Dynamic Balancing of Packet Filtering Workloads on Distributed Firewalls

Guanhua Yan[†] Songqing Chen[‡] Stephan Eidenbenz[†]

[†] Information Sciences (CCS-3)
Los Alamos National Laboratory
{ghyan, eidenben}@lanl.gov

[‡] Department of Computer Science
George Mason University
sqchen@cs.gmu.edu

***Abstract*—Firewalls are widely deployed nowadays to enforce security policies of enterprise networks. While having played crucial roles in securing these networks, firewalls themselves are subject to performance limitations. An overloaded firewall can cause severe damage to the protected enterprise network, because any legitimate communication through it is either degraded or even completely severed. In this paper, we address how to dynamically balance packet filtering workloads on distributed firewalls efficiently in large enterprise networks.**

We model dynamic load balancing on distributed firewalls as a minimax optimization problem, and show that it is strongly NP-complete even if we eliminate all precedence relationships among policy rules by rule rewriting. Accordingly, we propose a light-weight rule distribution scheme that quickly balances workloads among all firewalls. Our scheme is adaptive to incoming traffic. Moreover, dynamically placing and ordering policy rules on distributed firewalls reduces the probability that attackers successfully infer the rule distribution. Experimental results show that using a commodity PC, our approach can reduce the peak firewall workload in distributed firewall systems by 40% within less than five minutes, compared against alternative solutions that *only* optimize rule ordering on individual firewalls.

I. Introduction

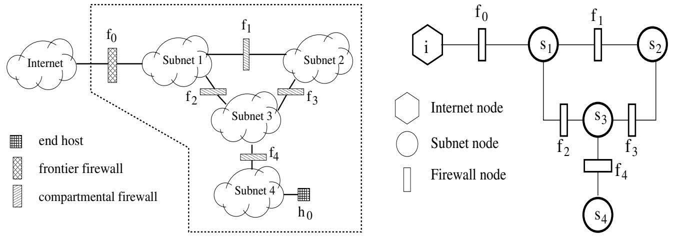
Firewalls are the most pervasive defense appliances that enforce security policies in today's enterprise networks. A firewall either controls the traffic that flows between an enterprise network and the outside Internet, or regulates the traffic among different domains in the same enterprise network. While having played crucial roles in securing these networks, firewalls themselves are subject to performance limitations. A firewall may be deployed on a high-speed link where its computational resource is the bottleneck. Next, as a response to a growing number of cyber-attacks in the Internet, enterprise networks tend to enforce tighter control over their traffic. As a result, the increasing complexity of security policies naturally increases the computational burden on firewalls. Moreover, from an attacker's perspective, overloading firewalls can cause desirably severe damage to the victim enterprise network, since any legitimate communication through these overloaded firewalls is either degraded or even completely severed. This is not impossible given the recent advances in firewall policy reconstruction by active probing [15].

It is thus important to protect firewalls from being overloaded. Two research directions have been pursued independently to address this problem. The first approach is optimizing rule ordering on individual firewalls [11], [1]. The second one exploits parallelism, that is, use multiple firewalls on the same link to perform packet inspection simultaneously [14], [7]. In this paper, we propose a solution that integrates merits of both methods to optimally balance packet inspection workloads on distributed firewalls. It is noted that a large enterprise network is usually comprised of multiple domains (or branches), each of which often has its own firewalls and policy rules. The key idea of our approach is to balance packet filtering workloads among the distributed firewalls within the same enterprise network such that the maximum workload among all individual firewalls is minimized. Our solution extends the wisdom from both research directions as mentioned: it not only exploits the parallelism inherent in the multiple firewalls within the same enterprise network, but also dynamically reorders the rules on each individual firewall based on traffic characteristics. Our approach, however, distinguishes itself from existing work by dynamically relocating firewall rules on the distributed firewalls as a response to the changing traffic characteristics.

Our key contributions in this paper are summarized as follows. We formulate the optimization of dynamic firewall rule distribution as a minimax problem subject to precedence constraints among firewall rules. We show that this problem is strongly NP-complete, even if we eliminate all the precedence constraints by rewriting firewall rules. Accordingly, we propose a light-weight rule distribution algorithm that quickly balances the packet filtering workload among all firewalls. Our approach does not demand any additional resource/infrastructure support, is adaptive to the changing traffic characteristics, and also reduces the probability that attackers successfully infer the rule distribution. Experimental results show that using a commidity PC, our algorithm can reduce the peak firewall workload in distributed firewall systems by 40% within less than five minutes, compared against alternative solutions that *only* optimize rule ordering on individual firewalls.

II. Background And Motivation

A distributed firewall system in an enterprise network enforces security policies on the traffic that enters or departs from the enterprise network or that traverses between internal subnets. We model a distributed firewall system as a set of firewalls falling into two categories: *frontier firewalls* and *compartmental firewalls*. The former separate the enterprise network from the outside Internet, and the latter reside between two subnetworks inside the enterprise network. In Fig. 1(1), we show these two types of firewalls in an example enterprise network. We further abstract the enterprise network topology into an undirected graph $G(V, E)$. In this graph, we define three types of nodes. The *internet node* i represents the outside Internet, a *subnet node* represents an internal subnet, and a *firewall node* represents a firewall. An edge is added between two nodes if the corresponding components in the original topology are directly connected. We use sets S and F to denote the entire set of subnet nodes and firewall nodes in the graph respectively. Following the same example, the enterprise network topology is abstracted into a graph with 10 nodes, as illustrated in Fig. 1(2). In this paper, we assume that there exists a unique routing path between any two nodes in graph $G(V, E)$. The assumption is reasonable because multi-path traffic engineering is rarely deployed in enterprise networks. Let set J be $\{i\} \cup S$ and we call nodes in set J *domain nodes*.



(1) Network topology (2) Abstracted graph

Fig. 1. An enterprise network topology and its abstracted graph

The security policy of an enterprise network regulates what types of traffic should be allowed among domain nodes in set J. In distributed firewalls, security policies are represented as firewall rules. A firewall rule is defined as a sextuple: $\langle pid, src\_ip, src\_port, dst\_ip, dst\_port, action\rangle$, where pid is protocol ID, src_ip and dst_ip are source and destination IP addresses, src_port and dst_port are source and destination port numbers, and $action$ indicates how a packet matching the rule should be processed. Typical firewall rule actions include "accept", "deny", and "drop", which may be coupled with "log" actions. Both "deny" and "drop" actions drop the packet, but the "deny" action sends a denying feedback to its source. Wildcard * is used to match a range of fields. In addition, we define the *matching space* of a firewall rule as the entire set of packets that matches it.

In the example, suppose that the IP address of the end host is 202.133.0.22 and there are two rules, r_1 and r_2, and r_1 precedes r_2. r_1 is $\langle \text{TCP}, 18.*.*.*, *, 202.133.0.22, 80, \text{accept}\rangle$ and r_2 is $\langle *, 18.*.*.*, *, 202.133.0.22, *, \text{deny}\rangle$. We assume that traffic from the outside Internet reaches host h_0 by traversing along the path $f_0 \rightarrow s_1 \rightarrow f_1 \rightarrow s_2 \rightarrow f_3 \rightarrow s_3 \rightarrow f_4 \rightarrow s_4 \rightarrow h_0$. Placing the two rules on *any* of the firewalls on this path can ensure that host h_0 accepts only HTTP traffic with port number 80 from the 18.0.0.0/8 network.

Such flexibility in policy rule placement is helpful in balancing firewall workloads, since we can strategically place firewall rules so that the maximum workload among all firewalls is minimized. This is the key idea of our approach. In this paper, we propose a *systematic* solution to the following questions: (1) *In order to achieve the security goal of the enterprise network, what constraints does a rule distribution scheme have to satisfy?* (Section III) (2) *To balance firewall workloads, what data should we collect from the system and how should we do that?* (Section IV) (3) *How should we characterize firewall workloads and what is the optimization objective function for balancing firewall workloads?* (Section V) (4) *How difficult is the problem?* (Section VI) (5) *What algorithm should we use to balance firewall workloads quickly and effectively?* (Sections VII and VIII). We evaluate the effectiveness and efficiency of our solution in Section IX.

III. Rule Distribution Constraints

Correlations among some policy rules prevent us from distributing the rules in a straightforward fashion. In the example, if we put r_1 on f_0 and r_2 on f_4, every packet destined to host h_0 from network 18.0.0.0/8 is dropped by firewall f_4 because of rule r_2. This suggests that a feasible rule distribution scheme is subject to certain constraints. To characterize these rule distribution constraints, we need to know how rules are organized on firewalls. List-based firewalls are a family of firewalls widely used in large enterprise networks [2]. In list-based firewalls, after the first rule matching the packet is found, the action specified by the rule is performed on the packet under inspection. Here, we consider only list-based firewalls.

RULES WITH PRECEDENCE RELATIONSHIPS

Let Φ denote the entire set of policy rules in an enterprise network. Define *precedence relationship* $\prec$ on two policy rules as follows: $r_i \prec r_j$ if rule r_i has a higher priority than rule r_j. Let Γ denote the entire set of precedence relationships among rules in Φ. We say that pair $\langle \Phi, \Gamma\rangle$ is *feasible* if it satisfies two conditions: (1) in Γ, there are no cyclic precedence relationships among rules in Φ, and (2) for any two rules in Φ whose matching spaces overlap with each other, there exists a precedence relationship between them in Γ. Here, we do *not* assume that a feasible pair $\langle \Phi, \Gamma\rangle$ is exhaustive, that is, some traffic may not match any rule in Φ. As to such traffic, an enterprise network must specify a default action. In our framework, we do not specify the default action, and such a decision is left to the network administrator for simplifying the presentation of rule set Φ.

Such flexibility comes at a cost in a distributed firewall system. If the default action is "deny" and if a packet does not match any rule on the last firewall along its path, this firewall does not know whether the packet matches a policy rule in Φ at an upstream firewall. To overcome such an ambiguity, we utilize some unused fields in IP headers and introduce a new *allowed-to-pass* (*ATP*) field. When a packet reaches the first firewall in the system, its *ATP* field is cleared. We assume that a firewall can tell whether it is the first firewall on the path of an incoming packet based on the port from which it comes from. When a packet is accepted by the first rule along its path, its *ATP* field is changed to 1. When a firewall does not find any rule to match a packet, it checks whether it is the last firewall on its path. If so and if it also finds that the *ATP* field of the packet is 0, it processes the packet according to the default action; otherwise, it forwards the packet.

CONSTRAINTS OF RULE DISTRIBUTION

Modeling constraints of rule distribution requires new notations. Let $\alpha(r)$ be the *entire* set of source domain nodes from which any traffic matching rule r can originate, and $\beta(r)$ be the *entire* set of domain nodes to which any traffic matching rule r can be destined. Both $\alpha(r)$ and $\beta(r)$ are computed from the source and destination IP address field of rule r. Following the example in Section II, $\alpha(r_1) = \{i\}$ and $\beta(r_1) = \{s_4\}$. Note that it is possible that $\alpha(r)$ or $\beta(r)$ for a specific rule r contains more than one domain node in J. In addition, we use $P(a, b)$, where $a, b \in J$, to denote the entire set of firewalls on the path from domain node a to domain node b. In the same example, $P(i, s_4) = \{f_0, f_1, f_3, f_4\}$.

Rule distribution matrix. A firewall rule distribution scheme can be modeled as a *rule distribution matrix* M. In the rule distribution matrix, $M[r][f]$, where $r \in \Phi$ and $f \in F$, is the order of rule r on the rule list at firewall f if rule r is placed on firewall f, or 0 otherwise. That is to say, if there are k rules, which we denote by $\tilde{r}_1$, $\tilde{r}_2$, ..., and $\tilde{r}_k$, on firewall f_s, and without loss of generality, we assume the order of rule $\tilde{r}_i$ is i, then we have $M[\tilde{r}_i][f_s] = i$.

Obviously, the number of rules placed on a firewall is the maximum order that a rule can have on it and no two rules on the same firewall should have the same order. Hence, a rule distribution scheme must satisfy the following two constraints in the form of first-order logic:

$$\begin{aligned}&\forall r \in \Phi, \forall f \in F :\\ &\quad 0 \le M[r][f] \le \textstyle\sum_{t\in\Phi} 1\{M[t][f] > 0\}, \qquad (1)\\ &\forall f \in F, \forall r \in \Phi, \forall t \in \Phi :\\ &\quad (r \neq t \wedge M[r][f] > 0 \wedge M[t][f] > 0)\\ &\qquad\qquad \rightarrow M[r][f] \neq M[t][f], \qquad (2)\end{aligned}$$

where $1\{p\}$ is the indicator function: If proposition p holds, it returns 1; otherwise, it returns 0. Hence, $\sum_{t\in\Phi} 1\{M[t][f] > 0\}$ gives the number of rules that are placed on firewall f.

Completeness constraint. We define the *traffic demand space* of rule r, denoted by $\Upsilon(r)$, as a set that includes every possible $\langle a, b\rangle$ pair where $a \in \alpha(r)$, $b \in \beta(r)$, and $a \neq b$. Intuitively, under an effective rule distribution scheme, every rule r in Φ should be placed on every domain node pair in $\Upsilon(r)$. That is, all possible paths from nodes in $\alpha(r)$ to nodes in $\beta(r)$. This, however, may not be necessary in some cases. For instance, suppose that Subnet 4 in Fig. 1 has prefix 202.133.0.0/24 and Subnet 3 has prefix 202.133.1.0/24. There are two policy rules: r_3 is $\langle \text{TCP}, 202.133.0.*, *, 202.133.1.*, 8080, \text{accept}\rangle$, r_4 is $\langle \text{TCP}, *, *, 202.133.1.*, 8080, \text{deny}\rangle$, and r_3 precedes r_4. We have: $\alpha(r_3) = \{s_4\}$, $\beta(r_3) = \{s_3\}$, $\alpha(r_4) = \{i, s_1, s_2, s_3, s_4\}$, and $\beta(r_4) = \{s_3\}$. We also assume that no domain nodes except s_4 have a routing path to node s_3 that traverses firewall f_4. It is obviously unnecessary to put rule r_4 on firewall f_4 because all traffic matching rule r_4 at firewall f_3 must be checked against rule r_3 first.

To deal with such a situation, we use $\mathcal{D}(r)$, where $r \in \Phi$, to denote the set of domain node pairs $\langle a, b\rangle$ that satisfy the following condition: any traffic from domain node a to domain node b must match another rule preceding rule r. If $\langle a, b\rangle \in \mathcal{D}(r)$, we say that rule r is *dominated* between node pair $\langle a, b\rangle$. In the example, we have $\mathcal{D}(r_4) = \{\langle s_4, s_3\rangle\}$. We say that a rule distribution matrix M is *complete* if every rule r in Φ is placed on all possible paths from nodes in $\alpha(r)$ to nodes in $\beta(r)$ except those covered by node pairs in $\mathcal{D}(r)$. The completeness constraint can be formalized as follows:

$$\begin{aligned}&\forall r \in \Phi : (\forall \langle a, b\rangle \in \Upsilon(r) - \mathcal{D}(r) :\\ &\quad (\exists f \in F : f \in P(a, b) \wedge M[r][f] > 0)) \qquad (3)\end{aligned}$$

Soundness constraint. We say that a rule distribution matrix M is *sound* if precedence relationships are never violated under M. In other words, a sound rule distribution matrix means that if $r_a \prec r_b$, then for every packet matching rule r_b, rule r_a must be checked before rule r_b. One obvious solution is that we always put rule r_a before rule r_b on every firewall rule list where rule r_b is placed. It is, however, not a necessary condition to satisfy the soundness property. To illustrate that, we first introduce a minor change to how a rule is processed (later in Section V, we will show that this change is unnecessary by adding some special rules): when a packet with its *ATP* set matches a rule, the packet is immediately forwarded, regardless of the action of the rule. Now, suppose that rule r_b is placed on firewall f. The precedence relationship is never violated if the following condition holds: for each path on which traffic matching rule r_a can possibly traverse, if firewall f is on this path, there always exists an upstream firewall, relative to firewall f, that installs rule r_a.

To formalize the soundness constraint, we let $\varphi(a, b, f)$ denote the order of firewall f among those on the path from domain node a to domain node b; if firewall f is not on the path from domain node a to domain node b, $\varphi(a, b, f)$ is $+\infty$. The soundness constraint can be formalized as follows:

$$\begin{aligned}&\forall f \in F, \forall r \in \Phi, \forall t \in \Phi :\\ &\quad (r \prec t \wedge M[r][f] > 0 \wedge M[t][f] > 0)\\ &\qquad \rightarrow M[r][f] < M[t][f], \qquad (4)\\ &\forall r_1 \in \Phi, \forall r_2 \in \Phi, \forall f \in F : (r_1 \prec r_2 \wedge M[r_2][f] > 0) \rightarrow\\ &\quad (\forall \langle a, b\rangle \in \Upsilon(r_1) : \varphi(a, b, f) < +\infty \rightarrow\\ &\qquad (\exists f' \in F : \varphi(a, b, f') \le \varphi(a, b, f) \wedge M[r_1][f'] > 0)).\\ &\qquad\qquad (5)\end{aligned}$$

IV. Hit Counting

Besides the distribution constraints, optimizing policy rule distribution also needs the frequencies at which rules are hit. Our approach ensures that hit counts collected are consistent across different rule distribution schemes. We assign a unique ID to each domain node in J. Let $uid(a)$ be the unique ID of domain node a. We also utilize some unused fields in IP headers for hit counting. The three new fields in an IP header are $\langle src\_nid, dst\_nid, flag\rangle$. The first two fields store the unique IDs of the source and destination domain nodes, and the $flag$ field stores a value chosen from $\{Marked, Hit\}$. The two flag values are some special bits so that they are distinguishable from the values when the $flag$ field is unused.

When a packet from the outside Internet arrives at a frontier firewall, the firewall checks whether the packet spoofs the source IP address by using a fake IP address belonging to the enterprise network; if not, the packet can be used for hit counting purpose. When a packet from the internal network arrives at a firewall, the firewall checks whether the packet comes from a domain node that is directly connected to it; if so, the packet can also be used for hit counting purpose. If a firewall finds a packet that can be used for hit counting purpose, it, with a predefined probability $p \in (0, 1]$, calculates the unique IDs of the source and destination domain nodes based on the source and destination IP addresses in the packet, inscribes these two unique IDs into the src_nid and dst_nid fields, and then sets the $flag$ field as $Marked$. We say that we *mark* a packet if we set its $flag$ field as $Marked$. Each firewall f also maintains a counter $K_m(f)$, which keeps the number of packets that are marked locally.

For each rule r placed on a firewall f, firewall f maintains a counter, denoted by $w_{r,f}^{\langle a,b \rangle}$, for every possible domain node pair $\langle a, b \rangle \in \Upsilon(r)$ satisfying $f \in P(a, b)$. When a marked packet matches rule r on firewall f, firewall f increases $w_{r,f}^{\langle a,b \rangle}$ by one, where a and b are the src_nid and dst_nid fields of the packet, and then updates the $flag$ field in the packet as Hit. But if the $flag$ field is already Hit, the corresponding counter is *not* updated, even if it matches a local rule.

A firewall also keeps a set of neighboring domain nodes to which it is directly connected. When a firewall fails to match any rule on it against a marked packet, it checks whether the dst_nid field of the packet matches any of its neighboring domain nodes; if it does, the firewall increases by one a locally maintained counter $u_f^{\langle a,b \rangle}$, where a and b are its src_nid and dst_nid fields. $u_f^{\langle a,b \rangle}$ stores the total number of packets that fail to match any rule on the path from domain node a to domain node b. As $u_f^{\langle a,b \rangle}$ is only updated at the last firewall on the path from domain node a to domain node b, domain node b must be directly connected to firewall f.

Recall that p denotes the marking probability. We measure:

$$w_r^{\langle a,b \rangle} = \frac{1}{p} \sum_{f \in F} w_{r,f}^{\langle a,b \rangle}, \quad r \in \Phi, \langle a, b \rangle \in \Upsilon(r) \qquad (6)$$

$$u^{\langle a,b \rangle} = \frac{1}{p} \sum_{f \in F} u_f^{\langle a,b \rangle}, \quad a \in J, b \in J, a \neq b. \qquad (7)$$

V. Optimization Objective Function

IMMEDIATE ALLOW-TO-PASS RULE

Given a complete and sound rule distribution matrix and the hit count for each rule, it is still difficult to characterize the workload on a firewall. For example, we have two rules r_a and r_b, whose matching spaces partially overlap. Suppose that rule r_a is placed on an upstream firewall relative to the one that rule r_b is placed on. To characterize the workload on the firewall where r_b is placed, we need to know how much traffic matches both rules. This part of traffic is forwarded immediately when checking it against rule r_b due to its positive ATP field. For the traffic that matches only rule r_a, it should be checked against subsequent rules.

To circumvent this problem, we introduce a special rule, which is called the *immediate allow-to-pass (IATP) rule*. The IATP rule is placed on every firewall *and* it is always the first rule on a firewall rule list. This rule checks the *ATP* field of every traversing packet, and if the *ATP* field is set, the packet is forwarded immediately. As a packet with a positive *ATP* field has already been accepted by a rule placed on an upstream firewall, the introduction of such special rules does not affect the completeness and soundness property.

Let r^{IA} be the IATP rule. The following constraint says that the first rule on each firewall rule list must be the IATP rule:

$$\forall f \in F : M[r^{IA}][f] = 1. \qquad (8)$$

As the special rule is added on every firewall, we need to modify Constraint (1) slightly as follows:

$$\begin{aligned} &\forall r \in \Phi, \forall f \in F : (M[r][f] = 0) \vee \\ &\quad (2 \leq M[r][f] \leq 1 + \sum_{t \in \Phi} 1\{M[t][f] > 0\}). \qquad (9) \end{aligned}$$

FIREWALL WORKLOAD CHARACTERIZATION

The workload on a firewall depends on the processing overhead of each rule on it, the order of the rules on it, and the incoming traffic pattern. Let $c_i(r)$ be the *inspection overhead* of rule r and $c_a(r)$ be the *action overhead* of rule r if a packet under inspection matches rule r. We assume that for any rule in $\Phi \cup \{r^{IA}\}$, its inspection cost is the same for all the packets that are checked against it.

The workload on a firewall includes several components. First, we consider the workload imposed when the firewall finds a regular rule in Φ that matches the packet under inspection. Due to the IATP rule, a packet can be checked against the rule that it matches only once. Let $\hbar_1(f)$ denote the workload of firewall f due to processing packets that matches a rule on firewall f. Then:

$$\begin{aligned} \hbar_1(f) = &\sum_{\forall r \in \Phi : M[r][f] > 0} \\ &\sum_{\forall \langle a,b \rangle \in \Upsilon(r) : Q_1(a,b,r,f)} w_r^{\langle a,b \rangle} (\sigma_{r,f} + c_i(r^{IA}) + c_a(r)), \quad (10) \end{aligned}$$

where

$$\sigma_{r,f} = \sum_{\forall r' \in \Phi : M[r'][f] \neq 0 \wedge M[r'][f] \leq M[r][f]} c_i(r'). \qquad (11)$$

and Proposition $Q_1(a, b, r, f)$ is defined as follows:

$$\begin{aligned} &Q_1(a, b, r, f) = \\ &\quad f \in P(a, b) \wedge \\ &\quad \neg(\exists f' \in F : M[r][f'] > 0 \wedge \varphi(a, b, f') < \varphi(a, b, f)) (12) \end{aligned}$$

Second, we consider the workload of firewall f due to checking against the IATP rule, denoted by $\hbar_2(f)$. We introduce the *primary action function* of a rule ζ: $\Phi \rightarrow \{accept, drop\}$; $\zeta(r)$ returns *accept* if rule r accepts the packets matching it or *drop* if rule r drops the packets matching it. Then, we have:

$$\hbar_2(f) = \sum_{\forall r \in \Phi : \zeta(r) = accept} \sum_{\forall \langle a,b \rangle \in \Upsilon(r) : Q_2(a,b,r,f)} w_r^{\langle a,b \rangle} \cdot (c_i(r^{IA}) + c_a(r^{IA})), \quad (13)$$

where Proposition $Q_2(a, b, r, f)$ is defined as follows:

$$\begin{aligned} &Q_2(a,b,r,f) = \\ &\quad f \in P(a,b) \wedge \\ &\quad (\exists f' \in F : M[r][f'] > 0 \wedge \varphi(a,b,f') < \varphi(a,b,f)). \end{aligned} \quad (14)$$

Third, we consider the workload of firewall f when it *cannot* find a rule that matches the packet under inspection. In this case, there are two sub-cases: it has to process those packets that do not match any rule on the firewalls along their paths, and it also has to process those packets that match a regular rule in Φ but that rule is placed *only* on downstream firewalls. We use $\hbar_3(f)$ and $\hbar_4(f)$ to characterize the workloads due to these two sub-cases, respectively. Let c_a^{def} denote the cost of the default action on a packet that does not match any rule in Φ, and c_a^{fwd} the cost of forwarding a packet when a firewall cannot find a rule that matches the packet but the packet is not destined to a domain node adjacent to it. Note that c_a^{fwd} should be no less than $c_i(r^{IA})$. We then have:

$$\hbar_3(f) = \sum_{\forall a \in J, \forall b \in J : a \neq b \wedge f \in P(a,b)} u^{\langle a,b \rangle} \cdot (\xi_f + c_a^{def}), \quad (15)$$

$$\hbar_4(f) = \sum_{\forall r \in \Phi : M[r][f] = 0} \sum_{\forall \langle a,b \rangle \in \Upsilon(r) : Q_1(a,b,r,f)} w_r^{\langle a,b \rangle} \cdot (\xi_f + c_a^{fwd}), \quad (16)$$

where

$$\xi_f = c_i(r^{IA}) + \sum_{\forall r' \in \Phi : M[r'][f] > 0} c_i(r'). \quad (17)$$

When marking an IP packet for hit counting purpose, a firewall also spends some CPU cycles on calculating its source and destination domain node IDs. We use $\hbar_5(f)$ to denote the workload of firewall f due to calculating domain node IDs. Let c_m denote the processing overhead on computing source and destination domain node IDs for a single packet. We have:

$$\hbar_5(f) = K_m(f) \cdot c_m, \quad (18)$$

where $K_m(f)$ denotes the total number of packets for which firewall f computes source and destination domain node IDs.

The overall workload on firewall f, denoted by $\mathcal{H}(f)$, is the sum of all the workloads $\hbar_1(f)$ through $\hbar_5(f)$:

$$\mathcal{H}(f) = \hbar_1(f) + \hbar_2(f) + \hbar_3(f) + \hbar_4(f) + \hbar_5(f). \quad (19)$$

The processing power of firewalls may differ significantly. Let the normalized workload function $\eta(f)$ be $\frac{\mathcal{H}(f)}{\mathcal{C}(f)}$, where $\mathcal{C}(f)$ is the processing capacity of firewall f. We then minimize the maximum normalized workload over all the firewalls and formulate it as a minimax optimization problem:

$$\min_{M} \max_{f \in F} \eta(f), \quad (20)$$

$$\text{subject to: } (9), (2), (3), (4), (5), \text{and } (8). \quad (21)$$

VI. Tractability Analysis

Optimizing the order of rules with precedence relationships on a *single* firewall is NP-hard [11]. As the optimal rule ordering problem on a single firewall is a special case of our problem, our problem must also be NP-hard. However, if no precedence relationship exists between rules, the optimal rule ordering problem on a single firewall is solvable in polynomial time. The solution is similar to that of the single-machine job scheduling problem without precedence constraints [16]. One may wonder whether our problem in the distributed setting is still solvable in polynomial time if no precedence constraints happen to exist among rules or if we eliminate all precedence relationships by arduously rewriting policy rules. In this section, we show that even if no precedence constraints exist, the minimax problem (20) is still *strongly* NP-complete. The theory of NP-completeness requires us to recast our problem to a decision problem. Hence, we define the *optimal non-precedence rule distribution problem* as follows:

Definition 1: **Optimal non-precedence rule distribution problem**. Let $\varnothing$ denote an empty set. Given are an undirected graph $G(V, E)$, a firewall set F, a feasible pair $\langle \Phi, \varnothing \rangle$, $\alpha(r)$, $\beta(r)$, and $\mathcal{D}(r)$ for each $r \in \Phi$, $P(a, b)$ for each $a \in V - F$ and each $b \in V - F$, $\varphi(a, b, f)$ for each $a \in V - F$, each $b \in V - F$ and each $f \in F$, $\mathcal{C}(f)$ for each $f \in F$, vectors $\vec{W}$ and $\vec{U}$, $c_i(r)$ and $c_a(r)$ for each rule $r \in \Phi \cup \{r^{IA}\}$, c_a^{def}, c_a^{fwd}, c_m, $K_m(f)$ for each firewall $f \in F$, and a given number Y. We ask: does there exist a rule distribution matrix M subject to Constraints (9), (2), (3), and (8) such that for every firewall f in F, $\eta(f)$ is no greater than Y?

Our proof of strong NP-completeness of the optimal non-precedence rule distribution problem involves the reduction of the 3-Partition problem, which is strongly NP-complete [8]. We establish the following theorem (proof given in [17]):

Theorem 1: The optimal non-precedence rule distribution problem is strongly NP-complete.

Given the difficulty in finding an optimal solution to the rule distribution problem, we resort to heuristic-based algorithms. To reduce the solution space, we establish two guidelines. First, if no traffic matching rule r can possibly traverse firewall f, or any traffic matching rule r, if traversing firewall f, must also match a higher priority rule r', then r should not be placed on firewall f. The second guideline avoids redundant rule deployment as much as possible: if on every possible path of rule r that traverses firewall f, there exists an upstream firewall deploying rule r, it is unnecessary to place rule r on firewall f. These two guidelines are formalized as follows:

$$\begin{aligned} &\forall r \in \Phi, \forall f \in F : \\ &\quad \neg(\exists \langle a,b \rangle \in \Upsilon(r) - \mathcal{D}(r) : f \in P(a,b)) \rightarrow M[r][f] = 0. \end{aligned} \quad (22)$$

$$\begin{aligned} &\forall r \in \Phi, \forall f \in F : \\ &\quad (\forall \langle a,b \rangle \in \Upsilon(r) : f \in P(a,b) \rightarrow \\ &\qquad (\exists f' \in P(a,b) : \varphi(a,b,f') < \varphi(a,b,f) \wedge M[r][f'] > 0)) \\ &\qquad\quad \rightarrow M[r][f] = 0. \end{aligned} \quad (23)$$

VII. Single-Firewall Rule Ordering

From a high level perspective, a rule distribution scheme has two phases: rule placement phase and rule ordering phase. The first phase decides how to place rules on firewalls, and the second one decides how to order the rules placed on each firewall. Once the first phase finishes, the second phase changes only workload function $\hbar_1(f)$ in Eq. (19). More specifically, the rule ordering scheme only affects the following component in Eq. (19):

$$\tilde{H}(f) = \sum_{\forall r \in \Phi : M[r][f] > 0} \quad \sum_{\forall \langle a,b \rangle \in \Upsilon(r) : Q_1(a,b,r,f)} w_r^{\langle a,b \rangle} \cdot \sigma_{r,f}. \quad (24)$$

Obviously, $\tilde{H}(f)$ includes neither action overheads associated with the regular rules nor the processing costs associated with the IATP rule. The problem of finding a rule ordering scheme to minimize $\tilde{H}(f)$ can be reduced to the classical $1|prec|\Sigma w_j C_j$ single-machine job scheduling problem. This problem is known to be strongly NP-hard, and a 2-approximation algorithm for it is provided in [10]. As this algorithm is based on an LP (Linear Programming) relaxation technique, we call it the *LP-based algorithm*.

Suppose we already have an ordering solution X that minimizes $\tilde{H}(f)$ given a set of rules placed on firewall f. If we apply the LP-based algorithm whenever we want to add a new rule onto firewall f, the computation cost will be very high, as demonstrated in later experiments. We thus use a light-weight alternative, which is called the *fast rule insertion algorithm*.

Fast rule insertion algorithm. We use a linked list to maintain rules placed on a firewall. Let $\mathcal{L}(f)$ denote such a list kept for firewall f and the rules on $\mathcal{L}(f)$ be $\tilde{r}_0$, $\tilde{r}_1$, ..., and $\tilde{r}_k$. Note that $\tilde{r}_0$ is the IATP rule. For ease of presentation, we add a virtual rule $\tilde{r}_{k+1}$ at the tail. Suppose that we want to insert rule r onto list $\mathcal{L}(f)$. The precondition is that rule r is not on list $\mathcal{L}(f)$. We traverse the linked list, looking for $\underline{r}$, the last rule that precedes over rule r, and $\overline{r}$, the first rule that is preceded by rule r. If such rule $\underline{r}$ cannot be found, we let it be $\tilde{r}_0$; similarly, if such rule $\overline{r}$ cannot be found, we let it be $\tilde{r}_{k+1}$. Obviously, $\overline{r}$ now cannot be the same as rule $\underline{r}$. We thus distinguish two cases: rule $\underline{r}$ appears before rule $\overline{r}$ (**Case A**) and rule $\underline{r}$ appears after rule $\overline{r}$ (**Case B**). In either case, we let the rules between $\underline{r}$ and $\overline{r}$ are $\tilde{r}_i$, $\tilde{r}_{i+1}$, ..., and $\tilde{r}_j$. Let set R^* contain all these rules. Let $\tilde{w}(r')$ denote $\sum_{\forall \langle a,b \rangle \in \Upsilon(r') : Q_1(a,b,r',f)} w_{r'}^{\langle a,b \rangle}$. We also number the position before rule $\tilde{r}_i$ as $i-1$, the one immediately after it as i, ..., the one immediately before rule $\tilde{r}_j$ as $j-1$, and the one immediately after it as j. The fast rule insertion algorithm inserts rule r only at one of these positions.

In **Case A**, rule r can be inserted in any place among positions $i-1$, i, ..., and j. We note that where to insert rule r does not affect the inspection costs of the rules before rule $\tilde{r}_i$ and those of rules after rule $\tilde{r}_j$. Hence, we only need to minimize the total inspection cost associated with rule r and rules $\tilde{r}_i$ through $\tilde{r}_j$ that is induced by the traffic matching *only* these rules. We use $\mathcal{C}_{inp}(k)$ to denote this cost when rule r is inserted at position k. Let $\Delta_{k,k+1}$ be $\mathcal{C}_{inp}(k+1) - \mathcal{C}_{inp}(k)$. We then have: $\Delta_{k,k+1} = \tilde{w}(r) \cdot c_i(\tilde{r}_{k+1}) - \tilde{w}(\tilde{r}_{k+1}) \cdot c_i(r)$. We can use one pass on rules $\tilde{r}_i$ through $\tilde{r}_j$ to compute the value of $\mathcal{C}_{inp}$ when rule r is placed at position $i-1$, and then use the iterative formula $\Delta_{k,k+1}$ to find the position where $\mathcal{C}_{inp}$ has the lowest value with another pass. Thereafter, we insert rule r at that position.

Case B is more complicated than the first one. For each rule r' on list $\mathcal{L}(f)$, we keep both sets of rules on $\mathcal{L}(f)$ that precede rule r' and are preceded by rule r', and they are denoted by $\underline{S}(r')$ and $\overline{S}(r)$, respectively. Then, we use BFS (Breadth First Search) or DFS (Depth First Search) to traverse the rules according to their precedence relationships and obtain a sequence of rules $\underline{R}$ that satisfy the following conditions: (a.1) $\forall r' \in \underline{R} : r' \in R^*$, (a.2) $\forall r' \in \underline{S}(r) : r' \in \underline{R}$, (a.3) $r' \in \underline{R} \rightarrow (\forall r'' \in \underline{S}(r') : r'' \in \underline{R})$, and (a.4) the relative ordering of rules in $\underline{R}$ remains the same as they are on list $\mathcal{L}(f)$. Similarly, we derive a sequence of rules $\overline{R}$ that satisfy the following conditions: (b.1) $\forall r' \in \overline{R} : r' \in R^*$, (b.2) $\forall r' \in \overline{S}(r) : r' \in \overline{R}$, (b.3) $r' \in \overline{R} \rightarrow (\forall r'' \in \overline{S}(r') : r'' \in \overline{R})$, and (b.4) the relative ordering of rules in $\overline{R}$ remains the same as they are on list $\mathcal{L}(f)$. Finally, we obtain a sequence of rules R' that satisfy: (c.1) $\forall r' \in R' : r' \in R^*$, (c.2) $\forall r' \in R^* - \underline{R} - \overline{R} : r' \in R'$ and (c.3) the relative ordering of rules in R' remains the same as they are on list $\mathcal{L}(f)$.

Let the numbers of rules in $\underline{R}$, $\overline{R}$, and R' be $\underline{n}$, $\overline{n}$, and n'. We also use $\underline{Z}[k]$ and $\overline{Z}[k]$ to denote the number of rules in $\underline{R}$ and $\overline{R}$ that appear before rule $R'[k]$ on list $\mathcal{L}(f)$, respectively. We further define vectors $\underline{W}$, $\overline{W}$, W', $\underline{C}$, $\overline{C}$, C' as follows:

k	Hit-counts	Costs
$1, ..., \underline{n}$	$\underline{W}[k] = \sum_{t=1}^{k} \tilde{w}(\underline{R}[t])$	$\underline{C}[k] = \sum_{t=1}^{k} c_i(\underline{R}[t])$
$1, ..., \overline{n}$	$\overline{W}[k] = \sum_{t=1}^{k} \tilde{w}(\overline{R}[t])$	$\overline{C}[k] = \sum_{t=1}^{k} c_i(\overline{R}[t])$
$1, ..., n'$	$W'[k] = \sum_{t=1}^{k} \tilde{w}(R'[t])$	$C'[k] = \sum_{t=1}^{k} c_i(R'[t])$

With a constant number of passes on the rules $\tilde{r}_i$ through $\tilde{r}_j$, we can obtain the values of all these vectors.

Our algorithm dealing with Case $\mathcal{B}$ guarantees that after rule insertion, (1) rules in $\underline{R}$ appear before rule r and rules in $\overline{R}$ appear after rule r, (2) all rules appearing before rule r conform to the same order on the original list and the same to the rules that appear after rule r, and (3) all the rules in R' appear in the same order as before the insertion. Similar to Case $\mathcal{A}$, we only need to minimize the total inspection cost associated with rule r and rules $\tilde{r}_i$ through $\tilde{r}_j$ that is induced by the traffic matching *only* these rules. We use $\mathcal{C}_{inp}^{(k)}$ to denote this cost when there are k rules in R' that are placed before rule r. The baseline ordering scheme is putting all rules in R' after rule r. Under this scheme, we use another pass on the rules $\tilde{r}_i$ through $\tilde{r}_j$ to compute $\mathcal{C}_{inp}^{(0)}$. Let $\Delta^{(k,k+1)}$ be $\mathcal{C}_{inp}^{(k+1)} - \mathcal{C}_{inp}^{(k)}$. We compute $\Delta^{(k,k+1)}$ by:

$$\begin{aligned}\Delta^{(k,k+1)} = {} & c_i(R'[k+1]) \times \\ & (\underline{W}[|\underline{R}|] - \underline{W}[\underline{Z}[k+1]] + \tilde{w}(r) + \overline{W}[\overline{Z}[k+1]]) \\ & - \tilde{w}(R'[k+1]) \times \\ & (\underline{C}[|\underline{R}|] - \underline{C}[\underline{Z}[k+1]] + c_i(r) + \overline{C}[\overline{Z}[k+1]]), \quad (25)\end{aligned}$$

where $|X|$ is the number of rules in X. With $\mathcal{C}_{inp}^{(0)}$ and

$\Delta^{(k,k+1)}$, we can compute $\mathcal{C}_{inp}^{(1)}$, $\mathcal{C}_{inp}^{(2)}$, ..., and $\mathcal{C}_{inp}^{(|R'|)}$. Let $\mathcal{C}_{inp}^{(k^*)}$ be the smallest one among them and $\mathcal{C}_{inp}^{(0)}$. Then, in the final solution, there are k^* rules in R' before r.

Given the description of the fast rule insertion algorithm, we can establish its performance with the following lemma:

Lemma 1: Given that there are n rules that are already ordered, the number of precedence relationships among them is m, and the time to check the existence of a precedence relationship between two rules in Φ is $t_{\prec}$, the time complexity of the fast rule insertion algorithm is $O(n \cdot t_{\prec} + m)$.

The time complexity of $t_{\prec}$ depends on the implementation: if we use a matrix to store the pairwise precedence relationship between rules, $t_{\prec}$ is constant (this can be true even if the matrix is sparse [6]); if we instead use a two-level trie to store the precedence relationships, $t_{\prec}$ is $O(log(|\Phi|))$.

VIII. Rule Distribution Algorithm

In this section, we introduce a rule distribution algorithm, which works in a centralized fashion once hit counts per rule have been collected from all firewalls. Using our algorithm, a dedicated machine computes a new rule distribution matrix and then uses it to reconfigure the rules on each firewall. The process repeats as new rule hitting counts are collected and sent to the dedicated machine.

PHASE I: INITIAL SETUP

We place each rule r as follows: for every node pair $\langle a, b\rangle \in \Upsilon(r) - \mathcal{D}(r)$, we place rule r on the first firewall along the path from domain node a to domain node b. The intuition behind it is as follows: if we put a rule whose primary action is *drop* on firewalls close to its source domain nodes, traffic matching this rule can be dropped early, imposing no workload on downstream firewalls; if we put a rule whose primary action is *accept* on firewalls close to its source domain nodes, packets matching this rule can have their *ATP* fields set early and downstream firewalls can thus immediately forward such traffic due to the IATP rule. The initial rule placement scheme as described must satisfy the following proposition:

$$\begin{aligned} \forall r \in \Phi, \forall f \in F: \quad & M[r][f] > 0 \\ & \rightarrow (\exists \langle a, b\rangle \in \Upsilon(r) - \mathcal{D}(r) : \varphi(a, b, f) = 1) \end{aligned} \tag{26}$$

It is easy to see that the rule distribution matrix M from the initial rule placement scheme must satisfy Constraints (9), (2), (4), (8), (3) and (5), and Guidelines (22) and (23).

Before entering Phase II, our algorithm initializes a few data structures. For each rule r placed on firewall f, we keep the set of domain node pairs between which the rule is effective. Let $\mathcal{I}(r, f)$ be this set and initialize it as follows:

$$\mathcal{I}(r, f) = \{\langle a, b\rangle \mid \langle a, b\rangle \in \Upsilon(r) - \mathcal{D}(r) \wedge \varphi(a, b, f) = 1\}.$$

Given $\mathcal{I}(r, f)$, we can derive the total hit count of rule r on firewall f as $\sum_{\langle a,b\rangle \in \mathcal{I}(r,f)} w_r^{\langle a,b\rangle}$. This is used to compute the total workload on firewall f or order the rules on firewall f by the fast rule insertion algorithm. Moreover, for each firewall f in Φ, we also keep a set of domain node pairs for every rule r that used to be on list $\mathcal{L}(f)$; we use $\widehat{\mathcal{I}}(r, f)$ to denote this set and initialize it to be empty.

PHASE II: RULE MIGRATION

In this phase, we migrate rules between firewalls such that the maximum normalized workload among all firewalls monotonically decreases. The *rule migration algorithm* always tries to migrate rules away from the firewall with the highest normalized workload. Suppose that this firewall is f_0 and its normalized workload is η_{max}^{old}. One input parameter to this algorithm is its *resolution* ϵ ($\epsilon > 0$). The resolution parameter means each successful attempt on reducing the workload on firewall f_0 should make its new normalized workload at most $\eta_{max}^{old} - \epsilon$. We define *color*$(f)$ as the *color* of firewall f, which has three possible values: *white*, *red*, and *black*. Initially, we set the color of firewall f^* as *red* and that of any other firewall as *white*. We put firewall f^* on the *red firewall list*, which keeps all the firewalls with the red color. Moreover, we define the *state* of a firewall f as a collection of information regarding it, including $\mathcal{L}(f)$, $\mathcal{I}(r, f)$ for each $r \in \mathcal{L}(f)$, and $\widehat{\mathcal{I}}(r, f)$ for each rule r that used to be on $\mathcal{L}(f)$; we use $s(f)$ to denote the current state of firewall f. We also let set Π contain all the firewalls whose states have been changed when the current red firewall is processed. By slightly abusing notation Π, we use $\Pi(f)$ to denote the state information kept for firewall f.

Algorithm 1 Aggressively migrate rules to reduce the highest normalized workload among all firewalls

```
loop
  $f^* \Leftarrow \text{argmax}_{f \in F}\eta(f)$, and $\eta_{max}^{old} \Leftarrow \eta(f^*)$
  $color(f^*) \Leftarrow red$, and put $f^*$ onto the red firewall list
  while the red firewall list is not empty do
    extract a firewall $f$ from the red firewall list
    add $f$ onto $\Pi$ and keep $s(f)$ in $\Pi(f)$
    if MigrateRules($f$, $vertical$) returns false then
      if MigrateRules($f$, $horizontal$) returns false then
        for each $f \in \Pi$, rollback $s(f)$ to $\Pi(f)$
        the algorithm terminates
      end if
    end if
  end while
  clear $\Pi(f)$ for each $f \in \Pi$, and then clear $\Pi$
end loop
```

The main body of the algorithm is shown in Algorithm 1. Note that it calls twice function $MigrationRules(f, mode)$ in Algorithm 2. The two modes are $vertical$ and $horizontal$. Both modes start from the tail of the firewall list, and iteratively test whether the rule can be migrated onto the all immediate downstream firewalls. The difference between them is that in the vertical mode, a rule is migrated to downstream firewalls *only if* it does not make any of them overloaded, that is, all downstream firewalls after rule migration must have a normalized workload at least ϵ below the current highest normalized workload η_{max}^{old}. By contrast, in the vertical mode, we migrate a rule away from firewall f' *even if* it makes the normalized workload on some downstream firewalls higher than $\eta_{max}^{old} - \epsilon$. In this mode, we change the colors of such

Algorithm 2 MigrateRules(f, $mode$)

```
Require: $f \in F$ and $mode \in \{vertical, horizontal\}$
{/* INITIALIZATION */}
$T_0 \Leftarrow \emptyset$, $T_1 \Leftarrow \emptyset$, $r \Leftarrow$ last rule on $\mathcal{L}(f)$
PROCESS_RULE:
if $(\exists \langle a,b \rangle \in \mathcal{I}(r,f) : \varphi(a,b,f) = |F|)$ or $(\exists r' \in \mathcal{L}(f) : r \prec r')$ then
    if $r$ is the first rule on $\mathcal{L}(f)$, return false
    $r \Leftarrow$ the next rule on $\mathcal{L}(f)$ in reverse order
    goto PROCESS_RULE;
end if
for all $\langle a,b \rangle \in \mathcal{I}(r,f)$ do
    $f' \Leftarrow$ next firewall on the path from $a$ to $b$
    $T_0 \Leftarrow T_0 \cup \{f'\}$
end for
for each $f' \in T_0$, initialize $W[r][f']$ to be 0
for all $\langle a,b \rangle \in \mathcal{I}(r,f)$ do
    $f' \Leftarrow$ the next firewall on the path from $a$ to $b$
    $W[r][f'] \Leftarrow W[r][f'] + w_r^{\langle a,b \rangle}$
end for

{/* TRY MOVING RULE DOWNSTREAM */}
for each $f'$ in $T_0$ do
    $T_1 \Leftarrow T_1 \cup \{f'\}$, keep $s(f')$ (state of $f'$) in $\pi(f')$
    if $f' \notin \Pi$, add $f'$ onto $\Pi$ and keep $s(f')$ in $\Pi(f')$
    for each $\langle a,b \rangle$ in $\widehat{\mathcal{I}}(r,f')$, $W[r][f'] \Leftarrow W[r][f'] + w_r^{\langle a,b \rangle}$
    add rule $r$ onto $\mathcal{L}(f')$ with fast rule insertion algo.
    recompute $\eta(f')$
    if $\eta(f') > \eta_{max}^{old} - \epsilon$ then
        if $mode$ is $vertical$ then
            for each firewall $f_x \in T_1$, rollback $s(f_x)$ to $\pi(f_x)$
            if $r$ is the first rule on $\mathcal{L}(f)$, return false
            $r \Leftarrow$ the next rule on $\mathcal{L}(f)$ in reverse order
            goto PROCESS_RULE;
        else if $mode$ is $horizontal$ then
            if $color(f')$ is $white$ then
                $color(f') \Leftarrow red$, put $f'$ on the red firewall list
            else if $color(f')$ is $black$ then
                for each $f_x \in T_1$, rollback $s(f_x)$ to $\pi(f_x)$
                if $r$ is the first rule on $\mathcal{L}(f)$, return false
                $r \Leftarrow$ the next rule on $\mathcal{L}(f)$ in reverse order
                goto PROCESS_RULE;
            end if
        end if
    end if
end for

{/* RULE REVOCATION PROCESS */}
for each $f'$ in $T_0$ do
    for each $\langle a,b \rangle \in \widehat{\mathcal{I}}(r,f)$ do
        add $\langle a,b \rangle$ onto $\mathcal{I}(r,f')$, and remove it from $\widehat{\mathcal{I}}(r,f')$
        find firewall $f''$ in $P(a,b)$ with $r \in \mathcal{L}(f'')$ and $\langle a,b \rangle \in \mathcal{I}(r,f'')$, remove $\langle a,b \rangle$ from $\mathcal{I}(r,f'')$, and if $\mathcal{I}(r,f'')$ becomes empty, remove $r$ from $\mathcal{L}(f'')$
        for each $f''' \in P(a,b)$ between $f'$ and $f''$ do
            remove $\langle a,b \rangle$ from $\widehat{\mathcal{I}}(r,f''')$
        end for
    end for
end for

{/* RULE REMOVAL */}
remove $r$ from $\mathcal{L}(f)$
for each $\langle a,b \rangle \in \mathcal{I}(r,f)$ , move it from $\mathcal{I}(r,f)$ to $\widehat{\mathcal{I}}(r,f)$
if $r$ is the first rule on $\mathcal{L}(f)$, return false
$r \Leftarrow$ the next rule on $\mathcal{L}(f)$ in reverse order
goto PROCESS_RULE
```

firewalls to *red* and put them onto the red firewall list. As the algorithm iteratively migrates rules from firewalls with red color until it becomes empty, the horizontal mode tries to reduce the workload of a firewall by involving firewalls that are multiple hops away. To ensure the convergence of the algorithm, we do not allow a firewall with a black color to turn red again. Hence, if a rule added to a black downstream firewall makes its normalized workload higher than $\eta_{max}^{old} - \epsilon$, we do not migrate this rule. This is the reason why we try the vertical mode before trying the horizontal mode: the former does not increase the number of red firewalls in the system.

Function $MigrationRules(f, mode)$ uses the fast rule insertion algorithm to add a rule onto a downstream firewall. It is worthy mentioning some special cases when a rule is being migrated onto a firewall f'. First, if the rule is already on that firewall, we remove it from the firewall list and reapply the fast rule insertion algorithm to adjust its position. Second, if the rule used to be on that firewall but now has been migrated onto further downstream firewalls, we need to activate the *rule revocation process*. This is because if we place rule r on firewall f', these downstream firewalls also deploying rule r *will not* see any traffic that matches rule r and also traverses firewall f'. The rule revocation process updates the corresponding states along the path from firewall f' to these downstream firewalls.

The algorithm requires recomputing the workload of a firewall in several places. To reduce the computation cost, we let each rule on a firewall carry a list of domain node pairs. We compute the workload of each firewall after the initial setup, and in the course of the algorithm, we incrementally update it based on the information carried along with the domain node pairs. The algorithm terminates after polynomial time and the proof is given in [17].

IX. Experiments

EXPERIMENTAL SETUP

Topology. In our experiments, we use two different topologies: full d-ary trees and random graphs. In the experiments with tree topologies, we let d be 4 and vary the height of the tree h between 3 and 4. In a tree topology, we assume that all leaf nodes are domain nodes; hence, there are either 16 or 64 domain nodes in total. The random graphs are generated using the Doar-Leslie model [5]; this model has the nice feature that the total number of edges in the graph can be controlled by the mean degree of an node. In a random graph, we assume every node is a domain node. For both tree topologies and random graphs, we add an extra node i that represents the outside Internet. In tree topologies, we connect node i to the root node; in random graphs, we randomly pick a node connecting to node i. In each of these topologies generated, we put a firewall on every edge.

Rules. In all experiments, the primary action of a rule is randomly chosen between *accept* and *drop*. The inspection cost of a rule is uniformly distributed between 0.0001 and 0.0009. If the primary action of a rule is *accept*, its action cost is 0.0006; otherwise, its action cost is 0.0004. The total

ID	Topology	#Firewalls	#Domains	#Rules	TD
X_1	Tree	21	17	2100	TD_1
X_2	RG	20.4	17	2040	TD_1
X_3	Tree	85	65	4250	TD_1
X_4	RG	82.3	65	4115	TD_1
X_5	Tree	85	65	4250	TD_2

TABLE I
SIMULATION SCENARIOS (RG STANDS FOR RANDOM GRAPH AND TD STANDS FOR THE SETTING OF RULES' TRAFFIC DEMAND SPACES).

number of rules is k times the number of firewalls in the topology. We choose k between 50 and 100 in our experiments. Let μ be the number of rules that regulate traffic from or to the outside Internet. We call such rules *external rules*. The probability that an external rule regulates inbound traffic is 0.5 in all experiments. We vary μ between 0.25, 0.5, and 0.75 and assume that for each rule r, $\mathcal{D}(r)$ is empty.

Traffic demand space. We consider two cases here. In **Case** TD_1, we assume that each rule's traffic demand space only has one domain node pair. Hence, if a rule is an external rule, we choose the internal domain node randomly; otherwise, we choose two different internal domain nodes randomly. **Case** TD_2 applies only to the tree topology. If a rule is an external rule, we choose a node in the tree (not node i, but not necessarily leaf nodes) randomly; otherwise, we choose two different nodes in the same way. If an internal node in the tree is chosen, then all the leaf domain nodes in the subtree rooted at it are covered by the rule.

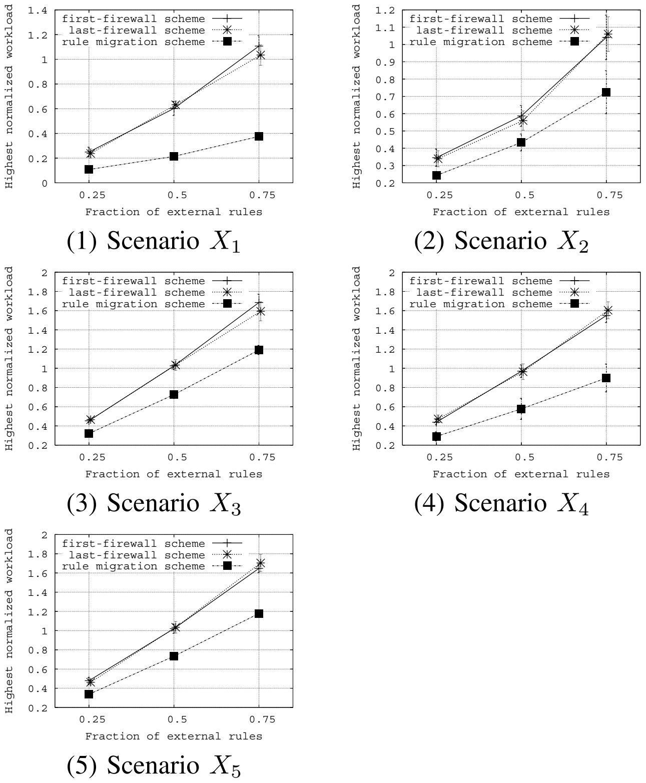
Hit counts. We generate $k \cdot |F|$ rules sequentially. If a new rule's traffic demand space overlaps with that of a rule that has already been generated, the probability that the latter precedes the new rule is 0.5. Note that overlapping traffic demand spaces of two rules do not mean that their matching spaces also overlap. We apply the Zipf's law to generate the hit count of each rule, based on the observation in [4]. In all experiments, we assume that the total hit counts for all the rules is 10000, and the exponent in the Zipfian distribution is 1. Between each pair of domain nodes, the hit count for the traffic that does not match any rule on the path is uniformly drawn between 0 and 100.

Miscellaneous. $c_i(r^{IA})$ = 0.0001; $c_a(r^{IA})$ = 0.0005; c_a^{fwd} = c_a^{def} = 0.001. c_m = 0.0005. $\epsilon = 0.1\%$. We set c_a^{fwd} and c_a^{def} higher than the average rule action cost because if a firewall cannot find a rule on it to match a packet, it needs to check whether it is the last firewall on the packet's path.

The scenarios considered in our experiments are summarized in Table I. The capacity of each firewall is 2000 in Scenarios X_1 and X_2, and 4000 in the remaining ones. For each scenario we simulate 10 sample runs.

EXPERIMENTAL RESULTS

Effect. First, we show how effectively our algorithm helps reduce the highest normalized workload among all firewalls. We compare our algorithm against two straightforward schemes: *first-firewall scheme* and *last-firewall scheme*. In the first scheme, for each rule generated, we deploy it on the first firewall on the path between every domain node pair in its traffic demand space, and in the second one, we put it on the last firewall on the path between every domain node pair in its traffic demand space. Fig. 2 depicts the highest normalized workload under Scenarios X_1 through X_5 as we vary μ, the fraction of external rules. Regardless of what rule distribution scheme is applied, the highest normalized workload increases with μ. This is because as we increase μ, the workload of the frontier firewall becomes heavier. When μ is 0.75, the highest normalized workloads in Scenarios X_3, X_4, and X_5 reach about 160%, suggesting that 60% of the incoming traffic has to be dropped. From the attacker's view, this is desirable because legitimate traffic is dropped as well.

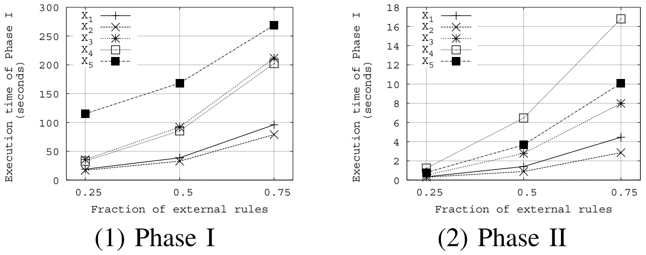


(1) Scenario X_1 (2) Scenario X_2

(3) Scenario X_3 (4) Scenario X_4

(5) Scenario X_5

Fig. 2. Highest normalized workloads (conf. interval 95%)



(1) Phase I (2) Phase II

Fig. 3. Execution times of Phases I and II

Our algorithm lessens such damage by rule migration. In all five scenarios, the rule migration scheme reduces the highest normalized workload by about 40% on average, compared against the other two straightforward schemes. It is noted that rule migration may not eliminate firewall overloading completely in some cases. For example, in Scenarios X_3 and X_5, when μ is 0.75, the scheme reduces the highest normalized workload from 160% to 120%. Although some legitimate traffic inevitably has to be dropped, the damage caused by the attack significantly decreases. However, in Scenarios X_1, X_2, X_4, when μ is 0.75, our rule migration scheme can completely absorb the attack effects.

Performance. We evaluate our approach on a machine with a 1.4GHz CPU and 3G memory. It installs Redhat Enterprise Linux kernel version 2.4.18. The implementation of the LP-based rule ordering algorithm in Phase I uses GLPK version 4.16, an open source linear programming module [9]. Fig. 3 depicts the execution times of both phases in our algorithm under different scenarios. In all the cases, our algorithm terminates within five minutes. Moreover, although the machine we used has a large memory, the runtime memory usage is less than 2% in all the sample runs.

Moreover, we note that the execution time of Phase I is one order of magnitude longer than that of Phase II, regardless of the simulation scenario. In all the scenarios, Phase II takes less than 20 seconds to finish rule migration. This suggests that to further improve the efficiency of our algorithm, we should focus on shortening the time needed to produce an ordered rule list for each firewall in Phase I. The rule ordering time by the LP-based algorithm grows super-linearly with the number of rules on a firewall. This is confirmed by the observation that the execution time of Phase I increases monotonically with the fraction of external rules, even though the total number of rules in the system does not change. A higher fraction of external rules leads to a longer rule list at the frontier firewall in the initial setup and thus a longer time to order these rules by the LP-based algorithm. There are two approaches to further reduce the execution time of Phase I. First, the execution workload in Phase I is parallelizable. We can split firewalls into groups and let a processor to order the firewall rule lists in each group. Second, as observed in [4], the frequency at which a rule is found to match an incoming packet is skewed in normal conditions. So we can use the LP-based algorithm to order only rules with high hit counts and those preceding them, and for the remaining rules, any ordering solution that does not violate precedence relationships is acceptable.

X. Related Work

Most existing work on improving performance of firewalls focuses on rule organization on individual firewalls. Hamed and Al-Shaer suggest that firewall rule ordering should take traffic characteristics into consideration [11] and they later propose using alphabetic trees to accelerate packet filtering on individual firewalls [12]. OPTWALL accelerates packet filtering by partitioning the original list of firewall rules into a hierarchical set of mutually disjoint rule subsets [1]. Our work aims to optimize rule distribution among distributed firewalls; we assume list-based firewalls in our work due to their popularity. The idea of rule migration for load balancing discussed in Section VIII is still applicable if other types of data structures are used on individual firewalls, although the workload functions presented in Section V need be revisited.

Current research on distributed firewalls mainly considers rule consistency and implementation issues. In [3], Al-Shaer et al. propose an algorithm to discover anomalies among distributed firewall rules. Yuan et al. have developed FIREMAN, which detects not only violations of user-specified security policies but also inconsistencies and inefficiencies among firewall rules [18]. In [13], Ioannisdis et al. discussed some implementation issues of distributed firewalls. Instead, our work focuses on how to dynamically optimize placement and ordering of policy rules on distributed firewalls according to changing traffic characteristics. To the best of our knowledge, this topic has not been investigated before.

XI. Conclusion

Firewalls are the most pervasively deployed security appliances nowadays. Due to its own resource limitation, a firewall can become a point of failures under severely high computational workload. In this paper, we address how to optimize placement and ordering of distributed firewall rules to mitigate the worst-case damage that can occur to individual firewalls. We model the problem as a minimax optimization problem and propose a heuristic-based algorithm to migrate rules among distributed firewalls. Experimental results show our solution can balance workloads on distributed firewalls effectively and efficiently.

References

[1] S. Acharya, M. Abliz, B. Mills, and T. Znati. Optwall: A hierarchical traffic-ware firewall. In *Proceedings of NDSS'05*, Febuary 2007.

[2] S. Acharya, J. Wang, Z. Ge, T. F. Znati, and A. Greenberg. Traffic aware firewall optimization strategies. In *Proceedings of ICC'06*, June 2006.

[3] E. Al-Shaer, H. Hamed, R. Boutaba, and M. Hasan. Conflict classification and analysis of distributed firewall policies. *IEEE JSAC*, 23(10), October 2005.

[4] E. Cohen and C. Lund. Packet classification in large ISPs: Design and evaluation of decision tree classifiers. In *Proceedings of SIGMETRICS'05*, 2005.

[5] M. Doar and I. Leslie. How bad is naive multicast routing? In *Proceedings of IEEE INFOCOM'93*.

[6] M. L. Fredman, J. Komlos, and E. Szemeredi. Storing a sparse table with 0(1) worst case access time. *Journal of the ACM*, 31(3), 1984.

[7] E. W. Fulp. Parallel firewall designs for high-speed networks. In *Proceedings of High Speed Networking Workshop, INFOCOM'06*, 2006.

[8] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W H Freeman & Co, 1979.

[9] Glpk (gnu linear programming kit). http://www.gnu.org/software/glpk/.

[10] L. A. Hall, A. S. Schulz, D. B. Shmoys, and J. Wein. Scheduling to minimize average completion time: Off-line and on-line approximation algorithms. *Mathematics of Operations Research*, 22, 1997.

[11] H. Hamed and E. Al-Shaer. Dynamic rule-ordering optimization for high-speed firewall filtering. In *Proc. of ASIACCS'06*, March 2006.

[12] H. Hamed, A. El-Atawy, and E. Al-Shaer. Adaptive statistical optimization techniques for firewall packet filtering. In *Proceedings of IEEE INFOCOM'06*, 2006.

[13] S. Ioannidis, A. Keromytis, S. Bellovin, and J. Smith. Implementing a distributed firewall. In *Proceedings of ACM CCS*, Athens, Greece, November 2000.

[14] C. Kopparapu. *Load Balancing Servers, Firewalls, and Caches*. John Wiley & Sons, Inc., 2002.

[15] T. Samak, A. El-Atawy, E. Al-Shaer, and H. Li. Firewall policy reconstruction by active probing an attacker's view. In *The 2nd Workshop on Secure Network Protocols*, 2006.

[16] W. E. Smith. Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3(1), March 1956.

[17] G. Yan, S. Chen, and S. Eidenbenz. Dynamic balancing of packet filtering workloads on distributed firewalls. Technical Report LA-UR-07-3281, Los Alamos National Laboratory, 2007.

[18] L. Yuan, J. Mai, Z. Su, H. Chen, C. Chuah, and P. Mohapatra. Fireman: A toolkit for firewall modeling and analysis. In *Proceedings of IEEE Symposium on Security and Privacy*, May 2006.