Architecture for a Hardware Based, TCP/IP Content Scanning System

David V. Schuehler James Moscola John Lockwood
Applied Research Laboratory, Washington University
{dvs1, jmm5, lockwood}@arl.wustl.edu

Abstract

Hardware assisted intrusion detection systems and content scanning engines are needed to process data at multi-gigabit line rates. These systems, when placed within the core of the Internet, are subject to millions of simultaneous flows, with each flow potentially containing data of interest. Existing IDS systems are not capable of processing millions of flows at gigabit-per-second data rates.

This paper describes an architecture which is capable of performing complete, stateful, payload inspections on 8 million TCP flows at 2.5 gigabits-per-second. To accomplish this task, a hardware circuit is used to combine a TCP protocol processing engine, a per flow state store, and a content scanning engine.

1 Introduction

TCP is the workhorse protocol of the Internet today. Most of the data transmitted through the Internet transits the network using the TCP/IP protocol. In order to monitor, capture, filter, or block traffic on the Internet, it is necessary to be able to directly process the TCP protocol in hardware. Because TCP is a stream oriented protocol that operates above an unreliable datagram network, there are complexities associated with processing TCP data.

High speed Network Intrusion Detection System (NIDS) are needed to detect and prevent several types of threats. Further, other types of content scanning systems have been proposed that detect and prevent the transmission of confidential material to or from external entities. In order to use these new features in backbone networks, network throughput must not be inhibited.

Gilder's law states that bandwidth grows at least three times faster than computing power [8]. As the gap between network bandwidth and computing power widens, improved techniques are needed to monitor and filter network traffic without limiting throughput.

In order to address these issues, an architecture is presented which supports content scanning and flow blocking for millions of flows at gigabit line rates. The rest of this paper consists of four sections. The first provides related work. The second describes background information. The third explains the proposed system architecture. The final section makes concluding remarks.

2 Related Work

By their very nature, Intrusion Detection Systems (IDS) need to perform deep packet inspections on all traffic traversing through the network. This task is difficult when data rates are high and there are large numbers of simultaneous flows that need to be tracked. Software IDS solutions, such as Snort [11], work well when aggregate bandwidth rates are low.

It is difficult to implement an external monitor which is capable of tracking the state of a TCP connection. Bhargavan *et al* discuss the complexities associated with tracking various properties of a protocol utilizing language recognition techniques [1]. General solutions to this problem may become unbounded with respect to time and space.

The task of reassembling and monitoring flows, as is needed for an Intrusion Detection System, is even further complicated when direct attempts are being made to evade detection. Handley *et al* expound on this topic [5]. One such technique for evading detection, would be the modification of an end system protocol stack such that TCP retransmissions contain different content than original data transmissions.

A passive monitoring system (IPMON) has been developed which is able to capture and accurately time stamp packets at data rates up to OC-48 (2.5Gbps) [3]. Data captured by multiple monitoring systems in a wide area network can be correlated utilizing the highly accurate time stamps. Optical splitters are employed to deliver a copy of the network traffic to the monitoring station. The system saves the first 44 bytes of each packet are stored. Analysis of the captured data is performed out-of-band.

Network World Fusion tested 6 commercially available gigabit Intrusion Detection Systems by sending 28 attacks along with 970Mb/s of background traffic [10]. After system tuning, only one system was able to detect all 28 of their attacks while processing data on a gigabit Ethernet link. In general, software based systems are not capable of performing regular expression matching at gigabit rates.

Previous work exists in the area of string matching on FPGAs, most notably by Sidhu and Prasanna [13] and by Franklin, Carver and Hutchings [4]. The work by Sidhu and Prasanna was primarily concerned with minimizing the time and space required to construct Nondeterministic Finite Automata (NFAs). They run their NFA construction algorithm in hardware as opposed to software. To perform string matching, Franklin, Carver and Hutchings followed with an analysis of this approach for the large set of regular expressions found in a Snort database [11].

3 Background

Previous research performed at the Washington University Applied Research Laboratory led to the development of a reconfigurable hardware platform. Layered protocol wrappers and a regular expression string matching circuit have been developed which operate in reconfigurable hardware.

The Field-Programmable Port Extender (FPX) is a hardware platform developed to perform high speed packet processing [7]. The FPX card places a Xilinx Virtex 2000E Field Programmable Gate Array (FPGA) in the data path of a multi-gigabit network. Circuit designs can be programmed into this device which operate on the network traffic flowing through the device. The device utilizes a 32 bit wide data path and when clocked at 80MHz, it is capable of processing data at OC-48 (2.5Gb/s) line rates.

The Layered Protocol Wrappers [2] are a set of circuit designs which provide protocol processing in reconfigurable logic devices. The wrappers include an ATM Cell Wrapper, an AAL5 Frame Wrapper, and an Internet Protocol (IP) Wrapper. These wrappers provide lower layer protocol processing for the TCP architecture

3.1 TCP-Splitter

The TCP-Splitter is a circuit design which supports the monitoring of TCP data streams. A consistent byte stream of data is delivered to a client application for every TCP data flow that passes through the circuit. The TCP-Splitter accomplishes this task by tracking the TCP sequence number along with the current flow state. Selected out-of-order packets are dropped in order to provide the client application with the full TCP data stream without requiring large stream reassembly buffers.

The dropping of packets to maintain an ordered flow of packets through the network has the potential to adversely affect the overall throughput of the network. Analysis of out-of-sequence packets in Tier-1 IP backbones was performed by Jaiswal *et al* [6]. They noted that approximately 95% of all TCP packets were detected in proper sequence. Network induced packet reordering accounted for a small fraction of out-of-sequence packets, with the majority resulting from retransmissions due to data loss. Greater than 86% of all TCP flows observed, contained no out-of-sequence packets.

The first implementation of the TCP-Splitter [12] stored 33 bits of state information for each active flow in the network. By utilizing a low latency SRAM module, 256k simultaneous flows were supported. That TCP-Splitter circuit utilized 32 bit wide data path of the FPX card and could operate at 100MHz. At that clock rate, a maximum throughput of 3.2Gbps was supported.

3.2 Content-Scanning Engine

The content-scanning engine is a hardware module that is capable of scanning the payload of packets for a set of regular expressions [9]. To accomplish this task, the content-scanning engine employs a set of Deterministic Finite Automata (DFAs), each searching in parallel for one of the targeted regular expressions. Upon matching the content of a network data packet with any of these regular expressions, the content-scanner has the ability to either allow the data to pass or to drop the packet. The content-scanning engine also has the ability to generate an alert message that can be sent to a log server when a match is detected in a packet. The alert message contains the source and destination addresses of the matching packet along with a list of regular expressions that were found in the packet. The content-scanner currently operates at speeds of up to 2.5 Gbps.

4 Architecture

The new TCP based content scanning engine integrates and extends the capabilities of the TCP-Splitter and the Content-Scanning Engine. An overview of this new engine can be seen in figure 1. Data flow is from left to right. IP packets are passed into the TCP Protocol Processing Engine from the lower layer protocol processing engines contained within the network switch. An input packet buffer has been added to provide a limited amount of packet buffering when there are downstream processing delays. The TCP Protocol Processing Engine validates packets and classifies them as part of a flow. Packets along with the associated flow state information are passed onto the Packet Routing module where the packets are routed either to the Content Scanning Engines or the Flow Blocking module. Multiple Content Scanning Engines evaluate regular expressions against the TCP data stream. Packets returning from the Content

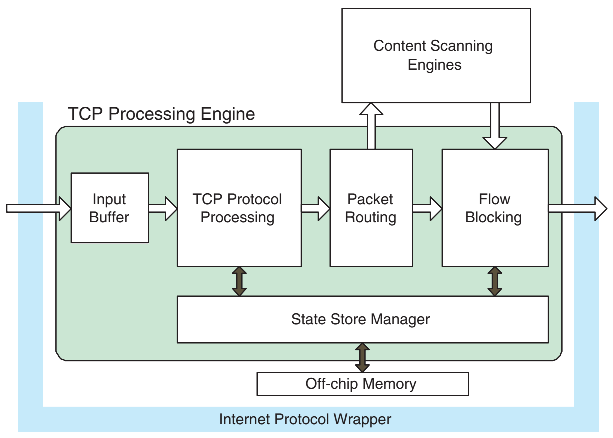

Figure 1. System Overview

Scanning Engines are passed to the Flow Blocking module where application-specific state information is stored and flow blocking is enforced. Packets that are not blocked are passed back to the network switch which forwards them on toward their end destination.

4.1 Design Requirements

This work overcomes some of the limitations associated with the original TCP-Splitter circuit. These limitations include (1) the inability to resolve hash collisions in the state store, (2) the limited size of the state store, and (3) the small amount of state information stored for each flow.

Gracefully handling hash table collisions is a difficult problem for real-time network systems of this nature. An efficient method for dealing with hash collisions is to have the new flow age out the previous flow whenever a collision occurs. This type of action leads to the incomplete scanning of TCP flows because the context scanning engine will loose the context information of the previous flow when it encounters a new flow with the same flow identifier. To ensure all flows are properly monitored, a linked list of flow state records can be chained off of the appropriate hash entry. The advantage to this approach is that all flows that encounter hash collisions in the state store can be fully monitored. The major drawback to this approach is that the time required to traverse a linked list of hash bucket entries can be excessive. The delay caused in retrieving flow state information can adversely affect the throughput of the system and lead to data loss. Another drawback of linked entries in the state store is the need to perform buffer management operations. This induces additional processing overhead into a system which is operating in a time critical environment. The State Store Manager overcomes this problem by limiting the length of the chain to a constant number of entries.

A hashing algorithm which produces an even distribution across all hash buckets is important to the overall efficiency of the circuit. Initial analysis of the flow classification hashing algorithm used for this system was performed against packet traces available from the National Laboratory for Applied Network Research[1]. With 26,452 flow identifiers hashed into a table with 8 million entries, there was a hash collision in less than .3% of the flows.

Additional features of the enhanced TCP processing circuit are proposed to support the following services:

Flow Blocking This will allow a flow to be blocked at a particular byte offset within the TCP data stream.

Flow Unblocking A previously disabled flow can be re-enabled and data for a particular flow will once again be allowed to pass through the circuit.

Flow Termination A selected flow will be shut down by generating a TCP FIN packet.

Flow Modification This would potentially provide the ability to sanitize selected data contained within a TCP stream.

4.2 Flow State Store

At any given moment, a high speed router may be forwarding millions of individual traffic flows. To support this large number of flows along with a reasonable amount of state information stored for each flow, a 512MB Synchronous Dynamic Random Access Memory (SDRAM) module can be utilized. The memory interface to this memory module has a 64 bit wide data path and supports a maximum burst length of eight operations. By matching our per flow memory requirements to the burst width of the memory module, the memory bandwidth can be optimized. Storing 64 bytes of state information for each flow enables the memory interface to match the amount of per flow state information with the amount of data in a burst transfer to memory. This configuration provides support for eight million simultaneous flows. Assuming \$100 as the purchase price for a 512MB SDRAM memory module, the cost to store context for eight million flows is only .00125 cents per flow or 800 flows per penny.

Of the 64 bytes of data stored for each flow, the TCP processing engine utilizes 32 bytes to maintain flow state and memory management overhead. The additional 32 bytes of state store for each flow will hold the application-specific data for each flow context. The layout of a single entry can be seen in figure 2.

The source and destination IP addresses along with the source and destination TCP ports are hashed into a 23 bit value. This hash value is used as a direct index to the first

[1]Sponsored by the National Science Foundation Cooperative Agreement No. ANI-9807479

63 32	31 0
Flow Id	Hash Value
Flags/Next Flow Id	Source IP
Sequence #	Dest IP
Blocking Sequence #	Ports
Application Data	Application Data
Application Data	Application Data
Application Data	Application Data
Application Data	Application Data

Figure 2. Flow State Record Layout

entry in a hash bucket. The format of the record allows the hash table to contain 4 million records at fixed locations and an additional 4 million records to form a linked list of records for hash collisions. The use of linked list records enables the storing state information for multiple flows that hash to the same bucket. To ensure that the system is able to maintain realtime behavior, the number of link traversals is constrained by a constant.

The State Store Manager can cache state information utilizing on-chip block RAM memory. This can provide faster access to state information for the most recently accessed flows. A write-back cache design provides for improved performance.

4.3 Stream-Based Content Scanning

The new content-scanner processes streams from the TCP-Splitter to allow a match to span across multiple packets. The use of the TCP-Splitter requires that the content scanner process interleaved flows. Because each content scanner only holds the state of one flow, it needs to be able to save and restore the current state of a flow and perform a context switch whenever a new flow arrives. When a packet arrives at the content scanner on some flow, the content scanner must restore the last known matching state for that flow. When the content scanner has finished processing the packet, it must then save the new matching state of the flow which can be done by using the state store resources of the TCP processing circuit.

Figure 3 shows the design of the content-scanning engine combined with the TCP-Splitter. When a new packet arrives from the TCP-Splitter, it arrives concurrently with a flow ID and state information for that context. Once the state of the search engine is loaded, the content scanner can process the packet. If no matches are found in the packet, then the packet is allowed to pass through the module. If a match is discovered in the packet, then the packet may be dropped or the whole flow may be blocked. The content scanner also has the ability to send out an alert message if a match occurs. The format of the alert message is a UDP datagram which is also used for logging events in the system.

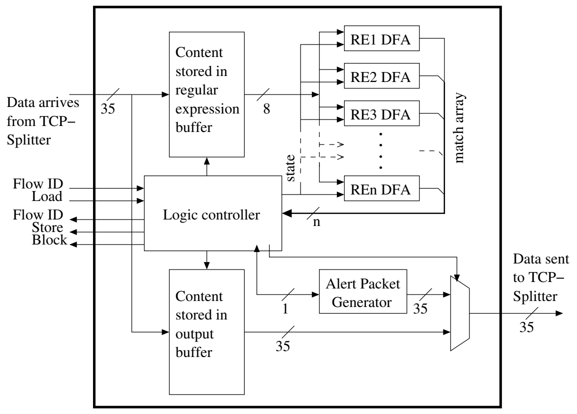
Data arrives from TCP–Splitter
35
Content stored in regular expression buffer
8
RE1 DFA
RE2 DFA
RE3 DFA
REn DFA
match array
state
Flow ID Load
Flow ID Store
Block
Logic controller
n
Alert Packet Generator
1
35
Content stored in output buffer
35
Data sent to TCP–Splitter
35

Figure 3. Block Diagram of the Content Scanner

As shown in figure 4, the overall throughput of the content scanner increases by putting four scanning engines in parallel and processing four flows concurrently. Incoming packets are dispatched to one of the scanning engines based on a hash of a flow ID provided by the TCP-Splitter. By dispatching packets in this fashion, the possibility of hazards that may occur when two scanners are processing packets from the same flow simultaneously can be eliminated.

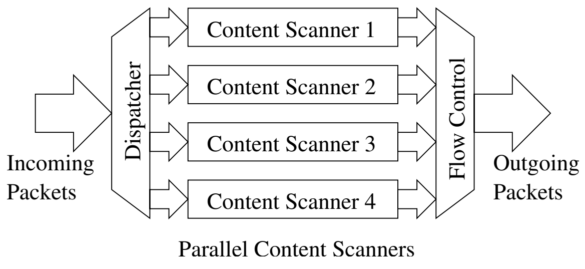

Figure 4. Arrangement of Parallel Scanner

4.4 TCP Protocol Processing

As was shown in figure 1, data is received on the left from the Internet Protocol Wrappers [2] and passed into an input buffer. Frames are buffered here in the event that there is congestion in the TCP Protocol Processing engine. This can occur at instants when there are hash table collisions and the State Store Manager has to walk through a linked list in order to locate the proper flow context.

From the input buffer, IP frames are passed to the TCP Protocol Processing Engine which is shown in figure 5. An input state machine tracks the processing state within a single packet. Data is forwarded to (1) a Frame FIFO which stores the packet, (2) a checksum engine which validates the TCP checksum, and (3) a flow classifier. Once the flow classifier has computed a hash value for the packet, information is passed to the State Store Manager which retrieves the state information associated with the particular flow. Results are written to a Control FIFO and the state store is updated with the current state of the flow.

An Output State Machine reads data from the Frame and Control FIFOs and passes it to the packet routing engine. Most traffic flows through the Content Scanning Engines where the data is scanned. Packet retransmissions bypass the Content Scanning Engines and are sent directly to the Flow Blocking module.

Data returning from the Content Scanning Engines is passed to the Flow Blocking module. At this stage, the per flow state store is updated with the latest application-specific state information. If flow blocking is enabled for a flow, it is enforced at this time. The sequence number of the packet is compared with the sequence number where flow blocking should take place. If the packet meets the blocking criteria, it is dropped from the network at this point. Packets that are not dropped are passed on to the outbound Protocol Wrapper.

The State Store Manager is responsible for processing requests to read and updates to write a flow state record. All interactions with SDRAM memory are handled along with the caching of recently accessed flow state information. The SDRAM controller exposes three memory access interfaces, a read-write interface, a write only interface, and a read only interface. Requests to these interfaces are prioritized in the same order, with the read-write interface having the highest priority.

The layout of the State Store Manager along with its interactions to the memory controller and other modules in the TCP Processing Engine can also be seen in figure 5. Upon processing a new packet, a flow identifier hash value is computed and a record retrieval operation is initiated. The State Store Manager utilizes the Read interface of the memory controller to retrieve the current state information for the flow and returns this information to the protocol processing engine. If the packet is determined to be valid and is accepted by the engine, an update operation is performed to store the new flow state. The flow blocking module also performs a SDRAM read operation in order to determine the current flow blocking state. If the flow blocking state has changed or there is an update to the application-specific state information, a write operation is also performed to updated the flow's saved state information.

In a worse-case scenario, where there is at most a single entry per hash bucket, a total of two read and two write operations to SDRAM are required for each packet. These operations are an eight word read to retrieve flow state, an 8 word write to initialize a new flow record, a 4 word read

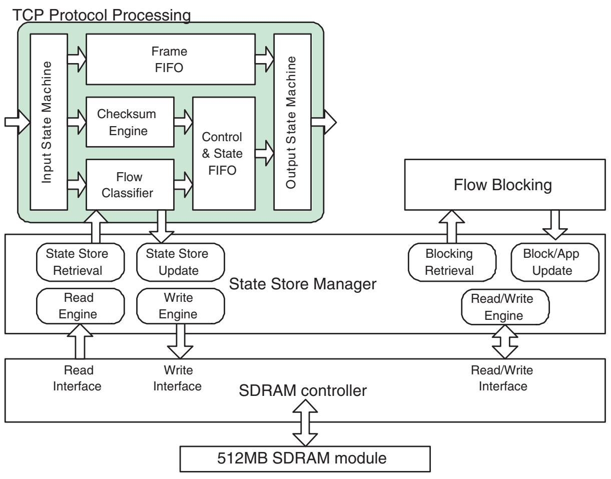

Figure 5. TCP Protocol Processing Engine & State Store Manager

to retrieve flow blocking information, and a 5 word write to update application-specific flow state and blocking information. No memory accesses are required for TCP acknowledgment packets that contain no data. Analysis indicates that all of the read and write operations can be performed during the packet processing time if the average TCP packet contains more than 120 bytes of data. If the TCP packets contain less than this amount of data, insufficient time may be available to complete all of the memory operations while processing the packet. If this occurs, the packet may be stalled while waiting for a memory operation to complete. The average TCP packet size on the Internet has been shown to be approximately 300 bytes[14]. It is important to note that the TCP Protocol Processing engine does not need to access memory for acknowledgment packets that contain no data. Given that half of all TCP packets are acknowledgments, the average size of a packet requiring memory operations to the state store will be larger than the 300 byte average previously stated. Processing larger packets decrease the likelihood of throttling due to memory access latency. On average, the system will have over twice the memory bandwidth required to process a packet when operating at OC-48 rates.

5 Conclusions

This paper discusses an architecture for performing content scanning of TCP flows within high speed networks. The circuit design is targeted for the Xilinx XCV2000E FPGA on the FPX platform with an operational clock frequency of 80MHz. This architecture provides for the monitoring of eight million simultaneous TCP flows at OC-

48 (2.5Gb/s) line rates. Utilizing a 512MB commodity SDRAM memory, 8M flows can be stored with at a cost of .00125 cents per flow. By storing 64 bytes per flow, it is possible to maintain the full context of the scanning engine for each flow.

By developing a circuit that operates in a Field Programmable Gate Array (FPGA) device, run-time changes can be made to the list of scanned content. Having the ability to quickly react to new filtering requirements, makes this architecture an ideal framework for a network based Intrusion Detection System.

New FPGA devices are available which have 4 times the number of logic gates and operate at over twice the clock rate of the XVC2000E used on the FPX platform. The latest memory modules support larger densities, higher clock frequencies, and Double Data Rate (DDR) transfer speeds. Utilizing these new devices, the TCP based content scanning engine could achieve OC-192 (10Gb/s) data rates without requiring major modifications.

6 Acknowledgments

This work is supported by a grant from Global Velocity.

References

[1] K. Bhargavan, S. Chandra, P. J. McCann, and C. A. Gunter. What packets may come: Automata for network monitoring. In *Proceedings of the 28th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 206–219. ACM Press, 2001.

[2] F. Braun, J. W. Lockwood, and M. Waldvogel. Layered Protocol Wrappers for Internet Packet Processing in Reconfigurable Hardware. In *Proceedings of Symposium on High Performance Interconnects (HotI'01)*, pages 93–98, Stanford, CA, USA, Aug. 2001.

[3] C. Fraleigh, C. Diot, B. Lyles, S. Moon, P. Owezarski, D. Papagiannaki, and F. Tobagi. Design and deployment of a passive monitoring infrastructure. *Lecture Notes in Computer Science*, vol 2170:page 556, 2001.

[4] R. Franklin, D. Carver, and B. L. Hutchings. Assisting Network Intrusion Detection with Reconfigurable Hardware. In *IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, Napa, CA, USA, Apr. 2002.

[5] M. Handley, V. Paxson, and C. Kreibich. Network intrusion detection: Evasion, traffic normalization, and end-to-end protocol semantics. In *Proceedings of the 10th USENIX Security Symposium.*, Aug. 2001.

[6] S. Jaiswal, G. Iannaccone, C. Diot, J. Kurose, and D. Towsley. Measurement and classification of out-of-sequence packets in a tier-1 IP backbone, 2002.

[7] J. W. Lockwood. An Open Platform for Development of Network Processing Modules in Reprogrammable Hardware. In *IEC DesignCon'01*, pages WB–19, Santa Clara, CA, Jan. 2001.

[8] E. P. Markatos. Speeding up TCP / IP : Faster processors are not enough. Technical Report 297, 2001.

[9] J. Moscola, J. Lockwood, R. P. Loui, and M. Pachos. Implementation of a content-scanning module for an internet firewall. In *IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, Napa, CA, USA, Apr. 2003.

[10] Network World Fusion. Gigabit intrusion-detection systems. http://www.nwfusion.com/reviews/2002/1104rev.html.

[11] M. Roesch. Snort - lightweight intrusion detection for networks. In *13th Administration Conference, LISA'99*, Seattle, WA, Nov. 1999.

[12] D. V. Schuehler and J. Lockwood. TCP-Splitter: A TCP/IP Flow Monitor in Reconfigurable Hardware. In *Proceedings of Symposium on High Performance Interconnects (HotI'02)*, pages 127–131, Stanford, CA, USA, Aug. 2002.

[13] R. Sidhu and V. K. Prasanna. Fast regular expression matching using FPGAs. In *IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, Rohnert Park, CA, USA, Apr. 2001.

[14] K. Thompson, G. J. Miller, and F. Wilder. Wide-area internet traffic patterns and characteristics. *IEEE Network Magazine*, vol 11 no. 6:10–23, Nov/Dec 1997.