Improving the Sending Per-Packet Processing Overheads for High-speed Network using Specialized RISC Processor

Dr. Mohamed Elbeshti
Dept. of Computer Science- faculty of Science, Zawia University

Abstract:

The promise of 40 and 100 gigabit Ethernet in the near future shows that the processing needed for a network protocol is scaling at least as quickly as communication speed. To keep up with this rapid increase in speed, the end nodes have to increase the amount of the packet processing to avoid the bottleneck in the network. Large Sending Offload LSO is a defacto standard, which is offloaded to network interface for sending packets up to 10 Gbps. In this paper, we have provided an appropriate approach that can scale the LSO to high-speed communication line beyond the 10 Gbps. A specialized cost-effective RISC core with low rate power

has been designed to execute the new scheme for the LSO to support communication rate up to 100 Gbps. Other devices are; also implemented to support the RISC like the DMA. The processing cycles that the RSIC needed for TCP/IP and UDP/IP has been measured. The RISC's performance is also presented. A moderate rate with 423 MHz RISC core can support the sending-side processing for up to 100 Gbps transmission speed for the TCP/IP and UDP/IP protocols. A DMA with 2115 MHz is applied in order to reduce the idle cycles of the RISC core.

Keywords; *Large Sending Offload (LSO); RISC core; VHDL behavior model; Cycle-accurate performance evaluations.*

1.Introduction:

Using ASIC to design NIs provide a greater energy efficiency and better integration than programmable-based. ASIC-based implementations can also offer better performance than off-the-shelf processor-based implementations. However, ASIC also limits flexibility, limits upgradability, and makes NI design tailoring difficult in changing the algorithm of the protocol or supporting a new version of protocols.

The recent advances in the area of CAD tools and Hardware Description Languages (HDL) have made the design of embedded processors for the performance of certain functions possible (e.g., recent chips from Cavium [1] or Tilera [2]). System-on-chip technology has also enhanced the possibility of integrating the hardware blocks required in the NI and the General Purpose Processor (GPP) to be carried on one chip [3].

The main contribution of this work is to enhance the Large Sending Offload (LSO) processing cycles. These enhancements in the protocol processing (e.g., the TCP/IP and UDP/IP) focus on reducing the pre-packet

processing overhead inside the network interface for the next generation network.

The second contribution is designing and implementing a sending-side for network interface that can run the proposed algorithm for high-speed communication lines up to 100 Gbps using an embedded processor. Many cost effective embedded cores have become available and can be embedded to the Ethernet NI chip. However, these processors are not optimized for LSO. Since these processors are designed to support general functions, such as the control unit has to support general functions, complex instructions long and variable execution time. These GPP also have a large number of registers to accommodate all the possible use. These features of the GPP might not be needed in NI. These advances in the GPP have directed this research in investigating the use of specialized RISC cores to process the developed LSO. At this stage, a behavior model of the LSO processing is done to study the efficiency, scalability, and performance of the designed model. In the future, we will target the design to a fitting device to record a delay and energy consumption of the proposed model.

The rest of this paper is organized as follows: Section 2 discusses the sending side processing. In section 3, the model structure for sending side. The behaviour model is in section 4. The core design is highlighted in section 5. The VHDL-based simulation results are discussed in the next section.

2. Large Sending Offload Processing :

The LSO feature is helpful only on the transmit side, which is freeing an OS from the task of segmenting the application's that are larger

than the Maximum Transport Unit MTU [4]. The core engine in the NI is responsible for handling these tasks related to transport layer. For instance, the core engine in the NI divides the application data into Maximum Segment Size (MSS) (i.e. 1460 bytes for TCP segment or 1472 bytes for UDP fragment). The core also requires generating the packet header for each outgoing MSS before sending the packet to MAC unit. Performing complete packets (MSS + TCP/IP or UDP/IP header) are according to the protocol type. For instance, TCP/IP protocol uses the two identifiers in each packet; the Sequence Number (SN) and the Acknowledgment Number (AKN). The beginning segment carries the start sequence number of data and the AKN, which is a SN of the next expected data portion of the transmission [5]. Fragmenting the UDP is quite different from TCP where the fragmentation of the UPD messages is based on the IP header, such as the MF field and the offset fields [6]. Each of the IP headers carries their own header length, packet length (PL), and application data (AD). Where the data packet is s defined as the packet length (PL) minus the length of its IP header (PHL). From Equation 1, two fundamental concepts that the core engine can be derived: the fragment offsets, and number of constituting fragments.

$$AD = (MTU - HL) + (MTU\text{-}HL) + ADn$$
$$AD = (n\text{-}1)\,(MTU\text{-}\,PHL) + And \qquad (1)$$

The datagram can be sized up to 64 K byte (the receiver's TCP window size is set to 64 K byte). At the NI, the core engine after reading all the information related to the moved datagram, such the position of the message inside the NI's buffer and the MSS, and then core examines the size of the moved datagram. If the datagram is larger than the MTU, then the core engine starts generating the network header for each MSS data.

The smart NI [7, 8] holds a TCP/IP header template that has the IP total length, the initial SN. A copy of the template header whenever there are a segment data needs to be sent to a network. It updates the essential fields inside the TCP and the IP header of the copied headers before sending a packet to MAC unit, such as the SN inside the TCP and the datagram total length inside the IP header. However, these processing scenarios in these implantations are successful in offloading the LSO to the NI, but still cannot be scale it to express network since the header copy itself required at least 10 cycles over the 64-bit bus (40 bytes of headers). In addition, there was no explanation for the data movement's methods inside the NI. Moreover, there is no proven if the selected model can be used with other protocols such as the UDP/IP.

This paper provides an alternative method for sending data faster to the physical and MAC units. This approach has focused on the header process and data movements. For header processing, we have provided a new algorithm for enhancement of the flow of the packets processing. After the a host CPU stores the specified data that needed to be sent to a network in the NI's Buffer, the core engine inside the NI then starts examining the moved data. If the application data is over the MTU then, the first MSS of the application data is a Beginning of Message (BOM). The core starts with packet generating the network headers for BOM. Conceptually, each packet required to a SN and AKN number inside the TCP header [5]. Within the processing of the BOM, the IP total length is 1500 bytes (the default MTU), unless the two networks ends specifies different size during the connection setting up. The second part of application data is a Continuation of Massage (COM). With COM, only needs to update some fields inside network packet. For instance, the core engine requires updating the SN and

the AKN inside the TCP. The total length of the COM remains the same (1500 bytes). In addition, the previous AKN is a SN of the current packet. End of Message (EOM) is the last part of the message, which contains the remaining data of the sent datagram. With EOM, the packet length not always the same as the COM, then it needs to update the packet length inside the IP header with the actual reaming data size. When the host CPU sent a small sized data to NI (less than MTU), this is considered as a Single Segment Message (SSM), can be sent as is to the MAC layer. Packets with zero length are (signaling packets) are send as to MAC unit for further processing.

To reduce the required processing that the core processor needed generating the headers, we have added additional features to the LSO processing BOM, COM, EOM or SSM, which is overlapping processing technique. Processor can benefit from the transfer of data period by implementing other processing required for the followed packets. We have adjusted the LSO assembly code to keep the core engine busy while transferring data. For example, the core calculates the remaining datagram size inside the NI's buffer to figure out which subroutine code should follow either COM or EOM.

3.Netwrok interface model for sending side:

In designing the target NI model, we have avoided using multiprocessing cores as processing cores at the NI to serve a single function, the LSO [9]. This study aims to use a single specialized RISC core for processing the outgoing TCP and UDP packets. The core structure and performance are addressed in this work. The have structured the proposed sending-side the NI into three parts: communication Line

Interface (LI), kernel processing and Host Interface (HI) “Figure 1”. The HI and LI are implemented in hardware. The processing unit in the NI, which commonly processed functions that are related to header processing, is an embedded specialized RISC. Since the receiving-side and sending-side operations are completely independent, the NI is designed to handle both operations in parallel. Two RISC-cores are considered for being in the NIC: one for *Sending-side* and the other for *receiving-side.*

A Communication with the host CPU:

The NI communicates with the host through two FIFO buffers. The FIFOs were implemented as memory-based, and the pointer of each FIFO is stored in the RISC's register. The RISC reaches any FIFO after reading its address. After the host CPU moves the message to the Sending buffer (SB), the host required to notify the sender RISC by sending the location of the message inside the SB and other necessary information needed for segmented the message, such as the MSS through FIFO 2. The SEP also sends the notification of the sending data through FIFO1to host CPU. These FIFFO, therefore, is implemented to reduce the interrupt mechanism that happens during the exchange of information which affects the overall performance of NI or the host CPU [10]. Interrupting the host CPU or RISC cores (Sending Embedded Processor (SEP)) during their processing time effects the packet processing time [11], especially when the budget time for processing a packet is small (for 100 Gbps the budget time is 123 ns per packet [12]).

B. Buffers :

The Sending Buffer Interface (SBI) VHDL based contains two buffers each of which holds one packet (1500 bytes). The sequential machine controls the SBI. Only one buffer activated to receive data at a given time. The buffer will remain enabled until the complete packet has been stored. Using two buffers contribute to the process of parallel processing between the core engine and the MAC unit.

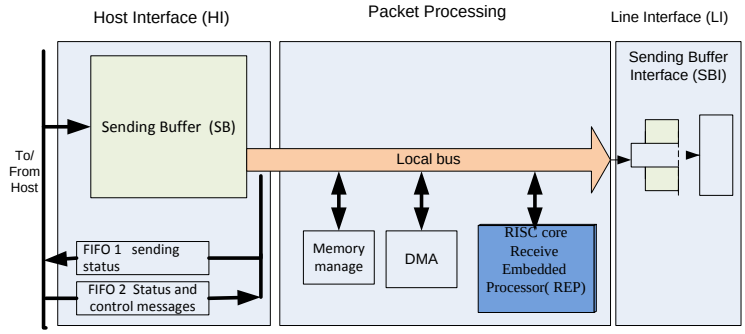


Figure 1: Sending Block Diagram

The sequential machine allows storing data in one buffer. After the buffer gets fill, then sequential machine switches to the other SBI buffer. SB is designed to be dual port memories. The SEP and the host can access the local memories simultaneously. A 64-bit wide bus is used in this work for transferring data from the SB to SBI.

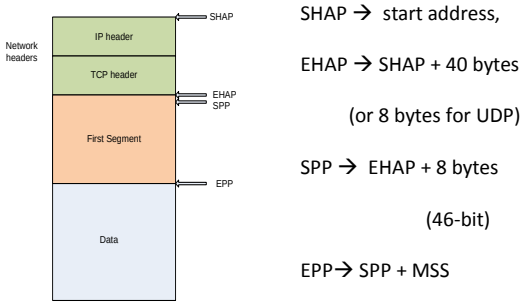
C. Data movement methods:

Using the Programmed I/O (PI/O) method for data movement makes the RISC core controlling the bus while data is moved. This means the RISC processor is busy with the transfer of data from one location to

another, especially when moving a large amount of data (1640 bytes) and cannot executes other instruction. In this work, the DMA is used for transferring data between the SB and the SBI. The RISC core initiates and controls the DMA. Since the local bus is shared between the DMA and the RISC core, the RISC core requires releasing the local bus to let the DMA controller performing the data transfer. Each transfer of 64-bits consumes two cycles. First cycle, the DMA controller reads the source buffer to get 64-bits to the DMA's register. During the second DMA cycle, the words move from the DMA's register to the destination buffer. The DMA state machine will then provide the read and write signals to the source and destination buffers. The state machine in the DMA is also responsible for incrementing of the address counter. A DMA with a single channel is chosen in this work since only a single transfer at a time. Although, the use of the DMA reduces the RISC processor instructions cycles from transferring data, but it is not always the case that the DMA is better for all data transfers. If the packet size is small (i.e. 64 bytes), the payload part is only 6 bytes [12], the possible transfer can be handled more efficiently in the PI/O. The overhead of setting up the DMA activity becomes comparable to the cost of moving the data in PI/O. The decision on whether to use the hardware or the software mode depends on the size of the data.

We have simulated the data movements for segmentation and fragmentation function for TCP and UDP packets. The DMA controller is responsible for moving the packet header as well as the payload part from SB to SBI for both TCP and UDP protocol. If we consider the core engine, responsible sending the packet header from its register to SBI, then it copies the data to its register then stores it to SBI buffer. The core has to wait for the DMA controller to release the local bus in order to deliver the

headers from its registers to SBI. In this simulator, the RISC core initiates the DMA to transfer data from SB to SBI. The core is responsible to update the packet headers for each segment inside the SB. Manage the messages inside the SB, the processor uses several pointers in order to continue updating packet headers inside the SB "Figure 2"; the Start Header Address Pointer (SHAP), End-Header Address Pointer (EHAP), Start Payload Pointer (SPP) and End-payload Pointer (EPP). The RISC core uses the SHAP pointer for reach the network headers in side the SB. The SPP pointer helps the RISC to locate the start of the message. The EPP is used to point at the end of the last segment. The RISC updates this pointer during the data movements of the first packet (the BOM).



4. Cycle processing of the behavior model:

The TCP payloads are varying in size from 6 to 1640 bytes [12]. The DMA required moving data (i.e MSS is 1460 bytes) from the SB to SBI is 366 cycles (183 cycles to read payload data over the 64-bit bus to the DMA's data register and 183 cycles to store it to RB). Clearly, the RISC core will be in idle mode until the DMA completes moving the data "Figure 3". The RISC can execute 6 instructions during the data moments and becomes idle with MSS at about 359 instructions. The idle cycle's

time affects the performance of the network card and its capabilities to deal with high speed networks. Small size packets, such as 64 until 256 bytes, may require less DMA cycles than other packets that have more payload bytes. However, using these small size packets could improve the NI's performance, yet it affects the end node's throughput [13]. We have focused on the 512 bytes packet to 1500 bytes packet. The use of small packets can be studied on this Model, but they bear little payload data and may not be able to achieve 100Gbps.

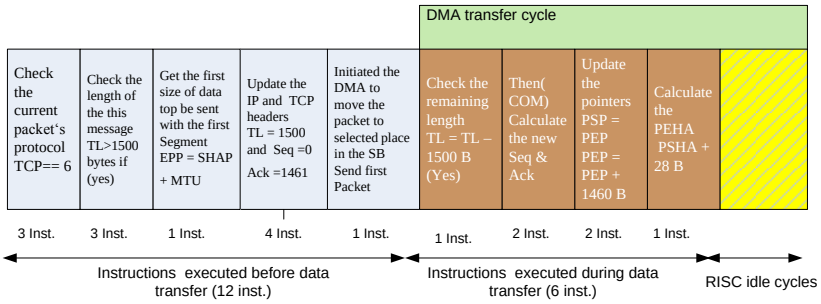


Figure 3: SEP processing the BOM required 18 instructions

The total RISC cycles are measured for BOM, COM, EOM and SSM for both TCP and UDP protocol "Figure 4". The network performance becomes poor to perform the target goal, the 40 and 100 Gbps. The RISC instruction recorded 50 cycles (packet processing and idles cycles) when performing the BOM messages. Table 1 shows the total instruction that the RISC need to complete the TCP or UDP packets. It is clear that, the idle cycles are reducing the NI's performance. We have studied the ways that can be used to reduce or to eliminate the idle cycles of the RISC. One of these solutions is the use of a multi-bus based on the sending side. The RISC can access the multiport memories while the DMA controller moves

data. The other approach is to use a DMA that runs at a higher clock rate than the RISC

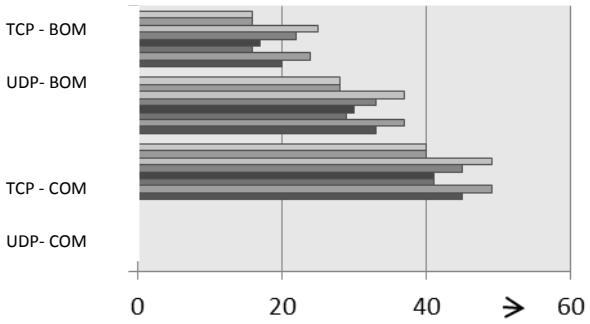


Figure 4: Total RISC processing instructions for TCP and UDP packets

Table 1: Total RISC Instructions for Segmentation and Fragmentation when the DMA has Five Clock Cycle of the RISC

Packet Type		Packet Size					
		1500 bytes		1024 bytes		512 bytes	
		Total RISC Inst.	Idle Inst.	Total RISC Inst.	Idle Inst.	Total RISC Inst.	Idle Inst.
TCP	SSM	45	37	33	25	20	12
	BOM	49	31	37	19	24	5
	COM	41	31	29	19	16	6
	EOM	41	37	30	25	17	11
UDP	SSM	45	37	33	25	22	13
	BOM	49	31	37	19	25	7
	COM	40	32	28	20	16	8
	EOM	40	37	28	25	16	12

We have adapted the way of using it that we presented in "Figure 1", since it is a straightforward data path scenario and easily implemented

without any changes in the NI's architecture. We have started adjusting the DMA's clock to reduce the idle cycles. In order to study and analyse the cycle –accurate NI simulator, we sent different large packets to the sending side. Each time we increased the DMA's clock to reduce the idle cycles, we have noticed that the RISC core and DMA controller were working quickly to complete each message and transfer it to RB. When DMA's clock has five times the embedded processor core, the NI performance is increased significantly, where most, if not all, the idle cycles are reduced (Table 2). Table 2 presents the total RISC and idle cycles for TCP/IP and UDP/IP packets when the DMA become five times the clock rate of the RISC core. When the packet size is 512 bytes, the idle cycles are reduced significantly. The DMA and the RISC core clock rate have measured "Figure 5". The DMA with 2115 MHz is found when the packet size is 512 bytes. This naturally increases the speed of the DMA with increased sending packets from SB to SBI. We have fixed the DMA clock rate to 2115 MHz and used this rate with other packet sizes (larger than 512 bytes). This rate of the DMA's clock helps to reduce the idle cycles in the other packets those are larger than the 512 bytes, such as 1500 bytes. The performance of the NI is enhanced significantly when the DMA is 2115 MHz. This is because the number of messages that the core needs to send is less than in the case of 512, which is only 81274382 packets per second when the packet size is 1500 bytes.

Table 2: Total RSIC Instructions to Complete Processing THE LSO, when the DMA Becomes 2115 MHz

Packet Type		Packet Size					
		1500 bytes		1024 bytes		512 bytes	
		Total RISC Inst.	Idle Inst.	Total RISC Inst.	Idle Inst.	Total RISC Inst.	Idle Inst.
TCP	SSM	18	9	15	6	20	11
	BOM	23	4	20	1	24	5
	COM	14	4	11	1	16	6
	EOM	15	9	11	5	17	11
UDP	SSM	8	0	8	0	22	13
	BOM	23	5	20	2	25	7
	COM	13	5	11	3	16	8
	EOM	13	9	11	7	16	12

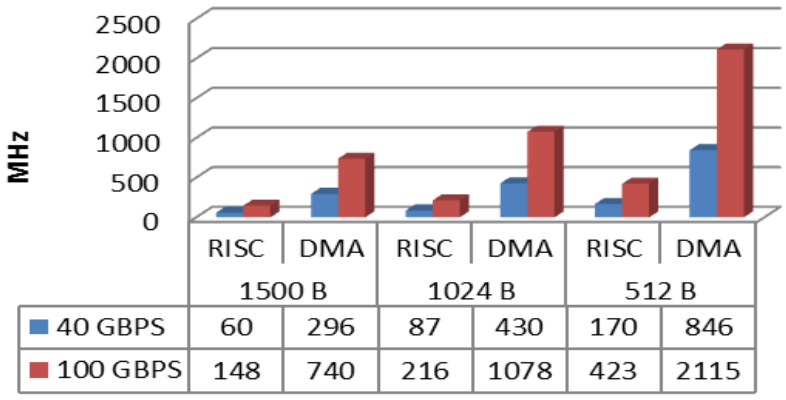


Figure 5: The RISC and DMA clock rate in MHz for TCP Segmentation and UDP Fragmentation (When the DMA has five RISC's clock rate)

5. RISC core :

Design a RISC core for specialized application, namely NI control and data path, is simpler than using the off-the-shelf GPP processors. These general-purpose embedded processors are not optimized for a LSO function. Hence, some portions of GPP instructions that support general-

purpose applications may not be required for the ENI design. For example, the Floating-Point Unit is not necessary for network interfaces. Also, we found that, using a data cache to store data is not required since it will not enhance the NI's performance or reduce the RISC' clock for this application. The elimination of these units in the design the core simplifies the process of development of NI and reduces the size and cost.

RISC pipelines divide the execution of an instruction into a number of steps, or pipeline stages. The depth of a pipeline corresponds to the number of pipeline stages. The NI RISC core has been designed to execute one instruction in three-pipeline stage: Fetch an instruction from local memory (Fetch stage). Decode/execute the instruction and registers read (Decode/Execute stage). Store results back into the destination register (write back, or W/B, stage).

We have noticed also that the RISC performs a few of the instructions to complete processing the LSO. These instructions are load, store, arithmetic and logic operation and conditional branches. The minimum type the instructions set used in the LSO function would make the control unit design simple and fast. In addition, the limited number of instructions that are required to support the Ethernet interface processing can reduce the size and complexity of the control unit leading to an increased speed.

6. Simulation results:

In the LSO function processing, it is clear that the RISC processing time becomes less when the DMA has a clock rate faster than the RISC core is (Five times faster the RISC's clock) where all the idle cycle associated with the RISC core processing has eliminated. We monitored the highest clock rate of the RISC core during processing the different packets. We have found the VHDL behavior model for the sending unit of the network interface has a 148 MHz RISC processor that can support 100

Gbps lines, when the DMA speed is 2115 MHz, and the packet size is 1500 bytes "Table 3". A RISC core with 423 MHz can be used to process the LSO at 100 Gbps when the packet size is 512bytes.

A comparison of the RISC performance with the two approaches has been implemented. The First is copying the TCP and IP headers (that Host CPUT sent) from the SB to RISC's internal register. The Second is to updating the packet headers is side the SB. After generating the packet header for each segment, the RISC needs to send the TCP and IP header to SBI for further processing. As a result, with the first LSO processing the RISC with 893 MHz is needed for 100 Gbps "Table 4". It is clear that the RISC could spend more cycles than second approach since it needs to transfer the packet header from the internal buffer to SBI. Modify the original header packet within the SB enhances and improves process LSO. Initiating the DMA to transfer the packet header from the SB to SBI reduces the power of the RSIC to 423 MHz. The second approach of processing LSO also gives the RISC core more space executing other functions that do not need to use local bus, such as calculating the next header's fields or checking the remaining size of the application data inside RB.

Table 3: RISC Clock Rate for LSO using DMA for data Transfer (When the DMA 2115MHz)

	RISC MHz		
Packet Size	1500 byte	1024 byte	512 byte
40 GBPS	60	87	170
100 GBPS	148	216	423

7. Conclusion:

We have reported on improving the Large Sending Offload per-packet processing overheads inside the network interface to keep up with

the rapid increase in speed. We have presented a computer simulation results to measure the amount of process-support a wide range of transmission line speed, up to 100 Gbps. A 423 MHz RISC core can support the sending side processing for for TCP/IP and UDP/IP. Assuming a fast DMA (2115 MHz) is required to eliminate the RISC idle cycles. The DMA clock is considered high; this is because of the size of the local bus is 64-bit. The DMA clock rate decreases

support a wide range of transmission line speed, up to 100 Gbps. A 423 MHz RISC core can support the sending side processing for for TCP/IP and UDP/IP. Assuming a fast DMA (2115 MHz) is required to eliminate the RISC idle cycles. The DMA clock is considered high; this is because of the size of the local bus is 64-bit. The DMA clock rate decreases significantly if the local bus becomes wider (i.e. 320 bit [14]). The scalable NI based programmable could provide the flexibility needed for adding a new, or modify, protocol functions while ASICs based solutions could provide better performance but are not flexible enough to add new or modify features.

Table 4: A comparison of the RISC performance with header processing approaches.

	RISC MHz			
	RISC responsible for Header movements		Our Method, using initiating the DMA for LSO	
	40 Gbps	100 Gbps	40 Gbps	100 Gbps
1500 bytes	125	313	60	148
1024 byes	182	455	87	216
512 bytes	357	893	169	423

References:

[1] Cavium, Octeon II CN63XX Intelligent Network Adapter Family http://www.cavium.com/pdfFiles/OCTEON_II_CN63XX_Adapter.pdf?x=2

[2] Telera, Network interface-based processor http://www.tilera.com/sites/default/files/productbrief/ TILEProEncore_PB024_v5_0.pdf (accessed Sep,2016)

[3] C. Cranor et al .Architecture considerations for CPU and network interface integration IEEE Micro, January–February (2008), pp. 18–26.

[4] RFC 1191- MTU discovery. Nov 1990 .

[5] J. B. Postel, "Transmission Control Protocol," NIC- RFC 793, Information Sciences Institute, Sept. 1981 .

[6] J. Postel RFC 791 Internet Protocol, protocol specification 1981

[7] O. Cardona and J. B. Cunnlngham." System Load Based Dynamic Segmentation for Network Interface Card." U. S. Patent 0295098 A1. 2008

[8] G.Wiilium and W. Paul." ofload of TCP Segmentation to a Smart Adapter." U. S. Patent 5937169. 2014 .

[9] H. Kim, V. S. Pai and S.Rixner, " Exploiting task-level concurrency in a programmable network interface," Proceedings of the ninth ACM SIGPLAN symposium on Principles and practice of parallel programming, pp 61-72, 2003.

[10] C. Mogul and K. K. Ramakrishnan. Eliminating receive livelock in an interrupt-driven kernel. ACM Transactions on Computer Systems, 15(3):217-252, 1997.

[11] H. Jin and C. Yoo. "Impact of protocol overheads on network throughput over high-speed interconnects: measurement, analysis, and improvement." Journal of Supercomputing,Volume 41,Number 1/July,200

[12] G. Held "Ethernet Networks (4th ed)," Design, Implantation, Operation and Management. John Wiley publisher LTD, 2003.

[13] S. Makineni and R. Iyer," Measurement-based analysis of TCP/IP processing requirements," In 10th International Conference on High Performance Computing (HiPC 2013),Hyderabad, India, December 2003.

[14] Altera."40- and 100-Gbps Ethernet MAC and PHY" June 2014