Designing Packet Buffers for Router Linecards

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@stanford.edu, kompella@cs.stanford.edu, nickm@stanford.edu

Abstract -- ***All routers contain buffers to hold packets during times of congestion. When designing a high-capacity router (or linecard) it is challenging to design buffers because of the buffer's speed and size, both of which grow linearly with line-rate, R. With today's DRAM technology, it is barely possible to design buffers for a 40Gb/s linecard in which packets are written to (read from) memory at the rate at which they arrive (depart). Over time, the problem will get harder: Link rates will increase, line cards will connect to more lines, and buffers will get larger. Ideally, we would like a memory with the density of DRAM, and the speed of SRAM. And so some commercial routers sometimes use hybrid packet buffers built from a combination of small fast SRAM and large slow DRAM. The SRAM holds ("caches") the heads and tails of packet FIFOs, allowing arriving packets to be written quickly to the tail, and departing packets to be read quickly from the head. The large DRAMs are used for bulk storage, to hold the majority of packets in each FIFO that are neither at the head nor the tail. Because of the relatively long time to write to (or read from) the DRAMs, data is transferred between SRAM and DRAM in large fixed-size blocks, consisting of perhaps many packets at a time. A memory manager shuttles packets between the SRAM cache and the DRAM with two goals: (1) Arriving packets are written to DRAM before the SRAM overflows, and (2) Departing packets are guaranteed to be in the SRAM when it's their turn to leave. In this paper we find optimal memory managers that achieve both goals, while minimizing the size of the SRAM cache. When the delay through the buffer is minimized, the size of the SRAM cache is proportional to $Q \ln Q$, where Q is the number of FIFOs that the buffer maintains. There is a tradeoff between the size of the SRAM and the minimum pipeline delay through the packet buffer. When a pipeline delay can be tolerated, we find memory managers that reduce the required SRAM cache size so as to be proportional to Q.***

I. Introduction

A. Background

Internet routers need buffers to hold packets during times of congestion. Packet switching relies for its efficiency on statistical multiplexing in which multiple packets can arrive simultaneously destined for the same outgoing line. Because no more than one packet can depart on each line at a time, the remaining packets are held in a packet buffer until it's their turn to leave.

On the face of it, packet buffers are simple devices to build. Arriving packets are written to a packet memory as they arrive; and then when it's time to leave, the packets are read from memory and sent to the outgoing line. But two characteristics make it difficult to build packet buffers for high performance routers: the buffer's *speed* and its *size*.

The *speed* of a packet buffer depends on how fast packets can arrive and depart. If there is one packet buffer per linecard, then the buffer must be able to store (retrieve) packets as fast as they arrive (depart), respectively. With line-rate R, the memory bandwidth must be at least $2R$; because memory devices typically share a single data bus, packets arriving at rate R must compete for the data bus with packets departing at rate R. If N linecards share a single packet buffer, then the buffer must be able to store (and retrieve) N packets simultaneously, and needs a memory bandwidth of at least $2NR$.

The *size* of a packet buffer depends on many factors, such as the traffic characteristics, the desired performance (loss rate and throughput), and the congestion control mechanism. For example, if we know the arrival process, we can size the buffer so it never drops a packet. But the Internet is dominated by unpredictable, elastic, best-effort, TCP traffic [1] with a closed-loop congestion control algorithm that greedily fills the packet buffers at the bottleneck link. And so one could argue that the buffers should be made as large as possible; a larger buffer will drop fewer packets, and always keep the bottleneck line busy. More pragmatically, there is a widely used rule-of-thumb that says that, for TCP to work well, the buffer should hold as many packets as could pass through the buffer in one end-to-end round trip time; i.e. the buffer size should be $R \times RTT$, where RTT is the average round trip time between active end hosts [2]. It is beyond the scope of this paper to consider whether this rule-of-thumb is correct, but it suffices to say that commercial routers today have buffers of approximately this size [3][4].

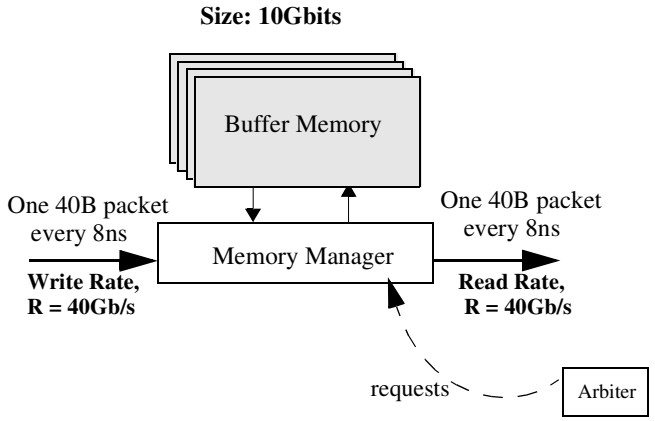


Figure 1: Packet buffers for a 40Gb/s router line card.

In summary, both the *speed* and *size* of a packet buffer grow linearly with the line-rate, R. Figure 1 illustrates the speed and size of a 40Gb/s (OC768c) router linecard.[1] Assuming a constant stream of 40-byte packets,[2] the packet buffer must write and read a packet every 8ns (i.e. one memory operation every 4ns); and assuming an average Internet RTT of 0.25s [6], the buffer must hold 10Gbits (1.25Gbytes).

B. Memory Technology

Most packet buffers are built from commercially available memory devices, such as SRAM (static RAM) or DRAM (dynamic RAM). SRAMs are relatively fast but small, and power-hungry. On the other hand, DRAMs are relatively large but slow, and consume much less power. At the time of writing, the largest commercial SRAM holds 16Mbits, has a random access time of 4ns, and consumes 250mW/Mbit. The largest DRAM holds 1Gbit, has a random access time of 40ns and consumes 4mW/Mbit.

For our 40Gb/s linecard example, size dictates that we use DRAM (with SRAM we would need over 600 devices, consuming 2.4kW!); but with packets arriving or departing every 4ns, a DRAM is not fast enough. Unfortunately, this shortfall in memory bandwidth is not going to be solved anytime soon, and illustrates a problem that will get worse, rather than better over time. DRAMs are optimized for size rather than speed, and their speed increases by only 10% every 18 months [7].[3] Line-rate, R, increases 100% every 18 months [8] and hence DRAM will fall further and further behind the needs of high speed packet buffers.

C. Using Parallelism

An obvious question to ask is: If the throughput required for a packet buffer is larger than the bandwidth of a single DRAM memory device, why don't we just use lots of DRAM devices in parallel? An example of this is shown in Figure 2. To some extent this is possible, but it is not as simple as it seems at first. In its simplest form, the technique only works if the buffer holds just one FIFO queue. In this case, we can use the arrangement shown in Figure 3(a). A memory manager assembles arriving packets into large blocks which, when full, are written to the same address (the tail of the FIFO queue) in multiple memory devices at the same time. Likewise, blocks of data are read one at a time from the head of the queue, and packets sent to the outgoing line. If the random access time for one memory is T, then the block size, b, must be at least $2RT$ bits. In our example with $R = 40$ Gb/s this gives $b = 4000$ bits, if $T = 50$ ns DRAMs are used.[4]

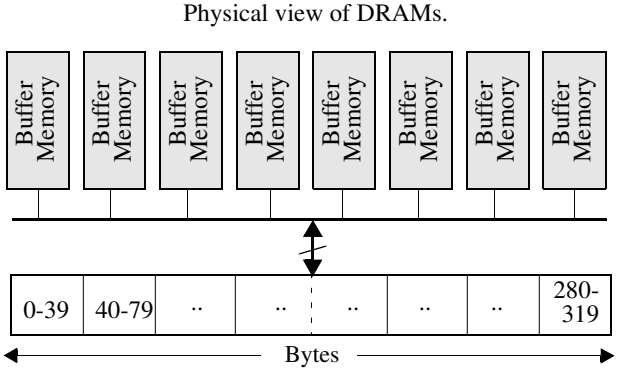


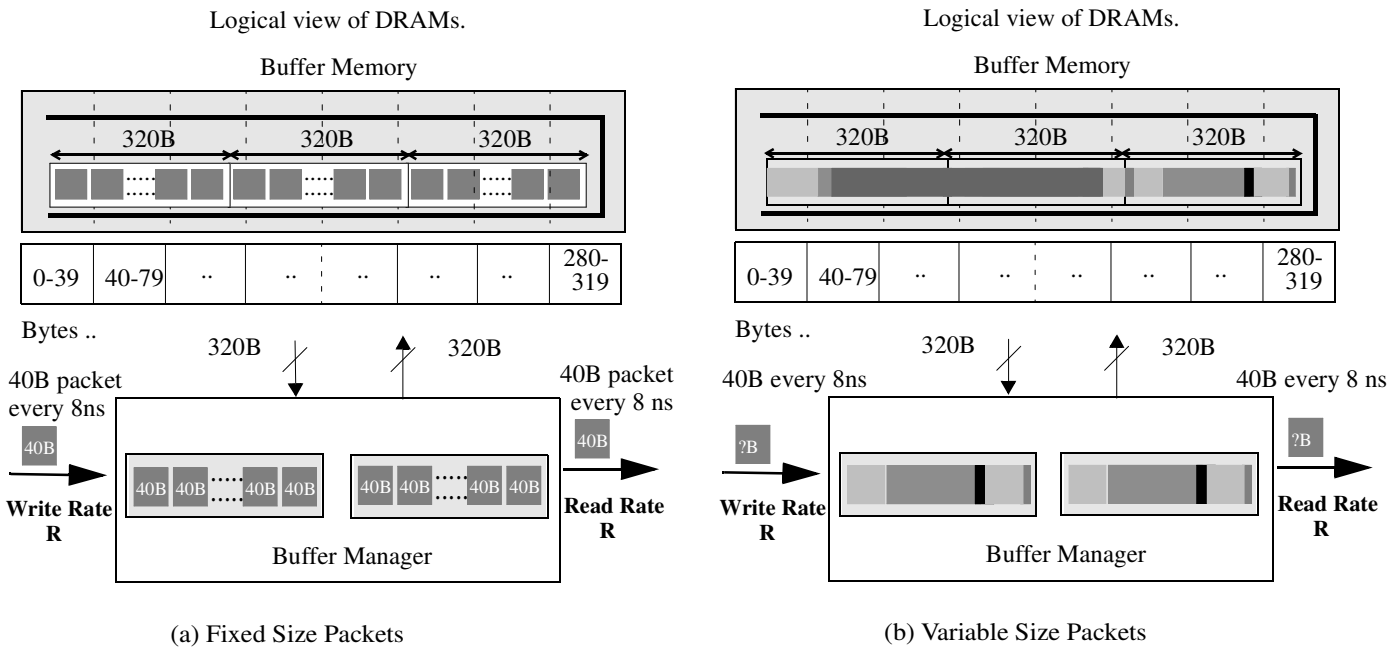
Figure 2: Using memories in parallel to increase throughput. A block size $b = 320$ bytes is written in chunks of 40 bytes over 8 parallel memories.

Figure 3(a) shows the case when all arriving packets are 40bytes long, but the same technique can be used when the arriving packets are of variable length, as shown in Figure 3(b). If a block fills in the middle of an arriving packet, the block is written to memory, and the rest of the packet becomes the start of the next block. This works because blocks are read in FIFO order and full packets can be reassembled by the memory manager and sent to the outgoing line.

Unfortunately, packet buffers almost always hold multiple FIFOs, and so this technique is not much in use. At the very least, a packet buffer will hold one FIFO for each output (e.g. virtual output queues, VOQs, in an input queued router; or per-output queues in a shared memory router). If the router performs per-flow or per-class queueing, the number of FIFO queues can be enormous.[5] The problem is that successive arriving packets might belong to different queues and depart in a different order than they arrived. For example, consider a packet buffer with two FIFO queues: high priority and low priority. If a low priority packet arrives followed by a high priority packet, the high priority packet will probably depart first. They can't be written to the tail of the same FIFO as part of the same block; for if they were, they would be read from memory at the same time, even though it's not time for the low-priority packet to depart.

---

1. At the time of writing, 2.5Gb/s (OC48c) linecards are widespread, 10Gb/s (OC192c and 10GE) are being deployed, and 40Gb/s (OC768c) linecards are in development.
2. 50% [5] of packets that pass through a router are minimum length IP packets that, most often, contain a TCP acknowledgement.
3. Newer memories, such as ESDRAM [9] and RDRAM [10] have on-chip caches and novel high speed I/O pins. But they are still based on a regular DRAM core with a random access time of approximately 40ns.
4. The databus is not necessarily b bits wide; it simply means that b bits must be delivered to/from the memory every T seconds. In SDRAM and RDRAM devices, the I/O pins run much faster than the core memory, allowing the databus to be quite narrow. For example, an RDRAM device might carry a block of 1024 bits every 80ns over a 16-bit databus operating at 12.8Gb/s. In our example, sufficient memory bandwidth could be achieved with just four memory devices.
5. In routers 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.



(a) Fixed Size Packets

(b) Variable Size Packets

Figure 3: Packet buffer architectures when there is only one FIFO queue.

D. Arbitrary and unknown departure order

The sequence in which packets are read from a packet buffer is determined by a scheduling algorithm. For example, in a shared memory router, the time at which each packet is read from memory is determined by an output-link scheduler (e.g. a WFQ scheduler [11][12], a FCFS scheduler, or a strict priority scheduler). Similarly, in a router with input buffers on each line card, the time at which each packet is read from memory is determined by an arbiter that determines the configuration of a switching fabric. In general, we can think of requests being made by an external arbiter for a particular packet to be retrieved from the buffer. The sequence of requests is not known by 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.

The packet buffer is usually part of the router's processing pipeline. When the packet buffer receives a request for a packet, it has some fixed time in which to retrieve the packet from memory. A generic model of a packet buffer with an external arbiter is shown in Figure 1.

E. Problem Statement

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

In other words, we would like a packet buffer as fast as SRAM and as large as DRAM, in which all requests from an unpredictable arbiter are fulfilled within a fixed time. For example, the packet buffer in our 40Gb/s linecard may receive a new request every 4ns from an external arbiter (e.g. from a crossbar arbiter if it's an ingress buffer, or from an output scheduler if it's an egress buffer). To fulfill the stream of requests, the packet buffer is required to deliver a new packet every 4ns. In this paper we will be considering two cases: In one case, the packet buffer is required to deliver a new packet immediately when it receives the request. In the second and more relaxed case, the packet buffer is allowed to respond within a fixed bounded time; for example, within 100ns of receiving the request. Of course, it must still process a new request every 4ns so as to maintain the throughput.

II. Hybrid SRAM-DRAM Packet Buffer

For the remainder of this paper, we will be using the memory hierarchy shown in Figure 4. Memory bandwidth is increased by reading (writing) up to b bytes from (to) DRAM memory in parallel, respectively. When packets arrive to the line card, they are stored temporarily in an SRAM, waiting for their turn to be written into DRAM. At the appropriate time (determined by a memory management algorithm), a block of bytes is written into the DRAMs at the same time. We can think of the 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. A memory management algorithm

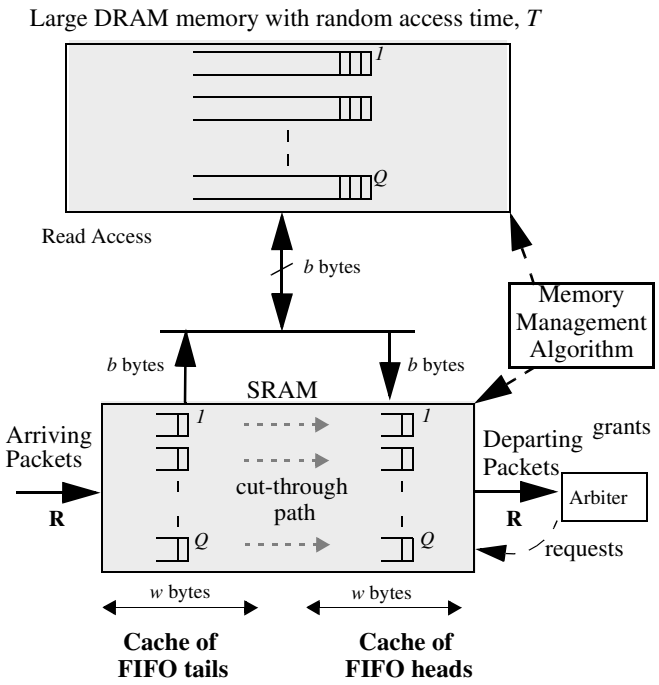


Figure 4: Memory hierarchy of packet buffer, showing large DRAM memory with heads and tails of each FIFO maintained in a smaller SRAM cache.

schedules writes and reads to the DRAM, manages row and bank conflicts, and inter-leaving of memory accesses. The SRAM behaves like a cache, holding packets temporarily when they first arrive and just prior to their departure. If a packet is requested by the arbiter but is not present in the SRAM, there can be a pipeline delay — which is defined as the time elapsed from when the arbiter issues the request, until the packet is delivered.[6]

The scheme is not new, and is already deployed in proprietary systems. But there is no literature which describes or analyzes this technique, and we believe that existing designs are based on ad-hoc statistical assumptions without hard guarantees. In these systems, the memory hierarchy only gives statistical guarantees for the time to access a packet, similar to interleaving or pre-fetching used in computer systems [13][14][15][16][17]. Examples of implementations that use commercially available DRAM controllers are [18][19]. While statistical guarantees might be acceptable for a computer system (in which we are used to cache misses, TLB misses, and memory refresh), it is not generally acceptable in a router where pipelines are deep and throughput is paramount. There are two key performance goals, which are associated with such a memory hierarchy.

Our goal is to design a hybrid SRAM-DRAM packet buffer which guarantees that a requested packet will always be available in the SRAM cache, regardless of the sequence of requests. As we will see, this requires a memory management algorithm that will decide when to replenish the SRAM FIFOs so as to ensure the packet is available when needed. We will see that the size of the SRAM grows with the number of queues and the line-rate, and so we find an algorithm that minimizes the size of the SRAM. Then, in an modified version of the problem, we will allow a fixed, non-zero pipeline delay between when the request is made and the packet is delivered. We find that, up to a point, the longer the pipeline delay, the smaller the SRAM can be. Again, we find optimal memory management algorithms. Finally, we find optimal algorithms that allow a designer to tradeoff SRAM size with pipeline delay.

A. Approach

Our approach is to 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 meet a specific pipeline delay. A good design will require a smaller SRAM that, ideally, can be placed on-chip. As we will see, the size of the SRAM depends on the pipeline delay. Not surprisingly, as the pipeline delay is reduced, we need a larger SRAM.

The rest of this paper is organized as follows. In Section III, we will first describe a memory management algorithm (MMA), which gives zero pipeline delay (i.e. a requested packet is guaranteed to be in SRAM) using an SRAM of size $O(Qb\ln Q)$, where Q is the number of queues to be maintained in the packet buffer, and b is the size of the block written to DRAM on each transaction. We show that the MMA is optimal. In Section IV, we relax the constraints and describe an algorithm that gives a bounded, non-zero, pipeline delay. At one extreme, the algorithm requires an SRAM of size $O(Qb)$, with $O(Qb)$ pipeline delay. We find an expression for the tradeoff between the pipeline delay and the size of the SRAM. In Section V, we show how the SRAM can be made smaller in return for a slightly more complicated SRAM memory manager. We find a lower bound on the size of the SRAM and a memory management algorithm that achieves it. Finally, we consider implementation of the algorithms in Section VI.

As an example of how these results may be used, consider an input-queued router with 32 ports, each operating at 40Gb/s (OC768c). Each input maintains 32 virtual output queues. We will assume that DRAMs have a random access time of $T = 51.2ns$ and that up to 64 bytes[7] can be written to each DRAM once every $51.2ns$. (A 16bit wide bus, running at

[6] It is convenient to assume that the pipeline delay to read a byte already present in the SRAM is zero.

[7] It is common in practice to write data in sizes of 64 bytes internally as this is the first power of 2 above the sizes of ATM cells and minimum length TCP segments (40 bytes).

400Mhz DDR on each interface is sufficient to achieve this). Our results indicate that such a packet buffer can be built using: (1) 1.6Mbits of SRAM with zero pipeline delay (i.e. packets are always in SRAM when requested), (2) 460kbits of SRAM (dynamically allocated) with a pipeline delay of $2.9\mu s$, or (3) 920kbits of SRAM (statically allocated) with a pipeline delay of $2.9\mu s$. Alternatively, the designer could bound the pipeline delay to any value between zero and $2.9\mu s$, and choose a corresponding SRAM size.

III. A Memory Management Algorithm which Guarantees Zero Pipeline Delay

A. Buffer Operation

Referring to Figure 4, the head and tail of each FIFO resides in SRAM, while the middle portion of the FIFO resides in DRAM. As packets arrive to the packet buffer they are written to the tail of a FIFO in the SRAM cache, where they wait for the MMA to transfer them to the corresponding FIFO in DRAM. The MMA always transfers a block of b bytes at a time — never smaller — from an SRAM FIFO to the corresponding DRAM FIFO.[8] Similarly, the MMA always transfers packets, as needed, from DRAM to SRAM in blocks of size b bytes.

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 packet 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 duration $2T$, where each cycle consists of one write and one read. An obvious requirement for the packet buffer to sustain the line rate R is that $b \geq 2RT$. In this paper, we take it as a requirement to minimize the total memory bandwidth (and hence the power and board space), and so we assume $b = 2RT$.

Transfers between SRAM and DRAM are under the control of the MMA. The MMA decides when to transfer bytes between SRAM and DRAM based on the occupancy of the SRAM FIFOs. It transfers blocks of data from SRAM to DRAM so as to prevent the SRAM ingress FIFOs from overflowing. Similarly, it transfers blocks of data from DRAM to SRAM so as to prevent the SRAM egress FIFOs from underrunning, and therefore being unable to service new requests from the external arbiter.

[8] Blocks smaller than b bytes are never transferred to DRAM — if a packet is requested by the arbiter before it is written to DRAM, it is simply taken directly from the SRAM FIFO without passing through the DRAM.

We will assume that the arbiter requests one byte at a time. In practice, of course, the arbiter would request a whole packet at a time. But it makes the theorems and explanations simpler if we assume here that each request is for one byte. Exactly the same algorithms and theorems apply if the arbiter requests one packet at a time, rather than just one byte.

B. 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 bytes arrived at the tail SRAM, they would be written into DRAM. This would require $b-1$ bytes of storage in the tail SRAM buffer so as to store the bytes which arrived before the b^{th} byte. Similarly, the head SRAM buffers would require $b-1$ bytes of storage. When the head SRAM buffer receives a request from the arbiter, the MMA reads b bytes from DRAM, grants one byte to the arbiter and stores the remaining $b-1$ bytes in SRAM. An extra b bytes are needed in case the arbiter stops requesting when there are $b-1$ bytes in SRAM. The SRAM will still be replenished with b new bytes (we don't know if and when the arbiter will need them), and so we need a total of $2b-1$ bytes of storage. Notice that every time b new requests arrive from the arbiter, b new bytes are read from DRAM, and so the packet buffer can always issue packets at the rate it receives requests.

Things get more complicated when there are more FIFOs in the system. For example, let's see how a FIFO in the head SRAM buffer can under-run (i.e. the arbiter makes a request that the head SRAM buffer can't fulfill) even though the FIFO still has bytes in DRAM. We'll assume that the packet buffer maintains $Q > 1$ FIFOs. Each time a FIFO is read, bytes corresponding to the read request depart from the head SRAM buffer, and it may trigger the need to replenish the FIFO so as to prevent under-run of the FIFO in the future. If consecutively departing bytes cause different FIFOs to need replenishment, then a queue of read requests will form waiting for bytes to be retrieved from DRAM. The queue builds because in the time it takes to replenish one FIFO (with b bytes), b new requests can arrive. It is easy to imagine a case in which a replenishment is needed, but every other FIFO is also waiting to be replenished, and so there might be $Q-1$ requests ahead in the queue.

If there are too many queued requests for a single FIFO, it is possible that the FIFO in the head SRAM buffer is not replenished before it under-runs. Put another way, the head SRAM buffer might have to contain a very large number of bytes for each FIFO so as to hold sufficient reserves to cover the worst case sequence of departures.

In our discussion above, we focussed on the head SRAM buffer. It is interesting to note that the tail and head SRAM buffers are essentially symmetrical in the sense that the tail (head) buffer has an unpredictable arrival (departure) process, and predictable departure (arrival) process, respectively.[9] It

turns out that, for a given MMA, the tail and head buffers are the same size. So for brevity, we will only analyze the head buffer — the results extend simply to the tail buffer. In this section, we shall assume that the SRAM is statically partitioned, and a separate area of memory (of size w bytes) is reserved for each FIFO queue. In what follows we will show a lower bound on the size of the head SRAM for any MMA which guarantees zero pipeline delay.

Definition 1: ***Occupancy Counter*** $X(i,t)$ *: The number of bytes present in the head SRAM for FIFO* i*.*
When a request arrives and a byte is delivered, $X(i,t)$ *is decremented by one. When* b *bytes are read from DRAM and placed into the SRAM,* $X(i,t)$ *is incremented by* b*.*

Definition 2: ***Deficit*** $D(i,t)$*: The number of unsatisfied read requests for FIFO* i*.*
Arbiter requests for FIFO i *for which no byte has been read from the DRAM (even though there are outstanding cells in the DRAM), are called unsatisfied read requests. If a FIFO* i *has cells in the DRAM* $D(i,t) = w - X(i,t)$ *else* $D(i,t) = 0$*.*

Theorem 1: (Necessity) To guarantee that a byte is always available in SRAM when requested (regardless of the MMA), the SRAM must contain at least $Qw > Q(b-1)(2+\ln Q)$ *bytes.*

Proof: Consider the particular traffic pattern with the following pattern of requests. The pattern progresses in a number of iterations, where each iteration consists of several time slots. Each successive iteration lasts fewer time slots than the previous one, as follows:

1st iteration (Q ***time slots):*** In time slots $t = 1, 2, 3, \ldots Q$, a request arrives for FIFO t. It takes b timeslots to read b bytes from the DRAM and replenish the SRAM cache. At the end of time slot Q, at most Q/b FIFOs will have received b bytes from the DRAM, and so at least $Q(1-1/b)$ FIFOs will have $D(i,Q) = 1$.

2nd iteration ($Q(1-1/b)$ ***time slots):*** In the 2^{nd} iteration, consider the $Q(1-1/b)$ FIFOs for which $D(i,Q) = 1$. In the next $Q(1-1/b)$ time slots, we'll assume that a request arrives for each of these FIFOs. At the end of the 2^{nd} iteration, $Q(1-1/b)/b$ of these FIFOs will be replenished, and $Q(1-1/b)^2$ will have $D(i,t) = 2$.

xth iteration ($Q(1-1/b)^x$ ***time slots):*** By continuing this argument, we can see that at the end of x iterations there will be $Q(1-1/b)^x$ FIFOs with $D(i,t) = x$. Solving for $Q(1-1/b)^x = 1$, we get that

$$x = \log_{(b/(b-1))} Q = (\ln Q)/(\ln c), \quad (1)$$

where $c = b/(b-1)$. Since, $\ln(1+x) < x$, we get

$$\ln c = \ln\left[1 + \frac{1}{(b-1)}\right] < \frac{1}{(b-1)}, \quad (2)$$

and it follows that $x > (b-1)\ln Q$.

So far, we've proved that if each FIFO can hold $(b-1)\ln Q$ bytes, then in $(b-1)\ln Q$ iterations one FIFO can have a deficit of $(b-1)\ln Q$ bytes. Imagine that this FIFO is left with an occupancy of $b-1$ bytes (i.e. it initially held $(b-1)(1+\ln Q)$ bytes, and 1 byte was read in each of $(b-1)\ln Q$ iterations). If in successive times slots we proceed to read b more bytes from the most depleted FIFO, it will certainly under-run (because it has never been replenished). Since we do not know *a priori* the queue for which this traffic pattern may occur, we require that $w > (b-1)(1+\ln Q)$ bytes to prevent under-run. But we're not quite done. Imagine that we initially fill every FIFO with precisely $w = (b-1)(1+\ln Q)$ bytes, and assume that the arbiter reads one byte from one FIFO then stops indefinitely. The FIFO now contains $(b-1)(1+\ln Q)-1$ bytes, but is replenished b time slots later with b bytes from DRAM. Therefore, the SRAM needs to be able to hold $w > (b-1)(2+\ln Q)$ bytes per FIFO, and so $Qw > Q(b-1)(2+\ln Q)$ bytes overall. ❐

C. The Most Deficit Queue First MMA (MDQF-MMA)

Motivated by the proof above, we'll now consider a memory management algorithm (called MDQF-MMA) that guarantees a byte will available in SRAM when requested.

Algorithm: MDQF-MMA attempts to refresh a queue if at least b time slots have passed since the last refresh and chooses the queue that has the maximum deficit amongst all queues which satisfy the following:

- **Condition 1:** There exists b bytes for that queue either in the tail SRAM or the DRAM, and
- **Condition 2:** There is sufficient space in the head SRAM to accept b bytes for that queue.

If multiple queues have the same maximum deficit, MDQF-MMA chooses one at random.

1) Sufficiency

We will now prove a bound on the size of the SRAM buffer needed by the MDQF-MMA.

Definition 3: ***Total Deficit*** $f(i,t)$*: At time t,* $f(i,t)$*, denotes the sum of the deficits of the* i *queues with the most deficit in the head SRAM. In other words, suppose* $v = (v_1, v_2, \ldots v_Q)$*, are the values of the deficits* $D(i,t)$*, for each of the* $i = \{1, 2, \ldots, Q\}$ *queues at any time* t*. Let* π *be an ordering*

[9] There is a slight asymmetry in the design on the head and tail SRAM which will become clear in Section III.

of the queues $(1, 2, 3, \ldots Q)$ *such that they are in descending order i.e.* $v_{\pi(1)} \geq v_{\pi(2)} \geq v_{\pi(3)} \geq \ldots \geq v_{\pi(Q)}$ *. Then,*

$$f(i,t) \equiv \sum_{k=1}^{i} v_{\pi(k)} . \quad (3)$$

Definition 4: ***Maximum Total Deficit*** $F(i) : F(i)$*, denotes the maximum value of* $F(i,t)$ *as seen by MDQF-MMA over all timeslots and over all request patterns. Note that MDQF-MMA samples the deficits at most once every* b *time slots in order to choose the queue with the maximum deficit. Thus, if* $\tau = (t_1, t_2, \ldots)$ *denotes the sequence of times at which MDQF-MMA refreshes the SRAM, then*

$$F(i) \equiv Max\{\forall t \in \tau, F(i,t)\} . \quad (4)$$

D. Upper Bounds for the MDQF-MMA

Lemma 1: Under the MDQF-MMA, which services requests without any pipeline delay, $F(1) < b[2 + \ln Q]$**.**

Proof: Assume that t is the first time slot at which $F(1)$ reaches its maximum value under the MDQF-MMA. Suppose that this maximum was attained for some queue i. Hence, $D(i,t) = F(1)$. Trivially, we have $D(i, t-b) \geq F(1) - b$. Also, since queue i reaches it maximum deficit at time t, it could not have been served by MDQF-MMA at time $t-b$, because if so, then either, $D(i,t) < F(1)$, or it is not the first time at which it reached a value of $F(1)$, both of which are contradictions to the above assumption. Hence there was some other queue j which was served at time $t-b$. By the property of MDQF-MMA, it had a larger deficit than queue i at time $t-b$. Formally,

$$D(j, t-b) \geq D(i, t-b) \geq F(1) - b . \quad (5)$$

Hence, we have:

$$F(2) \geq F(2, t-b) = D(i, t-b) + D(j, t-b) . \quad (6)$$

This gives,

$$F(2) \geq F(1) - b + F(1) - b . \quad (7)$$

Now, consider the first time slot t when $F(2)$ reaches its maximum value. Assume that at time t, some two queues, m and n had the most and second most deficit amongst all queues. As argued before, neither of the two queues could have been serviced at time $t-b$. Hence, there is some other queue p, which was serviced at time $t-b$ which had the most deficit at time $t-b$. We know that $D(p, t-b) \geq D(m, t-b)$ and $D(p, t-b) \geq D(n, t-b)$. Hence,

$$D(p, t-b) \geq \frac{D(m, t-b) + D(n, t-b)}{2} \geq \frac{F(2) - b}{2} . \quad (8)$$

This gives,

$$F(3) \geq F(3, t-b) . \quad (9)$$

Substituting the deficits of the three queues m, n and p we get,

$$F(3) \geq D(m, t-b) + D(n, t-b) + D(p, t-b) . \quad (10)$$

Hence,

$$F(3) \geq F(2) - b + \frac{F(2) - b}{2} . \quad (11)$$

Proceeding similarly, we get, $\forall i \in \{1, 2, \ldots Q-1\}$

$$F(i+1) \geq F(i) - b + \frac{F(i) - b}{i} . \quad (12)$$

We also have that

$$F(Q) < Qb . \quad (13)$$

Solving these recurrence relations gives us,

$$F(1) \leq b\left[1 + \sum_{i=1}^{Q-1} \frac{1}{i}\right] . \quad (14)$$

Also since,

$$\forall N, \sum_{i=1}^{N-1} \frac{1}{i} < \sum_{i=1}^{N} \frac{1}{i} < 1 + \ln N , \quad (15)$$

we get,

$$F(1) < b[2 + \ln Q] . \square \quad (16)$$

Lemma 2: Under the MDQF-MMA, which services requests without any pipeline delay,

$$F(i) < bi[2 + \ln(Q/i)], \forall i \in \{1, 2, \ldots, Q-1\} . \quad (17)$$

Proof: See Appendix A.[10] $\square$

E. The Size of the SRAM

Theorem 2: (Sufficiency) For MDQF-MMA to guarantee that a requested byte is in SRAM (and therefore available immediately) it is sufficient for the head SRAM to hold $Qw = Qb(3 + \ln Q)$ *bytes, and the tail SRAM to hold* $Qw = Qb(2 + \ln Q)$ *bytes.*

Proof: From Lemma 1, we would have to allocate $F(1) \leq b[2 + \ln Q]$ bytes for each of the queues in the head SRAM, since we cannot predict *a priori* which queue may under-run.

However even though the maximum deficit of a queue as seen by MDQF-MMA is no more than $F(1)$ (which is reached at some time t), the queue can lose up to $b-1$ more cells in the next $b-1$ time slots, before it gets refreshed at time $t+b$. Hence, the size of the head SRAM that is sufficient so that no queue will underflow with MDQF-MMA, is $w = b[2 + \ln Q] + (b-1) < b[3 + \ln Q]$ bytes.

---

[10] Note that this is a weak inequality and in general the inequality in Equation (28) (in Appendix A) should be used. However this loose bound will be used later on to study the rate of decrease of the function $F(i)$ and hence the decrease in the size of the SRAM.

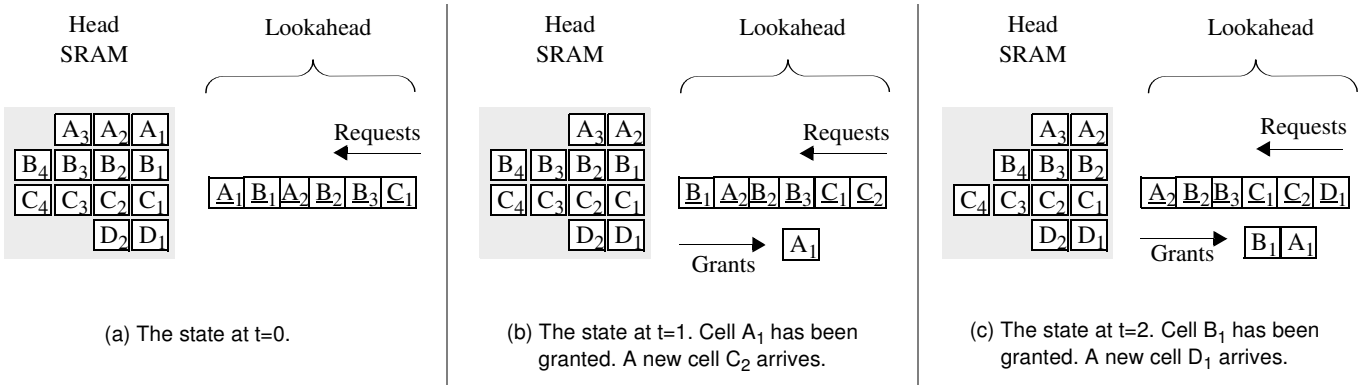


Figure 5: The packet buffer hierarchy with a lookahead buffer. The example is for $Q = 4$ and $b = 3$ bytes. Requests are shown underlined. The lookahead buffer has size 6 bytes. Each request is granted after a pipeline delay of 6 time slots. The MDQFP-MMA can choose any of queues 'A' or 'B' for refresh from DRAM at time t=0, since both these queues will have the maximum deficit when the lookahead is taken into consideration.

For the tail SRAM, $F(1)$ (which is reached at some time t') represents the maximum number of cells that a FIFO i can accumulate. When b bytes from a FIFO i are chosen to be written into the DRAM at time t', the space occupied by those b bytes is instantaneously freed from the tail SRAM. Hence a tail SRAM of size $Qw = Qb(2 + \ln Q)$ is sufficient. ❐

F. Optimality

While we do not prove rigorously that MDQF-MMA is optimal, we observe that the sufficiency bound for MDQF-MMA obtained in Theorem 2 is close to the necessary bound for any MMA obtained in Theorem 1.

IV. An MMA which Tolerates Bounded Pipeline Delay

In the previous section, we saw that in order to guarantee that a packet is immediately available when requested, we need an SRAM buffer of size $O(Q \ln Q)$ bytes. However most router designs naturally tolerate a fixed, albeit bounded pipeline delay. 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 to avoid under-run. 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 as shown in Figure 5. The MMA uses the lookahead buffer to delay new requests while it takes a peek at which FIFOs are receiving requests. This allows the MMA to make sure that by the time the new request has finished being buffered, the byte that is being requested has been fetched from DRAM into SRAM. In this manner, the lookahead buffer increases the time from when a request is issued until bytes are delivered. If the pipeline delay is small and bounded, it will be acceptable in most applications. As an example, Figure 5(a,b,c) show how the lookahead buffer advances every time slot. The first request in the lookahead buffer at time slot $t = 0$ (request $\underline{A}_1$ in Figure 5a) is granted at time slot $t = 1$ (as shown in Figure 5b). Also a new request arrives to the tail of the lookahead buffer every time slot (request $\underline{C}_2$ in Figure 5b).

A. Definitions

In what follows, we will use the following definitions.

Definition 5: ***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.*

Lookahead and pipeline delay are related in the following way. If the packet buffer can tolerate a pipeline delay of x timeslots, then it is equivalent to saying that the MMA has a lookahead of x requests. As an example notice that in Figure 5(b), request $\underline{C}_2$ which arrives at time slot, $t = 1$, will be granted after another 6 time slots after the request makes it way to the head of the lookahead buffer of size 6.

Definition 6: ***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 bytes in the SRAM.*

A critical queue will under-run unless the MMA replenishes the SRAM quickly enough.

Definition 7: ***Earliest Critical Queue:*** *The earliest critical queue is the queue (from among those that are currently critical) that turned critical the earliest.*

Definition 8: ***Maximum Total Deficit based on pipeline delay*** *$F_x(i)$: $F_x(i)$ denotes the maximum value of $F(i, t)$ as seen by MDQFP-MMA which operates with a pipeline delay of x, over all timeslots and over all request patterns. Note that in the previous section when there was no pipeline delay we dropped the subscript i.e. $F_0(i) \equiv F(i)$.*

B. The Most Deficit Queue First with Pipeline Delay (MDQFP-MMA).

Algorithm Description: Every b time slots, MDQFP-MMA considers replenishing FIFOs that meet Conditions 1 and 2. If there are critical queues it chooses to replenish the earliest critical queue. Otherwise, it replenishes the queue that — based on current information — appears most likely to become critical in the future. This is the queue that — with the current set of requests in the lookahead buffer — will have the largest deficit at time $t+x$, assuming that no queues are replenished between now and $t+x$. If multiple queues will have the same deficit at time $t+x$, MDQFP-MMA chooses one at random.

Lemma 3: (Sufficiency) Under the MDQFP-MMA policy, and a pipeline delay of $x > 0$ time slots,

$$F_x(1) \le \left\lceil \underset{(\forall i)}{MAX} \left\{ \frac{F(i)}{i} - \left\lfloor \frac{x}{b} \right\rfloor \frac{b}{i} + b - 1 \right\} \right\rceil, \; i = 1, 2, \ldots Q \,. \quad (18)$$

Proof: We shall derive a bound on the deficit of a queue in the MDQFP-MMA system by comparing it with the MDQF-MMA system. Let $v = (v_1, v_2, \ldots v_Q)$, denote the values of $D(j,t)$ for each of the $j = \{1, 2, \ldots, Q\}$ queues at any time t, for the MDQF-MMA system respectively. Let π be an ordering of the queues $(1, 2, 3, \ldots Q)$ such that they are in descending order i.e. for v we have, $v_{\pi(1)} \ge v_{\pi(2)} \ge v_{\pi(3)} \ge \ldots \ge v_{\pi(Q)}$. Note that given an instance of the MDQF-MMA system, MDQFP-MMA has a further x timeslots in which it can decrease the deficits of a number of queues, by making an additional $\lfloor x/b \rfloor$ DRAM refreshes. By definition, MDQFP-MMA refreshes the queue with the most deficit multiple times, and tries to make it "equal" to the deficit of the 2^{nd} queue with the most deficit. However since the MDQFP-MMA has to perform operations at multiples of b bytes, it can only guarantee that the two queues are within $b-1$ bytes of each other. Then, the two queues with the most deficit would be refreshed until they "equal" (i.e. are within $b-1$ bytes of each other) the deficit of the 3^{rd} queue with the most deficit. In general, given any distribution of the deficits $v = (v_1, v_2, \ldots v_Q)$ of the queues, MDQFP-MMA will be able to decrease the deficits of some i queues ($1 \le i \le Q$) which have the most deficits at time t, such that these i queues have a deficit not more than $b-1$ bytes within some value $P(i,x)$ at time $t+x$. Hence, we have that

$$i(P(i,x) - (b-1)) \le F(i) - \left\lfloor \frac{x}{b} \right\rfloor b \,. \quad (19)$$

Thus,

$$P(i,x) \le \frac{F(i)}{i} - \left\lfloor \frac{x}{b} \right\rfloor \frac{b}{i} + b - 1 \,. \quad (20)$$

In general, we do not know the value of i, since it depends on the distribution v of the queues. However we can write for the MDQFP-MMA policy,

$$D(i, t+x) \le MAX(\forall i) P(i,x) \,. \quad (21)$$

Since $F_x(1)$ is the maximum deficit of a single queue under the MDQFP-MMA system,

$$F_x(1) \le \left\lceil \underset{(\forall i)}{MAX} \left\{ \frac{F(i)}{i} - \left\lfloor \frac{x}{b} \right\rfloor \frac{b}{i} + b - 1 \right\} \right\rceil, \; i = 1, 2, \ldots Q \,. \quad (22)$$

❒

Theorem 3: (Sufficiency) With MDQFP-MMA and a pipeline delay of x time slots, it is sufficient to use a head SRAM of size $Qw = Q(F_x(1) + b)$ bytes and a tail SRAM of size $Qw = QF_x(1)$ bytes.

Proof: The proof is similar to Theorem 2. ❒

Corollary 1: With MDQFP-MMA and a pipeline delay of $x \to Qb^-$ time slots, the sizes of the head and tail SRAM are minimized and are of size $Qw = Q(2b)$ and $Qw = Qb$ respectively.

Proof: This follows by substituting $x = Qb$ and solving Equation (22). It is easy to see that the maximum is reached when $i = Q$ and then $F_x(1) = b-1$. When $x < Qb$ and $x \to Qb^-$, we will get $F_x(1) = b$. Now we can use Theorem 3 to obtain the bounds on the sizes of the head and tail SRAM ❒

C. The tradeoff between SRAM size and pipeline delay.

Intuition tells us that we can use a smaller SRAM if we can tolerate a larger pipeline delay. Let's try and determine the tradeoff between the two, and find an expression for the SRAM size as a function of x. To do this we need to eliminate the variable i from Equation (22). Unfortunately because of the floor and ceiling functions, the function $F_x(1)$ (which determines the size of the SRAM) is not differentiable. But we can find a good approximation by: (1) Replacing $F(i)$ in Equation (22) with the weak inequality in Equation (17), (2) Ignoring the floor and ceiling functions, and (3) Allowing x and i to take real values.

$F_x(1)$ is maximized when

$$\frac{\partial}{\partial i}\{b[2 + \ln(Q/i)] - ((x/i) + b - 1)\} = 0 \quad (23)$$

which happens when $i = x/b$. We know this is a maximum, because

$$\frac{\partial^2}{\partial i^2}\{b[2 + \ln(Q/i)] - (x/i)\} = \frac{-b}{i^2} < 0 \,, \quad (24)$$

corresponds to the maximum value of $F(1)$ when $i = x/b$. Substituting $i = x/b$ in Equation (22), we get,

$$F_x(1) \le b[1 + \ln(Qb/x)], \; x > 0. \quad (25)$$

Equation (25) tells us (approximately) the rate of decrease of $F(1)$ with an increase in x. We note that since

$$\frac{d}{dx}F_x(1) = -\frac{1}{x}, \quad (26)$$

the rate of decrease of $F_x(1)$ (and hence the size of the SRAM) is $\approx -1/x$. This indicates that even a small increase in the pipeline delay from zero will result in a large decrease in the size of the SRAM. This is illustrated by the example in Figure 6. The size of the tail SRAM as a function of the pipeline delay x for a system with $Q = 1000$ and $b = 10$ bytes is plotted. If no pipeline delay could be tolerated then the size

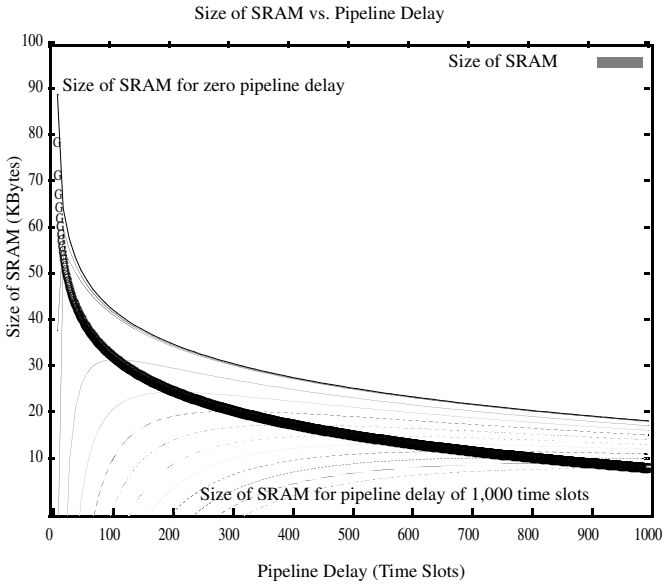


Figure 6: The SRAM size (in bold) as a function of pipeline delay (x). The example is for 1000 queues ($Q = 1000$), and a block size of $b = 10$ bytes. The dotted lines are plots of the function $QP(i, x)$, for $i \in \{1, 2, \ldots, Q\}$. The SRAM size for a given pipeline delay of x timeslots is plotted by joining the maximum values of $QP(i, x)$.

of the SRAM is 90Kbytes. If instead a pipeline delay of $Qb = 10000$ time slots was acceptable, then the size of the tail SRAM reduces to 10KBytes. Note that with a small pipeline delay of 300 time slots (this corresponds to 60ns of delay in a 40Gb/s line card) the SRAM need only be 20Kbytes. In other words, if we can tolerate a small pipeline delay, we can get most of the benefit of a small SRAM. [11]

[11] The "SRAM size vs. pipeline delay" curve is not plotted when the pipeline delay is between 1000 and 10,000 time slots since the curve is almost flat in this interval.

V. Further Decreasing the Size of the SRAM Buffer.

Until now we have assumed that each FIFO queue has a fixed space allocation in the SRAM; i.e. the SRAM is statically allocated, with space equally divided among the FIFOs. Although a static allocation is easier to maintain than a dynamic allocation (static allocation uses circular buffers, rather than linked lists), we can expect a dynamic allocation to be more efficient. This is because it's unlikely that all the SRAM FIFOs will fill up at the same time. A dynamic allocation can exploit this to devote all the SRAM to the occupied FIFOs, whereas a static allocation scheme will waste valuable SRAM space on FIFOs that are empty.

In what follows we quantify how much SRAM can be saved by using dynamic allocation instead.

The basic idea is that at any one instant, some FIFO queues are closer to becoming critical than others. The more critical queues need more buffer space, while the less critical queues need less. When we use a lookahead buffer, the MMA knows which queues are close to becoming critical, and which are not. The MMA can therefore dynamically allocate more space in the SRAM for the more critical queues, borrowing space from the less critical queues that don't need it.

A. A necessity bound on the size of the head SRAM buffer.

Theorem 4: (Necessity) The head SRAM buffer must be at least $Q(b-1)$ bytes for any MMA to service a sequence of requests within a bounded pipeline delay.

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

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

B. ECQF-MMA

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

Algorithm Description: Every b time slots, ECQF-MMA considers replenishing FIFOs that meet Conditions 1 and 2. If there is a critical queue, it replenishes the earliest one. Otherwise it does nothing.

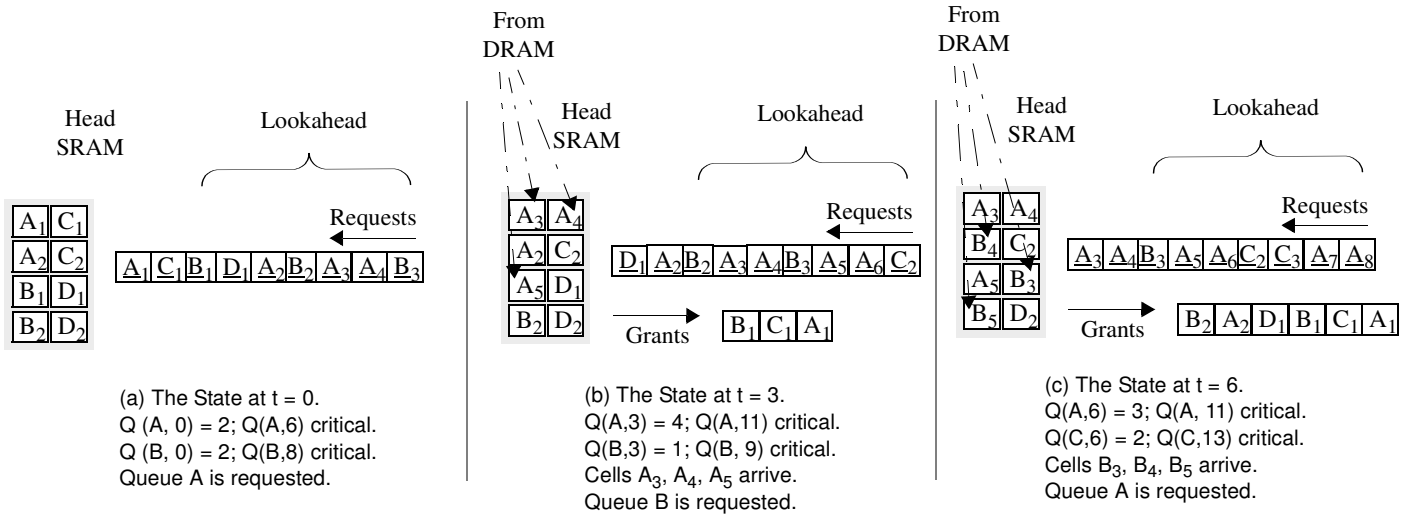


Figure 7: The ECQF-MMA algorithm with $Q = 4$ and $b = 3$ bytes**.** The size of the head SRAM is 8 bytes and the lookahead is $Q(b-1)+1 = 9$ bytes. The example shows a dynamically allocated head SRAM.

Example of ECQF-MMA:Figure 7 shows an example of ECQF-MMA with $Q = 4$ and $b = 3$**.** Figure 7a shows that the MMA (at time $t = 0$) determines 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. Bytes from queues A, C, B leave the head SRAM at times $t = 0, 1, 2$.

In Figure 7b the ECQF-MMA determines that B is the earliest critical queue and it is chosen for service from the DRAM. Bytes from queues D, A, B leave the head SRAM at times $t = 3, 4, 5$**.**

The occupancy of the head SRAM at time $t = 6$ is shown in Figure 7c. Queue A is again the earliest critical queue and is chosen for service from the DRAM.

1) Optimality

In what follows we show that ECQF-MMA needs the same amount of SRAM as required by the traffic pattern in the necessity condition of Theorem 5. Hence, ECQF-MMA is optimal.

2) Sufficiency

We now derive the size of the head 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 head SRAM is full and has $b-1$ bytes 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 bytes present in the DRAM for that queue.

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

C. 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 4: 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 head 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 b bytes arrive after every b time slots and exactly b bytes 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 bytes 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 5: (Sufficiency) A head SRAM buffer of size $Q(b-1)$ bytes and a lookahead of size $Q(b-1)+1$ bytes is sufficient for ECQF-MMA to service any sequence of arbiter requests with a pipeline delay of $Q(b-1)+1$ time slots.

Proof: The proof is in two parts. First we show that the head SRAM buffer does not overflow. Second we must show that

each packet is delivered within a bounded pipeline delay 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 V.B and from Lemma 4, the ECQF-MMA always reads exactly b bytes from the earliest critical queue every b time slots. Thus the total occupancy of the head SRAM buffer does not change. Since, this is true for all time slots the size of the head 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 byte is either present or not present in the SRAM. No pipeline delay is encountered by a byte of a packet, that is present in the SRAM. If the byte 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 q'. Then, this request for a byte could not have arrived earlier than $(q'+1)b$ time slots from the start. The DRAM would have taken no more than $q'b$ 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 byte is present in the head SRAM.

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

D. Relaxing the assumptions

We now show that Lemma 4 and Theorem 5 hold when our assumptions are relaxed. In order to do this we shall look upon the bytes in the previous section as "virtual" bytes as defined below.

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

Definition 8: ***Virtual Bytes:*** *The sum of both "real" bytes present in the DRAM and head SRAM and byte placeholders.*

E. Generalized Scenario

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

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

Assumption 3. (Request Every Time Slot) Till now we assumed that there was a request for a cell every time slot. It is easy to see that if there were time slots for which there were no requests then it only makes it easier for ECQF-MMA to meet its packet delay deadlines since it needs to service lesser packets as compared to a system which requested a packet every time slot.

VI. Implementation Considerations

A comparison of the sizes of SRAM required for both statically allocated and dynamically shared SRAM and for different values of pipeline delay as derived in the previous sections are shown in Table 1. We are now ready to discuss a few issues regarding the implementation complexity of the design presented in the paper.

1. **Complexity of the algorithm:** Both MDQF-MMA and ECQF-MMA require counters to maintain queue deficits. Similarly, MDQFP-MMA which is a combination of both MDQF-MMA and ECQF-MMA also requires counters. In addition, MDQF-MMA (and hence MDQFP-MMA) must identify the queue with the maximum deficit every b time slots. While this is possible to implement for a small number of queues using dedicated hardware or perhaps using a heap data structure [20], it may not scale when the number of queues is very large. In contrast, the ECQF-MMA is simpler to implement. It just needs to identify when a deficit counter becomes critical and replenish the corresponding queue.
2. **Complexity of implementing queues in the SRAM:** Statically allocated SRAM and dynamically shared DRAM are common place. Static allocation is usually implemented with a circular buffer, while dynamic allocation requires a linked list.

VII. Previous Work

Computer systems and network processing elements often build packet buffers from a number of DRAMs in parallel. We classify these techniques into the following categories:

1. ***Techniques which rely on DRAM row or bank properties:*** A simple technique to obtain high throughputs using DRAMs (using only random accesses) is by striping a

TABLE 1 : Tradeoffs between SRAM size and pipeline delay

SRAM Design	Pipeline delay (time slots)	Head SRAM (bytes)	Tail SRAM (bytes)	MMA
Static	0	$Qb(3+\ln Q)$	$Qb(2+\ln Q)$	MDQF
Static	Qb	$Q(2b)$	Qb	MDQFP
Dynamic	$Q(b-1)+1$	$Q(b-1)$	$Q(b-1)$	ECQF

packet[12] across multiple DRAMs [21]. In this approach each incoming packet is split into smaller segments and each segment is written into different DRAM banks; these banks reside in a number of parallel DRAMs. With this approach the random access time is still the bottleneck. To decrease the access rate to each DRAM, packet interleaving can be used [22][23]. In this technique, consecutive arriving packets are written into different DRAM banks. However since — at the time of writing the packets into the buffer — we don't know the order in which they will depart, it may happen that consecutive departing packets reside in the same DRAM row or bank, causing row or bank conflicts and momentary loss in throughput.

2. ***Techniques which provide statistical guarantees:*** Here the memory management algorithm (MMA) is designed so that the probability of DRAM row or bank conflicts is reduced. These include designs that randomly select memory locations [24][25][26], so that the probability of row or bank conflicts in DRAMs are considerably reduced. Under certain conditions, statistical bounds (such as average delay) can be found.

Our work on packet buffer design was first described in [27][28] and has some similarities with previous work done in [29][30][31][32].

VIII. 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, and decreases with an increase in the pipeline delay, (2) A lookahead buffer (if any) to hold requests, and (3) 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 too 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 that are commercially available today, and should enable packet buffers to be built for several generations of technology to come.

References

[1] Available at *http://www.caida.org/analysis/workload/byapplication/oc48/stats.xml*

[2] Villiamizar C. and Song C., "High performance tcp in ansnet," *ACM Computer Communication Review* (1995).

[3] Cisco 12000 Series Gigabit Switch Router (GSR) Gigabit Ethernet Line Card, available at *http://www.cisco.com/warp/public/cc/pd/rt/12000/prodlit/gspel_ov.htm*

[4] M-series Routers available at *http://www.juniper.net/products/dsheet/100042.html*

[5] "Packet Length Distributions", available at http://www.caida.org/analysis/AIX/plen_hist/

[6] "Round-Trip Time Measurements from CAIDA's Macroscopic Internet Topology Monitor", available at http://www.caida.org/analysis/performance/rtt/walrus2002

[7] D. A. Patterson and J. L. Hennessy, *Computer Architecture*, A Quantitative Approach, Section 8.4., pp. 425-432, Morgan Kaufmann, 1996,

[8] K.G. Coffman and A. M. Odlyzko, "Is there a "Moore's Law"" for data traffic?," Handbook of Massive Data Sets, eds., Kluwer, 2002, pp. 47-93.

[9] http://www.edram.com/products/legacy/ESDRAMlegacy.htm

[10] www.rambus.com/technology/rdram_overview.shtml

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] R. Bhagwan and B. Lin. Fast and scalable priority queue architecture for high-speed network switches. In Proceedings of IEEE INFOCOM'00, 2000.

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

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

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

[24] L. Carter and W. Wegman, "Universal hash functions," *Jour. of Com-*

[12] This is sometime referred to as *bit striping*.

puter and System Sciences 18, 1979, pp. 143-154.

[25] R. Impagliazzo and D. Zuckerman, "How to recycle random bits", *Proc. of the Thirtieth Annual Symposium on the Foundations of IEEE*, 1989.

[26] 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.

[27] S. Iyer, R. R. Kompella, and N. McKeown, "Analysis of a memory architecture for fast packet buffers," *In Proc. IEEE HPSR*, Dallas, Texas, 2001

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

[29] 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.

[30] 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.

[31] 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.

[32] D. Shah, S. Iyer, B. Prabhakar, N. McKeown, "Maintaining Statistics Counters in Router Line Cards," IEEE Micro, pp. 76-81., Jan-Feb. 2002.

APPENDIX A: Proof of Lemma 2

Lemma 2: Under the MDQF-MMA, which services requests without any pipeline delay,

$$F(i) < bi[2 + \ln(Q/i)], \forall i \in \{1, 2, \ldots, Q-1\}. \quad (27)$$

Proof: The case when $i = 1$ is already proved in Lemma 1 and when $i = Q$ it is obvious as derived in Equation (13). For $\forall i \in \{2, \ldots, Q-1\}$, we again solve the recurrence relation obtained in Equation (12) to obtain,

$$F(i) \le i\left[b + b\left(\sum_{j=i}^{Q-1} \frac{1}{j}\right)\right], \ \forall i \in \{2, \ldots, Q-1\} \quad (28)$$

We can write the second term in the above equation as,

$$\sum_{j=i}^{Q-1} \frac{1}{j} = \sum_{j=1}^{Q-1} \frac{1}{j} - \sum_{j=i-1}^{Q-1} \frac{1}{j}, \ \forall i \in \{2, \ldots, Q-1\} \quad (29)$$

Since,

$$\forall N, \ \ln N < \sum_{i=1}^{N} \frac{1}{i} < 1 + \ln N, \quad (30)$$

we can use Equation (29) and Equation (30) to re-write Equation (28) as a weak inequality,

$$F(i) \le bi[2 + \ln(Q-1)/(i-1)], \ \forall i \in \{2, \ldots, Q-1\} \quad (31)$$

Thus we can write $\forall i \in \{2, \ldots, Q-1\}$,

$$F(i) < bi[2 + \ln(Q/i)]. \ \square \quad (32)$$