PAPER *Special Section on Reconfigurable Systems*

FPGA-Based Intrusion Detection System for 10 Gigabit Ethernet

Toshihiro KATASHITA[†a)], **Yoshinori YAMAGUCHI**[††], ***Members***, **Atusi MAEDA**[††], ***Nonmember***, ***and*** **Kenji TODA**[†], ***Member***

SUMMARY The present paper describes an implementation of an intrusion detection system (IDS) on an FPGA for 10 Gigabit Ethernet. The system includes an exact string matching circuit for 1,225 Snort rules on a single device. A number of studies have examined string matching circuits for IDS. However, implementing a circuit that processes a large rule set at high throughput is difficult. In a previous study, we proposed a method for generating an NFA-based string matching circuit that has expandability of processing data width and drastically reduced resource requirements. In the present paper, we implement an IDS circuit that processes 1,225 Snort rules at 10 Gbps with a single Xilinx Virtex-II Pro xc2vp-100 using the NFA-based method. The proposed circuit also provides packet filtering for an intrusion protection system (IPS). In addition, we developed a tool for automatically generating the Verilog HDL source code of the IDS circuit from a Snort rule set. Using the FPGA and the IDS circuit generator, the proposed system is able to update the matching rules corresponding to new intrusions and attacks. We implemented the IDS circuit on an FPGA board and evaluated its accuracy and throughput. As a result, we confirmed in a test that the circuit detects attacks perfectly at the wire speed of 10 Gigabit Ethernet.

key words: *intrusion detection system, intrusion protection system, exact string matching, FPGA, 10 Gigabit Ethernet*

1. Introduction

The purpose of the present paper is to propose a practical approach to a complete construction of an intrusion detection system (IDS) based on the FPGA and to demonstrate the feasibility of its implementation on 10 Gigabit Ethernet.

As network services become increasingly important in our society, the demand for network security systems is increasing. The IDS is one such system that inspects network traffic and detects intrusions and attacks. The system is required to process the traffic at wire speed and to support a large rule set in order to detect harmful packets without omissions. Snort [1] is a widely used open source software-based IDS. However, the processing speed of Snort is much slower than wire-speed because of the software-based string matching. In order to accelerate Snort, several approaches to string matching circuits have been investigated. A string matching circuit based on the Nondeterministic Finite Automaton (NFA) was proposed in [2]–[4]. This circuit has expandability of throughput and flexibility of updating matching patterns with an FPGA. In a previous study, we proposed a method to generate a lightweight scalable NFA-based string matching circuit [5]. Although an automated circuit design was also proposed in [6], their circuit was not implemented in a practical environment and the accuracy of the circuit was not verified.

In the present paper, we implemented the proposed IDS circuit, which processes 1,225 rules at 10 Gbps on a single FPGA. The circuit has the following features: packet classification, header inspection, payload inspection, Snort rule identification, intrusion notification, and intrusion protection. We evaluated the circuit in a 10 Gigabit Ethernet environment. As a result, the circuit detected attacks and intrusions correctly at the wire speed of 10 Gigabit Ethernet. We also developed a tool that automatically generates the circuit from the Snort rule set using the NFA-based method. The proposed circuit can be updated by the automation tool and the FPGA device.

The present paper is organized as follows. Section 2 introduces related research. Section 3 describes the implementation and automated generation tool in detail. Section 4 presents the results of verification and evaluation. Section 5 shows two examples for the construction of practical systems. Section 6 describes areas for future research, and Sect. 7 presents conclusions.

2. Related Research

A number of studies have considered the use of the string matching circuit for the IDS, NFA [2]–[4], DFA [7], CAM [8], [9], KMP [10], and other memory-based approaches [11], [12]. Only the NFA-based approach was able to achieve 100-Gbps throughput, because it was difficult to extend the processing width in the other approaches.

Memory-based approaches that were able to update the rule with ASIC were presented. However, since online update was not supported, these approaches also needed duplicative systems to support the online-rule update. Although the KMP-based circuit that updated the matching rules online was proposed in [10], since the processing width was fixed to eight bits, the throughput was up to 2.4 Gbps.

An implementation approach of the NFA-based circuit was proposed in [2]. A method with a Self Reconfigurable Gate Array (SRGA) was presented to update the matching

Manuscript received April 10, 2007.
Manuscript revised June 29, 2007.

[†]The authors are with the National Institute of Advanced Industrial Science and Technology (AIST), Tsukuba-shi, 305–8568 Japan.
[††]The authors are with University of Tsukuba, Tsukuba-shi, 305–8577 Japan.
a) E-mail: t-katashita@aist.go.jp

DOI: 10.1093/ietisy/e90–d.12.1923

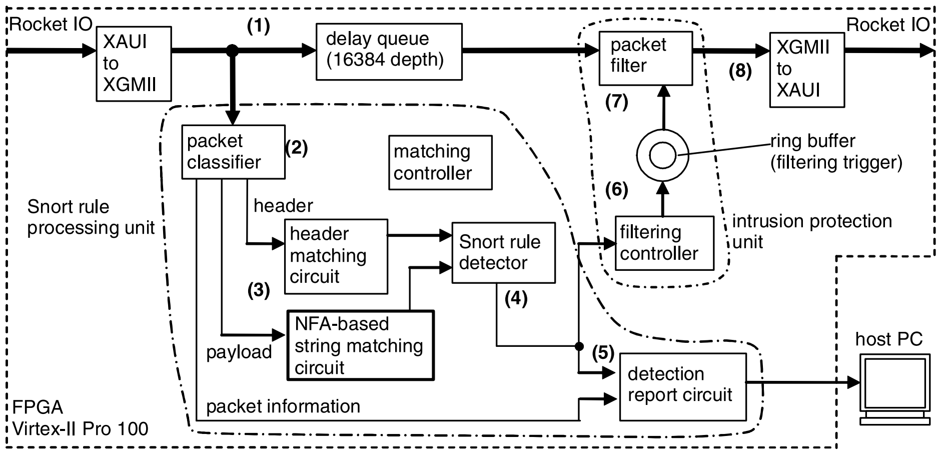

Fig. 1 Architecture of the IDS circuit.

rule dynamically. However, the SRGA requires much larger hardware resources than the FPGA because the SRGA has multiple configuration memory sets and a dynamic reconfiguration mechanism. Moreover, a practical SRGA device has not yet been developed.

An automated circuit generation technique was investigated in [6]. This approach was similar to that if the present study in terms simplifying the implementation by dividing the matching pattern. However, the method of division was not effective for NFA-based circuit because it maximized common characters in patterns for its CAM-based string matching circuit. Moreover, no circuit has been implemented practically on an FPGA board.

An IDS for 10 Gigabit Ethernet was proposed by Force10 [13]. This system supports up to 626 rules, whereas, as of Feb 20, 2007, there were 1,225 rules that have the "content" payload option in the Snort snapshot rule set.

3. IDS Implementation

In order to simplify the implementation of the IDS, we designed a simple flow-though architecture and developed a tool that automatically generates the circuit from the Snort rule set.

3.1 Architecture

Figure 1 shows the architecture of the IDS circuit. The circuit consists mainly of a Snort rule processing unit, an intrusion protection unit, a long delay queue, and 10 Gigabit Ethernet interfaces. The circuit inspects network traffic through the following five steps:

(1) Convert the input traffic format from XAUI (10 Gigabit Attachment Unit Interface) to XGMII (10 Gigabit Media Independent Interface) in order to handle "idle" patterns easily. It is difficult to handle "idle" on XAUI because there are three types of patterns, all of which are scrambled. In contrast, it is simple to use one type of pattern on XGMII. The converted traffic is placed on the delay queue and copied to the Snort rule processing unit. Invalid packets are deleted in this step.

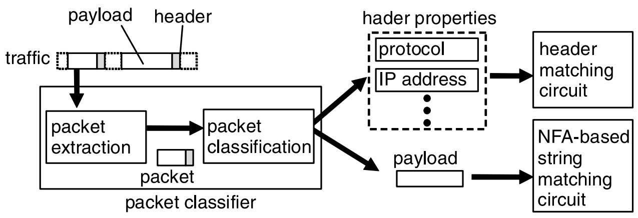

Fig. 2 Detail of Step (2).

(2) Extract packets from the traffic and divide them into a header and a payload. Packets are classified by protocol and type, i.e., UDP, TCP, IP, and ICMP. The header is split into properties, a frame type, an IP address, a port number, and so on (see Fig. 2).
(3) Inspect the header and the payload. The header properties are checked by the comparator arrays. The payload is searched for the matching patterns by the NFA-based string matching circuit. Matching result signals are generated for each rule and are passed to the Snort rule detector in parallel.
(4) Detect the Snort rule by combining the header and payload matching results. A rule detector for each rule generates the results in parallel. The result signals are masked by the mask resisters in order to inform the rule detection once at each packet. The results are encoded into an 11-bit ID and a validation bit at the priority encoder. Figure 3 shows the detail of the Snort rule detector.
(5) Report the intrusion detection result to a host PC. The detection report circuit counts and tabulates the frequency for each rule. The circuit also accumulates the lengths of matched and unmatched packets, respectively, at every 125 ms in order to measure the traffic throughput. These results are reported to the host PC.

The intrusion detection is processed within the flow of

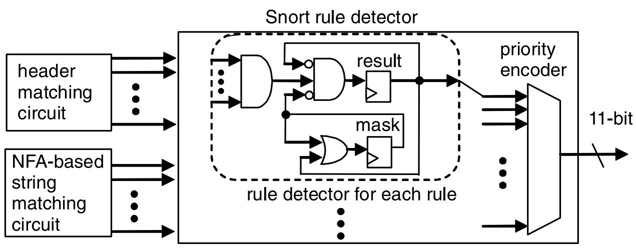

Fig. 3 Detail of Step (4).

network packets through the delay queue. Consequently, the circuit latency is of fixed length.

The circuit has an intrusion protection function wherein the matched attacks and intrusions are removed by the following three additional steps:

(6) Determine whether the matched packet should be removed. The filtering controller checks the result id and the validation bit and writes a filtering bit onto a ring buffer at end of each packet. The controller is configured as all the matched packets are removed in the case of implementation.
(7) Remove the suspicious packets according to the filtering bit. The packet filter loads the bit from the ring buffer when the packet leaves the delay queue. The packet that should be removed is replaced by the "idle" pattern of XGMII.
(8) Convert the output traffic format from XGMII to XAUI. Hence, the filtered packets are handled as "idle", and the attacks and intrusions are removed continuously at wire speed.

The main data-path of the circuit is 64 bits in width and so processes the traffic at 10 Gbps in 156.25-MHz clock operation. The latency and maximum packet length are associated with the depth of the delay queue of which the depth is configured to be 16,384, namely, the maximum packet length is 16,384 bytes in order to support jumbo frames. The latency is $16,384 + \alpha^{\dagger}$ cycles (approximately 105 μs).

Rule updating is supported offline by reconfiguring the FPGA. Depending on the rule, three blocks are primarily changed: the header matching circuit, the NFA-based string matching circuit, and the Snort rule detector. other blocks, such as the priority encoder and the detection report circuit, are fixed except for exceeding 2,048 rules, because the width of the ID bus in the implementation is 11 bits. We developed typical templates of the fixed blocks for the number of the rules.

3.2 Automatic Generation Tool

We developed an IDS circuit generation tool with Java. The tool automatically generates a Verilog HDL source code of the three changeable blocks from the Snort rule set. The other fixed circuits are provided as templates. The entire IDS circuit is constructed with the tool and the templates. Figure 4 shows the procedure for generating the circuit. The newly developed tool processes the following six steps:

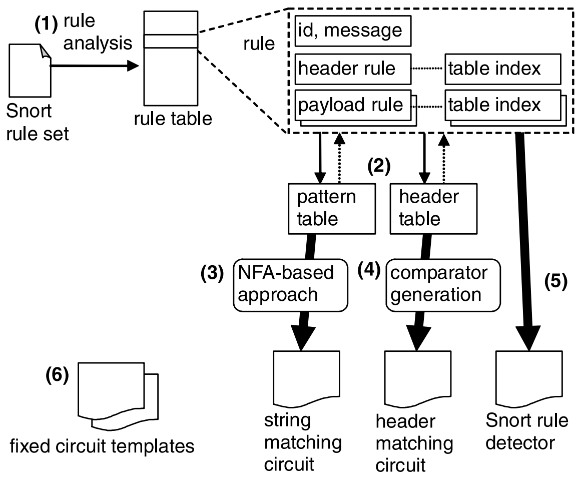

Fig. 4 Procedure for generating the IDS circuit.

(1) Analyze the Snort rule set. Each rule in the rule set is decoded into three properties: messages, a header rule, and payload rules.
(2) Enter the header and payload rules into the header and pattern table, respectively. Duplicative rules that are already registered in the tables are removed. When the property definition is preceded by a '!', a negation bit is enabled. The negation bit indicates that the signal is alerted if the property does not match. The table index and the negation bit are stored in the rule table. The table index indicates whether the header and payload matching result corresponds to the Snort rule.
(3) Generate the string matching circuit with the NFA-based approach [5] from the pattern table.
(4) Generate the header matching circuit employing comparator arrays. The comparator arrays are constructed independently for each source port, destination port, packet type, and so on. The comparator supports exact and range matching.
(5) Generate the rule detector for each rule. One-hot state machines are generated from the table indexes and the negation bits.
(6) Provide the templates of the other fixed circuits according to the number of rule IDs.

The tool generates the Verilog HDL source codes, a Snort message table, debug information, and a generation log. The tool finishes the circuit generation within one minute. The three changeable circuits have scalable architectures in the processing width. We have determined that the string matching circuit achieved over 100 Gbps with a 512-bit processing width [14]. In the implementation, we selected a 64-bit width for 10-Gbps processing in 156.25-MHz operation. The templates for 64-bit processing and the XAUI interface are developed. We also provide 8-bit processing templates and a GMII interface for Gigabit Ethernet.

The Snort message table is used to identify the Snort

[†]The latency of the XAUI-XGMII translator and the packet filtering circuit.

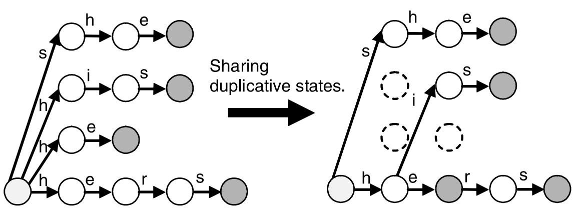

Fig. 5 Example of sharing duplicative states.

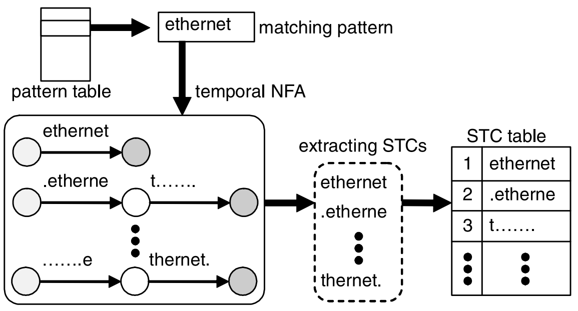

Fig. 6 Generating the string matching circuit: Step (1).

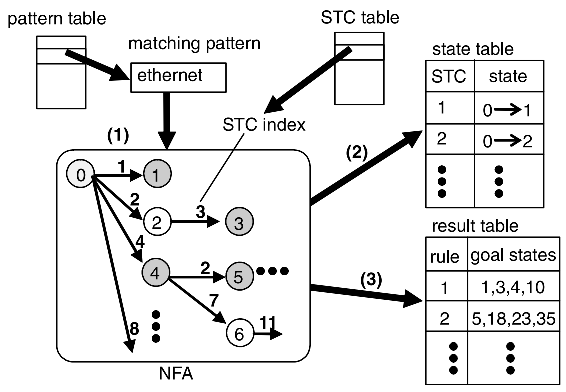

Fig. 7 Generating the string matching circuit: Step (2).

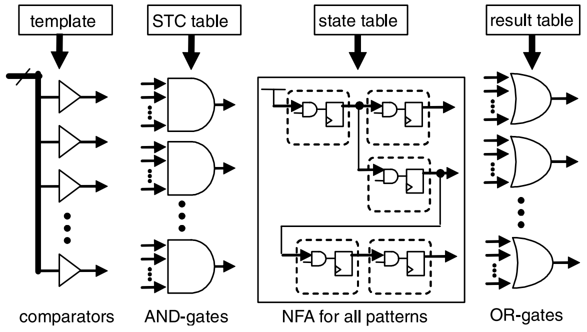

Fig. 8 Generating the string matching circuit: Step (3).

rule from the detection reports on the host PC. The debug information includes the header and payload properties needed to generate pseudo intrusions and attacks for testing the system. The generation log shows the processing report, for instance, an estimated quantity of the circuit, Snort analysis details, and an unsupported options report.

The IDS is composed primarily of the string matching circuit. Consequently, optimizing the string matching circuit is crucial in order to improve the performance and resource utilization of the system. For unsophisticated approaches, the resource requirement of the circuit increases markedly as the processing width expands [4]. On the other hand, our automation tool generates a lightweight scalable circuit using the NFA-based method proposed in [5]. The NFA-based method reduces the resource utilization of the circuit without reducing its performance by sharing redundant states and state transition conditions (STCs). Figure 5 shows an example of sharing duplicative states in an 8-bit processing NFA with four patterns: "he", "she", "his", and "hers". The states that have the same prefix characters can be shared. There are five different STCs in the right NFA, "h", "e", "r", "s", and "i", whereas nine STCs. In other words, four STCs are duplicative. For other processing widths, the redundant states and STCs can be shared in the same manner.

The NFA-based string matching circuit is generated from the matching pattern through the following three steps:

(1) Extract the duplicative STCs (see Fig. 6). All of the STCs are extracted by constructing the temporal NFA of each matching pattern. The STCs already registered in the table are removed. The NFA is discarded at end of Step (1).
(2) Generate a state tree of the NFA (see Fig. 7). The NFA tree for all matching rules is generated by employing the construction of a goto-function in the Aho-Corasick approach [15]. The state table and the result table are constructed. The state table includes the previous state and the STC for each state. The result table includes the respective states of matching results.
(3) Build the circuit from the STC table, the state table, and the result table (see Fig. 8). The circuit consists of comparators, AND-gates, an NFA, and OR-gates. Flip-flops are placed on each output signal to improve the maximum delay.

We reduced the resource requirement by over 50% compared with a previous study [4]. However, the proposed method generated large fan-out signals because the reduction was achieved by sharing redundant resources. The large fan-out signals consume limited interconnect resources on the FPGA. In particular, the large fan-out caused by sharing STC signals constituted a limiting factor of the implementation. Upon first implementation, we failed in routing due to an excessively high STC signal density. Therefore, we customized the newly developed tool such that the string matching circuit is divided into several units in order to reduce the fan-out density of the STC signals. The shared states are maintained by dividing the patterns according to these prefixes, because reducing the number of states is effective for decreasing the fan-out of the STC signals. Only the NFA refers to the STCs.

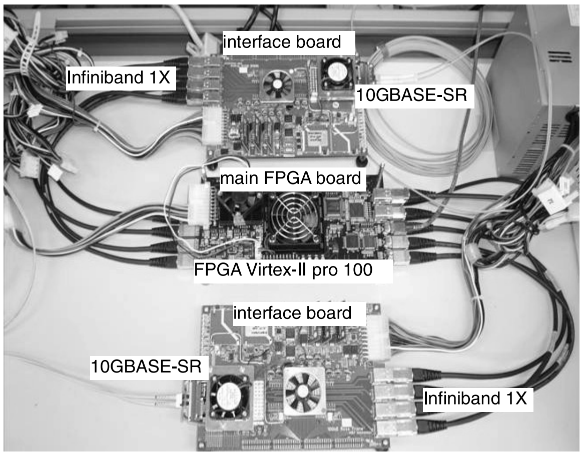

Fig. 9 Photograph of the IDS.

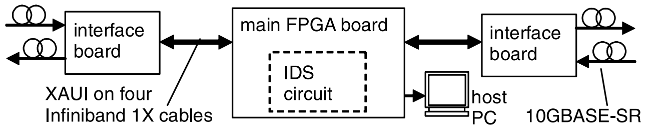

Fig. 10 Composition of the IDS.

3.3 Implement on the FPGA System

We implemented the IDS on a high-performance FPGA board system. The system consists of two types of FPGA boards, a main board, and an interface board. The main board equips a Virtex-II Pro 100 device [16], ten Infiniband 1X ports, two DDR-SDRAM SO-DIMM sockets, four individual DDR2-SRAMs, and four flexible PCB connectors. The interface board equips a Virtex-II Pro 7 device, four Infiniband 1X ports, and an XPAK socket with a 10GBASE-SR module. Figures 9 and 10 show photographs of the system and its composition, respectively. The main board is connected to two interface boards with four Infiniband 1X channels each. The throughput of each channel is 3.125 Gbps bi-directionally with an InfiniBand cable that has a performance margin that enable translation at over 2.5 Gbps. The packets are transferred through the four channels at 10 Gbps employing the XAUI format. We previously verified the channels and adjusted a pre-emphasis parameter of Rocket IOs. The main board is also connected to the host PC through IEEE 1394 in order to report intrusion detection.

The system notices the detection report at each 125 ms as the throughput and the detection frequency of each Snort rule. Figure 11 shows an example of the graphical report. The top graph shows the throughput of the input traffic that is divided into normal packets and attack packets. The middle graph shows the categories of intrusions and attacks.

We generated an IDS circuit that has 1,225 rules and processes at 10 Gbps from the Snort snapshot rule set on Feb 20, 2007. The number of rules was approximately twice that of Force10 IDS. The rules had "content" in their payload op-

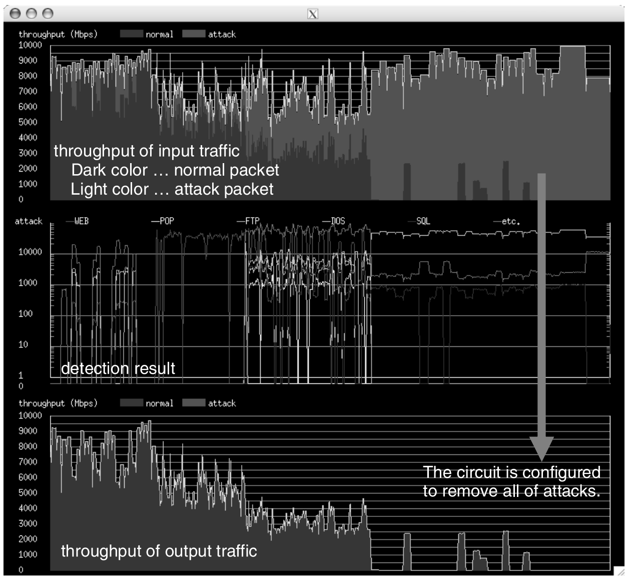

Fig. 11 Example of an intrusion detection report.

tions. The payload options contained 1,221 string patterns. The total length of the string patterns was 14,404 characters, and the average length was 11 characters. Table 1 presents the content of the subset rule. The rules are categorized based on alert messages. For example, the "EXPLOIT ssh CRC32 overflow NOOP" rule in the "EXPLOIT" category alerts of an attack to exploit vulnerable versions of the SSH daemon. Detail information of the rules is provided in the doc/signatures directory of the Snort rule set.

The string matching circuits were generated along six variations of division: non-division, 2, 3, 4, 5, and 6 units. The maximum operational frequency was configured as 156.25 MHz (64-bit, 10 Gbps). Xilinx ISE 8.2 sp3 was employed as a synthesis and routing tool. A timing-driven mapping and a middle effort level of routing were enabled. The entire IDS circuit was implemented on the main board. An interface board was used as a 10-Gigabit-Ethernet transmitter.

Table 2 shows the implementation results. The resource utilization is represented as the LUTs and FFs. The Slices show the area occupied by the LUTs and FFs. The routing failed in the case of non-division and two-unit design because of the excessively high density of the interconnecting signals. The three-unit design could not fit the timing constraint. The designs of four, five, and six units could be implemented successfully. As a consequence, string matching unit division is verified to be effective for simplifying the implementation, although the resource utilization increased with the number of units.

Although the number of Slices of the four-unit design was fewer than that of the three-unit design, the resource utilization increased. This was probably caused by small fan-out because it was easy to pack the LUTs and FFs into the Slices. Figure 12 shows the floor plan of the four-unit design. The design has 1,221 (= 309 + 320 + 283 + 309)

Table 1 Categories of the subset rule.

rule category	#rules
ATTACK-RESPONSES	16
BACKDOOR	457
CHAT	25
DDOS	27
DNS	17
DOS	5
EXPLOIT	57
FINGER	12
FTP	17
ICMP	25
IMAP	7
INFO	5
MISC	35
MULTIMEDIA	4
MYSQL	4
NETBIOS	11
ORACLE	10
OTHER-IDS	3
P2P	20
POLICY	27
POP2	2
POP3	12
PORN	21
RPC	36
RSERVICES	13
SCAN	11
SHELLCODE	21
SMTP	29
SNMP	9
SPYWARE-PUT	61
MS-SQL/SMB	28
MS-SQL	50
TELNET	18
TFTP	14
WEB-CGI	4
WEB-CLIENT	10
WEB-COLDFUSION	16
WEB-IIS	16
WEB-MISC	64
WEB-PHP	4
X11	2
total	1225

Table 2 Results of IDS circuit implementation.

#unit	routing	timing	Slices	LUTs	FFs
non-division	ng	ng	31865	58347	53196
2	ng	ng	33183	61108	56059
3	ok	ng	35191	62370	57653
4	ok	ok	34996	63663	59173
5	ok	ok	35978	64397	60170
6	ok	ok	36573	65642	61188

strings for 1,225 Snort rules. Most of the resources are occupied by the string matching circuit.

4. Evaluation

We evaluated the circuit of the four-string-matching-unit design with a network test system. The test system verifies filtering accuracy while measuring performance on 10 Gigabit Ethernet. Verification was performed by the following three steps (see Fig. 13):

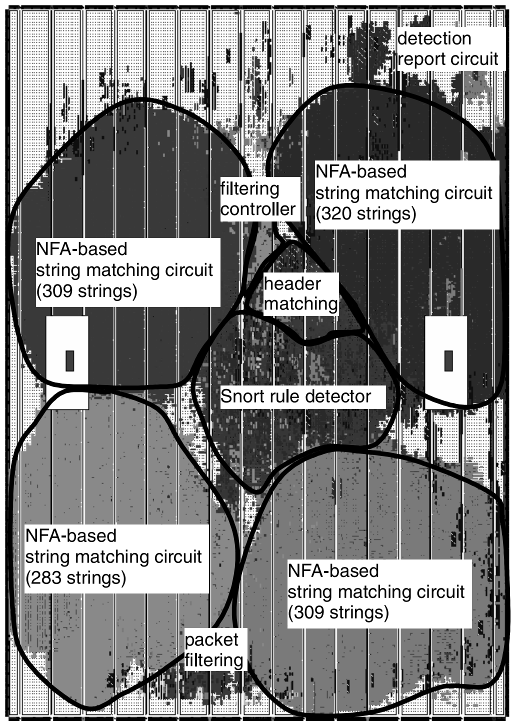
detection report circuit
NFA-based string matching circuit (320 strings)
filtering controller
NFA-based string matching circuit (309 strings)
header matching
Snort rule detector
NFA-based string matching circuit (283 strings)
NFA-based string matching circuit (309 strings)
packet filtering

Fig. 12 Floor plan of the IDS circuit.

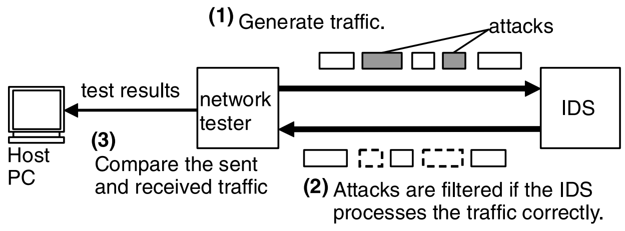
(1) Generate traffic.
attacks
test results
network tester
IDS
Host PC
(3) Compare the sent and received traffic
(2) Attacks are filtered if the IDS processes the traffic correctly.

Fig. 13 Testing the IDS.

(1) Generate traffic that includes attacks and intrusions from an intrusion model that presents the properties of the attacks and intrusions. For instance, the traffic of a DDoS attack has numerous packets that include specific ICMP payloads and different source addresses. The generated traffic is sent from the tester to the IDS.
(2) Let the IDS process the traffic. If the IDS operates correctly, all of the attacks and intrusions will be filtered. The tester receives the returned traffic.
(3) Compare the returned traffic with the sent traffic to verify the filtering accuracy. The tester verifies the traffic at wire speed. Finally, the result is reported to the host PC.

The attack packets were generated from the debug information of the circuit generation tool, namely, if the IDS worked correctly, 1,225 types of attacks were detected. The normal packets were ICMPs that had a specific length range of random payloads. The lengths of the normal packets were controlled within a specific range to change the throughput of the traffic. The normal packets were confirmed in ad-

Table 3 Power consumption of the IDS.

	voltage (V)	current (A)	power (W)
main board	3.3	7.61	25.113
	5.0	0.34	1.700
	12.0	0.09	1.080
summary			27.893
interface board (two boards)	3.3	1.24	4.092
	5.0	2.99	14.950
	12.0	0.18	2.160
summary			21.202
total			49.095

Table 4 Power consumption of a computer-based IDS.

	voltage (V)	current (A)	power (W)
ATX unit	3.3	4.96	16.37
	5.0	4.53	22.65
	12.0	5.99	71.88
summary			110.90

vance not to match the attack rules. The throughput was also controlled by gaps between the packets. We confirmed that the traffic was controlled dynamically in with respect to throughput, rate of the attacks, and types of attacks.

We verified the applicability of the proposed circuit using various pseudo traffic. As a result, the circuit filtered the attacks and intrusions accurately in all of the traffic. For example, either the sent traffic had 595,642 packets that included 93,086 attacks, or the received traffic from the IDS included 502,556 normal packets and no attacks. The number of received normal packets was identical to the number of sent packets, $595,642 - 93,086 = 502,556$. Consequently, the results indicate that the IDS removed the attacks accurately. In addition, the maximum throughput at which the circuit processes correctly without dropping packets was the wire speed of 10 Gigabit Ethernet.

We also evaluated the power consumption of the circuit. The FPGA system is powered by an ATX supply unit. We measured the power consumption through the ATX power cable using a digital clamp meter. Table 3 shows the power consumption of the main board and the two interface boards.

On the main board, the 3.3-V line was consumed mainly by the FPGA and the memory devices. Other lines were used by peripheral circuits and fans. On the interface boards, 5.0-V line was consumed by the 10GBASE-SR modules and the FPGAs. The 3.3-V and 12.0-V lines were used by IO pads on the FPGA and fans, respectively. The total power consumption of the system was 49.095 W, which is much less than the 110.90 W required the Xeon 3.06-GHz computer-based IDS for 1 Gigabit Ethernet (See Table 4).[†]

The main reason for the low power consumption is that most of the states are stable in the NFA-based string matching circuit. In other words, most of the duplicative signals that enter the hot state synchronously have been minimized by the proposed method. Since there are few signal switchings on FPGA, the dynamic power consumption is small. Furthermore, the states barely became hot in the 64-bit pro-

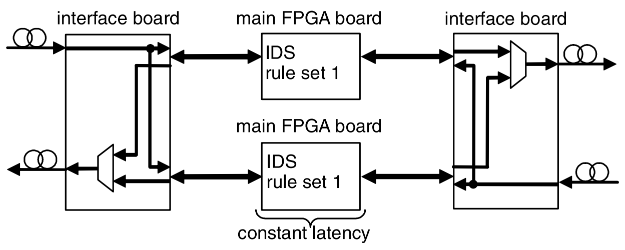

Fig. 14 IDS structure to support online-rule update.

cessing. For example, the probability of the STC string "ethernet" appearing in the 64-bit data bus is $(1/256)^8$.

5. Constructing a Practical System

In this section, we show two approaches to operating the IDS with the proposed circuit, online-rule update, and large rule set support.

5.1 Online-Rule Update

The system can be composed so as to support online update of the Snort rule with two main FPGA boards and extended interface boards. Figure 14 shows the system composition. The interface boards require eight Infiniband 1X ports. In other words, two XAUI ports are needed. The input traffic is duplicated on each main board, and the output traffic is selected from the data of both of the main boards. The interface boards switch the flow of the traffic synchronously without blocking the IDS processing. The output timing of the main boards is simultaneous because the latency of the circuit on the main board is constant. Figure 15 shows the procedure for online update of the Snort rule. The system updates the rule set by the following three steps:

(1) Reconfigure one of the main boards. The board is switched to be offline and is reconfigured to support a new rule. The reconfiguration will be performed within one second.
(2) Seek a switching point. When the reconfiguration is finished, the main boards process the same flow by each rule. The removed packets are different because of the different rules. Therefore, the switching point must be at an "idle" cycle in both traffics. The interface board finds the switching point.
(3) Switch the flow at the point. The flow to the other main board is cut. The board is reconfigured with the same rule set.

5.2 Support Large Rule Set

In the previous section, the implementation results indicate that the resource utilization for the 1,225 Snort rules was

[†]The PC-based IDS was measured in the same manner as the proposed IDS. The power consumption of Force10 IDS has not been reported.

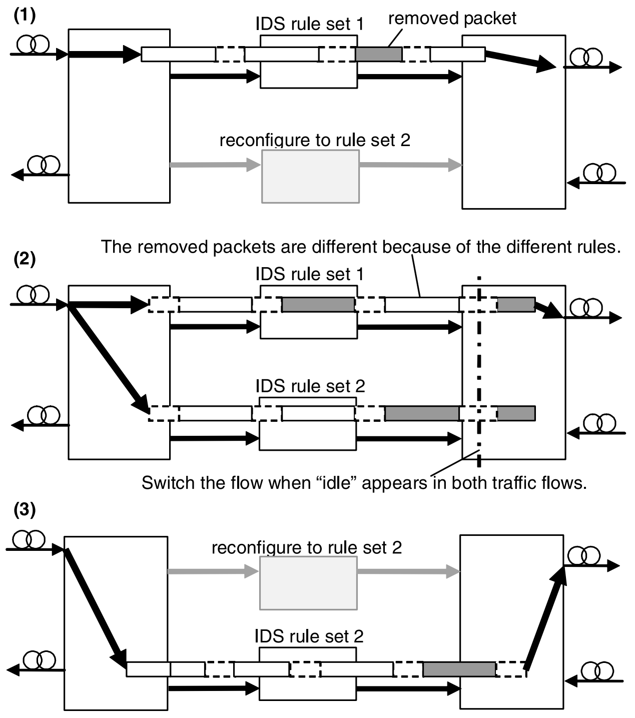
(1)
IDS rule set 1
removed packet
reconfigure to rule set 2
(2)
The removed packets are different because of the different rules.
IDS rule set 1
IDS rule set 2
Switch the flow when "idle" appears in both traffic flows.
(3)
reconfigure to rule set 2
IDS rule set 2

Fig. 15 Procedure for online-rule update in the IDS.

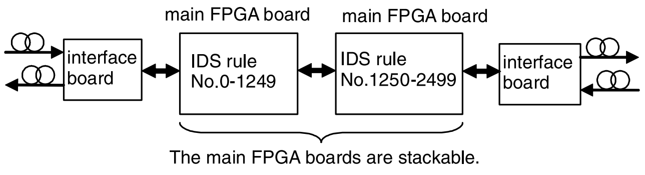

Fig. 16 Support large rule set.

34,996 Slices (79% of the device), 63,663 LUTs, and 59,173 flip-flops in the four-unit design. Consequently, it is difficult to support a much larger rule set with a single FPGA (Virtex-II pro 100). However, the proposed IDS architecture can support the stacking of multiple main boards. Figure 16 shows an example of stacking two boards, the 2,500 rules are divided into two rule sets. The latency and power consumption increase in proportion to the number of stacked boards. In the case of 10,000 rules, the system requires eight main boards and two interface boards. The main board consumes 27.893 W and has a latency of 105 μs, and the two interface boards consume 21.202 W. Therefore, the power consumption is $27.893 \times 8 + 21.202 = 244.346$ (W), and the latency is $105 \times 8 = 840$ (μs).

6. Future Work

In the present paper, we described a practical IDS for 10 Gigabit Ethernet with 1,225 Snort rules. In the future, we intend to investigate rule support enhancement, packet preprocessing, and verification on a practical network.

The proposed system supports the subset rules that have "content" in their payload options. However, there are other options. We intend to support these options by improving the circuit generation tool so as to handle regular expressions and by developing a sequencer to handle non-pattern-matching rules.

In addition, packet preprocessing will be incorporated into the proposed system in order to reassemble fragmented frames. Frame reassembly will enable the detection of matching patterns that stride across the fragmented frames. Hardware-based preprocessing was investigated in [17].

Finally, we intend to evaluate the utility of the proposed system and determine the number of rules that is required for practical use. Although there are 6,880 rules in the Snort rule set, a subset is used in the practical IDS. For example, if there is no Web server at a site, its related rules are not needed.

7. Conclusion

In the present paper, we proposed an approach by which to construct an FPGA-based IDS from the Snort rule set and demonstrated the implementation of the proposed system. In addition, we developed an automation tool that generates the circuit from the rule set with the NFA-based algorithm. The system is able to update the rules corresponding to new intrusions and attacks employing the tool.

We implemented the circuit that has 1,225 rules on the single FPGA for 10 Gigabit Ethernet. The tool was configured to divide the string patterns. Our implementation revealed that the division of the string matching circuit was effective in simplifying the routing. The IDS was also verified to process 1,225 rules correctly at the wire speed of 10 Gigabit Ethernet and to have a low power consumption of 49.095 W.

References

[1] M. Roesch, "Snort — Lightweight intrusion detection for networks," 13th Systems Administration Conference, LISA '99, pp.229–238, 1999.
[2] R. Sidhu and V.K. Prasanna, "Fast regular expression matching using fpgas," Proc. IEEE FCCM 2001, pp.227–238, 2001.
[3] B.L. Hutchings and D.C.R. Franklin, "Assisting network intrusion detection with reconfigurable hardware," FCCM2002, pp.111–120, 2002.
[4] C.R. Clark and D.E. Schimmel, "Scalable pattern matching for high speed networks," FCCM2004, pp.249–257, 2004.
[5] T. Katashita, A. Maeda, K. Toda, and Y. Yamaguchi, "A method of generating highly efficient string matching circuit for intrusion detection," FPL2006, pp.799–802, 2006.
[6] Z.K. Baker and V.K. Prasanna, "A methodology for synthesis of efficient intrusion detection systems on FPGAs," FCCM2004, pp.135–144, 2004.
[7] Y. Sugawara, M. Inaba, and K. Hiraki, "Over 10 Gbps string matching mechanism for multi-stream packet scanning systems," FPL2004, pp.225–234, 2004.
[8] I. Sourdis and D. Pnevmatikatos, "Pre-decoded cams for efficient and high-speed nids pattern matching," FCCM2004, pp.258–267, Napa, CA, 2004.
[9] J. Singaraju, L. Bu, and J.A. Chandy, "A signature match processor architecture for network intrusion detection," FCCM2005, pp.235–242, Napa, CA, 2005.

[10] Z.K. Baker and V.K. Prasanna, "Time and area efficient pattern matching on fpgas," FPGA'04, pp.223–232, 2004.
[11] G. Papadopoulos and D. Pnevmatikatos, "Hashing + memory = low cost, exact pattern matching," Field Programmable Logic and Applications, 2005, pp.39–44, 2005.
[12] P. Katta, M. Nourani, and R. Panigrahy, "String matching engine using parallel hashing," PDCS 2006, pp.478–483, 2006.
[13] "Force 10 networks p-series, motel p10 intrusion prevention system performance evaluation," Tech. Rep. 206126, THE TOLLY GROUP, 2006.
[14] T. Katashita, A. Maeda, K. Toda, and Y. Yamaguchi, "Highly efficient string matching circuit for ids with FPGA," FCCM2006, pp.285–286, 2006.
[15] A.V. Aho and M.J. Corasick, "Efficient string matching: An aid to biblographic search," ACM Commun., vol.18, no.6, pp.333–340, 1975.
[16] Xilinx Inc, DS083, Virtex-II Pro and Virtex-II ProX Platform FPGAs: Complete Data Sheet.
[17] D.V. Schuehler and J. Lockwood, "Tcp-splitter: A tcp/ip flow monitor in reconfigurable hardware," High Performance Interconnects, 2002, pp.127–131, 2002.

Toshihiro Katashita received his B.E. in 1997, his M.E. in 1999, and his Ph.D (Eng.) degree in 2006 from University of Tsukuba. He is a researcher at AIST. His research interests are FPGA based systems, circuit design, and network security.

Yoshinori Yamaguchi received his B.S. degree in electrical engineering from the University of Tokyo in 1972. He received his PhD degree from The University of Tokyo in 1993. He is a professor at the Graduate School of Systems and Information Engineering at University of Tsukuba. Before coming to University of Tsukuba, he worked at Electro Technical Laboratory from 1972 to 1999. His research interests are areas of computer architecture, parallel computer systems, computer networking and real-time systems. He has engaged in investigations of high-level language machines and parallel computer projects EM-3, EM-4, EM-X at Electro Technical Laboratory. He is currently interested in FPGA-based systems and high-speed network systems. He is a member of the IEEE Computer Society and the Information Processing Society of Japan. He received the IPSJ best paper award in 1991 and the Ichimura Award in 1995.

Atusi Maeda received his B.E. in 1986, his M.E. in 1988 and his Ph.D.(Eng.) degree in 1997 all in mathematics from Keio University. He was a research associate in University of Electro-Communications from 1997 to 1999. He is currently an Associate Professor at Graduate School of Systems and Information Engineering, University of Tsukuba. His research interests are programming language implementation, system programming, and garbage collection. He is a member of ACM, IPSJ, and JSSST.

Kenji Toda received the M.S. degree from the Keio University, Japan in 1982. He is the leader of Real-Time Embedded System Semi-Group in National Institute of Advanced Industrial Science and Technology (AIST). His research interests are real-time computing, embedded systems, and network applications. He is a member of IPSJ.