A Modular System for FPGA-Based TCP Flow Processing in High-Speed Networks[*]

David V. Schuehler and John W. Lockwood

Applied Research Laboratory, Washington University
One Brookings Drive, Campus Box 1045
St. Louis, MO 63130-4899 USA
{dvs1, lockwood}@arl.wustl.edu
http://www.arl.wustl.edu/projects/fpx/reconfig.htm

Abstract. Field Programmable Gate Arrays (FPGAs) can be used in Intrusion Prevention Systems (IPS) to inspect application data contained within network flows. An IPS operating on high-speed network traffic can be used to stop the propagation of Internet worms and to protect networks from Denial of Services (DoS) attacks. When used in the backbone of a core network, the device will be exposed to millions of active flows simultaneously. In order to protect the data in each connection, network devices will need to track the state of every flow. This must be done at multi-gigabit line rates without introducing significant delays.
This paper describes a high performance TCP processing system called TCP-Processor which supports flow processing in high-speed networks utilizing multiple devices. This circuit provides stateful flow tracking, TCP stream reassembly, context storage, and flow manipulation services for applications which process TCP data streams. A simple client interface eases the complexities associated with processing TCP data streams. In addition, a set of encoding and decoding circuits has been developed which efficiently transports this interface between multiple FPGA devices. The circuit has been implemented in FPGA hardware and tested using live Internet traffic.

1 Introduction

Including reconfigurable networking technology within the core of the Internet offers enhanced levels of service to users of the network. New types of data processing services can be applied to either all traffic traversing the network, or to just a few selected flows.

This paper presents a modular circuit design of a content processing system implemented in FPGA hardware. A circuit has been built that reassembles TCP/IP data packets into their respective byte streams at multi-gigabit line rates. The implementation contains a large per-flow state store which maintains

[*] This work was supported by a grant from Global Velocity. The authors of this paper have received equity from the the license of technology to that company. John Lockwood has served as a consultant and co-founder to Global Velocity.

J. Becker, M. Platzner, S. Vernalde (Eds.): FPL 2004, LNCS 3203, pp. 301–310, 2004.

64 bytes of state information per active flow and supports 8 million bidirectional TCP flows concurrently.

The technology described in this paper supports the coupling of other FPGA-based data processing circuits in order to develop larger, more complex processing systems. This technology enables a new generation of network services to operate within the core of the Internet.

The remainder of this paper is divided into the following sections. Section 2 provides motivation for this work. Section 3 describes related work on high-performance processing systems. Section 4 describes the design of the system. Section 5 describes current results. Section 6 outlines future work and section 7 provides concluding statements.

2 Motivation

Over 85% of all traffic on the Internet today uses the TCP/IP protocol. TCP is a stream-oriented protocol providing guaranteed delivery and ordered byte flow services. Processing of TCP data flows at a location in the middle of the network is extremely difficult. Along the way, packets can be dropped, duplicated and re-ordered. Packet sequences observed within the interior of the network can be different from packets received and processed at the connection endpoints. The complexities associated with tracking the state of end systems and reconstructing byte sequences based on observed traffic are significant [2].

Many types of network services require access to the TCP stream data traversing high-speed networks. These services may include those which detect and prevent the spread of Internet worms/viruses, those that detect and remove spam, those that perform content-based routing operations, and those that secure data. The TCP processing circuit described in this paper enables these complex network services to operate at gigabit speed by providing an environment for processing TCP stream data in hardware.

A vast number of end systems communicate over the Internet. This traffic is concentrated to flow over a relatively small number of routers which forward traffic through the core of the Internet. Currently, Internet backbones operate over communication links ranging in speed from OC-3 (155 Mbps) to OC-768 (40 Gbps) rates. Table 1 illustrates a breakdown of common communication links, their corresponding data rates, and the rate at which packets of different sizes can be transmitted over those links. It is important to note that with faster links processing smaller packets, circuits must be able to process millions of TCP packets per second. Other existing TCP processing circuits are unable to operate at high bandwidth rates and manage millions of active flows. Instead, these other network monitors typically operate on end systems or in local area networks where the overall bandwidth and/or the total number of flows to be processed is low.

The TCP processing circuit described in this paper handles the complexities associated with flow classification and TCP stream reassembly. It exposes network traffic flow data through a simple client interface. This enables other

Table 1. Optical links and associated data rates

Link type	Data rate	40 byte pkts/sec	64 byte pkts/sec	500 byte pkts/sec	1500 byte pkts/sec
OC-3	155 Mbps	.48 M	.3 M	38 K	12 K
OC-12	622 Mbps	1.9 M	1.2 M	.16 M	52 K
GigE	1.0 Gbps	3.1 M	2.0 M	.25 M	83 K
OC-48	2.5 Gbps	7.8 M	4.8 M	.63 M	.21 M
OC-192/10 GigE	10 Gbps	31 M	20 M	2.5 M	.83 M
OC-768	40 Gbps	130 M	78 M	10 M	3.3 M

high-performance data processing sub-systems to operate on network content without having to perform complex protocol processing operations.

3 Related Work

The majority of existing packet capturing and monitoring systems are software-based and have severe performance limitations that prevent them from effectively monitoring high speed networks. Software-based solutions like Bro [11], Snort [12], and WebSTAT [15] perform analysis of TCP flows, but are limited to processing data at rates less than 100Mbps.

FPGA-based network processing systems can process network traffic at much higher data rates. A hardware circuit developed at Georgia Tech called TCP-Stream Reassembly and State Tracking can analyze a single TCP flow at 3.2Gbps [10]. The circuit was tested using a FPGA device which tracks the state of a TCP connection and performs limited buffer reassembly. Because the circuit operates only on a single flow, additional copies of the circuit need to be instantiated in order to process multiple flows simultaneously. Using this scheme, a maximum of 30 flows can be processed with a Xilinx Virtex 2000E FPGA device.

Another FPGA-based approach to TCP processing was undertaken at the University of Oslo [7]. This design provides TCP connection state processing and TCP stream reassembly functions for both client and server directed traffic on a single TCP connection. A 1024-byte reassembly buffer is maintained for both client-side and server-side traffic. Packets lying outside of the reassembly buffer space on receipt are dropped. Portions of the circuit design have been implemented and are able to process data at 3.06Gbps. A separate instance of this circuit is required to process each individual TCP connection. A Xilinx Virtex 1000E FPGA can support a maximum of 8 TCP flows. Thus for hardware, existing systems are severely limited in the number of flows that they can process. Additionally, they provide no simple mechanisms to support interfacing with other types of data processing sub-systems.

Hardware sub-systems have been developed which match patterns in data streams using deterministic finite automata (DFA) and nondeterministic finite automata (NFA) circuits. Sidhu and Prasanna implemented a regular expression matching engine in FPGA logic that constructs NFAs [14]. Franklin *et al.*

extended this work by creating FPGA-based regular expressions corresponding to the Snort rule database [6]. Moscola *et al.* developed a DFA-based approach to regular expression processing in hardware [9]. Although the time and space requirements associated with constructing DFAs can be exponentially long in the worst case, the authors show that such scenarios rarely occur and that their DFA implementation on average contains fewer states than a NFA implementation. Other work has led to the development of content matching systems implemented with Bloom filters that can scan for very large numbers of strings [5]. Circuits have been developed which use multiple hashes and probabilistic matching in order to scan for 35,475 unique signatures on a Virtex 2000E [1].

All of these types of FPGA-based data processing circuits are prime candidates for integration with the modular TCP processing system outlined in this paper.

4 Design

The TCP flow processing architecture described in [13] has been implemented in FPGA logic. It is capable of monitoring 8 million bidirectional TCP flows on an OC-48 (2.5 Gbps) network link. This circuit provides a simple client interface enabling other FPGA circuits to easily process TCP data streams. Network data packets are annotated with additional control signals which provide information about which data bytes correspond to the IP header, the TCP header, and the TCP payload section. There are also signals that indicate which TCP data bytes are part of a consistent stream of data and which bytes should be ignored because they are retransmissions. Signals which indicate the beginning and end of flow are included along with a unique flow identifier so that the client can independently manage per-flow context.

The architecture for the TCP-Processor consists of six distinct components. These include an *input buffer*, a *TCP processing engine*, a *state store manager*, a *routing module*, an *egress module*, and a *statistics module*. The layout of the components along with data flow and component interactions can be seen in Figure 1. There are two additional components, an *encode* module and a *decode* module, which encode and decode data for transport between multiple FPGA devices. The main flow of data traverses the circuit following the path indicated by the bold/white arrows. The *state store manager* stores and retrieves per-flow context information from off-chip memory. The dotted line through the middle of the circuit indicates that the ingress and egress portions of the TCP processing can be separated from each other and placed on different devices.

Data enters the engine as IP data packets from the Layered Protocol Wrappers [3,4] via a 32-bit wide data bus. Packets are initially processed by the *input buffer* component which buffers packets if there are any downstream processing delays. These delays can be caused by client applications which can induce backpressure into the system by sending a flow control signal.

Data from the *input buffer* flows into the *TCP processing engine*. Packets are classified and associated with the appropriate flow context information retrieved

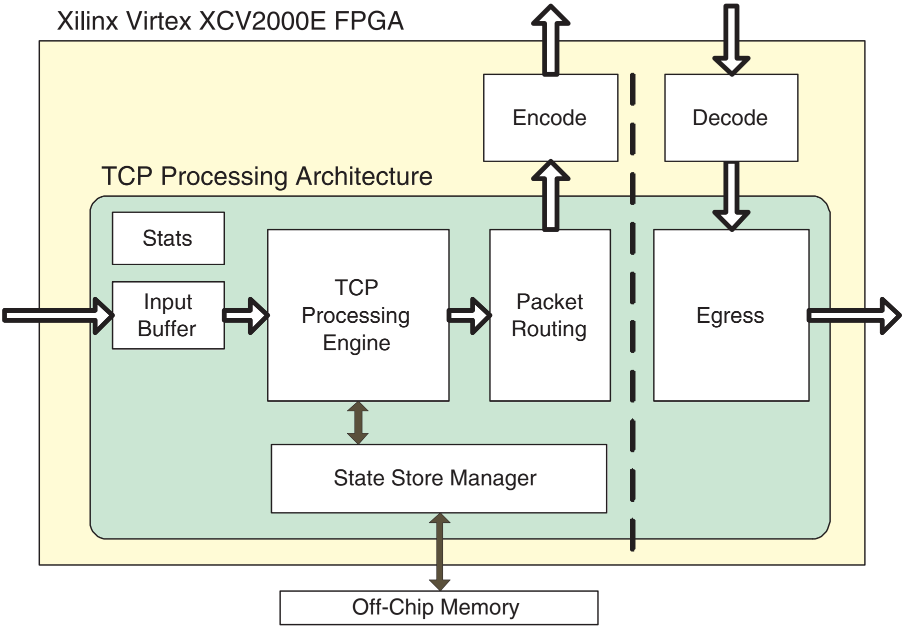

Fig. 1. Layout of TCP-Processor

via the *state store manager*. The TCP checksum is validated and other TCP-specific processing occurs at this time. New flow context information is written back to the *state store manager* and the packet is passed on to the *packet routing module*.

The *packet routing module* provides the client interface. Figure 2 shows the timing diagram of the transmission of a single TCP data packet to the monitoring application. The DATA and FLOWSTATE busses contain an indication of the data that would be present at each clock cycle during normal processing. The components of the IP and TCP headers are shown with the control signals. Signals are provided which indicate the start of frame (SOF), start of IP header (SOIP), and the start of IP payload (SOP). The TCP data enable signal (TDE) indicates that there is TCP stream data on the DATA bus and the BYTES vector identifies which of the four bytes on the DATA bus contain in-order TCP stream data. In this example, the NEWFLOW signal indicates that the data contained within this packet represents the first few bytes of a new data stream and that the monitoring application should initialize its processing state prior to processing this data. Additional information can be clocked through the data bus before and/or after the packet data. This allows for lower-level protocols, such as VCIs, Ethernet MAC addresses, shims, or packet routing information to pass through the hardware along with the packet. On the `DATA` bus of Figure 2, the `AAA` and `BBB` data values represent lower layer protocol and control information prior to the start of the IP packet. CCC and DDD represent trailer fields that follow the IP packet. The AAA, BBB, CCC, and DDD content is ignored by the monitoring application but is passed through to the client interface of the TCP-Processor for use in outbound packet processing.

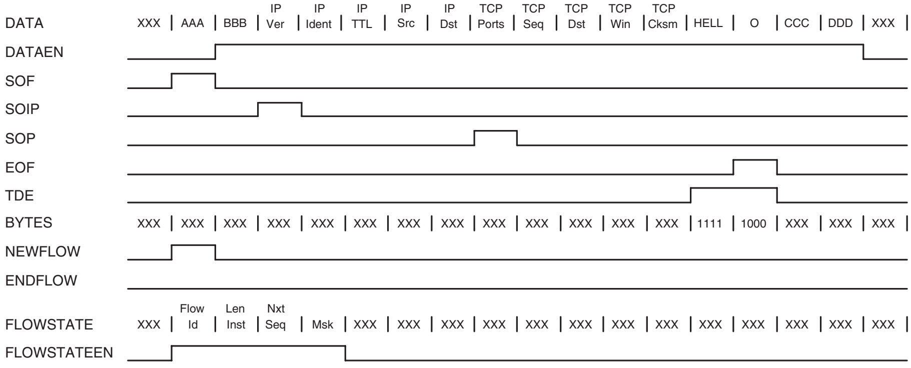

Fig. 2. Timing Diagram showing Client Interface

The *state store manager* manages interactions with a large external memory module to store per-flow context information. By default, 64 bytes of context information is stored for each flow. When the TCP-Processor is configured for bidirectional monitoring, 32 bytes of storage are used for each direction of traffic flow. Because the context information needs to be retrieved and updated when processing each packet, the circuit that implements the *state store manager* is highly optimized.

The *statistics module* collects and maintains event counts from the other components in the system. 28 independent counters collect information that includes the number of packets processed by each module, the number of new flows, the number of terminated flows, the number of active flows, counts of the TCP data bytes processed, counts of the total bytes processed, counts of the out-of-sequence packets and the number of retransmitted packets. This data is accumulated in separate 16, 24 and 32 bit counters, depending on the expected frequency of each event. On a predetermined periodic interval, all of the collected statistics are formatted into a UDP packet and transmitted to an external application where the data can be written to disk or plotted in real-time.

This TCP processing architecture includes a mechanism for efficiently transporting signals from one FPGA device to another, which enables the development of complex intrusion prevention systems implemented on multiple FPGAs. Signals on one device are encoded and transported to a second device where the information is decoded and the original waveform is reproduced. The encoded format uses an 8-bit header containing a 4-bit type and a 4-bit header length. This format is both self describing and extensible and therefore enables additional information to be added to the encoded data as it is passed from device to device. This encoded header information is prepended to the network data packet and sent across the inter-device data path. Older processing circuits can easily skip past new control headers by examining the common header fields and advancing to the next control header. Figure 3 shows the encoding and decoding process.

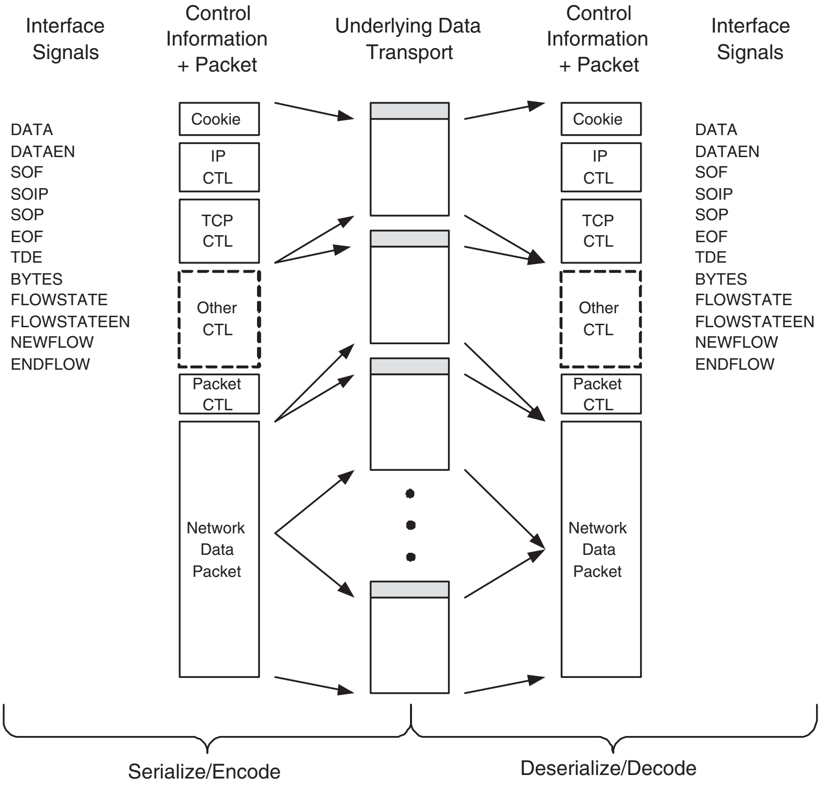

Fig. 3. Encoding signals for TCP/IP flows processed by multiple FPGAs

5 Results

The TCP-Processor circuit has been implemented in FPGA logic. When targeting a Xilinx Virtex XVC2000E-8 FPGA, the TCP-Processor has a post place-and-route frequency of 85.565MHz utilizing 59% (95/160) of the block RAMs and 35% (7279/19200) of the slices. The device is capable of processing 2.9 million 64-byte packets per second. A diagram of the TCP-Processor circuit layout on a Xilinx Virtex 2000E FPGA is shown in Figure 4. The layout is loosely divided into regions which correspond to the various processing functions of the circuit.

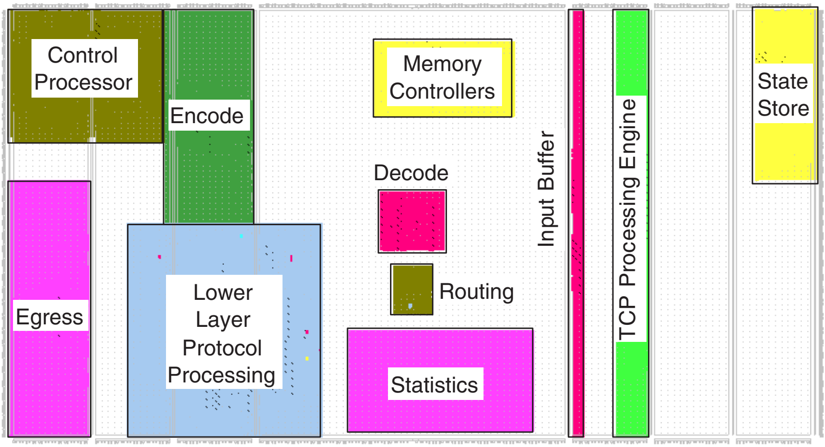

Fig. 4. TCP-Processor circuit layout on Xilinx XCV2000E

The TCP-Processor has been tested on the Field-programmable Port Extender (FPX) platform [8]. The FPX platform contains a Xilinx Virtex 2000E-8 FPGA, two 2.5Gbps interfaces, two 2MB external ZBT SRAM devices and two 512MB external PC100 SDRAM devices. The two network interfaces are positioned such that devices can be stacked on top of one another. Figure 5 shows FPX devices in a stacked configuration. A Gigabit Ethernet line card used to transfer data out of the FPX platform is visible in the foreground.

6 Future Work

By incorporating newer FPGAs, such as the Xilinx Virtex-II Pro, traffic can be processed at OC-192 (10Gbps) data rates using a circuit like the one described in this paper. A few additional mechanisms can be employed to increase the performance of the TCP-Processor for use in OC-768 (40Gbps) networks. In a faster circuit, memory latency could prevent the circuit from fully processing a steady stream of minimum length packets. By instantiating two copies of the *TCP processing engine* and using an *input buffer* to route traffic between the two engines, the memory latency issue can be overcome by threading.

A new extensible networking platform is currently under development at the Washington University Applied Research Laboratory. This platform will incorporate the Xilinx Virtex-II Pro FPGA, Double Data Rate (DDR) memory, a Ternary Content Addressable Memory (TCAM) and an Intel network processor

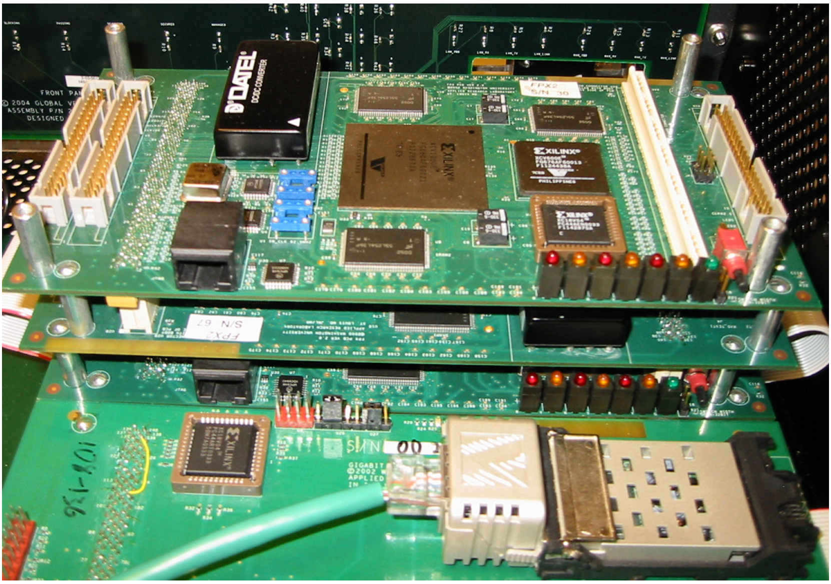

Fig. 5. Stacked FPX devices

to support traffic processing at 10 gigabits per second. We plan to support the TCP-Processor on this new research platform.

7 Conclusion

This paper described the design and implementation of a high-performance TCP flow monitoring system called TCP-Processor for use in an extensible environment. The TCP-Processor was implemented and tested on a Xilinx XCV2000E FPGA utilizing the FPX platform. The circuit operates at 85MHz and is capable of monitoring 8 million bidirectional TCP flows at OC-48 (2.5 Gbps) data rates. This design could be enhanced to monitor higher speed networks by employing parallel processing circuits and using current FPGA and memory devices.

The TCP-Processor provides a simple client interface for monitoring TCP flows which annotates existing network data packets with additional signals. The design of the TCP-Processor is both modular and flexible and can be easily adapted to other extensible networking environments which employ FPGA devices.

References

1. M. Attig, S. Dharmapurikar, and J. Lockwood. Implementation results of bloom filters for string matching. In *IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, Napa, CA, Apr. 2004.

2. 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.
3. F. Braun, J. Lockwood, and M. Waldvogel. Reconfigurable router modules using network protocol wrappers. In *Proceedings of Field-Programmable Logic and Applications*, pages 254–263, Belfast, Northern Ireland, Aug. 2001.
4. 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.
5. S. Dharmapurikar, P. Krishnamurthy, T. Sproull, and J. Lockwood. Deep packet inspection using parallel bloom filters. In *Proceedings of Symposium on High Performance Interconnects (HotI'03)*, pages 25–29, Stanford, CA, USA, Aug. 2003.
6. 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.
7. S. Li, J. Toressen, and O. Soraasen. Exploiting stateful inspection of network security in reconfigurable hardware. In *Field Programmable Logic and Applications (FPL)*, Lisbon, Portugal, Sept. 2003.
8. J. W. Lockwood, N. Naufel, J. S. Turner, and D. E. Taylor. Reprogrammable Network Packet Processing on the Field Programmable Port Extender (FPX). In *ACM International Symposium on Field Programmable Gate Arrays (FPGA'2001)*, pages 87–93, Monterey, CA, USA, Feb. 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. M. Necker, D. Contis, and D. Schimmel. TCP-Stream Reassembly and State Tracking in Hardware. FCCM 2002 Poster, Apr 2002.
11. V. Paxson. Bro: A system for detecting network intruders in real-time. *Computer Networks*, 31(23-24):2435–2463, 1999.
12. M. Roesch. SNORT - Lightweight Intrusion Detection for Networks. In *LISA '99: USENIX 13th Systems Administration Conference*, November 1999.
13. D. V. Schuehler, J. Moscola, and J. Lockwood. Architecture for a hardware based, tcp/ip content scanning system. In *Proceedings of Symposium on High Performance Interconnects (HotI'03)*, pages 89–94, Stanford, CA, USA, Aug. 2003.
14. 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.
15. G. Vigna, W. Robertson, V. Kher, and R. Kemmerer. A Stateful Intrusion Detection System for World-Wide Web Servers. In *Proceedings of the Annual Computer Security Applications Conference (ACSAC 2003)*, pages 34–43, Las Vegas, NV, December 2003.