EFFECTS OF PROCESSING DELAY ON FUNCTION-PARALLEL FIREWALLS[1]

Ryan J. Farley and Errin W. Fulp
Department of Computer Science
Wake Forest University
Winston-Salem, NC 27109, USA
email: farlrj4@wfu.edu
`nsg.cs.wfu.edu`

ABSTRACT
Comprehensive security policies are an integral part of creating a secure network and commonly firewalls are used to accomplish this. Firewalls inspect and filter traffic arriving or departing a network by comparing packets to a set of rules and performing the matching rule action, which is accept or deny. Unfortunately, traffic inspection of this type 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 investigates the performance of a function parallel firewall architecture that distributes the original policy across an array of firewalls. A packet is processed by all the firewalls simultaneously and a gate then makes a final decision (accept or deny) based on the results of the individual firewalls. Since the individual firewalls have fewer rules to process (only a portion of the original policy), the function parallel system has lower delays (e.g. 74% lower for a four firewall array) and a higher throughput than other data parallel (load-balancing) firewalls. However, the performance increase is dependent on the speed of the gate. The potential speed increase and the impact of the gate will be demonstrated empirically.

KEY WORDS
Interconnection Networks, Security, Parallel Computing Systems.

1 Introduction

Traditional firewall implementations consist of a single dedicated machine, similar to a router, that sequentially applies a rule set to each arriving packet. However, packet filtering can represent a significantly higher processing load than routing decisions [15, 17, 22]. Successfully handling this high traffic becomes more difficult as policies become more complex [2, 14, 22]. 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 [2, 4, 10, 12]. Building a faster single firewall is possible [5, 15, 17, 18, 20]; 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.

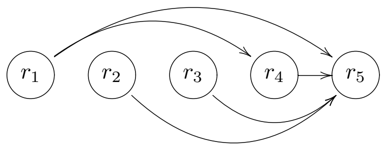
A data parallel (load-balancing) firewall is a scalable approach for increasing the speed of inspecting network traffic [2, 12, 14, 22]. As seen in figure 2(a), the system consists of multiple identical firewalls connected in parallel. Each firewall node implements the complete security policy and arriving packets are distributed across the firewall nodes such that only one firewall node processes any given packet [2]. How the load-balancing algorithm distributes packets is vital to the system [11, 12]. Although data parallel firewalls achieve higher throughput than traditional firewalls [2] and have a redundant design, the performance benefit is less evident in moderate 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 [14].

This paper investigates the performance of a new parallel firewall architecture designed for increased network speeds and traffic loads [6]. This architecture segments the security policy across the firewall nodes, reducing the processing time required per packet on any firewall node. When a packet arrives to the system, it is processed by every firewall node in parallel, validating it against the entire policy. A gate node then makes the final decision on the packet based on the results of the individual firewall nodes. With proper rule distribution this preserves policy integrity, ensuring that 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. However, the perfor-

[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.

		Source		Destination		
No.	Proto.	IP	Port	IP	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.



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

Figure 1. Example firewall policy and policy DAG.

mance gains of the function parallel design are dependent on the speed of the gate, which will be shown empirically.

The remainder of this paper is structured as follows. Section 2 reviews firewall policy models that are used for rule distribution in the proposed parallel system. Parallel firewall designs are described in section 3, including the new design and rule distribution methods. Then section 4 will demonstrate the experimental performance of the parallel design, including the impact of the gate delay. Section 5 reviews the parallel firewall design and discusses some open questions.

2 Firewall Policy Models

The main goal of any firewall design is to preserve policy integrity, and this section will review firewall policies and then describe a policy model that sustains this directive [8]. A policy is an ordered set of n rules denoted as $R = \{r_1, r_2, ..., r_n\}$. 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 [21, 22]. 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.

State can be viewed as a preliminary extension of the policy that contains a set of rules for established connections [21]. Note, the internal representation of the policy does not have to be a list [9, 15]. 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* method and is used in the majority of firewall systems [16]. 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 A packet $d = (d[1], d[2], \dots, d[k])$ matches rule r if $d[l] \subseteq r[l], l = 1, \dots, k$.

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

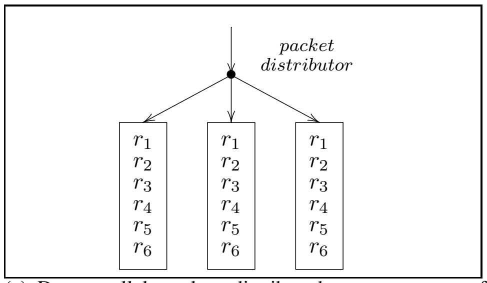
As previously mentioned, a policy has an implied precedence relationship where certain rules must appear before others if the integrity of the policy is to be maintained. For example, consider the policy in figure 1(a). Rule r_1 must appear before rule r_4, likewise rule r_5 must be the last rule in 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 [7, 8], 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 policy R, where vertices are rules and edges E are the precedence relationships (constraints). A precedence relationship, or edge, exists between rules r_i and r_j, if $i < j$ and the two rules intersect – the intersection of every associated tuple is non-empty [7]. In contrast, the rules r_i and r_j do not intersect, if at least one tuple is the empty set. Therefore, the intersection of two rules results in an ordered set of tuples that collectively describes the packets that match both rules.

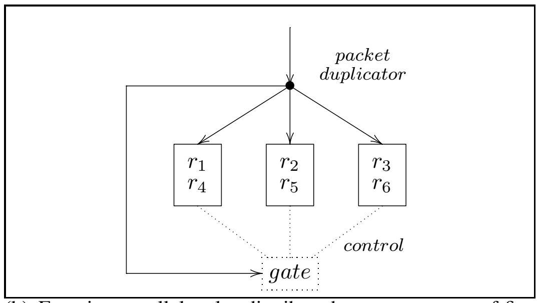
Although policy optimization is not the focus of this paper, the policy DAG and related optimization techniques can be used to reorder rules to reduce the search time [7, 8].

3 Parallel Firewall Architectures

As described in the introduction, parallelization can greatly enhance the performance of network firewalls by offering a scalable technique to reduce workloads. Using this approach an array of 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: pack-


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).

ets or rules. The design developed by Benecke et al. [2] consisted of multiple identical firewall nodes connected in parallel. 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) [2]; therefore different packets are processed in parallel. Using terminology developed for parallel computing, this design is considered *data parallel* since the data (packets) are distributed across the firewall nodes [3].

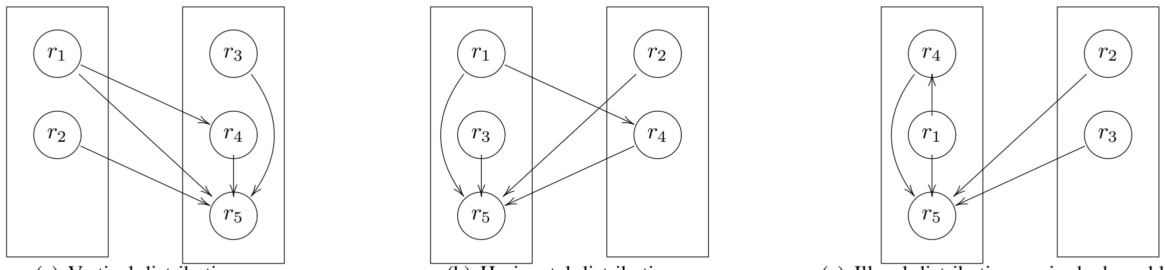
Although data parallel firewalls have been shown to achieve higher throughput than traditional firewalls [2], they suffer from two major disadvantages. First, stateful inspection requires all traffic from a certain connection or exchange to traverse the same firewall node, which is difficult to perform at high speeds using the data parallel approach [2, 14]. Second, distributing packets in this method is less beneficial under moderate traffic loads. As a result, new firewall architectures must solve these problems to meet future demands and increasing security threats.

In [6] a new firewall design was proposed that consists of multiple firewall nodes connected in parallel and a *gate node*, as seen in figure 2(b). However unlike the data parallel design that distributes arriving packets, this design assigns the policy 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 policy. Using parallel computing terminology, this design will be referred to as *function parallel* since the rule processing is distributed across the firewall nodes [3].

The operation of a function parallel system can be described as follows. When a packet arrives to the function parallel system it is duplicated to every firewall node and the gate. The gate node then caches the packet while each firewall node processes the duplicate packet using its local policy, 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 tracks the result from each firewall node 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 [21]. 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, information which is already present in the policy DAG.

As described in [6], 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, 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 [21]. 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 [19], the scalability limit is not an important concern. Another issue is the speed of the gate node. As will be demonstrated in the simulation results, the gate node must not incur so much delay as to negate the advantages of the distributed policy processing.



(a) Vertical distribution. (b) Horizontal distribution. (c) Illegal distribution, r_1 is shadowed by r_4.

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

3.1 Rule Distribution and Integrity

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

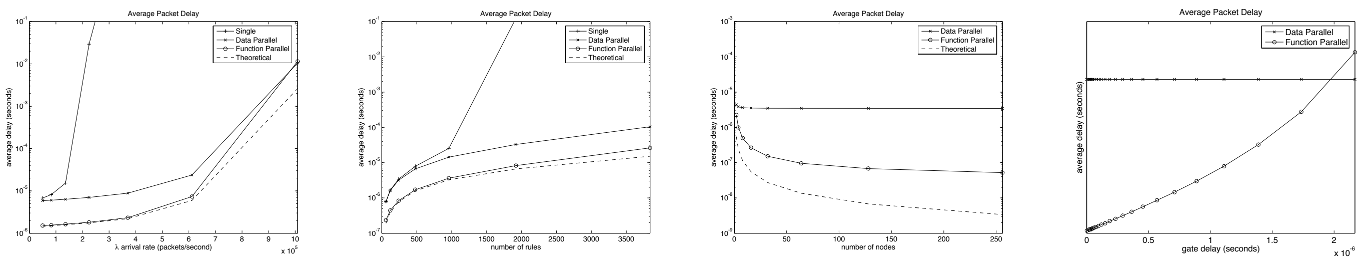
Proof: Let R' be the ordered subset of the rules in policy R that match packet d. Given a first match policy, the first rule in R' is the correct result. Furthermore, since every rule in R' matches d, the policy DAG for R' is completely dependent [7] (an edge exists between every rule) and only one order 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).

A simple modification to the rule distribution method can lead to another advantage of this design. By duplicating rules across the firewall nodes (copying the local policy to a neighbor) redundancy is provided without requiring additional firewall nodes. As done in [2], the firewall nodes are interconnected to determine if the redundant rules should be processed [6]. Unfortunately, details have been omitted due to space constraints.

4 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



(a) Increasing arrival rate. (b) Increasing number of rules. (c) Increasing firewall nodes. (d) Increasing gate delay.

Figure 4. Average packet delay for a single firewall, four node data parallel firewall, and four node function parallel firewall.

number of firewall nodes. An additional gate delay was added to the function parallel design; however, no additional delay was added to the data parallel system for packet distribution. Therefore, the delays observed for data parallel 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 [7, 13]. Rules were distributed in a horizontal fashion for the function parallel design, as described in section 3.1. Four sets of experiments were performed to determine the performance effect of increasing arrival rates, increasing policy size, increasing number of firewall nodes, and increasing function parallel gate delay. For each experiment 1000 runs were performed, then the average and maximum packet delays were recorded. The average results were also compared against the theoretical performance described in [6], 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 firewall nodes. This theoretical result does not include the gate processing delay, so it can be considered a lower bound on delay.

As described in [6] and seen in figure 4, both parallel designs perform better than a traditional single firewall under increasing traffic loads and policy sizes. In addition, the function parallel design consistently performed better than the data parallel design, yielding approximately 74% reduction in the delay. However, the function parallel delays are higher than the theoretical value. This is due to the additional function parallel design gate delay that was equivalent to processing three additional firewall rules. As seen in figure 4(a), the gate delay becomes more significant as the arrival rate increases. The impact of the gate is also evident in figure 4(c), which shows the delay as the number of firewall nodes is increased. As additional firewall nodes are added the performance difference between the function parallel firewall and theoretical limit becomes larger. The local policy delay becomes smaller as more firewall nodes are added; however, the gate delay remains constant, thus more prominent in the total delay experienced.

Another set of experiments were conducted to further investigate the impact of the gate delay. In this experiment, the gate delay is increased from zero to 0.00225 msec, while the arrival rate, number of rules, and number of firewall nodes were held constant. Each parallel design consisted of four firewall nodes, arrival rate was equivalent to 1 Gbps of traffic, and the policy consisted of 512 rules. As shown in figure 4(d), increasing the gate delay increases the average system delay for the function parallel system. Once the gate delay was greater than 0.002 msec (equal to processing 125 extra firewall rules), the function parallel firewall no longer performed better than the data parallel design. These results indicate the function parallel firewall can perform better than an equivalent data parallel design if the additional gate delay is only a fraction of the policy delay.

5 Conclusions

It is important that the firewall acts transparently to legitimate users, with little or no effect on the perceived network performance. Unfortunately, a firewall can quickly become a bottleneck given increasing traffic loads and network speeds. Packets must be inspected and compared against policies, which is a time consuming process. In addition, audit files must be updated with current connection information. As a result, a 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.

In [6], a new scalable firewall architecture designed was proposed for increasing network speeds and traffic loads. 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. The gate node then make the final decision for the packet based on the results of the individual firewall nodes. Therefore, the gate delay is a critical component of the overall system delay. Rule distribution across the firewall nodes must be done to maintain policy integrity, which ensures the parallel design and a 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. 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 Quality of Service (QoS) in future networks, it may be possible to distribute rules such that traffic flows are isolated. Another open question is the optimization of firewall node local policies, including redundant policies, using the methods described in [7, 8].

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

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

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

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

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

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

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

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

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

[16] 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.

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

[18] 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.

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

[20] 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.

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

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