Analysis of a Memory Architecture for Fast Packet Buffers

Sundar Iyer, Ramana Rao Kompella, Nick McKeown
Computer Systems Laboratory, Stanford University,
Ph: (650)-725 9077, Fax: (650)-725 6949
Stanford, CA 94305-9030
{sundaes, ramana, nickm}@stanford.edu

Abstract -- **All packet switches contain packet buffers to hold packets during times of congestion. The capacity of a high performance router is often dictated by the speed of its packet buffers. This is particularly true for a shared memory switch where the memory needs to operate at N times the line rate, where N is the number of ports in the system. Even input queued switches must be able to buffer packets at the rate at which they arrive. And so as link rates increase memory bandwidth requirements grow. With today's DRAM technology and for an OC192c (10Gb/s) link, it is barely possible to write packets to (read packets from) memory at the rate at which they arrive (depart). As link rates increase, the problem will get harder. There are several techniques for building faster packet buffers, based on ideas from computer architecture such as memory interleaving and banking. While not directly applicable to packet switches, they form the basis of several techniques in use today. In this paper we consider one particular packet buffer architecture consisting of large, slow, low cost, DRAMs coupled with a small, fast SRAM "buffer". We describe and analyze a memory management algorithm (ECQF-MMA) for replenishing the cache and find a bound on the size of the SRAM.**

***Keywords*--Memory Buffers, Packet Buffers, Memory Banks, Striping, Cell Interleaving.**

I. INTRODUCTION

A fundamental requirement of a packet switch (e.g. an Internet router or ATM switch) is its need to buffer packets. Often, the performance of a packet switch is limited by the speed at which it can buffer packets. This paper is about one practical method to overcome this limitation. There are, in fact, several approaches to this problem but due to the proprietary nature of different commercial solutions, we are not aware of any quantitative analysis or comparison of different techniques. While this paper describes and analyzes one packet buffer architecture, it is the longer term goal of our work to quantitatively compare a variety of techniques.

There are certain characteristics common to packet buffers in almost all types of packet switch. First, as the line rate R increases, the memory bandwidth[1] of the packet buffers must increase. For example, if a packet switch with N ports buffers packets in a single shared memory, then it requires a memory bandwidth of $2NR$. If on the other hand, the packet switch maintains a separate packet buffer for each input, it requires a memory bandwidth of $2R$. In both cases, the memory bandwidth scales linearly with the line rate.

Second, interfaces with faster line rates require larger buffers. As a rule-of-thumb, packet switches employ buffers of size approximately $RTT \times R$ (where RTT is the round trip time for flows passing through the packet switch) for those occasions when the packet switch is the bottleneck for the (TCP) flows passing through it. With an Internet RTT of approximately 0.25 seconds today, a 10Gb/s interface requires 2.5Gbits of memory. Because today this is bigger than a reasonably priced SRAM buffer (and because each successive generation of interface requires more memory), packet buffers are made from more cost-effective, lower power, but slower, DRAM whenever possible.

Third, packet buffers are arranged as one or more first-in first-out (FIFO) queues. For example, a shared memory switch maintains at least one FIFO for each output line; and an input-buffered packet switch usually maintains virtual output queues [1]. If the packet switch performs per-flow or per-class queueing, the number of FIFO queues can become very large.[2]

Fourth, the sequence in which packets are read from the buffer is determined by a scheduling algorithm. For example, in a shared memory switch, the time at which each packet is read from memory is determined by an output-link scheduler (e.g. a WFQ scheduler [2][3]). Similarly, in a packet switch with input buffers, the time at which each packet is read from memory is determined by an arbiter that determines the configuration of a switch fabric. In both types of switch we can think of there being an arbiter that requests a particular packet be retrieved from memory and delivered within a fixed time. Although deterministic, the sequence of requests is not known to the buffer manager at the time packets are written into memory. As far as the buffer manager is concerned, the sequence of requests is unpredictable, yet it is required to retrieve the packet within a fixed time. This is quite different behavior from how data is read from main memory in a computer — in a computer, it is considered acceptable if the time to retrieve data is variable.[3]

Last, it is common for packets to be segmented into fixed size units prior to storage. This is done for several reasons: (1) Mem-

1. The memory bandwidth is defined here to be the reciprocal of the time taken to write data to, or read data from a random location in memory. For example, a 32-bit wide memory with a 100ns random access time is said to have a memory bandwidth of 320Mb/s.

2. In packet switches that maintain per-flow FIFOs, the number of queues grows with R. This is because a faster line is likely to carry a larger number of multiplexed flows. OC192c line interfaces for ATM commonly maintain 256k queues or more.

3. Several factors lead to variations in the retrieval time. The data may or may not be cached. If cached, it may be in the primary or secondary cache. If not cached, the data may be retrieved from the same page as the previous data (which may mean a faster memory access). Alternatively, the data may be held up while the memory is refreshed.

ory can be utilized more efficiently if all buffers are the same size; (2) Switch fabrics generally operate on fixed size data units anyway; and (3) Arriving packets may in fact be 53-byte ATM cells. Throughout this paper, we will refer to the fixed size segments as “cells” of size C, even though they need not be equal in size to an ATM cell.[4]

In summary, packet buffers consist of multiple FIFO queues stored in DRAM that contain fixed sized cells. Given an unpredictable sequence of requests from an arbiter, the packet buffer must be able to retrieve cells within a fixed time period. The memory bandwidth and size of the packet buffer both increase linearly with R.

It is worth considering how fast a packet buffer must operate for different line rates. For example, a packet buffer for a single 10Gb/s interface (OC192c SONET or 10Gb/s Ethernet) requires a memory bandwidth of at least 20Gb/s. If cells are 64 bytes long and are written to and read from memory in one transaction (i.e. the memory is one cell “wide”), then one memory operation is required every 25.6ns, which is faster than the random access time of DRAMs today. If the line rate is 40Gb/s (e.g. OC768c SONET), then a memory operation is required every 6.4ns, much faster than commercial DRAM today, or any expected in the near future.

In this paper, we will describe and analyze a mechanism in which memory bandwidth is increased by reading (writing) *multiple* cells from (to) memory in parallel (i.e. there is a single address bus that controls all memory devices in parallel). In other words, when the unpredictable arbiter requests a cell, multiple cells are read from DRAM at the same time. The additional cells are stored temporarily in an SRAM until they are required. We can think of this memory hierarchy as a large DRAM containing a set of FIFOs; the head and tail of each FIFO is cached in a (possibly on-chip) SRAM as shown in Figure 1. The SRAM is sized so that whenever the arbiter requests a cell, it is always delivered within a bounded delay, regardless of the sequence of requests.

In this paper we will describe an algorithm that minimizes the SRAM size while guaranteeing a bounded latency.[5] As we shall see, the size of the SRAM is dictated by a memory management algorithm (MMA) that determines the order in which cells are read from the DRAM.

It should be noted that the general packet buffer architecture described here is not novel, and we believe it to be in use in several proprietary, but undocumented packet switches. Some previous work on a related problem is available in [4][5][6]. Our goal here is to systematically study and analyze this memory architecture for the benefit of designers who might use it in their designs. In particular, we will determine how large the SRAM needs to be, and propose algorithms for deciding when to replenish the SRAM cache so as to minimize its size, or to minimize latency.

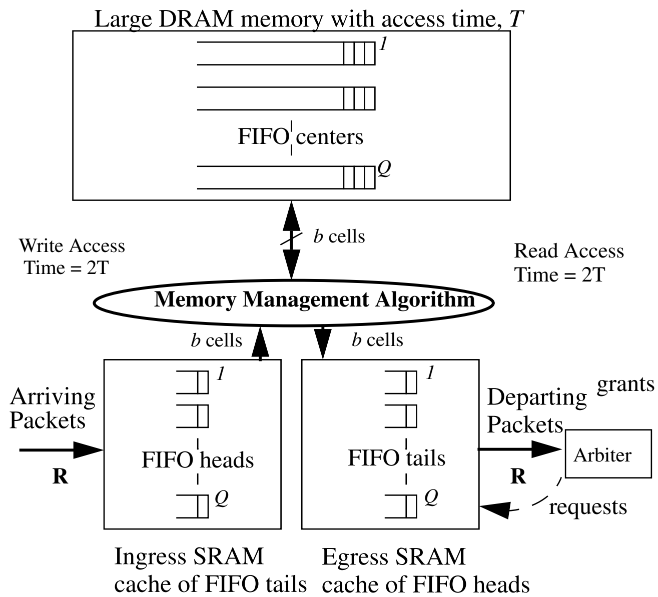

Figure 1: Memory hierarchy of packet buffer, showing large DRAM memory with heads and tails of each FIFO maintained in a smaller SRAM cache.

On the face of it, the scheme is similar to traditional data striping, banking, interleaving or pre-fetching used in computer systems [8][9][10][11]. Some other approaches that use Rambus DRAM controllers [12][13] and other techniques that hide latency are described in [14][15][16][17]. However, unlike computer systems, packet switches cannot tolerate bank conflicts that occasionally allow the data to be delivered at an unpredictable time. Also, simply striping a cell across multiple memory banks is not much use when the cell size is quite small. Another approach is to use statistical memory banking with pseudo-random functions or hash functions [18][19][20] so that the probability of the worst case request patterns is low. While this may work well most of the time, it does not support worst case patterns and does not guarantee a bound on the access time for each cell.

The main result of this paper is that if the DRAM contains Q FIFO queues, there exists a MMA for which an SRAM of size $A = Q(b-1)$ cells for each of the ingress and egress SRAM caches, is sufficient to bound the time from when the arbiter requests a cell to when the cell is retrieved. b is the number of cells written to or read from memory in each operation, as shown in Figure 1. The result holds for any arbiter and for any packet arrival/departure process. The maximum time from when a request is received until the cell is retrieved from DRAM is $Q(b-1)+1$ time slots, where one time slot is the time for a cell to arrive.

As an example of how these results may be used, consider an

[4] It is common in practice for cells to be 64 bytes long as this is the first power of 2 above the sizes of ATM cells and minimum length TCP segments (40 bytes).

[5] The special case when the latency is zero is described in [7].

input-queued router with 32 ports, each operating at 40Gb/s (OC768c). Each input maintains 32 virtual output queues in DRAM with a random access time of $51.2ns$ (requiring $b \geq 8$) and segments each packet into 64byte cells. Our results indicate that such a packet buffer could be built with 115Kbits of SRAM for each of the ingress and egress SRAM caches, and a latency bound of $2.9us$ for each cell.

II. A Memory Management Algorithm

A. Introduction

There are many alternative algorithms for deciding when to write data to, and read data from, the DRAM buffer, and the algorithm will determine the amount of SRAM needed. In what follows, we will first describe the memory hierarchy in more detail, then describe the ECQF-MMA (Earliest Critical Queue First Memory Management Algorithm) and show that it minimizes the size of the SRAM buffer.

B. Memory Hierarchy

Referring again to Figure 1, cells of size C arrive at rate R and are buffered in the ingress SRAM buffer, where they wait for the MMA to write them into a large FIFO in DRAM. All b cells written at one time are written to the tail of the same DRAM FIFO. Similarly, the MMA reads cells from the DRAM as needed, and places them into the corresponding FIFO in the egress SRAM buffer. Again, b cells are read at a time, all from the same FIFO in the DRAM. Each time the arbiter requests a cell, it is delivered from the SRAM buffer at rate R.[6]

So, the head and tail of each FIFO resides in SRAM, whereas the middle portion of the FIFO resides in DRAM. Of course, if the FIFO has few cells, they may all reside in SRAM.

We'll assume that each DRAM read and write operation takes T seconds, which is the random access time of the DRAM (i.e. the maximum time taken to access any location in the memory array). Each cell that is written into DRAM is read from DRAM sometime later; i.e. the number of write and read operations are equal (to be contrasted with a CPU in which data is commonly written once and read many times). We can therefore think of the memory operating in cycles of length $2T$, where each cycle consists of one write and one read. So an obvious requirement for the packet buffer to sustain the line rate R is that $b \geq 2RT/C$. In this paper, we take it as a requirement to minimize the memory width, and so we assume $b = 2RT/C$.

[6] For clarification, if a request arrives for a cell that is still in the ingress SRAM buffer (i.e. because the cell's FIFO contains fewer than b cells), then the cell is read directly from the ingress SRAM buffer.

C. What affects the size of the SRAM buffer?

If the packet buffer maintained just one FIFO queue, the operation would be simple: Each time b cells arrived at the ingress SRAM, they would be written into DRAM. This would require $b-1$ cells of storage in the ingress buffer so as to store the cells which arrived before the b^{th} cell. Similarly, the egress buffers would require $b-1$ cells of storage. When the egress SRAM buffer receives a request from the arbiter, the MMA reads b cells from DRAM, grants one cell to the arbiter and stores the remaining $b-1$ cells in SRAM. Every time b new requests arrive from the arbiter, b new cells are read from DRAM.

When there are more FIFOs the system is more complicated. For example, with Q FIFOs let's consider what happens as cells depart from the egress SRAM buffer. When a cell departs it may trigger the need to replenish the FIFO to prevent under-run of the FIFO in the future (i.e. a request arrives to find that the cell is in DRAM). If consecutively departing cells cause different FIFOs to need replenishment, then a queue of read requests will form waiting for cells to be retrieved from DRAM. If a read request is queued too long, a FIFO in the egress SRAM buffer might not be replenished before it under-runs. Put another way, the egress SRAM buffer might have to contain a very large number of cells for each FIFO so as to hold sufficient reserves to cover the worst case sequence of departures.

Fortunately, a MMA need not issue read requests to DRAM in the same order that FIFOs become depleted. If the MMA has some knowledge of how quickly a FIFO needs to be replenished, it can give priority to FIFOs with a more urgent need for replenishment. How does the MMA know how urgently a FIFO needs to be replenished? In the scheme considered here, the MMA uses a *lookahead buffer* of requests from the arbiter. The MMA uses the lookahead buffer to peek at which FIFOs are receiving the most requests. The lookahead buffer increases the latency from when a request is issued, until the cell is delivered. We believe that if the latency is small and bounded, it will be acceptable in most applications. In what follows, it will help if we clearly define the lookahead buffer and the latency in this context.

Definition 1: ***Lookahead** (L_t): The lookahead L_t is defined as the number of time slots in the future for which the arbiter's request pattern is known.*

Definition 2: **Latency,** *L: The latency of a request is the time elapsed from when the arbiter issues the request, until the cell is granted to the arbiter from the SRAM. We assume that the time to read cells from the SRAM buffer is negligible (i.e. zero).*

In our discussion above, we focussed on the egress SRAM buffer. It is interesting to note that the ingress and egress SRAM buffers are symmetrical in the sense that the ingress (egress) buffer has an unpredictable arrival (departure) process, and predictable departure (arrival) process, respectively. It turns out that, for a given MMA, the ingress and egress buffers are the same size. So for brevity, we will only analyze the egress buffer — the results extend simply to the ingress buffer.

D. Definitions

In what follows, we will use the following definitions.

Definition 3: ***Occupancy Counter*** $Q(i, t)$ *: Queue i 's occupancy counter at time t reflects the number of cells present in the egress SRAM for FIFO queue i.*
When a request arrives and a cell is delivered, $Q(i, t)$ is decremented by one. When b cells are read from DRAM and placed into the SRAM $Q(i, t)$ is incremented by b.

Definition 4: ***Critical Queue****: A queue i is said to be critical at time t if the queue has more requests in the lookahead buffer than it has cells in the SRAM.*

Definition 5: ***Earliest Critical Queue:*** *The earliest critical queue is the queue (from among those that are critical) that turns critical the earliest.*

Definition 6: ***Dynamic Shared SRAM Buffer:*** *Throughout this paper we shall assume that the SRAM consists of a buffer space which can be shared by cells of all queues.*

E. A necessity bound on the size of the egress SRAM buffer.

Theorem 1:(Necessity) An egress SRAM buffer of size $Q(b-1)$ cells is necessary for any MMA to service a sequence of requests within a bounded latency.

Proof: Consider the case when the FIFOs in DRAM are all non-empty. Let the arbiter request one cell from each queue in turn and make no more requests. Assume that each request leads to the retrieval of b new cells from the DRAM. The egress buffer sends one cell to the arbiter and must store the remaining $b-1$ cells for every queue. Hence the egress buffer must be at least of size $Q(b-1)$ cells. ❐

III. An MMA that Minimizes the size of the SRAM Buffer.

We now describe an MMA that minimizes the size of the SRAM buffer while bounding the latency. We call it Earliest Critical Queue First MMA (ECQF-MMA).

A. ECQF-MMA

ECQF-MMA uses a lookahead buffer to hold the unpredictable stream of requests from the arbiter.

Algorithm Description: Every cell time, ECQF-MMA has the opportunity to read from DRAM and decide which, if any, FIFO queue in the egress SRAM buffer to replenish. The algorithms simply reads b cells from the earliest critical queue and places them in the SRAM.

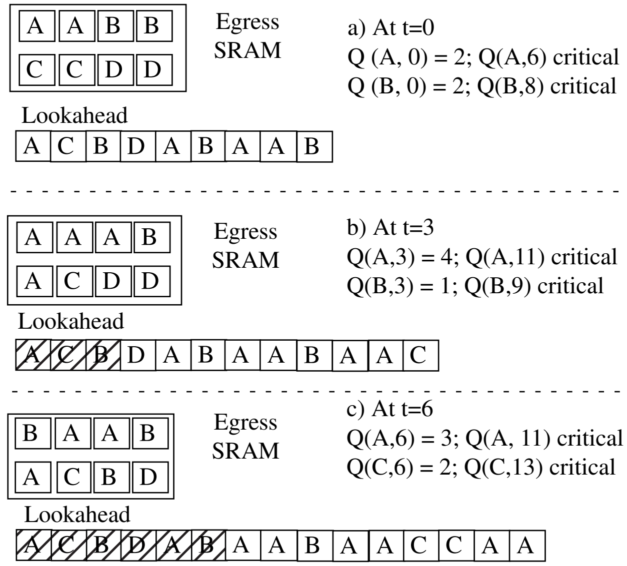

Figure 2: The ECQF-MMA algorithm with $Q = 4$ and $b = 3$. The size of the egress SRAM is 8 cells and the lookahead is 9 cells.

Example of ECQF-MMA: Figure 2 shows an example of ECQF-MMA with $Q = 4$ and $b = 3$. Figure 2a shows that the MMA at time $t = 0$ computes that queues A, B will become critical at time $t = 6$ and $t = 8$ respectively. Since A is the earliest critical queue, it is chosen for service from the DRAM. Cells from queues A, C, B leave the egress SRAM at times $t = 0, 1, 2$.

In Figure 2b the ECQF-MMA determines that B is the earliest critical queue and it is chosen for service from the DRAM. Cells from queues D, A, B leave the egress SRAM at times $t = 3, 4, 5$.

The occupancy of the egress SRAM at time $t = 6$ is shown in Figure 2c. Queue A is again the earliest critical queue and is chosen for service from the DRAM.

We now derive the size of the egress SRAM required for ECQF-MMA. In the following section we shall make three simplifying assumptions. We shall see later how these assumptions can be relaxed.

Assumption 1. (Queues Initially Full) At time $t = 0$, the egress SRAM is full and has $b-1$ cells in each queue i.e. the occupancy counter for each queue is $b-1$ and hence the total occupancy of the SRAM is $Q(b-1)$.

Assumption 2. (Queues Never Empty) Whenever the MMA decides to refresh a queue, there are always cells present in the DRAM for that queue.

Assumption 3. (Request Every Time Slot) The arbiter issues a new request every time slot.

B. Is there always an earliest critical queue to be read?

ECQF-MMA requires that there is always an earliest critical queue to read from.

Lemma 1: A lookahead of $L_t = Q(b-1) + 1$ time slots is sufficient to guarantee at least one critical queue.

Proof: The proof is by the pigeon-hole principle. We start with an egress SRAM of size $Q(b-1)$. This implies that the sum of the occupancy counters of all queues is $Q(b-1)$. The sum of occupancy counters of all the queues remains at $Q(b-1)$, since a set of b cells arrive after every b time slots and exactly b cells depart the system in this period. Since there are $Q(b-1) + 1$ time slots in the lookahead buffer, deducting the number of arbiter requests in this lookahead buffer for each queue would leave at least one queue with no cells and one arbiter request unsatisfied, making it critical. Hence, there is at least one critical queue which implies that there is always an earliest critical queue. ❐

Now we are ready to state the main theorem.

Theorem 2:An egress SRAM buffer of size $Q(b-1)$ and a lookahead of size $Q(b-1) + 1$ is sufficient for ECQF-MMA to service any sequence of arbiter requests with a latency of $Q(b-1) + 1$ time slots.

Proof: The proof is in two parts. First we show that the egress SRAM buffer does not overflow. Second we must show that each cell is delivered within a bounded latency of $Q(b-1) + 1$ time slots from when its request was issued.

To prove the first part, we know that with the assumptions 2 and 3 made in Section IIIA and from Lemma 1, the ECQF-MMA always reads exactly b cells from the earliest critical queue every b time slots. Thus the total occupancy of the egress SRAM buffer does not change. Since, this is true for all time slots the size of the egress SRAM buffer does not grow larger than $Q(b-1)$.

To prove the second part, for every arbiter request in the lookahead buffer the corresponding cell is either present or not present in the SRAM. No latency is encountered by a cell that is present in the SRAM. If the cell is not present in the SRAM, the occupancy counter is smaller than the number of requests in the lookahead buffer and the queue is critical. Suppose that the total number of queues that have ever become critical before this queue is p. Then, this cell could not have arrived earlier than $(p+1)b$ time slots from the start. The DRAM would have taken no more than pb time slots to service all these earlier critical queues. At least b time slots before this arbiter request arrives, the ECQF-MMA would have identified this as the earliest critical queue and would have serviced it, thereby ensuring that the corresponding cell is present in the egress SRAM.

Hence, by the time a request reaches the head of the lookahead buffer, the cell is always present in SRAM. Hence, the latency is bounded by the depth of the lookahead buffer: $Q(b-1) + 1$ time slots. ❐

C. Relaxing the assumptions

We now show that Theorem 2 holds when our assumptions are relaxed. In order to do this we shall look upon the cells in the previous section as "virtual" cells as defined below.

*Definition 7: **Cell Placeholder:** The space that will be occupied by a cell that is either in the ingress SRAM or has yet to arrive to the system. In particular the egress SRAM consists of only cell placeholders at time $t = 0$.*

*Definition 8: **Virtual Cells:** The sum of both "real" cells present in the DRAM and egress SRAM and cell placeholders.*

D. Generalized Scenario

Assumption 1. (Queues Initially Full) At $t = 0$ each queue contains $b-1$ virtual cells; hence this assumption holds for virtual cells.

Assumption 2. (Queues Never Empty) Whenever no "real" cells are available in the DRAM, assume that b cell placeholders are read from DRAM. We introduce a slight modification to the memory architecture. Assume that the ingress SRAM always fills the cell placeholders in the egress SRAM with their corresponding cells before writing to the DRAM. Thus the assumption is true for virtual cells.

Assumption 3. (Request Every Time Slot) Consider an additional $(Q+1)^{st}$ "*ghost*" queue. When the arbiter has no request, we can consider there to be a request to the ghost queue. Thus the lookahead buffer always contains $(Q+1)(b-1) + 1$ requests and the size of the egress SRAM buffer is bounded by $(Q+1)(b-1)$ cells (which is $b-1$ more cells that before). The analysis to derive a tighter bound of $Q(b-1)$ on the size of the egress SRAM is more complex and for brevity we do not include it here.

IV. Conclusions

Packet switches, regardless of their architecture, require packet buffers. The general architecture presented here can be used to build high bandwidth packet buffers for any traffic arrival pattern or packet scheduling algorithm. The scheme uses a number of DRAMs in parallel, all controlled from a single address bus. The costs of the technique are: (1) a (presumably on-chip) SRAM cache that grows in size linearly with line rate and the number of queues, (2) A non-zero, but bounded latency, from when requests are made until packets are available, (3) A lookahead buffer to hold $Q(b-1) + 1$ requests, and (4) A memory management algorithm that must be implemented in hardware.

While there are systems for which this technique is inapplicable (e.g. systems for which the number of queues is very large, or where the line-rate requires too large a value for b, so that the SRAM cannot be placed on chip), the technique can be used to build packet buffers faster than any commercially available today. For example, a 100Gb/s shared memory router that maintains 512 queues might use DRAM operating at $51.2ns$, $b = 20$ and 1.25Mbytes of SRAM. Or an input queued router with a 40Gb/s linecard that maintains 512 virtual output queues, $b = 8$ and 0.5Mbytes of SRAM.

REFERENCES

[1] Y. Tamir et al. “Symmetric crossbar arbiters for VLSI communication switches,” *IEEE Transactions on Parallel and Distributed Systems*, vol.4, no.1, Jan 1993, pp. 13-27.

[2] A. Demers, S. Keshav, and S. Shenker, “Analysis and simulation of a fair queuing algorithm,” *ACM Computer Communication Review* (*SIGCOMM'89*), pp. 3-12, 1989.

[3] A. K. Parekh and R. G. Gallager, “A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Single Node Case,” *IEEE/ACM Transaction on Networking*, Vol. 1, No. 3, pp. 344-357, June 1993.

[4] A. Birman, H.R. Gail, S.L. Hantler and Z. Rosberg, “An optimal service policy for buffer systems,” *Journal of the Association for Computing Machinery*, pp. 641-57, vol. 42, no. 3, May 1995.

[5] H. Gail, G. Grover, R. Guerin, S. Hantler, Z. Rosberg, M. Sidi, “Buffer size requirements under longest queue first,” *Proceedings IFIP'92*, vol. C-5, pp. 413-24, 1992.

[6] G. Sasaki, “Input buffer requirements for round robin polling systems,” *In Proceedings of 27th Annual Conference on Communication, Control and Computing*, pp. 397-406, 1989.

[7] S. Iyer, R. R. Kompella, and N. McKeown, “Techniques for fast packet buffers,” *In. Proceedings. of GBN 2001*, Anchorage, Apr. 2001.

[8] P. Chen and David A. Patterson, “Maximizing Performance in a Striped Disk Array,” *ISCA*, pp. 322-331, 1990.

[9] Y. Joo and N. McKeown, “Doubling Memory Bandwidth for Network Buffers,” *Proc. IEEE Infocom* 1998, vol. 2, pp. 808-815, San Francisco.

[10] D. Patterson, and J. Hennessy, *Computer Architecture: A Quantitative Approach*, 2nd. ed., San Francisco: Morgan Kaufmann Publishers, c1996.

[11] T. Alexander and G. Kedem, “Distributed Prefetch-buffer/Cache Design for High Performance Memory Systems,” *In. Proceedings. of the 2nd International Symposium on High-Performance Computer Architecture*, pp. 254-263, Feb. 1996.

[12] W. Lin, S. Reinhardt, D. Burger, “Reducing DRAM Latencies with an Integrated Memory Hierarchy Design,” *In Proc. 7th Int symposium on High-Performance Computer Architecture*, January 2001.

[13] S.I. Hong, S.A. McKee, M.H. Salinas, R.H. Klenke, J.H. Aylor, and Wm.A. Wulf, “Access order and effective bandwidth for streams on a direct rambus memory,” *In Proceedings of the Fifth International Symposium on High Performance Computer Architecture*, pp. 80-89, January 1999.

[14] J. Corbal, R. Espasa, and M. Valero, “Command vector memory systems: High performance at low cost,” *In Proceedings of the 1998 International Conference on Parallel Architectures and Compilation Techniques*, pp. 68-77, October 1998.

[15] B. K. Mathew, S. A. McKee, J. B. Carter, and A. Davis, “Design of a parallel vector access unit for SDRAM memory systems,” *In Proceedings of the Sixth International Symposium on High Performance Computer Architecture*, January 2000.

[16] S. A. McKee and Wm. A. Wulf, “Access ordering and memory conscious cache utilization,” *In Proceedings of the First International Symposium on High Performance Computer Architecture*, pp. 253-262, January 1995.

[17] S. Rixner, W. J. Dally, U. J. Kapasi, P. Mattson, and J. D. Owens, “Memory access scheduling,” *In Proceedings of the 27th Annual International Symposium on Computer Architecture*, pp. 128-138, June 2000.

[18] L. Carter and W. Wegman, “Universal Hash Functions,” *Jour. of Computer and System Sciences* 18, 1979, pp. 143-154.

[19] R. Impagliazzo and D. Zuckerman, “How to Recycle Random Bits,” *Proc. of the Thirtieth Annual Symposium on the Foundations of Computer Science,* Research Triangle Park, NC, Oct. 1989, pp. 248-253.

[20] B.R. Rau, M.S. Schlansker and D.W.L. Yen, “The Cydra 5 Stride-Insensitive Memory System,” *In Proc. Int Conf. on Parallel Processing*, 1989, pp. 242-246.