An FPGA-based network intrusion detection system with on-chip network interfaces

CHRISTOPHER R. CLARK†*, CRAIG D. ULMER‡, DAVID E. SCHIMMEL†

Network intrusion detection systems (NIDS) are critical network security tools that help protect computer installations from malicious users. Traditional software-based NIDS architectures are becoming strained as network data rates increase and attacks intensify in volume and complexity. In recent years, researchers have proposed using FPGAs to perform the computationally-intensive components of intrusion detection analysis. In this work, we present a new NIDS architecture that integrates the network interface hardware and packet analysis hardware into a single FPGA chip. This integration enables a higher performance and more flexible NIDS platform. To demonstrate the benefits of this technique, we have implemented a complete and functional NIDS in a Xilinx Virtex II Pro FPGA that performs in-line packet analysis and filtering on multiple Gigabit Ethernet links using rules from the open-source Snort attack database.

Keywords: FPGA; intrusion detection; pattern-matching; Gigabit Ethernet

1. Introduction

A network intrusion detection system (NIDS) is a device that uses a combination of hardware and software to monitor a computer network for attempts to violate a security policy. In addition to observing network operations, a NIDS can also be designed to react to attacks, either by filtering malicious packets from the network or by initiating appropriate countermeasures. Over the years, a number of NIDS architectures have been discussed in the literature. At a fundamental level, these architectures are all based on two primary components: a network interface (NI) unit that interacts with the physical network and an intrusion detection (ID) unit that examines and reacts to packets captured by the NI.

Early network intrusion detection systems consisted of application software running ID analysis algorithms on commodity servers equipped with off-the-shelf network interface cards. An example of this type of system is the open-source NIDS software package, Snort

* Corresponding author. Email: cclark@ece.gatech.edu
† School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA, USA
‡ Sandia National Laboratories, Livermore, CA, USA. Sandia National Laboratories is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under Contract DE-AC04-94-AL8500.

[Roesch 1999], which runs on a wide variety of operating systems and hardware platforms. While this approach provides straightforward development and economical deployment, its performance is limited because (1) there is a significant amount of I/O overhead associated with data transfers between the NI and the host CPU, and (2) general purpose CPUs are not able to take advantage of the available parallelism in the ID analysis tasks. For these reasons, software-based NIDS are often unable to keep up with the data rates of modern high-speed networks.

As a means of improving NIDS performance, researchers have turned to reconfigurable computing platforms where ID operations can be performed in hardware rather than software. In recent years, a number of designs have been presented that use field-programmable gate arrays (FPGAs) to implement intrusion detection analysis. A significant amount of work has been done on accelerating specific computationally-intensive tasks, such as TCP state tracking and stream reassembly [Necker et al. 2002; Li et al. 2003; Schuehler et al. 2004] and inspection of packet data [Franklin et al. 2002; Dharmapurikar et al. 2003; Moscola et al. 2003; Baker et al. 2004; Cho et al. 2004; Clark and Schimmel 2004; Sourdis et al. 2004]. While these FPGA designs provide significant performance gains over software-based implementations for ID processing, they do not address the other factor limiting NIDS performance: efficiently transferring data between the NI and ID components.

There are some complete NIDS platforms incorporating FPGAs described in the literature. All of these designs attempt to overcome both performance bottlenecks of software-based NIDS, but they differ in their choice of system components and the partitioning of functionality between components. In [Gokhale et al. 2002], a system is developed that uses FPGAs to offload most ID processing from a NIDS software application. A 1 Gb/s NI board is attached as a daughter card to an FPGA board through a custom interface. The combined board connects to a host PC via a standard PCI bus. In this design, FPGAs perform header and payload analysis on network packets and then send a list of matching rules to software running on the host processor that then takes appropriate action. Although the FPGAs can process network data at 2 Gb/s, the PCI interface between the hardware and software processors limits the overall throughput of the system to 700 Mb/s.

A different approach to partitioning NIDS tasks is taken by [Clark, Lee et al. 2004]. The aim of this work is to provide transparent intrusion detection functionality to a host by implementing a complete NIDS on a network card. The design used is based on the integration of off-the-shelf network processor and FPGA development boards. The network processor board connects to a host PC via a PCI connector and to the FPGA board via a separate PCI mezzanine connector (PMC). The use of a software-programmable network processor allows more complex stateful analysis to be carried out close to network before packets are sent across the bus to the host. The FPGA is used by the network processor as a pattern-matching accelerator. In this system, the overhead of PCI communication between the network processor and FPGA reduces pattern-matching throughput to less than 10 percent of that achievable by the FPGA design.

An extensible, FPGA-based firewall appliance is described in [Lockwood et al. 2003]. This system is based on a modular platform consisting of network line cards connected to custom FPGA boards. On each FPGA board, one FPGA is dedicated to interacting with the external network interfaces, and another FPGA is available to implement content processing applications, such as NIDS. All components of this custom platform support at least 2 Gb/s throughput.

In this paper, we study the integration of network interface and intrusion detection circuitry to provide a complete NIDS on a single FPGA device. This allows for a more flexible NIDS architecture that also avoids the bottlenecks associated with transfers between the NI and ID components.

2. An Integrated NIDS in an FPGA

Field-programmable gate arrays (FPGAs) are programmable logic devices that can be configured to implement custom digital systems. Over the years, FPGAs have evolved into more than just simple arrays of reconfigurable logic. Modern 'platform' FPGAs are system-on-a-chip devices that contain complex components, such as embedded SRAM, digital signal processing blocks, high-speed transceivers, and embedded CPU cores, in addition to large amounts of reconfigurable logic.

The Xilinx Virtex II Pro (V2P) FPGA [Xilinx-V2P] is an example of a modern platform FPGA architecture that includes these new components. Of particular relevance to this work are the V2P's multi-gigabit transceiver (MGT) modules, which are also referred to as Rocket I/O [Xilinx-RIO] ports. A Rocket I/O port is a versatile transceiver that enables the FPGA to physically communicate with a variety of network standards, including Gigabit Ethernet (GigE) [IEEE-802.3] and InfiniBand (IB) [IBTA]. Each Rocket I/O port is comprised of functional blocks that implement common physical-layer network operations, such as 8b/10b encoding, serialization/deserialization, embedded clock recovery, and CRC verification. The largest FPGA in the V2P family includes 20 Rocket I/O ports, with each port operating at up to 3.125 Gb/s full-duplex.

The fact that modern FPGAs are capable of interacting directly with a physical network enables us to propose an evolution of NIDS architecture: the integration of network interface and intrusion detection circuitry in a platform FPGA. We argue that there are multiple advantages to consolidating NIDS components into a single FPGA. From a platform design perspective, integration simplifies board layout and reduces the chip count and form factor of a NIDS. Integration also removes the need for large chip-to-chip data transfers, which can be a performance bottleneck in multi-chip systems. However, the main benefit of an integrated system is the increased level of customization that is made available to system designers. FPGAs are flexible architectures that can be reconfigured after fabrication time to meet new application demands. In order to maximize this opportunity, it is desirable to implement as much of the NIDS architecture as possible in reconfigurable logic.

There are a variety of means by which an FPGA-based NIDS can be customized. At the component level, individual units can be tuned to meet the needs of the application. For ex-

ample, the NI units in many NIDS applications blindly transfer data into and out of the network in an unsophisticated manner. As we discuss later, this knowledge can be used to simplify the NI implementation in order to conserve FPGA resources and decrease processing overhead. In addition, the ID component can be tailored to a particular set of detection rules, thereby reducing logic usage and increasing performance. Customization can also be applied at higher levels of the system architecture. By implementing the NIDS inside the FPGA, designers are free to implement a variety of data flow architectures and processing topologies. For example, we have used a single reference board to develop multiple NIDS implementations with different NI configurations. If the NI modules were not implemented in the FPGA, we would have had to fabricate multiple circuit boards or one complex board.

The remainder of this paper addresses issues associated with constructing a complete NIDS in a single platform FPGA. The network interface, intrusion detection, and monitoring modules are described in sections 3, 4, and 5, respectively. The developed components leverage architectural features of the Xilinx Virtex II Pro FPGA family. A proof-of-concept example of a complete NIDS is presented in Section 6. We implement and evaluate a NIDS that transparently monitors and filters up to four full-duplex GigE connections. In section 7, experimental results from a design space exploration are presented. Finally, the paper is concluded with some observations about this work and possibilities for future work.

3. Network Interface Module

The first component of an integrated, FPGA-based NIDS is the network interface (NI), which is responsible for coherently exchanging packets between the FPGA and the physical network. Given the nature of the work performed by the NIDS, it is possible to simplify the functionality of the NI to a minimum. Simplifying the NI reduces the overall size of the unit, which allows more detection rules to be instantiated in the system. A simplified diagram for the NI utilized in this work is depicted in Figure 1. The main components of this module are transmit and receive controllers for the Rocket I/O port and packet FIFOs for in-flight messages.

3.1. Rocket I/O Controllers

At the heart of an NI for the V2P FPGA is a Rocket I/O module. The Rocket I/O module provides low-level services for interactions with physical network standards such as Gigabit Ethernet (GigE). A Rocket I/O module effectively provides a raw byte stream interface to the network. Users must insert and extract data in lockstep with the Rocket I/O module in order to prevent data from being corrupted or lost. Because of the clock-by-clock operations that must take place at the interface of a Rocket I/O module, it is useful to construct wrappers around the hardware to present a more user-friendly interface to other circuits that need to interact with the network. Transmit and receive engines were constructed to provide a more usable interface to the network. The receive engine de-frames and extracts Ethernet packets from the incoming network stream. Conversely, the transmit engine frames and transmits outgoing

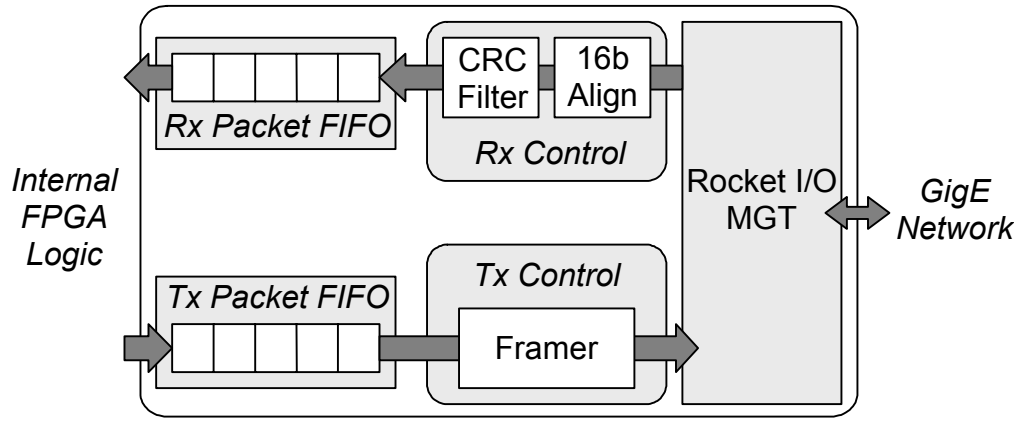

Figure 1: Gigabit Ethernet network interface module

messages. When the transmit engine does not have data to send, it transmits idle channel sequences to maintain clock synchronicity on the link. The receive and transmit engines strip and generate CRCs for incoming and outgoing messages in order to prevent the ID module from generating false positive matches on the CRC data. Additional circuitry monitors the physical layer status in the Rocket I/O module and automatically resets the unit as needed.

3.2. Packet FIFOs

The final component of the NI module is a pair of packet FIFOs for buffering incoming and outgoing packets. These units are implemented using a small number of dual-ported BlockRAM banks that are available within the FPGA. While internal SRAM is limited in the V2P architecture, there is still enough memory available to support 2 KB - 16 KB of messages for a packet FIFO in the smallest V2P part. The current implementation of the GigE NI does not support jumbo frames due to the limited amount of buffer space available in the FPGA. The dual-ported nature of the SRAM allows the producer to operate at a different rate and data width than the consumer. The packet FIFOs are also designed to allow an optional flag to be asserted after an injection that forces the last inserted message to be ignored by the consumer. This feature is useful in a NIDS, where the ID module's assessment of whether a packet should be dropped can be delayed by a small number of clock cycles. While packet buffering is not strictly necessary in an FPGA-based NIDS, doing so adds a great deal of design freedom to the architecture because it decouples the NI and ID modules. This decoupling allows the ID module to operate at a higher clock rate and with wider data streams. As explained later, these factors enable the NIDS to share an ID module among multiple NI cores.

3.3. Resource Utilization

While the V2P architecture has been available for some time, there have been few reports in the literature on the networking capabilities of the devices. The placed and routed design for the GigE NI used in this work was analyzed to better characterize the hardware. Our GigE NI without FIFOs occupies 259 V2P slices, which consumes approximately 5.3 percent of a V2P7 FPGA and less than 0.6 percent of a V2P100 FPGA. While smaller NI cores may be possible, this NI design is a relatively compact implementation that provides all the functionality needed for our NIDS. As a point of reference, the Xilinx sample implementations of minimal GigE MAC and PHY components without FIFOs requires 763 slices [Xilinx-Eth], which is about three times larger than our NI. As this optimization demonstrates, implementing the NI in reconfigurable logic is beneficial because it allows designers to customize the NI according to an application's characteristics.

4. Intrusion Detection Module

An intrusion detection (ID) FPGA core has been developed that inspects packets on high-speed networks in real-time. The design compares each packet against a large number of detection rules simultaneously. Each rule specifies multiple packet properties that together identify an interesting event. The open-source rule language and rule set from Snort [Roesch 1999] were chosen because of their flexibility and availability. The rule language supports relational comparisons on the packet header fields, as well as regular expression searches on the packet payload. The ID core will detect network packets that meet all the criteria of any rule. Even with a small FPGA, hundreds of rules can be programmed into a single chip.

Network data is streamed through the ID core, which operates on multiple bytes of input data per clock cycle. The core's input data width is configurable at compile-time to be two, four, or eight bytes. A block diagram of the ID module is shown in Figure 2. Each of the components is described in detail in the following sections.

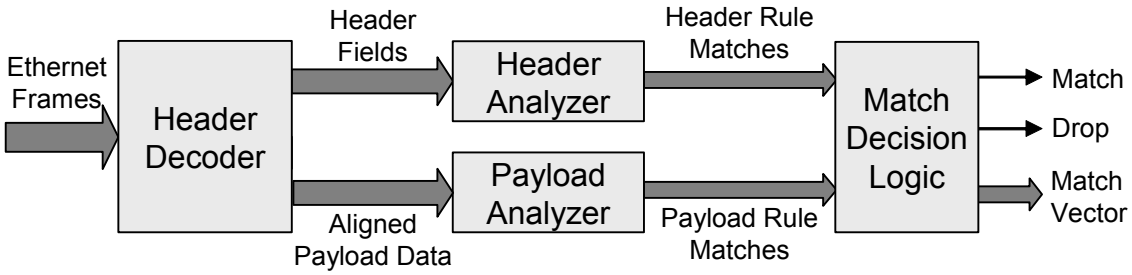

Figure 2: Intrusion detection (ID) module

4.1. Header Decoder

The header decoder accepts frames from an Ethernet interface and extracts fields from the data-link, network, and transport layer protocol headers. The supported protocols include Ethernet, ARP, IP, ICMP, TCP, and UDP. All of the relevant header fields are stored in registers and made available to the header analyzer. Also, for each packet, the decoder determines the end of the header data and produces a signal for the payload analyzer indicating that the payload data will begin on the next clock cycle. However, due to variable header lengths and the configurable input data width of the ID module, the start of the payload data does not always occur on an input word boundary. It is important to ensure that the payload analyzer processes all of the payload data (no more and no less) to avoid missing a potential match or falsely detecting a match. Therefore, the header decoder includes a small buffer of the input data and outputs a data stream with the payload aligned to the configured input data width.

4.2. Header Analyzer

The header analyzer consists of circuits that compare the fields of the incoming packet with each of the rule specifications. Only the fields with values defined in a rule are checked; all other fields are ignored. The supported comparison operations on each field include checking for specific values, ranges of values, and negated values or ranges. In order to save hardware resources, the circuit generator avoids creating duplicate circuitry for common comparison operations. For example, if there are multiple rules that trigger on traffic from web servers, the header analyzer would include only one comparison circuit to detect packets with a TCP source port of 80. There is a separate match output bit for each rule, which is true only when all of the field comparisons specified in the rule are true for the current packet.

4.3. Payload Analyzer

The payload analyzer searches each packet payload for the occurrence of patterns specified by the detection rules. The supported patterns include simple text or binary strings, as well as more complex expressions with wildcards. The pattern location within the payload can be left unspecified, specified relative to the beginning of the data, or specified relative to another pattern. The pattern-matching circuits in the payload analyzer are implemented using the globally-decoded non-deterministic finite automata (NFA) approach described in [Clark and Schimmel 2004]. This technique produces area-efficient circuits and scales to high throughputs and large numbers of patterns.

The input data is compared against all patterns in parallel at the rate of two, four, or eight bytes per clock cycle. Processing more than one input character per clock cycle allows the design to support higher throughput without increasing the clock frequency. However, there is a trade-off between throughput and circuit area. When processing N characters per clock cycle, a pattern may start at any position within each N-character block. Therefore, it is necessary to add logic to detect patterns at all N possible offsets.

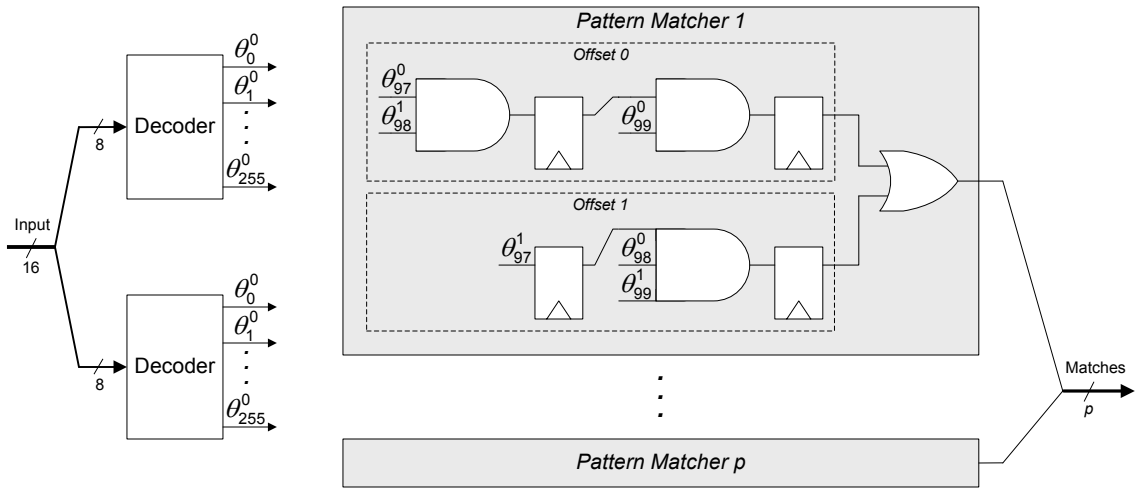

Figure 3: Payload analyzer circuit

Figure 3 shows the implementation of a payload analyzer that processes two input characters per clock cycle. Each input character is decoded into a one-hot set of character match indicator wires. For the 8-bit ASCII character set used there are 256 output wires from the decoders. Exactly one wire from each decoder will be asserted in each clock cycle. A wire labelled θ_k^c indicates whether or not the character read from the input text in character position c of the input word is the character represented by ASCII code k.

In the figure, Pattern Matcher 1 detects the character sequence {a, b, c} represented by the ASCII code sequence {97, 98, 99}. This circuit contains two parallel NFA state machine pipelines, each capable of detecting the pattern starting at one of the two possible offsets. The match indicator wires from the global character decoders are distributed to the appropriate NFA pipeline stages. Each stage of a pipeline outputs true if the output of the previous stage was true and the current stage's match indicator wires are all true. If there is a character sequence in the input that matches the pattern, a true value will propagate to the output of the final stage in the pipeline. For example, the state machine for Offset 0 of Pattern Matcher 1 will detect a match if the input contains an 'a' character in position 0 and a 'b' character in position 1 in one cycle, followed by a 'c' character in position 0 in the next cycle. The OR of the outputs from both pipelines determines the overall pattern matcher output. The match outputs of the pattern matcher circuits for all p patterns form a p-bit match vector that is used by the match decision logic.

4.4. Match Decision Logic

For each packet, the match decision logic uses the match outputs from the header analyzer and the payload analyzer to determine which rules, if any, were triggered by the packet. If one or more rules are triggered, the match output will be asserted and a decision will be made re-

garding what action should be taken based on the policy specified in the rule file. If any of the matching rules indicate that the packet should be blocked, the drop output will be asserted and the packet will be removed from the transmit buffer. This unit also outputs a match vector that indicates the matching rules.

4.5. ID Module Generation

Software has been written that generates an ID module given a file containing Snort-format rules. First, a Java program parses the rule file and translates the rules into an internal representation. Next, classes written with the Java Hardware Description Language (JHDL) [Bellows et al. 1998] libraries are used to assemble functional blocks into circuits capable of detecting packets meeting the rules' specifications. Finally, the JHDL structural circuit descriptions are converted to an EDIF netlist that is integrated with the other NIDS components by the FPGA vendor's compilation tools. The components of the ID module are implemented using a fixed configuration of the FPGA logic and routing resources, which provides high performance but requires the design to be regenerated and recompiled in order to change the rules matched by the NIDS.

5. Monitoring Module

In some network security applications, a NIDS can be designed to operate in an autonomous manner that does not require human guidance at runtime. However, most network security specialists find that it is useful to be able to interact with a NIDS in order to obtain detailed information about the dynamic status of the network. For example, a security specialist might use the information that a NIDS is filtering a large number of attacks from a particular user to direct countermeasures against the user. As a means of making the NIDS more interactive, we have constructed a runtime monitoring module that can be attached to a NIDS FPGA design at compile time. This monitor exports status information about network traffic and could easily be extended to allow a network administrator to make adjustments to the NIDS at runtime.

5.1. NIDS Monitor Functionality

At a fundamental level, a NIDS monitor performs two types of operations: system analysis and off-chip communication. For system analysis, the monitor examines the state of the NIDS and compiles runtime statistics for the system that are meaningful to network security specialists. While state information can only be captured with hardware circuitry, it is possible to implement analysis operations in either hardware or software. In a software-based approach, a CPU embedded in the FPGA is loaded with a program that parses state information and generates runtime statistics. The advantage of implementing this functionality in software is that new algorithms can be loaded into the monitor simply by changing the processor's program. This technique is also valuable because users can update a running NIDS without

having to recompile and reload the FPGA configuration. For our work we utilize one of the V2P's PowerPC CPUs to execute a simple analysis program that summarizes the number of malicious packets filtered out of the network by the NIDS.

The second type of operation performed by a NIDS monitor is off-chip communication. This hardware is used to transfer data between the NIDS and the network security specialist's console. The simplest means of facilitating this exchange is through in-band communication using the GigE ports of the NIDS. However, mixing NIDS monitor data with general network traffic is insecure. A more secure method is to use a separate communication interface for exchanging monitor data out-of-band. For our work we export monitor data to an external host through a standard RS-232 serial approach. This approach requires minimal hardware and provides a bi-directional channel for low-speed exchanges.

5.2. Monitor Architecture

Figure 4 depicts the architecture of the NIDS monitor constructed for this work. At the heart of the unit is an embedded PowerPC processor, which is a hard-core that is implemented in dedicated silicon in the V2P devices. The PowerPC includes both instruction and data caches, and features multiple interfaces that can be connected to the FPGA's reconfigurable logic. One of these interfaces is the Processor Local Bus (PLB). We utilize this flexible bus to connect an on-chip memory block and an RS-232 UART unit to the CPU. The PowerPC's program and runtime data are stored entirely in the on-chip memory block, thus eliminating the need for external memory.

The PowerPC exchanges data with the intrusion detection module through a custom-built ID monitor interface. This interface employs a small block of dual-ported memory that is connected on one side to the PowerPC's data-side on-chip memory (DS-OCM) port. This port enables the PowerPC to interact with hardware in a memory-mapped fashion, without the communication overhead of the PLB. The other side of the memory in the ID monitor is connected to hardware that captures state information about the intrusion detection module. If the data buffer in the interface becomes full, new information will be dropped and the ID module will continue to operate. In the current implementation, the hardware records both malicious packets ejected by the system and network traffic statistics. With this hardware, the CPU's analysis program can export the status of the NIDS to the user through the serial port, as well as the data contained in malicious packets.

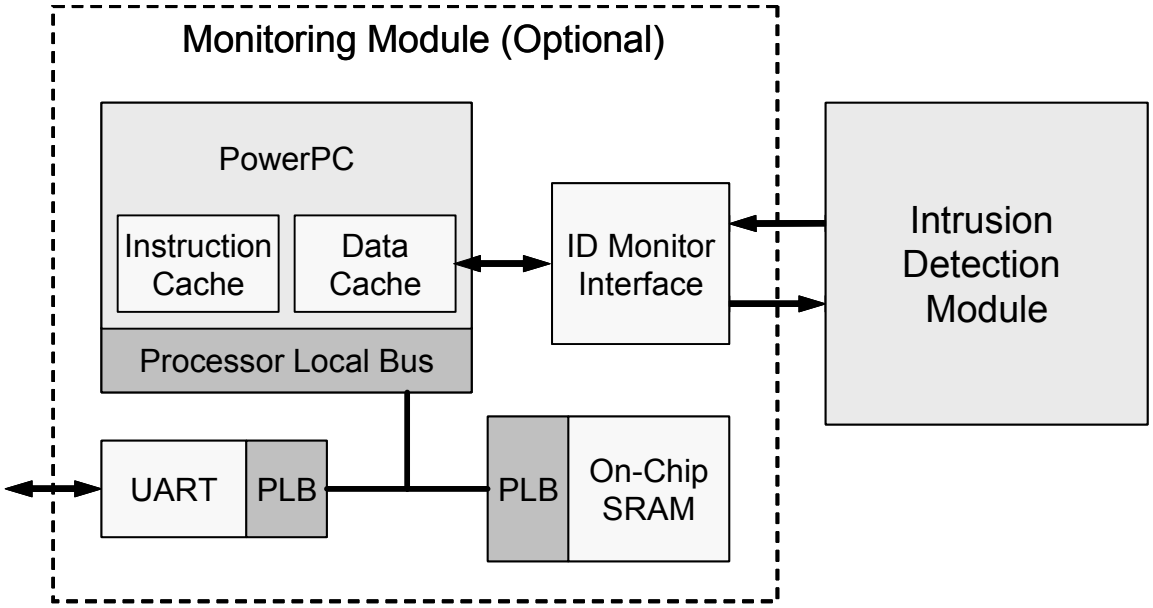

Figure 4: Architecture of monitoring module

5.3. Implementation Details

The monitoring module was implemented for the V2P architecture and added to our NIDS design as an optional unit. The on-chip memory available for PowerPC programs was set to 32 KB. While relatively small, this amount of memory proved sufficient for hosting most analysis programs. An additional 4 KB of on-chip memory is utilized in the ID monitor interface. Overall the monitoring module occupies approximately 957 V2P slices. In the V2P7, this consumes 20% of the FPGA's slices and 45% of its memory resources. In the largest V2P part, the V2P100, these system requirements equate to 2% and 5% of the FPGA's slice and memory resources, respectively. These experiments indicate that a software-controlled monitor can be added to a NIDS without significant resource utilization. However, it should be assumed that the remaining experiments in this paper do not include the optional monitoring hardware.

While the current implementation of the ID monitor is geared for exporting statistics, it would be straightforward to modify the monitor to enable users to steer the behaviour of the NIDS at runtime. In this work, administrators would issue commands through the serial connection that are interpreted and invoked by the PowerPC program. This steering could be leveraged in multiple ways. For example, hardware could be added to enable the administrator to modify the ID module's scheduling unit at runtime. This capability would enable an administrator to manually throttle the data rate of a GigE link that is suffering from an unknown attack. The monitor interface could be used in conjunction with an additional hardware mod-

ule to enable an administrator to insert new packet-filtering rules into the ID module at run-time. This capability would allow a limited number of temporary patches to be applied until an updated configuration can be compiled and programmed into the FPGA.

6. System Evaluation

A complete FPGA-based network intrusion detection system has been constructed to validate the concepts of this work and serve as a source for implementation details. The application goal of this NIDS is to transparently perform inline packet filtering on multiple network links. The NIDS in this case is effectively a man-in-the-middle system that inspects in-flight messages, dropping those that match a predefined rule signature. For clarity, a single monitored connection between a pair of hosts is referred to as a *filter bridge* (or simply a bridge) in this work.

As a means of demonstrating the flexible nature of FPGAs for NIDS, our implementation can support multiple filter bridges in a single FPGA. In this configuration, a single intrusion detection module is shared by multiple filter bridges. By operating the ID module at a data rate that is sufficiently greater than the data rates of the NI modules, the system can operate without unintentional packet loss. The design has been tested using a commercial FPGA development board containing four GigE interfaces.

6.1. Multiple Filter Bridge Architecture

A simplified diagram of the multi-filter bridge architecture is presented in Figure 5. In this NIDS, each filter bridge has two data paths that are routed through the ID module (one for each direction of the full-duplex bridge). Time-division multiplexing is employed to share the ID module among all the NI modules in the system. A scheduling unit monitors the status of each GigE interface in the system and determines the order in which NIs access the shared ID module. The current implementation utilizes a round-robin scheduler, although more sophisticated algorithms could be easily applied.

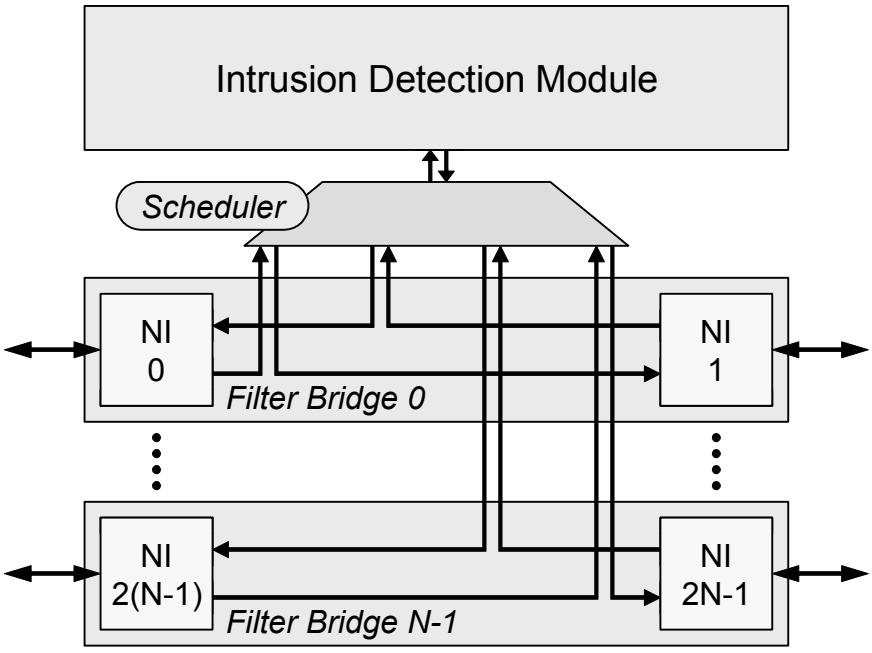

Figure 5: Multiple filter bridges sharing a single ID module

Packet loss in the multi-filter bridge design can be avoided if the ID module operates at a data rate that is greater than the aggregate data rate of the system's NIs. Given that each GigE NI captures data at a maximum rate of 1 Gb/s, a lossless system with N filter bridges and $2N$ NIs would require an ID module that processed data at a rate that is at least $2N$ Gb/s. Operating the ID module at a higher rate than the NI data streams is possible by buffering incoming messages, and using wider and/or faster data channels for communication between the NI and the ID modules. The 1 Gb/s data streams produced by the NIs in this work are 16-bit channels that are clocked at 62.5 MHz. Simply widening the ID data path to 32-bits or 64-bits provides a data rate speedup of 2x or 4x. An additional 2x data rate increase can be achieved by doubling the clock to 125 MHz and applying tighter timing constraints to the ID component of the design at compile time.

6.2. Testbench Hardware

The NIDS design was implemented and tested using Xilinx's ML300 platform [Xilinx-ML300]. The ML300 is a stand-alone reference board for the Virtex II Pro FPGA that contains a variety of hardware resources, including four fiber-optic GigE network ports that are driven by the FPGA's Rocket I/O modules. The ML300's FPGA is a V2P7, which is one of the smallest FPGAs in the V2P family. This device has roughly 11 percent of the logic resources of the largest FPGA in this family, the V2P100.

As a means of testing and debugging the system, the reference board was placed between a pair of workstations with standard GigE network interface cards. For tests that involved two filter bridges, a GigE cable was used to externally connect the two filter bridges in series. Xilinx's ChipScope [Xilinx-CSP] tool was used to observe the characteristics of the NIDS. ChipScope adds instrumentation hardware to a design at compile time, which can be used to monitor the internal signals of the NIDS at runtime. ChipScope is an incredibly valuable tool for designs such as a NIDS, where system behavior is highly dependent on external data sources. ChipScope was used to view the packets generated by the workstations' GigE cards and to extract low-level performance information about the NIDS design.

6.3. Minimal Rule Set Implementations

The initial set of experiments performed on the NIDS utilized ID cores that contained minimal rule sets to allow the performance of the rest of the system to be studied. First, a dummy ID module with negligible overhead was used to determine that the packet FIFOs have a maximum clock rate of 185 MHz. This information was used later in the development process to steer optimization efforts when real ID cores were applied to the NIDS.

A second experiment was conducted to observe resource allocations with a functional ID module. This ID core was generated to filter packets that matched one Snort rule. Table 1 presents the BlockRAM (BRAM), lookup table (LUT), and total slice allocations in a V2P7 FPGA for various NIDS configurations. In terms of buffering, the results reveal that doubling the FIFO queue size does not cause dramatic changes to logic utilization. However, the BlockRAMs used to implement the FIFO queues are a limited resource. The NIDS with two filter bridges does not have enough available BlockRAM to implement 16 KB queues. Increasing the width of the ID data path does cause a more noticeable jump in utilization. Fortunately, a doubling in width does not result in a doubling of size.

Table 1: FPGA resource utilization of 125 MHz single-rule ID module

Filter Bridges	GigE NI Modules	ID Interface Width	FIFO Size	V2P7 BRAM Utilization	V2P7 LUT Utilization	V2P7 Slice Utilization
1	2	16	8 KB	36 %	23 %	38 %
			16 KB	72 %	24 %	37 %
		32	8 KB	36 %	25 %	39 %
			16 KB	72 %	25 %	40 %
		64	8 KB	36 %	28 %	45 %
			16 KB	72 %	29 %	47 %
2	4	16	8 KB	72 %	43 %	63 %
		32	8 KB	72 %	46 %	68 %
		64	8 KB	72 %	51 %	82 %

Table 2: Rule set capacity and throughput of 125 MHz ID module (V2P7)

Filter Bridges	Aggregate Network Rate	16-bit ID (2 Gb/s)		32-bit ID (4 Gb/s)		64-bit ID (8 Gb/s)	
		Rules	Chars	Rules	Chars	Rules	Chars
1	2 Gb/s	130	1,046	71	505	21	250
2	4 Gb/s	71	505	21	250	14	200

6.4. Snort Rule Set Implementations

A Snort-based ID module was generated and inserted into the design to discover how many detection rules could be supported in the V2P7 FPGA. The design was configured to use 8 KB packet FIFOs in order to maximize the number of rules. Table 2 shows the approximate number of rules and pattern characters that can be implemented in the V2P7 as well as the throughput of designs with one and two filter bridges for different ID interface widths. The ID units in each of these designs were set to run at 125 MHz, which is twice the frequency of the GigE units. The higher clock rate enables an ID module with 16-bit input to meet the 2 Gb/s aggregate data rate of a full-duplex filter bridge. To ensure that the ID module can keep up with data from two filter bridges, the input width must be increased to at least 32 bits. The designs using a 64-bit ID interface could support four filter bridges on a board with eight GigE ports.

6.5. Latency Measurements

Tests were performed on the NIDS to determine the amount of time the NIDS requires to process packets. In the first set of these tests, ChipScope was used to count the number of clock cycles that take place between events in the data flow of the NIDS. We first measured the amount of latency associated with sending and receiving data using the Rocket I/O transceivers. This measurement was performed by connecting two filter bridges in series using an external cable and injecting packets into one end of the system. ChipScope was used to count the number of clock cycles between when the packet first touches the Rocket I/O output interface of the first bridge, and when data first appears on the Rocket I/O input interface of the second bridge. We observed a latency of 40 clock cycles at 62.5 MHz, which equates to a delay of 640 ns. Similar measurements were performed on the internals of the NIDS. Latencies of 2.4 μs and 1.6 μs were observed for systems that used an ID module with 1x and 2x data rates, respectively.

Another set of tests was performed to observe the effects of the NIDS on end applications. The NIDS was inserted between a pair of hosts that were connected by a GigE link. A simple UDP-based, ping-pong benchmark was constructed to measure the amount of time required for a message to be sent from one host to another, and then returned to the sender. Both short (43 byte) and long (1024 byte) messages were transmitted in these tests. When the hosts were

connected directly (i.e., without the NIDS in place), round-trip timings for short and long messages were approximately 119 μs and 224 μs, respectively. Inserting a single pass through the NIDS increased the timings to 123 μs and 244 μs. Making two passes through the NIDS (i.e. through two filters) resulted in timings of 128 μs and 291 μs. These tests illustrated that the NIDS does add a small amount of delay, and that the delay increases with message size. However, this overhead is negligible compared to the typical delays found in GigE networks.

6.6. Observations

Multiple observations can be made from the experiments performed with the multi-filter bridge design. First, data-path bit width affects the NI and ID portions of the design differently, due to the functionality of the units. The ID module is more sensitive to bit-width increases because it has a more computational nature than the NI modules. For a fixed amount of logic, a narrower bit-width data path results in larger number of instantiated rules. However, wider ID modules are still preferable in designs with multiple filter bridges because the necessary ID data rates cannot be obtained by clock rate increases alone, and it is too costly to replicate the ID module for each bridge.

Another observation of this work is that the architecture adds a small amount of latency to data transmissions. This delay is due to the fact that a message cannot be processed until it is received in its entirety. Cut-through techniques could be applied to improve performance. However, doing so is challenging in the context of multiple NIs sharing the same ID module.

Finally, while the V2P7 has proven to be a flexible platform for NIDS work, it lacks the logic capacity to support a large number of detection rules and functional network interface cores. If all eight of the V2P7's Rocket I/O modules were utilized for GigE interfaces, there would be little room for any additional circuitry. Even with fewer GigE interfaces, only a small number of intrusion detection rules can be implemented in the V2P7. In order to handle a production-level number of rules, a larger capacity FPGA would need to be employed.

7. Design Space Exploration

Due to the limited capacity of the V2P7 FPGA on our reference board, only a fraction of the default Snort rule set could be implemented. As a means of investigating more practical rule sets, we adapted the design to target larger FPGAs. This allowed us to study the trade-offs associated with some design parameters and to determine the maximum rule capacity of various FPGA devices.

7.1. Area-Throughput Trade-off

As mentioned previously, the throughput of the ID module can be increased by making its input data bus wider, but there is a trade-off between throughput and area. In order for the payload analyzer to process additional data in the same amount of time, its area utilization

must be increased. As described in section 4.3, this area increase is due to the fact that additional logic must be instantiated for each pattern in the rule set.

In order to show the effects of the area-throughput trade-off, multiple NIDS designs were compiled using the same rule set but with different configurations. The rule set used was taken from the Snort distribution and contained 246 rules, 218 pattern strings, and 2001 pattern characters. Twelve different design configurations were compiled, and the number of FPGA slices utilized by each was measured. Two design parameters were varied to create these designs: the number of GigE bridges and the ID input bus width. The number of bridges was varied from one to four. Each bridge is capable of sending up to 2 Gb/s of data to the ID module. The ID module was operated at twice the GigE clock rate, and input bus widths of 16, 32, and 64 bits were used, resulting in supported throughputs of 2, 4, and 8 Gb/s, respectively.

The slice utilization (area) of each design is plotted in Figure 6. The target FPGA used in these experiments was a V2P50. From the chart, it can be seen that, for designs with the same number of bridges, the area increases linearly with increases in ID input width, as expected. Also of note is that the bridges use relatively little area and that their size is nearly independent of ID input width.

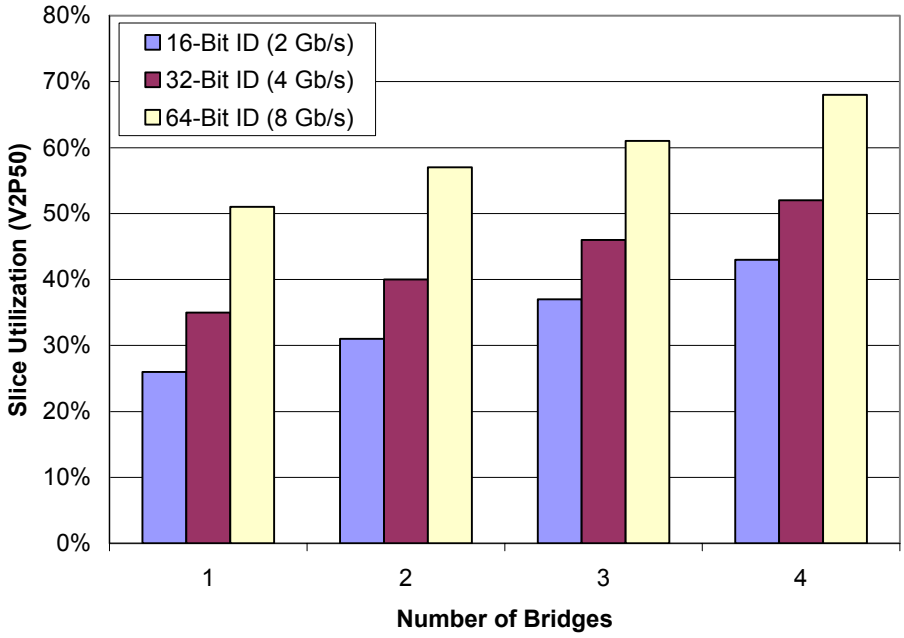

Figure 6: Area-throughput trade-off

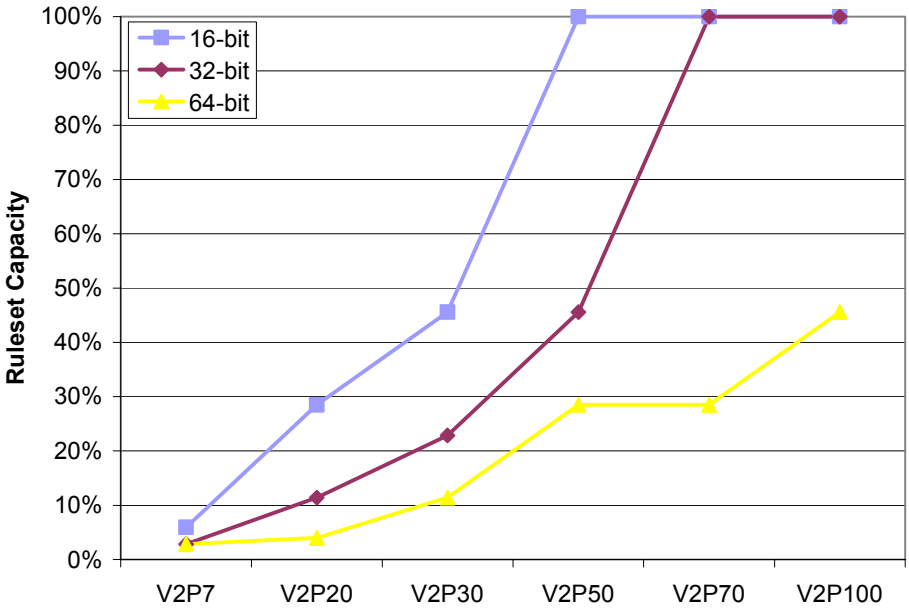

Figure 7: Rule set capacity of different FPGA devices

7.2. Rule Set Capacity

Capacity is defined as the maximum number of Snort rules that can be stored for a particular configuration of bridges, ID width, and FPGA device. The complete default rule set included in the Snort 2.0 distribution contains 1305 rules, 1512 pattern strings, and 17 537 pattern characters. We used different-sized subsets of the default rule set to determine the capacity of several FPGA devices from the Virtex II Pro family. For each chosen device, we compiled designs with increasingly larger subsets until at least 85% of the slices were utilized. This process was repeated for ID widths of 16, 32, and 64 bits. The capacities of different devices relative to the default rule set size are shown by the graph in Figure 7. In order to leave more room for ID rules, all of the designs in this experiment used a single GigE bridge. A V2P50 FPGA can support the entire default rule set when a 16-bit ID module is used, while a V2P70 FPGA is required for a 32-bit ID module. With a 64-bit ID module, about 50% of the default rule set fits into the largest device in the Virtex II Pro family, the V2P100.

8. Future work

There are a variety of opportunities for future work in this area. At a high level, the effectiveness of the NIDS can be improved by considering techniques through which intrusion detection information captured at one network point can be shared with other interested parties or devices. The V2P's PowerPC processor would serve as a good processing substrate for

coordinating the distribution of this information. Our initial experiments indicate that the PowerPC is suitable for capturing runtime statistics about the NIDS, and that serial or network ports can be used to exchange this information with external systems.

Another avenue of investigation for this work involves an exploration of how high-capacity FPGAs can be better leveraged in a large-scale NIDS. Our experiments with the V2P100 device indicate that current FPGAs are capable of operating at high throughput with large rule sets. However, compiling these circuits is very time consuming, taking many hours to complete on state-of-the-art workstations. The compilation times and clock speeds can be improved with better floor-planning. Therefore, the next step in this work will involve refining how placement information is added to the ID cores when the circuit descriptions are generated.

Finally, this work may be extended by considering methods in which an FPGA-based NIDS is incrementally updated with new patterns over time. In our current approach, the NIDS must be taken offline briefly and reconfigured whenever rule updates need to be applied. Because updates are infrequent and require only a few seconds of downtime, this approach is acceptable for many applications. However, if high availability is required, partial reconfiguration techniques are potential solutions. With partial reconfiguration, the NI modules would continue to operate and buffer incoming messages while the ID module is updated with new circuitry.

9. Summary

Network intrusion detection systems (NIDS) are a necessary tool for monitoring and protecting computer networks from malicious users. As network data rates have increased over the years, it has become necessary to consider new NIDS architectures that are able to meet stringent constraints. In this work, we have presented a flexible, high-performance NIDS architecture that integrates network interface and intrusion detection circuitry into a single FPGA chip. This integration eliminates the NI to ID data transfer bottleneck that has limited the performance of some NIDS designs. We have shown that a modern FPGA is capable of performing in-line intrusion detection and packet filtering on multiple Gigabit Ethernet links.

References

BAKER, Z. K. and PRASANNA, V. K., 2004, A Methodology for Synthesis of Efficient Intrusion Detection Systems on FPGAs. Proceedings of *IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 135-144.

BELLOWS, P. and HUTCHINGS, B. L., 1998, JHDL-An HDL for Reconfigurable Systems. Proceedings of *IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 175-184.

CHO, Y. H. and MANGIONE-SMITH, W. H., 2004, Deep Packet Filter with Dedicated Logic and Read Only Memories. Proceedings of *IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 125-134.

CLARK, C. R., LEE, W., SCHIMMEL, D. E., CONTIS, D., KONÉ, M. and THOMAS, A., 2004, A Hardware Platform for Network Intrusion Detection and Prevention. Proceedings of *Workshop on Network Processors and Applications at HPCA (NP-3)*, pp. 136-145.

CLARK, C. R. and SCHIMMEL, D. E., 2004, Scalable Pattern Matching for High-Speed Networks. Proceedings of *IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 249-257.

DHARMAPURIKAR, S., KRISHNAMURTHY, P., SPROULL, T. and LOCKWOOD, J. W., 2003, Deep Packet Inspection Using Parallel Bloom Filters. Proceedings of *Symposium on High Performance Interconnects (HotI)*, Stanford, CA, USA, pp. 44-51.

FRANKLIN, R., CARVER, D. and HUTCHINGS, B. L., 2002, Assisting Network Intrusion Detection with Reconfigurable Hardware. Proceedings of *IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 111-120.

GOKHALE, M., DUBOIS, D., DUBOIS, A., BOORMAN, M., POOLE, S. and HOGSETT, V., 2002, Granidt: Towards Gigabit Rate Network Intrusion Detection Technology. Proceedings of *International Conference on Field Programmable Logic and Applications (FPL)*, pp. 404-413.

InfiniBand Trade Association. http://www.infinibandta.org

IEEE, Get IEEE 802.3. http://standards.ieee.org/getieee802/802.3.html

LI, S., TORESSEN, J. and SORAASEN, O., 2003, Exploiting stateful inspection of network security in reconfigurable hardware. Proceedings of *International Conference on Field Programmable Logic and Applications (FPL)*, pp. 1153-157.

LOCKWOOD, J. W., NEELY, C., ZUVER, C., MOSCOLA, J., DHARMAPURIKAR, S. and LIM, D., 2003, An Extensible, System-On-Programmable-Chip, Content-Aware Internet Firewall. Proceedings of *International Conference on Field Programmable Logic and Applications (FPL)*, pp. 859-868.

MOSCOLA, J., LOCKWOOD, J. W., LOUI, R. P. and PACHOS, M., 2003, Implementation of a Content-Scanning Module for an Internet Firewall. Proceedings of *IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 31-38.

NECKER, M., CONTIS, D. and SCHIMMEL, D. E., 2002, TCP-Stream reassembly and state tracking in hardware. Proceedings of *IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 286-287.

ROESCH, M., 1999, Snort - Lightweight Intrusion Detection for Networks. Proceedings of *USENIX LISA Conference*.

SCHUEHLER, D. and LOCKWOOD, J., 2004, A Modular System for FPGA-based TCP Flow Processing in High-Speed Networks. Proceedings of *International Conference on Field Programmable Logic and Applications (FPL)*, pp. 301-310.

SOURDIS, I. and PNEVMATIKATOS, D., 2004, Pre-Decoded CAMs for Efficient and High-Speed NIDS Pattern Matching. Proceedings of *IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 258-267.

Xilinx, Inc., ChipScope Pro. http://www.xilinx.com/xlnx/xebiz/designResources/ ip_product_details.jsp?key=DO-CSP-PRO

Xilinx, Inc., Ethernet LogiCore. http://www.xilinx.com/products/design_resources/conn_central/grouping/ethernet.htm

Xilinx, Inc., Xilinx ML300 Overview. http://www.xilinx.com/products/boards/ml300/

Xilinx, Inc., Virtex-II Pro: RocketIO. http://www.xilinx.com/products/virtex2pro/rocketio.htm

Xilinx, Inc., Virtex-II Pro Platform FPGAs. http://www.xilinx.com/xlnx/xil_prodcat_landingpage.jsp?title=Virtex-II+Pro+FPGAs