Packet Reordering in Network Processors

S. Govind[1], **R. Govindarajan**[1,2] and **Joy Kuri**[3]
[1]Supercomputer Education and Research Centre
[2]Dept. of Computer Science and Automation
[3]Centre for Electronics Design and Technology
Indian Institute of Science
Bangalore 560012, India
{sgovind@hpc.serc, govind@serc, kuri@cedt}.iisc.ernet.in

Abstract

Network processors today consists of multiple parallel processors (microengines) with support for multiple threads to exploit packet level parallelism inherent in network workloads. With such concurrency, packet ordering at the output of the network processor cannot be guaranteed. This paper studies the effect of concurrency in network processors on packet ordering. We use a validated Petri net model of a commercial network processor, Intel IXP 2400, to determine the extent of packet reordering for IPv4 forwarding application. Our study indicates that in addition to the parallel processing in the network processor, the allocation scheme for the transmit buffer also adversely impacts packet ordering. In particular, our results reveal that these packet reordering results in a packet retransmission rate of up to 61%. We explore different transmit buffer allocation schemes namely, contiguous, strided, local, and global which reduces the packet retransmission to 24%. We propose an alternative scheme, Packet Sort, which guarantees complete packet ordering while achieving a throughput of 2.5 Gbps. Further, Packetsort outperforms the in-built packet ordering schemes in the IXP processor by up to 35%.

1 Introduction

In recent years there has been an exponential growth in Internet traffic leading to increasing network bandwidth requirements which has resulted in stringent processing requirements on network layer devices like routers. Present backbone routers on OC 48 links (2.5Gbps) have to process four million minimum-sized packets per second. In addition, the processing requirements on the routers have increased significantly, with a need to support encryption/decryption of data, intelligent routing, and quality of service guarantees at line rates. As a consequence, application specific processors in routers (network processors) have become the core element in design of routers.

1-4244-0910-1/07/$20.00 ©2007 IEEE.

Commercial network processors employ multiple parallel processors (microengines) to exploit packet level parallelism inherent in network workloads. This enables routers to support OC 48 line rates [6]. Further, each microengine supports multithreading and hence processes multiple packets to achieve efficient utilization of resources. Since packets can get allocated to threads in different microengines, packet order at the output of the network processor cannot be guaranteed. Certain network processors have in-built schemes to prevent reordering. For example the IXP series of processors use Inter Thread Signaling (ITS) and Asynchronous Insert Synchronous Remove (AISR) to prevent reordering. However, earlier works on performance evaluation of NPs do not analyze the impact of these schemes on the NP throughput. Similarly earlier work on packet reordering in routers [3] [10] [12], do not consider the impact of the network processor architecture on packet reordering/retransmission.

This paper analyzes the impact of NP architecture on packet reordering. Our results indicate that ITS and AISR can cause a significant degradation in the throughput (2.3 Gbps for ITS and 1.1 Gbps for AISR). Hence, we explore different ways to reduce reordering in IXP processors. We use a validated Petri net model of the Intel IXP 2400 processor running the IPv4 [1] forwarding application. The Petri net model captures the packet flow from *end-to-end* , i.e., from receive FIFO to transmit FIFO. This truly models the interaction between RFIFO/TFIFO and DRAM, unlike many of the earlier studies which assume packets to be resident in DRAM [17] [4]. A salient feature of the model is its ability to model the processor, application and their interaction in sufficient detail. The Petri net model is validated using the Intel SDK 3.51 for IXP 2400 architecture [8].

The Petri net model is then extended for multiple hops. The reordering and retransmission are measured as the number of duplicate acknowledgments sent by the destination. We report results for 1, 5, and 10 hops. We observe that the reordering/retransmission increases non-linearly with the number of hops. Our results indicate that the parallel architecture of the network processor can severely impact reordering and can cause up to 61% retransmission in a 10 hop scenario. We observe that in addition to reordering due to parallel processing, transmit buffer allocation for each thread in a microengine severely impacts packet reordering. In order to reduce the packet reordering, we explore different buffer allocation schemes, namely, global, local, and strided buffer allocation. These schemes involve different amount of synchronization and hence incur a performance penalty of varying degree. We evaluate the trade off between throughput and retransmission for various buffer allocation schemes for different packet sizes using out Petri net model.

Further, our results indicate that packet reordering reduces for a network processor with fewer number of microengines without sacrificing the performance (2.96 Gbps). This is because the throughput of the network processor saturates beyond a total of 16 threads [6]. Based on this observation we propose a scheme, packet sort, in which a few microengines/threads are dedicated to sort the packets in-order at the transmit buffer side. Packet sort is able to support a line rate of up to 2.5 Gbps without any packet reordering. Our results indicate that Packet sort achieves a significant throughput improvement, of up to 35% over the in-built schemes in the IXP, namely, ITS and AISR.

In the following section we provide an overview of the IXP architecture. Section 3 describes packet reordering in the IXP architecture. In Section 4, we present different ways to reduce packet reordering and evaluates them. Section 5 reviews the related work in this domain and Section 6 provides concluding remarks.

2 Background

The architecture of IXP 2400 processor consists of an Xscale core, eight microengines, external memory (DRAM and SRAM), and application specific hardware units like hash and crypto unit [7]. The Xscale is a 32 bit RISC processor, used to handle control and management plane functions [16] (e.g., routing table update) and to load the microengine instructions. IXP 2400 contains eight 32 bit microengines each running at 600 MHz. Each microengine contains eight hardware contexts, for a total of 64 threads. There is no context switch overhead. There are 256 programmable general purpose registers in each microengine equally shared between the eight threads. The IXP processor provides supports for off-chip SRAM and DRAM. The DRAM is used for packet buffering and SRAM stores state table information like routing table. The IXP chip has a pair of FIFOs, Receive FIFO and Transmit FIFO, used to send/receive packets to/from the network ports, each of size 8 KB. A dedicated data path exists between the microengines and DRAMs to the FIFOs for fast data transfer.

Next we describe packet flow in IXP processors. Packets arrive from the external link to the input ports and get buffered at the input port buffers. Packets are then transferred to the Receive FIFO (RFIFO) of the network processor through a high speed media interface. When a thread in a microengine is available, it takes control of the packet and transfers the packet from the RFIFO to the DRAM. The packet/packet header is read from the DRAM by the corresponding thread. The thread processes the packet header, performs an address lookup for the next hop, and writes back the new packet header to the DRAM. Next the thread places the packet in the Transmit FIFO of the network processor and writes the packet to the corresponding output port buffer through the media interface. The thread is then freed and the packet is forwarded to the next hop through the external link.

3 Packet Reordering

When packets belonging to a single flow, having the same source and destination IP address and port number, arrive at the destination in an order different from the sequence order, we say that the packets are reordered. Packet reordering is a well known phenomenon in the Internet [2] [3]. Studies on backbone traffic measurement [5] suggest that TCP accounts for 80% of the Internet traffic. When packets get reordered, the TCP receiver begins to generate duplicate ACKs. On receiving multiple duplicate ACKs (typically 3), the TCP sender concludes that packet drops have occurred due to congestion. The Congestion Avoidance algorithm [14] now kicks in. According to the Congestion Avoidance and Fast Retransmit algorithms [14], the sender retransmits the packets for which it has not received ACK. Further, it halves the transmit window size. Thus the effect of reordering is not only the retransmission of packets that are already transmitted but also an unnecessary reduction of the sender's congestion window leading to under-utilization of the network resources.

3.1 Reordering in Network Processors

In this section we explain NP induced packet reordering. Consider the following scenario shown in Figure 1. Packets P1, P2, P3, P4 of the same flow arrive at the receive buffer (RBUF) of the network processor in order. Packets are allocated to threads in different microengines in the

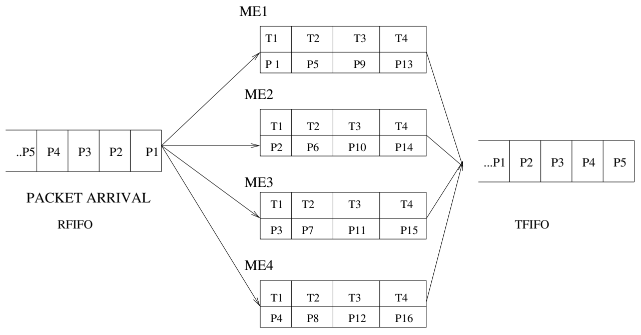


Figure 1. Packet Reordering in Network Processors.

following way : P1, P2, P3, P4 are allocated to ME1-T1 (Microengine1-Thread1), ME2-T1, ME3-T1 and ME4-T1 respectively. Now packet P1, being processed by ME1-T1, can get delayed with respect to P2, P3, P4. This can happen due to various reasons, e.g., processing of other threads, or pending memory requests in DRAM FIFO. So the thread ME1-T1 completes the processing of P1 only after ME2-T1, ME3-T1, and ME4-T1 have processed their respective packets. So packet P1 is delayed with respect to P2, P3, P4 and is transmitted only after P2, P3, P4 have been forwarded. This may result in a retransmission of P1. This example explains how the concurrent processing of packets can affect the ordering of packets. Note that multiple microengines is a common feature of the network processor and a network processor such as the IXP 2400 [7] has 64 threads and hence can process up to a total of 64 packets.

3.2 Transmit Buffer Induced Reordering

In this section we explain how packet order can be affected by the transmit buffer in the IXP architecture. Transmit buffer is a shared resource in the IXP architecture. So all the threads compete for a common transmit buffer space. Hence, to ensure proper access of the transmit buffer, all threads should execute mutual exclusion operation. This, as reported in Section 3.6, results in a significant drop in the throughput (62% drop in the transmit rate). Hence transmit buffer locations are allocated a priori to different threads. However, the transmit buffer dequeues packets in a strict FIFO order. This aggravates packet reordering as illustrated in the following example.

We consider a contiguous buffer allocation where different threads in different microengines are allocated contiguous space in the transmit buffer. More specifically, we will assume that ME1-T1 is allocated the first 64 bytes, ME1-T2 is allocated the next 64 bytes and so on (refer to Figure 2). Packets P1, P2, P3, P4 from flow F1 arrive strictly in that order in the receive buffer. Assume that P1, P2, P3, P4 are

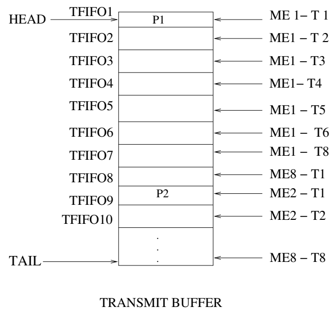


Figure 2. Transmit Buffer Reordering.

allocated to ME1-T1, ME2-T1, ME3-T1, and ME4-T1 respectively. After processing by different microengines, the packets P1, P2, P3, and P4 are stored in TBUF1, TBUF9, TBUF17 and TBUF 25 respectively. However, as mentioned earlier, packets are dequeued in a strict order of the transmit buffer location. Thus, before P2 is dequeued from TBUF9 location, other packets from TBUF2-TBUF8 will be dequeued. If packets from the same flow as P2 are allocated to threads in microengine 1, they will get forwarded before P2, causing the packet reordering problem. Hence, the transmit buffer can independently induce reordering. Note that in this example even if packets P1, P2, P3, and P4 complete in the same order, the dequeuing of packets from the transmit buffer causes reordering.

3.3 Packet Ordering Mechanisms in IXP

The IXP processor supports the following two mechanisms to maintain packet order in the network processor.

Inter Thread Signaling (ITS): In this mechanism threads in the IXP processor perform the start and finish the tasks of IPv4 forwarding sequentially. The packet processing functions are done in parallel and independently across all microengines. The sequential processing at the beginning and at the end of IPv4 ensures that packets are allocated in the transmit buffer and transmitted out in-order. So each thread is allocated a packet in sequential order. Assume that packets P1 and P2 arrive in the system in that order. Further, ME1-T1 is assigned P1 and ME1-T2 is assigned P2. This assignment occurs in a sequential order, across all the threads in the processor using Inter Thread Signaling (ITS). Each threads waits for a signal to start the sequential task, performs the allocation of packet or transmission of packet, and signals the neighboring thread. For example, ME1-T1 signals ME1-T2 and ME1-T8 signals ME2-T1.

Asynchronous Insert Synchronous Remove (AISR): In this scheme, the packet forwarding is divided into four stages namely, packet buffering stage, packet processing stage, reordering stage, and transmit stage. In the initial stage, namely the packet buffering stage, every packet is assigned a sequence number and buffered in the memory (DRAM). The sequence number is maintained for all the packets arriving in the system. The sequence number of a newly arriving packet in the system is one greater than the previous packet. After the packets are assigned sequence numbers, the packet processing stage processes packets independently and passes the packet handler to the next stage, the reordering stage. A counting sort of the packet handler is carried out by the reordering block to restore packet ordering. Here the packets are also assigned different transmit buffer addresses. This is passed on to the last stage, the transmit stage. The transmit block, moves the packet out of the DRAM to the network interfaces.

3.4 Petri Net Model for IXP 2400

We develop a Petri net model for IXP 2400. Here we briefly describe the Petri net model; a more detailed description of the model and the validation is given in [6].

3.4.1 Base Model

In this section, first, we describe a Petri net model of the IXP 2400 architecture running the IPv4 forwarding application [1] for a single hop (refer to figure 3). For clarity, only a part of the model that captures the flow of packets from the external link to DRAM through the MAC is shown. The firing time of a timed transition in our model takes either a deterministic or exponentially distributed values (over a mean).

The place *INPUT-LINE* represents the external link. Packets arrive at *IMAC*, the input MAC, from the external link at line speed. If an input port (*IPORT*) in the MAC is free and if there is sufficient MAC memory, i.e., at least a token in *IMAC*, the packet gets buffered in MAC. A token in *RMACMEM* indicates that a packet has been buffered in the MAC. If a thread is free, denoted by a token in *THREAD*, it takes control of the packet and transfers the packet to the receive buffer (*RFIFO*) in the IXP chip. The initial marking of place *THREAD* denotes the total number of threads in a microengine. If the microengine is free, represented by a token in *UE*, the thread executes for *UE-PROCESSING* amount of clock cycles, and moves the packet from *RFIFO* to *DRAM*. The thread swaps out, denoted by the arc from *SWAP-OUT* to *UE*, after initiating a memory transaction by placing the request for memory access in the microengine command queue (*UE-CMD-Q*). The availability of a free entry in the command queue is denoted by a token in the place *UE-CMD-Q*. The memory request is then moved from *UE-CMD-Q* to *DRAM-Q* through the command bus arbiter (*CMD-BUS*). The memory request gets processed by *DRAM* and a token is placed in *DRAM-XFER* indicating the

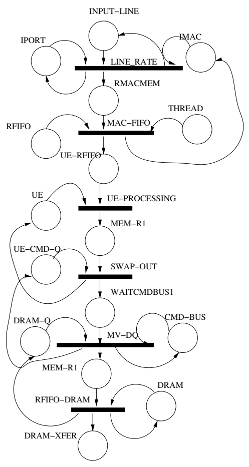


Figure 3. Petri Net Model for IPv4 on IXP2400

completion of the memory operation.

The places *UE*, *DRAM*, *CMD-BUS* represent conflicts, i.e., there can be two or more events competing for a common resource. Conflicts are resolved by assigning probabilities to the conflicting events. Our Petri net model assigns equal probabilities for accessing shared resources. Further, to model TCP flows, each token has been assigned a tuple consisting of a flow-id and packet-id. This tuple uniquely identifies each packet. This has been incorporated in CNET.

We use colored Petri nets for modeling multiple microengines. The processing done by each microengine (described in the earlier subsection) is represented by a color. The number of microengines is represented by tokens, of different colors, in the place *UE* .

3.4.2 Extensions to Multiple Hops

Earlier work [2] [10] on reordering in routers studied the reordering in a single router. This however ignores the impact of a multi hop scenario. Packet reordering induced

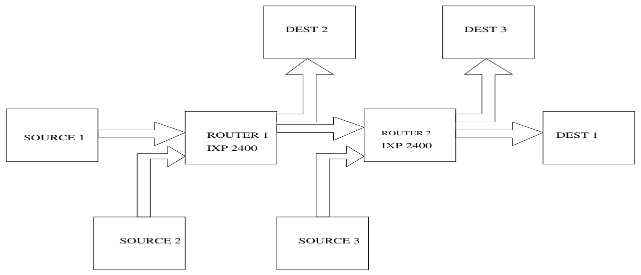


Figure 4. Simulated Network Topology.

by each router can cumulatively add up, leading to a significant degradation in the TCP throughput. Packets in the

Internet traverse, on an average, 16 hops to reach the destination [11]. We have simulated a network topology (depicted in Figure 4) with multiple routers. We assume that each router in the above topology uses IXP2400 to forward packets. The multi hop environment is incorporated by extending the IXP 2400 Petri net model, where the output of one router (one Petri net model) is given as input to the next router.

We measure packet reordering for one flow, between SOURCE 1 and DEST 1. Packets from other flows are used to simulate the network workload in the router. To reduce the complexity of the simulator and the time taken to simulate, we use the traffic going out of one router itself as the traffic from other sources/routers. This is reasonable since, in the steady state, the amount and characteristics of traffic leaving a router is similar to the traffic entering the next router. Hence, in our simulation, we model only multiple flows from a single source to destination through multiple routers; but we measure reorder/retransmit rates for 1 out of n flows (we use n = 10), leaving the other (n-1) flows to model the network traffic entering /exiting the router in the multiple hop. This models a real network scenario where each router forwards packets to different destinations.

3.5 Performance Metric

In our performance evaluation, we report reordering and retransmission rates as measures of packet reordering. Reordering is measured as the number of duplicate ACKs that will be sent by the destination back to the source. Retransmission corresponds to the number of retransmission packets where 3 or more duplicate ACKs cause a retransmission. Both reordering and retransmission are reported as a percentage of the total number of packets being transmitted. We use packet forwarding throughput (Gbps) as a measure of the network processor performance.

3.6 Simulation Methodology

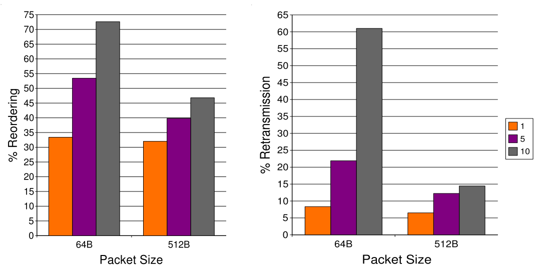
We have developed Petri net model for IPv4 running on IXP 2400. In order to take into account flow information used in determining packet sequence, each token is given two distinct attributes, a flow number and a sequence number. The Petri net model is simulated using CNET, a Petri net simulator [18]. We simulated up to 100,000 packets in each simulation. In order to validate the Petri net results, we have implemented IPv4 in MicroengineC [9], a high level programming language for Intel network processors and executed on SDK 3.51 [8]. The validation of the model is performed for different processor parameters. In this work, we validate the Petri net model of IXP 2400 for reordering and retransmission rates for a single hop for different flow sizes. We assume a contiguous buffer allocation in the validation. Further, we assume a packet size of 64 B and each flow to contain a fixed number of packets. Although we consider flow sizes of 640B, 6.4KB and 64KB, we use 6.4 KB as the default flow size which is also the the average flow size reported in the Internet [11].

We observe in our earlier work [6] that the IXP 2400 can support a maximum line-rate of 3 Gbps for IPv4 application. Hence we evaluate the performance of the IXP 2400 for a 3 Gbps line-rate. Note that this rate is higher than the maximum line rate currently supported (2.5 Gbps for OC-48) in routers. Further, we do not model the network flow of ACK packets (from destination to source). Nor do we assume rate reduction at source on retransmission.

Table 1 shows the comparison of reordering and retransmission rates obtained from the Petri net (CNET) and SDK simulations for a single hop. Further, the validation has been performed for different flow sizes. We observe that the reordering and retransmission rates obtained from the Petri net model closely match the SDK simulations for different flow sizes. A detailed description of validation of the Petri net model for different processor parameters can be found in [6] and is not described here due to space constraint.

3.7 Results

Figure 5 shows reordering and retransmission rates for different packet sizes. We observe that the reordering and retransmission rates increases with the number of hops, irrespective of the packet size. Further, for a 64 B packet size, the percentage of retransmitted packets is as high as 61% for 10 hops. However, for a packet size of 512 B, the average packet size in Internet [11], the reordering and retransmission rates are much lower (46% and 14% respectively). This



(a) Reordering (b) Retransmission

Figure 5. Packet Reordering in NP.

occurs as only 8K/512=16 packets can be buffered at the receive buffer at any given time in case of a 512 B packet size;

We observe in our earlier work [6] that the throughput stabilizes for 100,000 packets.

	Flow Size=640B		Flow Size=6.4KB		Flow Size=64KB	
	Reorder Rate	Retrans. Rate	Reorder Rate	Retrans. Rate	Reorder Rate	Retrans. Rate
CNET	31.7%	5.8%	35.85%	8.35%	36.36%	9.05%
SDK	32.4%	4.7%	33.4%	7.1%	33%	8.2%

Table 1. Petri Net Model Validation

whereas with a packet size of 64 B as many as 128 packets can be buffered. So only 25% of the total number of threads, i.e., 16 out of 64 threads are busy in the IXP processor. This reduces the extent of concurrent processing and correspondingly the packet retransmission in the network processor.

Although the retransmission rate is much lower for 512 B packets compared to that of 64 B, a 14% retransmission is still very high [10]. Earlier studies [10] indicate a retransmission greater than 10% can result in significant reduction (up to 60%) reduction in packet throughput. For a 16 hop network, the average number of hops for packets in Internet, the retransmission rate can further aggravate.

4 Reducing Packet Reordering

4.1 In-built schemes in the IXP Processor

Table 2 also reports the performance of in-built schemes, namely, ITS and AISR supported by IXP. We implement AISR with 1 microengine performing the buffering operation, 4 microengines performing the packet processing block, 1 microengine executing the reordering block, and 2 microengines running the transmit block. The ITS runs totally parallel, with threads in all microengines performing the complete IPv4 forwarding.

Scheme	Throughput (Gbps)	
	SDK	CNET
ITS	2.3	2.1
AISR	1.1	0.960

Table 2. Performance of ITS and AISR Schemes

It is interesting to note that AISR performs poorly as compared to the other schemes. While the ITS is able to support a line rates of 2.3 Gbps which is close to OC 48 line rates (2.5 Gbps), the AISR supports only a line rate of 1.1 Gbps. This occurs as there are only 8 threads buffering the packets to the DRAM, the first stage in AISR. This coupled with the saturation of the DRAM result in a lower throughput. With an increase in the number of threads to 16, we observe that the throughput drops to 0.9 Gbps since a global synchronization needs to be done across all the 16 threads. While our implementation of AISR may not be the most efficient, we observe that the maximum possible throughput with only one of the three stages, viz., the receive block of AISR is 2.1 Gbps. Hence the AISR throughput is limited to a maximum of 2.1 Gbps.

Thus in-built schemes AISR and ITS cause a sufficient degradation in throughput (63% and 23% respectively). Hence, in order to reduce packet reordering and its impact, we explore a few transmit buffer allocation schemes, as well as the impact of other architectural parameters.

4.2 Buffer Allocation Schemes

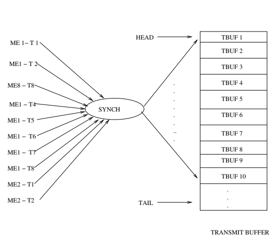
The transmit buffer allocation, as observed in Section 3.2, can independently induce packet reordering. Hence, we explore the following transmit buffer allocations to reduce reordering.

Global Buffer Allocation: In this scheme (depicted in Figure 6(a)), the competing threads are allocated transmit buffer space using global synchronization, a mutual exclusion operation. The mutual exclusion operation is performed across all threads in all microengines. The mutex variable is stored in the scratch pad as it is common to all the MEs. Since synchronization is performed across all the microengines this can result in a drop in the throughput.

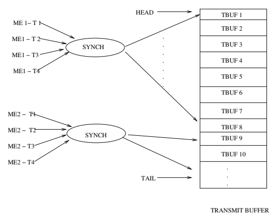
Local Buffer Allocation: In this scheme, shown in Figure 6(b), contiguous sets of locations are allocated to different microengines. But threads within a microengine compete for a common chunk allocated to that microengine through a mutual exclusion operation. The transmit buffer is allocated by performing a mutual exclusion operation locally within a microengine. There is one mutex variable for each microengine. Since only threads within a microengine share a single mutex variable, the overheads are relatively low compared to the global buffer allocation scheme.

Strided Buffer Allocation: This scheme (refer to Figure 6(c)), allocates buffers to microengines and threads a priori. However, unlike the contiguous case, the buffer is allocated in a strided way. The stride is dependent on the number of active microengines. The threads ME1-T1, ME2-T1, $\cdots$, ME1-T2 place packets in TBUF1, TBUF2,..,TBUF9 respectively.

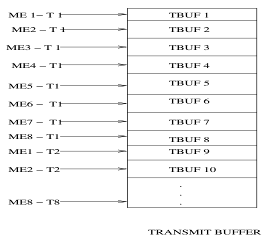
A disadvantage of contiguous and strided allocation, as compared to local or global buffer allocation, is that they assume a fixed buffer size. In our study we assume a packet



(a) Global



(b) Local



(c) Strided

Figure 6. Different Transmit Buffer Allocation Schemes.

size of 64 B (as in DoS attack, worst case scenario) [6] [13], or 512 B (average packet size) [11]. In a general situation, as the buffer size may vary from minimum to maximum packet size, a buffer size equal to the maximum packet size (1.5 KB) needs to be allocated. This may result in underutilization of the transmit buffer when the packet sizes vary widely. On the positive side, contiguous and strided buffer allocation schemes enjoy the benefit of not requiring any synchronization, which leads to better packet throughput.

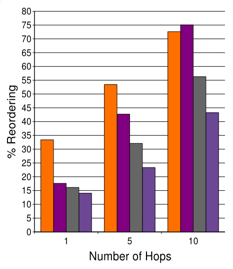
4.2.1 Performance of Buffer Allocation Schemes

Table 3 reports the throughput achieved for different buffer allocation schemes in a single hop network for 64 B and 512 B packet size. We observe that the local and global allocation schemes suffer significant reduction in throughput. On the other hand, the strided buffer allocation performs as well as contiguous allocation. The impact of various schemes on reordering and retransmission is shown in Figures 7 and 8 for 1, 5, and 10 hops. While strided and contiguous allocation result in significant retransmission rates (greater than 55%) for 10 hops, the local and global schemes reduce the retransmission rates to 45% and 33% respectively. However, the throughput achieved by global and local schemes,1.1 Gbps and 2.1 Gbps, are unacceptably low.

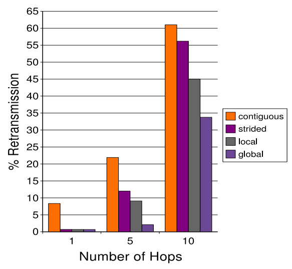
Schemes	Throughput (Gbps)	
	64 B	512 B
Contiguous	2.96	3.068
Strided	2.96	3.068
Local	2.1	2.3
Global	1.1	1.4

Table 3. Impact of Buffer Allocation schemes on Throughput (in Gbps).

On a more realistic situation, when the packet size is 512 B, the retransmission rates are 15%, 12%, 3%, and 2%, respectively, for contiguous, strided, local and global buffer allocation. While local and global allocation schemes achieve very low retransmission rate, their throughput is also very low. From this discussion we observe that there exists a trade-off between the throughput and the retransmission rate among different schemes. The retransmission rates of of strided allocation scheme (12%) is still considered to be high to cause significant degradation in TCP performance [10].



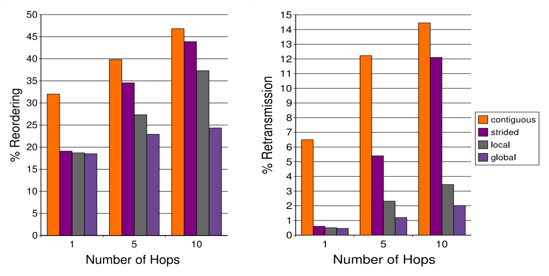
(a) Reordering



(b) Retransmission

Figure 7. Impact of Various Buffer Allocation (Packet Size 64 B) - CNET Result.

The global scheme completely eliminates packet reordering due to transmit buffer allocation. The reordering/retransmission experienced in this scheme is entirely due to the concurrent processing of packets by the MEs and threads. Hence, we study the impact of the architecture parameters, such as number of microengines and number of threads on retransmission in the following subsection. Further, since the strided buffer allocation while reducing the retransmission rate as compared to the contiguous allocation does not impact the throughput, we use the strided buffer allocation in the following sections.



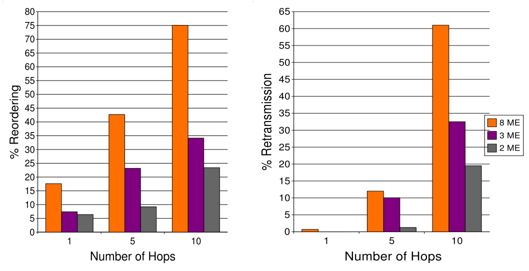
(a) Reordering (b) Retransmission

Figure 8. Impact of Various Buffer Allocation (Packet Size 512 B) - CNET Result.

4.3 Tuning Architecture Parameters

4.3.1 Impact of the Number of Microengines

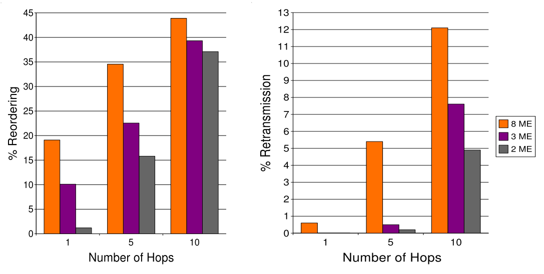
We observe that the throughput of the IXP 2400 saturates at 2.96 Gbps for a total of 16 threads (8ME $\times$ 2 threads, 4ME $\times$ 4 threads, or 2ME $\times$ 8 threads). This occurs as the memory (DRAM) saturates beyond 16 threads. Hence, a network processor with fewer microengines, while giving the same throughput can reduce the reordering due to concurrent processing. Figures 9 and 10 show the impact of number of microengines(each microengine running 8 threads) on packet reordering/retransmission.



(a) Reordering (b) Retransmission

Figure 9. Impact of Number of Microengines (Packet Size 64 B) - CNET Result.

We observe that the packet retransmission drastically reduces from close to 56% (64 B) and 12% (512 B), for 8 ME x 8 threads, to 19% (64 B) and 5%(512 B), for 2 ME x 8 threads. These reduction in retransmission are achieved without any penalty on packet throughput. Thus, a network processor using 2 or 3 microengines, while consuming lesser area, can reduce retransmission by up to 27% for 64 B packet and 5% for 512 B packet, while providing a transmit rate of a 8 microengine.

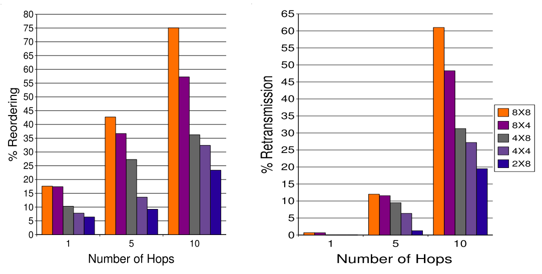


(a) Reordering (b) Retransmission

Figure 10. Impact of Number of Microengines (Packet Size 512 B) - CNET Result.

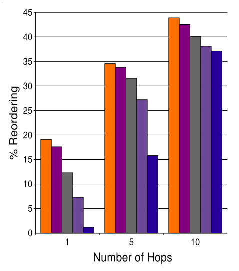
4.3.2 Impact of the Number of Threads

Figures 11 and 12 compare the impact of number of active threads on reordering and retransmission. In the above figure, a 4x8 configuration refers to 4 microengines with each microengine running 8 threads. It is interesting to note that configurations running the same total number of threads give different retransmission rates. For example, a 4x8 configuration reduces the retransmission for 1, 5, and 10 hops by up to 21% as compared to an 8x4 configuration. Both configuration give a throughput of 2.96 Gbps This indicates that the impact of multiple microengines on packet ordering is more severe than that due to multiple threads. A similar trend is observed for 512 B packet size although the reduction in retransmission/reorder rates are lower. This is due to the limited buffering possible with the 512 B packet size, as explained earlier.

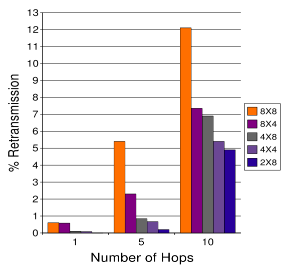


(a) Reordering (b) Retransmission

Figure 11. Impact of Number of Threads (Packet Size 64 B) - CNET Result.



(a) Reordering



(b) Retransmission

Figure 12. Impact of Number of Threads (Packet Size 512 B) - CNET Result.

4.4 Pipelined Packet Flow

Our study on buffer allocation schemes indicates that the penalty, in terms of throughput, in reducing packet reordering is very high. Our results on architecture parameter tuning indicate that the concurrent processing can cause up to 33% and 2% retransmission for 64B and 512B packets. So we explore a packet forwarding scheme, Packet sort, where the packet processing is pipelined as three stages (refer to Figure 13).

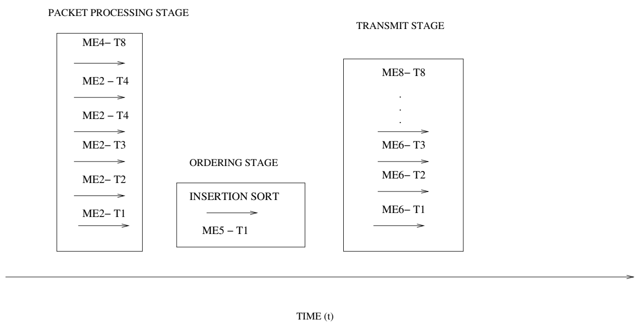


Figure 13. Packet Sort Implementation in IXP

In the first stage, m microengines concurrently move the packets from RFIFO to DRAM and subsequently process them (based on the packet forwarding application). Packets are placed in DRAM, by the threads from the first stage, based on the flow information. In the second stage, the n microengines sort the packets based on the flow information and store it in the scratch pad. The overhead involved in the sorting (insertion sorting is performed) is minimal as the microengine utilization is low in the second stage. The sorted packet addresses and the corresponding transmit buffer addresses are stored in the scratch pad and communicated to the remaining k microengines, which are responsible for DRAM-TFIFO transfer and TFIFO-MAC transfer.

Scheme	Concurrent Flows	Throughput (Gbps)	
		SDK	CNET
Packet Sort	32	2.56	2.3
	10	2.5	2.3
	1	1.7	1.6
ITS	NA	2.3	2.1
AISR	NA	1.1	0.960

Table 4. Comparison of Various Schemes to Overcome Reordering.

After extensive experimentation, we observe that m=4, n=1, k=3 gives a maximum throughput of 2.5 Gbps. Packet sort completely eliminates packet reordering/retransmission while supporting current line rates (2.5 Gbps). Table 4 reports the performance of packet sort for different number of flows. The throughput of packet sort is critically dependent on the number of concurrent flows and varies from 1.7 to 2.5 Gbps. This variation occurs due to the following reason. When packets from different flows arrive simultaneously in the NP, the number of packets processed concurrently in the NP being a constant, there will be fewer number of packets per flow. Hence lesser time will be spent in the sorting block resulting in an increased throughput.

Table 4 also reports the performance of in-built schemes, namely, ITS and AISR supported by IXP. We observe that packet sort gives a throughput improvement of at least 16% with respect to the upper bound of AISR. Packet sort also outperforms ITS for 10 or more concurrent flows It is also able to support OC 48 line rates for 10 or more flows. However, the throughput drastically reduces (to 1.7 Gbps) for 1 flow. It is important to note that typically routers encounter multiple flows (from multiple hosts/edge routers) and hence the single concurrent flow is not truly reflective of the type of traffic handled by a NP.

5 Related Work

Wolf et al. [17] use an analytical performance model for network processors to quantify different design alternatives and optimize the design for power consumed per unit area for different applications. Thiele et al. [15] develop a framework for design space exploration of network processors. These approaches have used an analytical models for their evaluation. Other approaches [4] [16] have used simulation for their evaluation. Crowley et al. [4] study the impact of different processor architectures on network applications. Spalink et al. [13] study the IXP 1200 processor for IPv4 forwarding and report forwarding rates for different number of threads. The above studies while analyzing the performance of network processors have ignored the impact of the IXP architecture on packet reordering.

Benett et al. [3] are one of the earliest to report the problem of reordering and its potential impact on the network throughput. Laor et al. [10] artificially introduce reordering and study the impact of reordering and retransmission on throughput. Their study indicate that a retransmission of 10% of packets can reduce the network bandwidth by up to 10%. Savage et al. [2] develop a metric to quantify reordering and use it to measure on different links. Jaiswal et al. [12] measure and classify the reordered/retransmitted packets as those arising from routing loops or network duplication or due to the loss of the packet. However, these studies have ignored the impact of the network processor and the concurrency support by it The penalty to maintain packet order in network processor is, as indicated in our results, very high and cannot be neglected.

6 Conclusions

This paper studies the impact of parallel processing in network processor on packet reordering and retransmission using a Petri net model. Our results show that the network processor architecture can cause up to 60% retransmission which can significantly degrade the TCP throughput. We explore different buffer allocation schemes that reduce the retransmission rates from 61% to 33% for 64 B packets and 14% to 2% for 512 B packets. However, these benefits come at the cost of a significant reduction in throughput. Our results reveal that a network processor with fewer microengines significantly reduces the number of retransmissions while giving the same throughput. Last our Packet Sort scheme, which is a pipelined approach that dedicates certain number of threads to sort the packets, eliminates retransmission while achieving a throughput close to the current line rate. As future work, we plan to investigate the impact on the retransmission and throughput when the TCP congestion control mechanism at the sender limits the transmit rate.

Acknowledgments

This work was partly supported by a research grant from the Consortium for Embedded and Internetworking Technologies (CEINT) and Arizona State University, Tempe, USA.

References

[1] F. Baker. RFC 1812 - Requirements for IP Version 4 Routers, June 1995.

[2] J. Bellardo, S. Savage. Measuring packet reordering. Proceedings of the ACM SIGCOMM IMW, Marseille, France, Nov. 2002.

[3] J. Bennett, C. Partridge, and N. Shectman. Packet Reordering is not a Pathological Network Behavior. IEEE/ACM Transactions on Networking, 7(6):789798, 1999

[4] P. Crowley, M. Fiuczynski, J.L. Baer, B. Bershad. Characterizing processor architectures for programmable network interfaces. In Proceedings of International Conference on Supercomputing, Feb 2000.

[5] C. Fraleigh, S. Moon, C. Diot, B. Lyles, and F. Tobagi. Packet-level traffic measurements from a tier-1 IP backbone. Technical Report TR01-ATL110101, Sprint ATL Technical Report, Nov. 2001.

[6] S. Govind and R. Govindarajan. Performance Modeling and Architecture Exploration of Network Processors. Technical Report TR-HPC-11/2005, High Performance Computing Lab., Supercomputer Education and Research Centre, Indian Institute of Science, Bangalore 560 012, June 2005. http://hpc.serc.iisc.ernet.in/Publications/govind2005.ps.

[7] Intel Corporation, Intel IXP 2400 Network Processor Hardware Reference Manual. Revision 7, Nov. 2003.

[8] Intel IXP2400/IXP2800 Development Tools Users Guide. Revision 11, Mar. 2004.

[9] Intel IXP2400/IXP2800 Network Processors Microengine C Language Support Reference Manual. Revision 9, Nov. 2003.

[10] M. Laor, L. Gendel. The Effect of Packet Reordering in a Backbone Link on Application Throughput. IEEE Network Sept/Oct. 2002.

[11] National Laboratory for Applied Network Research (NLANR). Insights into Current Internet Traffic Workloads.(http://www.nlanr.net/NA/tutorial.html)

[12] Sharad Jaiswal, G. Iannaccone, C. Diot, J. Kuorose, and D. Towsley. Measuring and classification of out-of-sequence packets in a Tier-1 IP Backbone, International Measurement Workshop(IMW), 2003.

[13] T. Spalink, Scott Karlin, Larry Peterson. Evaluating Network Processors for IP Forwarding. Technical Report TR-626-00, Department of Computer Science, Princeton University, Nov. 2000.

[14] R. Stevens. TCP/IP Illustrated, Volume 1: The Protocols, Addison-Wesley, 1994.

[15] L. Thiele, Samarjit Chakraborty, Matthias Gries, Simon Kunzli. Design space exploration of network processor architectures. 1st Workshop Workshop on Network Processors, Cambridge, MA, Feb. 2002.

[16] T. Wolf. and M. Franklin. Commbench : A Telecommunication benchmark for Network Processors. In Proceedings of the International Symposium on Performance Analysis of Systems and Software, Apr. 2000, pp. 154-162.

[17] M. Franklin, T. Wolf. A Network Processor Performance and Design Model with Benchmark Parameterization. 1st Workshop on Network Processors, Cambridge, MA, Feb. 2002.

[18] W. M. Zuberek. Modeling using Timed Petri Nets - event-driven simulation, Technical Report No. 9602, Dept. of Computer Science, Memorial Univ. of Newfoundland, St. John's, Canada, 1996 (ftp://ftp.ca.mun.ca/pub/techreports/tr-9602.ps.Z).