FPGA-based Real-time Abnormal Packet Detector for Critical Industrial Network

Jiwoong Kang
Dept. Info. and Comm.
Inha University
Incheon, Korea
jwkang@emcl.org

Taein Kim
Dept. Info. and Comm.
Inha University
Incheon, Korea
tikim@emcl.org

Jaehyun Park
Dept. Info. and Comm.
Inha University
Incheon, Korea
jhyun@inha.ac.kr

***Abstract*—As the information technology plays an important role in the smart factories, Ethernet-based industrial network has rapidly replaced the traditional field buses. To maintain this critical network secure, it is important to develop the real-time network intrusion detection system (NIDS). The widely used NIDS was developed for the general Internet environment where the average throughput to protect attacks from the large number of unknown network nodes is more important than the real-time detection capability. However, in the critical industrial network, the real-time protection is more important than the average throughput. In this paper, a FPGA-based abnormal Ethernet packet detector is proposed. Since it is designed for the closed industry network, packet detection is based on the whitelist that consists of the allowed network address and protocol numbers. The prototype system has been implemented using the Xilinx Zynq-7030 SoC running at 250MHz. The network header of the Ethernet packet is compared to the 256 whitelist ruleset within 0.032μsec, which means that the malicious packets from the abnormal network nodes are filtered out even before the whole packets arrives. This real-time packet filtering feature is useful in protecting highly secure network systems like the critical industrial control systems.**

***Index Terms*—Ethernet packet detector; network intrusion detection system; Modbus; FPGA**

I. Introduction

As the information technology plays an important role in the smart factories, Ethernet-based industrial network rapidly replaces the traditional field buses. Ethernet-based industrial network is, however, easily opened to the public Internet and has inherent security risk [1]–[3]. One of the efficient ways to protect a network node from the unknown or suspicious network activities is to adopt a network intrusion detection system (NIDS) that analyzes the incoming network packets and warns the users upon detection of a malicious network packet or a suspicious network access from unknown network nodes [4]. NIDS distinguishes the malicious packets among the received network packets based on predefined rules that define the communication pattern of the malicious packets. Since single cyber attack toward the industrial control network can cause a severe damage to the smart factory, the importance of NIDS is growing.

This work was supported by the Korea Institute of Energy Technology Evaluation and Planning (KETEP) and the Ministry of Trade, Industry & Energy (MOTIE) of the Republic of Korea (No. 20171510102110).

The most widely used NIDS software include the *Snort* and *Suricata*, that are open source-based software running on the various operating systems [5], [6]. These NIDS software usually run on the target network nodes that should be protected or on a stand alone server to protect the network subnets. While these software-based NIDS are flexible and easily reconfigurable, they still have shortcomings: first, since the incoming network packets are analyzed by software, it takes relatively long time to detect an abnormal packet and suspicious cyber attacks. This means that a real-time network protection is hardly implemented. Second, a server or system running a NIDS software consumes a large amount of resource that results in the packet loss even in a low-bandwidth network environment [7]–[10].

In order to overcome the problem of the software-based NIDS, a hardware-based NIDS using a FPGA has been proposed. Das *et al.* proposed a FPGA-based system that detects the exceptions by implementing the Principal Component Analysis (PCA) [11]. The proposed system provided the anomaly detection capability rather than the signature detection method provided by most of the IDS software. The Feature Extraction Module (FEM) was designed to differentiate the network and the PCA module was designed to detect the anomalies. The use of the PCA reduces the size of the data and the time during analysis. Rahmatian *et al.* proposed a hardware-based NIDS to determine whether a malicious program was running or not [12]. When a new program tries to run on an embedded system, the finite state machine (FSM) of the sequence recognizer examines the integrity of the execution of system calls on a particular operating system. The system called sequence is sent to the FPGA to verify whether the programs are executed in order. Hutchings *et al.* proposed a method for improving the performance of string matching using an FPGA [13]. They proposed a FPGA-based regular expression module generator that uses the Java Hardware Description Language (JHDL) to analyze various attributes of the Ethernet packets. Regular expressions are generated based on the strings extracted from the *Snort* rules and are used to create the FPGA circuits. The implemented hardware compares all the incoming strings using the Non-Deterministic Finite Automata (NFA) and significantly reduces the CPU workload compare to the software-based IDS. Kim

and Park proposed FPGA-based pattern matching architecture that can be used for the hardware-based NIDS [14]. Lunteren presented a new hardware-based scheme for pattern matching called BFPM that improves the performance of NIDS [15]. The BFSM-based pattern matching (BFPM) is based on a new programmable state machine technology, the FSM based on Bayesian network (BFSM). The BFPM scheme showed a high deterministic performance regardless of the input pattern characteristics. In addition, it showed other features such as high storage efficiency and dynamic updates feature.

Since these previous FPGA-based NIDS were designed for the general Internet network nodes, their performance were analyzed to the average throughput rather than real-time characteristics. In the general Internet applications, a certain amount of latency in detecting the abnormal packets are allowed because most of the applications does not have a strict real-time constraint. However, in the industrial network such as the smart manufacturing factory, the nuclear power plant, and the traffic control systems, there is a very tight real-time constraint within which cyber attack should be detected. The other characteristics of the critical industry network is closed network that limits the number of accessible network nodes. Hence, with considering these closed industrial network, whitelist-based NIDS can be efficiently implemented.

This paper proposes a FPGA-based packet detector or the whitelist-based NIDS within a closed network that is widely adopted in the critical network system including nuclear power plant. The overall system implemented is described in Section 2. In Section 3, the implementation results are explained, and conclusions are in Section 4.

II. Proposed Architecture

A. System Structure

Figure 1 shows the internal structure of the proposed system. There are two blocks in the SoC; a processor system (PS) slice and a programmable logic (PL) slice. Even though the Xilinx's Zynq 7000 series SoC was used in the paper, any type of SoC commercially available can be used. The hardware-based packet detector and dual-port BRAM for the whitelist are implemented in a PL slice. The standard AXI is used between the PS and PL through which it is possible for the user to read and write the ruleset storage in the PL's BRAM area. The detail structure of the packet detector implemented on a PL slice is shown in Figure 2. It consists of three blocks: the packet parser (PP), the comparator, and ruleset store. The packet parser breaks the incoming Ethernet packets into 7-tuple, source and destination MAC address, IP address, port number, and Modbus function code. The comparator compares the parsed packets with the whitelist ruleset. The ruleset store contains the whitelist generated by the application software running in the processor system(PS). The details of each block are described in the following subsections.

B. Ethernet Packet Analyzer

To parse the incoming Ethernet packets in real-time, a Media Access Controller (MAC) with a RGMII interface was

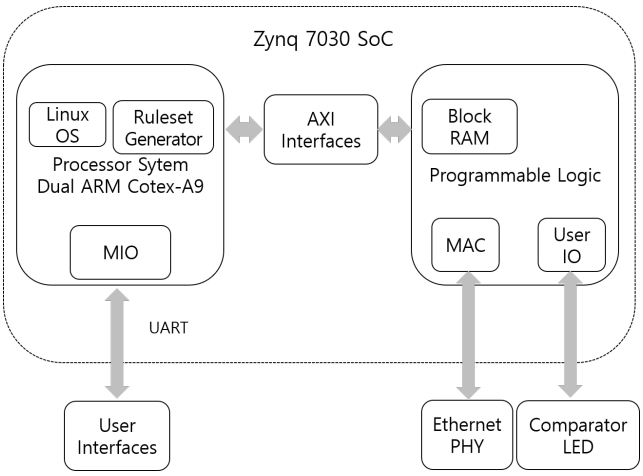


Fig. 1. System block diagram

implemented in the PL slice. If the standard commercial MAC was used, the packet parsing and the comparison can be started after whole Ethernet packet has been received. However, in this paper, to detect the abnormal packet in real-time before the whole Ethernet packet delivered, MAC has been implemented in the PL slice. The implemented MAC receives data from a PHY chip through the RGMII interface as shown in Figure 2. The received Ethernet packet is parsed into the 7-tuples in real-time. When each 7-tuple data in the Ethernet packet header is received, it is sent to the comparator to be compared with the whitelist stored in the ruleset store. It means packet parsing and comparison can be done while the remaining network payloads are receiving. Such real-time process is almost impossible in the software-based NIDS because the packet parsing can be done after the whole Ethernet packet has been received. For example, when the 48-th bit of the Ethernet streams, the last bit of the destination MAC address, is received, the destination MAC address is transmitted to the comparator. The data transferred to the comparator is held until the next Ethernet packet arrives and a new destination MAC address is received. In the MAC buffer, the data is continuously stored until the data of the source MAC address is received, and when the source MAC address data is received, the data is transmitted to the comparator. It repeats until the data from the destination MAC address to the source port number is received. MAC is designed to compare with the whitelist ruleset before all the Ethernet packets arrives by comparing it to the comparator every time the data for the tuple is received. Because the 100 Mbps Ethernet is used, the PHY transmitted 4 bits of data with a clock of 25MHz.

C. Ruleset

Since the proposed system aims the real-time packet detection used within the critical industrial network, the prototype system was built for the Modbus that is one of the most widely used industrial network [16]. To analyze the Modbus packets, the proposed system uses MAC addresses and Modbus function code(F-code) in addition to a normal *Snort*-like

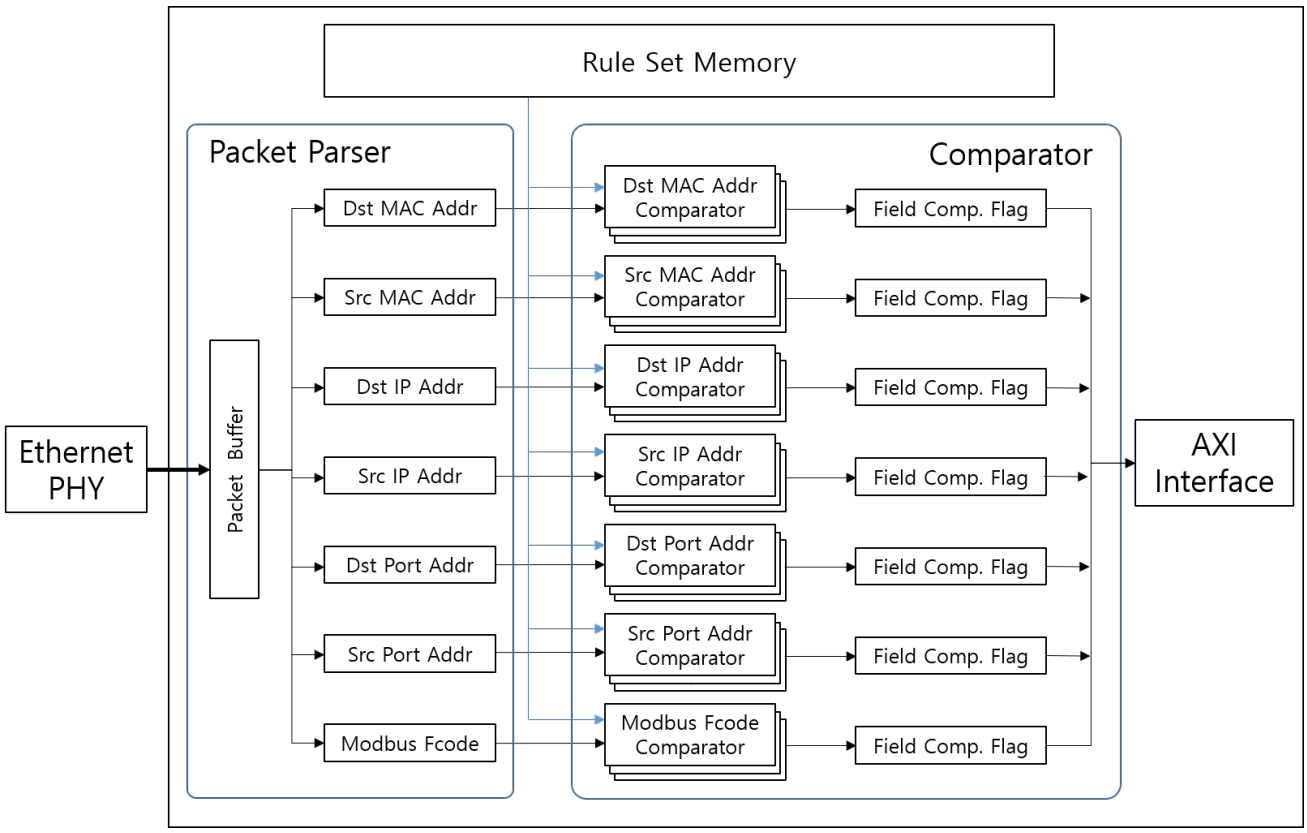


Fig. 2. FPGA block diagram

4-tuple rule that consists of IP address and port number of the source and destination nodes. The MAC address is very useful to manage the whitelist within a local area network like an industrial control systems. With this MAC address, more flexible and accurate packet filtering is possible. First, a user may not know the IP address of the system when the IP address is dynamically assigned by a private router or when the IP address is changed without the user's knowledge. By utilizing the MAC address, the rule can be defined even if the user does not know the IP address. Second, MAC addresses help enhancing the packet detection in the Ethernet packet header. The analysis time can be reduced by reducing the detection case of the Ethernet packet payload.

The FPGA should be reconfigured when the ruleset is modified. To overcome this inconvenience, a dual port BRAM for storing the ruleset is configured in the PL slice. Since the BRAM is implemented as a dual port memory, it can be read and written by both the PS and PL. When a user enters a rule through the interface provided by the PS, the rule is stored in the PL's BRAM via AXI.

Figure 3 shows the structure of the ruleset stored in the BRAM. To store the rules for a 7-tuple, the data width of the BRAM was set to 64 bits. The first line stores the 48 bit destination MAC address and the 16-bit destination port number. The second row stores the source IP address and the destination IP address. The third line stores the MAC address and the port number of the source node. In addition to these MAC, IP, and port number, maximum 64 bit of payload-specific rule such as Modbus F-code, can be stored in the last line.

```
NodeAddress ::= SEQUENCE {
   MACAddress    OCTET STRING (SIZE(6))
   IPAddress     OCTET STRING (SIZE(4))
   PortNumber    OCTET STRING (SIZE(2))
}

RulSet ::= SEQUENCE {
   DstNode     NodeAddress
   SrcNode     NodeAddress
   Fcode       OCTET STRING(SIZE(1))
}
```

Fig. 3. Stored ruleset in BRAM

D. Real-time Packet Detector

Since the whitelist rule is composed of 7-tuple, a single comparator block consists of 7-tuple parallel comparators to compare a whitelist rule at once. Figure 4 shows the structure of the single tuple comparator unit. As shown in the figure, the single tuple comparator unit has eight internal registers. During the system initialization, eight whitelist rules are copied from a

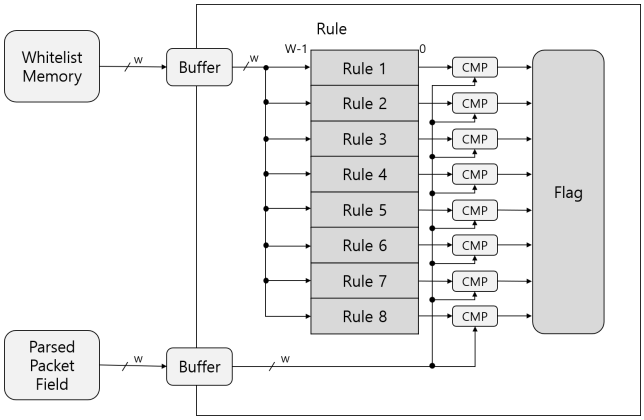


Fig. 4. Comparator block diagram for MAC address rule

block memory into the registers. The comparator then receives the parsed data from the MAC and compares it to the whitelist ruleset stored in the registers. As shown in 4, only 48 bits data out of 64 bit data of BRAM is necessary for the MAC address comparator. The MAC addresses stored in the eight registers are composed of different ruleset. The comparator receives the parsed data from the MAC and compares it to whitelist rules stored in registers. XOR operator was used for comparison. If a register has a whitelist ruleset, the result of XOR operator is 0 and reflected to the flag. Because each comparator has eight registers to store eight different whitelist, it can compare eight whitelist rules simultaneously. If there are more than eight whitelist rules, multiple comparator blocks can be used in parallel. In this configuration, up to 256 rulesets can be processed at the same time. The limitation on the number of comparator blocks is only the size of the PL slice of the SoC. The results from each comparator unit were OR'ed and stored in the flag register that represents whether the received packet is listed in the whitelist ruleset or not.

III. Implementation and Experiment Results

To prove the concept of the proposed architecture, a prototype system has been implemented using the Xilinx Zynq-7030 SoC running at 250MHz. An Ethernet packet detector for comparison and detection of 7-tuples was configured using the PL slice of the SoC. Total 256 rulesets consisting of the 7-tuple was stored in the BRAM as an experimental whitelist.

When the Ethernet packet was received, the `RGMII_CTL` signal became 1 indicating a new Ethernet packet has been arrived. Even before the whole data packet arrives, upon receiving each tuple, the received packet is parsed immediately and sent to the comparator for comparison. The parsed Ethernet header is compared to the 7-tuple ruleset in the whitelist. The 7-tuple could be compared to the whitelist ruleset stored without a delay time while the Ethernet packets were received. There are parallel comparators to compare the parsed 7-tuples with the multiple whitelist in parallel. A single whitelist ruleset is placed in each register in the same order. If the result of `OR`'ing of the comparator 0, it means the input Ethernet data is already included in the whitelist ruleset. If the output value is 1, it means the received packet is not yet listed in the whitelist.

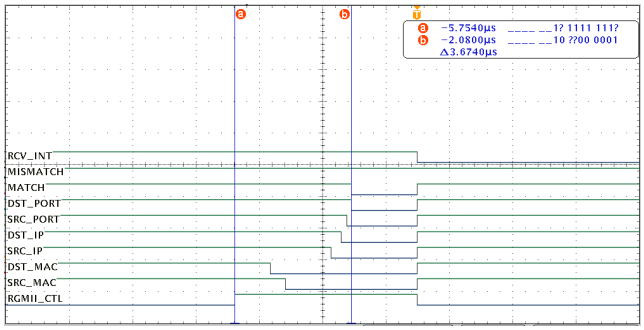
Figure 5 shows the timing chart when a normal packet of which header is listed in the whitelist arrives. `MATCH` signal means that the incoming packet is listed in the whitelist. The RGMII interface was used between the PHY and MAC, which enables 4 bit data can be delivered at each RGMII clock cycle. Since the Ethernet packet header is 384 bits long and the incoming packet is parsed and compared in real-time upon receiving, the elapsed time to determine whether it is matched or not is only 3.67μs. If the arrived packet is confirmed in the whitelist, the `RCV_INT` signal to the CPU (PS slice in Zynq SoC) indicates a normal Ethernet packet is received. Figure 5-(a) and Figure 5-(b) shows the timing chart of the shortest message and MTU(Maximum transfer unit) payload, respectively.

Figure 6 shows the timing chart when an abnormal packet of which header is not listed in the whitelist arrives. Figure 6-(a) and Figure 6-(b) shows the destination MAC and source port number is not listed in the whitelist, respectively. Considering the destination MAC address is 96 bits, it took only 12 RGMII clock, 0.96μs, to check the destination MAC address. In the experiment, it took 1.11μs. Even in the second case, destination port number, it took only 3.69μs to determine it is not in the list, that is 2.1μs before the whole packet of the shortest message(ICMP) received. When the incoming packet is determined as an abnormal packet, the `RCV_INT` signal remains false in order that CPU may ignore the incoming packet. This means packet filtering is processed in real-time without any software burden.

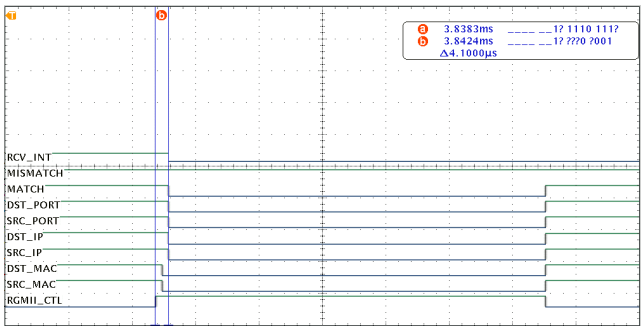
The internal clock of the PL operates at 250MHz. Since there are 32 comparators with the eight rules, 8 system clocks, 0.032μsec, are required to compare the 256 rules. This comparison makes it possible to complete the comparison with the whitelist ruleset for 7-tuple before the whole Ethernet packets are received. The implementation result shows that 3.14% of LUTs', 2.15% of Flip-Flops', and 2.83% of block RAM of the Zynq 7030 were used.

IV. Conclusion

In this paper, a FPGA-based Ethernet packet detector for a critical industrial network was proposed. The proposed system consists of a packet parser, a comparator, and an intrusion detection ruleset storage that are implemented in a programmable logic (PL) slice of the SoC. Since it is designed a closed industrial network, packet detection is based on the whitelist that contains allowed network node and protocol pairs. The whitelist ruleset used in this paper consists of 7-tuples; MAC address, IP address, TCP/UDP port number of the source and destination network nodes, and Modbus F-code that is an extended version of widely used *Snort* ruleset. The prototype system was implemented using the Xilinx Zynq-7030 SoC running at 250MHz. The implemented prototype used 3.14% of LUTs', 2.15% of Flip-Flops', and 2.83% of block RAM in the Zynq-7030. The Ethernet packet headers are compared to the 256 whitelist ruleset within 0.032μs on the

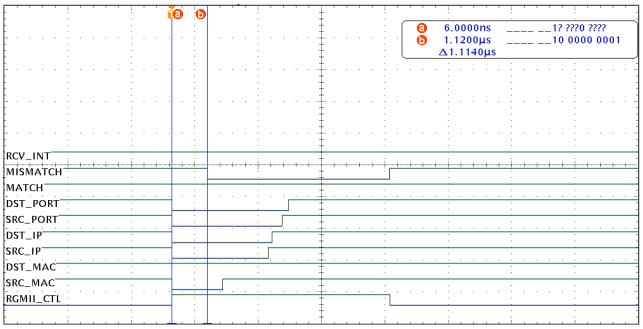


(a) Short message(ICMP)

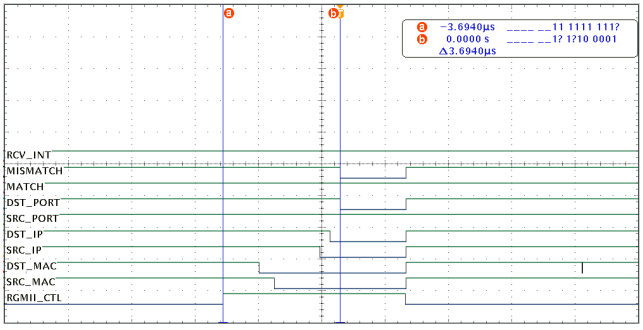


(b) Long message(FTP)

Fig. 5. Timing of the normal packet case



(a) Destination MAC address mismatch



(b) Source Port number mismatch

Fig. 6. Timing of abnormal packet case

implemented prototype system. This means that the malicious packets from the abnormal network nodes can be detected even before the whole packet arrives. This real-time detection performance can be achieved without consuming much CPU resource. It also showed the very accurate packet filtering capability without even single packet loss that might found in the software-based IDS.

REFERENCES

[1] M. Cheminod, L. Durante, and A. Valenzano, "Review of security issues in industrial networks," *IEEE Trans. Ind. Informat.*, vol. 9, no. 1, pp. 277–293, Feb 2013.

[2] D. Dzung, M. Naedele, T. P. Von Hoff, and M. Crevatin, "Security for industrial communication systems," *Proc. IEEE*, vol. 93, no. 6, pp. 1152–1177, June 2005.

[3] C. Queiroz, A. Mahmood, J. Hu, Z. Tari, and X. Yu, "Building a SCADA security testbed," in *2009 Third International Conference on Network and System Security*, Oct 2009, pp. 357–364.

[4] T. Cruz, L. Rosa, J. Proença, L. Maglaras, M. Aubigny, L. Lev, J. Jiang, and P. Simões, "A cybersecurity detection framework for supervisory control and data acquisition systems," *IEEE Trans. Ind. Informat.*, vol. 12, no. 6, pp. 2236–2246, Dec 2016.

[5] Cisco, "Snort user manual," http://www.snort.org, April 2017, [Online; accessed July 1, 2017].

[6] OISF, "Suricata user guide," http://www.suricata-ids.org, 2016, [Online; accessed July 1, 2017].

[7] H. Chen, Y. Chen, and D. H. Summerville, "A survey on the application of fpgas for network infrastructure security," *IEEE Commun. Surveys Tuts.*, vol. 13, no. 4, pp. 541–561, April 2011.

[8] R. Abdulhammed, M. Faezipour, and K. M. Elleithy, "Network intrusion detection using hardware techniques: A review," in *2016 IEEE Long Island Systems, Applications and Technology Conference (LISAT)*, April 2016, pp. 1–7.

[9] A. Mitra, W. A. Najjar, and L. N. Bhuyan, "Compiling PCRE to FPGA for accelerating SNORT IDS," in *ACM/IEEE Symposium on Architecture for Networking and Communications Systems*, Dec 2007, pp. 127–136.

[10] N. Stakhanova and A. A. Ghorbani, "Managing intrusion detection rule sets," in *Proceedings of the Third European Workshop on System Security*, ser. EUROSEC '10, 2010, pp. 29–35.

[11] A. Das, D. Nguyen, J. Zambreno, G. Memik, and A. Choudhary, "An FPGA-based network intrusion detection architecture," *IEEE Trans. Inf. Forensics Security*, vol. 3, no. 1, pp. 118–132, March 2008.

[12] M. Rahmatian, H. Kooti, I. G. Harris, and E. Bozorgzadeh, "Hardware-assisted detection of malicious software in embedded systems," *IEEE Embedded Syst. Lett.*, vol. 4, no. 4, pp. 94–97, Dec 2012.

[13] B. L. Hutchings, R. Franklin, and D. Carver, "Assisting network intrusion detection with reconfigurable hardware," in *Proceedings of 10th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, 2002, pp. 111–120.

[14] J. Kim and J. Park, "FPGA-based memory efficient shift-and algorithm for regular expression matching," in *Proceedings of 14th International Symposium on Applied Reconfigurable Computing. Architectures, Tools, and Applications*, May 2018, pp. 132–141.

[15] J. van Lunteren, "High-performance pattern-matching for intrusion detection," in *Proceedings IEEE INFOCOM 2006. 25TH IEEE International Conference on Computer Communications*, April 2006, pp. 1–13.

[16] N. Goldenberg and A. Wool, "Accurate modeling of Modbus/TCP for intrusion detection in scada systems," *International Journal of Critical Infrastructure Protection*, vol. 6, no. 2, pp. 63 – 75, 2013.