Efficient Techniques Exploiting Memory Hierarchy to Improve Network Processor Performance

A Thesis

Submitted For the Degree of

Master of Science (Engineering)

in the Faculty of Engineering

by

Girish B.C.

Supercomputer Education and Research Centre

Indian Institute of Science

BANGALORE – 560 012

June 2008

©Girish B.C.

June 2008

All rights reserved

TO

My Grandmother and Parents

Acknowledgements

I would like to thank my advisor, Prof. R. Govindarajan, for this guidance and encouragement during my course at IISc. He not only advised me on research topics but also inspired me to explore new areas for my work. He has nurtured an open environment in our lab which encourages constant discussion and collaboration. I thank Prof. Joy Kuri for discussing with us the aspects of TCP protocol which helped us to formulate the work on buffering for core routers.

I thank the administration of the Institute for providing us a superb campus with great facilities and a scintillating atmosphere to work and play. In particular, I would like to thank the Chairman of SERC for providing various resources and encouraging student activities.

HPC lab has been a pleasurable place to work in on account of the people. I would like to thank: Kaushik for always being ready for a lively discussion on topics ranging from research to films, Santosh for providing valuable inputs from the internet, Mani for his insightful comments, Sreepathi for giving timely 'fundae' and proactive administration of our temperamental lab resources, Gajjar for providing a different perspective, Abhishek for his enthusiastic attempts at making me a rock fan and Aditya for this wise quips. I would like to thank Govind Shenoy for helping me understand more about network processors during the initial part of my study and for providing valuable inputs during the course of my work.

I thank Naveen, Prashanth, Shiva, Manu, Holla, Hemanth, Ranganath, Subramanya, Srinivansan, Rajesh, Shyamprasad, Vishwas, Ramprakash, Balaji, Phani Sudheer and all my friends at IISc for making my stay truly memorable.

I am indebted to my family for supporting and encouraging me at all times. Last, but not the least, I thank Him for providing me this opportunity to pursue my studies at IISc.

Publications based on this Thesis

1. Girish B.C and R. Govindarajan, *A Petri Net Model for Evaluating Packet Buffering Strategies in a Network Processor*, in the Proceedings of the 4th International Conference on the Quantitative Evaluation of Systems (QEST) 2007, Edinburgh, Scotland, September 16-19, 2007

Abstract

The performance of network processors depends on the architecture of the chip, the network processing application and the workload characteristics. In this thesis, we model the memory hierarchy of a network processor and its interaction with the network application. We use the observed characteristics from internet traces to propose new packet buffer allocation policies, dynamic buffering for core routers and to propose a new cache replacement policy for caching the results of packet classification.

Previous studies have shown that buffering packets in DRAM is a performance bottleneck. In order to understand the impediments in accessing the DRAM, we developed a detailed Petri net model of SDRAM, the memory used as packet buffer in routers. This model is integrated with the Petri net model of IP forwarding application on IXP2400. We show that accesses to small chunks of data less than 64 bytes reduces the bandwidth realized by the DRAM. With real traces, up to 30% of the accesses are to smaller chunks, resulting in 7.7% reduction in DRAM bandwidth. To overcome this problem, we propose three buffering schemes which buffer these small chunks of data in the on-chip scratchpad memory. These schemes also exploit greater degree of parallelism between different levels of the memory hierarchy. Using real traces from the internet, we show that the transmit rate can be improved on an average by 21% over the base scheme without the use of additional hardware. Further, under real traffic conditions, we show that the data bus which connects the off-chip packet buffer to the microengines, is the obstacle in achieving higher throughput.

Earlier studies have exploited the statistical multiplexing of flows in the core of the internet to reduce the buffer requirement. We observe that due to over provisioning of links the buffer can be substantially reduced. This enables us to the use on-chip memory in the IXP 2400 to buffer packets during most regimes of traffic. We propose a dynamic buffering strategy in network processors which buffers packets in the receive and transmit buffers when the buffer requirement is low. When the buffer requirement increases due bursts in the traffic, buffer space is allocated to the packets by storing them in the off-chip DRAM. This scheme effectively mitigates the DRAM buffering bottleneck, as only a part of the traffic is stored in the DRAM. We show that with an input traffic rate of 6300Mbps, the dynamic buffering scheme has lower packet drop rate than the DRAM buffering scheme. Even with 90% output link utilization, the dynamic buffering scheme is able to support a line rate of 7390Mbps and a packet drop rate of 2.17%. This packet loss is still lesser than the DRAM buffering scheme which drops more than 4% of the packets even with a traffic rate of 5417Mbps.

The distribution of flows sizes in the internet is highly skewed with a few large flows which transfer a majority of packets and numerous small flows that contribute only a small fraction to the traffic. The skewed distribution of flows leads to degraded performance of result caches with LRU policy. This cache replacement policy tends to retain the results belonging to the short flows in the internet. We propose a new cache management policy, called Saturating Priority, which leverages this distribution of flows to remove results belonging to small flows from the result cache, while trying to retain the results for large flows. The proposed replacement policy outperforms the widely used LRU cache replacement policy for all traces and cache sizes that we considered. It also covers up to 74% of the gap between LRU and an oracle policy which inserts a result into the cache only if the flow has multiple packets.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Introduction

There has been a significant increase in the bandwidth supported by the network medium in recent years. At the same time, the complexity of applications running on routers is increasing [14]. These two trends have prompted the adoption of new processing paradigms in the network processing domain. The traditional approach of designing application specific integrated circuits (ASICs) for these applications has not been satisfactory in addressing these demands due to its large design and verification overheads. Programmable Network Processors (NPs) are a viable alternative to custom ASICs to support high speed network processing applications. These application specific integrated processors (ASIPs) achieve a high throughput by exploiting packet-level parallelism in a network application. Software controlled ASIPs effectively address the need to deploy complex network processing applications within a short duration.

Several NPs are available commercially such as Intel IXP2400 [25], IBM Power NP [8] and Motorola C-Port [4]. The architecture of the NPs are similar and follow the NP architecture described in [45]. The generic NP architecture consists of several simple processing engines with per processor instruction and data caches. These processors are used for performing data plane operations on packets. The processors share a common IO channel to the off-chip memory. This channel is used by the system controller to send packets to the individual processors for processing. Packets are received and transmitted through a cell based interface [5]. For each incoming packet a Packet Demultiplexer unit determines the processing required.The packet is enqueued until a processor becomes available for processing it. When a processor becomes free, the packet and the flow information is sent on the IO channel for processing. The processed packet is sent back to the Packet Demultiplexer which enqueues it before transmitting it on the routing fabric to the output port. Off-chip DRAM memory, called packet buffer, is used to store packets when they are being processed. Packet buffers also store packets during periods of congestion in the network.

Packet buffer size is usually equal to *round trip time* $*$ *transmit rate*. This could be as large as 75MB for an OC-48 link, assuming a 250ms round trip time. On account of their high storage capacity at low cost, DRAMs are the storage medium of choice for packet buffers. Since each packet is written into and read from the packet buffer, the DRAM must support twice the transmit bandwidth of the NP. The bandwidth requirement could be higher if parts of a packet are accessed multiple times.

NPs have specialized hardware units such as hash unit, for performing address lookup through a hash table and crypto unit, for encryption of packets. Off-chip SRAM and DRAM memories are provided for storing application data structures and packet data respectively. In addition, an NP can have a full fledged processor core for performing control plane tasks such as initializing the NP, setting up the processing data structure and updating it.

The performance of a network application is the result of a complex interplay between the workload and the hardware architecture such as the micro engines, threads, hardware computation units and the memory hierarchy. A number of previous studies have used different methods to evaluate performance of Network Processors: [21] uses a Petri net model based evaluation, [19] uses queuing based models whereas [6] uses analytical modeling. As mentioned earlier, the DRAM which is used for packet buffering must support at least twice the transmit bandwidth. Thus exploiting a high DRAM bandwidth and making the DRAM accesses efficient is a key to high throughput in network processing applications [24]. Hence we study in greater detail the DRAM performance specifically and the memory hierarchy of a network processor in this thesis. We use a colored Petri net model, similar to [21], which captures the timing of packet arrivals, access to packet buffers, on-chip and off-chip memory, the state of the hardware structures, threads for processing packets and the workload. Further, the workload characteristics such as the packet length distribution and variation in the traffic arrival rate are accurately modeled by our Petri net model. The design of an NP and provisioning of resources could vary substantially depending on the workload.

In this thesis, we study the performance of NPs and propose schemes for enhancing their performance. Although we use the IXP 2400 [25], a widely used network processor, the proposed schemes are generic and applicable to other NPs as well. This is because architecturally IXP 2400 is similar to the generic NP architecture described above.

1.2 Evaluation of Packet Buffer Allocation Strategies

Our Contribution

In the first part of the thesis, we build a detailed Petri net model of DRAM memory. Our colored Petri net model captures the various hardware structures of the DRAM, including banks, row and column, access to page buffers and the data bus. We also model the packet buffer allocation, DRAM address mapping scheme and cell based interface for packet transmission. The model enables us to study the effect of multiple DRAM banks in hiding the bank access latency. The detailed DRAM model is validated using a DRAM simulator [43]. We incorporate this DRAM model into the processing of the most common network application, IP forwarding, on IXP 2400 NP. The other levels of the memory hierarchy like the on-chip scratchpad memory and the off-chip SRAM are also modeled. Unlike a trace based approach, this integrated framework allows us to capture the timing information and to study the interaction of the program on a given hardware. This approach provides insight into the potential bottlenecks and enables us to measure the performance improvement from various packet buffer allocation schemes.

Presence of multiple banks in a DRAM allows data transfer from one bank to be overlapped with access of another bank. The achievable bandwidth from the DRAM drops if the large access latency to the banks is not overlapped by data transfers from other banks. In an NP, the size of data to be transferred to/from the DRAM is variable. We observe that small data transfers (less than 64 bytes) to/from DRAM do not hide the bank access latency sufficiently due to which, the bandwidth realized from the system decreases. With the naïve packet buffering scheme in IXP2400, a significant portion (up to 30%) of the accesses are of small size, which we call *narrow accesses.* Using our Petri net model, we show that the bandwidth realized reduces by 7.7% in such a scenario. We identify such narrow accesses as one of the causes for reduced DRAM bandwidth and propose packet allocation schemes that address this issue. In particular, we propose three buffering schemes: Header Buffering (HB), First Cell Buffering (FCB) and First Cell Buffering + Tail Buffering (FCB+TB). These schemes buffer different parts of the packet in the on-chip scratchpad memory in order to prevent narrow accesses to DRAM. Using the detailed Petri net model for IP forwarding, the throughput of the NP is studied using real traces from the internet. The three buffering schemes, namely, HB, FCB and FCB+TB improve the throughput of the IPv4 forwarding application by 9.91%, 15.4% and 21.3% respectively over the base scheme where the entire packet is buffered in the DRAM.

Previous studies [21] only considered traffic consisting of 64 byte sized packets. The authors demonstrate that under such conditions, a maximum throughput of 3.2 Gbps is achievable. For this trace, limited computation power of the hash unit is observed to be an impediment. However, we show that with real traces from the internet, a throughput of 6.2 Gbps is achievable. At higher transmit rates we show that, the data bus becomes the bottleneck. This demonstrates that different traffic patterns stress different components of the NP. Such behavior must be carefully considered while provisioning network resources for different traffic patterns.

Buffering packet data in on-chip memory is different from the earlier approaches proposed in [32, 11] where the application data structure is cached. These schemes exploit the temporal locality observed in accesses to program data such as trie, classification tables etc. In [21], packet headers are buffered in off-chip SRAM, but the main focus of there is exploiting the SRAM bandwidth along with DRAM bandwidth. This is different from our HB scheme where the header is buffered in the on-chip scratchpad memory. Buffering packet data in on-chip memory has not been studied previously as this data is large and there exists little temporal locality in their access. Storing small segments of the packet in on-chip memory has low space overhead and has the added benefit of distributing the memory accesses to different levels of the memory hierarchy.

1.3 Dynamic Packet Buffering

In the second part of the thesis, we propose a dynamic buffering scheme to reduce the buffer requirement in core routers. As mentioned in Section 1.1, the size of the packet buffer is governed by the thumb rule that its size is equal to the product of the router bandwidth and the latency of round trip. Previous studies such as [41] have supported this guideline since it achieves maximum throughput in a congested link. However this study considered only a small number of long flows. The operating conditions in the core of the internet are significantly different. There exists a large number of concurrent flows, more than 10,000. Appenzeller et al. [9] have shown that due to the statistical multiplexing of these flows, the buffer requirement reduces by a factor of $\sqrt{n}$, where n is the number of concurrent flows. They show that the link utilization utilization reduces by about 1%.

Our Contribution

We observe the fact that in core routers, the links are substantially overprovisioned and the link utilization is about 50% [9, 12]. This fact can be capitalized to reduce the buffer requirement even further. We propose a dynamic buffering scheme that buffers packets in the off-chip DRAM memory only when the packet arrival rate is high and the output link is congested. Under normal traffic rates, packets are buffered in the on-chip memory. Using a Petri net model, we show that under real traffic conditions, the on-chip memory in the IXP 2400 can be effectively utilized to buffer packets. Our simulation results show that at 76% link utilization, for a large majority of the time, buffering packets in the on-chip memory suffices and only 2.8% of the packets are buffered in DRAM memory due to high packet arrival rate. When the amount of buffering required increases due to increase in the rate of packet arrival, greater amount of storage is provided in the DRAM memory. As the amount of buffering is reduced by the dynamic buffering scheme, it effectively mitigates the DRAM bandwidth bottleneck that we observed previously in the DRAM buffering scheme where all the packets are stored in the DRAM memory.

We built a Petri net traffic generator that generates traffic with the characteristics present in traffic rates in uncongested links. The proposed dynamic buffering scheme performs better than the DRAM only buffering scheme under a wide range of traffic conditions. For traffic with a mean line rate of 5417Mbps and variation of 5.52%, the dynamic buffering scheme has a packet drop rate of 0.03% whereas the DRAM buffering scheme has a packet drop rate of 4%.

With dynamic packet buffering, the number of threads dedicated to processing affects the packet drop rates. With 8 threads, there is a high packet drop rate as the NP cannot effectively process packet bursts leading to overflow of the Receive Buffer (RBUF). However, with increased number of threads, the application is more resilient to packet bursts. The number of packets dropped is less than 1% with a mean input rate of 6215Mbps and 24 processing threads.

When the input traffic rate is 90% of the outlink bandwidth, there are substantial packet drops (2.17%) due to the limited size of the RBUF. However, this is still lower than the DRAM buffering scheme, which experiences 4% packet drop even for a link utilization of 66%. The higher utilization of the RBUF under this condition reduces the ability of the NP to absorb bursts in traffic, leading to higher packet drops.

1.4 Improving Performance of Result Caches for Packet Classification

Recently, digest caches have been proposed as an effective method to speed up packet classification in network processors [13, 31]. In the last part of the thesis, we propose a new cache replacement algorithm to improve the performance of such result caches in NPs for packet classification. In digest caches, a 32-bit digest is obtained by hashing the fields used in packet classification. This digest is stored along with the flow class in a small result cache whose size is a few kilobytes. Packet classification proceeds by checking for the result of the flow in the digest cache. Slower algorithmic classification is performed only when the digest is not found in the result cache.

Our Contribution

We show that the presence of a large number of small flows and a few large flows in the internet has an adverse impact on the performance of this digest cache. A few large flows transfer a majority of the packets whereas the contribution of small flows to the total number of packets transferred is small. In such a scenario, the LRU cache replacement policy, which gives maximum priority to the most recently accessed digest, tends to evict digests belonging to the few large flows. We propose a new cache management algorithm, called *Saturating Priority* (SP) which aims at improving the performance of result caches by exploiting the disparity between the number of flows and the number of packets transferred. Under the SP cache replacement policy, a new digest entry is inserted in a set with the lowest priority. Its priority increases and reaches the maximum priority as more accesses are made to it. During eviction from the cache, the item with the lowest priority in the set is removed. With real internet traces, we evaluate the miss rate with SP and LRU cache replacement policies. The proposed cache replacement policy outperforms the LRU cache policy for all traces and cache sizes considered. It covers 74% of the gap between LRU cache replacement and an oracle cache replacement policy with places digest entries in the cache only for flows that contain multiple packets.

We characterize the misses incurred by the result cache and show that conflict misses are small compared to capacity and cold misses. This shows that though result caches can reduce a substantial number of packet classification lookups, the size of the cache has to be substantially increased in order to reduce the misses to a small fraction.

1.5 Organization of the Thesis

The rest of the thesis is organized as follows. In the following chapter, we provide the necessary background on IXP 2400 network processor architecture, DRAM organization and access and an introduction of Petri nets, which form the basis of our model. Chapter 3 describes the Petri net model that we developed for DDR SDRAM, its integration with the IPv4 forwarding model and validation of the models. Also, we discuss the effect of narrow accesses and propose packet allocation strategies to overcome this effect and study the utilization of resources in the Network processor under different traffic conditions. A dynamic packet buffering scheme for IPv4 forwarding application in the core of the internet is proposed in Chapter 4. The evaluation of this scheme under different traffic conditions is also described in this chapter. In Chapter 5 we propose a new cache management policy for digest caches, which is motivated by the disparity between the number of flows and the packets transferred by these flows. Finally, we conclude in Chapter 6 with a brief summary of our work and directions for future work.

Chapter 2

Background

In this chapter we present the necessary background to the work presented in the remaining part of the thesis. The architecture of the IXP 2400 is explained in Section 2.1 along with the details of the packet receive and transmit interface in this NP. Since we study the DRAM packet buffering in an NP, a detailed explanation of DRAM technology, DDR SDRAM organization and data access protocol are presented in Section 2.2. We conclude the chapter with an introduction to Petri nets, their expressiveness and their suitability for evaluating network processor and workload interaction.

2.1 Intel IXP2400 Network Processor Architecture

Network processors are application specific processors that are geared towards processing of network applications. NPs have extensive support for multiple thread contexts which aim to exploit substantial amount of packet-level parallelism in network processing applications. Presence of multiple threads helps to hide the latency of accessing the program data in SRAM and packet data in DRAM. NPs typically offload specialized tasks like hash calculation and CAM lookup, frequently required by network processing applications, to hardware engines.

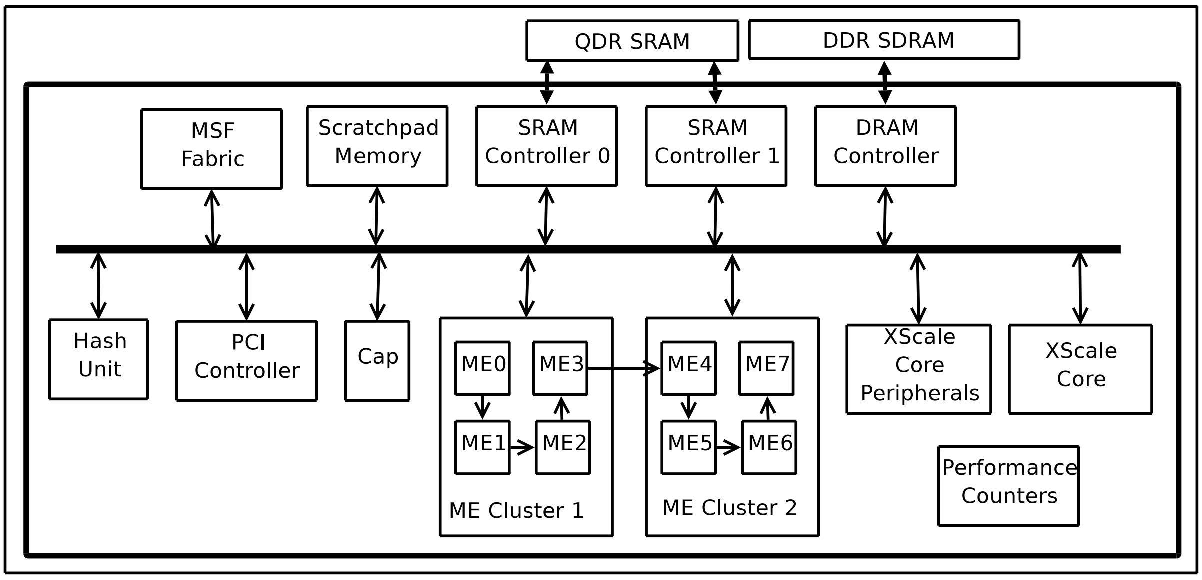


Figure 2.1: IXP2400 architecture

2.1.1 Organization of IXP 2400

Figure 2.1 shows the architecture of Intel IXP2400. This NP has 8 simple in-order cores called microengines. The microengines perform the data plane processing, which consists of that part of the network application which is performed on every packet. Each microengine(ME) can support up to 8 thread contexts. Threads within a ME share an instruction store which can hold 4K, 40-bit instructions. A hardware thread arbiter schedules the enabled threads in a round robin fashion. The threads are non pre-emptive, i.e., the next thread cannot be scheduled until the previous thread stalls or explicitly gives up control.

Each ME has 256 general purpose 32-bit registers which are partitioned among the threads. As a result, the threads have zero cycle context switch overhead. Specialized registers such as next-neighbor registers, SRAM and DRAM transfer registers are provided for data transfers. Next neighbour registers are used to communicate values with the adjacent ME. The SRAM transfer registers are used to read/write values SRAM, Hash unit, Scratchpad, Control Access proxy and MSF unit. DRAM transfer registers are used to communicate with the DRAM memory. Like the general purpose registers, the specialized registers are also partitioned among the threads.

Each thread in a ME is provided with 15 signals to aid asynchronous communication with off-chip memory and other functional units. This enables the programmer to extract parallelism in processing by initiating data transfers or specialized computation as early as possible. When the results are required, the completion of these tasks can checked by testing for the presence of the corresponding signal. If the signal is absent, the thread can release control of the ME to other threads until the activity has concluded. A thread can have multiple outstanding signals.

IXP 2400 has off-chip SRAM and DDR SDRAM memories. The dual-ported quad data rate (QDR) SRAM [3] is used to store the program data like the lookup trie. The SRAM is connected to two on-chip controllers. DDR SDRAM with 4 banks is used to buffer packets. The SDRAM is connected to the on-chip memory controller by a 64-bit channel operating at 200MHz. The SRAM size is usually 8MB whereas the DDR SDRAM size may vary from 64MB to 1GB. There is an on-chip scratchpad memory of size 16 KB, that may be used to store temporary variables, setup communication rings between threads and distribute memory accesses.

Apart from these storage locations, each ME has 640 32-bit word local memory space which is faster than scratchpad or off-chip memory. This local store can be used by the network application to store commonly used data items. Each ME contains a 16-entry Content Accessible Memory, which holds 32-bit key and 4-bit value pairs. The CAM supports LRU replacement under software control. The combination of Local memory and CAM unit can be used to realize a software managed cache [25]. There is a Cyclic Redundancy Check computation unit that provides CRC-CCITT and CRC-32 checksums. The IXP 2400 has hardware processing elements in order to speed up commonly used idioms in network processing. It has a hash unit which can calculate polynomial hash functions on 48-bit, 64-bit and 128-bit inputs.

The NP also has an XScale core which implements the ARM instruction set. It supports "Thumb" and DSP instruction set extensions but not floating point instructions. This core is responsible for the control plane processing like initializing the MEs, setting up the data structures in the SRAM, like creating the lookup trie and for handling exception conditions. Processing of special packets, such as packets with IP options set, is also off-loaded to the XScale core. Further details of the Intel IXP2400 are available in [25].

2.1.2 Packet Processing in IXP 2400

When a packet arrives at an ingress port, the interface card receives it and stores it in the Receive Buffer (RBUF). The network processor maintains a list of free threads which are ready to process packets. A thread from this list is assigned for processing the packet stored in the RBUF. The presence of multiple MEs and other hardware units provide flexibility regarding organization of the processing tasks. The processing may have a *pipelined* model [30], wherein each task of the network processing application is performed by a few dedicated threads. The threads synchronize with each other through signals and communicate data through rings in scratchpad or through specialized registers. In the *parallel threads* model [30], each thread performs all the tasks of the network processing application.

2.1.3 Packet Receive and Transmit Interface

As we focus on the buffering of packets in the NP, we discuss the processing involved in receiving, buffering and transmitting packets. The IXP communicates with the network physical device through the Media Switch Fabric (MSF). The MSF supports CSIX-L1 [5] interface which handles ATM and Ethernet traffic. It provides a cell based interface to the NP for receiving and transmitting packets. Each cell is called an mpacket. Its size can be configured to 64 or 128 or 256 bytes. We assume an mpacket size of 64 bytes in the rest of the study.

The MSF has two buffers called receive buffer (RBUF) and transmit buffer (TBUF). Each of these on-chip buffers is of 8KB size. The RBUF is used to store the packet until a thread is assigned for buffering it in the DRAM. After the processing of a packet, it is moved from the DRAM to the output port through the TBUF. The TBUF again handles the packet in fixed size mpackets. The state of the incoming mpackets is stored in a receive status word (RSW). The information about an out going mpacket is put in a control word.

The MSF interface maintains two internal lists. RX_THREAD_FREELIST is a list of free threads that are waiting to process a received packet. FULL_ELEMENT_LIST is a list of mpacket waiting to be handled by threads. When a packet is received by the MSF, it signals a thread from the free pool. The thread reads the corresponding RSW and is responsible for allocating DRAM buffers and moving the mpacket data from the receive buffer to the packet buffer. In order to move the packet data within the NP, the MSF has a data path to the MEs and off-chip SRAM and DRAM memories.

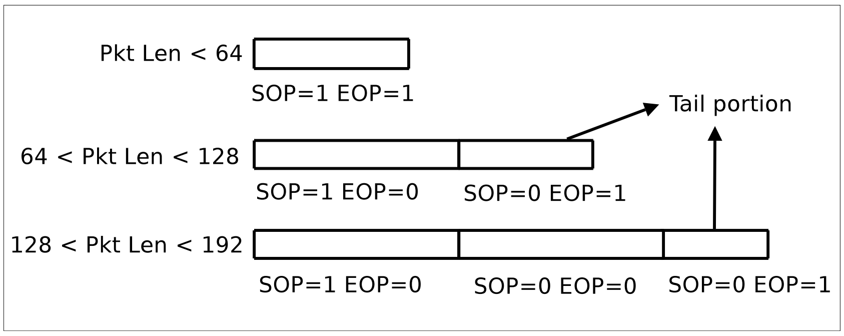


Figure 2.2: Cell based interface for packets

The RSW contains two bits called *start of packet (SOP)* and *end of packet (EOP)* which indicates the position of the mpacket within the packet. RSW also contains checksum, flags for length, error conditions etc. The setting of SOP and EOP bits for a given packet are as shown in Figure 2.2. On receiving a mpacket with SOP bit set, a new buffer is allocated [30]. For a packet of length $\leq$ 64 bytes, both the SOP and EOP bits are set. A packet whose length is between 65 and 128 bytes has two mpackets. The first mpacket has the SOP bit set and the second mpacket has the EOP bit set, as shown in Figure 2.2. Packets whose length is greater than 128 bytes have mpackets from *middle of packet(MOP)*, which do not have either of these bits set. Such a packet is shown in the third example in Figure 2.2. Since the packets in the internet are of variable length, the last mpacket is usually less than the maximum mpacket size. We call this the tail portion of the packet.

When a packet arrives at the NP, the size of the DRAM buffer allocated to it is equal to the packet size. If the packet size is not known from the first mpacket, the buffer size allocated is equal to the maximum possible length of the packet. More sophisticated allocation schemes try to allocate multiple free buffers to a given packet with a view to increase memory space utilization.

2.2 DDR SDRAM Architecture

The IXP 2400 uses a DDR SDRAM as the packet buffer. In this section we explain the architecture and working of this memory chip.

2.2.1 DRAM Technologies

DRAM chips have a number of properties that make them suitable for use as packet buffers. They have high storage capacity and dissipate lesser power then SRAMs of similar capacity. They are cheaper than SRAMs. DRAM chips have high density because each bit needs a single capacitor and a transistor whereas six transistors are required to store a single bit in SRAM.

DRAM devices are fabricated using a process optimized for small size of the storage cell, reduced leakage through the access transistor and high capacitance of the storage cell. This process is substantially different from the logic optimized process used in microprocessors. Logic circuits fabricated on DRAM optimized process are slower than similar circuits in logic optimized technologies. These considerations have forced the separation of devices using these technologies [44]. As a result, DRAM storage is off-chip and data has to be fetched to the computing unit through a data bus.

Figure 2.3 shows the circuit diagram of a DRAM that is used to hold a single bit [44]. A storage capacitor is used to hold the value of a single bit as an electric charge. When the

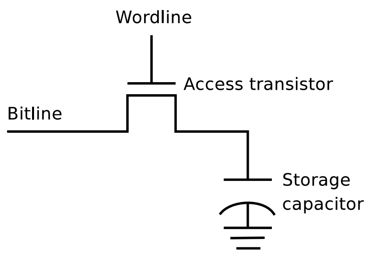


Figure 2.3: DRAM cell

bit value stored has to be read, the access transistor is enabled and the capacitor places its stored charge on the bitline. The capacitance of the storage capacitor is typically ten times smaller than the long bitline which is connected to hundreds of cells. The resulting voltage is difficult to measure. The stored value is deduced by comparing the voltage of the bitline with a reference voltage using a differential sense amplifier. A differential sense amplifier measures the difference between two voltages and amplifies it to a high or low voltage. In DRAMs, sense amplifiers also perform two other important functions. Once the storage cell has put its charge on the bitline, it does not retain enough charge to support another read operation. In order to overcome this problem, the sense amplifier restores the storage capacitor to the voltage that it contained before the read operation. After the bit value has been identified, the sense amps continue to service multiple read requests until a precharge operation is initiated. This array of sense amps into which the contents of a DRAM row are read forms the *page buffer*. During a precharge operation, the sense amp discharges and the voltages on the bit line pairs are equalized to enable the next read operation.

The storage capacitor used to store the bit value holds the voltage for a limited period of time as it gradually loses charge due to leakage current through the access transistor. As a result, DRAM cells have to be *refreshed* at regular intervals of time wherein the bits are read out from the cell and written back. This is called *refresh cycle*. Typically DRAM devices have a refresh cycle every 32msec or 64msec [44].

Over the years different technologies that have been used to build DRAM chips. Some of these technologies and their distinguishing features are as follows [15]

Fast page mode DRAM: FPM DRAM exploited page buffer locality to improve bandwidth. It allowed the row address to be kept constant while successive column addresses accessed data from the open page buffer.

Extended Data Out DRAM: EDO DRAM allowed faster precharge or next column access by providing a latch between the page buffer and output pins to hold the data.

Synchronous DRAM: Previous technologies had been asynchronous wherein the memory controller initiated the actions through timing strobes. SDRAM allowed the DRAM operations and multiple byte data transfers to conclude at predefined clock intervals. This feature seen in current DRAM memories also.

Rambus DRAM: This memory significantly changed the DRAM interface by having a fast bus which is used for both data and commands. It allows multiple outstanding requests to the memory by using a split request/response protocol. A packet based transaction is used between the memory controller and the DRAM chip. In order to supply the critical word first, burst reordering can be used, wherein data is supplied in a different order than the address order.

In spite of the various protocols and implementations, DRAM chips have retained the same basic organization and their memory operations can be described in terms of few generalized steps. Wang [44] has proposed a generic DRAM access protocol, which we use in modeling DRAM memory.

The organization of the DRAM device is shown in Figure 2.4. Such a device has 4 grids of DRAM cells called banks which can be accessed independently. Each bank has an associated sense amp array, called page buffer which is used to hold the contents of the currently accessed row. The page buffers share IO gates that connect them to the data pins. The banks are controlled by the address and command bus. These buses are used to send the data address and to issue commands to the DRAM device.

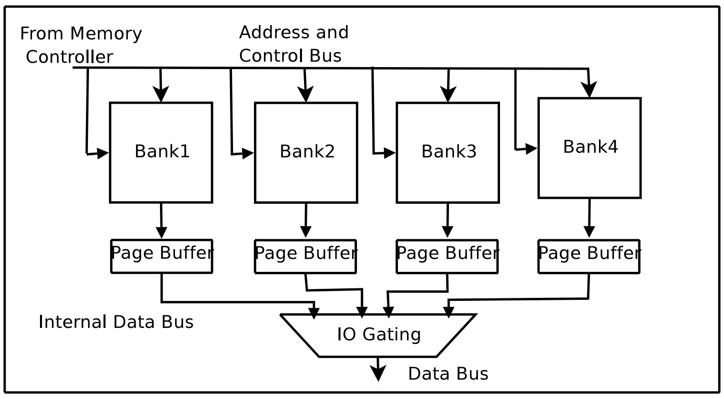


Figure 2.4: SDRAM architecture

2.2.2 SDRAM Data Access Process

The banked architecture allows the access of each bank to be pipelined. The other banks can be in various stages of memory access when data is fetched from a given bank. The generic DRAM access model uses a resource usage model to determine the memory operations that may be pipelined. This model allows two different steps that do not require the same hardware resource to occur concurrently [44].

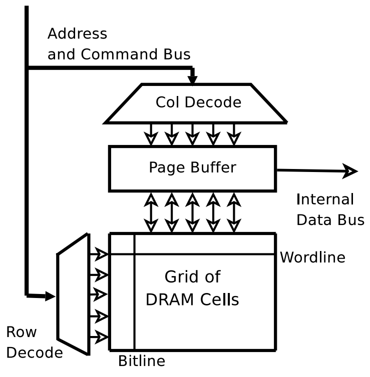


Figure 2.5: SDRAM data access steps

Figure 2.5 illustrates the various stages in DRAM data access. An access to the DRAM device consists of the following steps [44].

- *Precharge*: The row of sense amps is prepared for the next row access by discharging the stored values and setting the voltage on one of the lines to the reference voltage. Time to precharge is denoted by t_{RP}. In case of a write operation to the DRAM cells, the precharge operation is not started until the correct values have been stored in the DRAM cells. This is achieved by having a write recovery delay (t_{WR}) before t_{RP}.

- *Row access*: The row address is sent over the address bus and is latched on to the row decoder by enabling *row access strobe (RAS)* command. The entire row is read into the page buffer after a finite delay called row to column delay (t_{RCD}).
- *Column access*: After the column address is latched onto the column decoder through the *column access strobe (CAS)* command, the appropriate data from the page buffer is available on the data bus after a finite delay called the column access latency (t_{CL}). The data passes through the IO gates which connects the page buffers to the data pins. The data is transferred on the data bus for ‘t’ cycles, where ‘t’ is determined by the data access length and bus width.

DDR (Double Data Rate) SDRAM transports two beats of data on the data bus in one clock cycle. Here one beat of data is equal to the width of the data bus. The memory controller is responsible for maintaining the precise timing between these steps. A page miss is said to occur when two consecutive accesses to a bank go to different rows. A page hit occurs when consecutive accesses to a bank are to the same row.

Depending of the page miss behavior, the page buffers may be managed with either a *closed policy* or an *open policy.* In closed policy, the sense amps are immediately precharged following a read. Such a policy is useful when accesses to a bank are random and two consecutive bank accesses are likely to go to different rows. In such a scenario, closed policy reduces the delay for the next access by initiating the precharge as soon the previous access is serviced. However, with a *closed policy* when consecutive accesses go to the same row, the second access is delayed until the precharge, row and column accesses are performed. On the contrary, the *open policy* leaves the data in the page buffer and does not precharge it until an access to a different row in the bank is seen.

Such a policy is useful when page hits are more likely. Also, a page buffer that has data stored in it, consumes more power than the quiescent state. As a result, the page buffer management policy has a bearing on the power-performance operating point of the DRAM chip. Detailed architecture of DDR SDRAM memory and the steps involved in accessing it are elaborated in [44].

2.3 Petri Nets

Petri nets were proposed by Carl Adam Petri in 1962 to model concurrent systems. A Petri net is a graph with two types of nodes: places and transitions. In a Petri net diagram (refer Fig. 2.7), circular nodes represent places and bars represent transitions. There exist directed edges only from a place to a transition or vice-versa. If there exists an edge from a place p to a transition t, then p is said to an input place for t. Similarly, if there exists an edge from transition t to place p, then p is said to be an output place for t.

A Petri net provides a formal method to specify events and conditions. A transition is *enabled* only when a token is present in each of its input places. This models a condition in the system. An *event* is modeled by the firing of an enabled transition. A *firing* of a transition removes a token from each of its input places and puts a token in each of its output places. The movement of tokens generally enables other transitions. The execution of a Petri net evolves by the successive firing of enabled transitions. The state of the Petri net wherein tokens are located in different places is called a *marking*. The presence of a token in a place can be used to model the availability of a resource. Based on the resources originally present in the system, the initial marking of the Petri net is determined. The set of all markings that are reachable from the initial marking by firing none, one or more transitions is called the reachability set for that initial marking.

A Petri net can represent concurrency, decision making (when there is a conflict), synchronization and priorities. A transition that deposits tokens in two or more places could enable multiple transitions that can fire concurrently. Two or more transitions are called *conflicting* if they share a common input place (see Fig. 2.6(a)). The firing of one such transition disables the other(s). Conflicts can be resolved in a non-deterministic or probabilistic manner. A transition has to wait for firing until a token is present in each of its input places. This models synchronization in a Petri net (refer to Fig. 2.6(b)). The firing of a transition may be prevented by the presence of a token in another place (shown in Fig 2.6(c)). In such a case, the arc from this *inhibitor place* to the constrained transition is called an *inhibitor edge.* This property of a Petri enables the simulation of priority between two transitions.

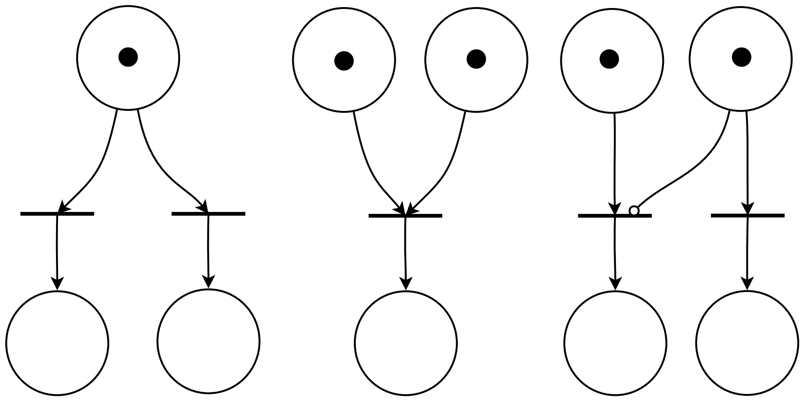

(a) Conflict (b) Synchronization (c) Inhibitor

Figure 2.6: Expressiveness of Petri nets

Petri nets with inhibitor nets are equivalent in expressive power to classical Turing machines. Without inhibitor nets, they generate a proper subset of context sensitive languages.

The transitions in a Petri net may be instantaneous or timed. Petri nets having timed transitions are called Timed Petri nets (TPNs). These transitions may have deterministic or stochastically distributed firing times. A Petri net whose transitions are instantaneous or exponentially distributed is called a Generalized Stochastic Petri Net (GSPN). GSPNs are a powerful tool in modelling discrete event systems because instead of assigning timings to each transitions, it is sufficient to associate exponentially distributed times with activities that are believed to have the maximum impact on the system [42].

A Petri net can elegantly model multiple resources, concurrency, synchronization, fixed or stochastic timed transitions in a system. Further, it is as powerful as a system of queues [33] and hence can be used for modeling complex computer systems.

However, using traditional Petri nets for such systems leads to a large increase in the number of places and transitions. In order to overcome this issue, the power of Petri nets can be combined with the flexibility of data types present in programming languages. This leads to the notion of colored Petri nets, where each token is assigned a color representing information associated with the token in a place. Colored Petri nets have the same expressive power as Petri nets [29]. Due to their structured expressions involving types (colors), colored Petri nets may be combined using well defined interfaces. They have been used to model a number of discrete event systems such as automated manufacturing systems [42], multiprocessor systems [22], network processing applications on IXP2400 [21] etc.

2.3.1 A Petri Net Example

Consider the Petri net shown in Figure 2.7 which models the work flow in a workshop with two workmen, *producer1 and producer2*, who *produce* items based on *orders* in the *OrderPool*. There is an *inspector* who checks the *items produced*. When an item is *rejected* by the inspector, its order is re-issued. If the item *passes* the quality check, it is considered *finished*. The presence of a token in *OrderPool*, indicates an order that has to be taken up for execution by the producers. Either of the workmen may take up the order, hence a token in *OrderPool* is consumed by either *Produce1* or *Produce2* transition. Both these transitions *conflict* as they share a common input place, *OrderPool*. The *Produce1* transition can fire only if there is a token in *Producer1* place (indicating that first workman is free) and a token in *OrderPool* (indicating that there is an order to be executed). This transition models necessary synchronization for the occurrence of this event. Both the workmen put in the items that are generated into a common pool called *ItemsProduced*. This phenomenon is called *merge*. The items have a p_i probability of being rejected and $1-p_i$ probability of being passed by the inspector. This is modeled by assigning the respective probabilities to the edges *ItemsProduced -Reject* and *ItemsProduced-Pass*. On a *Reject* transition, the corresponding order is put back into the *OrderPool* place. The item is moved to the *Finished* place if it passes the check. The transitions *Produce1*, *Produce2*, *Reject* and *Accept* are timed transitions whose durations represent the amount of time required to perform these actions.

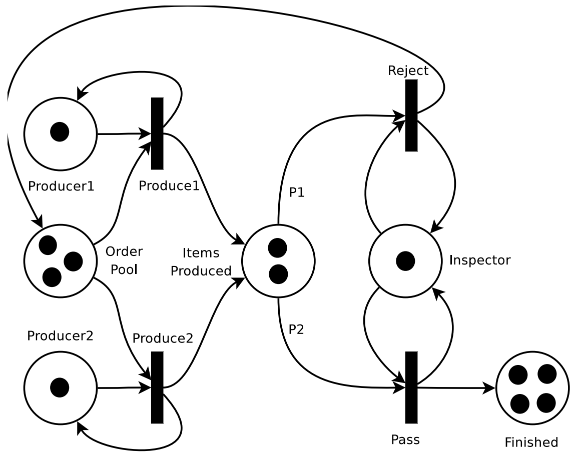


Figure 2.7: Modeling work flow in a workshop

IXP2400 has a multi threaded architecture to exploit substantial amount of packet level parallelism. A Petri net is a natural choice for modeling concurrency, synchronization and resource contention in network applications. Motivated by this observation, Govind et al. [21] used Petri nets for modeling a number of network processing applications on Intel IXP2400. This work highlights DRAM as a bottleneck resource and encourages us to look at the memory hierarchy of NPs in greater detail. We observe that DRAM access involves not only managing resource conflicts but also maintaining precise timing requirements between commands. These considerations drive us to build a detailed Petri net model of DDR SDRAM. This model enables us to study the resources and access patterns inhibiting higher throughput in packet buffering. We also use colored Petri nets to build a synthetic network traffic module that generates traffic with similar characteristics as those found in the internet core. This module is interfaced with a model of IPv4 forwarding application on the IXP2400 to compose a system modeling a router in a real scenario.

Chapter 3

Evaluating Packet Buffering Strategies

3.1 Introduction

On account of their low cost and high storage density, DRAM memories are used as packet buffers in routers. Their limited bandwidth is often the impediment in achieving higher packet processing rate [24]. Therefore improving the achievable DRAM bandwidth has a direct bearing on network processing application performance. From the architecture of DRAM memories, we see that the memory performance depends on a number of factors such as the memory access ordering, page policy, multi-banking and timing between DRAM operations. Apart from these, it is also important to consider workload characteristics such as packet length distribution in the internet which affects the amount of data stored in DRAM.

With a view to understand the complex interactions between network processing elements, workload characteristics and DRAM architecture, we developed a detailed Petri net model of IPv4 forwarding application on the IXP 2400. The DDR SDRAM Petri net that we build is independently validated against DRAMsim [43], an accurate DRAM simulator, before it is integrated with the IXP2400 Petri net model. The entire model is then validated with Intel IXA SDK [26]. Unlike earlier work [21], which recognized DRAM access latency as the cause for performance bottleneck, we identify narrow DRAM accesses, whose data transfer burst is not enough to hide the DRAM access latency, as one of the factors leading to declined DRAM performance. Also the resource utilization is highly dependent on the traffic conditions. Under real traffic conditions, the data bus connecting the DRAM with the ME impedes performance, but with a denial-of-service(DOS) traffic, the hash computational unit is the bottleneck resource. This observation clearly illustrates the fact that the worst case design of NPs leads to sub-optimal performance with real traffic.

In the next section, we describe our Petri net model for DDR SDRAM and the model for IP forwarding on IXP2400. The detrimental effects of narrow DRAM accesses, their causes and three packet buffer allocation strategies that improve network processor performance are discussed in Section 3.4. We report the performance results of the simulation and analysis of the workload on the NP structures in Section 3.6. We discuss the related work in Section 3.8 and conclude the chapter in Section 3.9

3.2 DDR SDRAM Petri Net Model

Our Petri net model considers in detail, the primary structures of DDR SDRAM such as the data bus, rows and columns of the memory array, page buffer, multiple banks, IO gating, address and command bus (see Figure 3.1). Unlike in a general purpose processor with caches, in an NP environment, the DRAM has to service requests of different sizes. Therefore the transmission time over the data bus depends on the size of the data. For the DDR SDRAM model we use the resource usage model proposed by Wang [44] which allows two accesses to be processed concurrently as long as they do not use a common resource. This generic access protocol enables us to model the various degrees of overlap associated with DRAM commands. The page buffers are managed with a *closed policy* as in IXP 2400. The presence of multiple resources is elegantly modeled using colored tokens.

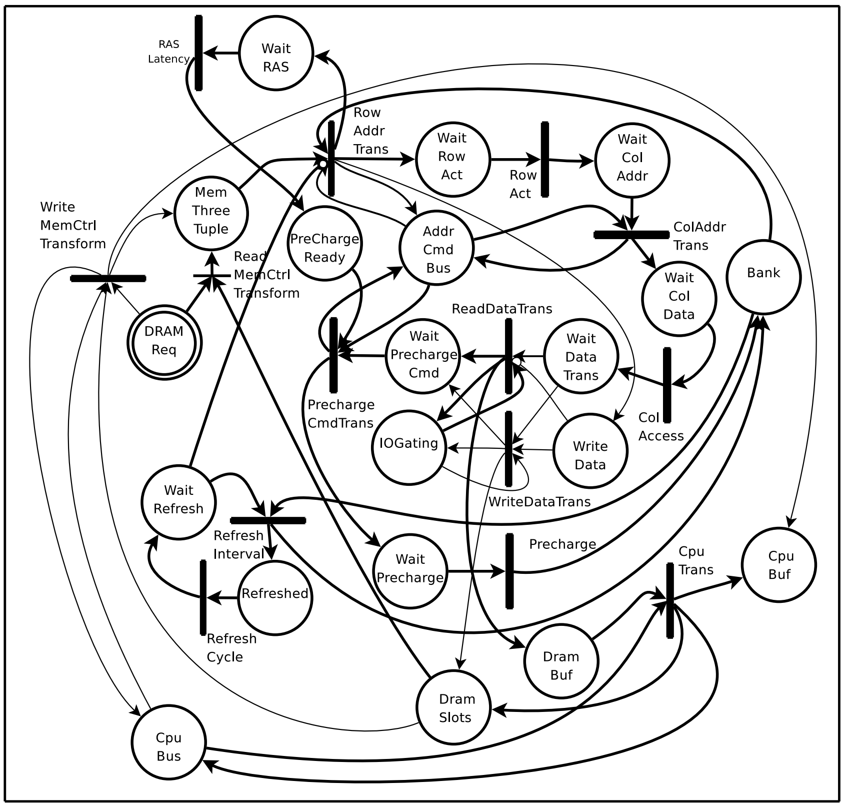


Figure 3.1: DDR SDRAM Petri net model

We describe the Petri net model for processing a read access to the DRAM. In the following discussion, the places and transitions in the Petri net are shown in *italics*. An instantaneous transition is represented by a thin line (for example *ReadMemCtrlTransform* in Figure 3.1) and a timed transition by a thick line (for example *RowAddrTrans* in Figure 3.1). In our model, each timed transition takes a fixed amount of time, determined by the respective activity. The availability of hardware resources such as data bus, IOgating, address and command bus is modeled by the presence of tokens in places like *CpuBus*, *IOGating* and *AddrCmdBus* respectively. Multiple banks are modeled by the presence of different color tokens in *Banks* place. Each DRAM access can proceed only when the corresponding bank is available. As a result, the Petri net model captures bank collisions.

Before we describe the model for DDR SDRAM in detail, we briefly recall that the steps involved in reading a chunk of data from this memory are: mapping the request to the corresponding bank, row and column; transfer of commands from the memory controller for row access, column access and precharge, and transfer of data over the data bus.

The transitions corresponding to these steps are described below and are highlighted with bold input and output arcs in Figure 3.1. A request is presented to the DRAM by placing a token in the *DRAMReq* place (shown by a double circle). The token contains the size of the access, type (read/write) and address of the DRAM request. From the address of the memory request, the bank, row and column numbers are identified by the memory controller based on the address mapping scheme. This action is modeled by the *ReadMemCtrlTransform* transition. The address mapping activity is performed by the memory controller.

When a bank is free, the row address along with the row activation command is sent over the address and command bus (*RowAddrTrans* transition). This transition models the row access step of DRAM access. After the row activation delay, modeled by *RowAct* transition, the row is read into the page buffer. This is captured by the presence of tokens in the *WaitColAddr* place.

The column address is sent to the DRAM device over the address and command bus. This activity corresponds to the column access step of DRAM access. The use of the address and command bus for *RowAddrTrans* and *ColAddrTrans* is modeled by arcs between the *AddrCmdBus* and these transitions.The data that is read is available for output after a fixed interval of time denoted by the *ColAccess* transition.

The data passes through the IOgating that connects the page buffer to the input-output pins (*ReadDataTrans* transition). The *CpuTrans* transition captures the transmission of data over the data bus to the MEs. The time interval for which the bus is used is determined by the size of the data being transferred. The request is completed by placing an output token in *CpuBuf* place.

Since we model a closed page policy, the bank is precharged following the row access. After the transfer of data through the IOgating, the row is precharged. These actions are captured by *PrechargeCmdTrans* and *Precharge* transitions. The minimum delay between the row access command and precharge within a bank is modeled by *RASLatency*. Following the precharge, the bank can be used to service the next access.

The delays between these commands are determined by the DRAM access protocol. The memory controller can service only a fixed set of DRAM accesses simultaneously. This is modeled by the presence of fixed number of tokens in the *DRAMSlots* place.

The DRAM memory is refreshed at regular intervals of time which is modeled by *RefreshCycle*. During refresh, row activation is inhibited by the presence of a token in *WaitRefresh* place. The DRAM does not accept any requests for *RefreshInterval* transition time.

The parameters used for the DRAM simulation are shown in Table 3.1 and are as given in the data sheet [1]. The command cycle is the time for which the command bus is used to transmit a command from the memory controller to the DRAM. Row activation time is the time between the row address being placed on the address bus and the data moving to the sense amps. The time to restore the charge from the sense amps into the DRAM cells is called the read to precharge delay. The row may be precharged after this interval. The time taken to place the required data on the data bus after the issue of the column access command is called the column activation delay. The time to precharge a row is called the precharge delay. The banks have to be refreshed once every refresh cycle. The time taken to refresh each row is called to the refresh interval. The time interval for which the data is transferred on the data bus is called the burst time and is determined by the amount of data transmitted. The DRAM cycle time is 100MHz and data bus operates at a rate of 200MHz.

Our Petri net model, models in detail all the above aspects of DRAM access with closed page policy. The data transfer time through the data bus, contention for banks, address bus are elegantly modeled by the Petri net model.

DRAM activity	DRAM cycles
Command Cycle	1
Row Activation	3
Column Activation	3
Read to Precharge delay	2
Precharge	3
Refresh Cycle	6.4e6
Refresh Interval	15

Table 3.1: Parameters for the DRAM model

3.2.1 Validation of DRAM Petri Net Model

We use CNET Petri net simulator [48] to obtain performance metrics from the model. This simulator provides a rich set of features to model Petri nets including colored tokens, timed transitions and priority based transitions.

In contrast to analytical modeling, simulating a Petri net model for performance evaluation of the DRAM has the following advantages:

1. It helps us to obtain performance metrics such as DRAM throughput and utilization of various resources in the DRAM.

2. Analysis of large Petri nets is complex.

3. The modular design of using colored Petri nets allows us to embed the DRAM model in a Petri net model for IPv4 packet forwarding model in IXP 2400 [21].

We validate our Petri net based DDR SDRAM model with the DRAMsim [43] simulator which accurately models DDR SDRAM. DRAM access traces of the IP forwarding application on IXP2400 simulator and the IPSec application on IXP2850 simulator of the IXA SDK are used as inputs to both the simulators. The accessed data size is fixed at 64 bytes as DRAMsim cannot vary the size of accesses within a trace. Table 3.2 shows the throughput observed from both models. The Petri net model predicts throughput within 4% of the throughput predicted by DRAMsim. DRAMsim only supports DRAM

Application trace	DRAM throughput in Gbps	
	DRAMsim	Petri net model
IPSec	6.66	6.39
IPv4 forwarding	7.29	7.23

Table 3.2: Comparison of DRAM throughput

accesses whose size is equal to the cache line size. However, our DRAM Petri net model can support accesses of different lengths and is suited for modeling NP applications.

3.3 IPv4 Forwarding Petri Net Model

In this section we describe a colored Petri net model of IPv4 forwarding application running on IXP2400. A packet arriving at the ingress port is received by the interface card and stored in the Receive Buffer (RBUF). A parallel threads model (see Section 2.1.2) is used to process the packets, where each thread performs all tasks related to network processing. The thread assigned for packet processing allocates memory in the DRAM and moves the mpackets into the off-chip memory. During processing, the header is read into the registers. The time-to-live field is decremented and the destination address is used to hash into the lookup table entry in the SRAM. This provides the output interface for the next hop. The modified header is written back into the DRAM after recomputing the checksum. The packet is then enqueued in the transmit buffer. The control words for the transmit mpackets are suitably updated. This processing flow has been used in [39]. A Petri net model for IPv4 forwarding with the above processing steps was developed in [21].

3.3.1 Modifications to IPv4 Forwarding Petri Net Model

We modify the previously developed model to enable us to model the DRAM buffering accurately. Each token corresponding to a packet carries information about its allocation address. This allows us to model precisely the bank conflicts during DRAM access.

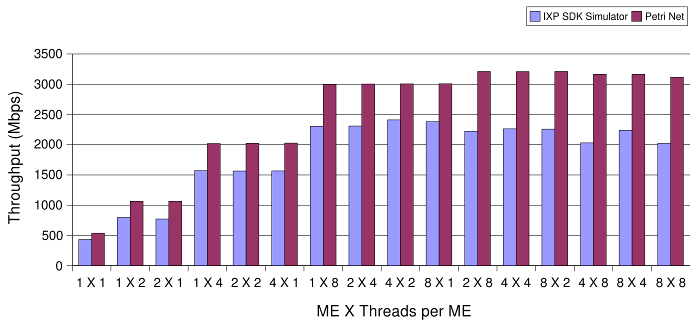
Unlike the model in [21], the Petri net model handles packets of different lengths. We model the cell based interface for receiving and transmitting packets (described in Section 2.1.3), the SRAM and scratchpad memories and the buses connecting them to the MEs. The DRAM model described in Section 3.2 is integrated with the IXP2400 model. The IXP2400 uses middle order interleaving to spread the accesses evenly across banks [25]. Thus, two consecutive 64 byte chunks are distributed to adjacent banks. High order interleaving scheme was proposed in [24] with a view to increase the locality of accesses to the most frequently used page buffer. Since a number of threads access the DRAM concurrently, we see that high order interleaving does not overload a single bank of the DRAM. All the four banks are observed to have the same utilization. Hence we use high order interleaving in our model.

3.3.2 Validation

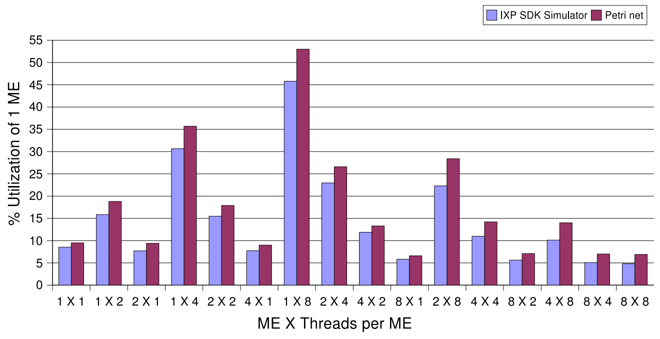
The Petri net model for IPv4 forwarding is validated with the IXP simulator available with the IXA SDK. We compare the ME utilization and the throughput metrics obtained from both the models.

As in [39], a stream of 64 byte packets is used to simulate the traffic pattern observed under denial of service(DOS) attacks. For each configuration, the rate of input traffic is the same as the maximum throughput supported by the NP in that configuration.

The throughputs obtained for various configurations of the MEs and threads per ME are shown in Figure 3.2(a). Figure 3.2(b) shows the ME utilization obtained with the IXA SDK simulator and with the Petri net model. We observe the trends in the throughput between IXA SDK simulator matches with the throughput measured from the Petri net model. The models differ in the peak throughputs achieved and the number of threads used to achieve it. A similar difference was observed in [21]. More specifically, in IXA SDK a maximum throughput of 2410 Mbps was achieved with 8 threads and after that the throughput drops marginally. This is because, the IXP implements back pressure on the MEs when the DRAM request queue length is above a certain threshold [25]. When the number of outstanding DRAM accesses crosses this threshold, 10 in case of



(a) Network processor throughput



(b) Comparison of micro engine utilization

Figure 3.2: Validation of Petri net model

IXP2400, the DRAM controller sends an *almost-full* signal to the MEs, as a result of which further requests from this ME are inhibited. This reduces the rate of execution of the MEs, which in turn reduces the requests to the DRAM. In [21], the authors show that backpressure reduces the throughput realized from the NP as it stalls the entire ME and inhibits operations such as SRAM access or hash computation that could have proceeded in parallel. Given that back pressure reduces the throughput of the NP, we did not implement that in our model. Therefore, we see that the throughput saturates at a higher throughput of 3210 Mbps and saturates beyond 16 threads.

Further, unlike in the IXA SDK and the model in [21], our DRAM Petri net model is more detailed in modeling DRAM accesses. This DRAM model is independently validated against a cycle accurate DRAM simulator before it is integrated with the IPv4 forwarding Petri net model. As a result, we do not expect the throughput obtained from the IPv4 Petri net model to closely match the throughput from IXA SDK model. The overall trends in the throughputs from the two models is the same. The utilization of the MEs follows the same patterns in the IXA SDK and the Petri net model. The Petri net model consistently reports a slightly higher utilization of the MEs. This is expected as the Petri net model does not implement backpressure and therefore enables higher utilization of the MEs.

After the validation of our Petri net model, we use it to evaluate the hardware structures in the NP and to identify the bottleneck resources in the memory hierarchy of the NP.

3.4 Packet Buffer Allocation Schemes

In this section, we first explain the detrimental effect of narrow accesses on DRAM bandwidth and quantify its impact. We then identify the causes for narrow accesses and propose allocation schemes to alleviate their negative impact.

3.4.1 Why Do Narrow Accesses Affect Bandwidth?

An access to the DRAM involves RAS, CAS and precharge steps, in addition to data transfer (refer to Section 2.2). While the latencies involved in RAS and CAS steps, viz t_{RCD} and t_{CL}, are fixed, the data transfer time is proportional to the size of the access. Note that, when the banks are free, the RAS steps (similarly in CAS step) corresponding to accesses for two different memory banks can be overlapped (as shown in Figure 3.3) except for the duration for which the address bus is used. When successive accesses to different memory banks transfer "sufficient" data, these accesses can be overlapped as shown in Figure 3.3, such that the data bus is fully utilized. In such a situation, the delays of RAS, CAS and precharge latencies are not exposed as consecutive memory accesses overlap. For the DDR SDRAM, such a situation occurs if the accesses are for 64 bytes or larger. We refer to these accesses as wide accesses and other accesses less than 64 bytes as narrow accesses. Figure 3.4 shows data transfers when some of the accesses are less than 64 bytes. In this situation, even though the data transfer for the first access is complete at time step 11 and the data bus is free, the transfer from the second access does not start as the CAS step is not complete. This results in under utilization of the data bus, which in turn leads to lower DRAM bandwidth.

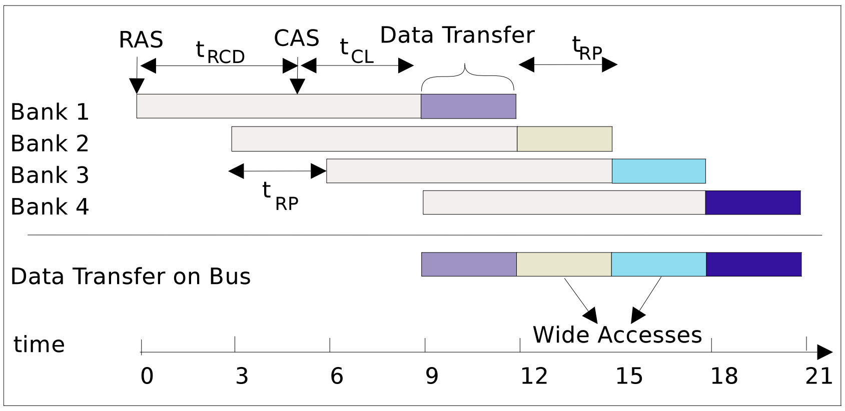


Figure 3.3: Wide DRAM accesses

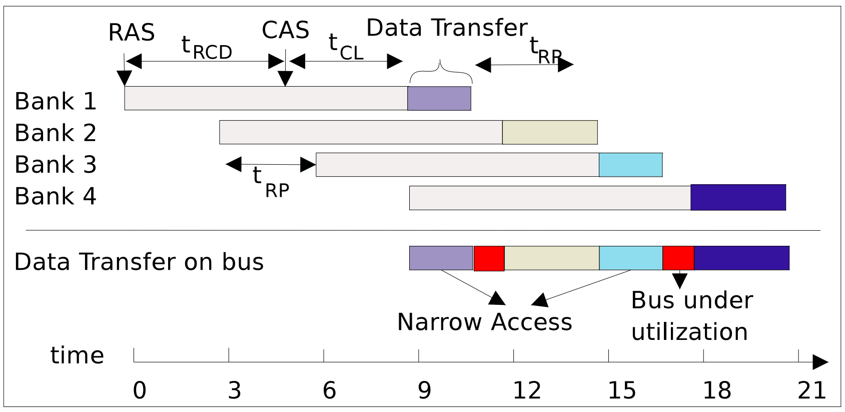


Figure 3.4: Narrow DRAM accesses

This raises the following two questions, which we address later in the chapter.

1. To what extent are narrow accesses present in IP forwarding applications in NP under real traces?
2. What is the impact of these narrow accesses on DRAM bandwidth?

3.4.2 Extent of Narrow Accesses in Network Applications

With 64 byte packet traffic, writing and reading the packet from the DRAM buffer transfers 64 bytes each. However, read and write of the packet header for IP lookup involves a narrow access. As a result, we find that 50% of the accesses to DRAM are narrow accesses.

The packet distribution in the internet leads to narrow accesses to packet buffer. In our study, we consider packet lengths in three real traces obained from the NLANR project [2]. PSC (PSC-1135643836-1) and FRG (FRG-1133697651-1) are edge traces whereas AMP (AMP-1124777370-1) is a core trace[1]. We study edge and core traces, as the traffic characteristics at the two sites are different. Figure 3.5 shows the packet lengths and their distribution in the three traces considered. Packet length is observed to be bimodal with modes around 40 bytes and 1500 bytes [38]. We observe that packets of length less than 64 bytes constitute about 42% of the total number of packets in the PSC trace. In the FRG and AMP traces, these numbers are 33% and 20% respectively. Such small packets lead to narrow accesses to the DRAM during both packet buffering and header access. We find 30% of the accesses to the DRAM are narrow under real traffic conditions.

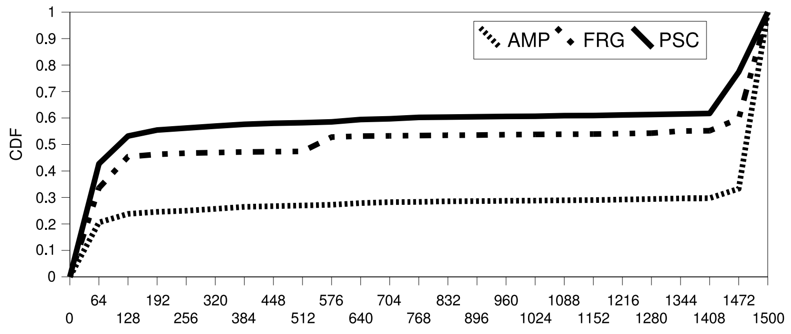


Figure 3.5: CDF of packet length distribution

[1]An edge trace is one that is collected at routers that typically connect a WAN, such as a university or large organization, to the internet. A core trace is collected at exchange points between large networks.

3.4.3 Impact of Narrow Accesses on DRAM Bandwidth

As mentioned earlier, DRAM bandwidth is critical to achieve higher processing rates in NPs. Using our Petri net model for DRAM, we evaluate the DRAM bandwidth when certain part of the traffic results in narrow accesses. In this experiment, we use a trace with read and write accesses of two different sizes, 64 byte accesses which utilize the data bus fully and 32 byte accesses which are narrow accesses. The trace has equal number of read and write accesses which are randomly distributed across banks. For different mixes of 64 and 32 byte accesses the DRAM bandwidth achieved is shown in Figure 3.6. It is observed that the bandwidth can reduce by up to 15% when 50% of the accesses are

% of 64 byte accesses	% of 32 byte accesses	DRAM bandwidth (in Gbps)	Bandwidth reduction
100%	-	10.03	-
70%	30%	9.33	6.97%
60%	40%	8.96	10.67%
50%	50%	8.55	14.75%

Figure 3.6: Effect of narrow accesses

32 bytes. With 30% narrow accesses, as in real traces, the bandwidth reduces by 6.97%. The bandwidth reduction would be higher if the data access is narrower.

In comparison, when DRAM is used in a general purpose processor, the size of memory access is fixed to the size of L2 cache line size. Also, latency of access, rather than bandwidth, is the critical factor in determining the performance of the processor. Due to this, memory access reordering [35] and burst interleaving [1] mechanisms are used to reduce latency. In contrast, higher latency in DRAM access can be tolerated as long as high bandwidth access to the packet buffers is achieved. Since DRAM bandwidth affects the throughput of the NP [24], we focus on preventing narrow accesses, which are detrimental to performance.

3.5 Buffering Schemes

As narrow accesses reduce the DRAM memory bandwidth, which in turn affects the the NP performance, we propose three buffering schemes which reduce the extent of narrow accesses in a packet forwarding application in NP. Since mpackets belonging to the middle of the packet are 64 bytes long, they require wide accesses and are stored in the DRAM. In the proposed schemes, we modify the buffering for the first mpacket and the tail portion of the packet.

1. *Header buffering (HB):* In header processing applications, the header is read from the DRAM after buffering the packet. In case of IP forwarding, the header is 20 bytes. Accessing it, results in a narrow access of 24 bytes [2]. We observe that the packet header is accessed multiple times and exhibits certain amount of locality. Thus storing the header in a faster on-chip memory such as the scratchpad memory, not only improves the access latency of packet header, but also prevents narrow accesses to DRAM which have an adverse impact on the bandwidth. Since the header occupies limited space, it can be stored in scratchpad memory. Govind, et al. [21] propose storing the header in the SRAM which is similar to our HB scheme. However the focus of their study is to only reduce the fully saturated DRAM utilization.

2. *First cell buffering (FCB):* Even with *header buffering*, there are still a number of narrow accesses. Accessing the remaining part of the first mpacket from the DRAM leads to a narrow access, especially if the packet itself is of small size($\leq 64 bytes$). For example, when header buffering is used for IP forwarding, the first 24 bytes are stored in the scratchpad memory. The remaining 40 bytes required to form the first cell of the packet would still lead to a narrow access. This can be overcome by storing the first cell in scratch memory. With FCB, the first and only mpacket of packets less than 64 bytes and the first mpacket of larger packets are entirely buffered in scratchpad memory. From the packet length distribution of the real

[2]An access to DRAM is always in multiples of 8 bytes

traces (refer Figure 3.5), it can be seen that 30 to 50% packets are stored in on-chip scratchpad memory. Note that the first mpacket can be easily identified as they have the SOP bit set(refer to the discussion in Section 2.1.3).

From our Petri net model, we observe that throughput saturates with 16 threads. Conservatively, even if we assume that 32 free threads are used in the free pool of threads and each thread is statically assigned 64 bytes in scratchpad for packet buffering, this scheme requires only 2KB of memory. This requirement can be easily met by Intel IXP2400 which has 16KB of scratchpad memory. Note that buffering all the mpackets in scratchpad memory is not a viable option as the storage required may exceed the scratchpad memory size.

3. *First cell buffering + Tail buffering (FCB+TB):* The other source of narrow accesses to the DRAM is the tail portion of the packet(shown in Figure 2.2). In case of ethernet traffic, packet length can vary from the minimum size of 40 bytes to the maximum length of 1500 bytes. As the length of a packet need not be a multiple of the cell size, the last part of the packet usually results in a narrow access to the DRAM. Thus we propose buffering the tail portion of a packet in the on-chip scratchpad memory along with the first cell. The last mpacket of a packet can be easily identified as it has the EOP bit set(refer to the discussion in Section 2.1.3). The tail portion of the packet occupies a bounded amount of space and buffering it in the scratchpad memory along with the first cell requires twice as much memory as FCB.

3.5.1 Remarks

The allocation information regarding the storage of different parts of the packet in various levels of the memory hierarchy must be stored during packet processing. For this purpose, two additional fields are added to the packet descriptor. These fields point to the first cell and tail portion of the packet in scratchpad. The amount of data stored in these portions can be derived from the packet length information in the packet descriptor.

For FCB+TB, each thread requires a bounded amount of scratchpad memory. This is equal to twice the mpacket size, 128 bytes in our case. The scratchpad space allocated for buffering is divided into fixed sized segments. The segments can be dynamically assigned to each thread to ensure maximum utilization of scratchpad memory. A key observation about our schemes is that the reported performance benefits are achieved by enabling greater utilization of existing resources and without the use of any additional hardware to the base IXP2400 scheme. Further, in the above approach other parts of the memory hierarchy, viz. scratchpad, are effectively used to improve the throughput.

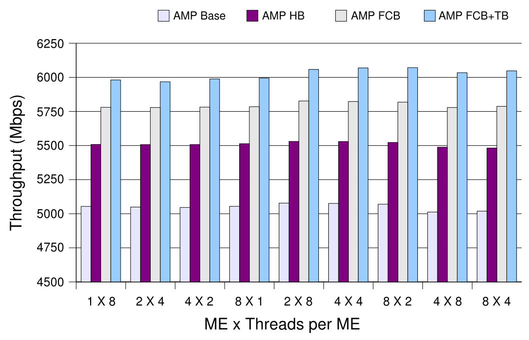
3.6 Improvement in Throughput

In this section the performance improvement achieved from the proposed schemes are quantified. Further, the Petri net model is used to identify the performance constraining resources in the memory hierarchy.

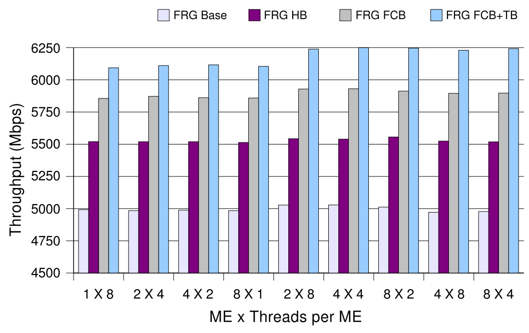
3.6.1 Impact of Buffering Schemes

The packet lengths from real traces (AMP, FRG and PSC) are used as input for the simulation. The lengths of first cell and tail portions of the packet that are buffered in on-chip scratchpad memory is known from the packet length (refer Section 2.1.3). According to the buffering scheme, the header, the first cell and the tail portion of the packet are buffered in the on-chip or DRAM memory. The Petri net simulation is run with different configurations with varying number of MEs and threads per ME. For all buffering schemes, we observe that the throughput increases when the number of threads increases from 1 to 8. Since we intend to study the effect of network processing on memory hierarchy at higher packet forwarding rates, we report results only for 8 or more threads.

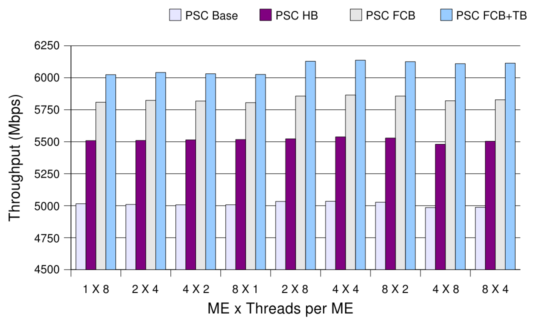
Figure 3.7 shows the throughput improvement due to various buffering schemes. In these graphs, the x-axis represents the total number of threads used, represented as the products of number of MEs used and the number of threads active in each ME. For each



(a) Throughput with AMP trace



(b) Throughput with FRG trace



(c) Throughput with PSC trace

Figure 3.7: NP throughput with AMP, FRG and PSC traces

configuration we report the throughput obtained by the base scheme(where all packet are completely stored in DRAM) and *HB, FCB and FCB+TB* schemes where some parts of the packet are buffered in scratchpad memory. As can be seen from Figure 3.7, the throughput with some of the traces exceeds 6000Mbps.

Header buffering(HB) leads to an average throughput increase of 9.08%, 10.65% and 10.01% over the base scheme for AMP,FRG and PSC traces respectively. *First cell buffering(FCB)* increases throughput over header buffering by 5.19%, 6.54%, 5.76% for AMP, FRG and PSC traces respectively. *FCB+TB* improves over *FCB* by 3.93%, 4.94% and 4.30% for AMP, FRG and PSC traces. Cumulatively, we achieve significant throughput improvement over the base scheme. When using *FCB+TB*, throughput improvement over the base scheme is 19.26%, 23.37% and 21.35% for the AMP, FRG and PSC traces. The reduction in narrow accesses to DRAM coupled with distribution of data to different levels of the memory hierarchy contribute to the increase in throughput.

Trace	Bandwidth in Gbps		Bus utilization	
Allocation	Base	FCB+TB	Base	FCB+TB
AMP	10.13	10.50	97.6%	99.1%
FRG	10.08	10.49	97.6%	99.1%
PSC	10.07	10.50	97.3%	98.8%

Table 3.3: Bandwidth realized by the DRAM

In order to understand the effect of reduction in narrow accesses, the bandwidth realized by the DRAM under the base scheme and with *first cell buffering + tail buffering (FCB+TB)* are measured (refer to Table 3.3). For the three traces, the bandwidth realized from the DRAM improves by up to 4%. This has contributed to some of the improvement in throughput achieved by the *FCB+TB* scheme. The remaining increase in throughput is attributed to the distributed allocation of packets, which improves the parallel utilization of resources. Further, with *FCB+TB*, a peak throughput of 6.25 Gbps is achieved with 16 threads. In contrast, the throughput saturates at 5.05 Gbps in the base scheme. An overall improvement of nearly 20% is observed in the throughput of the NP. The data bus utilization increases from 97.4% to 99%.

3.7 Utilization of NP Resources

Our Petri net model allows us to study the utilization of various resources within the NP and DRAM. Such a study helps us to identify the performance constraining structures in packet buffering.

3.7.1 Resource Utilization with Real Traffic

We study the utilization of resources in the NP with traffic from real traces. Packet buffering with *first cell buffering + tail buffering (FCB+TB)* allocation is considered.

Table 3.4 presents the utilization of data bus, DRAM banks, MEs and hash unit observed for a configuration with 4 MEs and 4 threads per ME. ME utilization is the

NP component	AMP	FRG	PSC
DRAM Banks(4)	72%	72%	72%
Data Bus(64 bit)	99%	99%	98.8%
Hash Unit	14.8%	18.5%	16.2%
Single ME	16.0%	16.8%	16.2%
Throughput(in Mbps)	6069	6249	6137

Table 3.4: NP resource utilization with FCB+TB buffering

fraction of time a thread in the ME is busy with computation. Bank utilization is the average utilization of the DRAMs banks. The bank utilization is observed to be uniform for all banks.

With real traces, the utilization of the bus reaches 99% at a throughput of around 6200Mbps, which is almost twice that observed with 64-byte sized packets (refer Table 3.6). Thus for real traces, we observe that the data bus is the bottleneck resource. Two sets of simulations were performed to substantiate this claim. In one experiment, the bus width is doubled keeping the number of banks equal to 4. In the other experiment, the number of banks is doubled with the bus width of 64bits. Table 3.5 shows the throughput with *FCB+TB* scheme, for different configurations when real traces are used as input. The throughput with base IXP configuration is shown in the first row. When the bus width is increased to 128 bits, keeping the number of banks constant, the throughput is observed to increase by 33%. However, with a 64 bit bus and 8 DRAM banks, the throughput increases by only 1%. This clearly shows that the data bus connecting the off-chip DRAM to the MEs is the bottleneck in achieving higher throughput. It also explains the moderate increase in bus utilization when using FCB+TB.

Bus width	Banks	AMP	FRG	PSC
		Throughput in Mbps		
64 bits	4	6069	6249	6137
128 bits	4	8063	8228	8156
64 bits	8	6124	6303	6198

Table 3.5: DRAM configurations

Bus width	Banks	Throughput (in Mbps)	Hash Unit Utilization	Bus Utilization	DRAM Bank Utilization	Single ME Utilization
64 bits	4	3210	92.8%	88.8%	82%	14.2%
128 bits	4	3460	100%	47.8%	75%	15.4%
64 bits	8	3460	100%	95.7%	47%	15.5%

Table 3.6: NP throughput in Mbps with 64-byte sized packet traffic

3.7.2 Resource Utilization with DOS Traffic

Table 3.6 shows the throughput and utilization of various NP resources with base allocation scheme for the DOS trace. The number of banks and data bus width are varied. It is seen that for such a traffic, the hash unit becomes the bottleneck resource. There is no significant improvement in throughput when the bus capacity is doubled or the number of banks is increased from 4 to 8. In fact, the utilization of these resources decreases by roughly 50%. The ME utilization and throughput, increases only marginally as the number of banks or data bus width is doubled. In all these configurations, the hash unit utilization is close to 100%, making it the bottleneck resource. This observation is in line with the result shown in [21].

From Table 3.4 and Table 3.6, we see that the utilization of NP resources, such as the hash unit, ME and data bus, varies widely with different traffic traces. The throughput of the NP is observed to be highly dependent on the input traffic. Traffic consisting of 64 byte packets, as in a DOS attack, requires greater compute resources. However if the traffic is similar to real traces, then greater throughput can be achieved by having multiple channels. Provisioning processing and IO capacity for a network application must consider these traffic conditions.

The disparate resource usage also supports the argument that the worst case traffic must to handled separately. High utilization of compute resources like the hash unit may be used to flag DOS attacks. By efficiently eliminating DOS traffic, the common traffic can be processed at higher rates. We leave the efficient recognition of DOS attacks to future work.

3.8 Related Work

Since packet buffering in DRAM is a bottleneck in router applications, a number of studies have focused on this problem. Hasan, et al. [24] study the issue in the context of IXP 1200 network processor with SDRAM packet buffer. They propose opportunistic mechanisms such as locality aware packet buffer allocation scheme and batched access algorithms in order to increase page hits. They also propose a using an open page policy to prevent page buffer misses. However, when the number of threads accessing the DRAM increases, it is difficult to maintain page locality. We observe that page misses are harmful only when the latency of the miss is exposed by narrow accesses. We try to reduce such accesses to DRAM by directing the narrow accesses to scratchpad. Improving row access locality provides performance benefit only when the data bus connecting the DRAM to the MEs is not a bottleneck. Our simulations show that the data bus is an impediment to achieve higher throughput. By keeping the most frequently used part of the packets in on-chip memory, we try to reduce the amount of data transferred on the bottleneck data bus.

Iyer, et al. [27] propose the use of an SRAM to store the entire packet before buffering it in the DRAM. Their scheme uses a large number of banks, a wide bus and look ahead mechanism to schedule requests to DRAMs, but does not try to improve the utilization of available resources. This proposal does not try to improve the utilization of the resources and is suitable for high end routers where guaranteed access to packet data is essential. In contrast, our proposal tries to maximize the utilization of existing resources to improve throughput.

The proposals in [20, 7] implement complex memory controllers for packet buffering in high end routers. These schemes implement look ahead, memory reordering or universal hashing mechanisms in the memory controller to provide packet buffer access guarantees. We propose distributing packet storage to improve utilization of resources. Our schemes require no changes in existing memory controller policy.

A Petri net model is used for evaluation and design space exploration of NPs in [21]. The study shows that accessing DRAM is a performance bottleneck. However the DRAM model used was approximate. Bank conflicts were modelled using a probabilistic approach. Also, the databus, row and column access was not modeled in [21]. In contrast, we developed a detailed model of SDRAM to study packet buffering issues. Narrow accesses to DRAM are identified as one of the factors leading to lower DRAM bandwidth in realistic traces where packet sizes vary.

3.9 Conclusions

In this chapter we studied the impact of DRAM performance on Network Processors especially for a header processing application like packet forwarding. Towards this end, we first developed a detailed Petri net model for DRAM accesses which modeled various aspects of the memory such as precharge latency, row and column delays, page buffers with closed page policy and data bus transfers. The DRAM Petri net model was validated with a cycle accurate DRAM simulator [43]. This model was then incorporated into a Petri net model for packet forwarding application on IXP 2400, developed in [21].

Our detailed DRAM model helped us to identify narrow accesses (accesses less then 64 bytes) as the cause for decrease in DRAM bandwidth, which in turn results in lower NP performance. To address this issue, we proposed three packet buffering schemes. These schemes helped to improve NP throughput by 20% and helped to achieve a throughput of 6250Mbps. We showed that the data bus connecting the DRAM to the MEs, impeded performance when using real traces. Also the effect of different workloads on network processor resources was reported.

Chapter 4

Buffering for Core Routers

4.1 Introduction

Network processors buffer packets to accommodate variations in network traffic and link utilization and to avoid packet losses in such scenarios. The size of the buffer needed to hold the network packets has been a topic of research in recent times [9, 16]. A widely used rule of thumb states that the buffer size required is equal to *round trip time* $*$ *transmit rate*, has been challenged. Appenzeller et al. [9] have shown that for core routers, the buffer size can be significantly reduced due to the statistical multiplexing of a large number of flows. The buffer requirement could be as small as 1% of the initial size suggested by the above mentioned rule of thumb, while still keeping the link utilization above 98%. It significantly reduces the memory requirements and hence the cost of the router. This result leads to a number of interesting possibilities for the architecture of network processors.

We observe that in a core router, the links are over-provisioned. As a result of which the link utilization in low during most regimes of network traffic. The reduction in utilization of the output links can be used to substantially reduce the buffer requirement. Further, we have shown in Chapter 3 that buffering packets in the DRAM is a performance bottleneck due to the large latency of accessing the DRAM banks and limited bandwidth of the data bus. The above observation on over provisioning of links and reduced memory requirement indicate that we need only a small amount of memory to buffer packets during processing. For this, we propose to use the on-chip receive and transmit buffers in the IXP 2400 for storing the network packets whenever there is no build up of packets in the network processor. However when there is an increase in the number of outstanding packets, we selectively buffer the packets in DRAM memory. This avoids the overflow of receive and transmit buffers and hence reduces packet drops. At the same time, it reduces the DRAM requirement and the cost of the router. This is possible because in network processors the buffering of packets is under software control. This programmable nature enables the processing threads to switch between DRAM buffering and direct transfer between the receive and transmit buffers. Such a scheme also mitigates the DRAM bottleneck and enables processing of packets at higher rates.

We show that with such a dynamic buffering scheme, the throughput supported by the network processor increases and the packet drop rate decreases. For evaluating the buffering strategy proposed, we use network traffic with similar characteristics as observed in the core of the internet. This effectively captures the behavior of the system under real workloads. Further, we explore how the system parameters such as the number of processing threads and the backlog to infer congestion (high water mark) affect the throughput and packet drop rate metrics.

The remaining part of the chapter is organized as follows. In the next section, we explain the theory from [9] which establishes that a small buffer size suffices for core routers. Section 4.3 describes our dynamic packet buffering scheme which uses different memory structures in the IXP 2400 instead of storing all the packets in the DRAM memory. The Petri net models for traffic generation and the dynamic packet buffering scheme are discussed in Section 4.4. Performance evaluation results are presented in Section 4.5. We present the current research in the related areas in Section 4.6 and conclude the chapter in Section 4.7.

4.2 Buffer Sizing for Core Routers

Packet buffers are needed in routers to absorb bursts of packets and to store packet during processing. We first explain the dynamics of TCP's congestion control algorithm, which leads to the result that the buffer size is equal to the product of the bandwidth and the round trip time. We then explain the observations by Appenzeller et al. [9] who propose smaller buffer sizes.

The TCP congestion control algorithm is designed to probe the network for greater capacity in case there is no congestion and to rapidly decrease sending rate when congestion occurs in the network. We explain the working of TCP congestion control algorithm and how it determines the buffer requirement.

4.2.1 TCP Buffer Requirement with a Single Flow

We briefly explain the theory behind buffer sizing model. In order to make the discussion simpler, we first describe the buffer requirement when only one flow is passing through the router as shown in Figure 4.1.

Consider the scenario in Figure 4.1 with a single TCP host, sending constant sized packets to the destination through a router. The output link of the router to the destination is the bottleneck link and has a data rate of C. The router has a queue of packets that have to be forwarded on the output link.

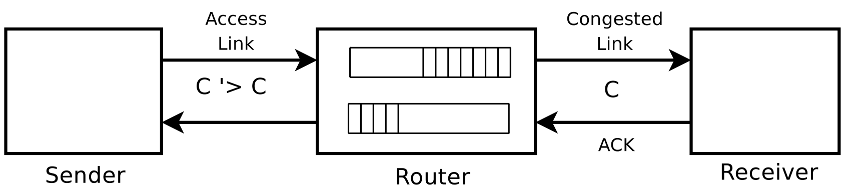


Figure 4.1: Single TCP flow through a congested link

TCP uses a sawtooth congestion control algorithm in the Additive Increase Multiple Decrease (AIMD) phase of the transmission. The sender maintains a counter W_{max} called congestion window, which is the number of outstanding packets sent on the network without receiving an acknowledgement. During the AIMD stage, TCP sender increases the congestion window by $1/W_{max}$ for every acknowledgement that it receives. Such a mechanism saturates any buffer until a packet drop occurs. On such an event, the sender infers congestion in the network and halves its congestion window to $W_{max}/2$. Since the sender has more outstanding packets than its congestion window, it pauses packet transmission and waits for acknowledgements for the $W_{max}/2$ outstanding packets. The packet buffer at the routers must be sized such that the output link is kept completely utilized during this period. This is the distance between the peak and trough $(t_2 - t_1)$ in Figure 4.2.

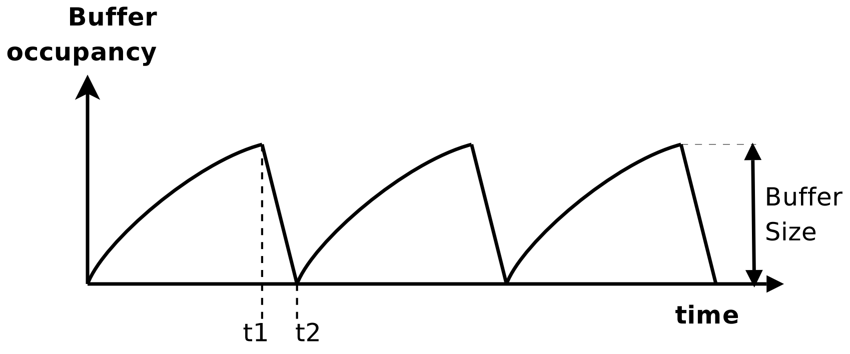


Figure 4.2: Buffer requirement for a single TCP flow

Assume the sender has sent W_{max} packets when the first packet drop occurs. As a result, it reduces its congestion window to $W_{max}/2$. Since there are more outstanding packets than the congestion window, the sender pauses until $W_{max}/2$ acknowledgements have arrived. Since the rate of acknowledgements is determined by the bottleneck link rate, $W_{max}/2$ acknowledgements arrive in $(W_{max}/2)/\mathrm{C}$ time. Also, a router buffer of size B will drain in time B/C. The buffer will not go empty if B/C $\geq W_{max}/2/C$; i.e.,

$$B \geq \frac{W_{max}}{2} \tag{4.1}$$

Consider the situation when the sender has resumed sending packets. In order to keep the bottleneck link completely utilized, the sender has to send packets at rate C. TCP rate R is given by

$$R = \frac{congestion\ window}{\overline{RTT}} \tag{4.2}$$

Therefore, we have

$$C = \frac{(W_{max}/2)}{\overline{RTT}} \Rightarrow \frac{W_{max}}{2} = C \ * \ \overline{RTT} \tag{4.3}$$

From equation 4.1 and 4.3 we get,

$$B \geq C \ * \ \overline{RTT} \tag{4.4}$$

When a number of large flows pass through a congested link, there is synchronization among them [36, 18, 46], i.e., they experience congestive losses at the same instant. The buffer requirement waveform for the individual flows are now "in phase". Due to this, the buffer requirement is the sum of the differences between the peak and trough of the individual flows. This is the same as the product of the aggregate bandwidth and average round trip time. Previous studies which used simulation [41] have also concluded that the buffer requirement should be the product of bandwidth and round trip time[1] in order to maintain full utilization of bottleneck network links. This study uses a model with up to eight long lived flows over a 40Mbps link to arrive at the buffer requirement.

However, in case of core routers, a number of factors differ from the assumptions made in the studies above, namely, the number of flows is much larger and the links are not congested. This is because internet links in the core are over provisioned in order to handle traffic bursts due to link failures.

4.2.2 Reduced Buffer Requirement with Multiple Flows

We now explain the reasons delineated by Appenzeller et al. [9] for reduced buffer requirement when the output links are uncongested.

In a core router, the start time and the propagation delay of the individual flows (t_{pi})

[1]Henceforth referred to as the Bandwidth Delay product

are independent of each other. As a result the flows in the internet core have varying RTTs. The senders therefore infer congestion at different points of time, leading to desynchronization among flows. The buffer requirement of the individual flows are now out-of-sync. This has been reported in previous studies like [9, 37]. When the flows are desynchronized, the difference between the peak and trough of the sum of the congestion window sizes reduces. The aggregate congestion window at time t is the sum of the congestion window sizes of the individual flows (W_i). It has been shown using central limit theorem, that the aggregate congestion window size has a Gaussian distribution around the mean congestion window size [9]. The queue size, which is the buffering required, at time t is given by

$$Q(t) = \sum_{i=1}^{n} W_i(t) - 2\sum_{i=1}^{n} T_{pi}.C - \epsilon \tag{4.5}$$

Here, ϵ is the number of packets dropped and is negligible when TCP is functioning correctly. The buffer size in the router is the difference between the sum of the congestion windows of the flows and the total number of packets transmitted on the link.

$$Q = W - 2.T_p.C \tag{4.6}$$

Since W is a Gaussian random variable, the number of packets in the router queue also has a Gaussian distribution. Assuming that individual flows vary uniformly between $[\frac{2}{3}W_i, \frac{4}{3}W_i]$, it has been shown in [9] that a buffer of size of B_{core} that can support a link utilization of 98.99% is given by

$$B_{core} = \frac{bandwidth\ delay\ product}{\sqrt{number\ of\ flows}} \tag{4.7}$$

4.3 Dynamic Packet Buffering

We observe that in the core of the internet, link utilization is kept around 50%. As a result, the network processor can forward packets as soon as the processing is completed and packet buffers could underflow. We use the fact that buffering is required only when there is a backlog of packets to be forwarded and when the output link is fully utilized. Such a scenario occurs when a burst of packets is received and is infrequent in the core of the internet due to the presence of a large number of flows and overprovisioning of the links.

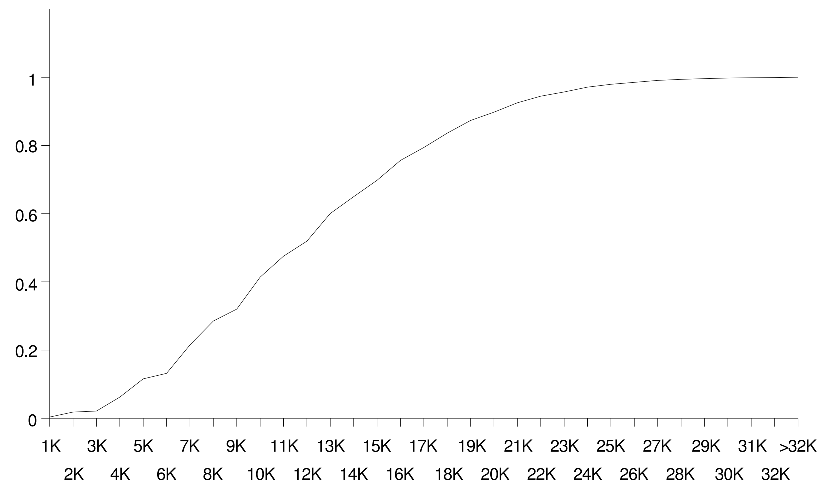


Figure 4.3: DRAM buffer utilization in the core internet

Figure 4.3 shows the DRAM buffer utilization for IPv4 forwarding application. This shows us the actual amount of buffer space required under uncongested output link conditions. In this experiment, we used a network core like traffic with a mean input rate of 4765Mbps for an interval of 200ms. A total of 24 processing threads running on 3 MEs were used. The aggregate outlink bandwidth is assumed to be 8100Mbps. This is the typical usage scenario in core routers where the network traffic is about half the output link capacity. The Petri net model used here is similar to the one described in Section 3.3. We used the First Cell Buffering (refer Section 3.5) scheme to buffer the packets in DRAM. The method used to generate the traffic is described in Section 4.4.3.

We studied the buffer requirement to store packets in the network processor. The x-axis shows the buffer size and the y-axis shows the fraction of time for which the buffer utilization is less than the given value. We see that buffer utilized does not exceed

36.5KB and is less than 16KB for 77% of the time duration.

This experimental result along with the argument that the buffer requirement is low in the core routers, which was explained in Section 4.2, motivates us to explore small buffer sizes that can be used under most traffic regimes. Buffering all packets in the DRAM may not be necessary, if the lookup can be performed when the packet is still stored in the on-chip memory. This network processor has 8KB of on chip memory called Receive buffer (RBUF) to store packets before they are buffered in the DRAM. Also, there is 8KB of Transmit buffer (TBUF) which is used to store packets before they are transferred to the egress ports. Based on the insight that buffer sizes in core routers can be much smaller than the bandwidth delay product, we propose a dynamic buffering strategy wherein packets are stored in the DRAM only when a backlog of packets is developed in the RBUF. When the queue length of the output ports is not very high, the packets could be moved directly from the RBUF to the TBUF. DRAM buffering is necessary only when a burst of packets is received.

A packet is buffered in the DRAM when the length of the queue corresponding to its output port increases beyond a threshold, which we call the *high watermark*. Packet build up at the output port implies the need for increased buffering and this larger buffer space is provided by storing the packets in DRAM. In IXP 2400, the data bus that connects the microengines to the DRAM buffer is an impediment in achieving higher packet forwarding rates (refer to Section 3.7). We do not store all packets in the DRAM, but only packets destined to the congested output ports, due to which the traffic to the DRAM reduces substantially. This effectively mitigates the DRAM bottleneck that we previously observed. When the backlog at the output port reduces, we revert to normal processing for that port, wherein packets are directly moved from the RBUF to the TBUF. We change to on-chip packet buffering when the queue length for that output port goes below a threshold called the *low watermark*. We refer to the above scheme as dynamic buffering scheme, whereas the original approach where all packets are buffered in the DRAM is referred to as DRAM buffering scheme.

Moving the mpackets directly from the RBUF to the TBUF in the dynamic buffering scheme requires a datapath between the two buffers. This path does not exist in the IXP 2400 architecture, described in Section 2.1. All the mpackets have to be moved explicitly through the micro engines. We propose the addition of this data path as it could lead to substantial improvements in the throughput of applications and frees up the threads for processing. The addition of this path does not incur significant cost as both the buffers are on chip. In the remaining part of the study, we assume the existence of such a databus with a bandwidth of 3.2GB/sec.

4.4 Petri Net Model

4.4.1 Petri Net Model for Dynamic Buffering

In order to evaluate the dynamic packet buffering scheme, we simulate the processing for a representative header processing application such as IPv4 packet forwarding on the IXP 2400 network processor. When a packet is received at an ingress port, it is moved to the RBUF and a thread from the free pool is assigned for processing it. The thread transfers the first cell into the scratchpad memory and reads the destination address into the micro engine registers. IP lookup is performed using the hash based scheme described in [39]. A hash of the destination address is used to index into the SRAM in order to find the output interface. The modified header is written back into the scratchpad memory after recomputing the checksum. After the output interface is known, the queue length of the output port is updated. In case the queue length for that port is above the high watermark, the packet is buffered in the DRAM by moving mpackets to the DRAM. Otherwise, the packet is moved from the RBUF to TBUF. The control words for the packet are suitably updated. The queue length corresponding to each port is adjusted whenever a packet enters and leaves the network processor. All packets belonging to a flow are forwarded to the same output port.

The simulation setup is depicted in Figure 4.4. The IP forwarding application with dynamic buffering is modeled using a detailed Petri net model similar to the one described in Section 3.3. Traffic from the flow based model (described in Section 4.4.3) is given as

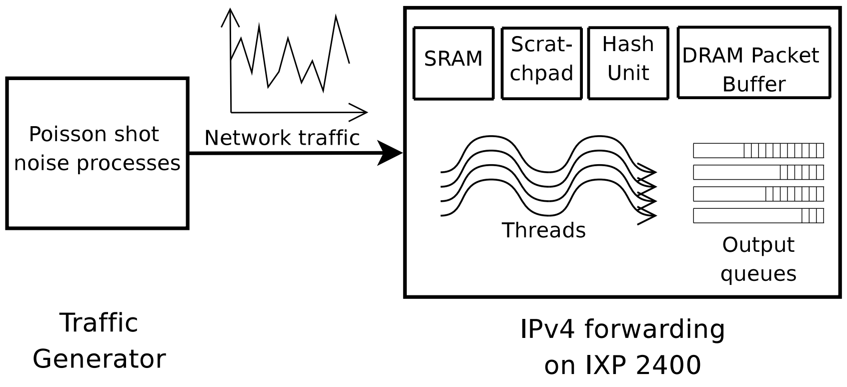


Figure 4.4: Simulation setup

input to the IPv4 forwarding application model.

The SRAM and hash unit that are used for IP lookup are accurately modeled. One queue is maintained for each output port. In case the length of the queue for a given port is greater than the high watermark, the corresponding packets are buffered in the DRAM. The DRAM Petri net model described in Section 3.2 is used for modeling the DRAM.

In the dynamic buffering scheme, RBUF storage is used when threads are processing the packet, i.e., when the thread is determining the output port of the packet. When the processing threads are unable to keep up with the packet arrival rate, the RBUF occupancy increases. A packet is dropped when there is not enough space in the RBUF to store it.

4.4.2 Comments

The First Cell Buffering (FCB) packet processing scheme described in Section 3.5 and the current processing scheme have some differences which are as follows. In FCB scheme, IP lookup is performed after the packet is buffered in the scratchpad memory (for the first cell) and the off-chip DRAM (for the remaining part of the packet). In the dynamic buffering strategy proposed, the first cell is stored in the scratchpad memory when a thread is assigned to its processing. However, the thread performs the IP lookup before buffering the remaining part of the packet. This is necessary because the buffering decision for the packet is taken based on the output queue length. This is known only after the IP lookup is performed.

The packet descriptor associated with each packet carries the location in memory where the packet is stored. When the packet has to be transmitted, this information can be used to find if it is buffered in the on-chip memory or in the DRAM. Packets that are stored in the DRAM have to be moved from the packet buffer to the TBUF for transmission on the output port.

4.4.3 Flow Based Traffic Generation Module

In real traces, various characteristics of network traffic such as the packet arrival rate, the packet sizes, the number of packets in a flow and the number of active flows vary with time. This leads to changes in the length of queues corresponding to different output ports. In order to accurately model the variation in characteristics of the network traffic, we use the flow based model proposed in [12]. This flow generation methodology described in [12] is realized using a colored Petri net. The tokens generated contain information about the packet length and forwarding port. This is given as input to the IPv4 forwarding application simulation module.

Core Network Traffic Generation Model

Barakat et al. [12] describe a flow based model for uncongested links in the internet that can capture the dynamics of the traffic at short timescales. The model uses the notion of flows to capture the characteristics of data streams such as changes in their rates, duration and their arrival rates. The flows have a Poisson arrival rate and the total traffic at any instant of time is the superposition of the active flows. The flow rates are independent of each other. Further, they show that in real traces, the flow sizes and duration are independent of the size and duration of other flows. To keep the analysis simple, they assume each flow is as a rectangular Poisson shot-noise process. The *shot* refers to the rate function of the flow. The rectangular shot-noise implies that the rate of each flow is constant. This model has been shown to be efficient is generating network traffic through simulation and can be used as a tool to determine the link capacity in the internet.

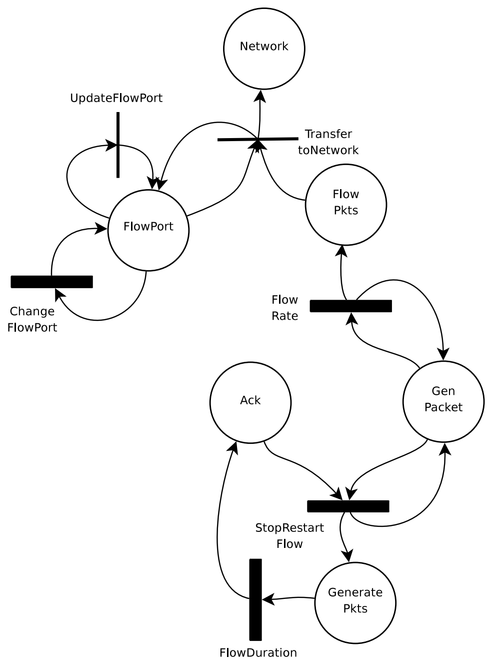


Figure 4.5: Petri net model for flow generation

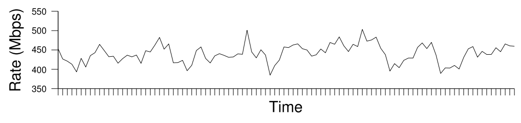
The Petri net model used to generate traffic using the above Poisson shot noise process model, is shown in Figure 4.5. The rate of packet generation for each flow is determined by the color of the token in *GenPacket* place. The traffic rates of these flows are independent of each other. Based on the color of the token in *GenPacket* place the mean firing time of *FlowRate* transition varies. This corresponds to the generation of a rectangular Poisson shot noise process. The *FlowRate* transition places a token signifying the length of the packet in the *FlowPkts* place. The packets generated by a flow are associated with a single port based on the color of the token in the *FlowPort* place. The transition *TransfertoNetwork* places the tokens at the output place *Network*.

Due to the arrival of new flows and completion of active flows, the number of active flows at a given instant of time varies. The *StopResetFlow* and *FlowDuration* transitions are responsible for stopping active flows and starting new flows. When a halt token is present in the *Ack* place, the *StopResetFlow* transition removes the packet generating token from *GenPacket* place, thus stopping the generation of packets for that flow. After the completion of the transition, the token is put back in the *GenPacket* place. The flow is again stopped when the halt token is moved from *GeneratePkts* place to *Ack* place by the *FlowDuration* transition. The transition time of *FlowDuration* corresponds to the interval for which the flow is active.

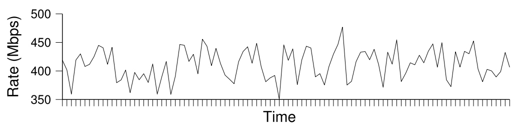
The packets are generated with the length distribution observed in the internet core trace, Abilene trace I2C-1091276356-2. The rate of the flows was chosen between 1Mbps and 1Kbps. A flow duration can be between 1ms to the entire simulation time, in case of long lived flows. These values are similar to the characteristics observed in [47].

Validation of the Traffic Generation Module

Two important characteristics of network traffic are its mean rate and the amount of variation in traffic arrival rate [12]. These properties of network traffic determine the utilization of various resources in a router. The traffic workload therefore plays a major part in this evaluation. We used the characteristics observed in the real network trace from the Abilene internet backbone (I2C-1091276356-2) to validate our traffic generation module. The traffic rates were measured in intervals of 200ms. The maximum and the minimum rates observed in 200ms windows were 503Mbps and 272Mbps. A maximum of 14.66% variation in the traffic was observed for a 10 second interval while the minimum variation in traffic rate was 3.05%. We use these observations to validate our traffic generation module.



(a) Real traffic



(b) Simulated Traffic

Figure 4.6: Traces from real traffic and simulation

We validate the model by checking that traffic generated has similar variation as the one observed in real network traffic in the core of the internet. In order to measure variation in traffic, we adopted the method in [12]. The network throughput is measured over 200ms windows, which is the typical round trip time. A plot of varying traffic rates observed at 200ms intervals in a real internet core router and the traffic generated by our model are shown in Figure 4.6. The real trace has a mean rate of 440 Mbps and a variation of 5.40%. The traffic generated by our model has a mean of 411 Mbps and a variation of 6.45%. For our experiments, since we need to simulate input traffic at higher rates, we increase the number of flows. As a result, the mean rate of the traffic generated increases.

4.5 Performance Evaluation

In order to evaluate the performance of the dynamic packet allocation scheme, we study the transmit rates and the number of dropped packets in the router. During different runs of the simulation with DRAM and dynamic packet buffering schemes, we provide exactly the same traffic input by using the same seed to the random number generator of the traffic module. This enables us to compare the two schemes under the same conditions of variation in network traffic characteristics. We assume that the aggregate bandwidth of the output links is 8100 Mbps. The mean rate of the input traffic is varied with rate ranging from 5000Mbps to 7200Mbps. The overprovisioning in our experiments is much lower than that in an actual setting in internet core routers where in the link utilization is usually 50% [9, 12]. We note that having lower provisioning for the output link would require more DRAM buffering and could potentially lead to more packet drops.

We report the transmit rate and packet drop rates over 200ms windows. This captures the behaviour of the schemes during varying regimes of rates. Simulations with input traffic rate of 5417Mbps and variation of 5.52%; and mean of 6193Mbps with 13.48% variation were carried out.

Input Traffic rate (Mbps)	DRAM buffering		Dynamic buffering		
	Transmit rate	Packets dropped	Transmit rate	Packets dropped	Buffered in DRAM
4906	4741	1.92%	4905	0.01%	0.60%
5571	5104	4.87%	5569	0.02%	1.14%
5422	5078	3.66%	5421	0.02%	1.28%
5519	5085	4.56%	5516	0.03%	2.41%
5667	5167	5.08%	5663	0.04%	1.35%

Table 4.1: Throughput and packet drop rates with a mean of 5417Mbps and 5.52% variation

4.5.1 Impact of Dynamic Buffering

Table 4.1 shows the performance of the two schemes over five 200ms intervals. We observe that when the network rate is high, the DRAM packet buffering scheme drops more packets. It is not able to provide the higher throughput rates. This is because of the DRAM bottleneck discussed in Section 3.7. As all packets are transferred over the bottleneck data bus, the DRAM does not have extra bandwidth to absorb bursts of traffic and the network processor experiences packet drops. This is seen from the increasing packet drop rates with higher traffic rates. On the other hand, with dynamic buffering scheme the packet drops are negligible and the scheme is able to support higher throughput. Also, the percentage of packet data that is buffered in the DRAM due to bursts in flows is shown. This percentage is very small. This is because the threads are able to handle most of the traffic at the given input rate. With higher rates, this percentage increases slightly.

Table 4.2 shows the performance of the DRAM buffering and dynamic buffering schemes under a mean traffic of about 6193Mbps and high variation of 13.48%. We see that the dynamic buffering scheme provides higher throughput and lower packet drop rates even with high variation in traffic.

In order to find the buffer utilized during processing, we measured the buffer requirement when a packet arrives at and departs from the NP. A packet arrival at the NP or departure from the NP is referred to as an event. Figure 4.7 shows the percentage

Input Traffic rate (Mbps)	DRAM buffering		Dynamic buffering		
	Transmit rate	Packets dropped	Transmit rate	Packets dropped	Buffered in DRAM
5802	5179	6.23%	5775	0.25%	1.43%
5633	5096	5.53%	5611	0.22%	2.88%
5496	5074	4.43%	5473	0.24%	3.37%
7505	5860	9.06%	7276	1.75%	8.36%
6531	5787	8.72%	6441	0.80%	7.05%

Table 4.2: Throughput and packet drop rates with a mean of 6193Mbps and 13.48% variation

of events for which the buffer requirement is as shown on the x-axis. We see that the probability of buffer requirement being greater than 10KB is very low.

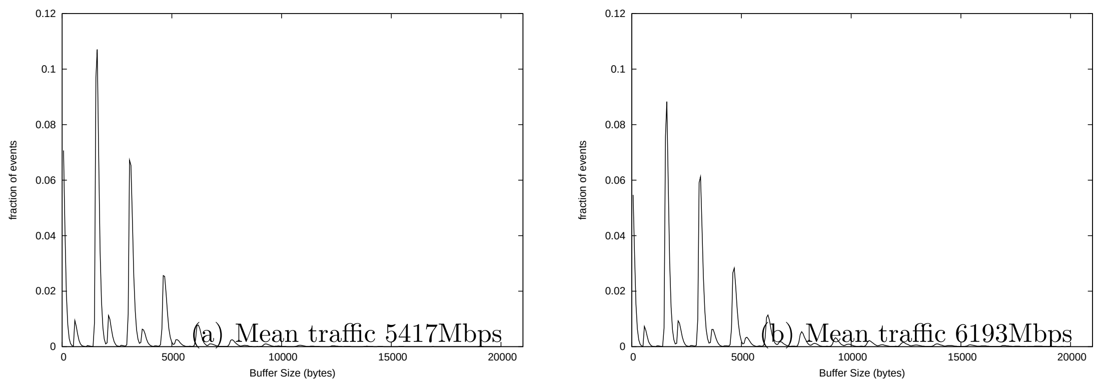
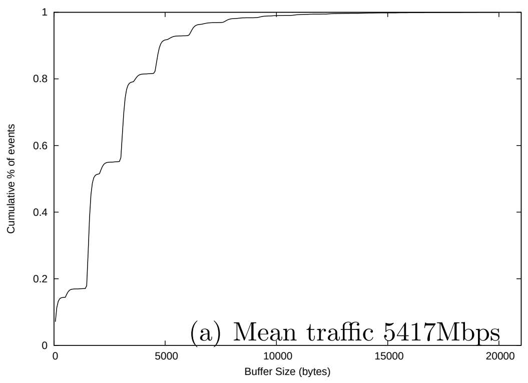


Figure 4.7: Buffer requirement with dynamic buffering

Figure 4.8 shows the cumulative percentage of events at which the buffer requirement was less than the value on the x-axis. With a mean traffic of 6193Mbps, the buffer requirement is less than 10KB for 94% of the time and with a mean traffic rate of 5417Mbps, the buffer requirement is less than 10KB for 99% of the time. The maximum buffer requirement is 36KB and 24KB respectively. We observe that more buffer space than the RBUF and TBUF memory is necessary to store packet during bursts in traffic. Due to dynamic buffering, this additional memory is provided in the DRAM.



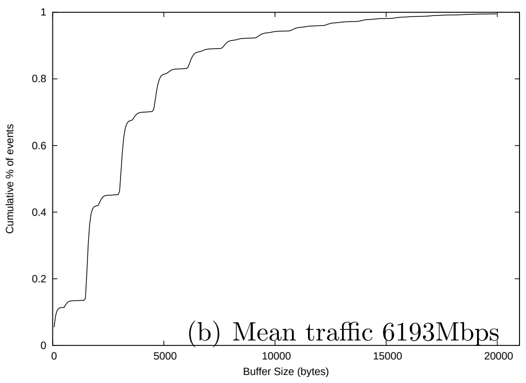


Figure 4.8: Buffer occupancy with dynamic buffering

Input rate (Mbps)	Threads	Transmit rate	Buffered in DRAM	Packets dropped
6215	8	5886	2.88%	2.94%
6215	16	6137	10.69%	0.72%
6215	24	6178	2.79%	0.34%
6215	32	6178	4.02%	0.34%
6215	40	6179	3.18%	0.33%

Table 4.3: Transmit rates with different number of threads

4.5.2 Effect of System Parameters

The dynamic buffering scheme provides a number of system parameters that can be tuned in order to improve the performance. We study the impact of two parameters namely, the number of threads dedicated to processing and the value of the high water mark that is used to determine the need for DRAM buffering. These parameters affect the packet drop rate and the amount of traffic that is buffered in the DRAM.

In order to choose the optimum number of threads, we ran simulations with different number of threads but with the same input traffic. The simulations were run for 400ms with a mean traffic of 6215Mbps. We studied the performance of the dynamic buffering strategy with varying number of processing threads. The throughput and packet drop rates achieved with different number of processing threads are shown in Table 4.3.

This provides insight into the number of threads that must be provisioned for network

Input rate (Mbps)	Threads	Transmit rate	Buffered in DRAM	Packets dropped
7390	8	6605	5.95%	5.66%
7390	16	7092	11.75%	2.25%
7390	24	7103	13.10%	2.17%
7390	32	7090	16.04%	2.35%
7390	40	7060	14.10%	2.58%

Table 4.4: Transmit rates with different number of threads at higher traffic rate

processing under bursts of traffic. As the number of threads are increased, we see a reduction in the packet drop rate due to a decrease in the waiting time. With only 8 threads, the packet drop rate is about 3%. This is because the small number of threads are unable to keep up with the rate of packet arrival. Due to overflow in the RBUF, packets are dropped even before they can be buffered or forwarded. Consequently, the throughput achieved drops. However, when the number of processing threads is increased to 16 or more, the packets are forwarded at the input rate, as a result of which packet drops reduce. A higher number of processing threads implies that each of the packets experiences lower waiting delay in the RBUF. This could reduce the number of packets dropped due to limited size of the RBUF. Packet bursts lead to buffering in the DRAM and there is a slight increase in the DRAM traffic with 24 and 32 threads. However, when there are 40 threads, IPv4 processing is more resilient to packet bursts. The lookup and forwarding can be serviced even at higher arrival rate, as a result of which we see a reduction in the DRAM buffering.

Table 4.4 shows the performance of dynamic buffering with a higher input traffic rate of 7390Mbps. We observe that packet drop rates are greater than 2%. This is due to larger packet build up in the RBUF. The packet drop rate with dynamic buffering is still less than the drop rate observed with DRAM buffering which dropped more than 5% of the packets with a traffic rate of 6193Mbps (refer Table 4.2). Interestingly, the packet drop rate does not decrease even with an increase in the number of threads because the high utilization of the RBUF and TBUF. Due to high utilization, packets cannot

Input rate (Mbps)	High Watermark(bytes)	Transmit rate	Buffered in DRAM	Packets dropped
6330	1600	6176	27.96%	1.39%
6330	2240	6222	16.93%	0.97%
6330	2880	6234	12.94%	0.85%
6330	3520	6241	8.73%	0.79%
6330	4160	6249	4.33%	0.73%

Table 4.5: Transmit rates with different threshold for high watermark

be moved from the RBUF to TBUF when there is a large burst of traffic, leading to increased packet drop rate.

In order to assess the effect of the high watermark on the amount of traffic buffered, we simulated the performance of the network processor with different values for this threshold. Due to burstiness in the network traffic, the output queue length for one of the ports crosses the threshold, leading to buffering of corresponding flows in the DRAM. A low threshold could excessively buffer all packets in the DRAM due to which the DRAM bottleneck previously observed would reappear.

Table 4.5 shows the throughput and packet drops with various values of the high watermark. Input traffic with a mean of 6330Mbps and 32 threads (4MEs with 8 thread enabled) was used. The high watermark in column 2 is the length of the queue associated with each port, beyond which all packets belonging to this queue are buffered in the DRAM. As the value of the high watermark is increased, the amount of traffic that is buffered in the DRAM decreases. When the high watermark, is set to 1600 bytes, about 28% of the traffic is buffered in the DRAM, requiring the RBUF to DRAM and DRAM to TBUF transfer of packets, reducing the bandwidth necessary to absorb traffic bursts. This, in turn causes more packets (1.39%) to be dropped. It should be noted that with the original DRAM buffering packet drop rates are even higher (more than 4%). However, when the value of the high watermark is increased, the amount of traffic through the DRAM decreases. In this case the available DRAM bandwidth is able to absorb the packet bursts, leading to a decrease in the packet drop rate. The packet drop

Resource	Thread Configuration				
Utilization	1 x 8	2 x 8	3 x 8	4 x 8	5 x 8
Hash Unit Utilization	14.2%	15.0%	15.0%	15.0%	15.0%
Thread Utilization	27.9%	14.5%	9.1%	6.8%	5.1%
ME Utilization	54.3%	28.6%	19.1%	14.3%	11.4%
Data Bus Utilization	3.2%	6.0%	4.8%	5.5%	3.40%
Average RBUF Queue len.(bytes)	1444	1356	1293	1284	1249
Average TBUF Queue len.(bytes)	1432	2250	2310	2327	2368
Packet Drop Rate	2.94%	0.72%	0.34%	0.34%	0.33%

Table 4.6: Utilization of resources in the NP with mean input traffic of 6215Mbps

rate and transmit rate do not change substantially when the value of the high watermark is above 2880 bytes. This is because, when the high watermark is over 2880 bytes, only a small fraction of the traffic is buffered in the DRAM; increasing the value of the high watermark does not improve the packet drop rate.

4.5.3 Resource Utilization with Dynamic Buffering

The Petri net model allows us to analyze the utilization of resources in order to understand the hurdles in achieving higher throughput rates. Table 4.6 shows the resource usage with a mean input traffic of 6215Mbps and different configurations of MEs and threads. The marking mxn corresponds to m MEs with n threads per ME. We observe that the hash unit utilization remains constant around 15%. The utilization of threads decreases, as their total number increases. This is expected as there are more threads available for processing. Interestingly, with a 1x8 configuration (one ME running eight threads), the ME utilization is highest at 54%, as only one thread can be running within a ME at a given time. Such a high utilization of a single ME prevents the NP from assigning more processing threads when there are packet bursts. It explains the high packet drop rate of nearly 3% that was observed in this configuration. The data bus bottleneck that previously manifested when all packets where buffered in the DRAM, has been effectively removed. Data bus utilization is below 6% in all the configurations. The RBUF and TBUF occupancy has a bearing on the ability of the NP to absorb sudden

Resource	Thread Configuration				
Utilization	1 x 8	2 x 8	3 x 8	4 x 8	5 x 8
Hash Unit Utilization	15.90%	17.30%	17.40%	17.50%	17.50%
Thread Utilization	36.70%	23.20%	16.70%	14.50%	11.70%
ME Utilization	61.20%	33.00%	22.20%	16.70%	13.20%
Data Bus Utilization	7.40%	14.60%	16.20%	19.80%	17.40%
Average RBUF Queue len.(bytes)	1790	1996	2026	2140	2154
Average TBUF Queue len.(bytes)	2096	3824	3883	3815	3806
Packet Drop Rate	5.66%	2.25%	2.17%	2.35%	2.58%

Table 4.7: Utilization of resources in the NP with mean input traffic of 7390Mbps

increase in traffic. A low utilization of these resources enables the NP to store packets when there is a traffic burst.

Table 4.7 shows the resource utilization with a mean traffic of 7390Mbps. As expected, the resource utilization is higher than in the previous case. An interesting point to note is the increased utilization of the RBUF with higher input traffic rate. This reduced the space in RBUF to accommodate packet bursts, leading to increased packet drop rates.

This leads us to conclude that the size of the RBUF and the number of threads provisioned for processing affect the packet drop rate. The role of the RBUF is to provide storage to packets when there is a burst of packets. The number of threads available for processing determines the duration for which the packet is stored in the RBUF. Fewer processing threads implies that the RBUF could overflow leading to packet drops.

4.5.4 Remarks

The proposed scheme has been evaluated for header processing applications. In packet processing applications, the payload of the packet needs to be brought into the MEs and cannot be directly transferred from the RBUF to the TBUF. Previous studies [21] have shown that the utilization of resources for these applications is different from header processing applications. Therefore, a more detailed evaluation is necessary to extend the proposed scheme to packet processing applications.

4.6 Related Work

Dukkipati, et al. [17] propound the principle of using the common case behavior of network applications to lead the design of routers and network protocols. They provide examples from [9] which shows that router buffers can be reduced by two orders of magnitude with respect to the bandwidth delay product rule. This reduces router queuing delay and simplifies the design of routers. They also draw attention to the fact that packet classification rules exhibit a Zipf like distribution in matching the packets, i.e. a few rules match most of the packets. This can be used to design faster classification tables through caching. However, a packet that suffers a miss in this cache, could possibly take more time to resolve the matching rule.

A number of papers have recently studied the size of packet buffers for internet links. The results provide greater insight into this interesting problem. Dhamdhere, et al. [16] provide a generalized analytical model to determine the size of packet buffers in routers, under various constraints on link utilization, total packet loss rate and queuing delay. They show that when the output link is congested, the buffer size should be equal to the bandwidth-delay product. However when the output links are not congested, as in the case of internet routers, the buffer requirement is given by the Stanford model [9]. These models assume that the output link has to be fully utilized. We observe that in core routers, since the output links are overprovisioned due to which link utilization is never full. In such a scenario, the buffer requirement can be reduced to less than 1% of the bandwidth-delay product. Taking a cue from this work, we propose a dynamic strategy where packet buffering in DRAM is provided only when there is a burst of packets in the traffic or there is congestion in the output link. This reduces the buffer requirement in the common case for core routers. Such a buffering scheme also results in significant improvement in packet throughput as DRAM buffering, which is an impediment to achieving higher throughput is eliminated in most cases.

Shorten et al.[37] discuss changing the back-off factor in the additive increase multiplicative decrease (AIMD) congestion control algorithm of TCP in order to reduce the size of buffering required. Using a large back-off factor increases the responsiveness of the TCP flows to congestion events, but decreases utilization of the output link. They aim to strike a balance between flow responsiveness and link utilization by varying the back-off factor between $[0.5, 0.8]$. The authors show that with such an implementation, TCP protocol can adjust to small packet buffer sizes. While this scheme requires changes in TCP implementation, our scheme does not envisage any change in the sender's TCP implementation.

4.7 Conclusions

Statistical multiplexing of flows and the low link utilization in core internet routers allowed us to reduce the amount of slow DRAM buffering. We showed that different on-chip storage locations present in IXP 2400 can be used to buffer packets during most regimes of the input traffic. We used a Petri net model to generate traffic with the same characteristics as real traces. The dynamic buffering strategy successfully addressed the issue of DRAM buffering being a bottleneck in network processing applications. We evaluated the dynamic buffering strategy in detail. The utilization of the resources under different traffic rates and variation conditions were measured. With dynamic buffering the utilization of the DRAM data bus reduced significantly. We showed that a substantial amount of the traffic moved from the RBUF to the TBUF, removing the bottleneck due to limited bandwidth of the DRAM. The achieved network throughput was large and the packet drop rate was low. Having more threads for processing increased the ability of the processor to handle traffic bursts. The choice of the high water mark determined the amount of traffic that is buffered in the DRAM.

Chapter 5

Improving Performance of Result Caches in Network Processors

5.1 Introduction

In this chapter we study the performance of result caches for packet classification application and propose a new cache replacement algorithm to improve its performance. A number of algorithmic techniques have been proposed which address the problem of packet classification [40, 10, 23]. However these methods need large tables and require multiple accesses to main memory. Other methods proposed for speeding up this processing involve caching the data structures used [32, 11] or caching the results of the lookup [31, 13]. Cache based methods exploit temporal locality observed in internet packets. Consequently, the efficiency of these schemes is dependent on the access characteristics observed in real traces from the internet. Also, network processors have small amount of local memory in each ME that can be used as caches for network processing applications.

In the internet, a large number of flows have a single packet, whereas a small number of flows have multiple packets and contribute to the majority of the traffic. From a caching perspective, flows with a single packet do not exhibit any (temporal) locality and hence it is not beneficial to be stored in the result cache. Also these single packet flows may evict information belonging to flows with multiple packets. We propose to exploit this characteristic of internet flows to improve the performance of result cache by using a new cache replacement policy called *Saturating Priority.* Also we analyze the misses and show that majority of them are cold or capacity misses. This stresses the fact that algorithmic improvements are necessary in order to effectively address the issue of faster packet classification.

The rest of the chapter is organized as follows. In the next section, we present the necessary background for digest caches. In Section 5.3, we present the characteristics of internet traffic in terms of flow size distribution and motivate our Saturating Priority cache replacement algorithm. Section 5.4 deals with the initial performance results of the Saturating Priority algorithm. We present the related research proposals in this area in Section 5.5. Finally we conclude in Section 5.6.

5.2 Using Digest Caches to Aid Network Processing

In the remaining part of the study, we consider a packet classification application [23] in which a tuple consisting of the source IP address and port, destination IP address and port and the protocol field are used to identify packets belonging to a particular network flow. This 5 tuple is called the flow identifier and is used to map the packets of a flow to a particular traffic QoS class. This methodology may also be used in packet forwarding application where the port on which the packet has to be forwarded is stored along with the flow identifier. In network address translation (NAT) application, the digest cache can be used to lookup the address translation information.

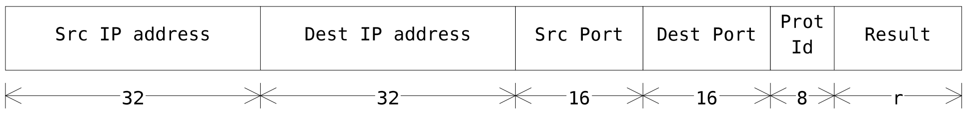


Figure 5.1: Result cache entry for packet classification

In case of packet classification for IPv4, result caching involves storing the 104 bit 5-tuple along with the QoS or forwarding information of size r bits as shown in Figure 5.1. The memory size required to realize such a cache with sufficiently high hit rate could be large due to the large size of the entries. A recent proposal has been to use a smaller digest of the 5-tuple instead of the actual values in the fields [13]. The authors show that the misclassification rate due to distinct tuples mapping to the same digest, is negligible. For a c bit digest used in d way associative cache, the probability p of collision is

$$p = \frac{d}{2^c} \tag{5.1}$$

For a 32-bit digest, the error probability is one in a billion. This is lower than the errors in TCP checksum which could fail once in 32000 [13]. Further, the authors suggest that network programs are inherently designed to tolerate such errors in network processing and recover from them.

5.2.1 Operation of a Digest Cache

We now describe the operation of the digest cache which was presented by Chang et al. [13]. In case of IPv4, a digest cache works by using a hashing algorithm on the 104-bit 5 tuple to generate a 32-bit hash. The least significant s bits are used to index into a set associative cache of 2^s sets. The remaining $(32 - s)$ bits of the hash are used for tag check, after which a cache hit or miss is known. In case of a miss, the processing proceeds by using a full classification algorithm. The digest that missed in the cache replaces another entry in the set using a cache eviction policy which chooses the victim. Steps involved in accessing a digest cache are shown in Figure 5.2.

Each entry in the digest cache consists of a tag of $(32 - s)$ bits and r bits of classification information. Thus the total size of a digest cache with 2^s sets and k blocks per set (associativity) is

$$Digest\ Cache\ Size = \frac{((32 - s + r) * 2^s * k)}{8}\ bytes \tag{5.2}$$

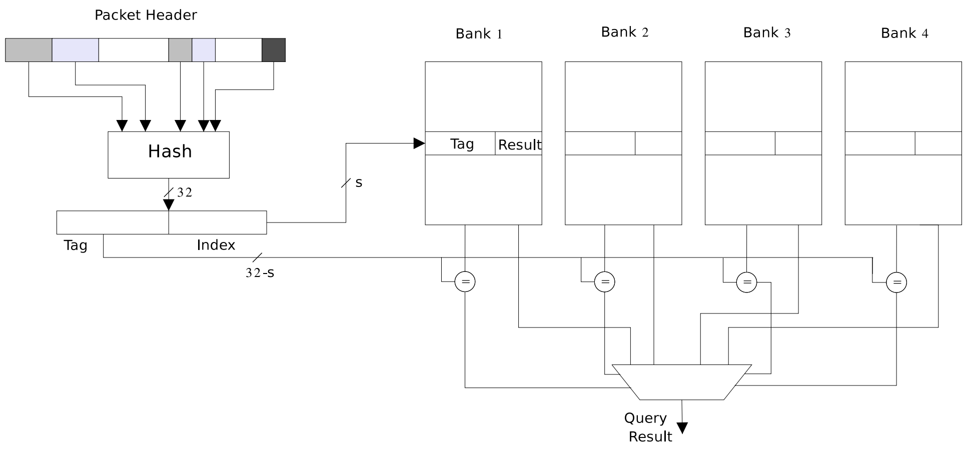


Figure 5.2: Accessing an entry in a digest cache

Entries	Digest Cache Size	
	4-way assoc.	8-way assoc.
512	2112 bytes	2176 bytes
1024	4096 bytes	4224 bytes
2048	7936 bytes	8192 bytes

Table 5.1: Cache sizes for different number of entries and associativities

The size of a cache with different number of entries and associativities are shown in Table 5.1. Here, we assumed that 1 byte IP lookup or classification information is stored along with the digest. This is sufficient for various diff serve classes for which classification has to be performed.

The digest cache acts as a filter, servicing the frequently accessed digests. Only those packets whose digests miss, go through the slower packet classification algorithm. A higher hit rate in the digest cache is critical for higher classification rates as it decreases the number of packets going through the slower packet classification step.

Trace	Label	Type	Files
Abilene Indianapolis router	Abilene	Core	I2K-1091235140-1.erf.gz to I2K-1091253948-2.erf.gz
National Center for Atmospheric Research	NCAR	Edge	20031229-223500.gz to 20031229-221000.gz
Front Range Gigapop	FRG	Edge	FRG-1133754905-1.tsh.gz to FRG-1133818534-1.tsh.gz
Pittsburgh Supercomputing Center	PSC	Edge	PSC-1145580225-1.tsh.gz to PSC-1145742328-1.tsh.gz
San Diego Supercomputer Center to Abilene connection	SDA	Edge	SDA-1141865872.erf.gz to SDA-1142027975.erf.gz

Table 5.2: Traces from the internet used in this study

5.3 Improving the Performance of Digest Caches

Digest cache is a *traffic aware* mechanism for network processing. The number of flows in the internet, their sizes and rates influences the performance on the digest cache. In order to understand the workload of the digest caches, we study the above aspects from real traces.

5.3.1 Flow Size Distribution in the Internet

We use the following definition of an active flow in our study. A flow is said to be active when it sends the first packet and expires when it does not send a packet for 1 second since the last packet. A similar definition is used on [47][1]. The traces that we use in our study and their properties are shown in Table 5.2.

For the three traces, Abilene, NCAR and SDA, Figure 5.3 shows the cumulative percentage of flows having packets less than the value on the x-axis. The graph also shows the cumulative percentage of the packets that are transferred by these flows. These traces have 3.1 million, 15.4 million and 4.4 million flows respectively. In NCAR trace, more than 95% of the flows have only a single packet. Abilene and SDA traces

[1]Authors in [47] use a time out of 60 seconds. Even with a time out of 1 second, there are about 6000 concurrent flows in all our traces. This is more than the number of entries in the digest cache.

have 58% single packet flows. Single packet flows, which are a significant part of the total number of flows, contribute less 6% of the packets transferred in Abilene and SDA traces. In NCAR trace, single packet flows transfer 25% of the total packets. Less than 0.2% of the flows have more than 1000 packets, but they transfer 52% of the packets on an average.

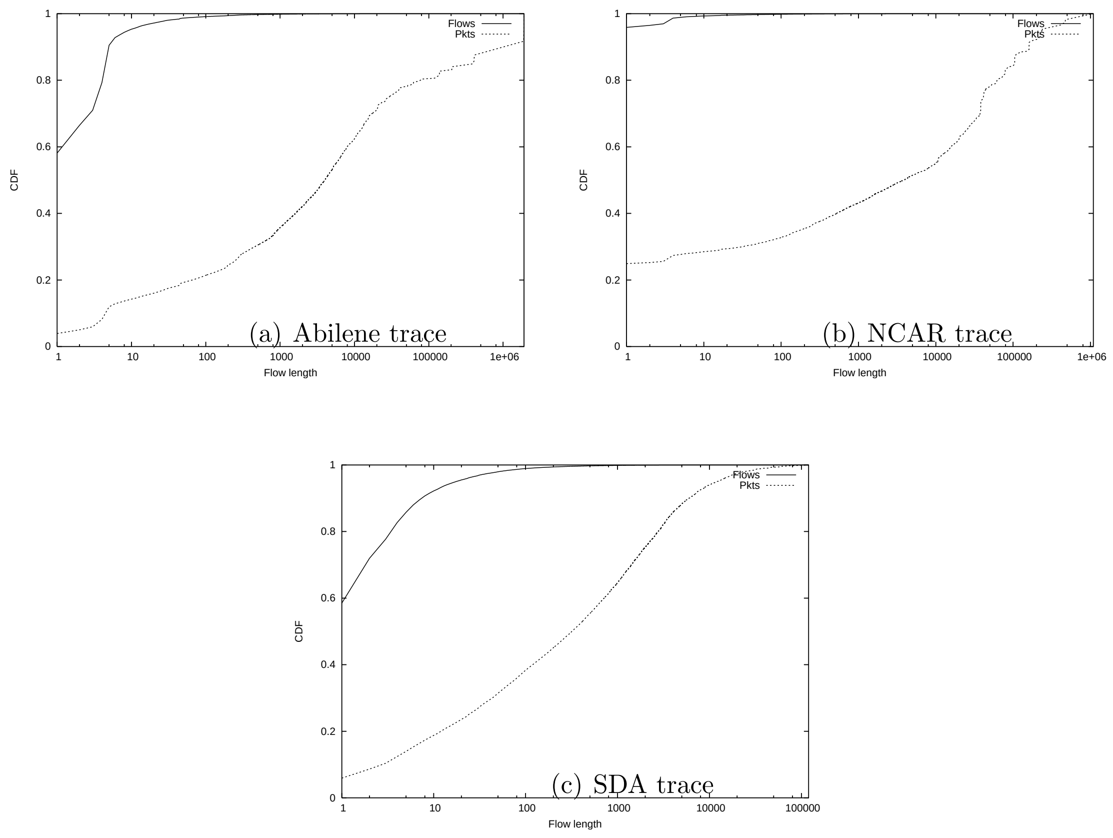


Figure 5.3: Disparity in the flow sizes and packets transferred

Substantial number of flows in the internet have only a single packet. These flows get entries into the digest cache, but they are never accessed again. Presence of a large number of such flows has a detrimental effect on the performance of the digest cache as they tend to evict the digests of few flows that contain a large number of packets.

5.3.2 Replacement Algorithm for Digest Caches

The above observations provide insight into the functioning of the digest cache. The cache has to service a large number of flows, but only a small fraction of the flows have multiple packets that are accesses again. From Figure 5.3, we infer that the cache replacement policy of a digest cache must strive to retain the digests belonging to large flows within the cache while preventing the single packet flows from occupying entries in the cache.

Given the large number of single packet flows in the internet and the small cache size, an LRU managed digest cache, may suffer from degraded performance due to cache pollution. Previous studies using digest caches used LRU cache replacement strategy as it performed better than LFU and probabilistic insertion policies [31, 13]. LRU cache management gives preference to the most recently accessed entry in the set and puts it in the most-recently-used (MRU) location. In case of network processing applications, most of the recently accessed digests will never be accessed again as they belong to single packet flows. But they stay in the cache until they are slowly pushed out by the LRU replacement algorithm. This suggests that LRU may not be the best cache management policy.

In order to overcome this effect, we propose a *Saturating Priority* cache replacement policy that exploits the disparity between the number of flows and the residency of the digests in the cache. Each entry in a set has a priority that increases each time the entry is accessed, until it reaches a maximum value. Further accesses to the same entry do not increase its priority. Whenever an entry with a lower priority is accessed, it swaps its priority with the entry with next higher precedence. A new entry is added to a set with the least priority after evicting the item with the lowest priority.

Figure 5.4 shows the change in priorities of the entries in a 4-way set associative set. The set initially contains the digests marked *a,b,c* and *d*. Their priorities are also marked in the figure. Here, a larger number signifies a higher priority. When digest d is accessed two times, it swaps its priority with digests c and b respectively. It thus has a priority of 3. The priority of digests b and c decreases. Digest c now has the lowest priority in

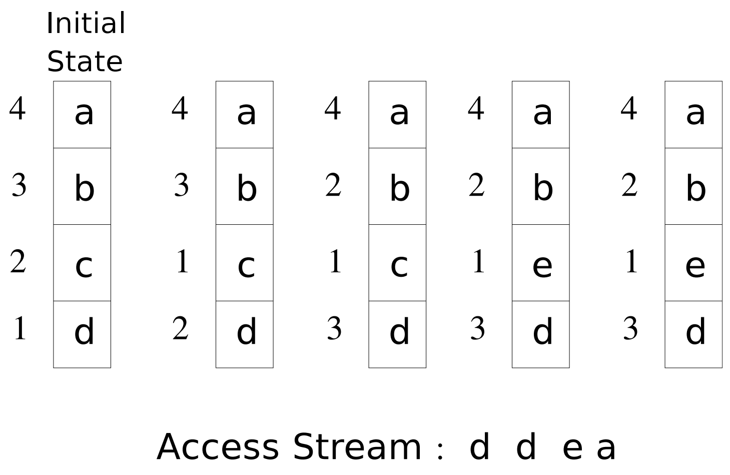


Figure 5.4: Saturating Priority cache replacement policy

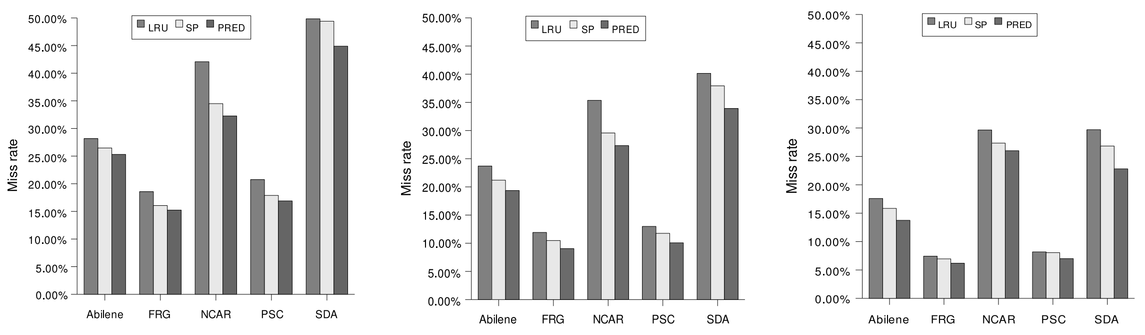
the set. As a result, the miss caused by digest *e* evicts digest *c* from the cache. Access to digest *a* does not increase its priority as it has already got the maximum priority.

Such a cache replacement evicts any entry that was brought into the cache, but is not subsequently accessed. Entries that are accessed only a few times are likely to be evicted. Also in a digest cache, entries are accessed a number of times when the a flow is active, but the accesses stop when the flows end. The cache replacement policy proactively removes such entries as their priority decreases rapidly.

5.4 Performance Evaluation

In order to evaluate the improvement in performance of the digest cache that is possible by preventing the detrimental effect of single packet flows, we implemented an oracle cache management policy that inserts a digest into the cache only when it has more than one access in a window of 10,000 accesses in the future. The cache entries are managed with a LRU policy. We call this PRED policy.

We evaluate the performance of the SP, LRU and PRED replacement policies for different digest cache configurations. We use the traces listed in Table 5.2. The source



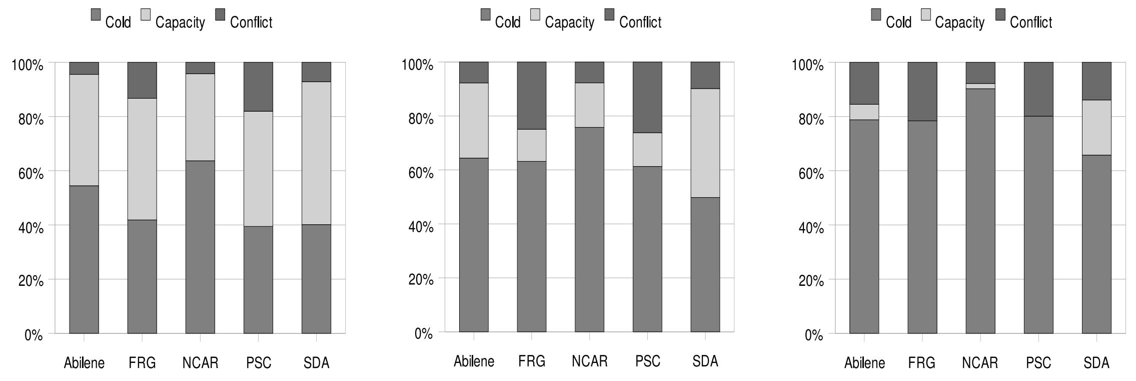
(a) 512 entries (b) 1024 entries (c) 2048 entries

Figure 5.5: 4-way associative digest cache performance

IP address, destination IP address, source port, destination port and protocol id from the packet headers in each packet of the trace was used to calculate the MD5 hash. 32 bits of this hash was used as the digest.

The SP policy outperforms the LRU cache replacement policy in terms of miss rates for all cache sizes evaluated. The infeasible PRED cache management policy has lower miss rate than the other two policies. The performance improvement is dependent on the trace considered and the size of the cache. NCAR trace shows 18% improvement in miss rate for a 512-entry cache and 8% improvement for a 2048-entry cache. For a 512-entry cache, with FRG, NCAR and PSC traces, SP covers more than 74% of the gap between the LRU and PRED cache management policy. Even for higher cache sizes, SP covers substantial gap between LRU and PRED replacement. With PSC trace, SP cache replacement does not show much improvement with large caches, however the miss rate for such caches is already low. For a 512-entry cache with SDA trace, SP replacement policy has a small improvement over LRU replacement policy. But it has 10% improvement over LRU replacement policy for a 2048-entry cache, covering 41% of the gap between LRU and PRED policies.

We see that Abilene, NCAR and SDA traces suffer from more than 13% miss rate even for a 2048-entry cache with LRU cache replacement. In order to understand the reason for this, we classified the misses as cold, capacity and conflict misses. Misses due to accesses that were not present in a window of previous 10,000 accesses are classified as cold misses. We used this definition of cold misses because when network flows stop, their digests are evicted from the cache. When a packet with the same digest is seen next, it is considered a new flow. Accesses to digests that are present in the window of previous 10,000 accesses but not in a fully associative cache are classified as capacity misses. Conflict misses are those which occur when a set associative cache is used instead of a fully associative cache of the same size.



(a) 512 entries (b) 1024 entries (c) 2048 entries

Figure 5.6: Types of misses in digest caches

Conflict misses may be reduced by using a better cache placement and management policy. From Figure 5.6 we observe that for a 1024-entry, 4-way set associative digest cache less than 10% of the misses are due to cache conflicts in case of Abilene, NCAR and SDA traces. Traces that already have a high hit rate, such as FRG and PSC, have about 25% conflict misses. This observation shows that the cold and capacity misses dominate the misses in a digest cache. As expected, for larger caches, the ratio of capacity misses decreases but the number of cold misses does not decrease. This is mainly because of the large number of small flows (refer Figure 5.3). As a result, continuously increasing the cache size leads to small improvements in performance. Instead, it may be worthwhile to use better algorithmic or data structure caching approaches to improve the hit rate.

5.5 Related Work

Kang Li et al. [31] propose the use of digest caches to aid packet classification. They evaluate different associativities, probabilistic insertion and LFU cache replacement policies and different hash functions for these caches. The probabilistic insertion policy only partially addresses the problem of presence of large number of flows. However, the benefit of this algorithm is not significant compared to the LRU replacement policy. On the other hand, we first study the distribution of flow sizes in the internet and their effect on LRU cache replacement. The proposed cache replacement policy exploits the widely observed disparity in flow sizes to find cache entries for replacement.

The characteristics of internet flows have been studied previously by Zhang et al. [47] using traces from different locations in the core and edge of the internet. Disparity in the flow sizes is shown to be more drastic than the disparity in the rate of the flows. They also show that there is a correlation between the size of the flow and its rate. They show that big, fast flows transfer more than 80% of the data in all the traces considered. However, on an average, such flows comprise only 7% of the total number of flows. This disparity in number of packets present in a few flows inspired us to propose a new cache management scheme for digest caches.

Mudigonda et al. [32] propose the use of caches for network processing applications. They show that caching the data structures of the program benefits a wide range of network processing applications. On the other hand, we exploit the traffic patterns observed in real traces to improve the effectiveness of small result caches for packet classification application. Due to the reduction in number of lookups through the slower off chip memory, the processing time reduces. The insight gained from the traffic patterns can also be applied to data structure caches. We leave this to future work.

Qureshi et al. [34], propose a LRU insertion policy (LIP) for level-2 caches in general purpose processors which inserts the cache lines in the least recently used location in the set instead of the MRU location. Lines are moved to the MRU location in case they are accessed in the LRU location. This is aimed at applications that have a larger working set than the cache size. For applications that have a cyclic reference pattern, it prevents thrashing of the cache by retaining some entries in the cache so that they contribute to cache hits. On the other hand, we observe that the large disparity in the flow sizes in the internet leads to poor performance of LRU managed result caches in network applications. We propose a cache replacement policy that quickly recognizes digests belonging to large flows. These entries are retained in the cache with a view to improve cache performance.

5.6 Conclusions

Digest caches provide an effective mechanism to reduce the number of expensive off-chip lookups. However, they suffer from poor performance due to the large number of single packet flows in the internet. We proposed a new cache replacement policy, called Saturating Priority, that overcomes the detrimental effects of these flows on the performance of digest caches. We showed that Saturating Priority covers nearly three fourth of the gap between LRU cache replacement and oracle cache replacement, which places an entry in the cache only when there are multiple packets in the flow.

Chapter 6

Conclusions

6.1 Summary

Performance of a network processor involves modeling a number of parameters such as the NP resources, memory hierarchy and the traffic workload. The interplay between these factors is difficult to analyze analytically or through simple queuing models.

6.1.1 Modeling Packet Buffering

In order to understand the factors affecting packet buffering in network processing applications, we built a detailed model of IPv4 forwarding on the IXP 2400 network processor. This model considered the packet buffer architecture in detail, as it has been shown previously [24, 21] that this step in network processing applications is a performance hurdle.

DRAM bandwidth depends on numerous factors such as the number of banks, page buffer management policy, data bus width, data transfer size and bank conflicts. We built a Petri net model that considered these aspects in detail. This model was integrated with the IPv4 forwarding Petri net model.

Using this model, we showed that narrow accesses to the packet buffers in NPs reduce the bandwidth realized by DRAMs. In order to overcome this problem, we proposed three packet buffer allocation algorithms viz. Header Buffering, First Cell Buffering and First Cell Buffering + Tail Buffering. We showed that these schemes lead to 21% average performance improvement in throughput with real traffic traces. The on-chip memory memory requirement for the proposed schemes is low. Other header processing applications such as network address translation have similar DRAM access characteristics [21] and can benefit from the buffering schemes proposed.

6.1.2 Effect of Workload on Network Processor Resource Utilization

Another aspect that has to be carefully considered while designing Network Processors is the workload characteristics. In networking hardware and applications, worst case traffic is usually considered [39, 21]. However, this could lead to design points that are sub-optimal for real workload traffic. Using an integrated Petri net model, the utilization of NP resources was shown to vary widely with different types of network traffic. We showed that with real traces, the data bus connecting the DRAM to the MEs is the bottleneck, however with a DOS trace, the hash unit is the bottleneck resource.

6.1.3 Packet Buffering for Core Routers

Packet buffering required in the core of the internet varies significantly from the buffering in congested environments. There is low traffic variation over small time intervals of few seconds. Also the output links are over provisioned to take twice the normal traffic. We exploited these characteristics to utilize on-chip memory to satisfy the small buffer requirements. DRAM memory was used only when many packets are received in a burst. We showed that under real traffic conditions, with sufficient number of threads provisioned for packet processing, the on-chip memory may be used to process most of the input traffic. The packet loss rate was less than 1% when the link utilization was 80%. The DRAM bottleneck that was previously encountered was successfully removed.

6.1.4 Improving Performance of Result Caches in Network Processors

There exists a disparity in the number of flows in the internet and their sizes. That is, there are a large number of small flows, but only a few large flows. The few large flows contribute to most of the packets transferred, whereas the fraction of packets transferred by the small flows is small.

In digest caches, which are used for packet classification, the result of the packet classification step is stored with a hash of the packet classification fields. However, due to the presence of large number of flows, the useful digests belonging to large flows are evicted from the cache by the prolific number of small flows.

LRU cache management policy gives highest priority to the most recently accessed digest and tends to retain them in the cache for longer duration. As a result, the digests belonging to a large number of small flows evict the digests belonging to a few large flows. This problem is especially acute in a constrained environment such as a network processor which have small result caches. We proposed a new cache management algorithm called Saturating Priority, wherein the digests are associated with a priority which increases whenever it is accessed in the digest cache, until it reaches a saturating value. This policy ensures that the digests belonging to small flows are evicted quickly.

We showed that the improvement in performance is dependent on the traces considered and the size of the digest cache. It outperformed LRU cache management policy for all cache sizes and traces considered. It covered up to 74% of the gap between LRU and practically infeasible PRED policy. Further, we showed that most of the remaining misses are due to cold and capacity misses, while the contribution of conflict misses was small. The size of the result caches must be increased or new algorithmic methods must be used to improve the performance of packet classification.

6.2 Future Work

The application, viz IPv4 forwarding, that we considered is representative of header processing applications. However, packet processing applications such as IPSec encryption, intrusion detection systems have different memory access characteristics. Optimizing the DRAM bandwidth or provisioning buffers for these applications have to be evaluated in detail.

Also the network processor design that is optimal for these applications could be different from the design suited for header processing applications. Finding optimal design points suited for different applications is left for future work.

The flow distribution observed in the internet have a profound effect on traffic aware caching schemes that aim to improve network application performance. We considered only result caches in our study. The effect of flow distribution on schemes that cache the lookup data structures has be evaluated. This may help to find cache organizations and management policies that achieve higher performance in network processing scenarios.

Bibliography

[1] MT46V64M4 DDR SDRAM. http://micron.com.

[2] National Laboratory for Applied Network Research. http://pma.nlanr.net.

[3] QDR SRAM - The High Bandwidth SRAM family. http://www.qdrsram.com.

[4] C-5 Network Processor D0 Architecture Guide. http://e-www.motorola.com/collateral/C5NPD0-AG.pdf, 2001.

[5] CSIX-L1: Common Switch Interface Specification-L1. http://www.oiforum.com/public/documents/csixL1.pdf, 2001.

[6] A. Agarwal. Performance Tradeoff in Multithreaded Processors. In *IEEE Transactions on Parallel and Distributed Systems*, volume 3, pages 525–539, Sep 1992.

[7] B. Agrawal and T. Sherwood. Virtually pipelined network memory. In *MICRO 39: Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 197–207, Washington, DC, USA, 2006. IEEE Computer Society.

[8] J. R. Allen, B. M. Bass, C. Basso, R. H. Boivie, J. L. Calvignac, G. T. Davis, L. Frelechoux, M. Heddes, A. Herkersdorf, A. Kind, J. F. Logan, M. Peyravian, M. A. Rinaldi, R. K. Sabhikhi, M. S. Siegel, and M. Waldvogel. IBM PowerNP network processor: Hardware, software, and applications. *IBM J. Res. Dev.*, 47(2-3):177–193, 2003.

[9] G. Appenzeller, I. Keslassy, and N. McKeown. Sizing router buffers. In *SIGCOMM '04: Proceedings of the 2004 conference on Applications, technologies, architectures,*

and protocols for computer communications, pages 281–292, New York, NY, USA, 2004. ACM.

[10] F. Baboescu and G. Varghese. Scalable packet classification. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 199–210, New York, NY, USA, 2001. ACM.

[11] J-L. Baer, D. Low, P. Crowley, and N. Sidhwaney. Memory Hierarchy Design for a Multiprocessor Look-up Engine. In *PACT '03: Proceedings of the 12th International Conference on Parallel Architectures and Compilation Techniques*, page 206, Washington, DC, USA, 2003. IEEE Computer Society.

[12] C. Barakat, P. Thiran, G. Iannaccone, C. Diot, and P. Owezarski. A flow-based model for internet backbone traffic. In *IMW 02: Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*, pages 35–47, New York, NY, USA, 2002. ACM.

[13] F. Chang, Wu chang Feng, Wu chi Feng, and K. Li. Efficient packet classification with digest caches. In *NP 3: Workshop on Network Processors and Applications*, 2004.

[14] P. Crowley, M.A. Franklin, H. Hadimioglu, and P. Onufryk. *Network Processor Design : Issues and Practices*, volume 1. Morgan Kauffmann, 2003.

[15] V. Cuppu, B. Jacob, B. Davis, and T. Mudge. A performance comparison of contemporary dram architectures. *SIGARCH Computer Architecture News*, 27(2):222–233, 1999.

[16] A. Dhamdhere, H. Jiang, and C. Dovrolis. Buffer sizing for congested internet links. In *INFOCOM 2005: 24th Annual Joint Conference of the IEEE Computer and Communications Societies*, pages 1072–1083. Proceedings IEEE, 2005.

[17] N. Dukkipati, Y. Ganjali, and R. Zhang-Shen. Typical versus worst case design in networking. In *HotNets 05: Proceedings of the 4th ACM SIGCOMM Workshop on Hot Topics in Networks*, New York, NY, USA, 2005. ACM.

[18] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Trans. Netw.*, 1(4):397–413, 1993.

[19] M. Franklin and T. Wolf. A Network Processor Performance and Design Model with Benchmark Parameterization. In *NP 1: Workshop on Network Processors and Applications*, 2002.

[20] J. Garca, J. Corbal, L. Cerd, and M. Valero. Design and implementation of high-performance memory systems for future packet buffers. In *MICRO 36: Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture*, page 373, Washington, DC, USA, 2003. IEEE Computer Society.

[21] S. Govind and R. Govindarajan. Performance modeling and architecture exploration of network processors. In *QEST '05: Proceedings of the Second International Conference on the Quantitative Evaluation of Systems (QEST'05) on The Quantitative Evaluation of Systems*, page 189, Washington, DC, USA, 2005. IEEE Computer Society.

[22] R. Govindarajan, F. Suciu, and W.M. Zuberek. Timed Petri net models of multithreaded multiprocessor architectures. In *PNPM 1997 : Proceedings of Seventh International Workshop on Petri Net and Performance Models*, 1997.

[23] P. Gupta and N. McKeown. Algorithms for packet classification. *IEEE Network*, 12(2):24–32, 2001.

[24] J. Hasan, S. Chandra, and T. N. Vijaykumar. Efficient use of memory bandwidth to improve network processor throughput. In *ISCA '03: Proceedings of the 30th annual international symposium on Computer architecture*, pages 300–313, New York, NY, USA, 2003. ACM Press.

[25] Intel. *Intel IXP2400 Network Processor Hardware Reference Manual*, November 2003.

[26] Intel. *Intel IXP2400 Network Processor Development Tools User's Guide*, March 2004.

[27] S. Iyer, R. Kompella, and N. McKeown. Analysis of a memory architecture for fast packet buffers. citeseer.ist.psu.edu/iyer01analysis.html, 2001.

[28] R. Jain. *The Art of Computer Systems Performance Analysis.* John Wiley and Sons, 1991.

[29] K. Jensen. A brief introduction to colored petri nets. In *TACAS 97: Proceedings of TACAS-1997 Workshop*, pages 2003–208. Springer Verlag, 1997.

[30] E. J. Johnson and A. R. Kunze. *IXP 2400/2800 Programming.* Intel Press, 2003.

[31] K. Li, F. Chang, D. Berger, and Wu chang Feng. Architectures for packet classification caching. In *ICON 2003 : The 11th IEEE International Conference on Networks*, pages 111–117, 28 Sept.-1 Oct. 2003.

[32] J. Mudigonda, H. M. Vin, and R. Yavatkar. Overcoming the memory wall in packet processing: hammers or ladders? In *ANCS '05: Proceedings of the 2005 symposium on Architecture for networking and communications systems*, pages 1–10, New York, NY, USA, 2005. ACM Press.

[33] J. L. Peterson. Petri Nets. *ACM Comput. Survey*, 9(3):223–252, 1977.

[34] M. K. Qureshi, A. Jaleel, Y. N. Patt, S. C. Steely, and J. Emer. Adaptive insertion policies for high performance caching. In *ISCA '07: Proceedings of the 34th annual international symposium on Computer architecture*, pages 381–391, New York, NY, USA, 2007. ACM.

[35] S. Rixner, W. J. Dally, U. J. Kapasi, P. Mattson, and J. D. Owens. Memory access scheduling. In *ISCA '00: Proceedings of the 27th annual international symposium on Computer architecture*, pages 128–138, New York, NY, USA, 2000. ACM Press.

[36] S. Schenker, L. Zhang, and D. D. Clark. Some observations on the dynamics of a congestion control algorithm. *SIGCOMM Comput. Commun. Rev.*, 20(5):30–39, 1990.

[37] R. N. Shorten and D. J. Leith. On queue provisioning, network efficiency and the transmission control protocol. *IEEE/ACM Tran. on Net.*, 15(4):866–877, 2007.

[38] R. Sinha, C. Papadopoulos, and J. Heidemann. Internet packet size distributions: Some observations. http://netweb.usc.edu/∼rsinha/pkt-sizes.

[39] T. Spalink, S. Karlin, and L. Peterson. Evaluating network processors in IP forwarding. Technical Report TR-626-00, Princeton University, 2000.

[40] V. Srinivasan, G. Varghese, S. Suri, and M. Waldvogel. Fast and scalable layer four switching. *SIGCOMM Computer Communication Review*, 28(4):191–202, 1998.

[41] C. Villamizar and C. Song. High performance TCP in ANSNET. *SIGCOMM Computer Communications Review*, 24(5):45–60, 1994.

[42] N. Viswanadham and Y. Narahari. *Performance Modeling of Automated Manufacturing Systems.* Prentice-Hall of India Private Limited, 1994.

[43] D. Wang, B. Ganesh, N. Tuaycharoen, K. Baynes, A. Jaleel, and B. Jacob. DRAMsim: a memory system simulator. *SIGARCH Comp. Arch. News*, 33(4):100–107, 2005.

[44] David Wang. *Modern Dram Memory Systems: Performance Analysis And A High Performance, Power-Constrained DRAM Scheduling Algorithm.* PhD thesis, University of Maryland, 2005.

[45] T. Wolf and M. A. Franklin. Design tradeoffs for embedded network processors. In *ARCS '02: Proceedings of the International Conference on Architecture of Computing Systems*, pages 149–164, London, UK, 2002. Springer-Verlag.

[46] L. Zhang and D. D. Clark. Oscillating behaviour of network traffic: A case study simulation. *Internetworking: Research and Experience*, pages 101–112, 1990.

[47] Y. Zhang, L. Breslau, V. Paxson, and S. Shenker. On the characteristics and origins of internet flow rates. In *SIGCOMM '02: Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 309–322, New York, NY, USA, 2002. ACM.

[48] W. M. Zuberek. Modeling using timed petri nets - event-driven simulation. Technical Report Technical Report No. 9602, Memorial Univ. of Newfoundland, 1996.