An Independent Function-Parallel Firewall Architecture for High-Speed Networks (Short Paper)

Errin W. Fulp

Wake Forest University, Department of Computer Science,
Winston-Salem NC 27109, USA
fulp@wfu.edu
http://nsg.cs.wfu.edu

Abstract. A function-parallel network firewall is a scalable architecture that consists of multiple firewalls. Rules are distributed across the array such that each firewall implements a portion of the original policy. This resutls in significantly lower delays than other parallel designs; however, the design requires firewall intercommunication to coordinate the array which is difficult to implement and introduces additional delay.

This paper describes how the performance of a function-parallel firewall array can be increased if the individual firewalls can operate independently, without firewall intercommunication. By distributing rules using accept sets, the independent firewall array and a traditional single firewall will always arrive at the same decision (integrity is maintained). Simulation results will show the system is significantly faster than other designs and has the unique ability to provide service differentiation.

1 Introduction

Parallelization has been increasingly used as an approach for inspecting network packets in a high speed environment [1,2,3]. As seen in figure 2(a), a parallel firewall system consists of an array of firewalls connected in parallel. However as depicted in the figure, the systems differ based on what is distributed, packets (data-parallel) or policy rules (function-parallel).

Each firewall in a data-parallel system implements the complete security policy and arriving packets are distributed across the firewalls such that only one firewall processes any given packet [1]. Although the data-parallel firewall design achieves a higher throughput than traditional firewalls [1], 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 to maintain state information, which is difficult at high speeds [3].

As depicted in 2(b), a function-parallel design also consists of an array firewalls, however each firewall implements only a portion of the security policy [4]. When a packet arrives to the function-parallel system it is processed by every firewall in parallel, thus the processing time required per packet is reduced. Furthermore, maintaining state information is possible since a packet is inspected

P. Ning, S. Qing, and N. Li (Eds.): ICICS 2006, LNCS 4307, pp. 292–301, 2006.

by every firewall. Once processing is complete for a packet, results from the individual firewalls are sent to a *gate* device that stores the packet and determines the final action (accept or drop). The system can perform better than an equivalent data-parallel firewall [4]; however, the gate device implementation requires specialized hardware and introduces an additional delay.

This paper describes how function-parallel firewall array can operate independently (without a gate device) which will yield better performance. Independence can be achieved if firewall rules are distributed based on the security policy *accept set*, which describes the set of packets that will be accepted. Distribution must be done such that the union of each local accept set equals the original accept set (original and distributed policies accept the same packets), while the intersection of the local accept sets is the empty set (a packet will be accepted by only one firewall). By meeting these two requirements, it will be proven that policy integrity is maintained. Simulation results will show the independent function-parallel firewalls perform better under various conditions. In addition, accept sets can be designed such that certain types of packets are only processed on specific firewalls, yielding the ability to provide service differentiation which is a key component for maintaining network QoS. Thus, the function-parallel design has the most potential for successfully inspecting packets in a high-speed environment.

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 independent function-parallel design and rule distribution methods. Then section 4 will demonstrate the experimental performance of the parallel design. Section 5 reviews the parallel firewall design and discusses some open questions.

		Source		Destination		
No.	**Proto.**	**IP**	**Port**	**IP**	**Port**	**Action**
1	UDP	190.1.1.*	*	*	80	deny
2	UDP	210.1.*	*	*	90	accept
3	TCP	180.*	*	180.*	90	accept
4	TCP	210.*	*	220.*	80	accept
5	UDP	190.*	*	*	*	accept
6	*	*	*	*	*	deny

Fig. 1. Example security policy consisting of multiple ordered rules

2 Firewall Security Policies

A firewall rule r can be modeled as an ordered tuple, $r = (r[1], r[2], ..., r[k])$, where 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, firewall 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

[5]. In addition to the prefixes, each filter rule has an action, which is to accept or deny. Security can be enhanced with connection state and packet audit information [5].

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 [5]. Starting with the first rule, a packet p is sequentially compared against each rule r_i until a match is found, then the associated action is performed. This type of packet processing is referred to as a *first-match* policy and is typically the default for the majority of firewall systems including the Linux firewall implementation `iptables` [6].

When designing or verifying a firewall security policy it is important to determine the packets that will be accepted, denied, or not match any rule. Given a policy R, let A be the set of packets that will be accepted, let D be the set of packets that will be denied, and let U be the set of packets that do not match any rule. If the set of all possible packets is C, then a policy R is comprehensive if $U = \emptyset$ (i.e. $A \cup D = C$). Therefore, policy R is comprehensive if for every possible packet a match is found, which is an important objective. Furthermore, assume R does not necessarily equal R' in terms of the policy rules.

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. Consider two comprehensive policies R and R' that have accept sets A and A' respectively. The two policies are considered equivalent if $A = A'$. Therefore, if policy R is replaced by an equivalent policy R' then the **integrity** of R is maintained. Therefore, it is important to maintain the precedence constraints with implementing a firewall security policy.

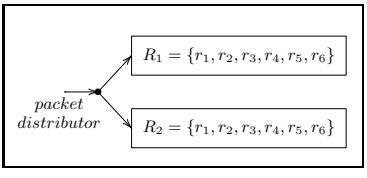
3 Parallel Firewalls

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 designs depicted in the figure differ based on what is distributed: packets or rules. Using terminology from parallel computing, distributing packets can be considered *data-parallel* since the data (packets) is distributed across the firewall [7]. In contrast, *function-parallel* designs distribute policy rules across the firewalls.

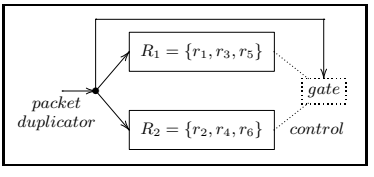
3.1 Data-Parallel Architecture

As shown in figure 2(a), data-parallel firewall architecture consists of an array of identically configured firewalls [1]. 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 the accept set for each firewall j equals the accept set of the original policy, $A_j = A$, policy integrity is maintained.

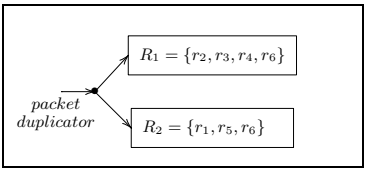
Distributing packets across the array allows a data-parallel firewall to increase system throughput as compared to a traditional (single machine) firewall [1]. However the data-parallel approach has three major disadvantages. First, stateful inspection requires all traffic from a certain connection or exchange to traverse the same firewall (where the stateful rule resides) or the constant distribution and management of stateful rules. As a result, successful connection tracking is difficult to perform at high speeds using the data-parallel approach [1,3]. Second, distributing packets is only beneficial when each firewall in the array has a significant amount of traffic to process (firewalls are never idle). The performance benefit (higher throughput) only occurs under high traffic loads. Finally, the design does not differentiate between traffic classes only load balancing. Therefore efficiently maintaining different QoS requirements is not possible.



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



(b) Function-parallel with gate, rules distributed across firewalls.



(c) Function-parallel, rules distributed across independent firewalls.

Fig. 2. Various parallel designs for network firewalls. The original security policy consists of six rules $R = \{r_1, ..., r_6\}$ and each design consists of two firewalls (depicted as solid rectangles, where local policies are given within each rectangle).

3.2 Function-Parallel Architecture

Unlike the data-parallel model which distributes packets, the function-parallel design distributes policy rules across the firewall array [4]. The function-parallel design consists of multiple firewalls connected in parallel and a *gate* device. As seen in figure 2(b), 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. Since the local policies are smaller than the original, the processing delay is reduced as compared to a traditional firewall. Once the firewall finishes processing a packet, it then signals the gate indicating either no match was found, or provides the rule number and

action if a match was found. The gate stores the results for the packet and determines the final action to perform.

Since firewalls only implement a portion of the original policy, it is critical that rule distribution is done to maintain integrity. The integrity of a policy R is maintained if the rules are distributed such that every rule in R exists in the system and if the precedence constraints of R are observed in each local policy R_j. As a result, the accept set of the gate equals the accept set of the original policy [4]. Several different distributions are possible that adhere the described guidelines. Essentially the rule numbers (indexes from the original policy) in each local policy must be in ascending order, as seen in figure 2(b).

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, 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. The new state rule can be placed in any firewall since a packet will be processed by every firewall.

There are three disadvantages of the function-parallel design. First, there 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 [8], the scalability limit is not an important concern. Second, the system is unable to differentiate traffic. Thirdly the gate requires specialized hardware and introduces an additional delay. It is preferable to eliminate the gate device and allow the firewalls to operate independently.

3.3 Independent Function-Parallel Architecture

As described in the previous subsection, a function-parallel system consists of an array of firewalls where arriving packets are duplicated and policy rules are distributed. Each firewall processes an arriving packet using its local policy and a gate device is required to ensure integrity is maintained. However, it is possible to allow the firewalls to operate independently, thus eliminating the gate device and any need for inter-firewall communications.

Consider a function-parallel system consisting of m firewalls that enforces a comprehensive security policy R. Each firewall j in the array has a local comprehensive policy R_j that is a portion of the security policy R. Therefore, each firewall has a local accept set A_j and a deny set D_j. Integrity will maintained without a gate device if rules are distributed such that a packet $d \in D$ is dropped by all firewalls, while a packet $a \in A$ is accepted by only one firewall. This is more formally stated in the following theorem.

Theorem 1. *An array of m firewalls arranged in a function-parallel fashion enforcing a comprehensive policy R can operate independently and maintain integrity if policy rules are distributed such that: each local policy is comprehensive, $\bigcup_{j=1}^{m} A_j = A$, and $\bigcap_{j=1}^{m} A_j = \emptyset$.*

Proof. The first requirement, comprehensiveness, ensures each local policy will either accept or deny a packet ($\bigcup_{j=1}^{m} U_j = \emptyset$). The second requirement $\bigcup_{j=1}^{m} A_j = A$ indicates that collectively the system will accept only the packets accepted by the policy R. The last requirement, $\bigcap_{j=1}^{m} A_j = \emptyset$, ensures multiple firewalls will never accept the same packet (no overlaps in the local accept sets), therefore only one copy of a packet will be accepted. As such, the integrity of the policy R is maintained by the parallel firewall.

An example distribution of the policy given in figure 1 across an array of two independent firewalls is shown in figure 2(c). In this case, the local policy of the upper firewall will accept only packets from the `210` and `180` address range, while the lower firewall will only accept packets from the `190` address range. Duplicating the deny all rule, r_6, is required to make the local-policies comprehensive. Other distributions are possible, such as distributing rules based on the protocol ($R_1 = \{r_1, r_2, r_5, r_6\}$ and $R_2 = \{r_3, r_4, r_6\}$) or destination ports ($R_1 = \{r_1, r_4, r_5, r_6\}$ and $R_2 = \{r_2, r_3, r_6\}$). Policy distribution can be done to balance the packet load (distribute popular rules across the array) or to achieve a certain QoS objective. Of course the number of distributions will depend on the original security policy, where fewer precedence edges allow more distributions.

Like the function-parallel system that relies on a gate device, the independent function-parallel system can manage state information since a packet is sent to every firewall. However, allowing the firewalls to operate independently has several important unique advantages. First, the elimination of the gate device causes the function-parallel design to be compatible with a variety of firewall devices since specialized equipment is not needed. Second, the independent function-parallel system will have lower processing delays than an equivalent data-parallel system or a function-parallel system with a gate device. Third, local-policies can be designed to process certain types of traffic on certain firewalls, yielding the ability to provide service differentiation which is an important component for maintaining QoS requirements.

Although the system has many significant advantages, it is not redundant. Integrity will be lost if a firewall fails since a portion of the policy (local accept set) will not be available. Fortunately, loss of a firewall will only result in a more conservative policy (fewer packets accepted), which is better than the previous function-parallel design with gate device. Redundancy can be provided by duplicating the local policy to another firewall. As done in [1], firewalls can be interconnected to determine if redundant rules should be processed.

4 Experimental Results

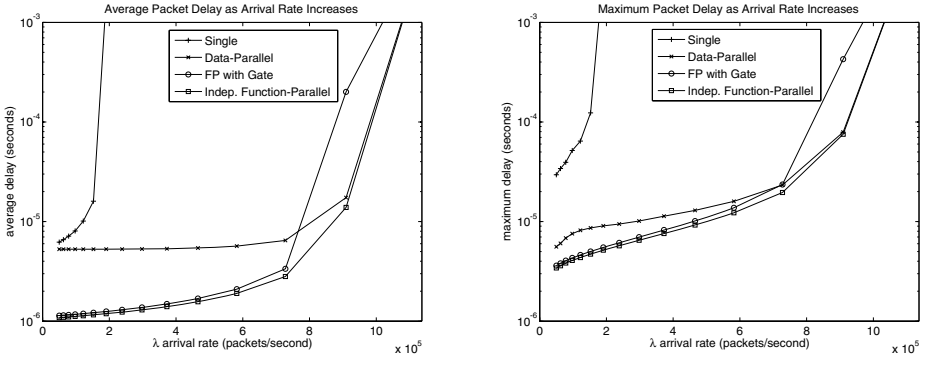
The performance of a traditional single firewall, the data-parallel firewall, and the function-parallel firewall (with gate device and independent) was measured under realistic conditions using simulation. Firewalls were simulated to process 6×10^7 rules per second, which is comparable to current technology.

For each experiment the parallel designs always consisted of the same number of firewalls. The gate device delay was equivalent to processing three firewall

rules. Short-circuit evaluation was simulated for the gated design, where the firewalls in the array are notified to stop processing a packet once the appropriate match was determined. 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 [9,8]. Rules were distributed for the function-parallel design such that no inter-firewall dependency edges existed, and if possible, more popular rules were located at the top of the local-policies. This distribution ensures integrity is maintained.

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.



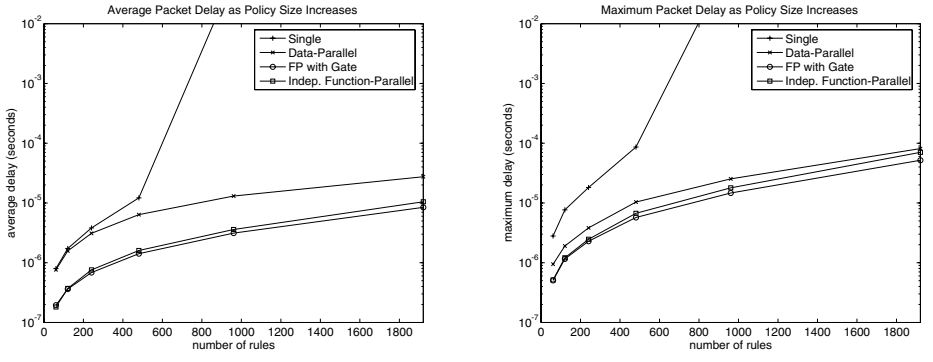
(a) Average delay. (b) Maximum delay.

Fig. 3. Packet delay as arrival rate increases. Parallel designs consisted of five firewalls.

The impact of increasing arrival rates on the four firewall designs is shown in figure 3. In this experiment, each system implemented the same 1024 rule firewall policy [8] and both parallel designs consisted of five firewalls. The arrival rate ranged from 5×10^3 to 1×10^6 packets per second (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 4.0 times lower than the data-parallel design, while the independent function-parallel design average delay was 4.3 times lower. This is expected because each firewall in the function-parallel design is used to inspect arriving packets regardless of the arrival rate. 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 34% lower than the data-parallel design, while the independent function-parallel design was 38% lower.



(a) Average delay. (b) Maximum delay.

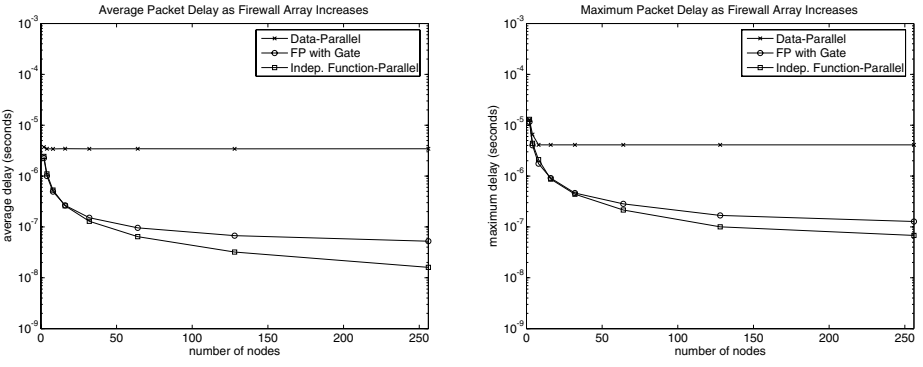
Fig. 4. Packet delay as rule number increases. Parallel designs consisted of five firewalls.

The effect of increasing the policy size (number of rules) for the four 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 five firewalls. Policies ranged from 60 to 3840 rules.

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 4.12 times lower than the data-parallel design, while the independent firewall was 3.79 times lower. This slight difference is primarily due to short-circuit evaluation, where the gate informs firewalls to stop processing a packet once the appropriate match is found. However this is only a marginal gain given the inter-firewall communication and specialized hardware required for short-circuit evaluation.

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 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 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



(a) Average delay. (b) Maximum delay.

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

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.

5 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 in a high-speed environment or if traffic requires specific network Quality of Service (QoS). Unfortunately, the firewall can quickly become a bottleneck given increasing traffic loads and network speeds. Therefore, new firewall designs are needed to meet the challenges associated with the next generation of high-speed networks.

This paper introduced a scalable firewall architecture consisting of multiple independent firewalls, where 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 (the parallel design and a traditional single firewall always reach the same decision for any packet) and independence (no inter-firewall communication required) were introduced. Simulation results showed the architecture achieved a processing delay significantly 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 and can be implemented with currently available technology. Therefore, the function-parallel firewall architecture is a scalable solution that can provide better performance and more capabilities.

While the function-parallel firewall architecture is very promising, several open questions exists. For example given the need for QoS in future networks,

it is important to develop methods for distributing 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. 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. Benecke, C.: A parallel packet screen for high speed networks. In: Proceedings of the 15th Annual Computer Security Applications Conference. (1999)
2. Goddard, S., Kieckhafer, R., Zhang, Y.: An unavailability analysis of firewall sandwich configurations. In: Proceedings of the 6th IEEE Symposium on High Assurance Systems Engineering. (2001)
3. Paul, O., Laurent, M.: A full bandwidth ATM firewall. In: Proceedings of the 6th European Symposium on Research in Computer Security ESORICS'2000. (2000)
4. Fulp, E.W., Farley, R.J.: A function-parallel architecture for high-speed firewalls. In: Proceedings of the IEEE International Conference on Communications. (2006)
5. Ziegler, R.L.: Linux Firewalls. second edn. New Riders (2002)
6. Ranganath, V.P., Andresen, D.: A set-based approach to packet classification. In: Proceedings of the IASTED International Conference on Parallel and Distributed Computing and Systems. (2003) 889–894
7. Culler, D.E., Singh, J.P.: Parallel Computer Architecture: A Hardware/Software Approach. Morgan Kaufman (1999)
8. Wool, A.: A quantitative study of firewall configuration errors. IEEE Computer **37**(6) (2004) 62 –67
9. Leland, W.E., Taqqu, M.S., Willinger, W., Wilson, D.V.: On the self-similar nature of ethernet traffic. IEEE Transactions on Networking **2** (1994) 1 – 15