NETWORK INTRUSION DETECTION SYSTEMS ON FPGAS WITH ON-CHIP NETWORK INTERFACES

Christopher R. Clark
Georgia Institute of Technology
Atlanta, Georgia, U.S.A.
cclark@ece.gatech.edu

Craig D. Ulmer
Sandia National Laboratories[†]
Livermore, California, U.S.A.
cdulmer@sandia.gov

ABSTRACT

Network intrusion detection systems (NIDS) are critical network security tools that help protect distributed 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 a NIDS. In this work, we present the next logical step in NIDS architecture: the integration of network interface hardware and packet analysis hardware into a single FPGA chip. This integration allows for better customization of the NIDS as well as a more flexible foundation for network security operations. 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 filtering on multiple Gigabit Ethernet links using rules from the Snort attack database.

KEYWORDS

NIDS, FPGA, Gigabit Ethernet, Pattern Matching

1. BACKGROUND

1.1 Network Intrusion Detection Systems

A network intrusion detection system (NIDS) is a combination of hardware and software that monitors a computer network for attempts to violate a security policy. In addition to simply observing network operations, a NIDS can also be designed to react to assaults, either by filtering attack 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) 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 on commodity servers equipped with high-speed network-interface cards. While economical and easy to program, this approach is limited in terms of performance 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 optimized for the large pattern-matching operations that are commonly used in ID operations. As such, software-based NIDS are often unable to keep up with the data rates of modern high-speed networks.

[†] 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.

As a means of improving NIDS performance, researchers have turned to reconfigurable computing platforms where ID operations can be performed in custom hardware. In recent years a number of researchers have proposed a variety of ID architectures for field-programmable gate arrays (FPGAs) [Franklin et al. 2002; Gokhale et al. 2002; Moscola et al. 2003; Baker and Prasanna 2004; Cho and Mangione-Smith 2004; Clark and Schimmel 2004; Sourdis and Pnevmatikatos 2004]. While the previous work describes significant gains for FPGAs over CPU-based approaches, it has largely been focused on only the ID portion of a NIDS. Most of these efforts defer the task of interacting with the network to other devices without consideration for the issues involved in transferring data between the NI and ID components. In this paper, we describe one approach to designing an efficient NI-ID interface through the integration of the NI and ID circuitry, providing a complete NIDS on a single FPGA device.

1.2 Modern Field-Programmable Gate Arrays

Field-programmable gate arrays (FPGAs) are reconfigurable hardware devices that can be programmed to emulate custom application-specific circuitry. Recent commercial FPGAs have evolved into more than just simple arrays of reconfigurable logic. These "platform FPGAs" are essentially system-on-a-chip (SoC) 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 the work presented in this paper is 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 various circuits for network-specific operations, such as 8B/10B encoding, serialization/deserialization, embedded clock recovery, and basic CRC verification. The V2P100 is the largest FPGA in the V2P family. This FPGA supports 20 Rocket I/O ports, with each port operating at up to 3.125 Gb/s full-duplex.

2. AN INTEGRATED NIDS IN AN FPGA

The fact that modern FPGAs are capable of interacting directly with a physical network enables us to propose the next natural step in the evolution of NIDS architecture: the integration of network interface and intrusion detection circuitry in a single platform FPGA. We argue that there are multiple advantages to consolidating NIDS components into a single FPGA. From a physical design perspective, integration simplifies board layout and reduces the overall chip count for a NIDS. Integration also removes the need for off-chip data transfers, which can be a performance bottleneck in multi-chip systems. However, the main benefit for 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 place 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 example, the NI units in many NIDS applications blindly transfer data into and out of the network in an unsophisticated manner. As we discuss later in this paper, this knowledge can be used to simplify the NI implementation in order to conserve FPGA resources and decrease processing overhead. Customization can also be applied to higher levels in 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, in our work we have used a single reference board to implement multiple types of NIDS platforms. If the NI cores were not implemented in the FPGA, we would have had to fabricate multiple reference boards.

The remainder of this paper addresses issues associated with constructing a complete NIDS in a single platform FPGA. Sections 3 and 4 describe the development of the NI and ID units, with an emphasis on leveraging architectural features of the Xilinx Virtex II/Pro FPGA. A motivating example of a complete NIDS is presented in Section 5. In this example, we describe a NIDS implementation that transparently

monitors and filters multiple GigE connections. Implementation details are presented for a four-port GigE system. Finally, the paper is concluded with some observations about this work and possibilities for future work.

3. FPGA NETWORK INTERFACE

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 unit are transmit and receive controllers for the Rocket I/O core and packet FIFOs for in-flight messages. The current implementation of the GigE NI does not support jumbo frames due to the limited amount of buffer space available in the V2P FPGA.

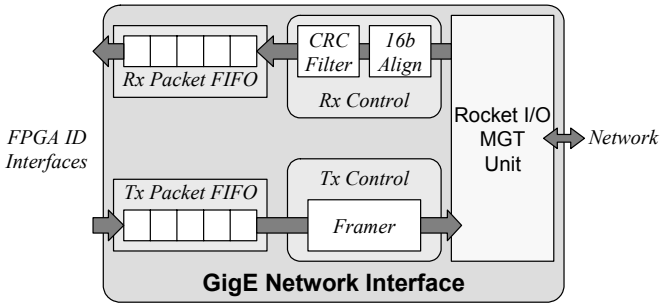

Figure 1: Gigabit Ethernet network interface

3.1 Rocket I/O Controllers

At the heart of a 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 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 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 unit from generating false positive matches. Additional circuitry monitors the physical layer's status in the Rocket I/O module, and automatically resets the unit as needed.

3.2 Packet FIFOs

The final component of the NI unit is a pair of packet FIFOs for buffering incoming and outgoing packets. These units are implemented out of a small number of dual-ported SRAM 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 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 pass/fail flag to be asserted after an injection, in order to force the last inserted message to be ignored by the consumer. This feature is useful in a NIDS, where the ID unit'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 units from the ID unit. This decoupling allows the ID unit to operate at a higher clock rate and with wider data streams. These factors enable the NIDS to share an ID unit among multiple NI cores.

3.3 Resource Utilization

While the V2P architecture has been available for over two years, there have been few reports on the network capabilities of the devices in academic literature. The placed and routed design for the GigE NI used in this work was analyzed to better characterize the hardware. A GigE NI with 4KB packet FIFOs was found to occupy 749 V2P slices, which consumes approximately 15% of a V2P7 FPGA and less than 2% of a V2P100 FPGA. While smaller NI cores are certainly possible, we believe that this NI design is a relatively compact implementation that provides the basic features necessary for our NIDS work. The send and receive packet FIFOs each account for a third of the GigE NI's resources. The send and receive state machines consume the remaining third of the NI core, with the send state machine's size being approximately half that of the receive state machine's. This difference in size can be attributed to the fact that the sending process is less sophisticated than the receiving process for this application. 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 UNIT

An FPGA intrusion detection (ID) 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 availability and quality. The rule language supports Boolean expressions on the packet header fields, as well as complex regular expressions 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.

Software was written in Java that parses Snort-format rule files and translates the rules into an internal representation. The Java Hardware Description Language (JHDL) [Bellows and Hutchings 1998] was used to write classes that translate the internal rule representation into circuits capable of detecting packets meeting the rules' specifications. The JHDL translator assembles functional blocks into circuits that check all rules in parallel as packet data flows though the system. 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 to be two, four, or eight bytes. A block diagram of the ID core is shown in Figure 2. Each of the components is described in detail in the following sections.

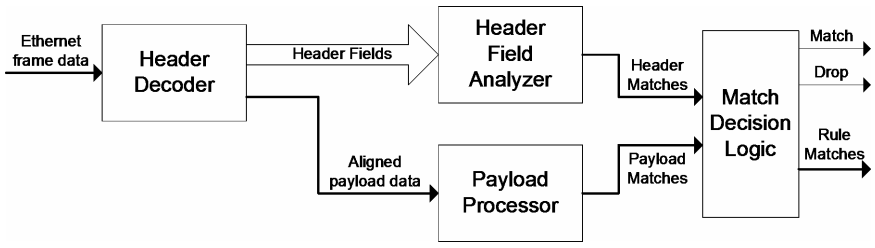

Figure 2: Intrusion detection (ID) core

4.1 Header Decoder

The header decoder accepts frames from an Ethernet interface and extracts header fields from the data-link, network, and transport layers of the protocol stack. The supported protocols include Ethernet, IP, ARP, ICMP, TCP, and UDP. All of the relevant header fields are stored in registers and made available to the field analyzer. Also, for each packet, the decoder determines the end of the transport layer header and produces a signal for the payload processor 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 core, the start of the payload data does not always occur on an input word boundary. It is important for the payload processor to analyze every byte of the payload to avoid missing a potential 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 Field Analyzer

The header field 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 field 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 Processor

The payload processor searches each packet payload for the occurrence of patterns specified by the detection rules. The input data is compared against all patterns in parallel at the rate of two, four, or eight bytes per clock cycle. 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-matcher design uses a non-deterministic finite state automata (NFA) approach described in previous work [Clark and Schimmel 2004]. The comparisons are pipelined and the match state is distributed throughout the pipeline. Therefore, no buffering or back-tracking of the input data is required. The NFA pipeline is implemented as static logic, which provides high performance but requires the design to be re-compiled in order to change the patterns stored in the FPGA.

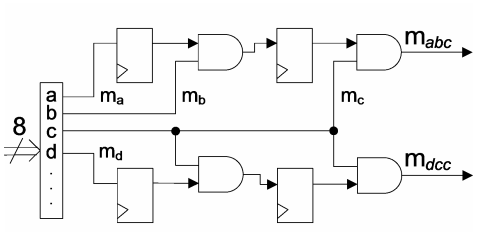

Figure 3: Pattern-matching circuit for patterns “abc” and “dcc”

Figure 3 shows a section of a pattern-matching circuit. Each input character is decoded into a set of single-bit character match indicator wires. For 8-bit characters there are 2^8, or 256, output wires from the decoder. Only one of these wires will be asserted for each input character—the wire corresponding to the value of the current character. The match indicator wires are distributed to the NFA pipelines created for each pattern. Each stage of the pipeline outputs true if the output of the previous stage was true and the current stage’s match indicator wire is true. If there is a character sequence in the input that matches one of the stored

patterns, a true value will propagate through the pipeline and the match output bit will be asserted for the corresponding pattern.

4.4 Match Decision Logic

For each packet, the match decision logic uses the match outputs from the header field analyzer and the payload processor to determine which rules, if any, were triggered by the packet. If one or more rules are triggered, this unit decides what action should be taken based on the policy specified in the rule file for the matching rule(s). If any of the triggered rules indicate that the packet should be blocked, the drop output will be asserted. The match output will be asserted if any rule is triggered. The unit also outputs a list of the matching rules.

5. A MULTI-LINK NIDS IMPLEMENTATION

A complete, FPGA-based network intrusion detection system was 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 better demonstrating the flexible nature of FPGAs for NIDS, the implementation was adapted to support multiple filter bridges in a single FPGA. In this architecture, a single intrusion detection core is shared by multiple filter bridges in the FPGA. By operating the ID unit at a data rate that is sufficiently greater than the data rates of the NI units, the system can operate without unintentional packet loss. The design was tested using a commercial FPGA development card with four GigE interfaces.

5.1 Multiple Filter Bridge Architecture

A simplified diagram of the multi-filter bridge architecture is presented in Figure 4. In this NIDS, each filter bridge has two data paths that are routed through the ID unit (one for each direction of the full-duplex bridge). Time-division multiplexing is employed to share the ID unit among the NI units 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 unit. The current implementation utilizes a round-robin scheduler, although more sophisticated algorithms could easily be applied.

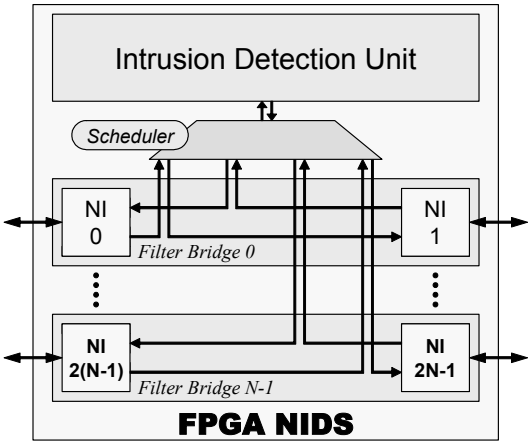

Figure 4: Multi-filter bridge in a single FPGA

Packet loss in the multi-filter bridge design can be avoided if the ID unit 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 2*N* NIs would require an ID unit that processed data at a rate that is at least 2*N* Gb/s. Operating the ID unit 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 units. 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 channel speedup of 2x or 4x. Given the current speed grades of the V2P, it is conceivable that an additional 2x of data rate speedup can be achieved by applying tighter timing constraints to the design at compile time.

5.2 Testbench Hardware

The NIDS design was implemented and tested using Xilinx's ML300 reference 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 GigE network ports that are driven by the FPGA's Rocket I/O modules. The ML300's FPGA is a V2P7-6, which is one of the smallest FPGAs in the V2P family. This FPGA has roughly $1/9^{th}$ the logic capacity as 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 that have Alteon ACEnic GigE cards. For tests that involved two filter bridges, a GigE cable was used to 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 the NIDS, where system behavior is highly dependent on external data sources. We used ChipScope to view the packets that the workstations' GigE cards were generating and to extract low-level performance information about the NIDS design.

5.3 NIDS Characteristics for a Minimal Rule Set

The initial set of tests performed on the NIDS utilized ID cores that contained minimal rule sets. The first test utilized a dummy ID module that simply dropped every other packet that passed through the system. This module had negligible overhead and therefore represented an ideal ID unit. The test revealed 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.

Table 1: Resource utilization for single-rule ID unit operating at 130 MHz

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

A second minimal rule set test was constructed to observe resource allocations with a functional ID unit. This test included an ID core that was generated to filter packets that matched one Snort rule. Table 1 presents the lookup table (LUT) and total slice allocations in a V2P7 FPGA for various NIDS configurations. In terms of buffering, this test revealed that doubling the FIFO queue size did not cause dramatic changes to

logic utilization. However, the block RAMs used to implement the FIFO queues are a limited resource. The NIDS with two filter bridges did not have enough block RAM to implement 16 KB queues. Increasing the width of the ID data path did cause a more noticeable jump in utilization. Fortunately, a doubling in width did not result in a doubling of size. Finally, it is important to observe that slice utilization was higher than LUT utilization. This is to be expected, given that the ID implementation style selected for this work favors routing pre-computed signals rather than computing values at every pattern matching block.

5.4 Snort Rule Set Implementations

A Snort-based ID system was constructed and inserted into the design to discover how many detection rules could be supported in the V2P7 FPGA. The design was configured to use 8KB packet FIFOs in order to maximize the number of rules. Table 2 shows the approximate number of rules and pattern characters that can be stored 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 unit with 16-bit input to meet the 2 Gb/s aggregate data rate of a full-duplex filter bridge. To ensure that the ID unit 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.

Table 2: ID capacity and throughput at 125 MHz (V2P7)

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

Due to the limited capacity of the V2P7, only a fraction of the default Snort rule set could be implemented in our reference board. As a means of investigating larger rule sets, we adapted the design to target the Xilinx V2P100 FPGA. We generated 16-, 32-, and 64-bit ID cores that implemented the complete Snort rule set (1,299 rules with 17,514 characters), and compiled NIDS designs that implemented a single filter bridge. While the 64-bit design was too large to synthesize, the 16- and 32-bit designs compiled in approximately 36 hours. The 16-bit design utilized 32% of the chip's LUTs and 47% of its slices, while the 32-bit design utilized 36% of the LUTs and 74% of the slices. The maximum clock rate for these designs dropped to 75 MHz. Although the large designs could not achieve a 2x clock rate, the 32-bit design using a 1x clock rate is capable of supporting the complete Snort rule set for one filter bridge. A close inspection of the placed and routed design revealed that the compilation tools had placed the NI modules and ID unit at opposite ends of the chip. While the V2P100 clearly has the capacity for implementing the complete Snort rule set, this test indicates that additional placement guidance is necessary to meet desired timing requirements in large designs.

5.5 Overhead

Tests were performed on the NIDS to determine the amount of delay 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 unit 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.

5.6 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 unit is more sensitive to bit-width increases because it has a more computational nature than the NI units. For a fixed capacity, a narrower bit-width data path results in larger number of instantiated rules. However, wider ID units are still preferable in designs with multiple filter bridges, because the necessary ID data rates cannot be obtained by clock rate increases alone.

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 unit.

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 stored in the V2P7. In order to handle a production-level number of rules, a larger capacity FPGA would need to be employed. We have found that the V2P100 has ample room for the complete rule set, and expect that the V2P50 or V2P70 would be suitable candidates and more cost-effective.

6. FUTURE WORK

There are a variety of opportunities for future work in this area. First, this NIDS work can be improved upon by considering techniques by 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 architecture indicate that current FPGAs are large enough to house very large rule sets. However, compiling these circuits is very time consuming, taking as long as two days to complete on state-of-the-art workstations. These 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 pattern matching circuitry is generated.

Finally, this work can be improved upon 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 units would buffer incoming messages while the ID unit is updated with new circuitry.

7. SUMMARY

Network intrusion detection systems (NIDS) are a necessary tool for monitoring and protecting computer networks from malicious users. As network data rates and malicious user sophistication have increased over the years, it is necessary to consider new NIDS architectures that will be able to meet stringent constraints. In this work we have presented the next logical step in NIDS architecture: the integration of network interface hardware and reconfigurable pattern matching hardware in a single FPGA system. We have implemented a complete and functional NIDS in a commercial FPGA chip. This system performs in-line packet filtering on multiple Gigabit Ethernet links using intrusion detection rules based on the Snort rule set.

REFERENCES

Baker, Z. K. and Prasanna, V. K., 2004, A Methodology for Synthesis of Efficient Intrusion Detection Systems on FPGAs. 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. 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. IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM), pp. 125-134.

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

Franklin, R., et al., 2002, Assisting Network Intrusion Detection with Reconfigurable Hardware. IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM), pp. 111-120.

Gokhale, M., et al., 2002, Granidt: Towards Gigabit Rate Network Intrusion Detection Technology. 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

Moscola, J., et al., 2003, Implementation of a Content-Scanning Module for an Internet Firewall. IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM), pp. 31-38.

Roesch, M., 1999, Snort - Lightweight Intrusion Detection for Networks. USENIX LISA Conference.

Sourdis, I. and Pnevmatikatos, D., 2004, Pre-Decoded CAMs for Efficient and High-Speed NIDS Pattern Matching. 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., 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