rrBox: A Remote Dynamically Reconfigurable Middlebox for Network Protection

Tze Hon Tan*, Chia Yee Ooi†, M. N. Marsono*
*Faculty of Electrical Engineering, Universiti Teknologi Malaysia, 81310 Skudai, Johor, Malaysia
†Malaysia-Japan International Institute of Technology, Universiti Teknologi Malaysia Kuala Lumpur, 54100 KL, Malaysia
Email: thtan5@live.utm.my, ooichiayee@ic.utm.my, nadzir@fke.utm.my

***Abstract*—This paper presents a remote dynamically reconfigurable middlebox for network protection by using NetFPGA 10G development board. The packet forwading and other network processing components in this middlebox can be updated remotely through the 1Gbps Ethernet connection without assistance from a host computer. Functional update is important to patch design flaws and bugs, to optimize design performance, and to cope with the changes on the functions of execution unit. In addition, this work demonstrates the use of the developed plaform for network protection. The proposed architecture uses a customized reconfiguration controller and the Internal Configuration Access Port available in the reconfigurable device. Based on the experimental result, the implemented middlebox achieved roughly 352Mbps reconfiguration throughput, which is important for mass updating the distributed middleboxes and decrease the devices down time during the updates.**

***Index Terms*—Remote dynamic reconfiguration, Partial reconfiguration, NetFPGA 10G, Middlebox, Network Protection**

I. INTRODUCTION

The inefficiency of the von Neumann machine paradigm leads to the integration of both von Neumann CPU (instruction driven) and non-von Neumann accelerators (data driven) implementation [1]. Systems-on-Chip (SoC) becomes a common implementation solution but it lacks the flexibility to cope with the rapid changes of functionality requirement. Thus, the focus has shifted toward replacing hardwired accelerators with reconfigurable devices, which offer better flexibility [2] that leads to the emergence of reconfigurable computing.

In reconfigurable computing, dynamic reconfiguration is a vital feature, which enables updates in the field, improves the area utilization and allows defect compensation [3]. Dynamic reconfiguration provides a solution to update a system so that it can adapt to changes in both functional and performance requirements. The applications that can possibly utilize this feature usually require processing power as much as hardware solutions, costly to apply update manually (that requires remote update) and continuously operating in the system.

Most network applications especially on middleboxes are required to operate in high throughput and are distributed. Middleboxes are required to remain active and operate continuously so that their connectivity with other end nodes are maintained and their functionality can be updated remotely. The cost to update such system is very high when manual reconfiguration is required and the device has to be shut down. This problem can be solved by utilizing the dynamic reconfiguration feature found in the reconfigurable device. However, the utilization of dynamic reconfiguration feature is not straightforward and requires a proper methodology in the design process. Therefore, a good framework that is able to efficiently perform dynamic reconfiguration on reconfigurable devices is required for network middlebox applications.

In this work, a remote dynamically reconfigurable middlebox architecture for network protection is proposed. The proposed architecture is implemented and tested experimentally using the NetFPGA 10G development board. The proposed platform aims to enable remote dynamic reconfiguration in NetFPGA 10G for functional updates purposes. The developed platform can be updated remotely through the 1Gbps Ethernet connection at run-time. In addition, network protection applications are developed to demonstrate the reconfiguration on the reconfigurable NetFPGA based middleboxes. In order to achieve a single-chip solution, the proposed architecture utilizes a customized reconfiguration controller together with Internal Configuration Access Port (ICAP) without relying on General Purpose Processor or a host computer to handle remote dynamic reconfiguration processes.

II. RELATED WORK

Sato and Fukase [4] developed a host-based IDS using reconfigurable hardware. The authors compared a software IDS and hardware IDS. The authors preferred hardware implementation mainly because of its high performance. Li et al. [5] implemented IDS using reconfigurable hardware to achieve higher processing speed and efficiency. This work uses Content-addressable Memory (CAM) to implement the matching unit and the performance has been significantly improved using the reconfigurable hardware. Distributed IDS has been proposed by Tummala et al. [6] in high-speed networks. In this work, the reconfigurable hardware offers higher degree of parallelism and flexibility for different data widths. The authors supported the fact that FPGA devices have advantages such as dynamic reconfiguration, low development cost, high speed, high scalability and short time to market.

Naous et al. [7] developed the NetFPGA development board to enable fast prototyping of network equipment. Antichi et al. [8] presented the path to migrate existing 1G design into the NetFPGA 10G development board. The framework of NetFPGA is versatile and an official web repository is provided to store the open source projects. By using the

978-1-4799-4152-0/14 $31.00 © 2014 IEEE
DOI 10.1109/CANDAR.2014.26

NetFPGA, Yin et al. [9] demonstrated customizable virtual network by using partial reconfiguration. In this work, the dynamic virtual network allocation relies on the dynamic reconfiguration feature in the FPGA. This work used the JTAG interface to load the partial bitstream dynamically into the FPGA device. Based on [10], loading partial bitstream with JTAG is less efficient compared when using ICAP. Similarly, Pontarelli et al. [11] used NetFPGA to develop a FPGA-based Network Intrusion Detection System (NIDS). In order to improve the utilization of logic resources, the authors exploit the dynamic reconfiguration feature in FPGA. Additionally, the implemented system is capable to work at wire speed. However, this work also uses JTAG that is inefficient [10]. Zhang et al. [12] designed a security system using NetFPGA development board and Virtex5 device. In the work in [12], the Virtex5 device is used to analyze captured malicious packets and forward the results to NetFPGA for updates. However, the remote reconfiguration on the NetFPGA requires control from the host PC for bitstream transmission and translation.

III. PLATFORM IMPLEMENTATION

The NetFPGA 10G development board is chosen as the implementation platform because the NetFPGA 10G development board comes with a Virtex 5 reconfigurable device that supports dynamic reconfiguration and SFP+ cages that support gigabit Ethernet module. Essentially, NetFPGA 10G development board is developed for the purpose of network applications prototyping. Additionally, NetFPGA 10G framework shown in the Figure 1 provides the flexibility for functional extension. Explicitly, extended functionalities are implemented in the form of Xilinx IP core and are integrated into the system using Xilinx Platform Studio. AMBA AXI4-Stream Interconnect links IP cores through high bandwidth unidirectional bus.

Figure 2 shows the top level block diagram of both NetFPGA 10G and rrBox, where rrBox is the core of the proposed reconfigurable middlebox. Figure 3 shows the functional block diagram of rrBox. In this work, the NetFPGA 10G is functioning in standalone mode and host computer is not required when using the proposed platform. All hardware designs were behaviorally described using Verilog Hardware Description Language and synthesized using Xilinx ISE DS 13.4. Furthermore, all hardware designs were verified using ModelSim simulation before tested experimentally in the NetFPGA 10G development board.

A. Partial Reconfiguration Design Flow

Xilinx provides several design flows for partial reconfiguration: Modular Method, Difference-based Method, Small Bit Manipulation Method, Early Access Method and Partition-based Method. This research uses the Partition-based Method because the other methods does not support Xilinx Virtex 5. The Partition-based Method is similar but less complex compared to the Early Access Method. The major design flow in the Partition-based Method includes modeling and

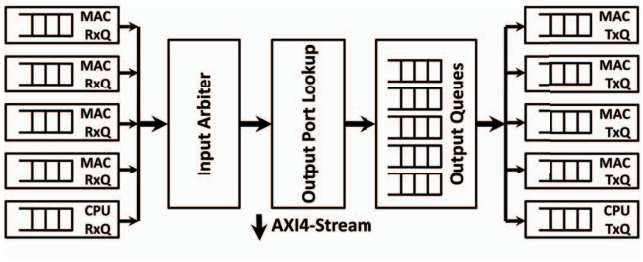

Figure 1. NetFPGA 10G Reference Pipeline [8]

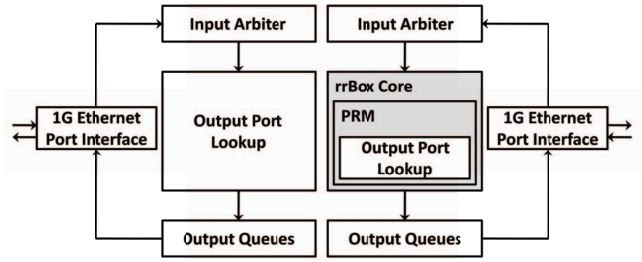

Figure 2. Top level architecture of NetFPGA 10G (left) and rrBox (right)

synthesis on all functional modules, defining partial reconfigurable module, defining design constraints and generating bitstream. The constraint file defines the area and location of the Partial Reconfigurable Region, the timing constraints of the implemented design and the location of physical pins.

B. FIFOs

Table I shows the list FIFOs and their role in the platform. These FIFOs are implemented using the on-chip BlockRAM in the FPGA device and functioning as the interface between blocks.

C. Header Parser

The Header Parser is responsible to parse the header of received packets and store the results in the Hdr_fifo. The header information extracted from the received packets includes the source and destination MAC addresses, source and destination IP addresses, UDP port number and segment number of bitstream packet. Besides that, this block differentiates bitstream packet from data packet based on the UDP port number and the unique identifier in the packet payload. Each platform has

Table I
THE LIST OF FIFOS IN FIGURE 3

FIFO Name	Role in the platform
A: Bit_fifo	Storing extracted partial bitstream for loading into reconfiguration port
B: In_fifo	Storing received Ethernet packets for forwarding to destination port
C: Hdr_fifo	Storing the extracted packet header informations
D: Bit_stat_fifo	Storing the extraction status of partial bitstream
E: Hdr_proc_fifo	Storing the result from Header Processor
F: Pl_proc_fifo	Storing the result from Payload Processor

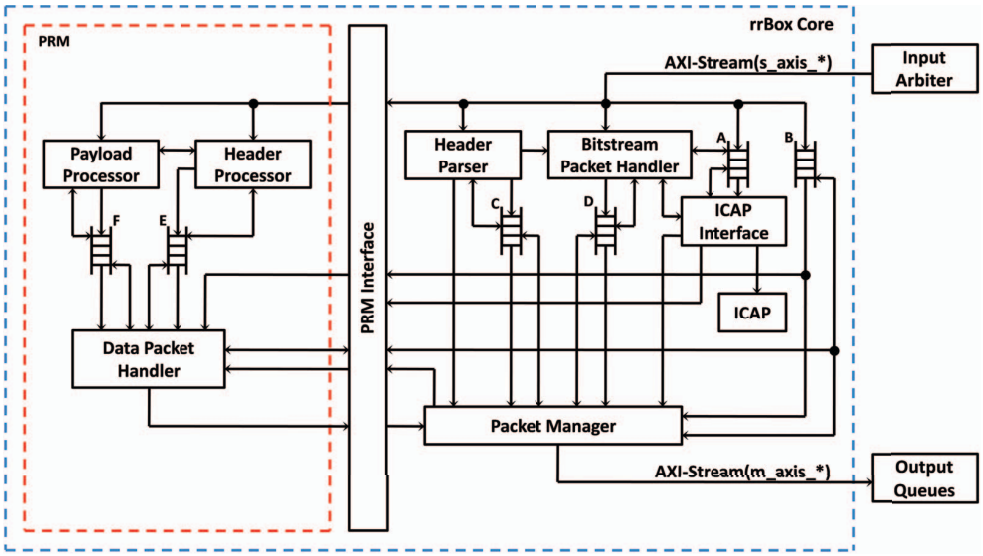

Figure 3. Functional block diagram of rrBox

a unique device ID in the packet payload for identification purposes and to enable mass deployment.

D. Bitstream Packet Handler

The Bitstream Packet Handler is responsible to extract partial bitstream from the bitstream packets. The extracted partial bitstream is temporarily stored in the Bit_fifo to wait for the acknowledgement before loading into the ICAP. This block also handles the bitstream packet verification and keeps track on the storage status of the partial bitstream. Any failure arises in storing the partial bitstream (due to the full FIFO) will result in unmarked bitstream packets. An unmarked bitstream packet is the acknowledgement to request resending from the rrBox Client. Each received bitstream packet is verified by the subsequent bitstream packet, where the arrival of $(N+1)^{th}$ bitstream packet will verifies the N^{th} segment of the partial bitstream, while the bitstream termination packet will verify the last segment of the partial bitstream. The verification mechanism is implemented to ensure the partial bitstream is stored and loaded into the ICAP in a proper sequence.

E. ICAP Interface

The ICAP Interface is responsible to handle the loading of the partial bitstream into the ICAP. Due to the verification mechanism in use, the ICAP Interface begins by loading the first segment of the partial bitstream into the ICAP upon the arrival of second segment. The loading of each partial bitstream segment takes several clock cycles because the ICAP bus width is only 32-bit. The ICAP Interface is essential because proper signals assertion and control are required to load the partial bitstream to the ICAP.

F. Packet Manager

The Packet Manager act as the coordinator for the rrBox Core. In general, the Packet Manager manages major processes in the rrBox Core by offloading processes to separate handlers. The Packet Manager activates either the Bitstream Packet Handler or the Data Packet Handler based on the type of packet received. Once the handler is activated, the Packet Manager will wait for the handler to complete the offloaded task before proceeding to the next task. During the dynamic reconfiguration, the Packet Manager handles the data packets on behalf of the Data Packet Handler. Additionally, the Packet Manager is responsible to initialize the Partial Reconfigurable Module after the remote reconfiguration process has been completed. In order to reduce the logic resources used, the communication protocol used for partial bitstream transmission is done using the User Datagram Protocol (UDP).

G. rrBox Client

The rrBox Client is responsible to read the generated partial bitstream and send it through the Ethernet to the NetFPGA. Since the size of the partial bitstream is large, the transmission of the partial bitstream to the NetFPGA requires several packets. In rrBox Client, the user can specify the size of each bitstream packet transmitted to the NetFPGA. The rrBox Client will resend any unmarked or timed out bitstream packets to ensure the partial bitstream is extracted and stored properly. Upon receiving the acknowledgement in the last segment of the partial bitstream, the rrBox will send the bitstream termination packet to the NetFPGA for verification.

H. Partial Reconfigurable Module

The Partial Reconfigurable Module consists of Data Packet Handler, Payload Processor, Header Processor and several FIFOs. Being located at the Partial Reconfiguration Region, all components in Partial Reconfiguration Module can be updated from time to time. The Partial Reconfigurable Module can be designed or modified based on the requirements from application.

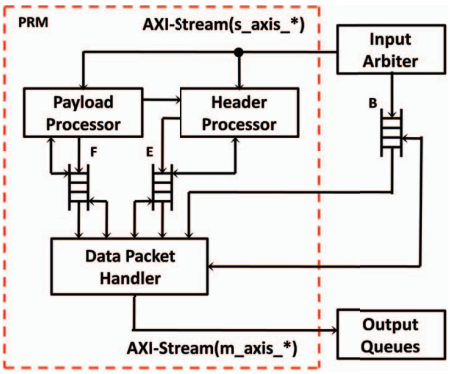

Figure 4. Conceptual view for designing PRM

1) Data Packet Handler: Data Packet Handler is responsible to forward data packets to Packet Manager based on the result from the Header Processor. The Data Packet Handler can be designed to do manipulation on data field based on specific application requirements. Additionally, the Data Packet Handler can be extended to support flow management by adding additional FIFO to store data packets. Lastly, the Header Processor can be designed to drop the data packet based on the specific preset conditions.

2) Header Processor: The Header Processor is responsible to process data packet based on header information. For example, the Header Processor for a switch controls data packet forwarding based on the source and destination MAC addresses in a packet. In fact, the packet forwarding algorithm in middlebox is implemented in the Header Processor.

3) Payload Processor: The Payload Processor is responsible to process data packet based on the packet payload. The Payload Processor is important to implement Deep Packet Inspection or Information Extraction for network applications such as intrusion detection system (IDS) and intrusion prevention systems (IPS). The Payload Processor can be customized based on different network application requirements.

IV. NETWORK PROTECTION IMPLEMENTATION

During the development of a certain network application, the implementation is focused on the design of Partial Reconfigurable Module as shown in Figure 4. For every different type of network application, the design and implementation of Payload Processor, Header Processor and Data Packet Handler may not be the same. Specifically, not all network applications require a Payload Processor or a Header Processor.

A. Port-based Firewall

In the OSI model, transport layer grants end-to-end communication services for applications within a network. Port-based firewall filters packets based on the TCP or UDP port number. Therefore, filtering packets based on its port number can effectively limits the communication service within a network as most of the services has been assigned with specific port number according to RFC 1700. Figure 5 shows the functional block diagram of a port-based firewall when implemented in

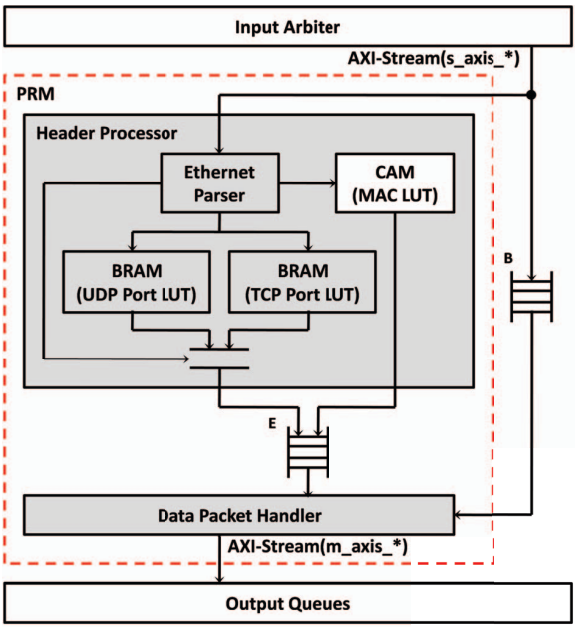

Figure 5. Functional block diagram of port-based firewall

the rrBox. The Header Processor consists of a Ethernet Parser to parse the header of packets, CAM for output port lookup based on the MAC address and BRAM for transport layer port lookup. In the functional block diagram, the Payload Processor is absent because a port-based firewalls process only data packets based on the packet header. The Data Packet Handler is responsible to forward or dropped data packets based on the result from the CAM lookup and TCP or UDP port lookup. With remote dynamic reconfiguration, the transport layer port lookup can be updated from time to time.

B. Stateless Network-based Intrusion Prevention System

Network-based intrusion prevention systems (NIPS) inspect the entire network to identify malicious activity and attemp to block it. Figure 6 shows the functional block diagram of a stateless network-based IPS when implemented in the rrBox. The NIPS uses signature-based detection, where it examines packets in the network to ensure the packets do not contain any pre-configured signatures. Thus, the implemented NIPS uses CAM-based string matching to examine the payload of the data packets. Figure 7 shows the architecture of CAM-based string matching for the NIPS. In the functional block diagram, the Header Processor consists of Ethernet Parser to parse the Ethernet frame and CAM for output port lookup based on the MAC address in the packet. The Payload Processor is made up of Signature Match, which are blocks of CAM-based string matching unit. The Data Packet Handler is responsible to forward or dropped the data packets based on the result from the Payload Processor and Header Processor. With remote dynamic reconfiguration, the signatures in the string matching block can be updated from time to time.

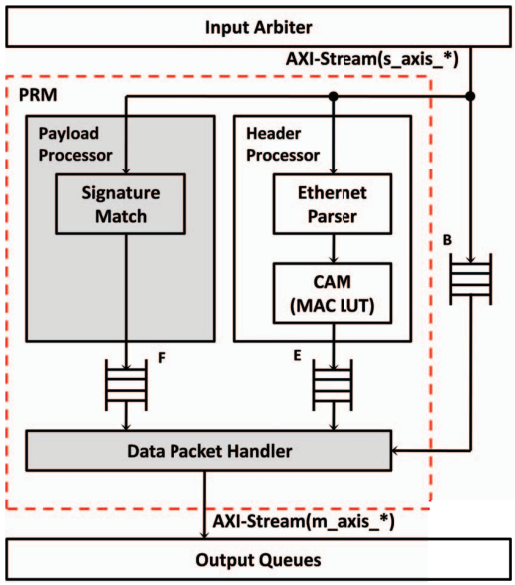

Figure 6. Functional block diagram of stateless NIPS

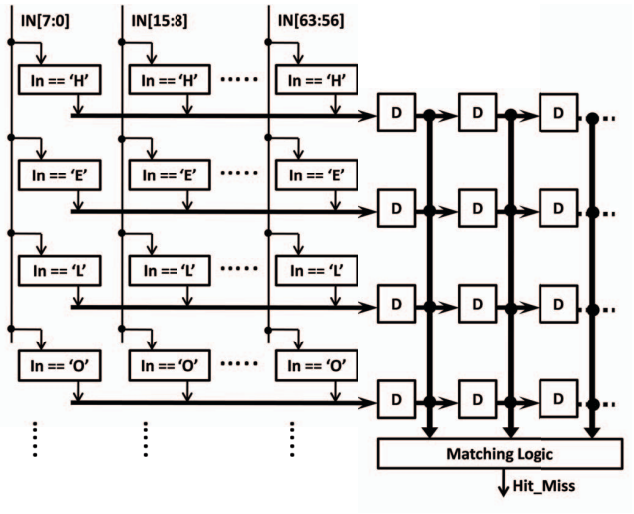

Figure 7. Architecture of CAM-based string matching

V. RESULT & DISCUSSION

A. rrBox Platform

Table II lists the logic resources required to enable remote dynamic reconfiguration in the NetFPGA 10G development board. In XC5VTX240T FPGA, there are 37440 slices, in which only 7137 is occupied (19% logic utilization) for the platform. Besides that, the utilization on the BlockRAM of the platform is approximately 23%. The full system includes the rrBox Core and other fundamental components in the NetFPGA framework (Input Arbiter, Output Queues and Ethernet Interfaces). Therefore, sufficient slices and BlockRAM are available for the implementation of the Partial Reconfigurable Module.

Table II
LOGIC UTILIZATION

Logic Resources	rrBox Core	Full System	Available
Number of Slice Registers	9672	16296	149760
Number of Slice LUTs	9640	16376	149760
Number of occupied Slices	4097	7137	37440
Number of BlockRAM	10	73	324
Number of BUFG	2	10	32

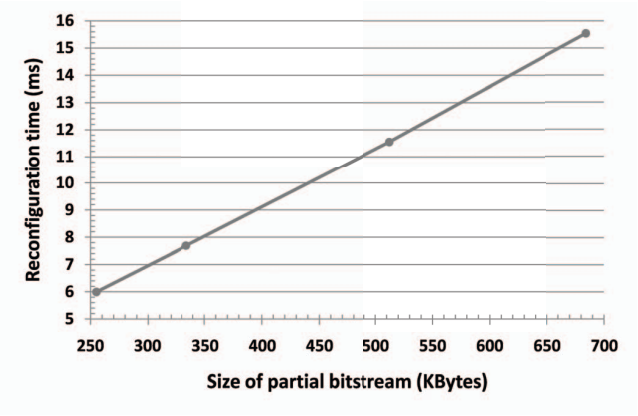

Figure 8. Reconfiguration time for various size of partial bitstream

Based on the implemented platform, various partial bitstreams sizes have been generated to obtain the platform performance. The generated partial bitstream sizes are 255KByte, 334KByte, 512KByte and 684KByte. These partial birstreams are generated on various area sizes of the Partial Reconfiguration Region. The reconfiguration time for each partial bitstream was recorded and plotted in graph depicted in Figure 8. From the graph, the linearity of the plotted line shows the consistency in reconfiguration throughput, which is around 350Mbps. The reconfiguration time shown in the graph includes the time required for the partial bitstream transmission through the Ethernet and the time required to load the partial bitstream into the configuration memory through the ICAP. In this platform, the reconfiguration frequency used is 50MHz and the ICAP bus width is 32 bit. In addition, the platform is operating at 100MHz and the AXI4-Stream bus width is 64 bit.

Table III shows the comparison between this work and other previous works. From the table, the reconfiguration frequency of this work is lower than [13], [14], [15], [16], [17] but the achieved reconfiguration throughput is the highest. This is because the reconfiguration controller is implemented using custom logics instead of using General Purpose Processor. Additionally, the 1Gbps Ethernet connection speeds up the partial bitstreams transmission. Although the reconfigurable device can affect the reconfiguration throughput, the reconfiguration throughput is still very dependent on the design architecture to achieve high efficiency in transmission of partial bitstream and the loading of partial bitstream. In fact, achieving high

Table III
COMPARISON WITH PREVIOUS WORK

Research work	FPGA family	Bitstream storage or transmission	Reconfiguration frequency (MHz)	Reconfiguration bus width (bit)	Size of partial bitstream (KByte)	Reconfiguration time (ms)	Reconfiguration throughput (Mbps)
[13]	Virtex 4	DDR-SDRAM	100	32	-	-	226.24
[14]	Virtex 2 Pro	Ethernet	100	8	200	-	80.00
[15]	Virtex 2	SRAM	66	8	290.838	56.8	40.96
[16]	Spartan 3	SRAM	-	8	-	-	35.50
[17]	Spartan 3	SDRAM	65	8	335	166	16.14
Proposed	Virtex 5	Ethernet	50	32	684	15.54	352.12

Table IV
LOGIC UTILIZATION FOR NETWORK PROTECTION

Logic Resources	Port-based Firewall	Stateless NIPS	Available
Number of Slice Registers	16320	>16566	149760
Number of Slice LUTs	16416	>16594	149760
Number of occupied Slices	7159	>7287	37440
Number of BlockRAM	77	73	324

reconfiguration throughput and high reconfiguration efficiency are a major milestone for the practical application of the developed platform.

Based on the functional block diagram of rrBox, the packet forwarding algorithms were implemented as a module in the Partial Reconfigurable Region. Thus, the packet forwarding algorithm can be updated remotely from time to time for different optimization and customization purposes. In order to verify the functionalities of the implemented platform, several packet forwading algorithms such as switch, hub and loopback were loaded into the platform remotely at run-time. The changes on the packet forwarding algorithms can be observed in client computers by using the packet injection and Wireshark packet analyzer.

B. Network Protection

Table IV lists the logic resources required to implement the port-based firewall and the stateless NIPS. The implementation of port-based firewall requires additional 22 slices and 4 BlockRAM added into the remote dynamically reconfigurable middlebox. This is because the port-based firewall requires additional BlockRAMs for the transport layer port lookup. However, the implementation of the stateless NIPS does not require any additional BlockRAM as the CAM-based string matching only requires slices for implementation. In CAM-based string matching, the slices utilization increase with the number of signatures.

In order to test the functionality of the implemented application experimentally, the developed platform was setup as in Figure 9. From the figure, there are 3 client computers connected to the developed platform and PC1 is injecting data packets to PC2, while PC3 dynamically reconfigure the developed platform. The test on port-based firewall involved packets injection to both blacklisted port and whitelisted port in PC2. As for the test on stateless NIPS, packet containing

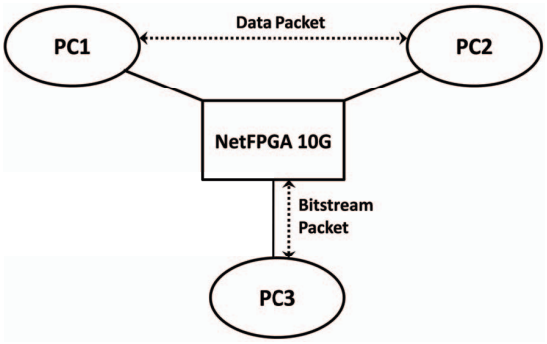

Figure 9. Platform setup for experimental test

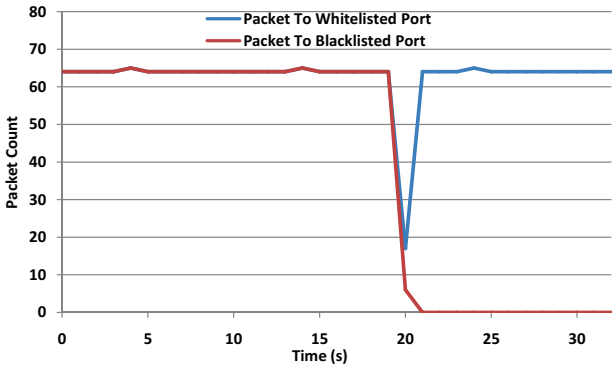

Figure 10. Data packets captured to test port-based firewall

signature and the ones without are sent to PC2. During the test, PC2 captures the data packets using Wireshark packet analyzer while PC3 dynamically reconfigures the platform with network protection extension. Figure 10 shows the graph of data packets captured in PC2 during the test of port-based firewall. Figure 11 shows the graph for the stateless NIPS. From the graphs, initially all data packets are allowed to pass through the developed platform with only packet forwarding capability. After the developed platform has been loaded with network protection extension, data packets that violate the preferred policy are dropped and not forwarded to destination computer.

The developed platform offers a number of benefits to middleboxes through remote dynamic reconfiguration. First,

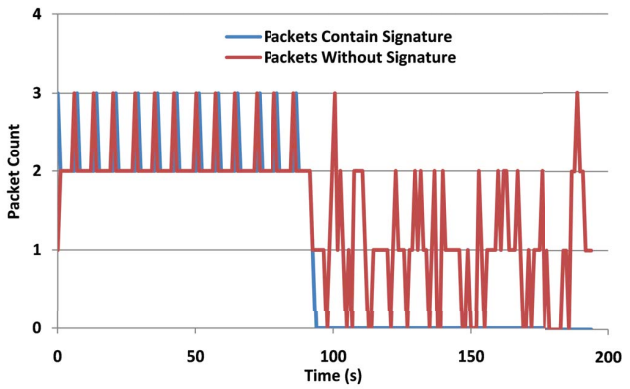

Figure 11. Data packets captured to test stateless NIPS

remote dynamic reconfiguration enables functional extension, where the platform is extended with network protection features at run-time. Second, remote dynamic reconfiguration allows the applications to adapt with the changes in functionality, where the configuration and policy in the platform can be updated from time to time. Third, remote dynamic reconfiguration provides flexibility to application for customization and optimization. For example, remote dynamic reconfiguration allows the implemented stateless NIPS to be optimized with various type of string matching algorithm, which can either CAM-based, hash-based or finite automata based. Lastly, the developed platform enables functional patch through remote dynamic reconfiguration. In essence, remote dynamic reconfiguration allows the application functionality to be extended, updated, optimized, customized or patched after the platform has been mass deployed.

Other network protection applications can also benefit from the remote functionality updates as part of the security vulnerabilities and execution requirements are hardly known during the application design time. Network traffic management is another network application that can benefit from remote dynamic reconfiguration as network traffic exhibit concept drift and traffic classifier requires regular updates to maintain its accuracy.

VI. CONCLUSION

In this paper, a remote dynamically reconfigurable middlebox has been implemented using the NetFPGA 10G development board. The developed middlebox is standalone and supports remote dynamic reconfiguration for applications updates through the Ethernet connection. Additionally, the developed middlebox provides flexibility to customize and optimize the packet forwarding algorithm. Network applications can be integrated into the platform by modifying the Partial Reconfigurable Modules, which are the Data Packet Handler, Header Processor and Payload Processor. Similarly, the demonstration of the developed plaform to implement network protection application is achieved by extending the network processing functionality in the Partial Reconfigurable Module. The developed middlebox is aimed to provide the feature to customize application after the deployment. The developed platform achieves 352.12Mbps of reconfiguration throughput, which is important for mass updating the distributed middleboxes and decrease the devices down time during the updates. In near future, traffic management application will be implemented using the developed platform for demonstration in practical applications.

ACKNOWLEDGEMENT

This work is supported in part by the Ministry of Science, Technology & Innovation of Malaysia (MOSTI) under ScienceFund Grant 01-01-06-SF1222 (UTM Vote No. 4S095).

REFERENCES

[1] R. Hartenstein, "The von Neumann Syndrome," in *Stamatis Vassiliadis Memorial Symposium*, 2007.
[2] J. Becker and R. Hartenstein, "Configware and morphware going mainstream," *Journal of Systems Architecture*, vol. 49, no. 4, 2003.
[3] A. Schallenberg, *Dynamic partial self-reconfiguration: Quick modeling, simulation, and synthesis*. Suedwestdeutscher Verlag fuer Hochschulschriften, 2010.
[4] T. Sato and M.-a. Fukase, "Reconfigurable hardware implementation of host-based IDS," in *Communications, 2003. APCC 2003. The 9th Asia-Pacific Conference on*, 2003.
[5] S. Li, J. Torresen, and O. Soraasen, "Exploiting reconfigurable hardware for network security," in *Field-Programmable Custom Computing Machines, 2003. FCCM 2003. 11th Annual IEEE Symposium on*, 2003.
[6] A. K. Tummala and P. Patel, "Distributed IDS using reconfigurable hardware," in *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, 2007.
[7] J. Naous, G. Gibb, S. Bolouki, and N. McKeown, "NetFPGA: Reusable router architecture for experimental research," in *Proceedings of the ACM workshop on Programmable routers for extensible services of tomorrow*, 2008.
[8] M. S. S. G. Antichi, Gianni and A. Moore, "From 1G to 10G: code reuse in action," in *First Workshop on High Performance and Programmable Networking 2013*, 2013.
[9] D. Yin, D. Unnikrishnan, Y. Liao, L. Gao, and R. Tessier, "Customizing virtual networks with partial FPGA reconfiguration," in *Proceedings of the second ACM SIGCOMM workshop on Virtualized infrastructure systems and architectures*, 2010.
[10] M. N. Krifa, B. Ouni, and A. Mtibaa, "Exploring the self reconfiguration of FPGA: Design flow, architecture and performance," *International Journal on Computer Science and Engineering*, vol. 3, no. 4, 2011.
[11] S. Pontarelli, C. Greco, E. Nobile, S. Teofili, and G. Bianchi, "Exploiting dynamic reconfiguration for FPGA based network intrusion detection systems," in *Field Programmable Logic and Applications (FPL), 2010 International Conference on*, 2010.
[12] K. Zhang, X. Ding, K. Xiong, B. Yu, and S. Dai, "RSS: A reconfigurable security system designed on NetFPGA and Virtex5-LX110T," in *1st European NetFPGA Developers Workshop*, 2010.
[13] M. Hubner, D. Gohringer, J. Noguera, and J. Becker, "Fast dynamic and partial reconfiguration data path with low hardware overhead on Xilinx FPGAs," in *Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*, 2010.
[14] P. Bomel, J. Crenne, L. Ye, J.-P. Diguet, and G. Gogniat, "Ultra-fast downloading of partial bitstreams through ethernet," in *Architecture of Computing Systems–ARCS 2009*. Springer Berlin Heidelberg, 2009.
[15] L. Braun, K. Paulsson, H. Kromer, M. Hubner, and J. Becker, "Data path driven waveform-like reconfiguration," in *Field Programmable Logic and Applications, 2008. FPL 2008. International Conference on*, 2008.
[16] E. Cantó, M. López, F. Fons *et al.*, "Self reconfiguration of embedded systems mapped on Spartan-3," in *4th International Workshop on Reconfigurable Communication Centric SoCs (ReCoSoC 2008)*, 2008.
[17] I. Gonzalez, E. Aguayo, and S. Lopez-Buedo, "Self-reconfigurable embedded systems on low-cost FPGAs," *Micro, IEEE*, vol. 27, no. 4, 2007.