PARALLEL FIREWALL DESIGNS FOR HIGH-SPEED NETWORKS

By

Ryan Joseph Farley

A Thesis Submitted to the Graduate Faculty of

WAKE FOREST UNIVERSITY

in Partial Fulfillment of the Requirements

for the Degree of

MASTER OF SCIENCE

in the Department of Computer Science

December 2005

Winston-Salem, North Carolina

Approved By:

Errin W. Fulp, Ph.D., Advisor _____

Examining Committee:

V. Paul Pauca, Ph.D., Chairperson _____

William H. Turkett, Ph.D. _____

Acknowledgements

The foundations of this paper were first laid out nearly four years ago while an undergraduate student at Wake Forest University. Since then, many people have supported me. To those not mentioned explicitly, I wish to thank you deeply.

In particular, I would like to thank my family for their constant encouragement and belief in what I can do. Also, I would like to thank Dr. Errin Fulp, who gave me the opportunity to help evolve, contribute to, and build on the ideas of this paper. And finally, thanks to the U.S. Department of Energy's funding through grant DE-FG02-03ER25581[1]. Their support has indirectly paid for my tuition and stipend over the past two years, making my Master's education, and this thesis, possible.

[1]This work was supported by the U.S. Department of Energy MICS (grant DE-FG02-03ER25581). The views and conclusions contained herein are those of the author and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the DOE or the U.S. Government.

Table of Contents

Illustrations

List of Tables

List of Figures

Abbreviations

Acronyms

ACL - Access Control List.
ARP - Address Resolution Protocol.
ASIC - Application Specific Integrated Circuit.
CERT - Computer Emergency Response Team.
DAG - Directed Acyclical Graph.
DoS - Denial of Service.
FPGA - Field Programmable Gate Array.
Gbps - Gigabits per second.
IP - Internet Protocol.
MAC - Medium Access Control.
Mbps - Megabits per second.
NIC - Network Interface Card.
NPU - Network Processing Unit.
QoS - Quality of Service.
TCP - Transmission Control Protocol.
UDP - User Datagram Protocol.

Symbols

A is the accept set of a policy R, and A_i is the accept set of a local policy R_i.
C is the set of all possible packets.
d is a packet, or datagram.
D is the deny set of a policy R, and D_i is the deny set of a local policy R_i.
$G = (R, E)$ is a DAG representing a policy, where E is the set of precedence edges.
h_i is the number of packet matches, or hits, over a specified time on rule r_i.
m is the number of nodes, or machines, running as firewall nodes in the system.
n is the rule count of a policy R, and n_i is the rule count a local policy R_i.
p_i is the hit ratio percentage of h_i to the sum of all rules' hit counts.
r is a rule, and r_i is the i^{th} rule in a policy.
R is an ordered set of rules known as a policy.
R_i is a local policy that exists on a node in the array, where $i = 1, \ldots, m$.
λ is the measure of throughput, or arrival rate, in a system.
μ is the processes per time measurement.

Abstract

PARALLEL FIREWALL DESIGNS FOR HIGH-SPEED NETWORKS

Ryan Joseph Farley

Firewalls enforce a security policy between two networks by comparing arriving packets against the policy rules to determine if they should be accepted or denied. Unfortunately, security processing imposes significant delays on routing activity in relation to the complexity and size of the policy. These delays become more apparent as network speeds increase and performance requirements heighten. Thus, the need to improve firewall performance will only increase over time.

This thesis introduces a novel parallel firewall design, where firewall nodes collectively enforce a security policy. The proposed model can perform inspections under increased traffic loads and higher traffic speeds in a scalable manner. To accomplish this, each parallel firewall node implements a portion of the policy, a form of function parallelism, and packets are processed by all firewall nodes simultaneously, ensuring a packet's exposure to the entire policy.

Since each firewall node has fewer rules to process per packet, the proposed function parallel system can achieve significantly lower delays and higher throughput than both non-parallel and data parallel (load-balancing) firewalls. Furthermore, unlike data parallel systems, the new function parallel design allows stateful inspection of packets, a critical component in preventing certain types of network attacks. These advantages will be demonstrated theoretically and empirically through experiments and simulations.

Chapter 1: Modern Network Security Issues

The Internet has become a broad medium of virtually unlimited purposes. However, while some are legitimate or benign, there are a vast number of malicious, or even illegal uses. Many of these malevolent actions are targeted at other users of the Internet, from corporations to individuals, and may not be driven by any particular motivation other than pure malice. It is easy to imagine that as the Internet grows, so will the number of abusive users and thus network security issues. In fact, CERT states that since 2000 reported incidents have increased from 21,756 to 137,529 [5]. In order to properly secure an internal network from traffic that could lead to an incident, network infrastructures need to provide control to internal resources. Technology to manage access control has found a place in daily operations with many organizations, including those with strong emphasis on confidential data such as military, government, and corporations with large data stores.

This chapter introduces methods for controlling and securing access to protected resources. In particular, an overview of firewalls is given, followed by a discussion on why firewall performance improvement is needed, particularly in high-speed networks. Then examples of current methods for optimizing current firewall designs

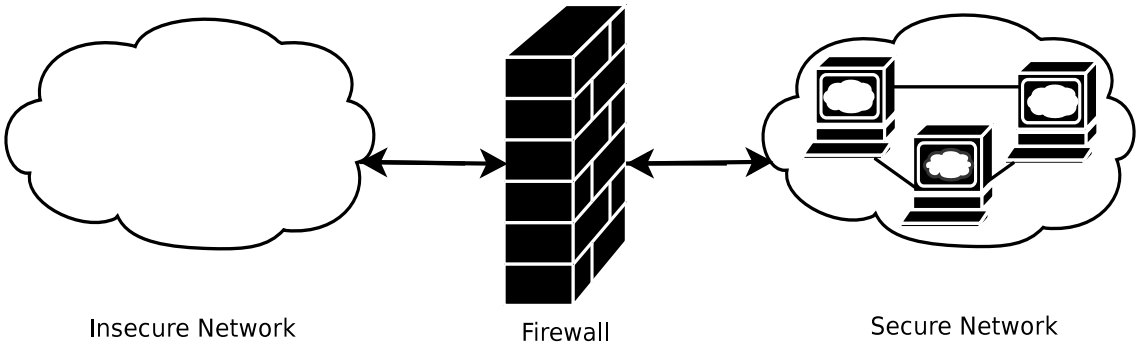


Figure 1.1: Figure of a firewall between two networks.

are listed. Finally, parallel firewalls will be introduced, including the new function parallel design.

1.1 Firewalls

The term firewall is used to describe an implementation which manages access controls for network traffic. This includes not just packet filtering, but also compatibility with network services such as Quality of Service (QoS) enforcement and auditing. Quality of Service is the idea that transmission rates, loss rates, and other characteristics of network traffic can be measured, improved, and, to some extent, guaranteed in advance. Filtering is accomplished according to an ordered set of rules, known as either a security policy, or Access Control List (ACL), that defines the action to perform on matching packets, as seen in figure 1.2 [33, 34]. That is, rules indicate the action to take place for each packet, such as accept, deny, forward, or redirect.

Security can be further enhanced with connection state information, allowing dynamic decisions to be made from the knowledge of previous decisions. For example, a table can be used to record the state of each connection, which is useful for preventing certain types of attacks (e.g., TCP SYN flood) [34].

Conventionally, incoming and outgoing traffic of a system is managed by a router presiding at the gateway to the external network or internal subnets. Routing is accomplished with basic table information that pairs requested destinations with known routes. Since this point is the possible intersection of different security policies, it is also a prevalent location for firewalls. Thus firewalls are often utilized in place of, or in conjunction with, a router. Traditionally, a firewall is implemented as a single dedicated machine running a service, as either software or hardware, that applies the policy to each arriving packet.

1.2 Policy Methods for Packet Matching

Every firewall is given a policy to enforce, generally in the form of an ordered set of rules. When incoming packets match rules in the policy an action is determined. In regards to which rule an arriving packet ultimately matches, there are three main schemes: best match, last match, and first match.

A *best match* method reviews the entire policy per packet and the rule that most tightly bounds the packet information is considered a match. This is similar in logic

1 Deny all traffic
2 Accept traffic from host x with any service
3 Deny traffic from any host with service y

Figure 1.2: Example pseudo-policy with "all traffic" rule at top

1 Accept traffic from host x with any service
2 Deny traffic from any host with service y
3 Deny all traffic

Figure 1.3: Example pseudo-policy with "all traffic" rule at bottom

to packet routing, except a routing algorithm, known as longest match prefix [7, 8, 11, 25], limits the decision to only the destination IP tuple. One-dimensional best match semantics, such as routing, can be performed quickly. However, multi-dimensional scenarios, such as with firewalls, require time that could contribute to a bottlenecking condition. In the example pseudo-policy figure 1.2, if traffic from host x with service z is tested, corresponding packets would match the first two rules, but since rule 2 is more specific, the action would be to accept the packet. If traffic from host x with service y is tested, then corresponding packets would match all rules, but additional time may be required to determine the final match (which rule had tighter geometric bounds). Note since rule ordering for the best match method is not important, the "all traffic" rule could be anywhere in the policy.

A *last match* method also reviews the entire policy for each packet, but instead of closest match, the last rule in the policy that matches is used. In the example pseudo-policy figure 1.2, if traffic from host x with service z is tested, corresponding

packets would match the first and second rules, and since rule 2 would be the last match, the packet would be accepted. If traffic from host x with service y is tested, corresponding packets would match all rules, but rule 3 would be the last match, and the packet would be denied. Note that the "all traffic" rule must be the first rule in policy, otherwise all rules preceding it would never be used.

A *first match* method compares packets sequentially to a policy, in a top-down manner. The first matching rule is applied and further comparison processing for that packet is stopped. Since a packet can at most match one rule, this method does not need to track previous matches, reducing decision overhead. In the example pseudo-policy figure 1.3, if traffic from host x with service z is tested, corresponding packets would match the first rule only and be immediately accepted without consideration of the last two rules. If traffic from host x with service y is tested, corresponding packets would not match the first rule, and rule 2 would be the first match, thus the packet would be denied. Note that the "all traffic" rule must be the last rule in the policy. If placed anywhere else, then no succeeding rules would be processed.

In summary, the best match method is advantageous since it provides the closest possible match to any rule, but unfortunately, handling the multifaceted metric can be more complicated and can require more time than the other two approaches. The last match method suffers in performance since, like the best match method, it requires

processing the entire policy to determine a match. The first match method has a similar pitfall to the last match method, in that improper rule order is an anomaly designers must consider [1, 2]. However, first match has a distinct advantage of not requiring a packet to traverse the complete policy before a decision can be made. While first match, last match, and best match method can be optimized by reducing the policy size, first match can be further optimized by reordering the rules (described in section 2.4).

1.2.1 Policy Implementation or Representation

Although the previous section describes a policy in terms of a list of rules, the internal representation can be a non-linear data structure such as a trie or graph [15, 14]. Consider a case where each layer of a trie is used to represent a portion of rule information. In comparisons, at each level the associated packet information could be used, and the leaf would represent the final action to execute [15]. The edges connecting nodes could be arranged as to be concordant with a particular list based match method. However, the trie traversal could allow quicker results than a sequential search.

While tree and graph traversal methods would result in better searching performance, management of a policy in tree or graph form is too complicated for most designers to input directly. Thus, an arrangement where input is in list form, but searching was done in tree form is advantageous for the policy designer [15]. This

person would be able to create and manage the policy in a human thought oriented list based environment, but the implementation could be executed with less time complexity by using optimized algorithms such as trie or tree traversal. In fact, in practice the majority of implementations use list based representations [25]. Thus, many of these implementations allow standard list input, and preprocessing could be done to convert the list into non-linear format.

1.3 Allowing for High Speed Networks

In section 1.1, it was described how firewalls are generally utilized in place of, or conjunction with, a router. However, packet filtering can incur a significantly higher processing load than routing alone [24, 27, 34], even with current network speeds. As an example, a firewall that interconnects two 100 Mbps networks would have to provide the capability to process over 300,000 packets per second [33].

Successfully handling traffic by a firewall proves more difficult over time not only as policies become more complex [3, 23, 34], but also as physical line speeds increase. Notable High Speed Networks (HSNs) include the United States Department of Energy's UltraScienceNet and the Experimental Science Network. These networks connect research labs at 5 Gbps across the US such as: Los Alamos, PNNL, and Oakridge National Labs.

In HSNs, a single firewall can easily become a bottleneck, increasing the suscepti-

bility to Denial of Service (DoS) attacks [3, 9, 17, 19]. For example, an attacker could simply inundate the firewall with traffic, delaying or preventing legitimate packets from being processed.

Most approaches for improving firewall performance for HSNs fall under the following categories: optimization of the policy to reduce time spent finding the appropriate action [14]; usage of optimized software [15, 14] or faster hardware [31, 32]; and, parallelization of the entire firewall implementation [6]. The first approach to improve firewall performance is to optimize a policy so as to reduce the average number of comparisons required for a packet, and thus eliminate unneeded time involved in processing. This is an interesting area of research that is applicable to all systems, including legacy devices. Several optimization concepts and guidelines and how they relate to the proposed design will be described in section 2.4, but advanced methods will be left open for future development.

Faster hardware to execute firewall software is another way to improve performance. While building a faster single firewall is possible [10, 24, 27, 29, 31], the benefits are temporary (traffic loads and interface speeds are always increasing). It is also not scalable in regards to traffic load, is a single point of failure, and is generally not cost-effective for all installations. This thesis will focus on the last solution, parallelization.

1.4 The Parallelization of Network Firewalls

A parallel firewall is a scalable approach for achieving the speed of network traffic inspection [3, 19, 23, 34] required for increased network speeds and traffic loads. In this section data parallel [3] and function parallel designs [12] for parallel firewalls are described.

Like their distributed computing definitions imply, data parallel segments the data set across the array of firewalls, while function parallel distributes the work set (policy in this case) across the array of firewalls. Distribution of policies and traffic (packets) is done in such a way as to preserve policy integrity, ensuring that parallel designs and traditional single firewalls always reach the same decision for any packet. Examples of policy distribution techniques are described in this thesis.

The data parallel approach is a scalable alternative to a single firewall that allows for greater throughput capabilities. Function parallel designs can reduce the processing time required on any firewall node yielding improved performance. Furthermore, unlike data parallel designs, the proposed function parallel architecture can provide stateful inspections. This thesis shows that function parallel designs are scalable solutions that can offer consistently better performance and more capabilities than other designs. Experimental results will show the new architecture can achieve a vast reduction in processing time as compared to other parallel firewall designs.

Chapter 2: Firewall Modeling Concepts and Considerations

This chapter details how a firewall accomplishes policy enforcement by examining rules, the policies themselves, and key concepts of firewall optimization. The basics of firewall optimization will be used in the following chapter to highlight the advantages of the proposed architecture.

2.1 Rule Structure and Matching

A rule r is an ordered tuple, $r = (r[1], r[2], \ldots, r[k])$. Each tuple $r[l]$ is a finite set that can be fully specified, given as a range, or contain wildcards `'*'` in standard prefix format [33, 34]. For the Internet, security rules are commonly represented as five tuples consisting of: protocol type, source IP address, source port number, destination IP address, and destination port number [33, 34]. For example, the prefix `192.*` would represent any IP address that has `192` as the first dotted-decimal number. In addition each rule has an action, which in the most basic form is to accept or deny. While rules can specify more information beyond the above 5-tuple [33, 34], such as arriving or departing network interface, these constraints are extraneous to the

packet header information required for firewalls and are situationally dependent on the implementing machine's configuration.

Similar to a rule, a packet (IP datagram) d can be viewed as an ordered k-tuple $d = (d[1], d[2], \ldots, d[k])$; however, ranges and wildcards are not possible for any packet tuple. Therefore, a packet is a single point in the set of all possible packets (which is finite), while a rule is an area in the set of all possible rules. A rule's action is applied when a packet d *matches* a rule r, $(d \Rightarrow r)$.

Definition A packet $d = (d[1], d[2], \ldots, d[k])$ matches rule r if $d[l] \subseteq r[l], l = 1, \ldots, k$.

A match is found between a packet and rule when every tuple of the packet is a subset of the corresponding tuple in the rule. Since the tuples of packets and rules directly correspond, a match will happen if the point representing the packet exists within the area representing the rule.

2.2 Policy Modeling Issues

A security policy can be modeled as an ordered set of n rules, denoted as $R = \{r_1, r_2, \ldots, r_n\}$. State, or the storing of previously determined matches for decisions on same connection related packets, can be viewed as a preliminary extension of the policy that contains a set of rules for established connections [33]. Since state information is the same as what is used for the rules, the two can coexist as long

No.	Proto.	Source IP	Source Port	Destination IP	Destination Port	Action
1	UDP	1.1.*	*	*	80	deny
2	TCP	2.*	*	1.*	90	accept
3	UDP	*	*	1.*	*	accept
4	TCP	2.*	*	1.*	20	accept
5	UDP	1.*	*	*	*	accept
6	*	*	*	*	*	deny

Table 2.1: Example security policy consisting of multiple ordered rules.

as state is processed before the policy. It is important to note that the internal representation of the policy does not have to be a list [16, 24]. In fact, in regards to state rules, the representative data structure can easily be a hash table or other structures which corresponds sequence numbers, or some other combination of data, to the associated action, as described in section 1.2.1.

There are essentially two types of policies, those that accept by default, and those that deny by default. The common assumption is that in a default accept strategy, the human policy management effort either relatively trusts the traffic on the connection and knows all permutations of traffic to block, or cannot keep up with a complex set of traffic to accept and the policy becomes easier to manage and quicker to process (i.e. smaller). Opposing this, the default deny strategy appeals to a scenario where the assumption is either the majority of traffic is untrustworthy (i.e. Internet gateway), or the set of accept exceptions is clear and manageable.

When considering the set of all possible packets, C, in regards to a policy, it

is possible to categorize each unique packet by which action (accept or deny) the policy would associate with that packet or if the policy would not be able to make an association. Note, this thesis will assume that accept and deny are the only actions available when deciding the outcome of a packet (while other actions may act as policy traversal controls). Those packets accepted will be collectively known as the accept set A, those denied will be collectively known as the deny set D, and those that the policy cannot decide on are in the undecidable set U. Therefore $A \cup D \cup U = C$.

Definition A policy R is comprehensive if for the set of all possible packets $\bar{D} = A$ and $U = \emptyset$.

This is to say, that a policy is comprehensive if for every possible packet, a match is found using R. Given the permutations of rules required to match all possible traffic, comprehensiveness can be maintained by using a policy default.

Definition A policy default for the first match method is a rule, or implied rule, that:

1. Is processed after all other rules in the policy fail to match.
2. Matches any packet that will be processed by the policy.
3. Executes an action that leads to no further deliberation on the packet (i.e. only accept or deny).

In the example policy table 2.1, using the first match method, r_6 is the policy default because it is the last rule processed, matches all packets, and has a deny action. As another point the default action of deny categorizes this policy as one that must explicitly accept traffic. Had the policy default action been accept then it would be a policy which must explicitly deny traffic.

Assume that two policies R and R', with accept sets A and A' respectively, are comprehensive and R does not necessarily equal R', then the two policies can be considered equivalent using the following definition.

Definition Two comprehensive policies R and R' are equivalent if $A = A'$.

Another important definition that builds on equivalence is integrity.

Definition If the policy R is replaced by the equivalent policy R', then the integrity of R is preserved.

Given that many firewall models exist, it is clear that there are many ways to implement any given policy or even modify it (e.g. reorder, combine, add, or remove rules), making it important to be able to decide equivalence, thus determining if the integrity of an original policy is preserved.

2.3 Precedence Relationships and Visualizations

Using the first match method, if the ordered tuples of a rule r_i are subsets of the tuples associated with r_j, where $i < j$, then a precedence relationship is established between r_i and r_j. This idea is important for policy optimization, during which the policy is reordered and manipulated to reduce the time spent determining matches.

A policy contains implied precedence relationships where certain rules must appear before others if the integrity of the policy is to be maintained. For example, consider the policy in table 2.1. Rule r_1 must appear before rules r_3, r_5, and r_6, because its tuples are subsets of r_3, r_5, and r_6. Likewise rule r_6 must be the last rule in the policy because its tuples are supersets of all the other rules. If rule r_5 was moved to the beginning of the policy, then it would *shadow* [2] the original rule r_1.[1] Shadowing is an anomaly that occurs when a rule r_j is a proper subset of a preceding rule r_i, where $i < j$. As a result of the relative order, r_j will never be utilized. Notice, there is no precedence relationship between rules r_1, r_2, or r_4 given in table 2.1. Therefore, the relative order of these three rules will not impact the policy integrity and can be changed to improve firewall performance [1, 2].

As described in [13, 14], the precedence relationship between rules in a policy can be modeled as a policy Directed Acyclical Graph (DAG). Let $G = (R, E)$ be a

[1]It is important to note that similar to the set comparison involved in packets matching rules, actions are not considered in establishing rule shadowing either.

policy DAG for a policy R, where vertices are rules and edges E are the precedence relationships.

Definition The intersection of rule r_i and r_j, denoted as $r_i \cap r_j$ is

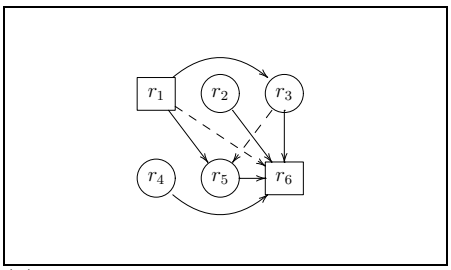
$$r_i \cap r_j = (r_i[l] \cap r_j[l]), \quad l = 1, \ldots, k.$$

Therefore, the intersection of two rules results in an ordered set of tuples that collectively describes the packets that match both rules.

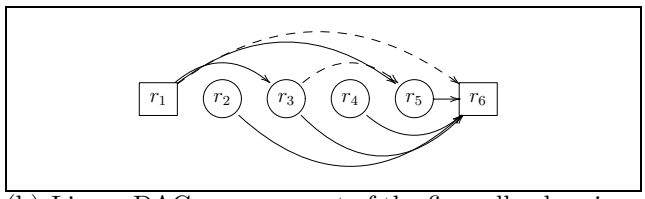
Definition Two rules r_i and r_j intersect, if every tuple of their intersection, $r_i[l] \cap r_j[l]$, is non-empty, $l = 1, \ldots, k$.

In contrast, rules r_i and r_j do not intersect, if at least one tuple is the empty set. Note the intersection operation is commutative; therefore, if r_i intersects r_j, then r_j will intersect r_i and the same is true for rules that do not intersect. A precedence relationship, or edge, exists between rules r_i and r_j, if $i < j$ and the rules intersect [13].

For example, consider the rules given in table 2.1, the intersection of r_1 and r_5 yields `(UDP, 1.1.*, *, *, 80)`. The rules' actions are not considered in the intersection or match operations. Since these two rules intersect, a packet can match both rules; for example, $d =$ `(UDP, 1.1.1.1, 80, 2.2.2.2, 80)`. The relative order must be maintained between these two rules and an edge drawn from r_1 to r_5 must



(a) Policy DAG of the firewall rules given in table 2.1.



(b) Linear DAG arrangement of the firewall rules given in table 2.1.

Figure 2.1: Two visualizations of a policy DAG. Vertices are rules while edges indicate precedence requirements. Note, a dashed arrow indicates a same-action precedence edge.

be present in the DAG, as seen in figure 2.1(a). In contrast consider rules r_2 and r_4 in the same policy. These two rules do not intersect due to the fifth tuple (destination port). A packet cannot match both rules indicating the relative order can change; therefore, an edge does not exist between them in the policy DAG.

2.4 Policy Optimization

A policy DAG is used to optimize a policy in order to reduce the time [13, 14] required to find a match while retaining integrity. This section will introduce the concept of a policy profile, then use the policy DAG to reorder the rules for optimization.

Statistically, in any firewall policy model, given a varying traffic demographic, it is probable that particular rules in the set will have a higher frequency of first matches than others. Matches are also known as 'hits,' denoted as h_i, where i is the i^{th} rule in the policy. The knowledge of this distribution gathered over time is the policy profile.

No.	Prob.
1	0.01
2	0.02
3	0.10
4	0.17
5	0.20
6	0.50

Table 2.2: Example security policy profile, corresponding to table 2.1.

A policy profile, example is seen in table 2.2, brings to light an optimization consideration when using a first match policy. Since traffic is processed by comparing each packet to the policy until a match is found, if rules with higher hit ratios are at the bottom of the policy, then the average comparisons per packet is non-optimal. Let $P = \{p_1, p_2, \ldots, p_n\}$ be the policy profile, where $p_i = h_i / \sum_{j=1}^{n} h_j$ is the probability, or hit ratio, that a packet will first match the i^{th} rule in the policy. In a comprehensive policy, every packet will find a match, thus $\sum_{i=1}^{n} p_i = 1$. To calculate the average comparisons before a match is found per packet, $E[n]$, on a first match policy, we use

$$E[n] = \sum_{i=1}^{n} i \cdot p_i \tag{2.1}$$

If a policy were reordered such that the hit ratios were in decreasing order, then the most common traffic would be handled in the quickest manner, reducing the average comparison per packet.

Unfortunately, precedence imposes a constraint when reordering a policy, preventing an ideal highest to lowest hit ratio order. The goal then is to rearrange a policy

without violating precedence issues. DAGs provide a method of maintaining relationships between rules while still allowing reordering. This starts with a policy in DAG form presented in a linear arrangement, as seen in figure 2.1(b). To transform a DAG into a linear arrangement, make an order of all DAG vertices such that any successor of a vertex V appears only after vertex V in the sequence. It is proven that any linear arrangement of a policy DAG maintains integrity [14].

Theorem 1 *Let G be a DAG for a policy R, then any linear arrangement of G maintains integrity.*

Proof: Assume a policy DAG G is constructed from a policy R that is free of shadowing. Consider any two rules r_i and r_j in the policy, where $i < j$. If an edge between r_i and r_j in G does not exist, then a linear arrangement of G can interchange the order of the two rules. An edge will not exist if the rules do not intersect; thus, a reorder will not effect integrity since a packet cannot match both rules. If an edge does exist between the rules, then their relative order will be maintained in every linear arrangement of G; thus maintaining precedence and integrity [14]. ■

While it is clear that DAG's provide a way to ensure integrity maintenance, providing a sorting algorithm is the difficult part and advanced methods remain an open question. This is primarily due to the fact that any reorder of the policy would change the initial variables required to sort the policy (e.g. hit ratio), creating a highly re-

cursive situation. When seeking the optimal rule order, the problem escalates to $\mathcal{NP}$-hard[14].

Chapter 3: Current Firewall Approaches

While the methods described in the previous chapter relate to theoretical models of firewall design and optimization, it is also important to include examples of currently available firewall technologies which utilize those models. Current firewall approaches include designs by both commercial entities and open-source projects. They can be implemented as software applications or as dedicated hardware appliances. This chapter will detail some of the options available for current firewall approaches on both single machine and parallel architectures.

3.1 Single Firewall Architectures

Traditional firewall designs involve a single machine to enforce a security policy. This machine can either: 1. require a resident operating system to run the firewall as an application; or 2. run the design on hardware as a custom designed/programmable circuit. This section will introduce some examples and give an overview of their advantages and disadvantages.

3.1.1 Software Firewalls

Generally speaking, the term software firewalls might imply a scope of only user-space programs like Zone Labs Zone Alarm Pro. In fact, software firewalls refer to any number of user-space or kernel level applications. While, due to performance issues, the user-space examples are usually only used on an individual or debugging level, kernel level solutions are commonly employed to protect many production level environments.

Some well known software firewalls include Sun's SunScreen, Linux's Netfilter (iptables with predecessors ipchains and ipfwadm), Symantec's Enterprise Firewall, BSD's IPFilter (ipf), OpenBSD's pf, and FreeBSD's ipfw. Most of these firewalls use list based policies. For instance, IPFilter employs the last match method by default, but can be made first match if every rule is given a 'quick' directive, while NetFilter and Solaris SunScreen both use the first match method and can not be changed. The arguments for the advantages and disadvantages of each model were described in 1.2.

The disadvantage of these solutions is the reliance on a resident Operating System (OS). Ultimately this limits speed capabilities to the bounds of the processor, bus, and network interface card. This also limits hardware compatibilities to within the OS's bounds. For instance, iptables can run on Linux compatible systems and Symantec's Enterprise Firewall operates on both Microsoft and Sun OS compatible hardware.

3.1.2 Hardware Firewalls

Firewalls are commonly used by organizations that wish to protect internal resources on a public networking medium. Given the lack of IT staff at many small to medium businesses, many rely on commercial vendors to provide security services. In the majority of cases this implies hardware firewalls, sold as network appliances.

A hardware firewall is a device that has hardware circuitry dedicated to processing network traffic streams. Some example commercial solutions are Celestix MSA Series, Check Point FireWall 1, Cisco PIX, CyberGuard, Juniper NetScreen, Phion NetFence, Sidewinder, SofaWare Technologies, and SonicWall. While it would be advantageous to know which of the policy models each of the market predominate variants use, it is impractical to seek this information usually due to reluctance, particularly with commercial variants, to release or make readily accessible what could be considered trade secrets. However, it is clear from example policies that Check Point's Firewall-1 and Cisco's PIX are both first match models.

Popular technologies for hardware packet filtering include: Field Programmable Gate Array (FPGA), Application Specific Integrated Circuit (ASIC), or Network Processing Unit (NPU) [22]. These devices seek to run packet filtering as close to line speed as possible. In essence they are finite state machines that once given an application, become dedicated to a singular task. In the case of an FPGA, applications

are run by a virtual circuit table to emulate hardware speeds [22]. However, the steep programming learning curve and relative high cost of FPGAs prevent in-house development for most organizations.

No matter how much a policy is optimized, and even whether the firewall is executed as software or hardware, policy decision processing speed will still be limited by the capabilities of a single device. The most common solution to this issue is to buy larger and more powerful single point of entry machines, a highly non-modular and cost ineffective approach. In situations where there is a surge of illegitimate traffic, such as Denial of Service (DoS) attacks, a non-scalable firewall solution like a single point of entry can quickly become overwhelmed. In such a scenario legitimate users may notice quickly declining network performance and lower QoS, and may loose all external network access if the firewall fails. If networks are to protect themselves in a cost effective manner while allowing for future growth, more scalable and dynamic solutions for firewall architecture must be devised.

3.2 Parallel Firewall Architectures

Parallelization can greatly enhance the performance of network firewalls by offering scalability to reduce processing loads. Parallelizing firewalls can be implemented by either dividing the traffic or the workload across an array firewall nodes.

There are two main ways to divide a single processor design with independent

discrete computations [6]. The first is to divide the data and the second is to divide the work. Using terminology developed for parallel computing, a design that distributes the data (packets) across the firewall nodes is considered *data parallel*[6].

Definition An array of m firewall nodes that enforces a stateless policy R operates in a data parallel fashion if:

1. Arriving traffic is divided (distributed) such that each packet is processed by only one firewall node.

2. Each firewall node i employs a local policy $R_i = R$, where $i = 1, \ldots, m$.

As seen in figure 3.1, a data parallel firewall system consists of multiple identical firewalls connected in parallel [3]. Each i^{th} firewall node, needs a local policy (rule set) R_i which will allow it to act independently. In other words, each local policy is a duplicate of the complete security policy. Arriving packets are then distributed across the firewall nodes such that only one firewall node processes any given packet [3]. Therefore different packets are processed in parallel and all packets are compared to the entire security policy. How the load-balancing algorithm distributes packets is vital to the system and is typically implemented as a high-speed switch in commercial products [18, 19].

Theorem 2 *An array of m firewall nodes arranged in a data parallel fashion will always maintain integrity of a policy R.*

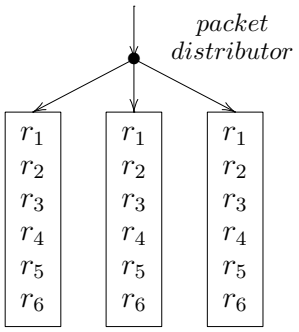


Figure 3.1: Data parallel, packets distributed across an array of equal firewall nodes.

Proof: Let A be the accept set of R. Since all firewall nodes employ the same local policy R_i, from the definition $R_i = R$ which implies $A_i = A$. From the definition of the data parallel system, a packet is processed by only one firewall node, therefore it is not possible for duplicates to be accepted by system. This shows equivalence between the two policies, therefore preserving integrity. ■

Although data parallel firewalls have been shown to achieve higher throughput than traditional firewalls [3] and have an innately redundant design, they suffer from major disadvantages. First, since the efficiency of scalability is limited to the system input load, the performance benefit over traditional firewall models is only evident under high traffic loads. Since the data parallel firewall system duplicates the policy, it does not reduce the processing time per packet, just the arrival rate into any firewall node.

Second, this design has difficulty enforcing QoS guarantees across networks. For example, since high priority traffic is processed on the same firewall nodes as low

priority traffic, the high priority traffic can encounter longer delays if it is queued behind low priority traffic that requires more processing. Under these circumstances, users notice poor network performance, which is a growing concern as more network applications require QoS assurances.

Third, since stateful inspection maintains tables of previously determined decisions, stateful inspection requires all traffic from a certain connection or exchange to traverse the same firewall node, which this design does not account for [3] and is difficult to perform at high speeds using the data parallel approach [23]. For instance, assume a parallel firewall system has state filtering enabled to verify traffic by corresponding TCP SYN and ACK flags [28, 33, 34]. If a single firewall node did not receive all packets in that connection stream, the ill matching flags would cause invalid state table entries to be made and intermittent packets could be lost. Therefore all packets belonging to a connection must traverse the same firewall node, or state information must be shared by all firewall nodes, specifications not listed in the original design. New parallel firewall architectures must solve these problems to meet future demands and increasing security threats.

Chapter 4: Function Parallel with Gate Design

This chapter introduces a new parallel firewall design, known as function parallel with gate node (FPG firewall). Once the initial design has been fully defined, several variations and options to improve performance are suggested. The chapter then presents simulation results of the FPG firewall design followed by a summary to compare the design to previously developed methods.

4.1 Overview of the Function Parallel with Gate Design

Figure 4.1 depicts a new function parallel design that consists of an array of firewall nodes and a gate node. The gate node is a device that acts as a control mechanism for the firewall system. In this new design, the security policy is distributed across all firewall nodes, while arriving packets are duplicated to all firewall nodes and the gate node.

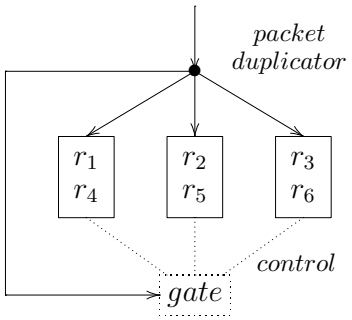


Figure 4.1: Function parallel with a gate node design, rules distributed across an array of firewall nodes.

Definition An array of m firewall nodes that enforces a policy R, with an accept set A, operates in a function parallel with gate fashion if:

1. Arriving traffic is duplicated to all firewall nodes and to an additional gate node.
2. Each firewall node i employs a local policy R_i, $i = 1, \ldots, m$, s.t. $\bigcup_{i=1}^{m} A_i = A$.
3. After a packet is processed by each firewall node i, the result of each R_i is sent to the gate node.
4. Only the gate node executes an action for any packet.

This design provides a new type of firewall node interaction. Instead of reducing the amount of input on any firewall node, this design reduces the processing time required for any input on any firewall node.

The operation of this function parallel system, as shown in figure 4.1, can be described as follows. Again, assume a first match policy. When a packet arrives at the function parallel with gate firewall, it is duplicated to each firewall node, plus

an additional node, known as the gate node. To ensure no more than one firewall node routes any packet into the system, only the gate node can execute an action on a packet. Thus, the gate node suspends the packet in a cache until it can determine the proper action to execute.

Each firewall node processes the duplicated packet using its local policy, including any state information. Note, the rule number may correspond to a state match if the packet belongs to an established connection. In a first match method this number should have a uniformly lower value since state rules are evaluated before policy rules [33]. Instead of a firewall node immediately executing actions on rule matches, it will signal the gate node with a 'vote.' A vote is inter-node communicable data that at minimum indicates the rule number of the match or if no match was found and some reference to what packet the vote is for. Therefore, *no-match* is a valid response and is required for the function parallel with gate firewall design. If all firewall nodes voted no-match, then the gate node would execute the policy default, and this implies that any local policy of a firewall node does not require an explicit default rule.

To accomplish a voting paradigm, the gate node establishes the packet cache and a vote array with the same number of elements as firewall nodes in the system. As each vote is received by the gate node it is recorded in the vote array at a pre-established corresponding element number. Once all votes have been recorded the gate

node searches the vote array for a vote that states the match method appropriate rule number. Thus, the first match method can be implemented by applying the lowest numbered matching rule. The gate node then executes the action corresponding to that rule number from the policy to the cached packet on the gate node. Again, if all votes report no-match, then the policy default is used. At a point in this process the gate node must inform the firewall nodes when to process the next packet. A control mechanism would also have to be in place to ensure the firewall nodes were not processing a different packet at that time – for instance, information in the message stating which packet was just processed. This can be accomplished with a heartbeat signal [3].

4.2 Maintaining Policy Integrity in Rule Distributions

As described in the previous section the gate node ensures only one decision is made for any packet. In this section, a proof will show that the design preserves integrity.

Theorem 3 *An array of m firewall nodes, arranged in a function parallel with gate fashion, that enforces the policy R will always maintain integrity if the local policies R_i are distributed to each node i, $i = 1, \ldots m$, with accept set A_i, such that $\bigcup_{i=1}^{m} A_i = A$.*

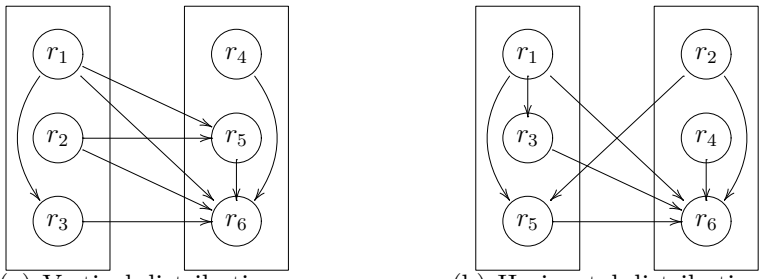
Proof: By the guideline given on rule distribution $\cup_{i=1}^{m} A_i = A$. This states that the system is divided into local policies, which, while they may intersect each other

($A_i \cap A_j \neq \emptyset$, $i \neq j$), the collective union will only accept any packet for which A also accepts. Intersections of local accept sets imply multiple accept votes are sent to the gate node. However, this is resolved by the gate node, since it chooses the final matching rule and is the only node which will accept a packet to be accepted. As such, the accept sets are in fact equal, which shows equivalence and proves that integrity is maintained. ■

4.3 Example Rule Distributions

In a function parallel with gate design, rule distribution guidelines which maintain integrity can be easily implemented by two simple rules. First, assign every rule to at least one firewall node. Second, a policy DAG edge should never point to a rule earlier within the firewall node, in order to ensure no shadowing exists on the local policy. This will prevent improper local policy ordering, which ensures that the accept sets and deny sets are equal. In this section, two examples of rule distribution techniques that obey the presented guidelines are presented for illustrative purposes.

Using the guidelines given in theorem 3, two example rule distributions are depicted in figures 4.2(a) and 4.2(b). The distribution given in figure 4.2(a) will be referred to as *vertical*. This can be achieved by sequentially distributing approximately n/m rules per firewall node's local policy. In this example, the first $n\%m$ firewall nodes have $n/m + 1$ rules and each firewall node afterwards has n/m rules.

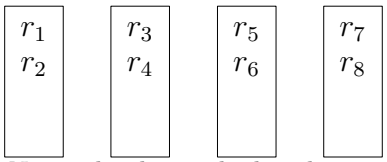


(a) Vertical distribution. (b) Horizontal distribution.

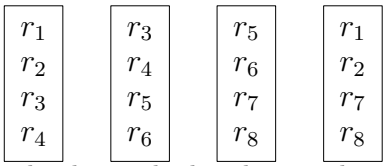
Figure 4.2: Example rule distributions on two firewall nodes using the policy given in table 2.1. The edge between rule r_3 and r_5 can be ignored since it is a same-action precedence edge.

The second distribution, given in figure 4.2(b), will be referred to as *horizontal*. A simple algorithm provides this by sequentially distributing each rule round robin into the local policies. In both cases the six rules are present and all policy DAG edges are downward; thus integrity is maintained.

Sequentially incrementing through each rule guarantees that every rule will be distributed into any local policy. In an original policy without shadowed rules, the sequential incrementation of rule distribution also assures that higher numbered rules will always be placed in local policies after lower numbered rules, which preserves all downward DAG edges and prevents the introduction of shadowing. Again these are only examples, but they provide legitimate methods on which to build further optimization features, such as redundancy.



(a) Non-redundant rule distribution.



(b) Redundant rule distribution that either overloads last firewall node or uses conditional processing.

Figure 4.3: Non-redundant and redundant function parallel designs, each consisting of four firewall nodes and an eight rule policy. No precedence edges are shown.

4.4 Redundancy

The proposed function parallel with gate firewall design shares a disadvantage with traditional firewall approaches in its inability to withstand single firewall node failure. In the FPG firewall design, if a firewall node fails, it's local policy would no longer be enforced, and integrity would be lost. A common solution for redundancy in the traditional firewall approach is to duplicate the entire system as a standby backup. However, this solution can be cost prohibitive, not efficient, and difficult to manage. This section aims to make allowances for firewall node failure by providing the system with redundancy through a simple modification to the example rule distribution methods.

It is possible that by duplicating local policies across the firewall nodes (copying a local policy to a neighbor) redundancy is provided without requiring additional firewall nodes. As long as the local policy copy is inserted such that no new upward DAGs are created (shadowing is not introduced), then integrity is preserved.

As an example, figure 4.3 shows two systems, each with four firewall nodes and

a vertical distribution of eight rules. Given the contiguous nature of vertically distributed rules we can duplicate each firewall node's local policy R_j by appending it to any R_i where $i < j$, in this example $i = j - 1$. This creates two situations, the first is that in a system of m firewall nodes, R_m does not have any rules appended. The second, which is more important, is that this does not allow for duplicating R_1. However, this could be accomplished by adding an additional firewall node. While this is better than duplicating the entire system, the goal is to provide redundancy without requiring extra firewall nodes. To resolve both problems, if R_1 is prepended to R_m, as in figure 4.3(b), then there is no need for an additional firewall node. No duplicates enter the system since the gate node still has the final decision power and will only choose the lowest rule number (using the first match method). If the lowest results are two identical rule numbers, then the gate node is free to choose either one. Since any duplicated rules are inserted in respect to rule order, no new upward policy DAGs are introduced and integrity is preserved.

Redundancy to one neighbor allows up to half the firewall nodes to fail before integrity is broken, as long as no two failed firewall nodes are contiguous. It should also be noted that, with redundancy, it would be possible to duplicate multiple firewall nodes onto others, which would proportionally increase the required number of firewall node failures before integrity is lost.

As an additional performance advancement, the firewall nodes could be interconnected [3] to determine if the redundant rules should be processed [12]. With an inter-node communication channel, it would be a simple addition to the method to allow each firewall node to not process the duplicated rules unless the original firewall node fails. For example, each firewall node could be made responsible for monitoring any other firewall node which it backs up. Upon recognizing a failure of any firewall nodes for which it is responsible, it first adds the duplicate rules into it's local policy then signals the gate node about the event. The gate node then marks the element in the vote array which is associated with the failed firewall node. Any new packet that arrives into the system will not require a vote from the failed firewall node to determine the associated action. This is extremely advantageous since with this dynamic insertion method, the gate node has to only wait, on average, the average processing time of one local policy instead of two. Other processes of avoiding unnecessary rule comparisons can also be used in the function parallel with gate design, such as short-circuit evaluation.

4.5 Short-Circuit Policy Evaluation

In the process described up to now, the gate node must receive votes from all firewall nodes before a decision can be made. This means that the gate node will always have to wait as long as it takes the slowest firewall node to send a vote. If a policy

distribution method were developed that resulted in highly uneven local policy sizes, this would play a significant role in the speed advantages of the system. This section introduces a process, known as short-circuit policy evaluation, to mitigate this complication.

An example implementation would use the same inter-node communication channel as the votes. As each vote arrives at the gate node it checks the information previously added for that packet. If it is possible to determine that a decision can be made, then the heart beat signal described earlier could be used to signal the firewall nodes to stop processing the current packet. Short-circuit evaluation requires the gate node to know how the rules are distributed as well as the dependencies, information which is already present in the policy DAG. For example, consider the rule distribution given in figure 4.2(a). Assume a packet matches r_4 in the second firewall node and this result is reported to the gate node. Using the policy DAG, the gate node can determine this is the first match since it is impossible for a packet that matches r_4 to also match any rules that precede it in the original policy (rules r_1, r_2, or r_3). Improvements through this method can demonstrated mathematically as well.

Let $t_i(d)$ denote the time for the firewall node i to decide on a vote for the packet d in the set of firewall nodes F, $i = 1, \ldots, |F|$. Also, let Z be the set of all firewall nodes, and X be the set of all firewall nodes that contain a rule that d matches. A

system without short-circuit evaluation requires a vote from all firewall nodes in Z to make a decision, while a system with short-circuit evaluation only requires the firewall nodes in X. Thus the maximum time for all firewall nodes to vote on the packet d without short-circuit evaluation is $v_r = max\{t_i(d)\}, \forall i \in Z$, and in a system with short-circuit evaluation the maximum time is $v_s = max\{t_i(d)\}, \forall i \in X$. While the packet d may match a rule on all firewall nodes, it is possible for this to not be the case, since $X \subseteq Z$. Therefore if $\bar{X}$ contains a firewall node j such that $t_j(d) > v_s$ then $v_s < v_r$. In other words, in such a case there would be a speed advantage in using the short-circuit method.

This process could be further improved using a process called pipelining. In pipelining, the information currently sent over the heartbeat signal is moved into a separate message, and the firewall nodes are not required to start new packet processing synchronously.

4.6 Pipelining

The performance of the function parallel with gate design can also be improved by allowing the firewall nodes to asynchronously process or *pipeline* packets [6]. This section will give an example of pipelining and describe the additional requirements on the FPG firewall design.

Pipelining is the method that occurs when, after a firewall node submits a vote

to the gate node, the firewall node does not have to wait for a heart beat signal from the gate node in order to begin processing the next packet. This process requires particular information to be exchanged between the gate node and the firewall nodes.

The primary difference from earlier designs is that the gate node must be capable of concurrently tracking multiple packets in the vote array. This will introduce additional overhead that will increase over time as the firewall nodes will be processing at different rates. Thus, there must an occasional throttling message sent to the firewall nodes from the gate node which will allow the slower firewall nodes to catch up, reducing the number of outstanding votes and allowing the data structure that contains the multiple vote arrays to shrink.

This method can be further improved if short circuit signal evaluation is in use. For this, the gate node must communicate which packets have been processed by it. This implies that the firewall nodes must now keep a reference of packets, or be able to determine a reference of packets, that it has not voted on yet and that the gate node says have already been decided on. When a firewall node finds one of these packets in it's queue, the firewall node can safely ignore the packet and move on to the next packet.

Since pipelining does require the gate node to keep track of multiple packets simultaneously, this could vastly complicate the overhead involved in handling decisions.

However, this process allows the firewall nodes to work without hindering the performance of each other and prevents starvation while waiting on the gate, thus increasing the overall system performance in terms of work efficiency.

4.7 Summary

As described, our function parallel with gate design has several significant advantages over traditional and data parallel firewalls [12]. First, our FPG firewall design results in faster processing since every firewall node utilizes a smaller local policy to find a match for a single packet.

Second, unlike data parallel firewall designs, FPG firewall designs can maintain state information about existing connections. Maintaining state can be viewed as the addition of a new rule corresponding to a requested connection [33]. Unlike data parallel designs, a new rule can be placed in any firewall node since a packet will be processed by every firewall node (as long as local policy integrity is maintained).

Third, any comprehensive policy without shadowing can be distributed across the firewall nodes and used without any modifications. As long as the gate node knows how to choose between multiple matches (i.e. which match method to use), then the policy integrity is maintained.

A disadvantage of function parallel systems is a possible limitation on scalability, since the system cannot have more firewall nodes than rules. However, given that the

size of most firewall policies range in the thousands of rules [30], this scalability limit is not an important concern.

Another disadvantage is the additional delay associated with the gate node processing and the difficulty of implementing the gate design on legacy systems. Since this design relies on the gate node for final decisions, the performance gains of the function parallel with gate firewall design are dependent on the speed of the gate node managing and interpreting the votes. However, with advanced rule distribution techniques it is possible to remove the necessity of the gate node, as discussed in the next chapter.

Chapter 5: Function Parallel Design

In the previous chapter a novel function parallel firewall design was introduced. It relied on a gate node to efficiently decide on results from an array of firewall nodes. This chapter introduces a modification to the design which allows it to function with no gate while still providing stateful filtering and strict QoS requirements.

5.1 Overview of the Function Parallel with no Gate Design

From the previous chapter a function parallel with a gate firewall (FPG firewall) is one which duplicates all traffic to all firewall nodes, and distributes local policies which collectively are equivalent (not necessarily equal) to the original policy. In this configuration, it has been proven through theorem 3 that the integrity of the system is ensured by using the gate. However, if the firewall nodes could be established to operate independently (with no gate node) then the design would become compatible with all firewall devices. A gateless design will referred to as simply 'function parallel firewall' (FP firewall).

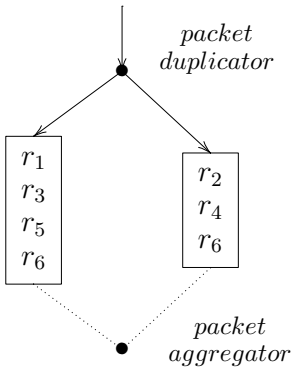


Figure 5.1: Function parallel without a control gate, rules distributed across an array of firewall nodes.

Definition An array of m firewall nodes that enforces a policy R operates in a function parallel fashion (does not require a gate node) if:

1. Arriving traffic is duplicated to all firewall nodes.

2. Each firewall node employs a local policy R_i where $i = 1, \ldots m$, s.t. $\bigcup_{i=1}^{m} A_i = A$ and $\bigcap_{i=1}^{m} A_i = \emptyset$.

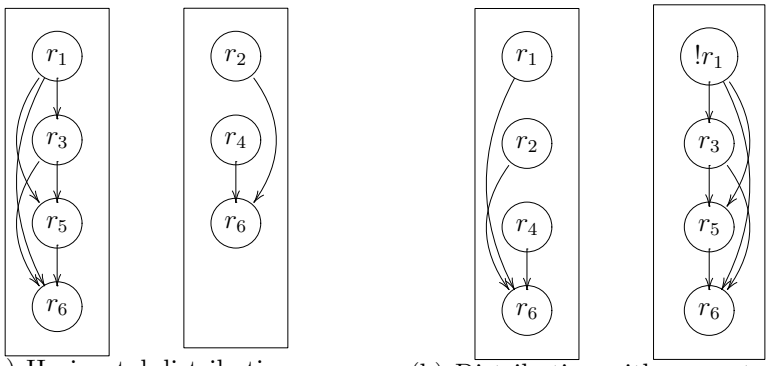
As an overview of the process, when a packet enters the system it is duplicated to all firewall nodes. Each firewall node then processes the packet according to the local policy. The firewall nodes do not need to synchronize with each other, as was done in the function parallel with gate design, allowing for a highly efficient form of pipelining [6]. However, in order to accomplish this, important rule distribution constraints must be followed which are described in the next section.

5.2 Maintaining Policy Integrity in Rule Distributions

Since every firewall node does not verify its local policy match against other firewall nodes' results, each firewall node must be able to act independently. More specifically, in order to avoid duplicates or failures of integrity, no firewall node should accept a packet that any other firewall node accepts. This section will define a guideline that allows independent policy distribution in a gateless environment.

Theorem 4 *An array of m firewall nodes, arranged in the above function parallel fashion, that enforces the policy R will always maintain integrity as long as local policies R_j are distributed to each node $j = 1, \ldots, m$, with accept set A_j, such that $\bigcup_{j=1}^{m} A_j = A$ and $\bigcap_{j=1}^{m} A_j = \emptyset$.*

Proof: By the guideline given on rule distribution is $\cup_{j=1}^{m} A_j = A$ and $\cap_{j=1}^{m} A_j = \emptyset$. This states that the system is divided into local policies, which have accept sets that will not intersect each other ($A_i \cap A_j = \emptyset$, $i \neq j$), and the collective union will only accept any packet for which A also accepts. The prevention of intersections of local accept sets implies no duplicate packets can enter the system. As such, the accept sets are equal, which shows equivalence and proves that integrity is maintained. ■



(a) Horizontal distribution. (b) Distribution with a negated r_1.

Figure 5.2: Example rule distribution on two firewall nodes using the policy given in table 2.1. The edge between rule r_3 and r_5 cannot be ignored since the gate machine has been removed.

5.3 Example Rule Distributions

From the earlier section it is evident that in order to maintain integrity, local policies must be independent (no accept set intersections). Definitive methods for converting all policies into independent sets are left as an area of future research, and this thesis will concentrate on only a few examples. Independent set creation can be most clearly illustrated with a default deny policy example.

In a default deny policy, accepted traffic must be stated explicitly. Thus when segmented across local policies, firewall nodes will not duplicate traffic if they all have a policy default of deny. If the policy contains only accept rules, and there are no precedence edges, then the policy could be evenly distributed using one of the methods described in section 4.3 and still maintain integrity. This case could happen when for each rule at least one of the tuples is fully defined, which is common for

policies. For instance, it could denote an organization which has a rule for each server and service pair they wish to allow access to from outside their protected network.

If a default deny policy contained precedence edges, where the source vertices all had an action of deny, then it is possible to duplicate these rules into any local policy of their dependents. This would preserve non-intersecting accept sets since introducing deny action rules can only decrease the accept set of any local policy. Thus while the deny sets may intersect, $\cap_{j=1}^{m} A_j = \emptyset$ is still true.

Given a default deny policy, which is now known to allow intersecting deny sets, it is possible to account for precedence edges from vertices, on other firewall nodes, which have an action of accept. If the action of the preceding accept rule is changed to deny (for brevity this will be referred to as *negating* the rule), it can be duplicated to other local policies as long as the rule number order is respected. This would allow reducing an accept set for a local policy by those packets which are matched by the negated rule.

Consider the policy in table 2.1 on a two node system. The default deny is put into each local policy. The independent rules r_1, r_2, and r_4 can be distributed in any manner. As an example, two possibilities that preserve integrity will be presented.

The first has rules r_3 and r_5 appearing in the same local policy as rule r_1, which requires no special cases and is done using the horizontal distribution method in

section 4.3. The final example has rule r_1 separate from rules r_3 and r_5, and the two coexist with a preceding negated r_1 on the same firewall node. Both of these examples illustrate integrity maintenance using theorem 4.

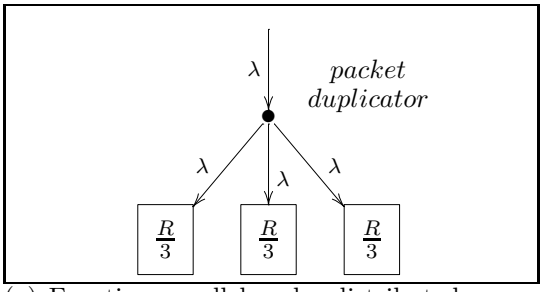
5.4 A Simple Theoretical Model Evaluation

Using the valid assumption that arrivals and service times are exponentially distributed [20], a function parallel firewall system can be considered an open network of M/M/1 queues (Jackson network) [4, 26]. Probabilities, which are given from the policy profile (hit ratio) described in section 2.4, can be assigned to each link to indicate the likelihood of moving to the next node. The average end-to-end delay for q cascading firewall nodes (traversal path) can be computed as

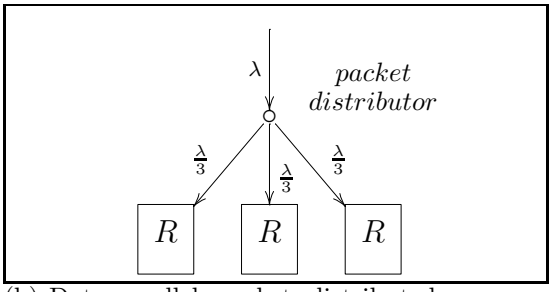
$$E(T) = \sum_{i=1}^{q} \frac{1}{\mu_i - \lambda_i}$$

where $1/\mu_i$ is the service time (processing and transmission) and λ_i is the arrival rate to node i. As a result, we have a theoretical model for the average delay across a firewall system. Consider the data parallel and function parallel firewall designs given in figure 5.3. Assume each system consists of m firewall nodes and implements the same n rule security policy. Let the total arrival rate to each system be λ packets per unit time and each node performs x rules per unit time.

For the data parallel firewall, traffic arrives at the packet distributor, which evenly



(a) Function parallel: rules distributed across an array of firewall nodes



(b) Data parallel: packets distributed across an array of equal firewall nodes.

Figure 5.3: Parallel network firewall designs consisting of 3 firewall nodes and a policy R, where λ is the arrival rate into a firewall node.

distributes the traffic. As a result the arrival rate to each firewall node is $\frac{\lambda}{m}$. The service rate is $\frac{x}{n}$, since each node implements the complete policy. The end-to-end delay across any firewall node in the data parallel firewall is

$$E_d(T) = \frac{1}{\frac{x}{n} - \frac{\lambda}{m}}$$

In contrast, rules are distributed across each firewall node in the function parallel firewall. Furthermore, all traffic arriving to the system is forwarded to each firewall node. If we assume the rules in each firewall node are independent (no policy DAG edges spanning firewall nodes), then the end-to-end delay across any firewall node in the function parallel firewall is

$$E_f(T) = \frac{1}{\frac{m \cdot x}{n} - \lambda}$$

The reduction in delay compared to a data parallel firewall system is

$$s_{f,p} = \frac{E_f(T)}{E_d(T)} = \frac{\frac{1}{\frac{m \cdot x}{n} - \lambda}}{\frac{1}{\frac{x}{n} - \frac{\lambda}{m}}} = \frac{\frac{x}{n} - \frac{\lambda}{m}}{\frac{m \cdot x}{n} - \lambda} = \frac{1}{m}$$

Therefore, the function parallel system has the potential to be m times faster than a data parallel system. However, the scalability of the function parallel system is dependent on the policy and hit ratio.

5.5 Summary

Our function parallel design has several significant advantages over traditional, data parallel, and our function parallel with gate firewall designs. First, our FP firewall design results in faster processing since every firewall node utilizes a smaller local policy to find a match for a single packet. As was demonstrated theoretically, reducing the processing time, instead of the arrival rate (the data parallel paradigm), yields better performance in terms of delay.

Second, unlike the data parallel design, the function parallel design can maintain state information about existing connections. This was also seen in the function parallel with a gate design. Unlike the data parallel design, a new rule can be placed in any firewall node since a packet will be processed by every firewall node, and only the appropriate firewall node will act accordingly.

One disadvantage of the system (in terms of preprocessing) is that the rule distribution has more constraints and could conceivably be more complex than that of traditional single firewall models. Optimization of these techniques are left open for future development.

Another disadvantage of the system is a possible limitation on scalability, since the system cannot have more firewall nodes than rules. However, given the size of most firewall policies range in the thousands of rules [30], the scalability limit is not an important concern.

Chapter 6: Parallel Firewall Simulation Results

While theoretical models provide insight into the performance of the different designs, they can only predict the average delay. If more empirical measurements are required, simulations can be used to measure both the average and maximum delay of the various designs.

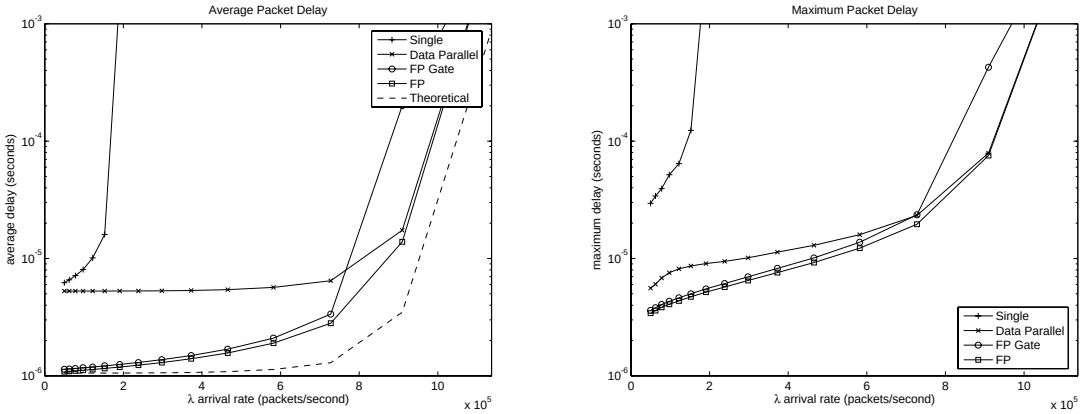
6.1 Overview of the Simulation Design Setup

A discrete event simulator was used to measure the performance of a traditional single firewall, the data parallel firewall, the function parallel with gate firewall (no redundancy, no short-circuit evaluation, no pipelining), and the function parallel with no gate designs. Each firewall node was simulated to process 6×10^7 rules per second, which, as determined in our labs, is comparable to current technology. For the experiments the parallel designs always consisted of five firewall nodes except for the increasing firewall nodes scenarios. Also, all experiments (except increasing number of rules) used a policy size of 1024 rules. A gate delay was added to the function parallel with gate design equivalent to processing three additional rules. However, no additional delay was added to the data parallel system for packet distribution. There-

fore, the delays observed for data parallel are better than what should be expected. Furthermore, in the data parallel simulations every packet was randomly distributed to a firewall node using a uniform distribution to closer mimic real connection oriented traffic.

Packet lengths were uniformly distributed between 40 and 1500 bytes, while all legal IP addresses were equally probable. The inter-arrival times were scheduled according to a Poisson distribution. Local policies were generated such that the rule match probability was given by a Zipf distribution, which is commensurate with actual firewall policies [13, 21]. Each policy generated for the simulations consisted of explicit accepts, a default deny, and no precedence edges. Rules were distributed in a horizontal fashion for both function parallel with and with no gate designs in order to maintain policy integrity, as described in section 4.3.

Three sets of experiments were performed to determine the performance effect of increasing arrival rates, increasing policy size, and increasing the number of firewall nodes. For each experiment 1000 repetitions of 100 seconds of traffic were performed, then the average and maximum packet delays were recorded. The average results were also compared against the theoretical performance described in [12] and section 5.4, which indicates the best average performance any function parallel firewall can achieve is $1/m$ the data parallel performance, where m is the number of nodes.

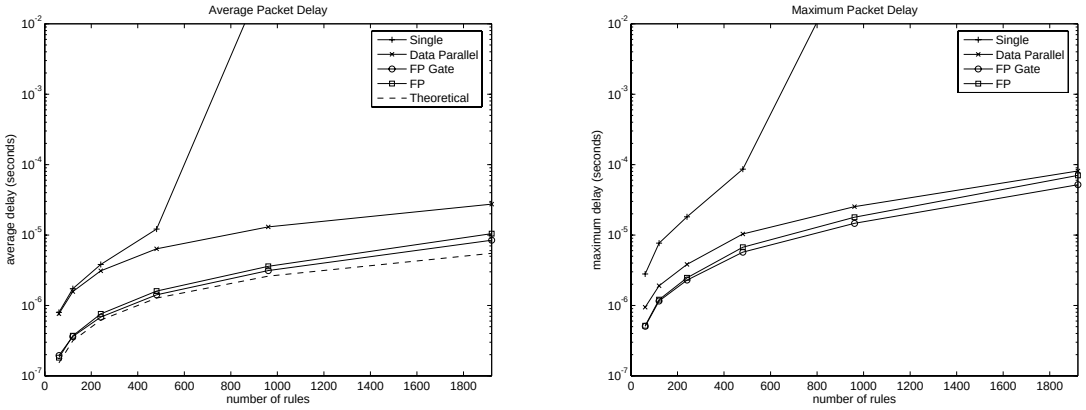


(a) Average Delay. (b) Maximum Delay.

Figure 6.1: Average and maximum packet delay as the arrival rate increases for a traditional, five node data parallel system, five node function parallel with gate system, and five node function parallel with no gate system.

6.2 Delay as a function of Arrival Rate

This set of experiments showed the capability of all systems under increasing traffic loads. The arrival rate was varied from 300 Mbps to 6 Gbps in order to measure the impact of high data rates. As is seen in the figure 6.1(a), all parallel systems outperformed the single firewall as the arrival rate increased, since the single firewall lacked the scalability that parallelism provides. However, as the arrival rate increased, gate delay became more prominent on the function parallel with gate design. As expected, the function parallel with no gate design closer approximated the theoretical limits than any other design measured, due to the reduced processing time and independence of firewall nodes. Note that the maximum arrival rates, figure 6.1(b), mimicked the behavior of the averages, showing a low deviation of results.



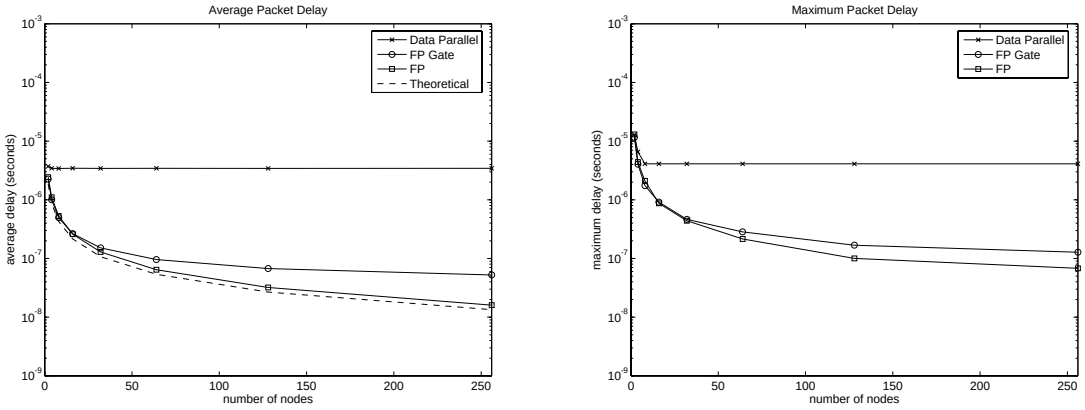
(a) Average Delay.

(b) Maximum Delay.

Figure 6.2: Average and maximum packet delay as policy Sizes increase for a traditional, five node data parallel system, five node function parallel with gate system, and five node function parallel with no gate system.

6.3 Delay as a function of Number of Rules

These experiments measured the impact of larger policies on the various designs. Arrival rate was established at 650 Mbps. The number of rules went from 2 to 2048. In all cases the parallel systems performed better than the single system. Average results are in figure 6.2(a) and maximum results in figure 6.2(b). It is also shown that the two function parallel designs resulted in lower delay than the data parallel system. However, contrary to what was expected function parallel with gate performed marginally better than function parallel with no gate, primarily because the gate delay was constant. As the rule processing delay increased and the gate delay remained the same, the gate delay became negligible.



(a) Average Delay. (b) Maximum Delay.

Figure 6.3: Average and maximum packet delay as the number of nodes increase for a five node data parallel system, five node function parallel with gate system, and five node function parallel with no gate system.

6.4 Delay as a function of Number of Nodes

This set of simulations show delay as a function of the number of firewall nodes in the system, thus only parallel designs were tested. The number of firewall nodes was increased from 2 to 256, and the results are in figures 6.3(a) and 6.3(b). Data parallel remained relatively the same for all test scenarios, illustrating that reducing load does not effect processing delay. However, the lower number of firewall nodes showed slightly higher delay due to increased loads on each firewall. The difference of delay between the function parallel with a gate and function parallel with no gate was the average additional delay associated with gate decisions. This difference proportionally increased since the additional gate delay was constant. Note that under these circumstances the function parallel with no gate performs at near theoretical averages.

6.5 Summary

As described in [12] and seen in figure 6.1(a), all parallel designs performed better than a traditional single firewall under increasing traffic loads and policy sizes. In addition, both function parallel designs consistently performed better than the data parallel design. The function parallel with no gate design yielded approximately a 66% reduction in the delay compared to the data parallel design and approximately a 13% reduction in the delay compared to function parallel with a gate design. However, the delays of the function parallel designs were higher than the theoretical values. In the case of function parallel with a gate, this is due to the additional gate delay that was equivalent to processing three additional firewall rules. As seen in figure 6.1(a), the gate delay became more significant as the arrival rate increased.

The impact of the gate node is also evident in figure 6.3(a), which showed the delay as the number of firewall nodes increased. The processing delay becomes smaller as more nodes are added, yet the gate delay remains constant, thus more prominent in the total delay experienced. As additional firewall nodes are added the performance difference between the function parallel with gate design and the theoretical average becomes larger. In all of the tests it should be stated that the results for function parallel with no gate are not best case. If additional rules were added at the top of each local policy to drop other nodes' accept sets then those packets would not have

to traverse the entire policy, reducing the average number of comparisons per packet and thus processing time.

Chapter 7: Function Parallel Firewall Implementation

The previous sections defined theoretical models used for simulation of the function parallel system (with no gate). While this is convenient for research comparisons, it is also important to verify the possibility of constructing the firewall system. This chapter will demonstrate how to implement a function parallel firewall system. Furthermore, objectives will maintain affordability, use only off-the-shelf components, and have high-speed capabilities.

7.1 Function Parallel Firewall System Components

As described in section 5.1, and depicted in figure 7.1, a function parallel system consists of an array of firewall nodes. Packets are duplicated to all firewall nodes as they enter the system. Rules must be distributed across the system such that they represent an accept set equal to the original accept set and no local policy's accept set can intersect another local policy's accept set, therefore a gate is not required. The accepted packets are then combined into one stream to reach the destination. A control plane is also necessary to allow general system management.

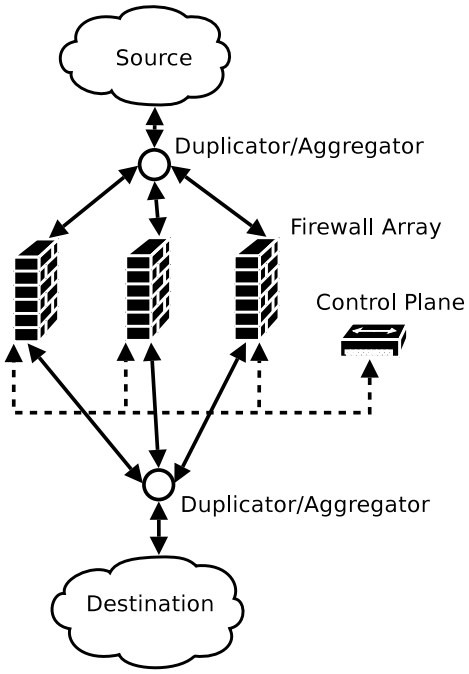


Figure 7.1: Figure of a function parallel firewall in bi-directional configuration.

7.1.1 Firewall Nodes

The firewall nodes consist of any computer with at least one network interface card for each network input and control plane. To consider affordability and off-the-shelf components, this design implements the firewall nodes as PCs running the Linux operating system with a kernel that supports iptables.

It should be mentioned that iptables was chosen for an advantageous feature of conditional rule processing. By default there are built-in sets of rules called chains separated by whether they handle traffic bound for processes listening on local input devices, intended for routing through the system to another network, or outbound and sourced from a local process. In addition iptables has support for user defined

chains. The user defined chains can be called if a packet matches a rule in a chain with a 'jump to' flag. Thus, although the input list is linear and the method is still first match, this allows for tree style traversal of the policy dependent on common precedence edges.

7.1.2 Packet Duplicator

One part of the definition of a function parallel system is to duplicate all arriving packets to every firewall node. This is required on all links which will input traffic into the system. In Ethernet networks less than 1 Gbps this is easily accomplished with a network hub since any frames arriving on a hub port are duplicated to all other hub ports. However, in high speed networks (at and above 1 Gbps) network hub technology is not available. To achieve duplication in high speed networks to the firewall nodes, the only available devices are known as network taps. Taps are devices used for duplicating (mirroring) network traffic, commonly used in intrusion detection systems that require network monitoring.

7.1.3 Control Plane

In a single firewall system, secure user interaction can be provided through the requirement of physical presence. To create a secure method of management, communication to the array of firewall nodes can be quarantined on a separate control plane. In the

most simple form this involves a separate subnet which all firewall nodes exist on. The control of the system can then be done using standard tools, such as ssh.

7.2 Component Integration

Combining these components into a usable configuration can be simplified into two network models. The first uses only one packet duplicator and can only provide security for traffic from one source. The second topology allows two networks to communicate bi-directionally through the system.

7.2.1 One Directional System

This design only considers packets moving in one direction; for example, only packets traveling from source to destination. The array of firewall nodes use the same IP address and MAC address and all will bring up the incoming (source) network interface card in promiscuous mode. All but one firewall node will disable ARP, allowing current networking equipment outside of the system to interact with no changes. The firewall node which does respond to ARP requests can also be allowed to respond to ICMP ping requests in order to make the system further compatible. For the outgoing (destination) network interface cards, any IP or MAC can be used as long as it allows communication with the destination. Given this, a standard networking switch can be used to combine the accepted traffic.

7.2.2 Bi-directional System

This design, figure 7.1, considers packets moving in both directions; for example, packets traveling from source to destination and then back to source. Establish the source network interface cards in the same manner as the one directional system. The destination network interface cards are setup using a different single IP and MAC. All NICs are brought up in promiscuous mode, and only one destination NIC is permitted to respond to ARP requests. Since this topology requires that both network duplicate packets, the tap device must be capable of combining traffic which is sent back through it onto the link with the destination network.

Chapter 8: Conclusions and Future Work

It is important that a firewall acts transparently to legitimate users, with little or no effect on the perceived network performance. This is especially true if traffic requires specific network Quality of Service (QoS), such as bounds on the packet delay, jitter, and throughput. The firewall should process the legitimate traffic quickly and efficiently. Unfortunately, the firewall can quickly become a bottleneck given increasing traffic loads and network speeds. Packets must be inspected and compared against complex policies and tables, which is a time consuming process. In addition, audit files must be updated with current connection information. As a result, the firewall and, more importantly, the network it protects can be quickly overwhelmed and are susceptible to Denial of Service (DoS) attacks. For these reasons, it is important to develop new firewall architectures that can support increasing network speeds and traffic loads, as well as minimize the impact of DoS attacks.

Previous research has proposed a new scalable firewall architecture for increasing network speeds and traffic loads, known as function parallel with a gate [12]. The parallel design consists of multiple firewall nodes and a gate node. Each firewall node implements a portion of the security policy. When a packet arrives to the system

it is processed by every firewall node simultaneously, which significantly reduces the processing time per packet. Since this design relies on the gate node for final decisions, the performance gains of the function parallel with gate firewall design are dependent on the speed of the gate node managing and interpreting the votes. Rule distribution across the firewall nodes must be done to maintain policy integrity, which ensures the parallel design and a traditional single firewall always reach the same decision for any packet. Rule distribution guidelines that maintain integrity were described in this thesis. The experimental performance showed that the proposed function parallel with no gate architecture can achieve a processing delay 66% lower than the previous data parallel firewall design and 13% less processing delay than the function parallel with a gate design. Furthermore unlike other designs, the proposed architecture can provide stateful inspections since a packet is processed by every firewall node. Therefore, the function parallel with gate architecture is a scalable solution that offers better performance and more capabilities than other designs.

Following this, a new design was proposed which allowed for the removal of the gate node while retaining integrity. This additional design divided work in a similar manner to the gate design, but each firewall node worked independently. This involved strict control over the policy distribution, but modeling showed a higher probability of approaching theoretical speedups.

The theoretical performance modeling shows that the proposed architecture is m times faster than previous parallel-firewall designs, where m is the number of firewall nodes. The performance increase was observed experimentally under realistic conditions using a simulation program which also showed speedup approaching the optimal increase.

While the function parallel firewall architecture is very promising, several open questions exists. For example, given the need for QoS in future networks, it may be possible to distribute rules such that traffic flows are isolated. In this case a certain type of traffic would be processed by a certain firewall node. Another open question is the optimization of local firewall node policies, including redundant policies, using the methods described in [13, 14]. However, optimization can only be done if policy integrity is maintained.

References

[1] Ehab Al-Shaer and Hazem Hamed. Firewall policy management advisor for anomaly detection and rule editing. In *Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management*, 2003.

[2] Ehab Al-Shaer and Hazem Hamed. Modeling and management of firewall policies. *IEEE Transactions on Network and Service Management*, 1(1), 2004.

[3] Carsten Benecke. A parallel packet screen for high speed networks. In *Proceedings of the 15th Annual Computer Security Applications Conference*, 1999.

[4] Gunter Bolch, Stefan Greiner, Hermann de Meer, and Kishor S. Trivedi. *Queueing Networks and Markov Chains*. John Wiley and Sons, Inc., 1998.

[5] CERT. Cert/cc statistics 1988-2005. `http://www.cert.org/stats/cert_stats.html`.

[6] David E. Culler and Jaswinder Pal Singh. *Parallel Computer Architecture: A Hardware/Software Approach*. Morgan Kaufman, 1999.

[7] Dan Decasper, Zubin Dittia, Guru Parulkar, and Bernard Plattner. Router plugins: A software architecture for next-generation routers. *IEEE/ACM Transactions on Networking*, 8(1), February 2000.

[8] Mikael Degermark, Andrej Brodnik, Svante Carlsson, and Stephen Pink. Small forwarding tables for fast routing lookups. In *Proceedings of ACM SIGCOMM*, pages 4 – 13, 1997.

[9] Uwe Ellermann and Carsten Benecke. Firewalls for ATM networks. In *Proceedings of INFOSEC'COM*, 1998.

[10] David Eppstein and S. Muthukrishnan. Internet packet filter management and Rectangle geometry. In *Proceedings of the Symposium on Discrete Algorithms*, pages 827–835, 2001.

[11] Anja Feldmann and S. Muthukrishnan. Tradeoffs for packet classification. In *Proceedings of the IEEE INFOCOM*, pages 397 – 413, 2000.

[12] Errin W. Fulp. Firewall architectures for high speed networks. Technical Report 20026, Wake Forest University Computer Science Department, 2002.

[13] Errin W. Fulp. Firewall policy models using ordered-sets and directed acyclical graphs. Technical report, Wake Forest University Computer Science Department, 2004.

[14] Errin W. Fulp. Optimization of network firewall policies using directed acyclical graphs. In *Proceedings of the IEEE Internet Management Conference (IM'05)*, 2005.

[15] Errin W. Fulp and Stephen J. Tarsa. Network firewall policy tries. Technical Report 20049, Wake Forest University Computer Science Department, 2004.

[16] Errin W. Fulp and Stephen J. Tarsa. Network firewall policy representation using ordered sets and tries. In *Proceedings of the IEEE International Symposium on Computer Communications (ISCC'05)*, 2005.

[17] Rainer Funke, Andreas Grote, and Hans-Ulrich Heiss. Performance evaluation of firewalls in gigabit-networks. In *Proceedings of the Symposium on Performance Evaluation of Computer and Telecommunication Systems*, 1999.

[18] X. Gan, T. Schroeder, S. Goddard, and B. Ramamurthy. LSMAC vs. SLNAT: scalable cluster-based web servers. *Journal of Networks, Software Tools, and Applications*, 3(3):175–185, 2000.

[19] Steve Goddard, Roger Kieckhafer, and Yuping Zhang. An unavailability analysis of firewall sandwich configurations. In *Proceedings of the 6th IEEE Symposium on High Assurance Systems Engineering*, 2001.

[20] Raj Jain. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling.* John Wiley & Sons, 1991.

[21] W. E. Leland, M. S. Taqqu, W. Willinger, and D. V. Wilson. On the self-similar nature of ethernet traffic. *IEEE Transactions on Networking*, 2:1 – 15, 1994.

[22] John Lockwood. Evolvable internet hardware platforms. *NASA/DoD Workshop on Evolvable Hardware*, pages 271–279, July 2001.

[23] Olivier Paul and Maryline Laurent. A full bandwidth ATM firewall. In *Proceedings of the 6th European Symposium on Research in Computer Security ESORICS'2000*, 2000.

[24] Lili Qui, George Varghese, and Subhash Suri. Fast firewall implementations for software and hardware-based routers. In *Proceedings of ACM SIGMETRICS*, June 2001.

[25] Venkatesh Prasad Ranganath and Daniel Andresen. A set-based approach to packet classification. In *Proceedings of the IASTED International Conference on Parallel and Distributed Computing and Systems*, pages 889–894, 2003.

[26] Mischa Schwartz. *Telecommunication Networks: Protocols, Modeling, and Analysis*. Addison-Wesley, 1987.

[27] Subhash Suri and George Varghese. Packet filtering in high speed networks. In *Proceedings of the Symposium on Discrete Algorithms*, pages 969 – 970, 1999.

[28] Andrew S Tanenbaum. *Computer Networks*. Prentice Hall, fourth edition, 2003.

[29] Priyank Warkhede, Subhash Suri, and George Varghese. Fast packet classification for Two-Dimensional Conflict-Free filters. In *Proceedings of IEEE INFOCOM*, pages 1434–1443, 2001.

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

[31] Jun Xu and Mukesh Singhal. Design and evaluation of a High-Performance ATM firewall switch and its applications. *IEEE Journal on Selected Areas in Communications*, 17(6):1190–1200, June 1999.

[32] Jun Xu and Mukesh Singhal. Design of a High-Performance ATM firewall. *ACM Transactions on Information and System Security*, 2(3):269–294, August 1999.

[33] Robert L. Ziegler. *Linux Firewalls*. New Riders, second edition, 2002.

[34] Elizabeth D. Zwicky, Simon Cooper, and D. Brent Chapman. *Building Internet Firewalls*. O'Reilly, 2000.

Vita

Ryan J. Farley

- 212 Radford Street Winston-Salem, NC 27106
  email: ryanfarley@ieee.org
  phone: (941) 284-0459

Education

- Master of Science, Computer Science
  Wake Forest University, Winston-Salem, NC
  Dec 2005
  Thesis: "Parallel Firewall Designs For High-Speed Networks"
  3.58 GPA

- Bachelor of Science, Computer Science
  Wake Forest University, Winston-Salem, NC
  Dec 2002
  3.33 GPA in Major

Publication

- Ryan J. Farley and Errin W. Fulp. "Effects of Processing Delay on Function-Parallel Firewalls." IASTED: PCDN Feb 2006

Patent

- "Function-Parallel Firewall," Errin W. Fulp and Ryan J. Farley. Patent Pending No. 60/638,436

Press

- Mary Beth Marklein. "Students Have 'Away' With Words." USA Today 29 March 2004: D7

- Laura Sessions Stepp. "Posting Their Lives, Moment By Moment." Washington Post 9 July 2004: C1,C2

Research and Experience

- Research Assistant
  Wake Forest University, Winston-Salem, NC
  Jan 2004 – Dec 2005
  Research sponsored by a grant from the U.S. Department of Energy. Investigated methods for optimizing firewalls and security policies in High Speed Networks. Developed theory and tested implementations of various distributed firewall architectures.

- Lead Software Engineer
  BuddyGopher, Winston-Salem, NC
  March 2003 – Aug 2004
  Internet startup used by over 12,000 people. Designed and implemented a user-friendly web-based frontend and a scalable AIM client, through reverse engineering, that would monitor and record away messages in realtime. Created and administrated security policies, backend database schema, DNS, and web server setup.

- Independent Researcher
  WFU Network Security Group, Winston-Salem, NC
  May 2002 – Dec 2002
  Developed intelligent QoS firewall system with open-source applications. Also designed hierarchical firewall system that lead to Master Degree thesis research.

Honors and Leadership

- Dean's List Spring 2002 and Fall 2002.
- Brian Piccolo Cancer Fund Achievement Award 1999-2002.
- Vice President of Upsilon Pi Epsilon Honor Society.
- Alumni Officer, Prudential Committee Member at Large, Pledge Educator, House Editor, and Webmaster of Alpha Sigma Phi General Men's Fraternity.
- Student Adviser to the WFU Computer Science Technology Acquisition Board.
- Member of ACM, IEEE, WFU Computer Science Club, WFU Network Security Group, Wake International Student Association, Wake Forest University Japanese Club, Waseda University International Club, AOPA, Diver's Alert Network.