Building Packet Buffers using Interleaved Memories

Gireesh Shrimali and Nick McKeown
Computer Systems Laboratory, Stanford University
Stanford, CA 94305-9030
{gireesh, nickm}@stanford.edu

Abstract -- **High end routers need to store a large amount of data. Dynamic random access memories (DRAMs) are typically used for this purpose. However, DRAM memory devices don't match the bandwidth requirements, especially in terms of random access speeds. In this paper, we analyze a generalized memory interleaving scheme. This scheme implements a large, fast memory using multiple, slower DRAMs. In the presence of small amount of speed-up, we show that reasonable statistical guarantees (i.e., low drop probabilities) can be provided by using small SRAM buffers that queue read/write requests to DRAMs. We then relate drop probabilities to SRAM buffer size for a wide range of statistical arrival patterns.**

I. Introduction

High-performance routers need buffers to store data during congestion. These buffers can either be shared by multiple line cards (e.g., in a shared memory router) or belong to a single line card (e.g., in a distributed memory router). These buffers are typically required to store a large amount of data at high rates, which translates to using fast, high density memories.

Though there can be multiple criteria for deciding the amount of storage required, a rule of thumb indicates that, in order for TCP to work well, the buffer should be able to store an amount of (per-port) data equal to the product of the line rate and the average round-trip-time (RTT) [18]. Though this rule of thumb has been challenged recently [2], it is still widely used.

In addition, the arriving/departing packets may cause memory accesses in an arbitrary and unpredictable (i.e., random) order, thus requiring random access guarantees (in packets/s) in addition to raw bandwidth guarantees (in bits/s).

To get a feel for the size and speed requirements, consider a packet buffer on a 40Gb/s (OC768c) linecard of a distributed memory router. Assuming an average TCP RTT of 0.25s [3], this buffer needs to store 10Gb of data. In addition, assuming a constant stream of 40-byte packets, which corresponds to minimum size IP packets containing TCP ACKs, the packet buffer must read and write a packet every 8ns. This translates to one memory operation every 4ns, or a random access speed of 250Mpackets/s

Note that a packet buffer supporting sixteen 2.5Gb/s ports in a shared memory router would have the same size and speed requirements. We now look at two popular memory devices - SRAM and DRAM - to see if they satisfy the above requirements.

Static random access memories (SRAMs) are relatively fast but small, and power-hungry. At the time of writing, state-of-the-art commercial SRAM [15] holds 32Mbits, has a random access time of 4ns, and consumes 1.6W. This means a 40Gb/s linecard would require over 300 SRAM devices and consume approximately 500W. So, even though SRAMs meet the speed requirement, implementing a packet buffer using only SRAM memories would be impractical in terms of area as well as power.

Dynamic random access memories (DRAMs) are relatively large and consume much less power, but are slow. At the time of writing, state-of-the-art commercial DRAM [16] holds 1Gbits, has a random access time of 40ns, and consumes 2W. This means that a 40Gb/s linecard could be implemented using 10 such DRAM devices consuming only 20W.

Size dictates that we use DRAMs in our 40Gb/s linecard example. However, with packets arriving or departing every 4ns, DRAMs are not fast enough. This shortfall in random access speed 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 random access speed, which increases by only 10% every 18 months [13]. Line-rate, R, increases 100% every 18 months [5] and hence DRAM will fall further and further behind the needs of high speed packet buffers.

A solution to the above problem is to interleave memory accesses across multiple slower DRAMs to mimic a large, fast memory. The scheme is not new [8][13], and is already deployed in proprietary systems as well as in commercially available DRAM controllers [7][9]. But there is no literature which analyzes this technique from a statistical perspective, and we believe that existing designs are based on ad-hoc statistical assumptions without hard guarantees or bounds on performance [1][6][10][11][14]. Our main contribution is to analyze this architecture in a networking context, and provide performance guarantees from a statistical perspective.

The rest of the paper is organized as follows. Section II describes the packet buffer architecture used to provide the statistical guarantees. Section III provides analytically derived statistical guarantees. Section IV discusses implementation details and provides a design example in the context of the 40Gb/s linecard introduced earlier in this section. Section V concludes the paper.

II. Interleaved Memory Architecture

A. Basic Idea

Memory interleaving has been traditionally used in computer systems to increase the performance of memory and disk-array sub-systems [8][13]. The idea is simple (Figure 1) - the bandwidth R of a single, fast memory is matched using k slower memories with bandwidth R/b, where $k \geq b$. Each of these memories is accessed independently of the others. A memory management algorithm schedules writes and reads to the DRAMs.

For sake of simplicity we assume that data is accessed in fixed-size cells. This cell can be a single byte or more, depending on the implementation. In case of variable size packets, the packet can be split into cells and written into multiple DRAMs.

We then define R as the rate at which these cells need to be accessed from the packet buffer. By our previous definition, a single DRAM memory would be capable of transferring 1 cell per b cell-times. Similarly, k memories would be capable of transferring k cells per b cell-times. In this context, we define the speed-up as $s = (k \times R/b)/R = k/b$.

As an example, consider our 40Gb/s linecard which requires a random access every 4ns. With 40ns random access time DRAMs, b turns out to be 10 ($= 40/4$). Now, if we require a speedup of 1.1, k would have to be 11 ($= 1.1 \times 10$).

B. SRAM FIFOs

If the packet buffer maintained just one flow, the operation would be simple: the arriving cells could be written immediately into the DRAM memories in a round-robin manner, without the need for any intermediate storage. The cells corresponding to the incoming requests would be read from the DRAM memories in the same manner. This scheme works since there is a cell written to (and read from) each memory every $k \geq b$ cell-times, and no DRAM memory is ever over-subscribed.

Things get more complicated when there are multiple (say Q) flows in the system. Let us assume for now that the memory management algorithm is writing to DRAM memories in a round-robin manner on a per-flow basis. It could happen that two consecutive arriving cells (at rate R) corresponding to two different flows get mapped to the same DRAM memory. Since a DRAM memory can accept only one of them per b cell-times, the other one has to be queued in a write FIFO (Figure 1). Multiple consecutive cells could be mapped to the same write FIFO, with most of them waiting for the DRAM memory.

Given the unpredictable nature of cell arrivals, and the fact that most realistic packet buffers deal with multiple flows, it would be impossible for any memory management algorithm to avoid short-term overloads on the DRAM memories. On-chip SRAM FIFOs are needed to buffer up these short-term overloads.

In our discussion above, we focussed on the write FIFOs. It is interesting to note that the write and read FIFOs are similar except that write FIFOs store actual cells whereas the read FIFOs store requests for cells. We expect their behavior to be identical otherwise.

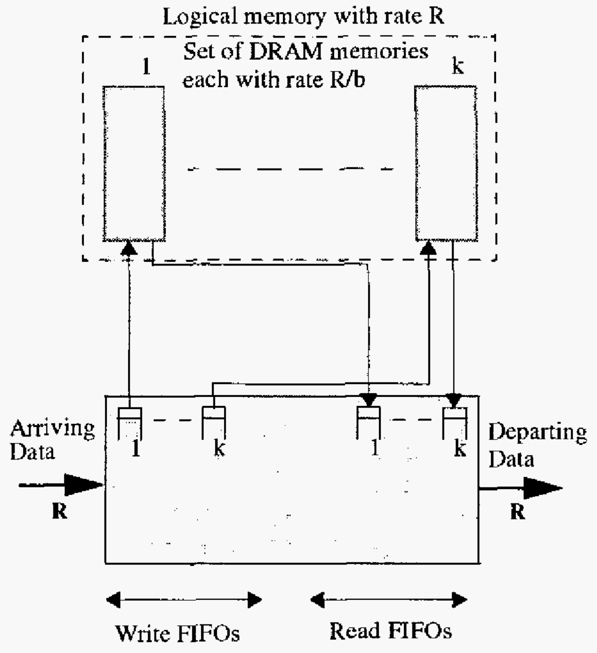
Logical memory with rate R
Set of DRAM memories each with rate R/b
1
k
Arriving Data
R
1
k
1
k
Departing Data
R
Write FIFOs
Read FIFOs

Figure 1: Memory hierarchy of packet buffer with multiple DRAM memories. DRAM write and read FIFOs are maintained in SRAM.

At this point it is relevant to mention per-flow queues - the logical queues that keep track of the ordering of cells on a per-flow basis. The per-flow queues are related to the per-DRAM FIFOs in the following manner. The write FIFOs contain the cells at the tails of the per-flow queues. Similarly, the read FIFOs contain the cells at the heads of the per-flow queues, whereas the DRAM contains the cells in the middle. Since the actual cell/request storage happens in the per-DRAM FIFO queues, the per-flow queues are relevant only from an implementation point of view, and we focus only on the analysis of per-DRAM FIFO queues in the remainder of the paper.

C. Memory Management

A DRAM memory interacts only with its own write and read FIFOs, independently of the other DRAM memories. As cells arrive to the packet buffer they are written by the memory management algorithm (MMA) to the tail of a write FIFO, where they wait to be written to the corresponding DRAM memory. Similarly, as requests arrive to the packet buffer, they are written to the tail of a read FIFO, where they wait for the corresponding cell to be read from the appropriate DRAM. The MMA comes into the picture only while writing a cell into the DRAM memories. For every flow, the order in which the DRAM memories are accessed is completely determined during the write operations. Thus, the per-flow requests must be queued into the read FIFOs in the same order the corresponding cells were written into the write FIFOs.

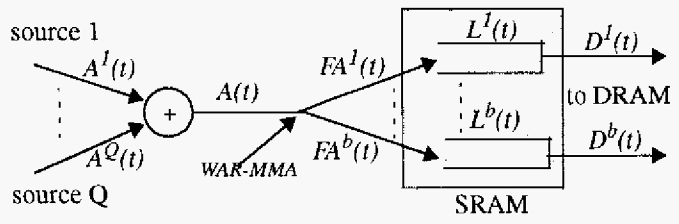

Figure 2: The SRAM model

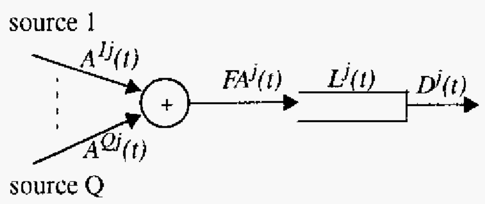

Figure 3: The FIFO model

D. *Problem Statement*

This scheme is not without its limitations. If the SRAM containing the FIFOs is not big enough, then cells may get dropped during overload situations. In addition, due to FIFO queueing, cells may experience variable latency in traversing the system.

In Section III we provide a memory management algorithm that ensures that the drop probability is minimized given fixed size SRAM. Using minimal assumptions on arrivals (and departures), we also provide statistical guarantees on drop probabilities and maximum latency. We show that reasonable performance guarantees, i.e., low drop probabilities and low maximum latency, can be provided using small values of speed-up.

III. Providing Statistical Guarantees

In this section, we focus on the analysis of the SRAM containing the write FIFOs. As mentioned earlier, the behavior of the read FIFOs is identical to the write FIFOs, and so is the analysis.

A. *Preliminaries*

We assume that $k = b$. This ensures that the SRAM buffer is rate stable since the cumulative service rate ($= 1$) is greater than or equal to the maximum incoming rate. We then analyze for the effects of speedup by looking at incoming rates that are less than unity.

We assume that the SRAM of size S is statically shared among the b FIFOs of size S/b each (Figure 2). Thus, we model the SRAM as a collection of b queues corresponding to the b FIFOs. Each of these b queues are served independently by a deterministic server with rate $1/b$.

For simplicity, we assume time to be a continuous variable - a similar analysis could be carried out in discrete time domain as well. We denote by $A(t)$ the cumulative number of cells arriving at the SRAM in $[0, t]$. $A(t)$ is assumed to be the sum of Q stationary and ergodic arrival processes $A^i(t)$ corresponding to Q flows, where $i \in [1, Q]$. $A^i(t)$ are assumed to have rates λ_i such that the sum of rates is less than 1 (to ensure rate stability).

We also assume $A^i(t)$ to be independent of each other. We believe the independence assumption to be a reasonable one since the traffic on an internet link, especially a high speed WAN link, usually comprises of traffic generated by thousands of independent sources.

B. *Memory Management Algorithm*

The memory management algorithm (MMA) assigns every incoming cell to one of the b write FIFOs uniformly and at random (u.a.r.). We refer to this MMA as the Write At Random MMA (WAR-MMA).

Since each incoming cell is assigned to the write FIFOs u.a.r., the same holds for cells belonging to a flow. The pattern of writes, i.e., the order in which the DRAM memories are accessed, uniquely determines the pattern of reads on a per-flow basis. Thus, on the read side, the per-flow requests would also be distributed u.a.r. across the DRAM memories.

The u.a.r. assignment means that each $A^i(t)$ can be written down as the following

$$A^i(t) = \sum_{j=1}^{b} A^{i,j}(t) \quad , i \in [1, Q] \tag{1}$$

where $A^{i,j}(t)$ are the independent processes generated by the u.a.r. sampling of $A^i(t)$. The arrival process $FA^j(t)$ to FIFO j can now be written as

$$FA^j(t) = \sum_{i=1}^{Q} A^{i,j}(t) \quad , j \in [1, b]. \tag{2}$$

Thus, we can envision FIFO j as shown in Figure 3. It has Q sources $A^{i,j}(t)$, with rate λ_i/b, multiplexing into it, and is work conserving with rate $1/b$, with total load λ. Rate stability is preserved per FIFO, i.e.

$$\sum_{i=1}^{Q} \lambda_i/b = \lambda/b < 1/b \,. \tag{3}$$

Now, similar to the arrival processes, we denote by $D(t)$ the cumulative number of departures from the SRAM in $[0, t]$, where $D(t)$ is the sum of b departure processes $D^j(t)$ from the FIFOs. We then denote the number of cells present in queue j, and the total SRAM occupancy at time t as $L^j(t)$ and

$$L(t) = \sum_{j=1}^{b} L^j(t) \,, \tag{4}$$

respectively.

Under WAR-MMA, since each FIFO j gets a b^{th} fraction of each $A^i(t)$ via u.a.r. sampling, we can intuitively expect the $FA^j(t)$ to be independent and identically distributed (i.i.d.). This, combined with the fact that each FIFO is served in an independent and identical fashion, leads us to expect that the steady-state occupancy of each of the b FIFOs will be i.i.d.. We formalize this in the following lemma [17].

Lemma 1: The occupancies $L^j(t)$ are i.i.d. under WAR-MMA.

C. Drop Probabilities

Since the event that the SRAM overflows is equivalent to any of the FIFOs overflowing, the overflow probability in the statically partitioned SRAM of size S can be given by

$$P_{overflow,SRAM}(S) \le \sum_{j=1}^{b} P_{overflow,FIFO(j)}(S/b). \qquad (5)$$

To find the drop probability from a finite SRAM of size S, we start by assuming that the SRAM is infinite, i.e., that each of the statically partitioned FIFOs are infinite. We then obtain the steady-state probability of the occupancy of an infinite FIFO exceeding S/b (i.e., $P(L^j > S/b)$) as a surrogate for the overflow probability in a finite FIFO of size S/b. This surrogate, also referred to as the buffer exceedence probability, suffices since it is generally an upper bound to the actual drop probability, i.e.,

$$P_{overflow,FIFO(j)}(S/b) \le P(L^j > S/b). \qquad (6)$$

Here the left hand term corresponds to the drop probability from a finite FIFO of size S/b, whereas the right hand term corresponds to the steady-state buffer exceedence probability (beyond S/b) in an infinite FIFO. Now Equation (5) becomes

$$P_{overflow,SRAM}(S) \le \sum_{j=1}^{b} P(L^j > S/b). \qquad (7)$$

Finally, since $L^j(t)$ are identically distributed (Lemma 1), we can re-write Equation (7) as

$$P_{overflow,SRAM}(S) \le b \times P(L^j > S/b). \qquad (8)$$

Equation (8) indicates that it suffices to find the steady-state distribution of buffer exceedence probability for any FIFO.

D. Buffer Exceedence Probability

We start by noting that using Lindley's recursion, the FIFO occupancy can be given by

$$L^j(t) = max_{0 \le s \le t}((FA^j(t) - FA^j(s)) - (t-s)/b). \qquad (9)$$

This indicates that, given the steady-state distribution of the arrival processes $A^i(t)$, it is theoretically possible to derive the steady-state distribution of the FIFO occupancy. However, for general arrival patterns it is often not easy to find a solution. The good news is that for a broad range of arrival traffic patterns $A^i(t)$, characterized by the following assumptions, it is possible to do so.[1]

We assume that each $A^i(t)$ is a simple point process satisfying the following properties [4]. First, the expected value of $A^i(t)$ in any interval $[0, t]$ is given by $\lambda_i t$. Second, a source cannot send more than one cell at a time. And third, the probability of many cells arriving in an arbitrarily small interval $[0, t]$ decays fast as $t \to 0$.

We tackle the problem of finding the exceedence probability in two steps. We first look at the case where the $A^i(t)$ are i.i.d.. Given above assumptions, we derive the following lemma [17] and our main theorem, as follows.

Lemma 2: $A^{i,j}(t)$ are stationary, independent, and identically distributed simple point processes.

Theorem 1: As the number of flows increases (i.e., $Q \to \infty$), the steady-state buffer exceedence probability, i.e $P(L^j > x)$ $\forall x$, approaches the corresponding exceedence probability with a Poisson source with the same total load.

Proof: We have Q sources $A^{i,j}(t)$ multiplexing into FIFO j that is work conserving with total load λ. From Lemma 2, $A^{i,j}(t)$ are stationary, independent, and identically distributed simple point processes. Thus, all the assumptions stated for $A^i(t)$ are also true for $A^{i,j}(t)$ (for a fixed j). This allows us to use Theorem 1 in [4] to get the main result.❐

Theorem 1 shows that as the number of multiplexed i.i.d. sources increases, the steady-state buffer exceedence probability approaches the corresponding probability assuming Poisson sources, which is known explicitly through the analysis of the resulting $M/D/1$ system [12].

We then look at the case where the $A^i(t)$ are independent but not necessarily identically distributed. Using a large deviation argument [17], we can show that the buffer exceedence probability in this case is upper bounded by the corresponding probability for the i.i.d. case. Thus, in order to get the drop probability from an SRAM, it suffices to analyze for the i.i.d. case

IV. Implementation Considerations

WAR-MMA is pretty simple to implement since it only requires randomly spreading the incoming traffic. This requires no state to be kept. In addition, statically allocated SRAM (using circular buffers) and dynamically allocated DRAM (using linked lists) are common place. Thus, the interleaved memory architecture is fairly easy to implement. We believe this to be its biggest strength.

We now provide a realistic design example. In Table 1 we list values of FIFO sizes (i.e., x) needed to guarantee a steady-

[1] These assumptions are fairly general and apply to a variety of traffic sources, including Poisson, Gamma, Weibull, Inverse Weibull, ExpOn-ExpOff, and ParetoOn-ExpOff. These point processes model a wide range of observed traffic [4], including wide area network traffic.

state buffer exceedence probability of 10^{-6} as well as 10^{-9}. These FIFO sizes are independent of Q, and depend only on λ. Q is important as long as it is large enough to guarantee the convergence indicated in Theorem 1.[2]

TABLE 1 FIFO Size v/s Exceedence Probabilities

Load	x when P(L(j)>x)=1e-6	x when P(L(j)>x)=1e-9
0.1	5	7
0.3	8	11
0.5	12	18
0.7	22	32
0.9	68	101

For example, to achieve a steady-state buffer exceedence probability of 10^{-9} at $\lambda = 0.9$, we need a FIFO size of approximately 101 cells. Since $\lambda = 0.9$ corresponds to a speedup of approximately 1.1 ($= 1/0.9$), at this speedup an SRAM of size $b \times 101$ would guarantee that the drop probability from the SRAM is upper bounded by $b \times 10^{-9}$ (Equation (8)).

In our 40Gb/s linecard example, with 40ns random access time DRAMs and 40-byte cells, b turns out to be 10 (Section II.A). Now, with $b = 10$ and speedup 1.1, an SRAM of size 1010 cells ($= 40400$ bytes) would guarantee a drop probability of less than 10^{-8}.

This example validates our claim that a small SRAM (~40K bytes) can support extremely low drop probabilities using a small amount of speedup.

V. Conclusions

Packet switches, regardless of their architecture, require packet buffers. The general architecture presented and analyzed here can be used to build high bandwidth packet buffers. The scheme uses a number of DRAMs in parallel, all controlled independently in an interleaved manner, as well as SRAM FIFOs to absorb short-term overloads on the DRAM memories.

In this paper, we establish exact bounds relating the SRAM size to the drop probability. In particular, we show that reasonable performance guarantees, i.e., low drop probability, can be provided using small SRAMs as well as a small amount of speed-up.

The costs of the technique are: (1) a (presumably on-chip) SRAM cache that grows in size linearly with line rate, and (2) A memory management algorithm that must be implemented in hardware.

While there are systems for which this technique is inapplicable (e.g. where the value of Q is small (<100), so that the convergence used in our analysis doesn't apply), 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.

[2] In practice, we start observing the convergence indicated in Theorem 1 when the number of flow exceeds 100.

References

[1] 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.

[2] G. Appenzeler, I. Keslassy, and N. McKeown, "Sizing Router Buffers", *In Proceedings of ACM SIGCOMM'04*.

[3] Caida, "Round-Trip Time Measurements from CAIDA's Macroscopic Internet Topology Monitor", available at http://www.caida.org/analysis/performance/rtt/walrus0202.

[4] J. Cao and Kavita Ramanan, "A Poisson Limit for Buffer Overflow Probabilities", *In Proceedings of IEEE INFOCOM'02*. pp. 994-1003. 2002.

[5] 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.

[6] 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.

[7] 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.

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

[9] 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.

[10] 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.

[11] 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.

[12] U. Mocci, J. Roberts, and J. Virtamo, "Broadband Network Teletraffic", *Final Report of Action COST 242*, Springer, Berlin, 1996.

[13] D. Patterson, and J. Hennessy, *Computer Architecture: A Quantitative Approach*, 2nd. ed., San Francisco: Morgan Kaufmann Publishers, 1996.

[14] 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.

[15] Samsung, "Samsung K7N323645M NtSRAM". Available at http://www.samsung.com/Products/Semiconductor/SRAM/index.htm.

[16] Samsung, "Samsung K4S1G0632M SDRAM". Available at http://www.samsung.com/Products/Semiconductor/DRAM/index.htm.

[17] G. Shrimali and N. McKeown, "Statistical Guarantees for Packet Buffers: The Distributed DRAM Case", *Stanford University HPNG Technical Report*, Stanford, CA, 2004.

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