct consequence of the high complexity in managing the security of a mesh network. It has a number of practical advantages. First, it ensures that traffic flows are going through their designated firewalls where appropriate security policies are enforced. Second, predictable routing paths simplify the security analysis in a complex network environment, and consequently, reduce the chance of error in firewall configuration. Third, most existing dynamic routing protocols are not secure. Counterfeit routing advertisement can divert traffic through insecure paths where the packets may be copied or tampered. Note that dynamic routing is still used inside each domain as long as it does not cross an interdomain firewall.

2.2 Notations

Let N be a set of n domains and M a set of m firewalls. Each firewall has two or more network interfaces. Different firewalls may have different numbers of interfaces. A network interface can be connected to any domain, forming a physical link between the firewall and the domain. In our model, two firewalls do not directly connect with each other because, otherwise, we would treat them as one firewall with combined interfaces; two domains do not directly connect with each other because, otherwise, we would treat them as one domain. Let e be the total number of network interfaces available on all firewalls. The maximum number of links in the topology is bounded by e. A network interface that has not been used to connect a domain is called a *free interface*.

Each domain has one address prefix. Static routes are defined to route interdomain traffic, which ensures that each traffic flow has a specific path going through certain firewall(s) where the security policy governing this flow will be enforced. In order to support stateful inspection, routing symmetry is assumed. It means that the routing path from domain x to domain y is the same as the path from y to x, $\forall x, y \in N$. This assumption is made to comply with Cisco's CBAC (context-based access control) and other firewalls' stateful inspection mechanisms, which allow the system administrator to only specify the rules for traffic from clients to servers, while the firewall automatically inserts the rules for the return traffic on the fly. CBAC requires that a connection uses the same (interdomain) path for two-way communication. We want to stress that this assumption is made only for practical reasons. Our analysis and algorithm design can be easily modified to work for asymmetric routing.

For each pair of domains $x, y \in N$, there is a set $R(x, y)$ of access control rules, defining the traffic flows that are permitted from domain x to domain y. The optimization of the rule set is beyond the scope of this paper. Let $r(x, y) = |R(x, y)|$. Similarly, the number of rules from y to x is denoted as $r(y, x)$. The total number of rules between the two domains is $r(x, y) + r(y, x)$. Once the routing path between x and y is determined, these rules will be enforced

TABLE 1
Notations

N	the set of domains
n	the number of domains, i.e., $n = \|N\|$
M	the set of firewalls
m	the number of firewalls, i.e., $m = \|M\|$
e	the total number of network interfaces of all firewalls
$r(x, y)$	the number of access control rules for flows from domain x to domain y
$w(f)$	the number of access control rules to be enforced on a firewall f

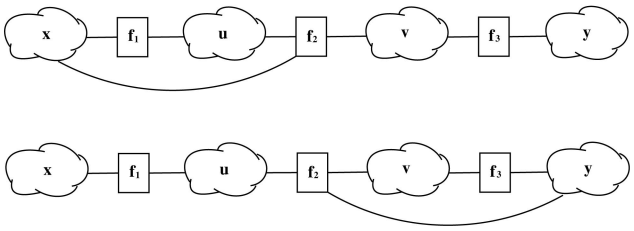

Fig. 1. Two topologies that connect domains, x, u, v, and y via firewalls f_1, f_2, and f_3 whose numbers of interfaces are 2, 3, and 2, respectively.

on the firewalls along the path. Each firewall may sit in the routing paths between many pairs of domains, and its rule set will be the aggregate of all rules between those domains. We want to construct the network topology and/or lay out the routing paths to avoid creating large firewall rule sets in the network. We assume that wild-card rules are processed separately. For example, if a domain requires to deny all external packets from reaching an internal subnet, a wild-card deny rule for that subnet will be installed at all firewalls adjacent to the domain. Since wild-card rules typically account for a small portion of a large rule set, for simplicity, when we compute the size of a firewall rule set, we ignore the contribution of wild-card rules in this paper.

For any firewall $f \in M$, let $W(f)$ be the set of access control rules to be enforced by f. Let $w(f) = |W(f)|$. If f sits in the routing path from domain x to domain y, then it enforces all rules between them, and thus, $R(x, y) \subseteq W(f)$; otherwise, it does not enforce those rules, and thus, $R(x, y) \bigcap W(f) = \emptyset$. Let $\Pi(f)$ be the set of domain pairs $\langle x, y\rangle$ with the routing path from x to y passing through f. We have

$$W(f) = \bigcup_{\langle x,y\rangle \in \Pi(f)} R(x, y),$$
$$w(f) = \sum_{\langle x,y\rangle \in \Pi(f)} r(x, y). \quad (1)$$

Some frequently used notations are listed in Table 1 for quick reference.

2.3 Problems

As the example in Fig. 1 shows, there are many ways to connect a set of domains via a set of firewalls. For any network topology, there are different ways to lay out the routing paths. In general, the rule sets to be enforced on the firewalls will be different when we change the network topology or the routing paths.

Definition 1. *The FPP is to 1) optimally connect a set of domains via a set of firewalls to form a network topology and 2) establish optimal interdomain routing tables on this topology such that the maximum number of access control rules on any firewall, i.e.,* $\max_{f\in M}\{w(f)\}$*, is minimized.*

Definition 2. *The* partial FPP *is the same as FPP except that it works on a given partially constructed topology that allows limited freedom in the way that the topology can be expanded.*

In practice, the network topology is often fixed and cannot be changed. By optimizing routing paths, we can still reduce the maximum rule set.

Definition 3. *The FRP is to construct optimal interdomain routing tables on an existing topology to minimize the maximum number of access control rules on any firewall.*

Definition 4. *The* partial FRP *problem is to complete the partially populated routing tables to minimize the size of the maximum firewall rule set.*

Moreover, we can introduce weights into the problem definition. Suppose each rule is assigned a weight, proportional to the expected traffic volume covered by this rule. Note that the legitimate traffic permitted by a rule is supposed to be routed through the firewalls where the rule is enforced (Section 2.2). Suppose each firewall is also assigned a weight, proportional to the firewall's capacity (such as processing speed). We define the *weight of a firewall rule set* to be the total weight of all rules in the set divided by the firewall's weight. It essentially measures the firewall's *normalized load*, defined as the total expected traffic volume (under normal conditions) divided by the firewall's capacity. The weighted version of the above problems is to minimize the maximum weight of any firewall rule set in the network. In other words, it is to minimize the maximum normalized load on any firewall. In the solutions for the weighted FPP/FRP problems, a firewall with a larger capacity is likely to take more rules or those rules with heavier traffic.

We will prove that all the above problems are NP-complete, and we will design a heuristic algorithm for them. *Instead of enumerating over all problems, our presentation will focus on FPP for analysis and algorithm design. We will show that the results can be trivially extended to other problems.* Focusing on FPP is only a presentation choice because it is easier to extend the solution for FPP to other problems. This presentation choice does not mean that our solution is only designed for topology construction in the network design phase. The solution can also be used for topology expansion and routing optimization in the operation phase, which is probably the more common scenario of application.

2.4 Rule Graph and Topology Graph

We use Fig. 2 to illustrate a few concepts. There are eight domains with ids from 1 to 8. The rule matrix ($r(x, y)$, x, $y \in N$) is shown in Fig. 2a. We construct a *rule graph* (denoted as G_r) in Fig. 2b, where each node is a domain and there is an undirected *edge* $\langle x, y\rangle$ if $r(x, y) + r(y, x) > 0$. The number of access control rules to be enforced between the two domains, i.e., $r(x, y) + r(y, x)$, is shown beside the link. G_r is a graphical representation of the rule matrix, specifying the security requirement. It will be the input to the algorithm that solves FPP and other problems (approximately, because they are NP-complete).

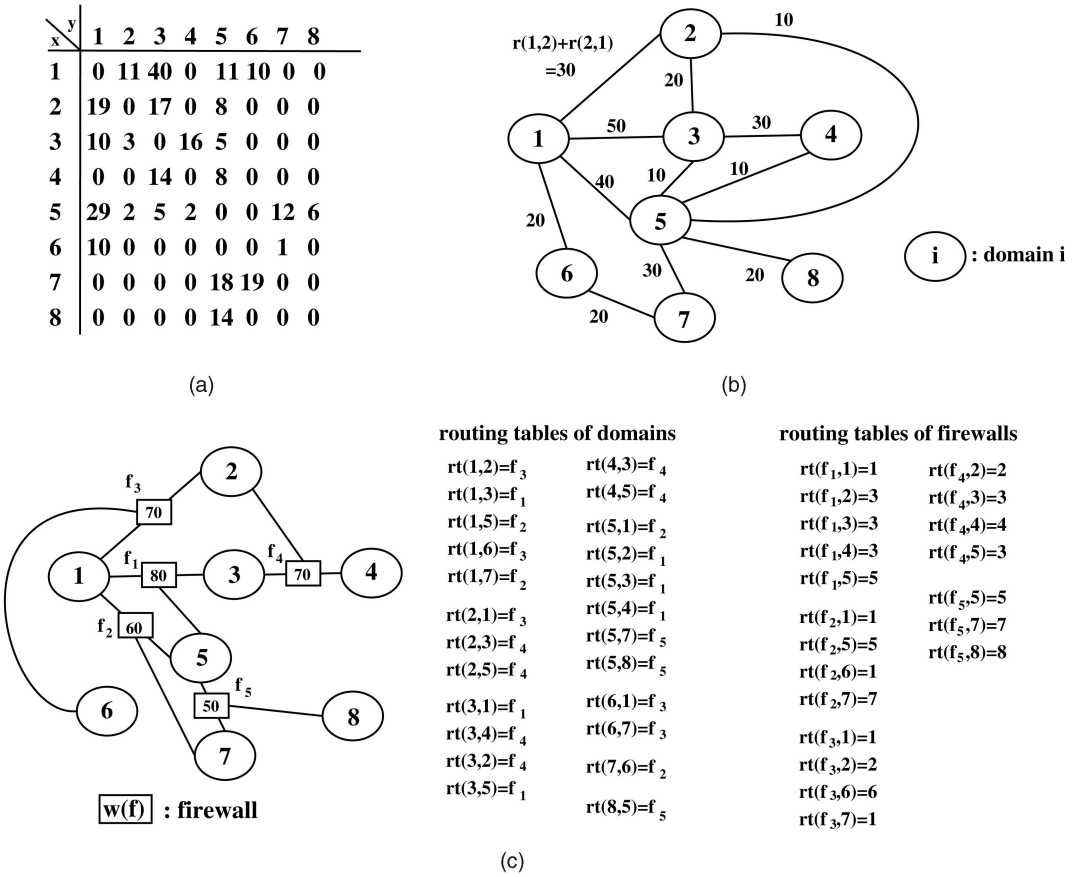

Fig. 2. Rule matrix, rule graph, and topology graph. (a) r(x,y). (b) Rule graph G_r. (c) Topology graph G_t.

For the output of the algorithm, we define a *topology graph* (denoted as G_t), which consists of a network topology and a routing structure. A node in G_t is either a domain or a firewall. An undirected *link* (x, f) represents a physical connection between a domain x and a firewall f. Note that we use the term "link (x, f)" in G_t, in contrast to the term "edge $\langle x, y \rangle$" in G_r. Each node has a routing table consisting of routing entries, each specifying the next hop for a destination domain.

Suppose there are five firewalls, each having three network interfaces. Fig. 2c shows the topology graph returned by the algorithm to be proposed in this paper. The number of access control rules enforced on a firewall is shown inside the box that represents the firewall. The routing tables are interpreted as follows: "$rt(1, 2) = f_3$" means that the routing table at domain 1 has an entry for destination domain 2 with the next hop being firewall f_3. In reality, the gateway in domain 1 which connects to firewall f_3 must advertise within the domain that it can reach domain 2. Consequently, the routing tables at the internal routers will each have an entry for domain 2, pointing toward that gateway. "$rt(f_1, 1) = 1$" means that the routing table at firewall f_1 has an entry for domain 1 with the next hop being domain 1. It implies that f_1 is directly connected to a gateway in domain 1. Of course, the actual routing entry uses that gateway as the next hop. "$rt(f_1, 2) = 3$" means that the routing table at firewall f_1 has an entry for domain 2 with the next hop being domain 3. It implies that f_1 is directly connected to a gateway in domain 3. The actual routing entry uses that gateway as the next hop and the address prefix of domain 2 as the destination. The other routing entries in the figure should be interpreted similarly.

Given a rule graph G_r and a set M of firewalls, for each feasible topology graph G_t, we can calculate $w(f)$, $\forall f \in M$. The topology graph that minimizes $\max_{f \in M}\{w(f)\}$ is the solution. FPP has the largest set of feasible topology graphs; partial FPP has a smaller set due to the restriction of a given partial topology. FRP has only one feasible topology with many possible routing structures, while partial FRP gives less freedom in constructing a routing structure.

2.5 Robustness

Robustness against node failure is an important issue in network design. While dynamic routing is used inside each domain, we must guard against firewall failure. The most common way to achieve high availability is to use dual firewalls. The state-synchronization solution and the load-balancing solution [6], [7], [8] are prevalent in practice. Fig. 3 shows one example for each approach. In both cases, firewalls in parallel disposition have the same rule set so that one of them can continue the service when the other fails. Identical colocated firewalls can be logically treated as one in our solution. Therefore, we will not explicitly discuss the use of dual firewalls in the sequel.

3 NP-Completeness

In this section, we prove that FPP is NP-complete. The same process can be used to prove the NP-completeness of

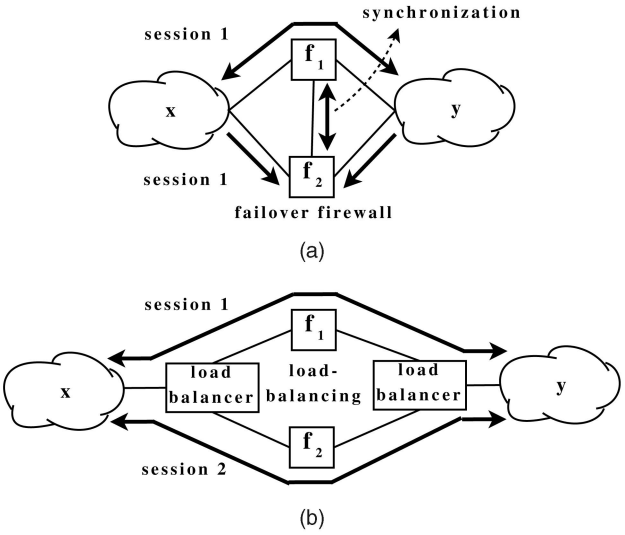

Fig. 3. High-availability solutions. (a) State-synchronization solution. (b) Load balancer solution.

partial FPP, FRP, partial FRP, and weighted FPP/FRP, which is omitted to avoid excessive repetition.

FPP is an optimization problem. We define the corresponding decision problem as follows: Given a rule graph and a set of firewalls, the *k-firewall decision problem* is to decide whether there exists a topology graph such that $w(f) \leq k$, $\forall f \in M$, where k is an arbitrary, positive integer. To prove the NP-completeness of FPP, it is sufficient to prove that its decision problem is NP-complete.

The proof consists of two steps. First, we show that the k-firewall decision problem $\in$ NP. Second, we show that it is NP-hard by reducing the set-partition problem (known to be NP-complete [9]) to the k-firewall decision problem in polynomial time.

3.1 k-Firewall Decision Problem $\in$ NP

To show the decision problem belongs to NP, we need to give a verification algorithm that can verify a solution G_t of the problem in polynomial time. G_t is a topology graph, specifying the network topology and the routing paths between domains. The verification algorithm is described as follows: Initially, $w(f) = 0$, $\forall f \in M$. For each edge $\langle x, y\rangle$ in G_r, we traverse the routing path between domain x and domain y in G_t. For each firewall f on the path, $w(f) := w(f) + r(x, y) + r(y, x)$. There are $O(n^2)$ edges in G_r and the length of a routing path is $O(n + m)$. Therefore, it takes $O(n^2(n + m))$ time to calculate $w(f)$, $\forall f \in M$. After that, it takes $O(m)$ time to verify $w(f) \leq k, f \in M$.

3.2 NP-Hardness

We show that the set-partition problem can be reduced to the k-firewall decision problem in polynomial time. In that case, because the set-partition problem is NP-hard [9], the k-firewall decision problem is also NP-hard. Our proof is to transform the set-partition problem into a specially constructed instance of the k-firewall decision problem in polynomial time such that a solution to the latter will be a solution to the former and all other NP problems.

Given a finite set A of positive integers, the set-partition problem is to determine whether there exists a subset $A' \subseteq A$ such that $\sum_{a\in A'} a = \sum_{a\in A-A'} a$. We reduce it to the k-firewall decision problem as follows:

First, for each member $a \in A$, we associate it with a pair of two domains $\langle x_a, y_a\rangle$, and let the number of access control rules from x_a to y_a be a. In total, there are $2|A|$ domains. $N = \{x_a, y_a \mid a \in A\}$. For domain pairs $\langle x_a, y_a\rangle$, $\forall a \in A$, $r(x_a, y_a) = a$, and for all other domain pairs $\langle x, y\rangle$, $r(x, y) = 0$.

Second, we use two firewalls, denoted as f_1 and f_2. The number of network interfaces of each firewall is $2 \times |A|$. k is set to be

$$\frac{\sum_{a\in A} a}{2}.$$

The reduction from the set-partition problem to the above k-firewall decision problem can be done in polynomial time since we only need to convert $|A|$ integers into $|A|$ domain pairs with the rule matrix $(r(x, y), x, y \in N)$ appropriately set.

Next, we prove that the set-partition problem is satisfiable if and only if the corresponding k-firewall decision problem is satisfiable.

First, suppose the set-partition problem is satisfiable, i.e., there exists a subset $A' \subseteq A$ such that

$$\sum_{a\in A'} a = \sum_{a\in A-A'} a = \frac{\sum_{a\in A} a}{2}.$$

We construct a topology graph as follows: For each member a in A', we connect both x_a and y_a to f_1, insert a routing path $x_a \rightarrow f_1 \rightarrow y_a$, and add $r(x_a, y_a)$, which equals a, to $w(f_1)$. For each member $a \in A - A'$, we connect both x_a and y_a to f_2, insert a routing path $x_a \rightarrow f_2 \rightarrow y_a$, and add $r(x_a, y_a)$, which equals a, to $w(f_2)$. Finally, we use the remaining free interfaces to make the graph connected. The constructed topology graph has the following property:

$$\begin{aligned} w(f_1) &= \sum_{a\in A'} r(x_a, y_a) = \sum_{a\in A'} a = \frac{\sum_{a\in A} a}{2} = k, \\ w(f_2) &= \sum_{a\in A-A'} r(x_a, y_a) = \sum_{a\in A-A'} a = \frac{\sum_{a\in A} a}{2} = k. \end{aligned} \tag{2}$$

Therefore, the k-firewall decision problem is also satisfiable.

Second, suppose the k-firewall decision problem is satisfiable, i.e., there exists a topology graph such that $w(f_1) \leq k$ and $w(f_2) \leq k$. Recall that

$$k = \frac{\sum_{a\in A} a}{2}.$$

We have

$$w(f_1) + w(f_2) \leq 2k = \sum_{a\in A} a. \tag{3}$$

Each rule has to be enforced by f_1, f_2, or both. Therefore,

$$w(f_1) + w(f_2) \geq \sum_{a\in A} r(x_a, y_a) = \sum_{a\in A} a. \tag{4}$$

By (3) and (4), we have

$$w(f_1) + w(f_2) = \sum_{a\in A} a. \tag{5}$$

HAF(G_r, G_t, M)
1. **for** each $\langle x, y\rangle$ of G_r in descending order of $(r(x,y)+r(y,x))$ **do**
2. Insert_Optimal_Path(G_t, x, y)
3. **return** G_t

Fig. 4. The pseudocode of HAF.

Consider an arbitrary member $a \in A$. The routing path from x_a to y_a must only pass either f_1 or f_2 but not both because, otherwise, the above equation could not hold. Let $\Pi(f)$ be the set of domain pairs $\langle x_a, y_a\rangle$ whose routing path passes a firewall f. $\Pi(f_1) \bigcap \Pi(f_2) = \emptyset$. Because $w(f_1) \le k$ and $w(f_2) \le k$, by (5), we have

$$w(f_1) = w(f_2) = \frac{\sum_{a\in A} a}{2}.$$

Consider $w(f_1)$ and we have the following equation:

$$\sum_{\langle x_a, y_a\rangle \in \Pi(f_1)} r(x_a, y_a) = \frac{\sum_{a\in A} a}{2}. \quad (6)$$

Let $A' = \{a \mid \langle x_a, y_a\rangle \in \Pi(f_1)\}$. The above equation can be rewritten as follows:

$$\sum_{a\in A'} r(x_a, y_a) = \sum_{a\in A'} a = \frac{\sum_{a\in A} a}{2}. \quad (7)$$

Therefore, the set-partition problem is also satisfiable.

4 HAF: A Heuristic Algorithm for FPP, Partial FPP, FRP, Partial FRP, and Weighted FPP/FRP

We propose HAF—a Heuristic Algorithm to approximately solve the Firewall problems defined in Section 2.3. Our description of the algorithm centers around FPP. In Section 4.7, we show that the algorithm can be used to solve other problems.

4.1 Overview

The input of HAF is a rule graph G_r, a set M of firewalls, and an initial topology graph G_t, which has no link for FPP, but is a partial or full topology for other problems. The output of HAF is a completed topology graph G_t, which consists of domains and firewalls as nodes, links connecting domains and firewalls, and routing tables.

We have shown that the global optimization problem of FPP, which is to find the optimal topology and routing paths that minimize the maximum firewall rule set in the network, is NP-complete. However, constructing an optimal routing path between one pair of domains is a polynomial problem. The basic idea behind HAF is to process the domain pairs one at time and iteratively insert the optimal routing path for each domain pair into a topology graph G_t. After the paths for all domain pairs are inserted, G_t is an approximate solution to the FPP problem. HAF is particularly useful when the physical network is gradually expanding. After the algorithm produces a topology G_t for the current domains, when a new domain is added to the network, the algorithm can be naturally invoked to process the new domain pairs on top of the existing topology.

The pseudocode of the HAF algorithm is given in Fig. 4. For FPP, G_t is initially a topology graph of n domain nodes and m firewall nodes with no link. For each edge $\langle x, y\rangle$ in G_r, the subroutine Insert_Optimal_Path(G_t, x, y) is called to perform the following three tasks:

1. Define the set of feasible routing paths between domain x and domain y.
2. Find the optimal routing path between x and y that minimizes the maximum rule set among all feasible routing paths.
3. Insert the optimal routing path to G_t.

The loop of Lines 2-3 processes the set of edges $\langle x, y\rangle$ in G_r in the descending order of $(r(x,y)+r(y,x))$, which is the total number of rules between domain x and domain y. The topology graph G_t keeps growing as the loop inserts one routing path to G_t in each iteration. In the following, we show how to implement the above three tasks of Insert_Optimal_Path.

4.2 Augmented Graph $G_t^{\langle x,y\rangle}$ and MinMax Path

To define the set of feasible paths between domain x and domain y, we first construct an augmented graph $G_t^{\langle x,y\rangle}$ from G_t as follows: The links already in G_t are called *physical links*. For each firewall f with one or more free interfaces, we add a new link between f and x if they are not already connected. Similarly, we add a new link between f and y if they are not already connected. These new links are called *virtual links*. A virtual link may be turned into a physical one if needed. G_t and the virtual links together form the augmented graph $G_t^{\langle x,y\rangle}$. A routing path between x and y in the augmented graph is *feasible* if the following three conditions are satisfied:

- *Routing condition: The routing path must be consistent with the routing tables at nodes on the path.* For an arbitrary link (v, u) on the path, if v already has a routing entry for destination y but the next hop is not u, then the path is not feasible.
- *Interface condition: When all virtual links on the path are turned into physical ones, no firewall uses more network interfaces than it has.* Suppose a firewall f has only one free interface and both (f, x) and (f, y) are virtual links in $G_t^{\langle x,y\rangle}$. A path (x, f, y) is not feasible because we cannot turn both (x, f) and (f, y) into physical links.
- *Connectivity condition: After the path is turned physical, G_t should still have enough free interfaces to turn itself into a connected graph.* Initially, G_t is not a connected graph. In the end, it has to be a connected graph. During the execution of HAF, there should always be enough free interfaces to make new links that are able to connect all separated topological components

in G_t. Therefore, if a routing path uses too many free interfaces that makes G_t no longer connectable, then the path is not feasible.

In other words, a path is feasible if we can turn it into a physical path without violating the current routing structure in G_t, exceeding the interface limitation of any firewall, or rendering G_t not connectable.

Definition 5. *The* MinMax path *in* $G_t^{\langle x,y\rangle}$ *is the optimal feasible path between* x *and* y *that minimizes the maximum rule set on the path.*

Based on the construction of $G_t^{\langle x,y\rangle}$, all virtual links either connect to x or connect to y. Therefore, only the first and last links on the MinMax path may be virtual links.

4.3 Find the MinMax Path in $G_t^{\langle x,y\rangle}$

We transform the problem of finding the MinMax path to a variant of the shortest path problem. We define a *cost* metric on nodes. The cost of a firewall is the size of its rule set, i.e., $w(f)$ as defined in (1). The cost of a domain is zero. The cost of a path is the maximum cost (instead of the sum of the costs) of all nodes on the path. One path is *shorter* than another path if the cost of the former is smaller or the costs of the two paths are the same but the former has a fewer number of hops. By this definition, the shortest path between x and y must also be the MinMax path.

We design an algorithm, called *HAF_Dijkstra*, to find the shortest path between x and y in $G_t^{\langle x,y\rangle}$. It is an all-source single-destination variant of Dijkstra's algorithm, designed for a graph with:

1. *virtual links* (subject to the interface condition stated in the previous section);
2. *routing restrictions*;
3. *node costs* instead of link costs;
4. path length defined as the *maximum node cost* instead of the sum of the node costs on the path.

Satisfying that the connectivity condition is a rather complex task, which will be ignored for now and addressed in the next section, where we will modify the construction of $G_t^{\langle x,y\rangle}$ to include only those virtual links that do not make G_t unconnectable.

Before giving the pseudocode of the algorithm, we define the following variables. $rt[v,d]$ is the routing table entry at node v for destination d. Its value is inherited from G_t. If G_t does not have such a routing entry, the value of $rt[v,d]$ is NIL. $c[v]$ is the cost of node v. $cost[v,d]$ is the *estimated* cost of the shortest path from v to d. $hops[v,d]$ is the *estimated* number of hops on the shortest path from v to d. These two variables are initialized to ∞, and then, improved by the algorithm until reaching the optimal values. $next[v,d]$ stores the next hop after v on the shortest path to d. Q is the set of nodes whose shortest paths to d have been found. Extract_Min(Q) and Relax(v, u) are two standard subroutines in Dijkstra's algorithm. Extract_Min(Q) finds the node u in Q that has the smallest $cost[u,d]$ value and, when there is a tie, has the smallest $hops[u,d]$ value. After the shortest path from u to d is found, Relax(v, u) propagates this information to all adjacent nodes v. The pseudocode of the HAF_Dijkstra algorithm is given in Fig. 5. ":= " is the assignment sign, s is the source node, and d is the destination node.

Routing_Condition(v, u, d) and Interface_Condition(v, u, s, d) make sure that the Relax subroutine is performed on link (v, u) only when both the routing condition and the interface condition are satisfied. By the construction of Shortest_Path($G_t^{\langle x,y\rangle}, s, d$), the Routing_Condition and Interface_Condition subroutines are executed iteratively for all links of the shortest path, and therefore, the returned shortest path must be feasible.

HAF_Dijkstra first uses x as the source node and y as the destination node to find the shortest path by calling Shortest_Path($G_t^{\langle x,y\rangle}, x, y$). Then it uses y as the source node and x as the destination node to find the shortest path by calling Shortest_Path($G_t^{\langle x,y\rangle}, y, x$). Finally, it returns the shorter one between these two paths. The reason for calling the Shortest_Path subroutine twice is due to the asymmetry caused by virtual links, which is illustrated in Fig. 6, assuming that each node only has routing entries for directly connected nodes. The clouds, blocks, solid lines, dashed lines, and bold lines represent domains, firewalls, physical links, virtual links, and the shortest paths, respectively. If f_2 has two free interfaces, the shortest path is shown in Fig. 7. Another more complicated example is shown in Fig. 8, where f_2 and f_3 each have one free interface.

4.4 Insert the MinMax Path to G_t

After finding the MinMax path for $\langle x,y\rangle$, we insert the path to G_t. The following operations are performed:

- Convert each virtual link on the MinMax path to a physical link.
- For each firewall f on the MinMax path, increase the size of its rule set by $r(x,y)+r(y,x)$.
- Let the path be $(v_1, v_2, \ldots, v_l)$, where $v_1 = x$ and $v_l = y$. For $1 \le i < l$, add a routing entry at v_i for each destination, $v_{i+1}, \ldots, v_l$, with the next hop being v_{i+1}. For $1 < i \le l$, add a routing entry at v_i for each destination, $v_1, \ldots, v_{i-1}$, with the next hop being v_{i-1}. This will keep the routing symmetry during the execution of the HAF algorithm.

4.5 Ensuring Connectivity

G_t may not be a connected graph. A *component* of G_t is a connected subgraph that is not contained by a larger connected subgraph. Let c be the number of components in G^t. Let ϕ be the total number of free interfaces of all firewalls. Note that a physical link can be inserted into the graph for each free network interface.

Property 2. *G_t can be turned into a connected graph if and only if* $\phi \ge c-1$.

Proof. We first prove that $\phi \ge c-1$ is a necessary condition for G_t to be turned into a connected graph. G_t has c components. For G_t to be turned into a connected graph, we must reduce c to 1 by adding at least $c-1$ new links, which means that there must be at least $c-1$ free interfaces.

Next, we prove $\phi \ge c-1$ is a sufficient condition. First, we consider a simple case where all domains belong to one component. The remaining components must be single firewalls, each having at least two free interfaces. To form a connected graph, we can simply connect these firewalls to any domains.

```
Routing_Condition(v, u, d)
1.  if rt[v,d] = NIL or rt[v,d] = u then
2.      return true
3.  else
4.      return false

Interface_Condition(v, u, s, d)
1.  if v = s ∧ next[u,d] = d and both (s,u) and (u,d) are virtual
      links but u has only one free interface then
2.      return false
3.  else
4.      return true

Relax(v, u, d)
1.  if max{c[v], cost[u,d]} < cost[v,d] or max{c[v], cost[u,d]}
      = cost[v,d] ∧ hops[u,d] + 1 < hops[v,d] then
2.      cost[v,d] := max{c[v], cost[u,d]}
3.      hops[v,d] := hops[u,d] + 1
4.      next[v,d] := u

Shortest_Path(G_t^⟨x,y⟩, s, d)
1.  for each node v ∈ N ⋃ M do
2.      cost[v,d] := ∞,  hops[v,d] := ∞,  next[v,d] := NIL
3.  cost[d,d] := 0,  hops[d,d] = 0
4.  Q := N ⋃ M
5.  while Q ≠ ∅ do
6.      u := Extract_Min(Q)
7.      if u = s then
8.          break out of the while loop
9.      Q := Q − {u}
10.     for every adjacent node v of u in G_t^⟨x,y⟩ do
11.         if Routing_Condition(v, u, d) and Interface_Condition(v, u, s, d) then
12.             Relax(v, u,d)
13. return the shortest path from s to d stored in the next variable

HAF_Dijkstra(G_t^⟨x,y⟩, x, y)
1.  p1 := Shortest_Path(G_t^⟨x,y⟩, x, y)
2.  p2 := Shortest_Path(G_t^⟨x,y⟩, y, x)
3.  return the better one between p1 and p2
```

Fig. 5. The pseudocode of HAF_Dijkstra.

Second, consider the case where the domains belong to at least two components. $\phi \geq c - 1 \geq 1$. There must be a firewall with a free interface. The firewall belongs to a component. There must be another component that has a domain. Connect the firewall and the domain, which uses one free interface and reduces the number of components by one. Therefore, the condition $\phi \geq c - 1$ remains true. Repeat the above process until all domains belong to one component. For this case, we have already proved that the graph can be made connected.

Therefore, $\phi \geq c - 1$ is a necessary and sufficient condition for G_t to be turned into a connected graph. □

Only the first and last links on the MinMax path may be virtual links. By inserting the MinMax path to G_t, we consume at most two free network interfaces. However, if the number of free interfaces is limited, the MinMax may be restricted to consume less than two free interfaces in order to leave enough free interfaces to ensure the connectivity of the graph. Assume that the condition $\phi \geq c - 1$ holds in G_t before the insertion of the MinMax path. We want to keep the condition true after the insertion. There are three cases.

- *Case 1:* $\phi \geq c + 1$ *before insertion.* The MinMax path is allowed to consume two free interfaces.
- *Case 2:* $\phi = c$ *before insertion.* The MinMax path is allowed to consume one free interface or two free interfaces if the path connects two components into one.
- *Case 3:* $\phi = c - 1$ *before insertion.* The MinMax path is allowed to consume one free interface if the path connects two components into one, or consume two free interfaces if the path connects three components into one.

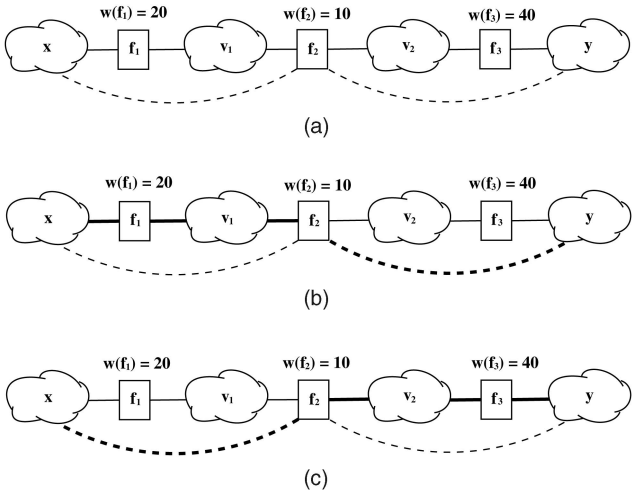

Fig. 6. (a) The augmented graph $G_t^{\langle x,y\rangle}$, where f_2 has one free interface and two virtual links. (b) The shortest path returned by Shortest_Path($G_t^{\langle x,y\rangle}, x, y$), where the relaxation is performed from y along the path to x. (c) The shortest path returned by Shortest_Path($G_t^{\langle x,y\rangle}, y, x$), where the relaxation is performed from x along the path to y. The best path is (x, f_1, v_1, f_2, y).

In order to enforce the above restrictions, we have to carefully redesign the subroutine of Insert_Optimal_Path, which is Line 3 of the HAF algorithm. Let $Com(x)$ be the component in G_t that contains x. The pseudocode of Insert_Optimal_Path is given in Fig. 9. We give a brief explanation below.

Lines 1-5 implement Case 1. There are plenty of free interfaces. For each firewall f with a free interface, the algorithm adds a virtual link between f and x (or y) if they are not already connected. It then runs the HAF_Dijkstra algorithm on the augmented graph to find the shortest path.

Lines 6-21 implement Case 2. Lines 7-12 add virtual links that connect different components. More specifically, for each firewall f with a free interface, if f and x (or y) belong to different components in G_t, add a virtual link between f and x (or y). When any one of these links is turned into a physical one, it consumes one free interface and also reduces the number of components by one.

Because $\phi = c$ in G_t, we are allowed to consume one free interface without reducing the number of components. In other words, the MinMax path is allowed to use a virtual link within the component that contains x, or a virtual link within the component that contains y, but not both. Lines 13-16 find the shortest path that may use a virtual link within the component of xs. Lines 17-20 find the shortest path that may use a virtual link within the component of ys. Line 21 returns the better of the two paths.

Lines 22-29 implement Case 3. Because $\phi = c - 1$ in G_t, we can use a virtual link only when it reduces the number of components by one. It means that the augmented graph can only have virtual links that connect different components.

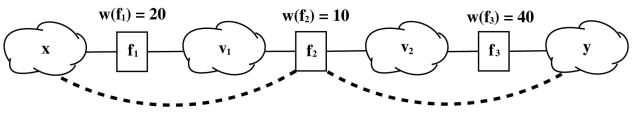

Fig. 7. The shortest path when f_2 has two free interfaces.

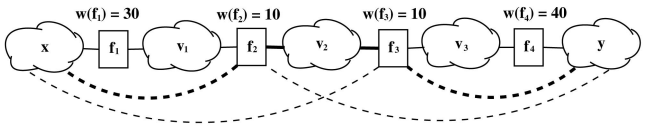

Fig. 8. The shortest path when f_2 and f_3 each have one free interface.

4.6 Complexity Analysis

The time complexity of the Shortest_Path subroutine is the same as the complexity of Dijkstra's algorithm, which is $O(e + (n+m)\log(n+m))$. The complexities of the HAF_Dijkstra and Insert_Optimal_Path subroutines are the same as that of Shortest_Path. HAF executes the Insert_Optimal_Path subroutine for at most $O(n^2)$ times. Therefore, the total time complexity is $O(n^2 e + n^2(n+m)\log(n+m))$.

4.7 Modifying HAF for FRP, Partial FRP, and Weighted FPP/FRP

To solve partial FPP, we simply initialize G_t as the existing partial network topology. To solve FRP, we initialize G_t as the existing network topology and set the number of free interfaces to be zero for all firewalls. For partial FRP, we further initialize the route entries $rt[v, d]$ whose values are known. The rest of HAF remains the same. To solve weighted FPP/FRP, we only need to change the definition of $r(x, y)$ and $w(f)$, while leaving the algorithm intact. Instead of $r(x, y) = |R(x, y)|$ as defined in Section 2.2, $r(x, y)$ should now be the sum of the weights of all rules in $R(x, y)$. Instead of $w(f) = \sum_{\langle x,y\rangle \in \Pi(f)} r(x, y)$ as in (1), $w(f)$ should now be interpreted as the weight of the rule set at f and defined as

$$w(f) = \frac{\sum_{\langle x,y\rangle \in \Pi(f)} r(x, y)}{\text{the weight of firewall } f}.$$

5 Simulation

In this section, we evaluate the performance of the HAF algorithm for FPP. The results for FRP and weighted FPP/FRP are omitted due to space limitation. To the best of our knowledge, this is the first paper that studies the FPP problem. We do not have existing algorithms to compare with. In our simulations, we implement two simple algorithms, called the *tree topology* algorithm (TTA for brevity) and the *full topology* algorithm (FTA for brevity), respectively.

For a given FPP problem, the TTA first constructs a tree topology, which defines unique routing paths between any two domains. To construct a tree, the algorithm begins with one domain as the root. A number of firewalls are selected to be the children of the root at the second level of the tree. We select firewalls in the descending order of their numbers of interfaces. For each second-level firewall, a number of domains are selected to be the children at the third level. We repeat this until the tree includes all domains or firewalls. The even levels of the tree are firewalls, while the odd levels are domains. The number of children of a firewall is limited by its number of network interfaces. We also restrict the average number of children per domain to be the same as the average number of children per firewall.

```
Insert_Optimal_Path(G_t, x, y)
1.  if φ ≥ c + 1 in G_t then
2.      initialize G_t^⟨x,y⟩ to be G_t
3.      for each firewall f with a free interface do
4.          add a virtual link in G_t^⟨x,y⟩ between f and x (or y)
            if they are not already connected
5.      p := HAF_Dijkstra(G_t^⟨x,y⟩, x, y)
6.  else if φ = c in G_t then
7.      initialize G'_t to be G_t
8.      if Com(x) ≠ Com(y) then
9.          for each firewall f with a free interface, Com(f) ≠ Com(x) do
10.             add a virtual link in G'_t between f and x
11.         for each firewall f with a free interface, Com(f) ≠ Com(y) do
12.             add a virtual link in G'_t between f and y
13.     initialize G_t^⟨x,y⟩ to be G'_t
14.     for each firewall f with a free interface, Com(f) = Com(x) do
15.         add a virtual link in G_t^⟨x,y⟩ between f and x
16.     p_1 := HAF_Dijkstra(G_t^⟨x,y⟩, x, y)
17.     initialize G_t^⟨x,y⟩ to be G'_t
18.     for each firewall f with a free interface, Com(f) = Com(y) do
19.         add a virtual link in G_t^⟨x,y⟩ between f and y
20.     p_2 := HAF_Dijkstra(G_t^⟨x,y⟩, x, y)
21.     p := the better one between p_1 and p_2
22. else if φ = c − 1 in G_t then
23.     initialize G_t^⟨x,y⟩ to be G_t
24.     if Com(x) ≠ Com(y) then
25.         for each firewall f with a free interface, Com(f) ≠ Com(x) do
26.             add a virtual link in G_t^⟨x,y⟩ between f and x
27.         for each firewall f with a free interface, Com(f) ≠ Com(y) do
28.             add a virtual link in G_t^⟨x,y⟩ between f and y
29.     p := HAF_Dijkstra(G_t^⟨x,y⟩, x, y)
30. Insert p to G_t
```

Fig. 9. The pseudocode of Insert_Optimal_Path.

The FTA first constructs a tree topology in the same way as the TTA does. It then *fully* utilizes all remaining free interfaces on the firewalls by making a link from each free interface to an arbitrary domain selected uniformly at random.[1] After that we run a shortest path algorithm to find the least-hops routing path between each pair of domains.

The default simulation parameters are shown in Table 2. The simulations will change the default values of the parameters one at a time. Here, n is the number of domains and m is the number of firewalls. Let $e(f)$ be the number of network interfaces on firewall f and $\overline{e(f)}$ is the average number of network interfaces per firewall. The value of $e(f)$, $\forall f \in M$, is generated from $[2..2\overline{e(f)} - 2]$ uniformly at random. $\overline{r(x,y)}$ is the average value of $r(x,y)$ among domain pairs $\langle x,y \rangle$ with $r(x,y) > 0$. p is the probability of $r(x,y) + r(y,x) > 0$ for an arbitrary domain pair $\langle x,y \rangle$. When $r(x,y) > 0$, its actual value is generated from $[1..2\overline{r(x,y)} - 1]$ uniformly at random.

1. The tree topology with cross links is often seen in organizations with hierarchical administrative structures.

Figs. 10, 11, 12, 13, 14, and 15 show the simulation results. In all figures, the y-axis is the size of the maximum rule set ($max_{f \in M}\{w(f)\}$) at any firewall. We abbreviate "the size of the maximum firewall rule set" as "the MFRS size." The x-axis is one of the parameters. The figures compare the MFRS sizes achieved by the three algorithms under different parameter values.

In Fig. 10, we vary the number n of domains in the simulation. When n is very small, the numbers of firewalls and interfaces are relatively plentiful such that most domain pairs are one firewall away from each other and the rules are well spread on the firewalls. The MFRS size is small for all three algorithms. As n increases, HAF performs far better than others. When $n = 120$, the MFRS size

TABLE 2
Default Simulation Parameters

n	m	$\overline{e(f)}$	$\overline{r(x,y)}$	p
100	40	4	10	0.7

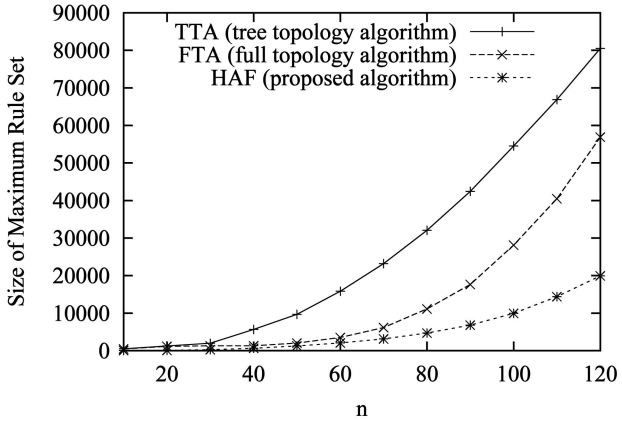

Fig. 10. Size of maximum rule set with respect to number n of domains. $10 \le n \le 120$, $m = 40$, $\overline{e(f)} = 4$, $\overline{r(i,j)} = 10$, $p = 0.7$.

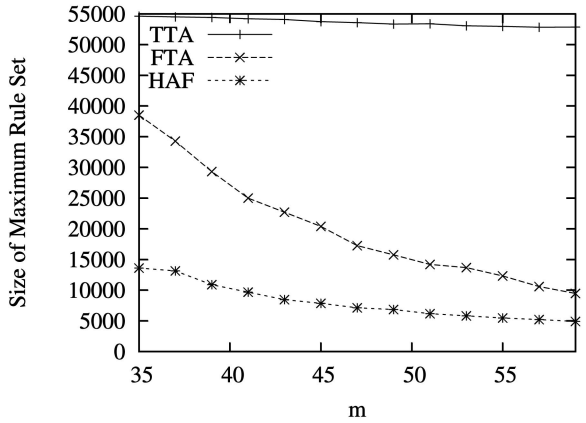

Fig. 11. Size of maximum rule set with respect to number m of firewalls. $n = 100$, $35 \le m \le 59$, $\overline{e(f)} = 4$, $\overline{r(i,j)} = 10$, $p = 0.7$.

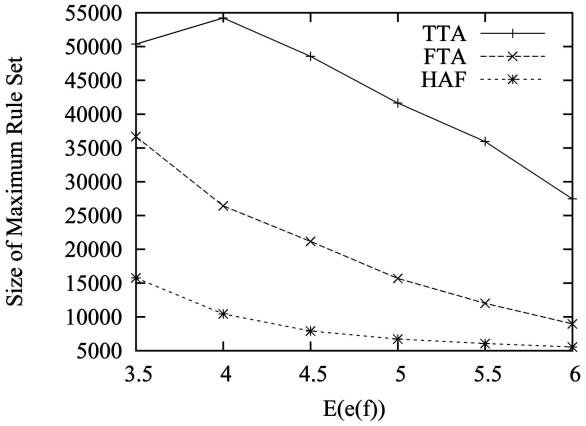

Fig. 12. Size of maximum rule set with respect to average number $\overline{e(f)}$ of network interfaces per firewall. $n = 100$, $m = 40$, $3.5 \le \overline{e(f)} \le 6$, $\overline{r(i,j)} = 10$, $p = 0.7$.

achieved by HAF is just 35.06 percent of that achieved by FTA, and 24.78 percent of that achieved by TTA.

In Fig. 11, we vary the number m of firewalls in the simulation. TTA is insensitive to the value of m because the tree topology cannot take full advantage of the increased number of firewalls. HAF performs much better than TTA and FTA. When $m = 35$, the MFRS size achieved by HAF is 35.31 percent of that achieved by FTA, and 24.90 percent of that achieved by TTA.

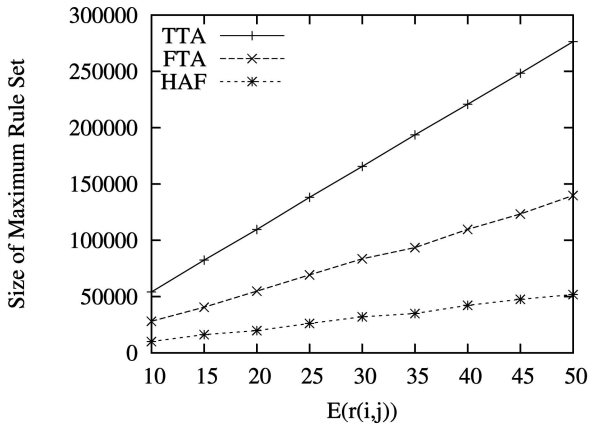

Fig. 13. Size of maximum rule set with respect to average number $\overline{r(i,j)}$ of rules per domain pair. $n = 100$, $m = 40$, $\overline{e(f)} = 4$, $10 \le \overline{r(i,j)} \le 50$, $p = 0.7$.

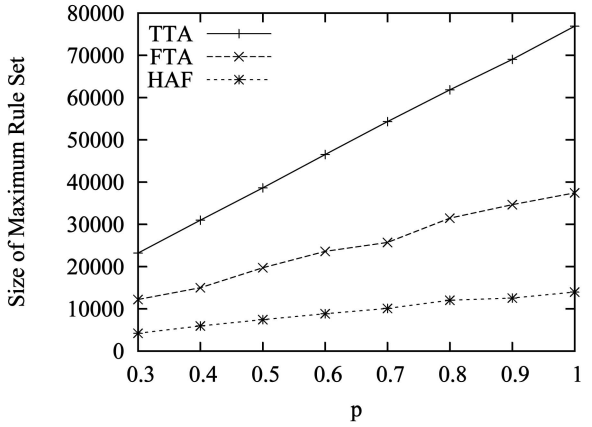

Fig. 14. Size of maximum rule set with respect to probability p. $n = 100$, $m = 40$, $\overline{e(f)} = 4$, $\overline{r(i,j)} = 10$, $0.3 \le p \le 1.0$.

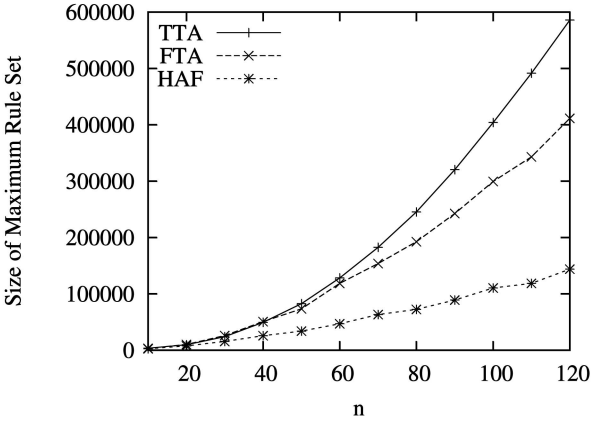

Fig. 15. Size of maximum rule set in sparse network. $10 \le n \le 120$, $m = (n-1)/(\overline{e(f)} - 1)$, $\overline{e(f)} = 4$, $\overline{r(i,j)} = 50$, $p = 1.0$.

In Fig. 12, we vary the average number $\overline{e(f)}$ of network interfaces per firewall. HAF performs the best among the three. When $\overline{e(f)} = 3.5$, the MFRS size achieved by HAF is 43.02 percent of that achieved by FTA, and 31.34 percent of that achieved by TTA.

In Fig. 13, we vary the average number $\overline{r(x,y)}$ of rules per domain pair. As $\overline{r(x,y)}$ increases, the MFRS size increases proportionally for all three algorithms. When $\overline{r(x,y)} = 100$, the MFRS size achieved by HAF is

37.05 percent of that achieved by FTA, and 18.74 percent of that achieved by TTA.

In Fig. 14, we vary the probability p for a domain pair $\langle x, y\rangle$ to have one or more rules. The value of p determines the density of the rule graph G_r. As p increases, the MFRS size increases for all three algorithms. HAF performs better than the other two algorithms for all p values used in the simulation. When $p = 1$, the MFRS size achieved by HAF is 37.37 percent of that achieved by FTA, and 18.19 percent of that achieved by TTA.

In Fig. 15, we study sparse network topologies with $m = (n-1)/(\overline{e(f)} - 1)$, which means that the number of firewalls is just enough to keep the topology connected. During the simulation, we discard the runs that have too few firewalls to form a connected topology. The figure shows that HAF works far better than others as n increases. When $n = 120$, the MFRS size achieved by HAF is 34.98 percent of that achieved by FTA, and 24.57 percent of that achieved by TTA.

6 Related Works

Gouda and Liu developed a sequence of five algorithms that can be applied to generate a compact rule set while maintaining the consistency and completeness of the original rule set [10]. They proposed a method for diverse firewall design and presented algorithms to detect discrepancies between two rule sets [11]. They also presented the first model of stateful firewalls [12]. Liu et al. studied the language and algorithm for firewall queries [13]. In [14], Liu investigated the theory and algorithms for firewall policy change impact analysis, and in [15], he developed a firewall verification tool. Recently, Liu et al. proposed a novel algorithm for minimizing security policies of a firewall [16].

Wool investigated the direction-based filtering in firewalls [17]. Fulp studied the problem of reducing the average number of rules that must be examined for each packet [18]. Al-Shaer and Hamed identified anomalies that exist in a single or multifirewall environment, and presented a set of techniques to discover configuration anomalies in centralized and distributed legacy firewalls [19]. Smith et al. studied the problem of how to place a set of firewalls in a complex network to minimize cost and delay [20] and the problem of how to increase comprehensiveness and level of confidence in protection [21]. El-Atawy et al. proposed to optimize packet filtering performance by traffic statistical matching [23]. Hamed et al. designed algorithms that maximize early rejection of unwanted packets and utilize traffic characteristics to minimize the average packet matching time [22].

Packet filtering can be viewed as a special case of packet classification [24], which is to determine the first matching rule for each incoming packet at a router. Much work has focused on solving the problem of how to find matching rules as quickly as possible by using sophisticated data structures or hardware-driven approaches [25], [26], [27], [28]. Other work proposed algorithms for removing redundancy in packet classifiers [29], [30], [31], [32].

7 Conclusion

This paper studies the firewall placement problem and its variations. The problem is to optimally place the firewalls in a network topology and find the routing structure such that the maximum size of the firewall rule sets in the network is minimized. We prove that the problem is NP-complete and propose a heuristic algorithm, called HAF, to solve the problem approximately. The algorithm can also be used to solve the firewall routing problem as well as weighted firewall placement/routing problems.

Acknowledgments

This work is supported in part by Cisco University Research Award (to Shigang Chen in 2007).

References

[1] A. Rubin, D. Geer, and M. Ranum, *Web Security Sourcebook.* Wiley Computer Publishing, 1997.
[2] S. Hinrichs and S. Chen, "Network Management Based on Policies," *Proc. SPIE Multimedia Computing and Networking Conf.,* Jan. 2000.
[3] J. Wack, K. Cutler, and J. Pole, *Guidelines on Firewalls and Firewall Policy.* Nat'l Inst. of Standards and Technology, Jan. 2002.
[4] Y. Bartal, A. Mayer, K. Nissim, and A. Wool, "Firmato: A Novel Firewall Management Toolkit," *ACM Trans. Computer Systems,* vol. 22, no. 4, pp. 381-420, Nov. 2004.
[5] A. Wool, "A Quantitative Study of Firewall Configuration Errors," *Computer,* vol. 37, no. 6, pp. 62-67, June 2004.
[6] H. Court, Knutsford, and Cheshire, "High-Availability: Technology Brief Firewall Load Balancing," High-Availability.Com, http://www.High-Availability.Com, 2003.
[7] "Firewall Load Balancing," Nortel Networks, www.nortel.com, 2009.
[8] "Check Point Firewall-1 Guide," Check Point, www.checkpoint.com, 2009.
[9] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein, *Introduction to Algorithms.* MIT Press, 2003.
[10] M.G. Gouda and A.X. Liu, "Firewall Design: Consistency, Completeness and Compactness," *Proc. Int'l Conf. Distributed Computing Systems (ICDCS '04),* pp. 320-327, Mar. 2004.
[11] A.X. Liu and M.G. Gouda, "Diverse Firewall Design," *Proc. IEEE Int'l Conf. Dependable Systems and Networks (DSN '04),* pp. 595-604, June 2004.
[12] M.G. Gouda and A.X. Liu, "A Model of Stateful Firewalls and Its Properties," *Proc. IEEE Int'l Conf. Dependable Systems and Networks (DSN),* June 2005.
[13] A.X. Liu, M.G. Gouda, H.H. Ma, and A.H.H. Ngu, "Firewall Queries," *Proc. Eighth Int'l Conf. Principles of Distributed Systems (OPODIS),* Dec. 2004.
[14] A.X. Liu, "Change Impact Analysis of Firewall Policies," *Proc. 12th European Symp. Research Computer Security (ESORICS),* Sept. 2007.
[15] A.X. Liu, "Formal Verification of Firewall Policies," *Proc. IEEE Int'l Conf. Comm. (ICC),* May 2008.
[16] A.X. Liu, E. Torng, and C. Meiners, "Firewall Compressor: An Algorithm for Minimizing Firewall Policies," *Proc. IEEE INFOCOM '08,* Apr. 2008.
[17] A. Wool, "The Use and Usability of Direction-Based Filtering in Firewalls," *Computers and Security,* vol. 23, no. 6, pp. 459-468, 2004.
[18] E.W. Fulp, "Optimization of Network Firewall Policies Using Ordered Sets and Directed Acyclical Graphs," *Proc. IEEE Internet Management Conf.,* 2005.
[19] E.S. Al-Shaer and H.H. Hamed, "Discovery of Policy Anomalies in Distributed Firewalls," *Proc. IEEE INFOCOM '04,* Mar. 2004.
[20] R.N. Smith, Y. Chen, and S. Bhattacharya, "Cascade of Distributed and Cooperating Firewalls in a Secure Data Network," *IEEE Trans. Knowledge and Data Eng.,* vol. 15, no. 5, pp. 1307-1315, Sept./Oct. 2003.
[21] R.N. Smith and S. Bhattacharya, "Firewall Placement in a Large Network Topology," *Proc. IEEE CS Workshop Future Trends Distributed Computing Systems (FTDCS '97),* 1997.

[22] H. Hamed, A. El-Atawy, and E. Al-Shaer, "On Dynamic Optimization of Packet Matching in High Speed Firewalls," *IEEE J. Selected Areas in Comm.*, vol. 24, no. 10, pp. 1817-1830, Oct. 2006.
[23] A. El-Atawy, T. Samak, E. Al-Shaer, and H. Li, "On Using Online Traffic Statistical Matching for Optimizing Packet Filtering Performance," *Proc. IEEE INFOCOM '07*, May 2007.
[24] P. Gupta and N. McKeown, "Algorithms for Packet Classification," *IEEE Network*, vol. 15, no. 2, pp. 24-32, Mar. 2001.
[25] P. Gupta and N. McKeown, "Packet Classification on Multiple Fields," *Proc. ACM SIGCOMM '99*, 1999.
[26] T. Lakshman and D. Stiliadis, "High-Speed Policy-Based Packet Forwarding Using Efficient Multi-Dimensional Range Matching," *Proc. ACM SIGCOMM '98*, 1998.
[27] A. Hari, S. Suri, and G. Parulkar, "Detecting and Resolving Packet Filter Conflicts," *Proc. IEEE INFOCOM '00*, Mar. 2000.
[28] V. Srinivasan, G. Varghese, S. Suri, and M. Waldvogel, "Fast and Scalable Layer Four Switching," *Proc. ACM SIGCOMM '98*, 1998.
[29] P. Gupta, "Algorithms for Routing Lookups and Packet Classification," PhD thesis, Stanford Univ., 2000.
[30] A.X. Liu and M.G. Gouda, "Removing Redundancy from Packet Classifiers," *Proc. Ann. IFIP Conf. Data and Applications Security*, Aug. 2005.
[31] C.R. Meiners, A.X. Liu, and E. Torng, "TCAM Razor: A Systematic Approach towards Minimizing Packet Classifiers in TCAMs," *Proc. IEEE Int'l Conf. Network Protocols (ICNP)*, Oct. 2007.
[32] A.X. Liu, C.R. Meiners, and Y. Zhou, "All-Match Based Complete Redundancy Removal for Packet Classifiers in TCAMs," *Proc. IEEE INFOCOM*, Apr. 2008.

MyungKeun Yoon received the BS and MS degrees in computer science from Yonsei University, Korea, in 1996 and 1998, respectively, and the PhD degree in computer engineering from the University of Florida in 2008. Since 1998, he has been working for the Korea Financial Telecommunications and Clearings Institute. His research interests include network security, network algorithm, and mobile network.

Shigang Chen received the BS degree in computer science from the University of Science and Technology of China in 1993, and the MS and PhD degrees in computer science from the University of Illinois at Urbana-Champaign in 1996 and 1999, respectively. After graduation, he had worked with Cisco Systems for three years before joining the University of Florida in 2002. His research interests include network security and wireless networks. He received the IEEE Communications Society Best Tutorial Paper Award in 1999 and the US National Science Foundation (NSF) CAREER Award in 2007. He was a guest editor for *ACM/Baltzer Journal of Wireless Networks (WINET)* and the *IEEE Transactions on Vehicle Technologies*. He served as a TPC cochair for the IEEE IWQoS '09 and the Computer and Network Security Symposium of IEEE IWCCC '06, the vice TPC chair for the IEEE MASS '05, the vice general chair for QShine '05, a TPC cochair for QShine '04, and a TPC member for many conferences including the IEEE ICNP, IEEE INFOCOM, IEEE ICC, IEEE Globecom, etc.

Zhan Zhang received the MS degree in computer science from Fudan University of China in 2003, and the PhD degree in computer science from the University of Florida in 2007. Since 2007, he has been working with Cisco Systems. His research interests include sensor networks and overlay networks.