NOTICE: This material is copyrighted, and I am required to inform you that its use is limited to permissions of the copyright holder of each particular manuscript. Please refer to the following list, sorted chronologically, for further information.

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

  The authors have transfered copyright in the Paper, transferring to IASTED and ACTA PRESS the exclusive right to publish, distribute, reproduce, or sell the Paper by printed, electronic or other means. the Paper cannot be re-published, distributed or sold ther than by IASTED and ACTA PRESS without the prior written permission of IASTED and ACTA PRESS. It is available for purchase at `http://www.actapress.com/Content_of_Proceeding.aspx?proceedingID=352`

- Errin W. Fulp and Ryan J. Farley. A Function-Parallel Architecture for High-Speed Firewalls. IEEE: ICC June 2006.

  ©2006 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.

- Ruishan Zhang, Xinyuan Wang, Ryan Farley, Xiaohui Yang, and Xuxian Jiang. On the Feasibility of Launching the Man-In-The-Middle Attacks on VoIP from Remote Attackers. ACM: ASIACCS March 2009.

  Copyright ©2009 by the Association for Computing Machinery, Inc. (ACM). Permission to make digital or hard copies of portions of this work for personal or classroom use is granted without fee provided that the copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page in print or the first screen in digital media. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted.
  To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a fee. Send written requests for republication to ACM Publications, Copyright & Permissions at the address above or fax +1 (212) 869-0481 or email permissions@acm.org.
  For other copying of articles that carry a code at the bottom of the first or last page, copying is permitted provided that the per-copy fee indicated in the code is paid through the Copyright Clearance Center, 222 Rosewood Drive, Danvers, MA 01923.

- Ruishan Zhang, Xinyuan Wang, Xiaohui Yang, Ryan Farley and Xuxian Jiang. An Empirical Investigation into the Security of Phone Features in SIP-based VoIP Systems. ISPEC April 2009.

  The copyright to the Contribution identied above is transferred to Springer-Verlag GmbH Berlin Heidelberg (hereinafter called Springer-Verlag). The copyright transfer covers the sole right to print, publish, distribute and sell throughout the world the said Contribution and parts thereof, including all revisions or versions and future editions thereof and in any medium, such as in its electronic form (ofine, online), as well as to translate, print, publish, distribute and sell the Contribution in any foreign languages and throughout the world (for U.S. government employees: to the extent transferable). The published article is available on Springers website `www.springerlink.com`.

- Ryan Farley and Xinyuan Wang. Roving Bugnet: Distributed Surveillance Threat and Mitigation. IFIP: SEC May 2009.

  See the previous copyright statement (Springer-Verlag).

A Function-Parallel Architecture for High-Speed Firewalls

Errin W. Fulp and Ryan J. Farley
Department of Computer Science
Wake Forest University, Winston-Salem, NC 27109-7311, USA
nsg.cs.wfu.edu

Abstract—**Firewalls enforce a security policy by inspecting and filtering traffic arriving or departing from a secure network. This is typically done by comparing an arriving packet to a set of rules and performing the matching rule action, which is accept or deny. Unfortunately packet inspections can impose significant delays on traffic due to the complexity and size of rule sets. Therefore, improving firewall performance is important, given the next generation of high-speed networks.**

This paper introduces a new firewall architecture that can perform packet inspections under increasing traffic loads, higher traffic speeds, and strict QoS requirements. The architecture consists of multiple firewalls configured in parallel that collectively enforce a security policy. Each firewall implements part of the policy and arriving packets are processed by all the firewalls simultaneously. Since multiple machines are used to process every packet, the proposed function-parallel system has lower delays (74% lower) and a higher throughput than other data-parallel (load-balancing) firewalls. These findings will be demonstrated empirically. Furthermore unlike data-parallel systems, the function-parallel design allows the stateful inspection of packets, which are critical to prevent certain types of network attacks.

I. Introduction

Network firewalls remain the forefront defense for most computer systems. Guided by a security policy, these devices provide access control, auditing, and traffic control [2], [22], [23]. As seen in figure 1(a), a security policy is a set of ordered rules that define the action to perform on matching packets. Given the packet and/or connection information, rules indicate the action to take place for each packet, such as discard, forward, or redirect. Security can be further enhanced with connection state information. 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) [23].

Traditional firewall implementations consist of a single dedicated machine, similar to a router, that sequentially applies the rule set to each arriving packet. However, packet filtering can represent a significantly higher processing load than routing decisions [16], [18], [23]. For example, a firewall that interconnects two 100 Mbps networks would have to process over 300,000 packets per second [22]. Successfully handling this high traffic becomes more difficult as rule sets become more complex [3], [15], [23]. Furthermore, firewalls must be capable of processing even more packets as interface speeds increase. In a high-speed environment (e.g. Gigabit Ethernet), a single firewall can easily become a bottleneck and is susceptible to DoS attacks [3], [5], [11], [13]. An attacker could simply inundate the firewall with traffic, delaying or preventing legitimate packets from being processed. Building a faster single firewall is possible [6], [16], [18], [19], [21]; however, the benefits are temporary (traffic loads and interface speeds are increasing); it is not scalable; it is a single point of failure; and it is generally not cost-effective for all installations.

A parallel firewall (also called a load-balancing firewall) is a scalable approach for increasing the speed of inspecting network traffic [3], [13], [15], [23]. As seen in figure 2(a), the system consists of multiple identical firewalls connected in parallel. Each machine, called a firewall-node, implements the complete security policy and arriving packets are distributed across the nodes such that only one node processes any given packet [3]. How the load-balancing algorithm distributes packets is vital to the system and typically implemented as a high-speed switch in commercial products [12], [13]. Although parallel firewalls achieve higher throughput than traditional firewalls [3] and have a redundant design, the performance benefit is only evident under high traffic loads. Furthermore, stateful inspection requires all traffic from a certain connection or exchange to traverse the same firewall-node, which is difficult to perform at high speeds [15].

This paper introduces a new scalable parallel firewall architecture designed for increasing network speeds and traffic loads. The design consists of multiple firewall nodes where each firewall node implements a portion of the security policy. Unlike the previous parallel design, when a packet arrives to the new architecture it is processed by every firewall node in parallel, thus the processing time required per packet is reduced. 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 are described in this paper. Simulation results will show the new architecture can achieve a 74% reduction in processing time as compared to other parallel-firewall designs. Furthermore unlike other designs, the proposed architecture can provide stateful inspections since a packet is processed by every firewall node. Therefore, the new parallel design is a scalable solution that offers consistently better performance and more capabilities than other designs.

The remainder of this paper is structured as follows. Section II reviews firewall policy models that are used for rule distribution in the proposed parallel system. Parallel firewall designs are described in section III, including the new design and rule distribution methods. Then section IV will demonstrate the experimental performance of the parallel design. Section V reviews the parallel firewall design and discusses some open questions.

II. Firewall Policy Models

Since maintaining policy integrity is a primary objective of any firewall design, this section will review firewall policies and describe a policy model that maintains this characteristic [9]. A firewall policy is an ordered set of firewall rules. In this paper, a rule r is modeled as an ordered tuple of sets, $r = (r[1], r[2], ..., r[k])$. Order is necessary among the tuples since comparing rules and packets requires the comparison of corresponding tuples. Each tuple $r[l]$ is a set that can be fully specified, given as a range, or contain wildcards `‘*’` in standard prefix format. For the Internet, security rules are commonly represented as a 5-tuple consisting of: protocol type, source IP address, source port number, destination IP address, and destination port number [22], [23]. Given this model, the ordered tuples can be supersets or subsets of each other, which forms the basis of precedence relationships. In addition to the prefixes, each filter rule has an action, which is to accept or deny. Similar to a rule, a packet (IP datagram) d can be viewed as an ordered k-tuple $d = (d[1], d[2], ..., d[k])$; however, ranges and wildcards are not possible for any packet tuple.

Using the previous rule definition, a standard security policy can be modeled as an ordered set (list) of n rules, denoted as $R = \{r_1, r_2, ..., r_n\}$. State can be viewed as a preliminary extension of the policy that contains a set of rules for established connections [22]. Note, the internal representation of the policy does not have to be a list [10], [16]. Starting with the first rule, a packet d is sequentially compared against each rule r_i until a match is found ($d \Rightarrow r_i$), then the associated action is performed. This is referred to as a *first-match* policy and is used in the majority of firewall systems including the Linux firewall implementation `iptables` [17]. 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.

Definition Packet d matches r_i if

$$d \Rightarrow r_i \quad \text{iff} \quad d[l] \subseteq r_i[l], \quad l = 1, ..., k$$

The rule list R is comprehensive if for every possible legal packet d a match is found using R. Furthermore, two rule lists R and R' are equivalent if for every possible legal packet d the same action is performed by the two rule lists. If R and R' are different (e.g. a reorder) yet the lists are equivalent, then the **policy integrity** is maintained.

As previously mentioned, a rule list has an implied precedence relationship where certain rules must appear before others if the integrity of the policy is to be maintained. For example consider the rule list in figure 1(a). Rule r_1 must appear before rule r_4, likewise rule r_5 must be the last rule in the policy. If for example, rule r_4 was moved to the beginning of the policy, then it will *shadow* [1] the original rule r_1. Shadowing is an anomaly that occurs when a rule r_j matches a preceding rule r_i, where $i < j$. As a result of the relative order r_j will never be utilized. However, there is no precedence relationship between rules r_1, r_2, or r_3 given in figure 1(a). Therefore, the relative order of these three rules will not impact the policy integrity and can be changed to improve firewall performance.

As described in [8], [9], the precedence relationship between rules in a policy can be modeled as a Policy Directed Acyclical Graph (DAG). Let $G = (R, E)$ be a *policy DAG* for a rule list R, where vertices are rules and edges E are the precedence relationships (constraint). A precedence relationship, or edge, exists between rules r_i and r_j, if $i < j$ and the rules intersect [8].

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

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

Therefore, the intersection of two rules results in an ordered set of tuples that collectively describes the packets that match both rules. The rules r_i and r_j intersect if every tuple of the resulting operation is non-empty. In contrast, the rules r_i and r_j do not intersect, if at least one tuple is the empty set.

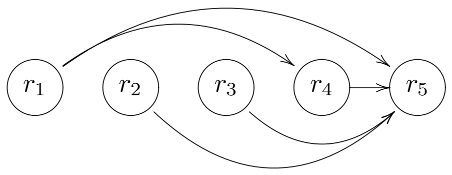
For example consider the rules given in figure 1(a), the intersection of r_1 and r_4 yields `(UDP, 1.1.*, *, *, 80)`. Again, the rule actions are not considered in the intersection or match operations. Since these two rules intersect, a packet can match both rules for example $d =$ `(UDP, 1.1.1.1, 80, 2.2.2.2, 80)`. The relative order must be maintained between these two rules and an edge drawn from r_1 to r_4 must be present in the DAG, as seen in figure 1(b). In contrast consider the intersection of rules r_2 and r_3 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 will not exist between them. Although policy optimization is not the focus of this paper, the policy DAG and related optimization techniques can be used to reorder rules to reduce the search time [8], [9].

III. Parallel Firewalls

As described in the introduction, parallelization offers a scalable technique for improving the performance of network firewalls. Using this approach an array of firewalls processes packets in parallel, as seen in figure 2. Again, the individual firewalls will be referred to as *firewall-nodes*.

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

(a) Example security policy consisting of multiple ordered rules.


$$\begin{tikzcd} r_1 \arrow[rrr, bend left] \arrow[rrrr, bend left] & r_2 \arrow[rrr, bend right] & r_3 \arrow[rr, bend right] & r_4 \arrow[r] & r_5 \end{tikzcd}$$

(b) Policy DAG representation, where vertices are rules while edges indicate precedence requirements.

Figure 1. Example firewall policy (ACL) and policy DAG.

However, the two designs depicted in this figure are different based on what is distributed: packets or rules. The design developed by Benecke et al [3] consisted of multiple identical firewalls-nodes connected in parallel, as shown in figure 2(a). Each firewall-node implements the complete security policy R, and arriving packets are distributed across the firewall-nodes for processing (one packet is sent to one firewall-node) [3]; therefore different packets are processed in parallel. Using terminology developed for parallel computing, this design is considered *data-parallel* since the data (packets) is distributed across the firewall-nodes [4]. How the load-balancing algorithm distributes the packets is important to the data-parallel system and typically implemented as a specialized high-speed switch [12], [13].

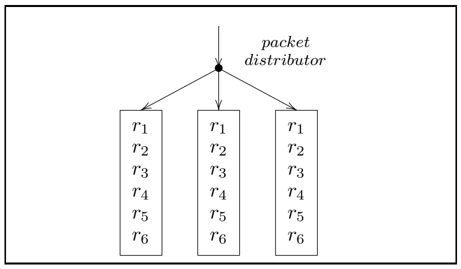
Although data-parallel firewalls have been shown to achieve higher throughput than traditional firewalls [3], they suffer from two major disadvantages. First, stateful inspection requires all traffic from a certain connection or exchange to traverse the same firewall-node, which is difficult to perform at high speeds using the data-parallel approach [3], [15]. Second, distributing packets is only beneficial under high traffic loads, which is explained in the next section.

A. Function-Parallel Architecture

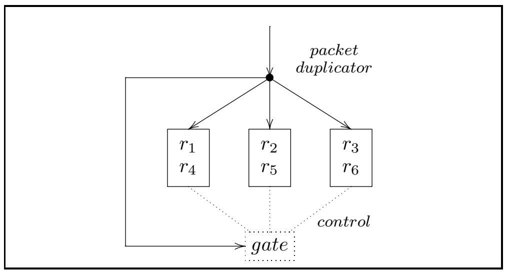
The new proposed firewall design consists of multiple firewall-nodes connected in parallel and a *gate* machine, as seen in figure 2(b). However unlike the data-parallel design that distributes arriving packets, the new design assigns the policy rules across the firewall-nodes. The distribution is done such that policy integrity is maintained. Therefore, a single traditional firewall and the new firewall design will always give the same result for any packet. The rules assigned to a firewall-node will be referred to as the local rule set or local policy. Using parallel computing terminology, this design will be referred to as *function-parallel* since the rules are distributed across the firewall-nodes [4].

The general operation of the system can be described as follows. When a packet arrives to the function-parallel system it is forwarded to every firewall-node and the gate. Each firewall-node processes the packet using its local rule list, including any state information. The firewall-node then signals the gate indicating either no match was found, or provides the rule number and action if a match was found. Therefore, *no-match* is a valid response and is required for the function-parallel design. The gate keeps track of the results and determines the final action to perform on the packet. Note, the rule number may correspond to a state match if the packet belongs to an established connection. This number would have a uniformly lower value since state rules are evaluated before policy rules [22]. A first match policy can be implemented by applying the action of the lowest numbered matching rule to the packet. Performance can be increased if the gate can signal the firewall-nodes that further processing of a certain packet is no longer needed (a short-circuit evaluation). This occurs when the appropriate match has been found by a firewall-node before the other firewall-nodes have completed processing the same packet. Short-circuit evaluation requires the gate to know how the rules are distributed as well as the dependencies. Redundancy can be provided by duplicating rules across the firewall-nodes (copying local rule-set to a neighbor). As done in [3], the firewall-nodes are interconnected to determine if the redundant rules should be processed [7]. Unfortunately, details have been omitted due to space constraints.

The function-parallel design has several significant advantages over traditional and data-parallel firewalls. First, the function-parallel design results in the faster processing since every firewall-node is utilized to process a single packet. Reducing the processing time, instead of the arrival rate (the data-parallel paradigm), yields better performance that will be demonstrated experimentally. Second, unlike the data-parallel design, function-parallel can maintain state information about existing connections. Maintaining state can be viewed as the addition of a new rule corresponding to a requested connection [22]. Unlike the data-parallel design, the new rule can be placed in any firewall-node since a packet will be processed by every firewall-node (integrity guidelines are given in the next section). The 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 [20], the scalability limit is not an important concern.


packet distributor
r_1 r_2 r_3 r_4 r_5 r_6
r_1 r_2 r_3 r_4 r_5 r_6
r_1 r_2 r_3 r_4 r_5 r_6

(a) Data-parallel, packets distributed across an array of equal firewall-nodes.



(b) Function-parallel, rules distributed across an array of firewall-nodes.

Figure 2. Parallel designs for network firewalls. In the diagrams, the security policy consists of six rules, $R = \{r_1, ..., r_6\}$ and each design consists of three firewall-nodes (depicted as solid rectangles).

B. Rule Distribution and Integrity

As previously described, the function-parallel design distributes the security policy rules across an array of firewall-nodes. Using this design, all the firewall-nodes process an arriving packet in parallel using its local rule-set. The rules must be distributed such that the integrity of the policy is maintained. Again, this paper will assume a first-match policy is desired. Therefore, the distribution of the rules must not introduce any anomalies into the original rule list, such as shadowing.

Let m be the number of firewall-nodes in the system and j be a firewall-node $j = 1...m$. Each firewall-node has a local rule-set that is a portion of the security policy. Denote the local rule list for firewall-node j as R_j that has n_j rules. The firewall-node will process an arriving packet using the local rule list in a top-down fashion to find the first match. Note, the actual software implementation of the firewall-node does not need to be a list, the function-parallel design is independent of software implementation. The gate will then apply the action of the lowest numbered rule matched by the firewall-nodes. Integrity is maintained if the rule distribution across the firewall-nodes adheres to the two conditions specified in the following theorem.

Theorem 1: Policy integrity is maintained by the function-parallel firewall if the rule distribution meets the following two conditions.

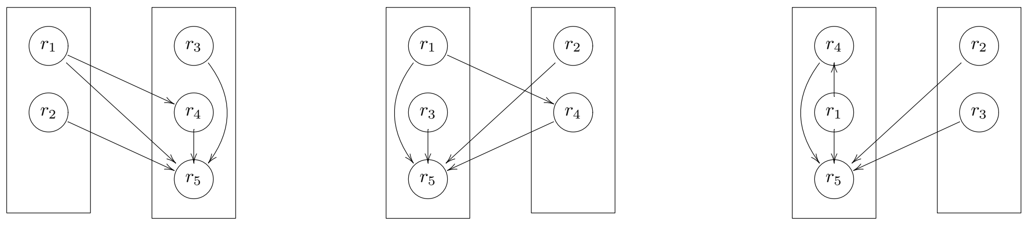
1. Every rule is assigned to at least one firewall-node.
2. A policy DAG edge never traverses upward within a firewall-node.

Proof: Let R' be the ordered subset of the rules in policy R that match packet d. Given a first match policy, the first rule in R' is the correct result. Furthermore, since every rule in R' matches d, the policy DAG for R' is completely dependent [8] (an edge exists between every rule) and only one order of these rules is possible. The first condition of the theorem ensures that these rules will exist in the system. The second condition ensures shadowing will never occur if multiple rules in R' exist in the same firewall-node. Each firewall-node can produce only one matching rule for d. If more than one rule in a local policy matches d then the local first match is produced. As a result, the gate may be given multiple matches from the set R' (local first match from multiple firewall-nodes) for d; however, the gate will determine the appropriate rule since it always applies the lowest numbered rule of the local first matches. Since policy DAG edges are only incident from lowered numbered rules to higher numbered rules, this always results the first rule in R' which is the correct rule; thus, policy integrity is maintained. ■

Consider a two firewall-node system and the five rule policy given in figure 1(a). Possible rule distributions are depicted in figure 3. The distribution given in figure 3(a) will be referred to as *vertical*, while the distribution given in figure 3(b) will be referred to as *horizontal*. In both cases the five rules are present and all policy DAG edges are downward; thus integrity is maintained. In contrast, figure 3(c) is an illegal distribution because rule r_4 shadows r_1 (upward policy DAG edge). The gate can also utilize the policy DAG to determine the first match. For example consider the distribution given in figure 3(a). Assume a packet matches r_3 in the second firewall-node and this result is reported to the gate. Using the policy DAG, the gate can determine this is the first match since it is impossible for a packet that matches r_3 to also match any rules that preceded it in the original policy (rules r_1 or r_2).

IV. Experimental Results

A discrete event simulator was used to measure the performance of a traditional single firewall, the data-parallel firewall, and the function-parallel firewall. Each firewall-node was simulated to process 6×10^7 rules per second, which is comparable to current technology. For each experiment the parallel designs always consisted of the same number of firewall-nodes. An additional gate delay, equivalent to processing three firewall rules, was added to the function parallel design. No additional delay was added to the data-parallel system for packet distribution; therefore, the delays observed for data-parallel are better than



(a) Vertical distribution.

(b) Horizontal distribution.

(c) Illegal rule distribution, r_1 is shadowed by r_4.

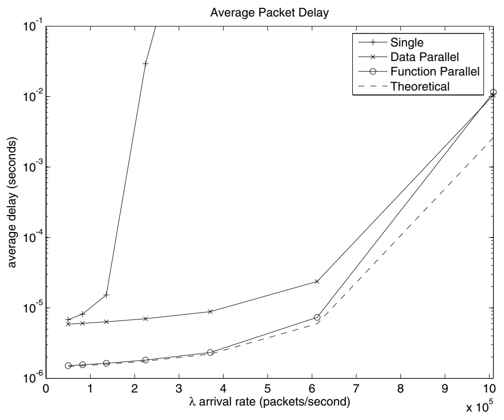
Figure 3. Example rule distributions for the rule set given in figure 1(a), using two firewall-nodes.

what should be expected. Packets lengths were uniformly distributed between 40 and 1500 bytes, while all legal IP addresses were equally probable. Firewall rules were generated such that the rule match probability was given by a Zipf distribution, which is commensurate with actual firewall policies [8], [14]. Rules were distributed in a horizontal fashion for the function-parallel design, as described in section B. Three sets of experiments were performed to determine the performance effect of increasing arrival rates, increasing policy size, and increasing number of firewall-nodes. For each experiment 1000 simulations were performed, then the average and maximum packet delay were recorded. The average results were also compared against the theoretical performance described in [7], which indicates the best average performance a function-parallel firewall can achieve is $1/m$ the data-parallel design, where m is the number of nodes. This theoretical result does not include the gate processing delay, so it can be considered a lower bound on delay.

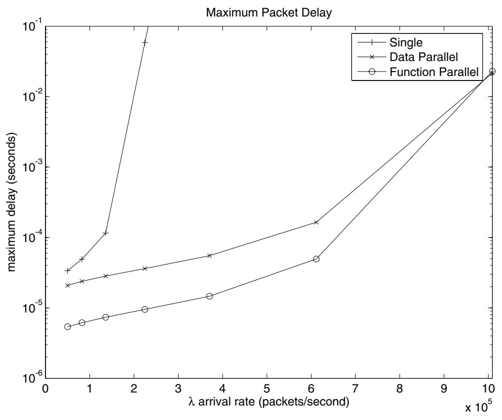
Figure 4 shows the impact of increasing arrival rates on performance of the three firewall systems. In this experiment, each system implemented the same 1024 rule firewall policy [20] and both parallel designs consisted of four firewall-nodes. The arrival rate ranged from 5×10^3 to 1×10^6 packets per second (the highest arrival rate resulted in more than 6 Gbps of traffic on average). As seen in figure 4, the parallel designs performed considerable better than the traditional single firewall. As the arrival rate increased, the parallel designs were able to handle larger volumes to traffic due to the distributed design. However as seen in figure 4(a), the function parallel firewall had an average delay that was consistently 3.5 times lower than the data parallel design. This is expected because each firewall-node in the function-parallel design is utilized to process the arriving packets. However, the gate delay incurred by the function-parallel design causes the average delay to be greater than the theoretical value, which is four times lower than the data-parallel design. The impact of the gate delay is more prominent as the arrival rate increases. Similar to the average delay results, the function parallel design had a maximum delay 72% lower than the data-parallel design.

The effect of increasing policy size (number of rules) on the three firewall systems is given in figure 5. In this experiment, the arrival rate was again 1×10^5 packets per second (yielding more than 0.5 Gbps of traffic on average) and both parallel designs consisted of four firewall-nodes. Policies ranged from 60 to 3840 rules [20]. When the policies consisted of relatively few rules, the single and data-parallel firewalls observed similar delays. However as seen in figure 5(a), the parallel designs performed considerable better than the traditional single firewall once the policy contained more than 1000 rules. The function parallel firewall had an average delay that was 3.8 times lower than the data parallel design. Therefore, the additional gate delay did not have a significant impact on the average performance. The maximum delay for the function parallel design was also lower than the other designs. For example, the maximum delay observed by the function-parallel firewall was 76% lower than the data-parallel firewall.

Figure 6 shows the effect of increasing number of firewall-nodes for the two parallel firewall designs. The number of firewall-nodes ranged from 2 to 256 in this experiment. The number of rules was 1024 and arrival rate was 2×10^5 packets per second (again, yielding more than 1 Gbps of traffic). As seen in figure 6(a), the function parallel design consistently performed better than the data-parallel firewall design. As machines were added, the function-parallel system always observed a reduction in the delay. This delay reduction trend is expected until the number of machines equals the number of rules. In contrast, the delay for data parallel design quickly stabilizes and the addition of more firewall-nodes has no impact. This is because after a certain point any additional firewall-nodes will remain idle, thus these additional firewall-nodes are unable to reduce the delay. As additional nodes are added the performance difference between the function-parallel firewall and theoretical limit becomes larger. The local policy delay becomes smaller as more nodes are added; however, the gate delay remains constant, thus more prominent in the total delay experienced. The function parallel design had a maximum delay 74% lower than the data-parallel design. As demonstrated in each experiment, the function-parallel design provides a better, scalable firewall solution.

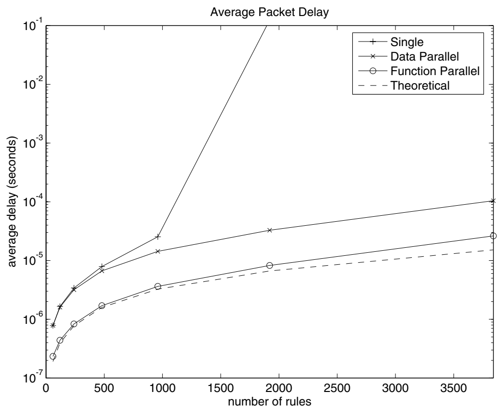


(a) Average delay.

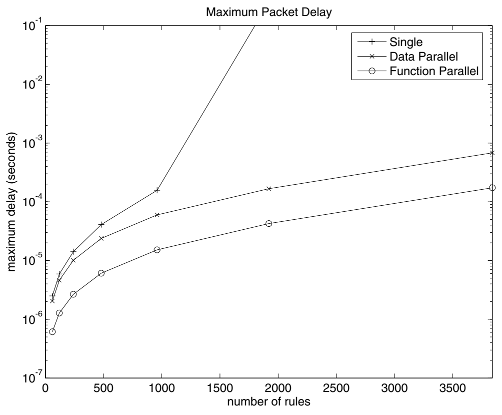


(b) Maximum delay.

Figure 4. Packet delay as the packet arrival rate increases. Parallel designs consist of four firewall-nodes.



(a) Average delay.



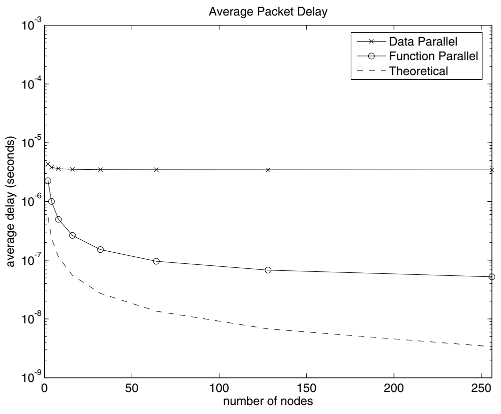
(b) Maximum delay.

Figure 5. Packet delay as the number of rules increases. Parallel designs consist of four firewall-nodes.

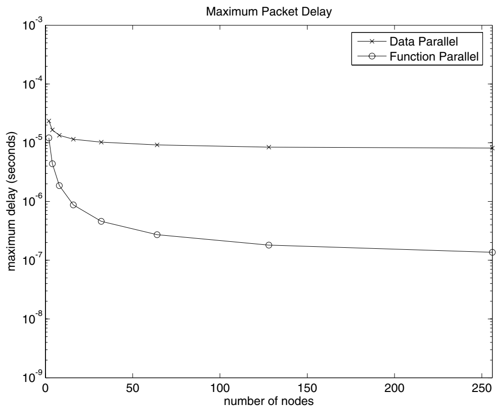
V. Conclusions

It is important that the 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 rule sets 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 is 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.

This paper introduced a new scalable firewall architecture designed for increasing network speeds and traffic loads. The proposed parallel design consists of multiple firewall-nodes. 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. In addition, 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 paper. The experimental performance showed that the proposed archi-



(a) Average delay.



(b) Maximum delay.

Figure 6. Packet delay as the number of firewall-nodes increases.

tecture can achieve a processing delay 74% lower than previous parallel-firewall designs. Furthermore unlike other designs, the proposed architecture can provide stateful inspections since a packet is processed by every firewall node. Therefore, the function-parallel firewall architecture is a scalable solution that offers better performance and more capabilities than other designs.

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 [8], [9]. However, optimization can only be done if policy integrity is maintained.

References

[1] E. Al-Shaer and H. Hamed. Modeling and Management of Firewall Policies. *IEEE Transactions on Network and Service Management*, 1(1), 2004.

[2] S. M. Bellovin and W. Cheswick. Network Firewalls. *IEEE Communications Magazine*, pages 50–57, Sept. 1994.

[3] C. Benecke. A Parallel Packet Screen for High Speed Networks. In *Proceedings of the 15th Annual Computer Security Applications Conference*, 1999.

[4] D. E. Culler and J. P. Singh. *Parallel Computer Architecture: A Hardware/Software Approach*. Morgan Kaufman, 1999.

[5] U. Ellermann and C. Benecke. Firewalls for ATM Networks. In *Proceedings of INFOSEC'COM*, 1998.

[6] D. Eppstein and S. Muthukrishnan. Internet Packet Filter Management and Rectangle Geometry. In *Proceedings of the Symposium on Discrete Algorithms*, pages 827–835, 2001.

[7] E. W. Fulp. Firewall Architectures for High Speed Networks. Technical Report 20026, Wake Forest University Computer Science Department, 2002.

[8] E. W. Fulp. Firewall Policy Models Using Ordered-Sets and Directed Acyclical Graphs. Technical report, Wake Forest University Computer Science Department, 2004.

[9] E. W. Fulp. Optimization of Network Firewall Policies Using Directed Acyclical Graphs. In *Proceedings of the IEEE Internet Management Conference (IM'05)*, 2005.

[10] E. W. Fulp and S. J. Tarsa. Network Firewall Policy Representation Using Ordered Sets and Tries. In *Proceedings of the IEEE International Symposium on Computer Communications (ISCC'05)*, 2005.

[11] R. Funke, A. Grote, and H.-U. Heiss. Performance Evaluation of Firewalls in Gigabit-Networks. In *Proceedings of the Symposium on Performance Evaluation of Computer and Telecommunication Systems*, 1999.

[12] 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.

[13] S. Goddard, R. Kieckhafer, and Y. Zhang. An Unavailability Analysis of Firewall Sandwich Configurations. In *Proceedings of the 6th IEEE Symposium on High Assurance Systems Engineering*, 2001.

[14] 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.

[15] O. Paul and M. Laurent. A Full Bandwidth ATM Firewall. In *Proceedings of the 6th European Symposium on Research in Computer Security ESORICS'2000*, 2000.

[16] L. Qui, G. Varghese, and S. Suri. Fast Firewall Implementations for Software and Hardware-Based Routers. In *Proceedings of ACM SIGMETRICS*, June 2001.

[17] V. P. Ranganath and D. 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.

[18] S. Suri and G. Varghese. Packet Filtering in High Speed Networks. In *Proceedings of the Symposium on Discrete Algorithms*, pages 969 – 970, 1999.

[19] P. Warkhede, S. Suri, and G. Varghese. Fast Packet Classification for Two-Dimensional Conflict-Free Filters. In *Proceedings of IEEE INFOCOM*, pages 1434–1443, 2001.

[20] A. Wool. A Quantitative Study of Firewall Configuration Errors. *IEEE Computer*, 37(6):62 –67, June 2004.

[21] J. Xu and M. 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.

[22] R. L. Ziegler. *Linux Firewalls*. New Riders, second edition, 2002.

[23] E. D. Zwicky, S. Cooper, and D. B. Chapman. *Building Internet Firewalls*. O'Reilly, 2000.