High-Speed Packet Filtering Utilizing Stream Processors

Richard J. Hummel and Errin W. Fulp

Department of Computer Science, Wake Forest University, Winston-Salem, NC, USA

ABSTRACT

Parallel firewalls offer a scalable architecture for the next generation of high-speed networks. While these parallel systems can be implemented using multiple firewalls, the latest generation of stream processors can provide similar benefits with a significantly reduced latency due to locality.

This paper describes how the Cell Broadband Engine (CBE), a popular stream processor, can be used as a high-speed packet filter. Results show the CBE can potentially process packets arriving at a rate of 1 Gbps with a latency less than 82 μ-seconds. Performance depends on how well the packet filtering process is translated to the unique stream processor architecture. For example the method used for transmitting data and control messages among the pseudo-independent processor cores has a significant impact on performance. Experimental results will also show the current limitations of a CBE operating system when used to process packets. Possible solutions to these issues will be discussed.

Keywords: Network security, firewalls, stream processors, parallel processing

1. INTRODUCTION

As network speeds, traffic volumes, and the demand for Quality of Service (QoS) continue to increase, so must the performance of network firewalls (packet filters). Unfortunately a firewall can easily become a bottleneck in such a demanding environment since packets must be inspected and compared against complex rule sets and tables, a time-consuming process. Therefore new packet filtering approaches are needed for current and future high-speed environments.

Parallelization has been shown to provide a scalable solution for the next generation of high-speed firewalls.[1–3] These systems consist of an array of firewalls configured in either a data parallel[1] or function parallel fashion.[2,3] Scalability is achieved by adding additional firewalls to the array which will either improve throughput (data parallel) or reduce latency (function parallel). While these parallel systems have several performance advantages, the latest generation of parallel processors can offer further improvements in performance. For example due to locality, the intercommunication latency between cores in a parallel processor is significantly less than the latency associated with inter-device communication. Therefore the current generation of multicore processors offers an interesting platform for network firewalls.

The stream processor offers a multicore architecture and programming paradigm that is applicable to high-speed packet filtering. This paper will describe and utilize the Cell Broadband Engine (CBE), a stream processor co-developed by IBM, Sony, and Toshiba. The heterogeneous processor consists of one Power Processor Element (PPE) and eight Synergistic Processor Elements (SPE). These SPE's can operate independently and have high speed memory and a high-bandwidth bus interface available. Given this design, it is possible to implement data and function parallel versions of the firewall; however as will be described and shown experimentally, data parallel design is a more suitable approach.

Experimental results using a Sony PlayStation 3 (PS3) will indicate the CBE can filter packets (apply a firewall policy) at a rate of 1 Gbps with a latency less than 82 μ-seconds. Using the system to filter actual traffic, the system is only able to operate at 30 Mbps. As will be explained, this limitation is due to the current operating system. Specifically the bottleneck occurs when reading packets from the NIC, **not** packet filtering. While the CBE-based firewall is still impressive, this paper will describe solutions that will help the CBE to operate closer to its true capacity.

The remainder of this paper is structured as follows. Section 2 reviews the firewall policies and parallel designs. Section 3 describes stream processing and stream processors. Stream processing designs for parallel network firewalls is given in section 4. Section 5 provides experimental results of a CBE-based firewall and discusses limitations. Finally, section 6 summarizes this paper and discusses future work.

2. FIREWALLS AND PACKET FILTERING

Inspecting traffic sent between networks, a firewall provides access control, auditing, and traffic control based on a security policy.[4–6] The security policy is a list of ordered rules that defines the action to perform on matching packets. A rule can be represented as a 5-tuple (protocol, IP source address, source port number, IP destination address, destination port number). Fields can be fully specified or contain wildcards `'*'` in standard prefix format. For example the prefix `192.*` would represent any IP address that has `192` as the first dotted-decimal number. In addition to the prefixes, each filter rule has an action, which is to accept or deny. An accept action passes the packet into or from the secure network, while deny causes the packet to be discarded. Rules are applied in order to every arriving packet until a match is found; otherwise, a default action is performed.[5,6] This is considered a *first-match* policy and is the default behavior for the majority of firewall systems including the Linux firewall implementation `iptables`.[7] A packet matches a rule when every tuple of the packet is a subset of the corresponding tuple in the rule.

2.1 Parallel Firewall Designs

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. Given the array of firewalls, different systems can be created, 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.[8] In contrast, *function-parallel* designs distribute the policy rules across the firewalls.

A data-parallel firewall architecture consists of an array of identically configured firewalls.[1] Distributing packets across the array allows a data-parallel firewall to increase system throughput (number of packets processed per unit time) as compared to a traditional (single machine) firewall.[1] Furthermore, increased throughput is easily achieved with the addition of firewalls; therefore, this approach is very scalable.

Unlike the data-parallel model which distributes packets, the function-parallel design distributes policy rules across the same firewall array.[2] When a packet arrives to the function-parallel system it is forwarded to every firewall. Each firewall processes the packet using its local policy. Since the local policies are smaller than the original, the processing delay is reduced as compared to a traditional firewall. However, the policy distribution must be done such that only one firewall will accept packets accepted by the original policy. These necessary policy distribution techniques are discussed in.[9,10]

3. STREAM PROCESSORS

Stream processors allow an application programmer to access a large number of processing cores indirectly through the use of kernel functions. Stream processors excel when computational complexity, data parallelism, and data locality are involved.[16] Given a predetermined input and output buffer size, the stream processor control unit manipulates the processing cores, bus control, and performs memory management without programmer intervention. Because memory transfers are expected to happen in bulk, stream processors are optimized for high bandwidth, not latency. There are similarities between kernel functions and thread level parallelism, however, stream processors have the ability to create and manage many times the number of kernels than a generic CPU can threads.[16,17] While mimicking function parallelism within stream processors is possible, it is not an inherent property by design.

The IBM/Sony/Toshiba Cell Broadband Engine (CBE) is a unique, media based, stream processor capable of 204 Gflops. It is a single chip containing a dual threaded 64-bit PowerPC core and eight SIMD based, single precision, coprocessors. The main element is called the Power Processing Unit (PPU) while the coprocessors are called Synergistic Processing Units (SPU) (also referred to individually as Synergistic Processing Elements (SPE)). The processor also has an integrated high-speed memory, high-bandwidth bus interface. The PPE and the SPE's communicate through an internal high-speed Element Interconnect Bus (EIB) which has a peak rate of 204 GBps.

The required work must be correctly managed across the PPU and SPE's to fully realize the CBE potential. The processor's communication channels allows the transmission of data among the computation elements. Each SPE has a small local store and a Direct Memory Access (DMA) controller that allows the high-bandwidth

transmission of data between the local store and main memory. Signaling between the SPE's and PPE occurs via mailboxes. Note communication between SPE's must occur using the PPE as an intermediary.

Using the array of SPE's, the CBE is capable of processing packets using two forms of parallelism: data and function parallel. In both modes, the PPU forwards packets to each SPE where the PPU is the policy enforcer and the SPE's are the policy resolvers. Processing packets in *data parallel* is achieved by sending a different packet to each SPE. Each SPE uses an identical kernel function and all resolve using identical policy sets. *Function parallel* is achieved by sending the same packet to each SPE. Each SPE uses an identical kernel function, however, each SPE resolves using only a unique portion of the entire policy set. The policy enforcer / policy resolver model was used in part because the SPE's are not capable of reading packets from the network interface card. While the SPU service based model is possible, where the SPE's directly service the network adapter, it requires extensive modification to the operating system and time did not allow the use of this approach.

4. STREAM PROCESSING TECHNIQUES

The driving requirements of the parallel firewall under test are efficiency, support of both function and data parallel modes, and reconfigurable policy sets. Data and function parallel modes of operation in our setup are compile time options, so mode switching during operation is not possible. Since both modes of operation are supported, neither is fully optimized and therefore potential bandwidth remains. Policy sets are loaded one time from a text file during program initialization.

Program division occurs along PPU and SPU lines. The PPU is responsible for packet capture and transferring packets to each SPE for policy resolution. The PPU performs policy enforcement during packet injection based on the returned SPE results. For efficiency, elegance, and simplicity's sake the SPE kernel is identical in our setup for both data and function parallel modes of operation.

4.1 DMA vs. Mailboxes

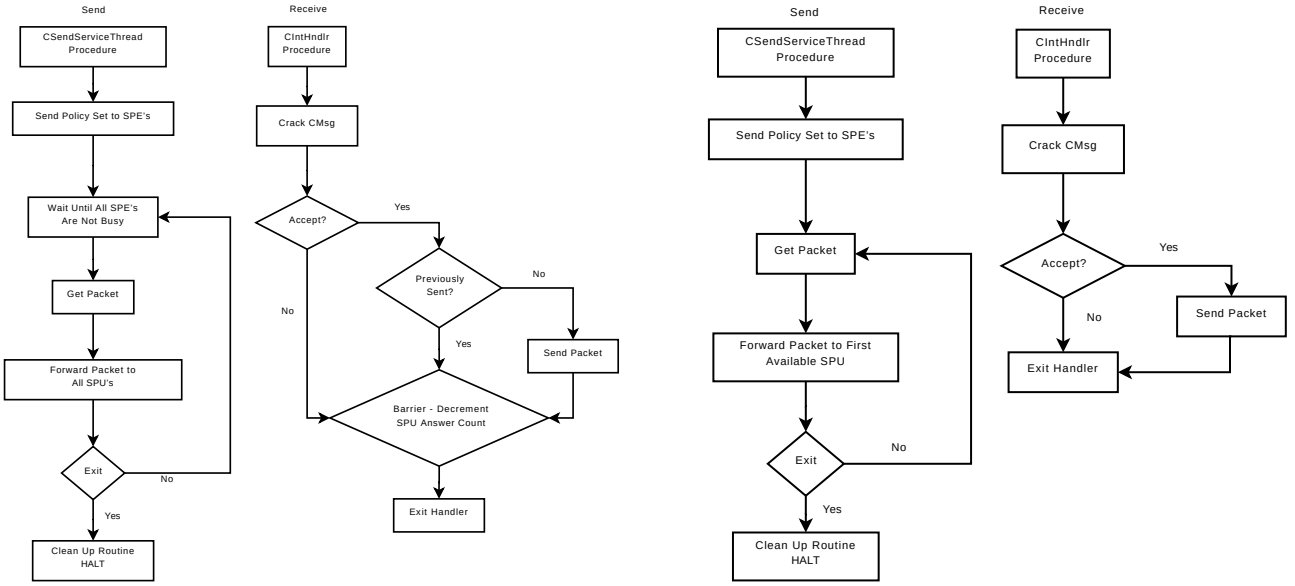
The CBE architecture presents two possible ways to process packets, singularly or in groups. The popular DMA transfer model yields high bandwidth at the cost of latency, however, it forces packet processing at the group level in order to maintain a reasonable DMA transfer costs to SPE processing ratio. Group processing creates local buffers in both the PPU and the memory starved SPE. Other considerations involving DMA transfers include transferring the packets from the network interface into a local buffer of custom data packet structures in order to obtain proper data alignment for the DMA transfer, the cost of memory transfers as a result of the DMA itself, and associated memory transfers in reverse order once the SPE has resolved the policies.

The simplistic model is one packet at a time. Using DMA transfers is not an efficient solution, however, mailboxes are as there are no memory transfers, custom data structures, or any "off chip" access sequences. Rather than transferring an entire packet, our method uses mailboxes to only send the required information needed for SPE policy resolution: source IP, destination IP, and protocol. Once the packet is received, the data items are passed directly from the pcap packet into the mailbox using direct memory references, a specification of the SDK API mailbox function call. Return messages are sent from the SPU to the PPU using interrupt driven mailboxes ensuring a disconnect between the PPU based sending and receiving threads.

4.2 Function Parallel

There are three primary packet scheduling methods in function parallel: lock-step, signaling, and open-loop. *Lock-step* based packet scheduling is when the sending process waits for all subordinate processes to return an accept, deny, or drop message before transmitting the packet. *Signaling* based packet scheduling is when the main process waits for one of the subordinate process to signal an accept before transmitting the packet. The default action is to deny or drop the packet. *Open-loop* based packet scheduling is when a packet distribution process forwards all packets to subordinate processes. A combining process receives accepted packets from subordinate processes, without feed back to the original process, and recombines them into a final output stream.

If the system consists of only one CBE, then only lock-step and signaling based processing are good choices. Open-loop based packet processing requires sophisticated queuing techniques that become reasonable to implement when two CBE's are available in order to distribute the required computational load as would be found in



(a) Function parallel processing.

(b) Data parallel processing.

Figure 1: PPU parallel processing flow charts.

a distributed inter-device firewall. Our firewall uses the lock-step method of packet processing and two features found in signaling based processing. Included is the use of a default deny policy model and transmission of the packet upon the first accept message.

Figure 1(a) diagrams the overarching function parallel program flow. The PPU application creates two threads. The send thread is responsible for sending the rule set to each SPE and then begins the packet processing loop. Processing begins by waiting on all SPE's to become idle, after which a single packet is then received via pcap and forwarded to each SPE while simultaneously marking each SPE as busy.

The receive thread implements the interrupt mailbox service routine (IMSR). The IMSR begins by validating the incoming message and then begins book-keeping. If the action is accept, the IMSR immediately attempts to inject the packet back into the network as soon as possible if it hasn't already done so. The second book-keeping act is to mark the SPE as idle and decrement the total running number of answers before exiting. The total number of SPE answers variable forms a lock-step barrier.

4.3 Data Parallel

There are three primary packet scheduling methods in data parallel: round-robin, fair queuing, and max-throughput. *Round-robin* based packet scheduling assigns a packet to the next available SPU in successive order. *Fair queuing* based packet scheduling uses statistical multiplexing and buffering to allow n data flows to fairly share the link capacity. Fair queuing resembles round-robin, however the first priority is maximizing the minimum data rate, the next highest priority is the next highest data rate, and so forth n times. *Max-throughput* based packet scheduling gives scheduling priority to the least expensive data flows in terms of consumed network resources per transferred amount of information.

For simplicity and in the light of support of multi-mode operation, round-robin packet distribution was chosen. Figure 1(b) diagrams the overarching data parallel program flow. The send thread processing loop receives a single packet and then forwards it to the first available SPE while simultaneously marking the SPE as busy.

The receive threads IMSR validates the incoming message and forwards the packet if the policy resolution response is an accept. Once the message is sent, the sending SPE is marked as idle and the IMSR exits.

5. PERFORMANCE EVALUATION

The performance the CBE-based packet filter was measured using an Sony PlayStation3 (PS3). The PS3 system consisted of a 3.2 GHz CBE processor, 256MB XDR Main RAM, a single 1000 BASE-T Ethernet adapter, and 60 GB hard drive. The PS3 has only six SPE's available for general use as opposed to all eight on other platforms, as one SPE is used by the system software for video display and the eighth SPE is defective on all PS3's. The operating system was Fedora Core 6 with a custom compiled kernel, version 2.6.23, to load only the required system modules. The firewall software was written in mixture of C and assembler, which is common for CBE development. The firewall software relied on lib pcap for network input and output. Lib pcap is a platform independent application programming interface (API) used for network packet-capturing and packet-injection. The primary reason for its use is because pcap resolves the issue of injecting packets back into the network which are not within the NIC's network address space.

The performance of the CBE-based packet filter was evaluated as traffic rate, number of rules, and number of SPE's increased. Two sets of experiments were conducted using either internally or externally generated packets. Experiments using internally generated packets were done to measure the latency and throughput of the CBE, and can be considered as the upper bounds on performance. Externally generated packets measured the actual performance of the firewall under realistic conditions. All experiments were performed in accordance to IETF RFC 3511 that describes firewall performance testing.[15]

5.1 Internal Packets

For this set of experiments packets were generated within the CBE, and will indicate the CBE theoretical performance when neither the networking subsystem or lib pcap are involved. There are a total of four experiments for data parallel and function parallel, each utilizing six SPU's. The policy size is set for each experiment (0, 1024, 2048, 4096) and the result is the maximum achieved bps and associated latency. The data rates listed were based on a 1500 byte payload. Table 1 suggests data parallel has a potential throughput of nearly a gigabit per second while Table 2 suggests function parallel has a potential of 55 Mbps.

Policy Size (number of rules)	Max Avg Throughput (Mbps)	Min Avg Latency (μ-sec)
0	939.5	81.5
1024	249.1	422.2
2048	131.6	771.4
4096	73.9	1472

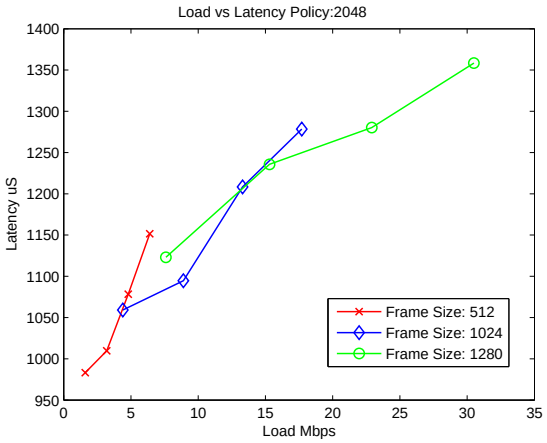
Table 1. Data parallel CBE results for increasing policy sizes using internally generated packets and 6 SPU's.

Policy Size (number of rules)	Max Throughput (Mbps)	Min Latency (μ-sec)
0	55.2	1000
1024	57.6	1577
2048	53.2	2420
4096	33.8	3487

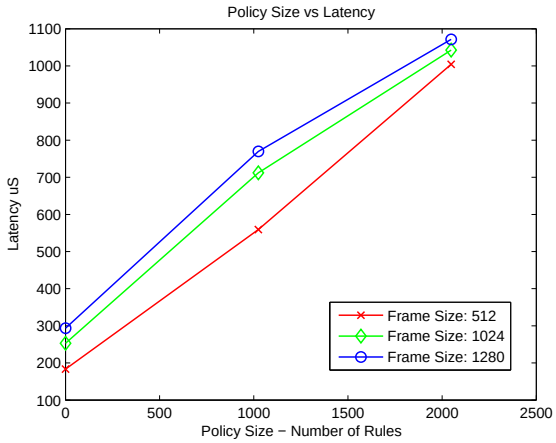
Table 2. Function parallel CBE results for increasing policy sizes using internally generated packets and 6 SPU's.

5.2 External Packets

Results from experiments using externally generated packets will show the actual performance of the CBE-based firewall. The network topology consisted of the PS3, a 1 Gbps switch, and a SmartBits SMB200. The SmartBits was equipped with two ML07710 100 Mbps cards and can determine the throughput and latency of networking



(a) Average latency as offered load increases.



(b) Average latency as policy (number of rules) increases.

Figure 2: Average packet delay for data parallel CBE packet filter and externally generated packets.

devices by sending packets at different rates. For these experiments the Smartbits sent packets to the PS3 packet filter. The PS3 read the packets using lib pcap, applied the policy, then sent accepted packets back to the Smartbits.

When the firewall was configured in a function parallel fashion, the maximum rate achieved with zero firewall rules was 390 Kbps with average latency of 235.5 μ-secs. Such poor performance demonstrates the lock-step version of function parallel is not a viable solution for a single CBE and did not warrant further exploration.

Data parallel configuration provided more promising results than the function parallel design. Figure 2(a) demonstrates throughput vs latency using a policy size of 2048 using different frame sizes. The maximum throughput of 30 Mbps and latency of 1.3 msecs was achieved, which is impressive for a software based packet filter and the given hardware platform. However, throughput using frame size of 512 bytes, when taken into consideration against frame sizes of 1024 and 1280, reveals pcap's packet processing limitations.

Figure 2(b) provides deeper insight into the CBE performance as latency between differing frame sizes are reasonably equivalent. The latency measurements for a single SPU, not shown, shadows 6 SPU's for each frame size, suggesting inefficiencies lie within pcaps ability to process packets. Figure 3 demonstrates throughput vs policy size for a maximum of 4096 rules, varying frame sizes, and SPU counts. Increasing the number of SPU's does increase throughput, therefore, the addition of two more SPU's in a fully functional CBE will further increase throughput. Figure 3 clearly illuminates the fact that smaller policy sets and differing packet sizes yields throughput in excess of 100 Mbps, of which, we were not able to fully measure with the given test equipment.

The difference between the internal and external packet results indicate there are limitations to using a CBE-based system as an actual packet filter. Despite the powerful feature of packet-injection, lib pcap is extraordinarily slow and forms a system bottle neck. Although the test system was a PlayStation 3, the limitations would be evident in any CBE-based system.[13] One solution is to use PF_RING to interface with the NIC, which provides significantly faster access to network data. However the current version of PF_RING only reads from the NIC and additional work is needed to allow it to write to the NIC. Another solution is to use a BSD-based operating system which is poll driven and provides faster access to network data.[14] Unfortunately a BSD port is currently not available for any CBE based board set. Another solution is to develop a new device driver to treat the filtered NIC's as serial ports, effectively by-passing all TCP stack based delays. The optimal solution is to use the SPE service based model and move the SPE's to ring 0 to directly service the filtered NIC's, however, this approach is the most difficult approach of all the suggestions.

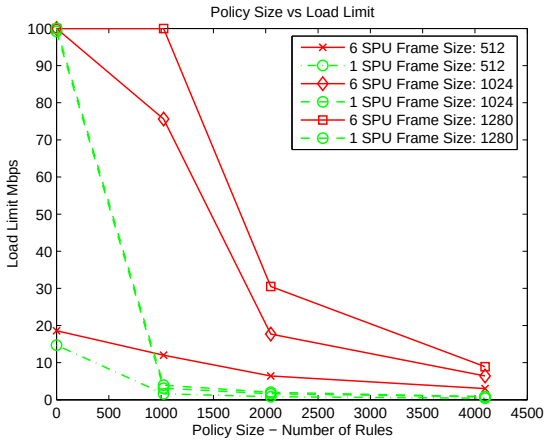


Figure 3: Maximum load possible as the policy size increases, using the data parallel CBE packet filter and externally generated packets.

6. CONCLUSIONS

Parallel firewalls are a scalable approach for providing high throughput, low latency firewalls. These systems consist of an array of processing elements configured in a data parallel or function parallel fashion. These systems have been implemented using multiple firewalls (multiple stand-alone systems), however the stream processor (a multicore architecture) offers an interesting implementation platform.

This paper described how the Cell Broadband Engine (CBE) can be used as a high-speed packet filter. The CBE is a popular stream processor that consists of a main Power Processor Unit (PPU) and eight Synergistic Processing Units (SPU). Given this design is possible to use the SPU's as an array of packet filters that are managed by the PPU. Both parallel designs are possible, but the data parallel approach translates best to this architecture. This is primarily due to the communication overhead associated with function parallel since the SPU's must operate in a *lock-step* fashion. Data parallel implementation allows the SPU's to operate more independently.

Experimental results using a Sony PlatStation3 (PS3) show the CBE can provide a high-speed, low latency packet filter. For example the data parallel packet filter was capable of processing packets at 1 Gbps with a latency less than 82 μ-secs. In these experiments packets were generated inside the processor. Performance is lower when the system reads and writes packets to the NIC. The maximum performance in this case is 30 Mbps with latency of 1.3 msec. The bottleneck is lib pcap, which has been documented by other networking applications. Possible solutions include using a modified version of PF_RING, porting a version of the BSD operating system to the CBE, or developing NIC drivers that provide direct access to network data. Each are areas of future work.

Other areas of future work in the networking related environment are intrusion detection and prevention systems (IDS/IPS). A Snort type system could benefit greatly from the CBE architecture because of its flexibility of SPU arrangements. The pipe-lining of packet processing where one SPE performs an initial match and subsequent validation is off loaded to the remaining SPE's, potentially in parallel, is one area. Packet pay-load analysis is another area. Finally, a mature CBE based packet filter optimized for function parallel and/or data parallel is an area of future work.

ACKNOWLEDGMENTS

The authors would like to thank Wake Forest University and GreatWall Systems, Inc. for their support. This research was funded by GreatWall Systems, Inc. via United States Department of Energy STTR grant DE-FG02-06ER86274. The views and conclusions contained herein are those of the author(s) 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] C. Benecke, "A parallel packet screen for high speed networks," in *Proceedings of the 15th Annual Computer Security Applications Conference*, 1999.

[2] E. W. Fulp and R. J. Farley, "A function-parallel architecture for high-speed firewalls," in *Proceedings of the IEEE International Conference on Communications*, 2006.

[3] E. W. Fulp, "An independent function-parallel firewall architecture for high-speed networks (short paper)," in *Proceedings of the International Conference on Information and Communications Security*, 2006.

[4] S. M. Bellovin and W. Cheswick, "Network firewalls," *IEEE Communications Magazine*, pp. 50–57, Sept. 1994.

[5] R. L. Ziegler, *Linux Firewalls*, 2nd ed. New Riders, 2002.

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

[7] 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.

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

[9] E. W. Fulp, M. R. Horvath, and C. Kopek, "Managing security policies for high-speed function parallel firewalls," in *Proceedings of the SPIE International Symposium on High Capacity Optical Networks and Enabling Technology*, 2006.

[10] M. R. Horvath, E. W. Fulp, and P. S. Wheeler, "Policy distribution methods for function parallel firewalls," in *Proceedings of the IEEE International Conference on Computer Communications and Networks*, 2008.

[11] Ryan J. Farley. Parallel Firewall Designs for High-Speed Networks. Masters Science Thesis, Wake Forest University Computer Science Department, 2005.

[12] Van Jacobson, Craig Leres and Steven McCanne, all of the Lawrence Berkeley National Laboratory, University of California, Berkeley, CA. PCAP MAN Pages. `http://www.tcpdump.org/pcap3_man.html`. September 23, 2007.

[13] PF_RING on-line documentation. `http://www.ntop.org/ PF_RING.html`. ntop.org. 1998-2007

[14] Luca Deri. Improving Passive Packet Capture: Beyond Device Polling. `http://luca.ntop.org/`. ntop.org. January 8, 2008.

[15] B. Hickman, D. Newman, S. Tadjudin, T. Martin. *RFC 3511 - Benchmarking Methodology for Firewall Performance*. Internet RFC/STD/FYI/BCP Archives, April 2003.

[16] Scott Rixner. *Stream Processor Architecture*. Kluwer Academic Publishers, Boston, MA, 2001.

[17] Brucek Khailany *The VLSI Implementation and Evaluation of Area and Energy-Efficient Streaming Media Processors*. PhD Dissertation, Stanford University. 2003.