Modeling and Analysis of IXP425 Network Processor

Fakhraldeen H. Ali **Omar F. Ahmed**
Computer Engineering Department
University of Mosul – IRAQ
Email: fhali310@yahoo.com

Abstract

There are mainly two kinds of network processor: coprocessors-centric model, and core-centric model. In the coprocessors-centric, the data-plane is handled by coprocessors. The core processes most of the data-plane packets yet offloading some tasks to coprocessors in the core-centric model. The IXP is one of Intel network processor series which is a core-centric model. They are optimized for home, small-to-medium enterprise, and networked embedded applications. This work aims to design a module of IXP425 performing VPN. The processing time for each stage is available in the datasheet of the IXP425. The Markov chain and Omnet++ has been adopted in this work to explore the system parameters and bottlenecks focusing on the buffer length and how to be optimized for each stage. Single process programming is considered in the IXP425 operation.

Keywords: VPN, network processor, configurability, flexibility, scalability.

نمذجة وتحليل اداء معالج الشبكات IXP425

فخر الدين حامد علي **عمر فخري أحمد**
قسم هندسة الحاسبات في جامعة الموصل – العراق

الخلاصة

هنالك نوعان رئيسيان من معالجات الشبكات الاختلاف الجوهري بينهما ينصب حول كيفية المعالجة على مستوى البيانات حيث اما يتم ذلك من قبل المعالج المركزي او من قبل المعالجات المساعدة. يركز هذا البحث على معالج شبكات انتجته شركة انتل حيث تم نمذجة هذا المعالج واعتماد الازمنة المتوفرة في المواصفات الفنية له لاجراء محاكات لقياس ادائه وتأشير الاختناق أو عنق الزجاجة مع التركيز على دوائر العزل والحجم الأفضل لكل مرحلة وصولا إلى أفضل أداء عند تبني Markov chain و Omnet++ .

Received: 11 – 4 - 2010 **Accepted: 26 – 6 - 2011**

1. Introduction

Security and content-aware processing are part of the features supported by network applications which demand much more powerful hardware platforms to achieve them. VPN (virtual private network) needs intensive computational process where general-purpose processors are often adopted to achieve it. However, the cost is considerable while the throughput is not satisfactory because of the heavy cryptographic operations [1]. On the other hand, the application specific integrated circuits (ASICs) can support the performance requirement with a circuitry designed for both networking and cryptographic processing, yet it does not have the ability of reconfiguration which makes it less appealing.

Network processors have been considered as an alternative solution to deal with the above-mentioned problems for their core-processor/coprocessors based architecture on which control and data-plane processing are separated for efficiency and re-programmability for functional adaptations. In the core network processor, the complicated operations and control messages are the core processor responsibility, while a number of multithreaded coprocessors, having specifically designed instructions for networking purposes (ASIC), are employed for mass data plane processing. This architecture called coprocessors-centric; is frequently applied as a core device, which requires simple configurability, but high scalability [2, 3, 4, 5]. Both control and data-plane packets are processed by the core processor in the implementation of an edge device which deals with relatively mild traffic volume. This model is also called the core-centric model. However, tasks need computational intensive works like receiving, transmission, and en-/de-cryption are offloaded to certain application-specific coprocessors [4].

Applications like DiffServ (Differentiated Services), VPN cryptographic algorithms, and intrusion detection and prevention (IDP) are studied by several researchers who discovered the feasibility of adopting the core-centric models in packet processing for the above network applications. Beside the evaluation through implementing both models to find out system bottlenecks, mathematical modeling is favored for the coprocessors centric model in order to unveil design implications that are unlikely to be observed through real benchmarking [3, 6, 7]. In this work, an aim to model IXP425 (Intel network processor) which is core-centric when it is performing VPN. The XScale core processor of IXP425 has the packet processing responsibility, yet the coprocessors execute receiving, transmission, and cryptographic operations [8].

2. Hardware architecture of IXP425

The IXP425 network processor is a highly integrated, versatile single-chip processor which can be used in a variety of products requiring network connectivity and high performance. It has a 533MHz XScale core processor which suites handling system initialization and software objects execution. Also, it has three 133MHz programmable coprocessors called network processor engines (NPEs) used to execute, in parallel, the code image stored in an internal memory. Each NPE has an ALU, self-contained internal data memory and an extensive list of I/O interfaces, together with hardware acceleration elements that target a set of networking applications providing functions like MAC filtering, CRC checking/generation, AAL2, AES, DES, SHA-1 and MD5.

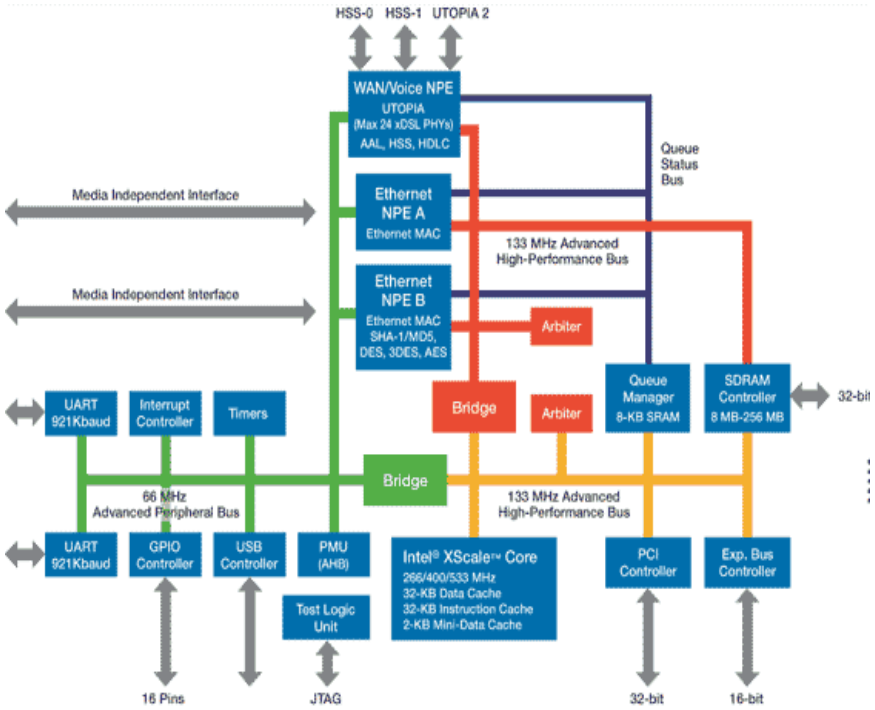


Figure (1) IXP425 network processor block diagram

In addition to NPEs, IXP425 has application-specific coprocessor which is used for encryption/decryption operation. The core, coprocessors, and other components are connected by three buses interconnected by two bridges as in figure (1) [9, 10, 6]. The IPX425 supports hardware multithreading with a single cycle context switch and that helps NPEs to reduce memory accesses time with an ideal state number. The XScale core and coprocessors communicate through hardware queue manager using interrupt and message queue mechanisms. The queue manager contains 8KB SRAM divided into 64 independent queues manipulated as circular buffers for allocating free memory space to incoming packets and for locating packets in the memory. The SDRAM can be expanded up to 256MB for storing tables, policies and OS applications in addition to packets [8].

3. Detailed packet flow in IXP425

When a packet arrives at the interface of an NPE, it is divided into a number of partitions of size 32 byte and stored at the Receive FIFO of an Ethernet coprocessor which in turn performs MAC-related operations. The queue manager of the NPE allocates those parts in corresponding addresses in SDRAM then interrupts the XScale of the reception for further processing. When the IXP425 executes procedures like IP and other higher layer protocol stacks, there is a possibility that authentic and cryptographic operations are needed to complete such procedures. The core can execute those functions or offloading the computation overhead to appropriate coprocessors (NPE).

4. VPN overview

Virtual Private Network (VPN) is used to get secure transmission over un-trusted networks. In the normal VPN, the IPSec protocol is adopted as the underlying technique because of the popularity of the Internet Protocol. VPN supports data authentication, integrity

and confidentiality, in which two gateways are employed as endpoints constructing a VPN tunnel for secure data transmission. Improving the performance of the gateways is decisive to the VPN throughput [11]. VPN process can be divided into five main stages (tasks). Three of them can be done by the coprocessors, yet the rest are executed by the core itself. The VPN five_stages are (1) receiving, (2) IPSec pre-processing, (3) en-/decryption, (4) IP processing, and, finally, (5) transmission. Figure (2) illustrates those tasks and which part executes each of them.

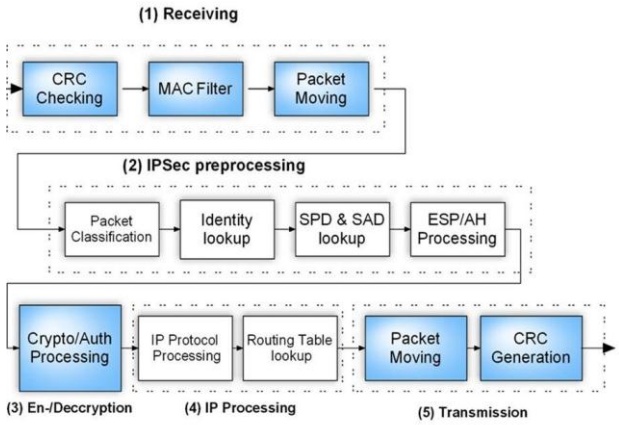


Figure (2) The five VPN stages

The tasks 1, 3 and 5, are offloaded to corresponding coprocessors, namely, the receiving coprocessor, computational coprocessor, and transmission coprocessor, respectively, whereas tasks 2 and 4 are handled by the core through multiprogramming and context switching.

4.1 Architectural assumptions

From figure (3) it can be understood that the core has the responsibility of both IPSec pre-processing and IP Processing; therefore, the core must have a mechanism to handle those tasks in parallel. Despite of the fact that many network processors can run hardware multithreading to reduce memory access latency through changing the running process with another when memory access is needed. In general, hardware multithreading requires duplicating buffers and registers, and that means increase in fabrication cost. This technique is efficient only when memory-access intensive applications, such as DiffServ and IDP are executed. Thus, a single thread is considered in each coprocessor because VPN is the application modeled in this work. Buffer space for each processing stage is also encompassed, except for the BW (Busy-Waiting) scheme, which needs no buffer between the core and the computational coprocessor.

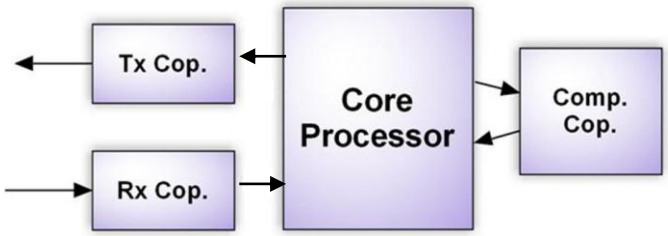


Figure (3) Processing flow and task allocation of the VPN application over IXP425 (logical view)

4.1.1 The busy-waiting scheme

Busy-Waiting (BW) is the first way to model the VPN. BW does not need any buffer between the core and computational coprocessor. That means, the core must wait until the computational coprocessor complete its work before resuming execution. In other words, after finishing IPSec preprocessing by the core, the result is passed (offloaded) to the computational coprocessor for en-/decryption. At the end of en-/decryption phase, the core

receives the result for IP processing. From the above description, the core and the computational coprocessor can be seen as different processes in a logical CORE processor, since only one of them can be active at any time. The BW scheme can be described as a series of queues consisting of three servicing stations, as shown in figure (4) [11].

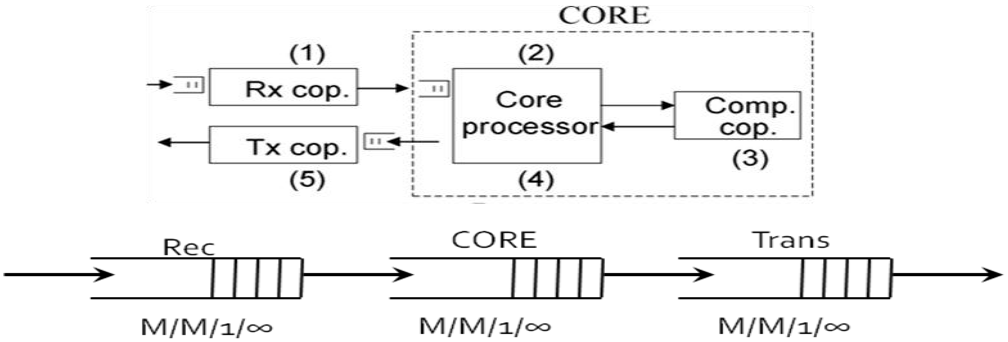


Figure (4) Modeling BW using queue

Each queue in the BW is M/M/1/∞ type, and the interval rate of each one is equal to the departure rate of the previous one except the Rec queue where its rate is equal to the input rate. The CORE utilization can be calculated using the following equation:

$$U_{CORE} = \frac{\mu R}{(T_{Core\_A} + T_{Cop} + T_{Core\_B})^{-1'}} \quad \text{---------- (1)}$$

Where μR denotes the mean arrival rate at the Core which is the same with the mean departure rate from the receiving coprocessor. TCoreA, Tcop and TCoreB represent the processing time for IPSec preprocessing, en-/decryption and IP processing, respectively. Finally, we can obtain the utilizations for core and computational coprocessor as:

$$U_{Core} = U_{CORE} x \frac{T_{Core\_A} + T_{Core\_B}}{T_{Core\_A} + T_{Cop} + T_{Core\_B}} \quad \text{---------- (2)}$$

$$U_{Cop} = U_{CORE} - U_{Core} \quad \text{---------- (3)}$$

This is because CORE contains both the core processor and computational coprocessor.

4.1.2 The interrupt-driven scheme

The second way to module VPN is the Interrupt-Driven (ID). The core does not need to wait the computational coprocessor to complete its current task. Instead the core passes the results from IPSec preprocessing to the computational coprocessor and resumes without being blocked. That means the core has to run IPSec preprocessing and IP processing in parallel; therefore, a buffer is required between the core and coprocessor. In other words, after the completion of IPSec preprocessing, the packet is passed to the coprocessor's buffer, the core switches, with a switching delay TD, to the other process for performing IP-related operations so that the core is not stalled. The switching does not happen after only finishing

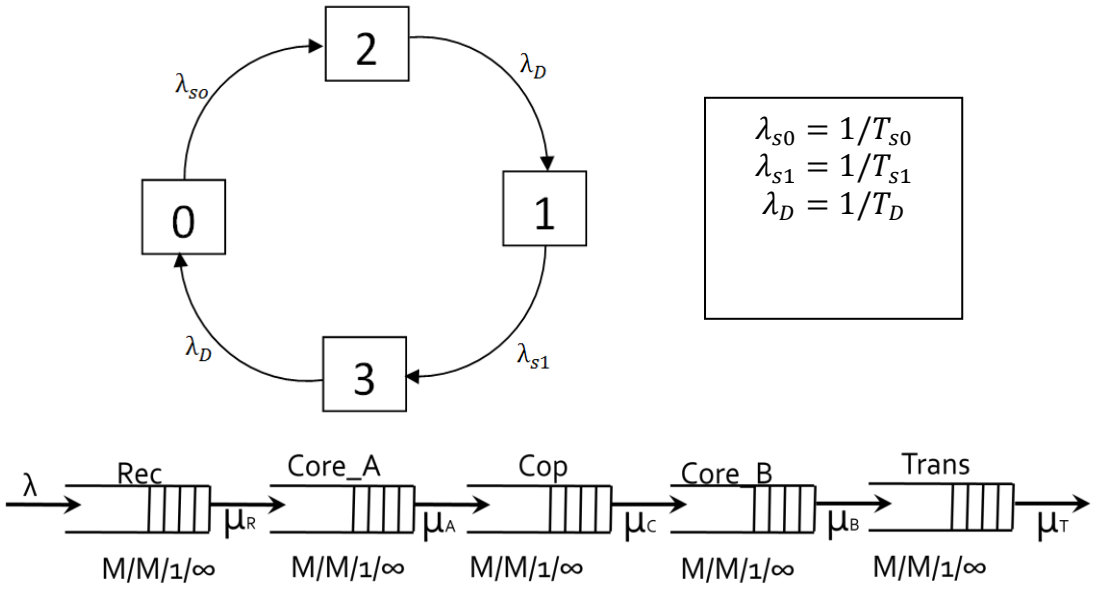
IPSec preprocessing but after a certain period of process run length. According to the above descriptions, the ID can be described using Markov chain, where the system performance is divided into six states, five of them are R, A, C, B and T for the five processing stages (Receiving, IPSec preprocessing referred to as CoreA, en-/de-cryption referred to as Cop, IP processing referred to as Core B, and Transmission) while the sixth S denotes the process switching. As shown in figure (5), S = 0/S = 1 means the core is processing packets at Core A/Core B; it then switches to an intermediate state S = 2/S = 3 after a run length of TS0/TS1, and waits for TD to finish the context switch. Using multi-process mechanism does not mean that the core will be in an execution state all the time and there is no wait state. The core will be in the wait state for the following situations:

1- Packet arrivals from its predecessor.
2- Available buffer slots in its successor to which the processing result is passed.

From the above descriptions, the effective core utilization can be derived by subtracting the busy-waiting overhead from overall core utilization. However, to achieve this in the simulation, the following steps must be considered.

(1) Setting appropriate run lengths TS0 and TS1 so that the processing resource is reasonably distributed.
(2) Characterizing appropriate transition conditions so as to ensure that context switches are performed upon those conditions [11].

$$ST = (R, A, C, B, T, S)$$



*	Λ	The mean of the packet arrival rate.
*	T_{s0}	The mean run length of the packet at Core A.
*	T_{s1}	The mean run length of the packet at Core B.
*	λ_{s0}	The mean switching rate of core from 0 to 1. λs0 = 1/TS0.
*	λ_{s1}	The mean switching rate of core from 1 to 0. λs1 = 1/TS1.
*	T_D	The mean context switch delay.
*	λ_D	1/TD
*	μ_X	The mean service rate of processing stage X .
*	$buff_X$	The buffer size of processing stage X .

Figure (5) The interrupt driven model

All possible transitions necessary to perform packet processing are listed in Table I and explained as follows: The core switches form 0/1 means CoreA/CoreB or reverse if the following conditions are met:

(1) The run length time has been reached (TS0 for CoreA and TS1 for CoreB).
(2) The corresponding active stage (either CoreA or CoreB in which the PCS resides) does not have any packet but the other does.

At the same time switching will not occur for the following conditions:

(1) The target stage to which PCS is going to switch has no packets.
(2) The successor of the target stage has no buffer left.
Whenever the switching happens from 0 to 1 or from 1 to 0 the delay TD must be added.

Table I State transitions in the ID scheme [10]

Component	Transition Type	Condition(s)	Rate
Receiving	Arrival	R < bufR	λ
	Departure	A < bufA	**μR**
Core A	Arrival	follower of Rec	n/a
	Departure	PCS=0, C < bufC	μA
Cop	Arrival	follower of Core A	n/a
	Departure	B < bufB	μC
Core B	Arrival	follower of Cop	n/a
	Departure	PCS = 1, T < bufT	μB
Transmission	Arrival	follower of Core B	n/a
	Departure	n/a	μT
PCS	0 => 2	A > 0, B > 0	λS1
		(A = 0, B > 0) or C =bufC	∞
		(B=0) or T =bufT	0
	2 => 1	n/a	λD
	1 => 3	A>0, B>0	λS2
		(A >0, B = 0) or T =bufT	∞
		(A=0) or C =bufC	0
	3 => 0	n/a	λD

Where:

λ: The mean of the Packet Arrival rate.
μX: The mean service rate of processing stage X.
bufX : The Buffer size of processing stage X.
PCS: Packet Context Switch.
λX : The mean Arrival rate of processing stage X.

Figure (6) illustrates a state example whose buffer size equals to 3 for each stage and the corresponding transitions in which are five transiting into the state and others are transiting out from the state. While the inbound and outbound transitions are comprehensible, special note needs to be considered for the PCS, namely the PCS in the example state never transits from 0 to 1. This is because the Core B cannot pass the processed packet to the transmission stage, which is already full. The transition of PCS from 0 to 1 simply

contributes to the unnecessary overhead and, therefore, is considered an invalid transition [11].

(R, A, C, B, T, S)

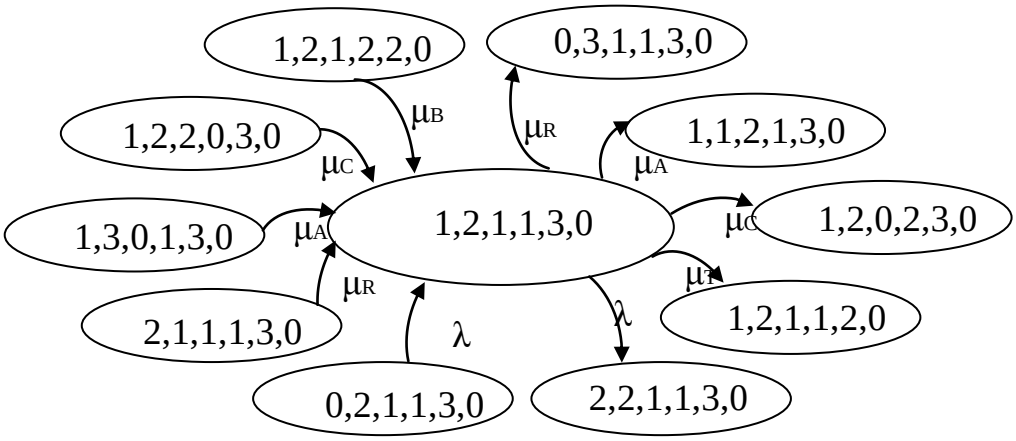


Figure (6) State transitions example of the interrupt-driven model

4.2 BW experiment using Omnet++

As mentioned in the previous section, the BW modeling can be described as one block. This block represents CorA, Cop, And CorB. That means the Cor is not only doing the core works (IPSec preprocessing and IP Processing) but also the coprocessor work (en/de-cryption). For this project, the exponential distribution is considered for generating the packets in the gen block to compare our result with the work in [11]. A variety of values used as a mean value of this function so the system can be tested under different load values and also different run time durations. The queues used are of type M/M/1/100. That means the maximum number of the packets which can be stored in it is 100 otherwise the packet will be discarded by the queue due to unavailability of spacing. Both the Receiver and Transmitter blocks work as delayer with values taken from the IXP425 datasheets. Whenever any of them finishes its job the completed packet is sent to the next fifo and a request signal is sent to the previous buffer. Thus, the previous fifo will send the oldest packet in its queue. The sink block is used for statistic operations. Thus, it collects the packet out from the transmitter block and calculates the time. When the Cor is in the idle state, for example the NP just work, the Cor sends a request signal to previous fifo asking for a packet. When a new packet arrives, it passes three processing stages (IPSec preprocessing, en/de-cryption and IP Processing). This can be represented as three delay blocks. The delay time for those blocks is also taken from the IXP425 datasheets. During the processing time of this packet, the Cor will be in the busy state and it can process any other packet. Figure (7) illustrates the BW scheme.

$$The\,core\,utilize = \frac{T_{Cor} - T_{EnDee}}{T_{\exp}} \qquad \text{---------- (4)}$$

Where the total working time of the Core block is T_{Cor}, T_{EnDe} is the working time of the Encryption/Decryption coprocessor, $T_{\exp}$ is the total experiment time.

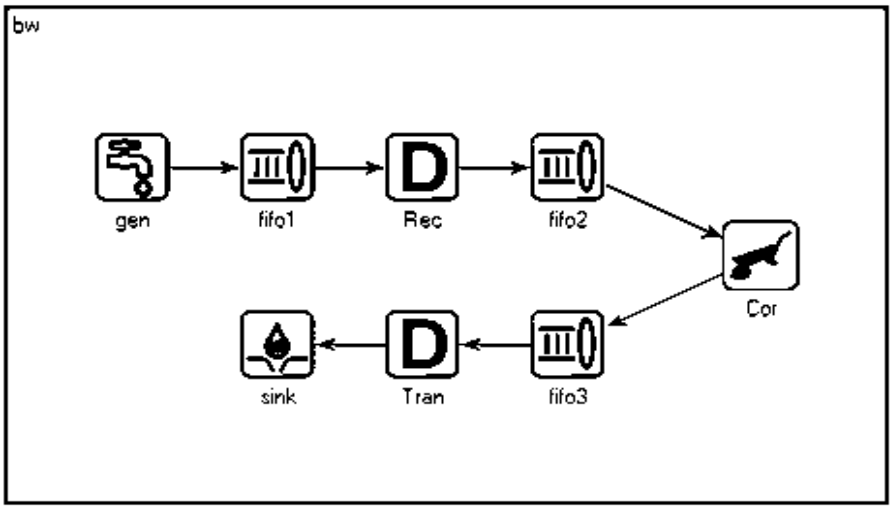


Figure (7) BW Scheme used in this work

Table II processing Time of the Tasks Evaluated in Real Network Processor [4]

Task	Processing Time (µs/packet)
Receiving (Rec)	27.3
IPSec Preprocessing (CorA)	32
En-/Decryption	12.6
IP processing (CorB)	49
Transmission (Trans)	27.3

4.3 ID experiment using Omnet++

As figure (8) illustrates, ID scheme which is similar to the BW scheme except the Cor part. Here the Cor is divided into CorA, EnDe, and CorB. The receiving and Transmission blocks do the same job that they did in the BW scheme. The CorA and CorB represent the core so that they cannot work at the same time. The time is divided between them under supervision of the switch as the following. At the beginning, the switch send No_Token message to both of them and then enter a waiting for request mode. This mode ends when either CorA or CorB send Request_Token message. The switch gives the token to the first request core (First Request First Served). The token that is given by the switch has a time expire (Ts). The token will go back to the switch in these situations: (1) there is a request from the other core and the time of the token reaches the limit, (2) the core completes the process of the packet so it does not need the token. The switch then resends the token again. The CorA/B has the same job. They start with idle state. In this state, the core sends a request signal to the previous queue every system clock. When a packet arrives, the core enters the request for token mode. A request massage is sent to the switch every system clock. When the token is received, the core enters working mode. This mode is interrupted when the switch takes the token. The core enters request for token mode. The core continuously requests for a token until it is received thus it returns to the working mode. This cycle continues until the end of the packet processing. The last block in the ID is the EnDe. The EnDe works as delayer. It is first continuously sends a request for a packet from the previous fifo signal. When it receives a packet, it enters the busy mode which ends after a certain period of time which is taken from table II. At the end of this period, the packet is sent to the next fifo by the EnDe.

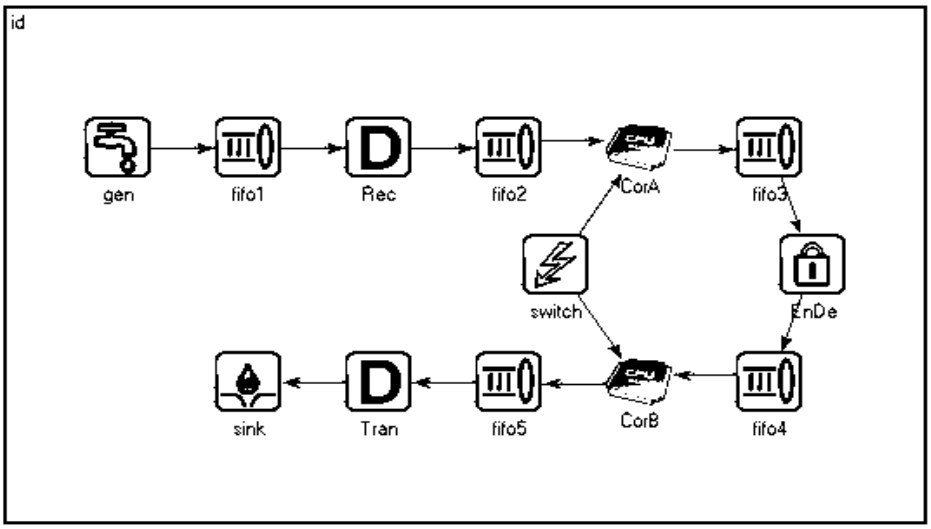


Figure (8) ID Scheme used in this work

$$The\,core\,utilize = \frac{T_{CorA} - T_{CorB}}{T_{\exp}} \quad \text{-- ---------- (5)}$$

Where T_{CorA} and T_{CorB} are the total working time for the core, while $T_{\exp}$ is the total time of the experiment.

5 Results and discussions

The packet generator used a random variant from the exponential distribution with mean value changed from (0.07ms to 0.1ms) to generate the testing packets. That changed in main values will vary the packets generation rate from 14.3k packets/sec to 10k packets/sec. Our main aim is to find out the following the number of the processed packets, lost packets, and the three main processors (transmitter coprocessor, Core, receiver coprocessor). Another interested result we are looking for is queues size for our mathematical model.

We choose 500ms to evaluate the two systems (ID and BW). This time is enough for the test because it is much bigger than the total time required to process one packet through both systems.

The total number of the processed packets though the two systems are illustrated in Figure (9). The mean time of the packet arrival using the exponential distribution is varied from 0.07 to 0.1ms with a step equal to 0.005ms. It is obvious from Figure (9) that the ID system has superior performance over BW.

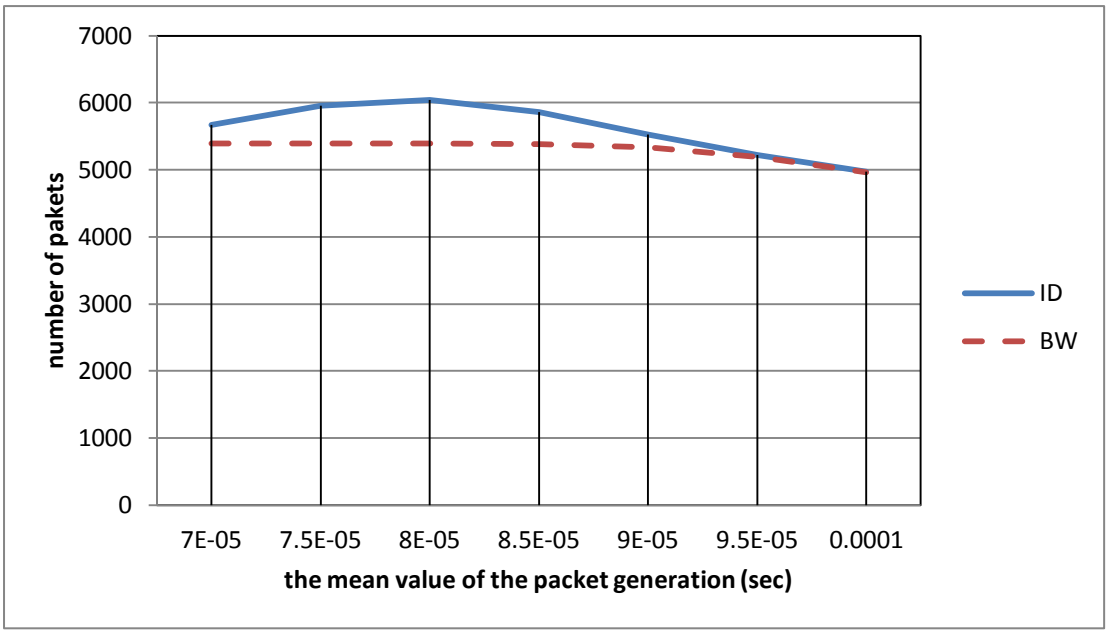


Figure (9) The number of the output packets during system running for 0.5 sec.

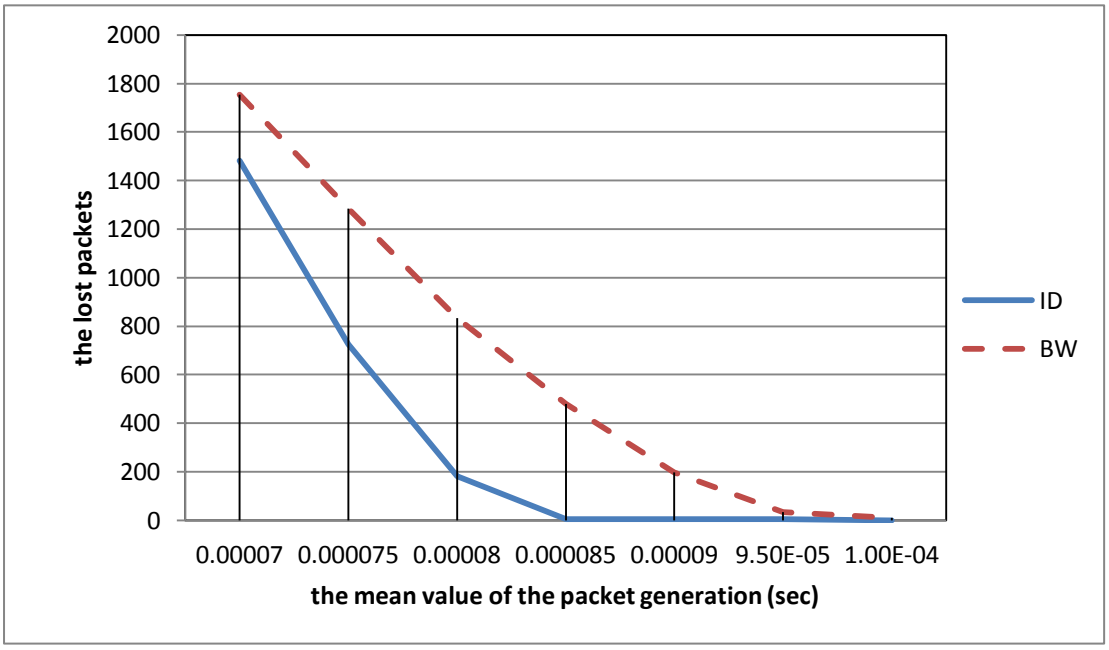


Figure (10) The number of the lost packets

The number of the lost packets due to the over arrival are illustrates in Figure (10). It is normal that the lost packets are increased by decreasing the mean value of the arriving packets. The BW has lower capacity to accommodate the high arrival packets (packets/sec).

The maximum number of packets in each of the three queues of the BW scheme illustrates in Figure (11). Due to the long processing time/packet of the Core, Fifo(2) has the highest number. The queue implementation has the maximum size of 100 packets. Because of this limitation some of the arriving packets discarded due to the overflow. On the other hands, fifo (1) has four as the highest number, yet fifo (3) is always empty. Therefore, fifo (3) can be removed from the scheme, and the size of fifo (2) can be reduced to four.

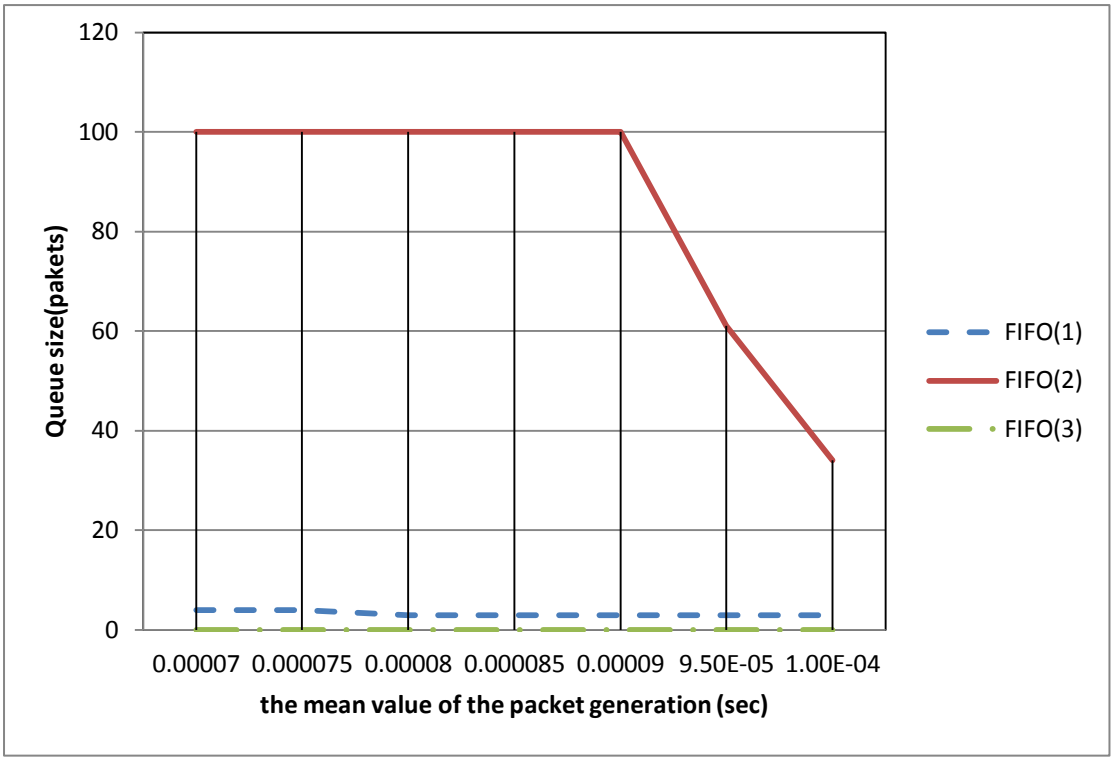


Figure (11) The maximum number of the packets in the BW scheme

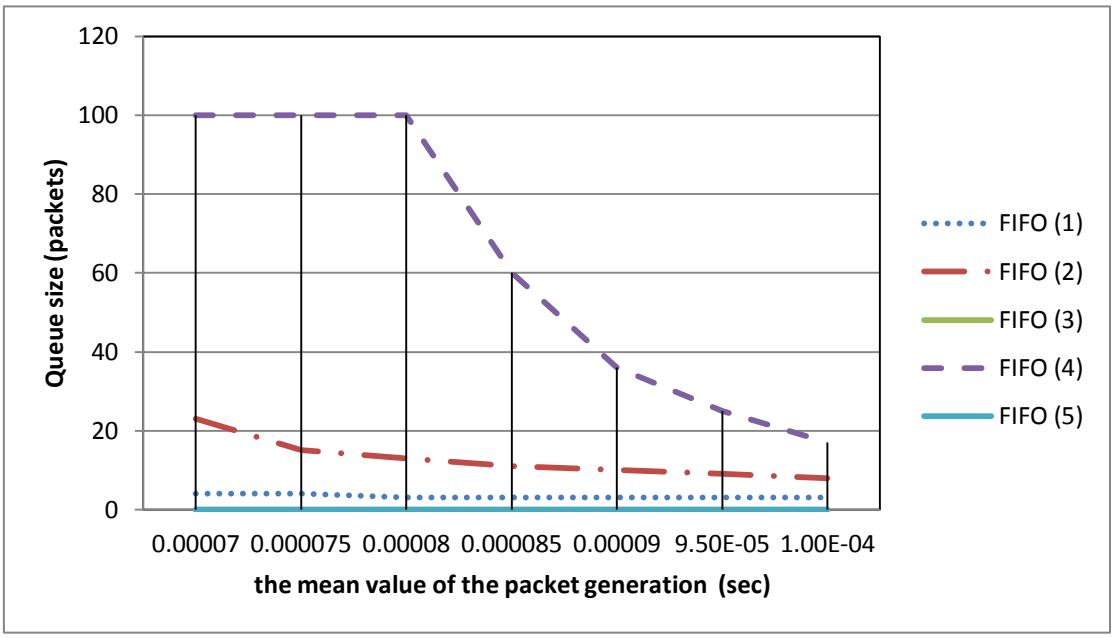


Figure (12) The maximum number of the packets in the ID scheme

Figure (12) illustrates the maximum number of the packets in each queue of the ID scheme. The fifo(4) has the highest number of packets, because of the long processing time of CorB. On the other hand, fifo (2) has 23 as the highest number,
and fifo (1) has 4 as the highest number, yet fifo (3) and fifo (5) are always empty. Therefore, fifo (3) and fifo (5) can be removed from the scheme, and the size of fifo (2) and fifo (1) can be adjusted.

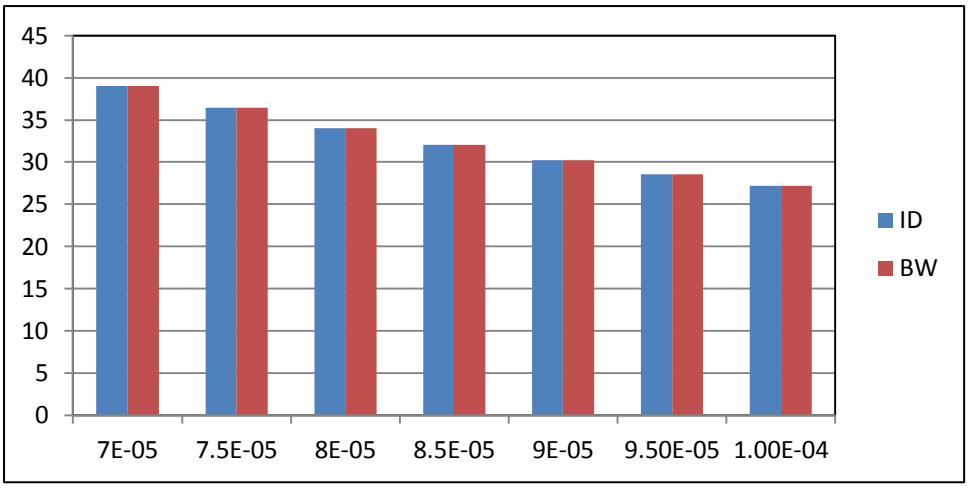


Figure (13) The receiver coprocessor percentage utilization

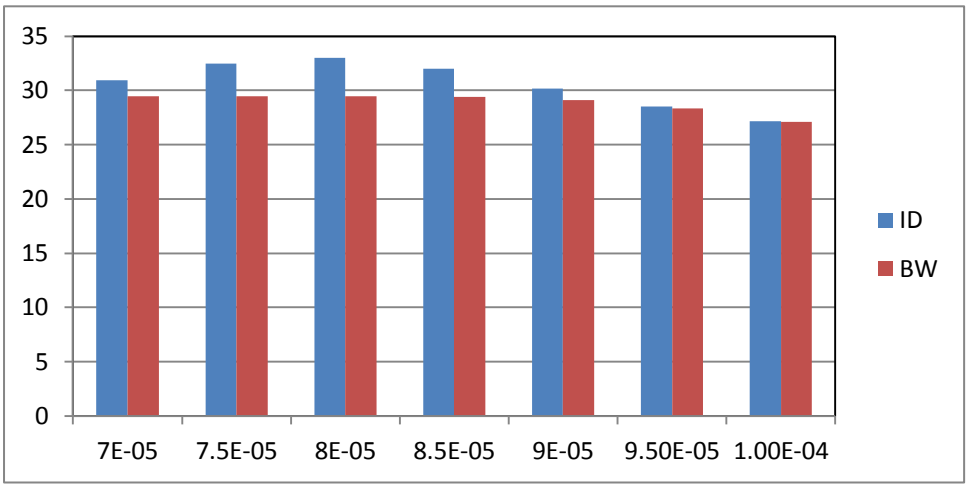


Figure (14) The transmitter coprocessor percentage utilization

Figure (13) and (14) demonstrate the receiver and sender coprocessor utilization. The receiver coprocessor is 60% of the time free and in the idle state, while the sender is 65% of the time free for both ID and BW. Therefore, both require increasing the utilization.

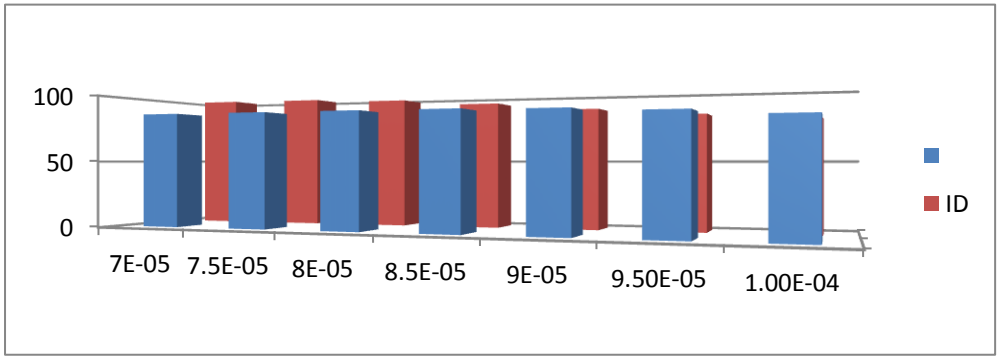


Figure (15) The core processor percentage utilization

Figure (15) demonstrates the Core utilization and shows that the core is 99.8% utilized in the ID , but just 86.3% for the BW. This means, the Core has more working time in the ID than in the BW which leads to increasing the system throughput and decreasing of the packets lost number.

6. Conclusions and future work

Since the paper of reference number (11) represents an elementary work in this area a comparison with its results is thought to be useful**.** The run length in this paper is set to 0.5 seconds while in reference (11) it is between 0.00005 and 0.006666 seconds and set to 0.0001 or 0.0002 seconds. On the other hand, the load in this paper is (10000 to 14300 packets per second) compared to (3 to 1000 packets per second) in reference number (11) as shown in its figure page (41:13).

- The Core can be fully utilized using ID and that is eventually better than BW system for high rate networks (Refer to Figure 9 and 10). BW is easier than ID for implementation and works fine for an average rate network. As a comparison, reference number (11) recommends BW rather than ID contrary to this paper but for lower loads.

- Figure (11 and 12) give good hint about the size of the buffers for the designer, which actually depends on the input packet rate. However, this is a different outcome compared to reference number (11) where the change of buffer size from 3 to 1000 pkts has shown no effect on the performance as figure (11) page (41:13) confirms.

- The ID has a total delay time for the packets higher than the BW, and that is because of the core time division between the processing of two packets in different stages.

As a future work, other applications can be modeled and tested. Another Network Processor can, for example, be modeled and a comparison can be driven from this modeling. A memory-access intensive application such as IDP (Intrusion Detection and Prevention) can be modeled so that the memory access delay time can be tested and also several measures can be examined to reduce it and give a good glow about the best method to adopt. Moreover, other interested implementation would be to implement general simulation programs where number of stages and the processing time of each stage can be entered in the profile to model multi stages network processor. Implementing a simulation system of a multi thread network process to figure out the best scheduling techniques, buffer size of each thread, and the optimum controller schedule is one of the current interesting problem.

References

[1] Braun, T., G¨unter, M., Kasumi, M., and Khalil, I. Virtual Private Network Architecture. Technical Report. IAM-99-001, CATI, 1999.

[2] Clark, C. at AL., A Hardware Platform for Network Intrusion Detection and Prevention. In Proceedings of the 3rd Workshop on Network Processors and Applications (NP3). Madrid, Spain, 2004.

[3] Comer, D. and Martynov, M., Building Experimental Virtual Routers With Network Processors. In Proceedings of the 2nd International Conference on Test beds and Research Infrastructures for the Development of Networks and Communities (TRIDENTCOM). 2006.

[4] Lin, Y.-N., Lin, C.-H., Lin , Y.-D., and Lai, Y.-C., VPN Gateways Over Network Processors: Implementation and Evaluation. In Proceedings of the 11th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'05). San- Francisco, CA. 2005

[5] Lin, Y.-D., LIN, Y.-N., Yang, S.-C., and Lin, Y.-S.,. DiffServ Edge Routers Over Network Processors: Implementation and evaluation. IEEE Netw. Vol. 17, No. 4, pp 28–34., 2003

[6] LU, J. and Wang, J., Analytical Performance Analysis of Network-Processor-Based Application Designs. In Proceedings of the 15th International Conference on Computer Communications and Networks (IC3N'06). IEEE Press, Los Alamitos, CA. pp 33–39, 2006.

[7] Wolf, T. And Franklin, M. K., Performance Models for Network Processor Design. IEEE Trans. Parallel Distribution System Vol. 17, No. 6, pp 548–561, 2006..

[8] Lekkas, P. C., Network Processors: Architectures, Protocols and Platforms (Telecom Engineering), McGraw-Hill, New York, 2003..

[9] Intel IXP425 Network Processor.
http://www.intel.com/design/network/products/npfamily/ixp425.htm.

[10] Intel IXP2400 Network Processor.
http://www.intel.com /design/network/products npfamily/ixp2400.htm.

[11] LIN , Y. , LIN, Y. , LAI, Y. , and TSENG, K., Modeling and Analysis of Core-Centric Network Processors. ACM Transactions on Embedded Computing Systems, Vol. 7, No. 4, Article 41, 2008.