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
Email: fulp@wfu.edu
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 policies. 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 firewalls are used to process every packet, the proposed function-parallel system has significantly lower delays (e.g. 74% lower for a four firewall system) 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 is 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 [1], [2], [3]. 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) [3].

Traditional firewall implementations consist of a single dedicated machine, similar to a router, that sequentially applies the policy to each arriving packet. However, packet filtering can represent a significantly higher processing load than routing [4], [5], [3]. For example, a firewall that interconnects two 100 Mbps networks would have to process over 300,000 packets per second [2]. Successfully handling this traffic load becomes more difficult as policies become more complex [6], [7], [3]. 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 [6], [8], [9], [10]. Building a faster single firewall is possible [11], [4], [5], [12], [13]; however, the benefits are temporary (interface speeds are increasing); it is not scalable; and it is a single point of failure.

A parallel firewall (also called a load-balancing firewall) is a scalable approach for increasing the speed of inspecting network traffic [6], [10], [7], [3]. As seen in figure 2(a), the system consists of multiple identical firewalls connected in parallel. Each firewall in the system implements the complete security policy and arriving packets are distributed across the firewalls such that only one firewall processes any given packet [6]. How the load-balancing algorithm distributes packets is vital to the system and typically implemented as a high-speed switch in commercial products [14], [10]. Although parallel firewalls achieve a higher throughput than traditional firewalls [6] 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, which is difficult to perform at high speeds [7].

This paper introduces a new scalable parallel firewall architecture designed for increasing network speeds and traffic loads. The design consists of multiple firewalls where each firewall implements only a portion of the security policy. Since the policy is divided across the firewalls, rule distribution guidelines are provided that maintain integrity, ensuring the new parallel design and a traditional single firewall always reach the same decision. Unlike the previous parallel design, when a packet arrives to the new architecture it is processed by every firewall in parallel, thus the processing time required per packet is reduced. Simulation results for the new architecture, (consisting of four firewalls) yielded 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. Therefore, the new parallel design is a scalable solution that can offer 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

Packets arriving at a network firewall are inspected using the rules defined by a security policy. Each firewall rule r in the policy can be modeled as an ordered tuple of sets, $r = (r[1], r[2], ..., r[k])$. 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 [2], [3]. The ordered tuples can be supersets or subsets of each other. This forms the basis of precedence relationships (rule order). In addition to the prefixes, each filter rule has an action, which is to accept or deny.

Using the rule definition, a 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 [2]. Starting with the first rule, a packet d is sequentially compared against each rule r_i until a match is found, 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` [15]. 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.

A policy R is comprehensive if for every possible legal packet d a match is found. Furthermore, two policies R and R' are equivalent if for every possible legal packet d the same action is performed. Note, R and R' may differ in terms of the actual rules (number and/or order) yet still be equivalent [17]. Therefore, if R is replaced by an equivalent R' then **policy integrity** is maintained.

There are many ways to implement a given policy (e.g. using a single or parallel firewall) or even modify it (e.g. reorder, combine, add, or remove rules); therefore, it is important to determine equivalence and policy integrity. As described at the beginning of this section, a policy has an implied precedence relationship where certain rules must appear before others to maintain policy integrity. The precedence relationship between rules in a policy can be modeled as a Policy Directed Acyclical Graph (DAG) [16], [17]. Let $G = (R, E)$ be a *policy DAG* for a policy 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 [16]. 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, the policy DAG for the rules given in figure 1(a) is given in figure 1(b).

It has been proven that any linear arrangement of the policy DAG maintains integrity [17]. Therefore, the policy DAG can be used to distribute rules across a parallel firewall system as well as rearrange rules to improve performance [16], [17].

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 m firewalls processes packets in parallel, as seen in figure 2. 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. [6] consisted of multiple identical firewalls connected in parallel, as shown in figure 2(a). Each firewall j in the system implements a local policy R_j, where $R_j = R$. Arriving packets are distributed across the firewalls for processing (one packet is sent to one firewall), allowing different packets to be processed in parallel. Since each packet is processed using the policy $R_j = R$, policy integrity is maintained.

Using terminology developed for parallel computing, this design is considered *data-parallel* since the data (packets) is distributed across the firewall [18]. How the load-balancing algorithm distributes the packets is important and typically implemented as a specialized high-speed switch [14], [10].

Although a data-parallel firewall can achieve higher throughput than a traditional (single machine) firewall [6], it suffers from two major disadvantages. First, stateful inspection requires all traffic from a certain connection or exchange to traverse the same firewall. Successful connection tracking is difficult to perform at high speeds using the data-parallel approach [6], [7]. Second, distributing packets is only beneficial when each firewall in the array has a significant amount of traffic to process (never idle), which only occurs under high traffic loads. This behavior is demonstrated in the experimental results section.

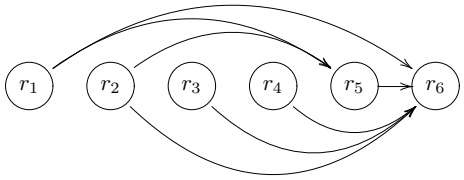
A. Function-Parallel Architecture

The new proposed firewall design consists of multiple firewalls connected in parallel and a *gate* device, 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 firewalls. Since firewalls only implement a portion of the original policy, it is critical that rule distribution is done to maintain policy integrity. Therefore, a traditional single firewall and the new firewall design will always give the same result for any packet. Using parallel computing terminology, this design will be referred to as *function-parallel* since the rules are distributed across the firewalls [18].

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 and the gate. Each firewall processes the packet using its local policy, including any state information. The firewall then signals the gate indicating either no match was found, or provides the rule number and action if a match was found. Since local policies are a subset of the

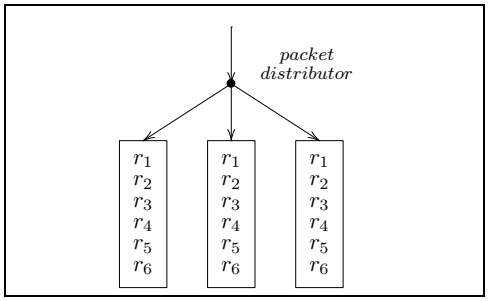
No.	Proto.	Source IP	Source Port	Destination IP	Destination Port	Action
1	UDP	190.1.1.*	*	*	80	deny
2	UDP	190.1.*	*	*	90	deny
3	TCP	180.*	*	180.*	90	accept
4	TCP	210.*	*	220.*	20	accept
5	UDP	190.*	*	*	*	accept
6	*	*	*	*	*	deny

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

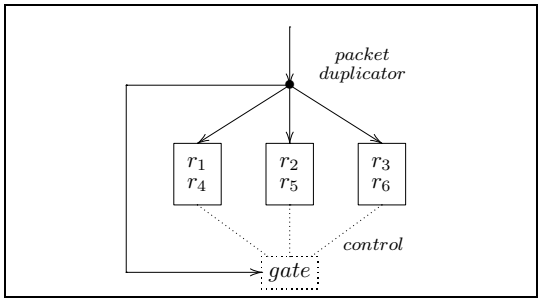


(b) Policy DAG representation, where vertices are rules while edges indicate precedence requirements.

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



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



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

Fig. 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 firewalls (depicted as solid rectangles).

original, a *no-match* is a valid response and is required for the function-parallel design. The gate stores the results and determines the final action to perform on the packet using the policy DAG. 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 [2].

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 firewalls 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 before the other firewalls have completed processing the same packet. Short-circuit evaluation requires the gate to know how the rules are distributed as well as the dependencies (policy DAG). Redundancy can also be provided by duplicating rules across the firewalls (copying local policy to a neighbor). As done in [6], the firewalls are interconnected to determine if the redundant rules should be processed [19].

The function-parallel design has several significant advantages over traditional and data-parallel firewalls. First, the function-parallel design results in faster processing since every firewall is utilized to process a single packet. Reducing the processing time, instead of the arrival rate (the data-parallel paradigm), yields better performance since each firewall in the array processes packets regardless of the traffic load. Second, unlike the data-parallel design, the function-parallel design can maintain state information about existing connections. Maintaining state can be viewed as the addition of a new rule corresponding to a requested connection [2]. Unlike the data-parallel design, the new rule can be placed in any firewall since a packet will be processed by every firewall (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 firewalls 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.

B. Rule Distribution and Integrity

As previously described, the function-parallel design distributes the security policy rules across an array of firewalls. Using this design, all the firewalls process an arriving packet in parallel using its local policy; therefore, 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 and policies are comprehensive.

Assume a function-parallel system consisting of m firewalls exists and will enforce a security policy R. Each firewall j has a local policy R_j that is a portion of the security policy R. Each firewall will process an arriving packet using its local policy to determine the first match. Note, the actual software implementation of the firewall 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 firewalls. Integrity is maintained if the rule distribution adheres to the conditions specified in the following theorem.

Theorem 1: An array of m firewalls and a gate device arranged in a function-parallel fashion enforcing a comprehensive policy R will maintain integrity if policy rules are distributed to each firewall j such that: every rule in R is assigned to at least one firewall and the precedence constraints of R are observed in R_j.

Proof: Let R' be the ordered subset of the rules in policy R that matches a packet d. Given a first match policy, the first rule in R' is the correct result. 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 from R' exist in the same local policy. Therefore the local first match produced for d is the correct local result. If rules from R' exist in different local policies then the gate will be given multiple matches from the set R' for d. However, the gate will determine the appropriate result since it always applies the lowest numbered rule of the local first matches, thus policy integrity is maintained. ■

An example distribution of the policy given in figure 1(a), across an array of three firewalls, is shown in figure 2(b). This *horizontal distribution* assigns rules in a round-robin fashion horizontally across the firewalls. Integrity is maintained since all rules are present in the system (assigned to at least one firewall) and rules are in ascending order in each local policy (shadowing is not introduced).

IV. Experimental Results

The performance of a traditional single firewall, the data-parallel firewall, and the function-parallel firewall was measured under realistic conditions using simulation. Firewalls were 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 firewalls. 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 (load balancing); therefore, the results observed for the data-parallel design are better than what should be expected.

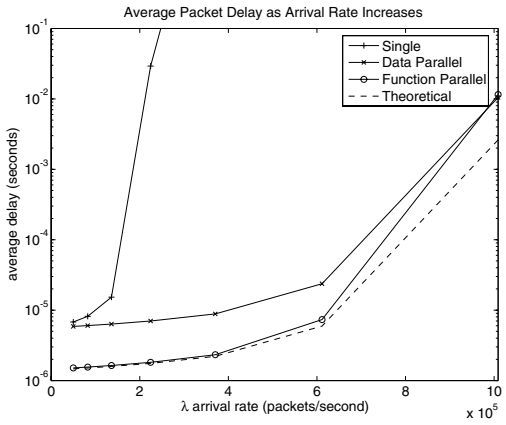
Packets lengths were uniformly distributed between 40 and 1500 bytes, while all legal IP addresses were equally probable. Firewall rules were generated such that the rule match probability was given by a Zipf distribution, which is commensurate with actual firewall policies [16], [21]. Rules were distributed in a horizontal fashion for the function-parallel design, as described in section III-B.

Three sets of experiments were performed to determine the effect of increasing arrival rates, increasing policy size, and increasing number of firewalls. 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 [19], 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 firewalls. This theoretical result does not include the gate processing delay, so it can be considered a lower bound on the average delay.

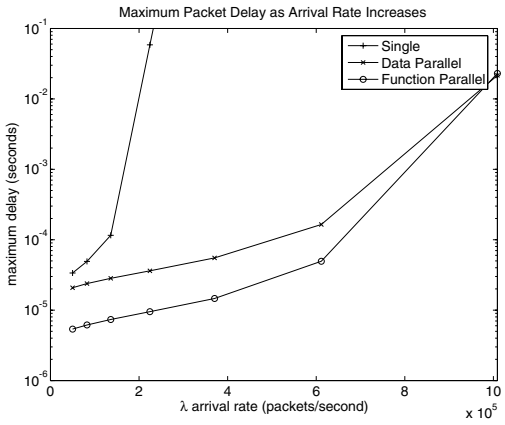
Figure 3 shows the impact of increasing arrival rates on the three firewall designs. In this experiment, each system implemented the same 1024 rule firewall policy [20] and both parallel designs consisted of four firewalls. The arrival rate ranged from 5×10^3 to 1×10^6 packets per second (the highest arrival rate resulted in more than 6 Gbps of traffic on average). As seen in figure 3, 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. As seen in figure 3(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 in the function-parallel design is used to inspect arriving packets regardless of the arrival rate. 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 designs is given in figure 4. 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 firewalls. 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 4(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 5 shows the effect of increasing number of firewalls for the two parallel firewall designs. The number of firewalls ranged from 2 to 256, 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 5(a), the function-parallel design consistently performed better than the data-parallel firewall design. As firewalls were added, the function-parallel system always observed a reduction in the delay. This delay reduction trend is expected until the number of firewalls equals the number of rules. In contrast, the delay for data parallel design quickly stabilizes and the addition of more firewalls has no impact. This is because after a certain point

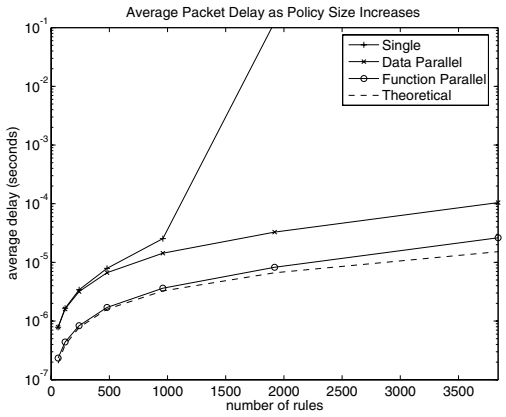


(a) Average delay.

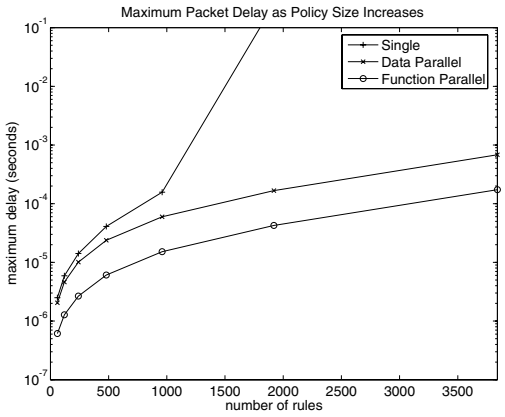


(b) Maximum delay.

Fig. 3. Packet delay as the packet arrival rate increases. Parallel designs consisted of four firewalls.

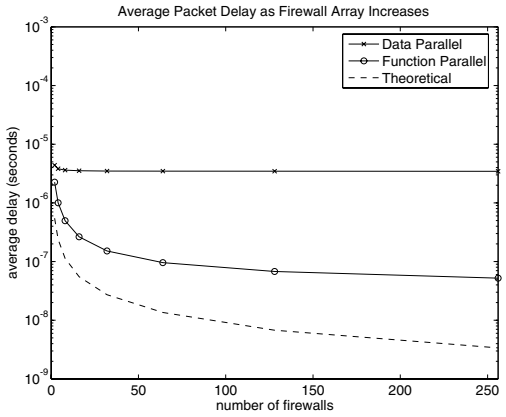


(a) Average delay.

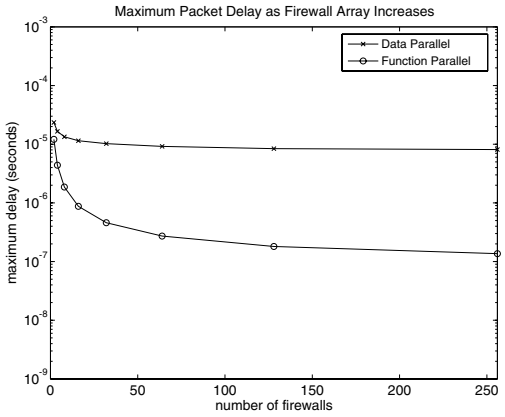


(b) Maximum delay.

Fig. 4. Packet delay as the number of rules increases. Parallel designs consisted of four firewalls.



(a) Average delay.



(b) Maximum delay.

Fig. 5. Packet delay as the number of firewalls increases. Policies consisted of 1024 rules.

any additional firewalls will remain idle, thus these additional firewalls are unable to reduce the delay. As additional firewalls are added the performance difference between the function-parallel firewall and theoretical limit becomes larger. The local policy delay becomes smaller as more firewalls 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.

V. Conclusions

It is important that a network 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. Unfortunately, the firewall can quickly become a bottleneck given increasing traffic loads and network speeds. Packets must be inspected and compared against complex policies, 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.

This paper introduced a new scalable firewall architecture designed for increasing network speeds and traffic loads. The proposed parallel design consists of multiple firewalls. Each firewall implements a portion of the security policy. When a packet arrives to the system it is processed by every firewall simultaneously, which significantly reduces the processing time per packet. In addition, rule distribution guidelines that maintain policy integrity were introduced, 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 architecture can achieve a processing delay 74% lower than previous parallel firewall designs (parallel architectures consisted of four firewalls). Furthermore unlike other designs, the proposed architecture can provide stateful inspections since a packet is processed by every firewall. 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. Another open question is the optimization of local firewall policies, including redundant policies, using the methods described in [16], [17]. However, optimization can only be done if policy integrity is maintained.

Acknowledgment

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 authors 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.

References

[1] S. M. Bellovin and W. Cheswick, "Network firewalls," *IEEE Communications Magazine*, pp. 50–57, Sept. 1994.
[2] R. L. Ziegler, *Linux Firewalls*, 2nd ed. New Riders, 2002.
[3] E. D. Zwicky, S. Cooper, and D. B. Chapman, *Building Internet Firewalls*. O'Reilly, 2000.
[4] L. Qui, G. Varghese, and S. Suri, "Fast firewall implementations for software and hardware-based routers," in *Proceedings of ACM SIGMETRICS*, June 2001.
[5] S. Suri and G. Varghese, "Packet filtering in high speed networks," in *Proceedings of the Symposium on Discrete Algorithms*, 1999, pp. 969 – 970.
[6] C. Benecke, "A parallel packet screen for high speed networks," in *Proceedings of the 15th Annual Computer Security Applications Conference*, 1999.
[7] 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.
[8] U. Ellermann and C. Benecke, "Firewalls for ATM networks," in *Proceedings of INFOSEC'COM*, 1998.
[9] 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.
[10] 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.
[11] D. Eppstein and S. Muthukrishnan, "Internet packet filter management and Rectangle geometry," in *Proceedings of the Symposium on Discrete Algorithms*, 2001, pp. 827–835.
[12] P. Warkhede, S. Suri, and G. Varghese, "Fast packet classification for Two-Dimensional Conflict-Free filters," in *Proceedings of IEEE INFOCOM*, 2001, pp. 1434–1443.
[13] 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*, vol. 17, no. 6, pp. 1190–1200, June 1999.
[14] X. Gan, T. Schroeder, S. Goddard, and B. Ramamurthy, "LSMAC vs. SLNAT: scalable cluster-based web servers," *Journal of Networks, Software Tools, and Applications*, vol. 3, no. 3, pp. 175–185, 2000.
[15] 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*, 2003, pp. 889–894.
[16] E. W. Fulp, "Firewall policy models using ordered-sets and directed acyclical graphs," Wake Forest University Computer Science Department, Tech. Rep., 2004.
[17] ——, "Optimization of network firewall policies using directed acyclical graphs," in *Proceedings of the IEEE Internet Management Conference (IM'05)*, 2005.
[18] D. E. Culler and J. P. Singh, *Parallel Computer Architecture: A Hardware/Software Approach*. Morgan Kaufman, 1999.
[19] E. W. Fulp, "Firewall architectures for high speed networks," Wake Forest University Computer Science Department, Tech. Rep. 20026, 2002.
[20] A. Wool, "A quantitative study of firewall configuration errors," *IEEE Computer*, vol. 37, no. 6, pp. 62 –67, June 2004.
[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*, vol. 2, pp. 1 – 15, 1994.