# Memory Subsystems in High-End Routers

As Internet routers scale to support next-generation networks, their memory subsystems must also scale. Several solutions combine static RAM and dynamic RAM buffering but still have major scaling limitations. Using a parallel architecture and distributed memory-management algorithms with hybrid SRAM/DRAM improves buffering performance. The parallel hybrid SRAM/DRAM memory system is also work conserving, which is particularly important under light traffic conditions.

**Feng Wang**
**Mounir Hamdi**
Hong Kong University of Science and Technology

Memory technologies are becoming bottlenecks for electronic devices that need buffering. This fact is particularly evident in the evolution of electronic routers, where memory size and latency significantly impact the design of high-end routers. First-generation routers used a fully shared buffering system. For these routers, the memory bandwidth had to be $N$ times faster than the line rate, where $N$ is the number of line cards. This architecture is obviously nonscalable when both $N$ and the line rate increase. Second-generation routers tried to distribute buffers into individual line cards, thus reducing the total bandwidth required on each memory. Crossbar technology let router designers further reduce the required memory bandwidth using input queuing or combined input/output queuing.

However, router buffers must also scale linearly with the line rate. With the ever-increasing Internet line rate, current memory technologies alone—namely static RAM and dynamic RAM—can't simultaneously satisfy both the router buffer's size and latency requirements.[1,2] SRAM is fast enough, with an access time of around 4 ns; however, current fabrication technologies limit its size to at most a few megabytes. In addition, SRAM also requires significant operating power. On the other hand, DRAM can be built with a large capacity, but its typical memory access time is around 40 ns. This performance discrepancy might be largely due to the fact that current memory technologies are mainly optimized for computers, not for routers. Extensive research has focused on designing computer buffers in which a portion of data and instructions in the memories is reused many times (typically referred to as the *locality* property[3]). Relatively less attention has been paid to router buffers, where data packets are seldom reused and each one is treated equally in terms of processing. The locality property lets computers function using a hierarchical, cache-based buffering system, consisting of small, fast, high-level memories (such as a register and SRAM) and large, low-speed buffers (such as DRAM and hard disks). However, the locality property doesn't hold for routers: each packet comes into the memory (usually



Published by the IEEE Computer Society

only once) and leaves sometime afterward, never to return.

This tutorial overviews major challenges in router buffer design and surveys some current solutions for high-end router buffers. These solutions have major scaling problems, especially with regard to the increasing number of network flows. Our technique, which uses parallelism to address the nonscalability issue, significantly outperforms previous solutions and is a promising technology for building next-generation routers.

## Challenges in high-performance router buffers

In general, the main challenges in the design of high-end router buffers are storage capacity, access time, and queuing scheme.

### Storage capacity: How large should a router buffer be?

The traditional rule of thumb is that for the transmission control protocol, TCP, to work well under network congestion, routers must maintain a size of $RTT \times R$,[4] where $RTT$ is a packet's round-trip lifetime and $R$ is the line rate. This formula assumes a single TCP flow. The flow adapts its rate following an additive-increasing-multiple-decreasing discipline. Appenzeler, Keslassy, and McKeown challenged this rule, saying that core routers could use a much smaller buffer.[5] However, disputes remain about router buffer sizing, and vendors still want to use the traditional rule. In fact, both rules indicate that the size of necessary memories must scale linearly with the line rate $R$. Nevertheless, researchers agree that current SRAM technology alone is insufficient and a combination of SRAM and DRAM is necessary for building high-end router buffers.[1,2,6]

### Access time: How responsive should a router buffer be?

When building high-performance routers, designers must also consider the buffer's latency, or access time. Assume, for example, a 40-byte TCP packet and a network line rate of 40 Gbps. The buffer should be able to accommodate a packet every 8 ns. This access requirement falls between the capabilities of current SRAM (with a latency of around 5 ns) and DRAM (with a latency of around 30 ns). Although it's possible to use multiple simultaneous accesses to a memory to increase its bandwidth, the access time can't be made any shorter. This fact also indicates that although a designer might improve a DRAM's bandwidth and that the DRAM can be large, coping with fast packet access still requires SRAM.

### Queuing scheme: How are packets buffered in routers?

A typical router maintains multiple flow queues, making the router buffer even harder to build. For example, input-queued switches rely on a virtual-output-queuing technique, in which each input buffer maintains $N$ FIFO queues corresponding to $N$ outputs.[7] Even in output-queued switches, buffers maintain separate queues for different flows to support the flows' quality-of-service (QoS) requirements.[8] Although each flow queue might follow a simple FIFO process, queue management is a challenging task for buffer designers, especially when using a combination of SRAM and DRAM.

## Combining SRAM and DRAM

To simultaneously meet the stringent latency and size requirements for buffers of high-end routers, a natural solution is to combine SRAM and DRAM to exploit their individual advantages. The basic idea is to use the SRAM in the head and tail for fast reading and writing, and DRAM in the middle for buffering most packets. A *memory-management algorithm* (MMA) controls the packets as they shuttle between the SRAM and DRAM. The MMA must find a way to match the access time gap between the SRAM and DRAM so the two technologies can work together smoothly to meet the outside line rate requirements and provide sufficient buffering capacities. Figure 1 shows a generic model of such a combined SRAM/DRAM buffer implementation.

Many such architectures and corresponding MMAs have been proposed. We categorize them into two groups: those in which the MMA explicitly handles packet flow information and those in which it does not.

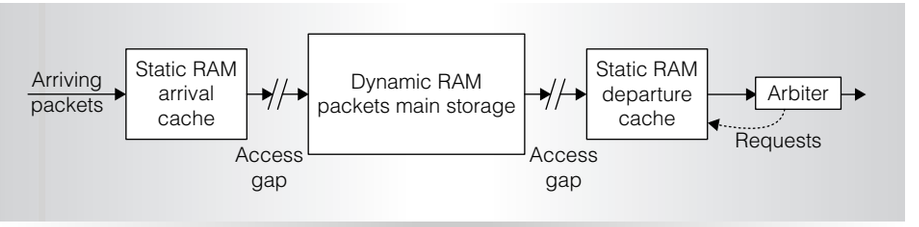


Figure 1. A generic SRAM/DRAM combination for packet buffers. Two SRAMs reside in the head and tail for fast packet reading and writing. DRAM resides in the middle for packet buffering.

## MMA with packet flow information

Router buffers must support multiple queues, both conceptually and in practice. A natural way to make the SRAM/DRAM buffer shown in Figure 1 support multiple flows is for each SRAM and DRAM to support multiple queues directly, as Figure 2 shows. The MMAs must match the latency gap between the SRAM and DRAM so no memory will overflow and the outside arbiter requests can be fulfilled within a reasonable delay.

To make the buffering system easy to analyze, we assume the system maintains $Q$ flows, and the DRAM's access time is $b$ times that of the SRAM. In practice, $b$ can be regarded as a constant (around 10 according to current manufacturing technologies), but $Q$ might range from hundreds to millions. For simplicity, we assume the SRAM access time is one time slot, and the DRAM access time is $b$ time slots.

Figure 2 shows the hybrid SRAM/DRAM memory architecture that Iyer, Kompella, and McKeown describe elsewhere.[1,9] To match the latency gap between the SRAM and DRAM, this hybrid SRAM/DRAM uses an *earliest-full-queue-first* (EFQF) MMA in the tail. The EFQF MMA waits until one of the queues in the tail SRAM reaches $b$ packets and then transfers these $b$ packets as a whole to the corresponding queue in the DRAM. Writing $b$ packets into the DRAM costs $b$ time slots. In this way, packets appear to move through the hybrid SRAM/DRAM smoothly.

The EFQF MMA works similarly in the head, because hybrid SRAM/DRAM is

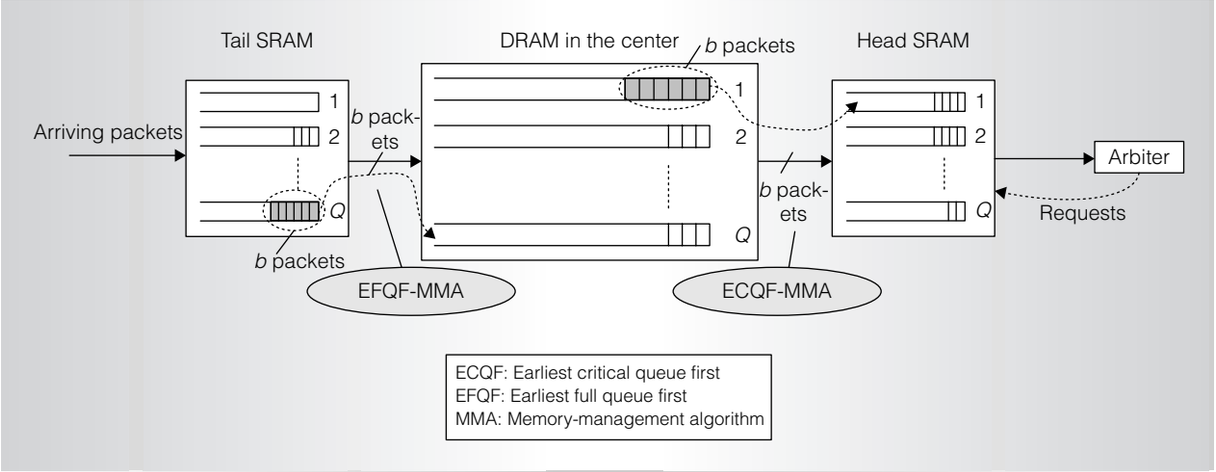


Figure 2. The basic hybrid SRAM/DRAM architecture for packet buffers maintaining $Q$ flows. In this architecture, SRAM and DRAM support multiple flows directly, while the memory-management algorithm (MMA) matches the latency gap between the two technologies.

symmetric. Here, the EFQF MMA gathers the outside requests into individual flow queues until one queue reaches $b$ packet requests. It then transfers those $b$ packets as a whole from the DRAM.

Because packets and requests should wait in the tail and head SRAM, respectively, until the buffer accumulates $b$ units, the SRAM should be sized to hold these waiting units, and outside requests might experience a delay before being fulfilled. Iyer et al. analyzed the worst case and showed that an SRAM size of $Q(b-1)$ is sufficient for any incoming traffic patterns, and packets might experience a delay of at most $Q(b-1)+1$ time slots for any sequence of outside requests.[9]

March and Corbal noticed that the SRAM size requirement in this scheme is $O(Qb)$ where $b$ is the DRAM access time, in time slots.[2] They improved the system by redesigning the DRAM part. They used interleaved DRAM technology, arguing that overlapping multiple accesses to interleaved DRAM banks can reduce the effective DRAM access time. Figure 3 shows this architecture.

Here, each DRAM group has $M$ banks interleaved. Theoretically, you can reduce the effective access time to the DRAM to $b/M$ by pipelining accesses uniformly into all the $M$ banks, at the cost of additional conflict-solving algorithms.[10] Therefore, the required size of the SRAM can be reduced to $O(Qb/M)$.

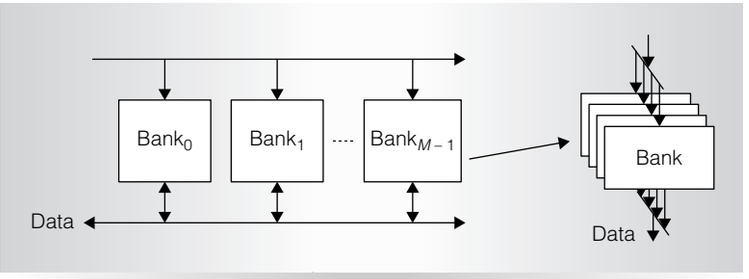


Figure 3. Two views of the hybrid SRAM/DRAM architecture, in which $M$ memory banks are interleaved to reduce the DRAM's effective access time.

## MMA without packet flow information

Another group of memory architectures doesn't explicitly consider packet flow queues.[6,11] To hide the $b$ times access time gap, they distribute $b$ write/read requests from the SRAM to the $k$ DRAMs in parallel ($k \geq b$). Each DRAM only has one queue, instead of $Q$ queues.

*Parallel DRAM with random access.* Figure 4a shows Shrimali, Keslassy, and McKeown's buffering architecture,[11] which we call *parallel DRAM with random access*. The SRAM's bandwidth $R$ is matched by $k$ parallel slower DRAMs whose bandwidth is $R/b$ ($b \leq k$). The SRAM maintains $k$ FIFO queues, and each DRAM maintains one FIFO queue corresponding to one of the SRAM's $k$ FIFO queues. Note that the $k$ queues in the SRAM aren't assigned to

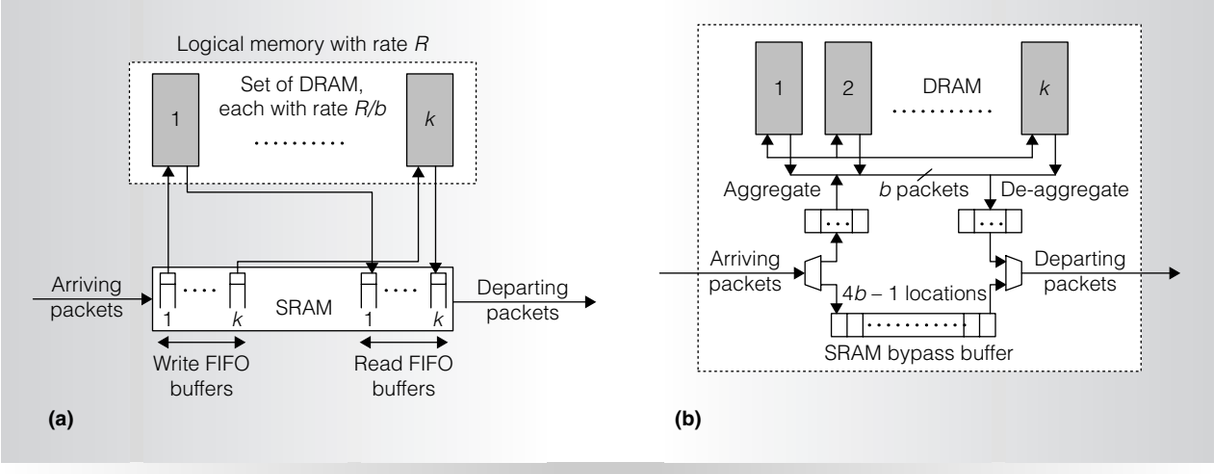


Figure 4. Architectures using MMA without packet flow information: parallel DRAM with random access (a), and parallel DRAM for packet with deterministic departures (b).

flows, and $k$ is much less than $Q$. In fact, packets from the same flow can be scattered into any one of the $k$ queues. Because it's assumed that the scheduler in the head SRAM knows exactly which DRAM a packet is placed in, the scheduler can maintain flow information. Agrawal and Sherwood proposed a write-at-random MMA to dispatch incoming packets to one of the $k$ FIFO queues uniformly and at random.[10] They calculated the drop probability with the SRAM's size and showed that this architecture can provide statistical performance guarantees (low drop probability) to every flow in the system with small SRAM.

*Parallel DRAM for packet with deterministic departures.* Kabra, Saha, and Lin delegated flow queue maintenance to outside schedulers.[6] Their approach assumes that when a packet enters the buffering system, it has a determined departure order. The outside scheduler decides the order, which doesn't change when the packet is in the memory. Thus, the buffering requires no queue information. We call this architecture *parallel DRAM for packet with deterministic departures.* This architecture is similar to the parallel DRAM with random access architecture, except that the SRAM doesn't maintain queues, as Figure 4b shows.

Consider the situation of $b$ incoming packets. Because the MMA knows exactly when each packet will depart, it can carefully choose $b$ of the $k$ DRAMs to write them, making sure that they won't cause read collisions in the future. Kabra, Saha, and Lin proved that $k \geq 3b - 1$ DRAMs are enough for the system to work well.[6]

## Scaling limitations of previous solutions

Relying on an outside scheduler to maintain flow queues—as in the parallel DRAM with random access and parallel DRAM for packet with deterministic departures techniques—is impractical in most high-end routers. In fact, to provide QoS, packets are scheduled according to their individual flows and the departure order can't be decided before it's time for the packet to go. Next-generation router buffers should maintain flow queue information explicitly in the system so the various scheduling algorithms can easily access it.

We base our research on the hybrid SRAM/DRAM buffering architecture. We notice, however, that both the SRAM size requirement and packet delay in this architecture is $O(Qb)$—that is, is linear with the number of flows—which is obviously nonscalable with the increasing number of flows in today's core networks. Iyer et al. showed that $O(Qb)$ is the worst case.[9] Here, we show that the expected SRAM occupancy and packet delay are also $O(Qb)$; it's this intrinsic limitation that makes hybrid SRAM/DRAM nonscalable.

In particular, we consider general practical traffic conditions, which can help us better understand a network system. By practical traffic, we mean that we make the following weak assumptions about the traffic:

- All the $Q$ flows are independent of each other.
- Each flow is a stationary and ergodic process.

These assumptions are reasonable and practical, because in reality a flow source doesn't usually interfere with other sources. Furthermore, the stationary and ergodic properties are shown in nearly all types of practical traffic, such as uniform, hot-spot, diagonal, and even bursty traffic in the long term. In addition, all flows don't necessarily have identical distributions.

To analyze the size of the SRAM required under this practical traffic, we assume an unlimited SRAM size and derive its expected occupancy. We focus on the tail SRAM in hybrid SRAM/DRAM. The EFQF MMA receives packets from outside and puts them in their individual flow queues in the tail SRAM. Once a queue has accumulated $b$ packets, the EFQF MMA transfers the block of $b$ packets as a whole from that queue to the DRAM. Based on this behavior, we can easily derive the expected occupancy of the tail SRAM in the hybrid SRAM/DRAM system.

**Theorem 1.** The expected occupancy of the tail SRAM in hybrid SRAM/DRAM with EFQF MMA is at least $Q(b - 1)/2$, with incoming traffic conforming to the aforementioned two assumptions.

**Proof.** The tail SRAM should be large enough to hold the residual packets for

each of the $Q$ flows that are still waiting for the EFQF MMA to transfer them to the DRAM. Suppose the EFQF MMA has run for a sufficiently long time, and each flow $i$ has $p_i$ packets that have arrived in the system, with $1 \leq i \leq Q$. Then we can represent each $p_i$ as $p_i = b \times m_i + q_i$, with $0 \leq q_i < b$. This representation tells us that for each flow $i$, there are at least $q_i$ packets residing in the tail SRAM, because the EFQF MMA only transfers packets in a batch of $b$ packets.

Therefore, the SRAM should be at least large enough to hold these $\sum_{i=1}^{Q} q_i$ residual packets.

By rewriting $\sum_{i=1}^{Q} q_i$, we get

$$\begin{aligned}\sum_{i=1}^{Q} q_i &= \sum_{i=1}^{Q} \mathrm{I}_A(q_i = 1) \cdot 1 \\ &\quad + \sum_{i=1}^{Q} \mathrm{I}_A(q_i = 2) \cdot 2 + \ldots \\ &\quad + \sum_{i=1}^{Q} \mathrm{I}_A(q_i = b-1) \cdot (b-1) \\ &= \sum_{j=1}^{b-1} \sum_{i=1}^{Q} \mathrm{I}_A(q_i = j) \cdot j \quad (1)\end{aligned}$$

Here $\mathrm{I}_A(\cdot)$ is an indicator function defined as following:

$$\mathrm{I}_A(x) = \begin{cases} 1 & \text{if } x \text{ is true} \\ 0 & \text{if } x \text{ is false} \end{cases}$$

Each $q_i$ is within the range $[0, b-1]$. Because all flows are independent of each other and each flow is stationary and ergodic, all their residuals $q_i$ should be uniformly distributed across the interval $[0, b-1]$, which indicates

$$\mathrm{E}\left[\sum_{i=1}^{Q} \mathrm{I}_A(q_i = j)\right] = Q/b, (0 \leq j < b)$$

Therefore, by taking expectations on both sides in Equation 1, we get

$$\begin{aligned}\mathrm{E}\left[\sum_{i=1}^{Q} q_i\right] &= \sum_{j=1}^{b-1} \mathrm{E}\left[\sum_{i=1}^{Q} \mathrm{I}_A(q_i = j)\right] \cdot j \\ &= Q/b \cdot \sum_{j=1}^{b-1} j \\ &= Q/b \cdot (b \cdot (b-1))/2 \\ &= Q(b-1)/2\end{aligned}$$

That is to say, to initiate a transmission, the tail SRAM should have at least $Q(b-1)/2$ packets, which is just half of its worst-case requirement. ■

**Corollary 1.** In hybrid SRAM/DRAM, the expected packet delay in the head SRAM with the *earliest-critical-queue-first* (ECQF) MMA is at least $Q(b-1)/2$, with outgoing traffic conforming to the aforementioned two assumptions.

**Proof.** Using a similar analysis, we can see that the ECQF MMA in the head SRAM looks ahead at least $Q(b-1)/2$ packet requests to issue a transmission from the DRAM. Therefore, the expected delay a packet might experience is at least $Q(b-1)/2$ time slots. ■

So, both the expected SRAM requirement and packet delay in hybrid SRAM/DRAM are only half of their worst-case performance and are still on the order of $O(Qb)$, which is obviously nonscalable with increasing $Q$.

## Using parallelism to scale hybrid SRAM/DRAM

By carefully investigating operations of the ECQF/EFQF MMA and the given proof, we can intuit that the DRAM's larger access granularity increases the SRAM occupancy and consequently increases packet delay. Specifically, the DRAM's larger access granularity makes hybrid SRAM/DRAM non-work-conserving. That is, packets or requests might wait for the MMA to accumulate $b$ packets or requests in one queue. All flows might experience non-work-conserving behavior, and the SRAM should hold all waiting packets or requests for each flow. Consequently, the SRAM size or packet delay should be proportional to the number of flows, which is at least linear with $Q$.

The major challenge in using a combined SRAM/DRAM structure is finding a way to match the latency gap between the two memory technologies. Parallelism is a natural approach for filling in latency gaps. In fact, other researchers have used parallelism to match the latency gap between SRAM and DRAM.[6,11] However, their solutions don't support explicit flow queue information in the memory system. Here, we try to build a parallel system based on the hybrid SRAM/DRAM with flow queue support. Figure 5 illustrates our solution, which we call *parallel hybrid SRAM/DRAM*. Basically,

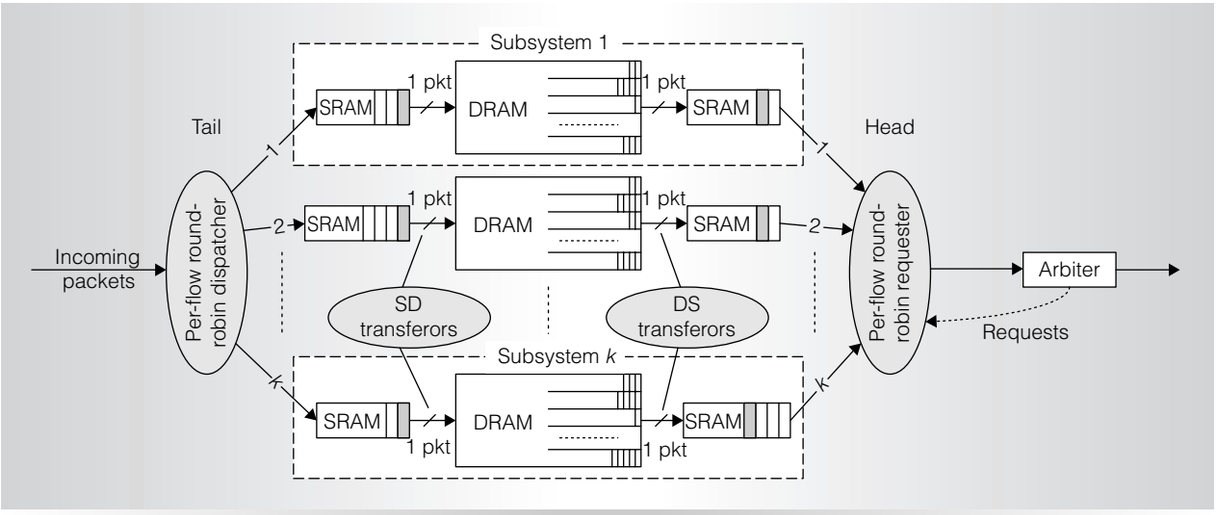


Figure 5. The parallel hybrid SRAM/DRAM system. Each of the $k$ subsystems uses a hybrid SRAM/DRAM architecture. However, only the DRAMs in the middle must maintain $Q$ flow queues.

it consists of $k$ ($\geq b$) parallel subsystems, each of which is a hybrid SRAM/DRAM structure. However, differences exist between these subsystems and the basic hybrid SRAM/DRAM system:

- In the subsystem in Figure 5, only the DRAM maintains $Q$ FIFO flow queues, whereas each SRAM maintains a single FIFO queue according to a packet's arriving order.
- Packet transmission between the SRAM and DRAM is in one-by-one mode, not $b$ packets in a batch as in the basic hybrid SRAM/DRAM system. That is to say, the DRAM's access granularity is also 1.

We designed a simple MMA for the parallel hybrid SRAM/DRAM system. The *round-robin SRAM/DRAM* (RRSD) MMA is fully parallel, asynchronous, and distributed in each of the $k$ subsystems. The algorithm's tail has two components:

- a *per-flow round-robin dispatcher* and
- an *SD transferor* between the SRAM and DRAM.

When a packet arrives at the memory system, the per-flow round-robin dispatcher first determines which subsystem the packet should go into according its flow ID and the round-robin rule. The dispatcher sends the packet to the subsystem's tail SRAM. According to the round-robin rule, if a packet is the $i$th packet in a flow, it should be dispatched into the $j$th subsystem, where $j = i \bmod k$. The dispatcher finishes writing to a SRAM in just one time slot.

The SD transferor keeps transferring the SRAM's head packets one by one into the corresponding flow queue in the DRAM whenever the SRAM is nonempty. The SD transferor completes a packet transfer in $b$ time slots.

The parallel hybrid SRAM/DRAM architecture is symmetric. We can use a similar MMA in the head part if we view the outside requests as virtual packets and buffer them in the head SRAM. Transferring packets from the DRAM to the head SRAM is similar to transferring virtual packets from the head SRAM into the DRAM. The head MMA is the same as the tail MMA and shares the same performance analysis. Therefore, we analyze only the tail MMA here.

## Performance analysis and simulations of parallel hybrid SRAM/DRAM

Parallel hybrid SRAM/DRAM is a simple system. It maintains the flow queue information while overcoming hybrid SRAM/DRAM's

non-work-conserving characteristics. That is to say, whenever the tail (or head) SRAM contains packets (or requests), the SD (or DS) transferor is busy transferring packets between the SRAM and DRAM. This will intuitively save a lot of space in the SRAM.

In addition, the RRSD MMA is as simple as having $O(1)$ complexity. This is because the parallel hybrid SRAM/DRAM is a distributed, asynchronous system, and complexity is amortized into every packet. The hybrid SRAM/DRAM system, on the other hand, is a centralized synchronous system, in which a central MMA determines which queue among $Q$ flows can be selected and then transfers $b$ packets from that queue synchronously.

To see how much SRAM is needed to make the parallel hybrid SRAM/DRAM work efficiently, we've performed extensive simulations comparing parallel hybrid SRAM/DRAM with hybrid SRAM/DRAM. In our simulations, we mainly test the SRAM occupancy and packet delay with regard to increasing the number of network flows.

In all the simulations, we set $b = 10$ and $Q$ from 100 to 10,000. All simulations run for $10^9$ time slots, and we record the maximum SRAM occupancies and packet delays. For comparison, we feed both hybrid SRAM/DRAM and parallel hybrid SRAM/DRAM exactly the same traffic in every simulation.

In Figure 6, we set $k = b$ in parallel hybrid SRAM/DRAM and use a 100 percent uniform traffic load. Both the SRAM occupancy and packet delay in hybrid SRAM/DRAM scale linearly with the increasing number of flows. However, in parallel hybrid SRAM/DRAM, these two performance metrics scale slowly, nearly at a speed of $O(\ln Q)$. The results in Figure 7 are more interesting: when we make $k$ a little larger, the maximum SRAM occupancy keeps unchanging with the increasing number of flows.

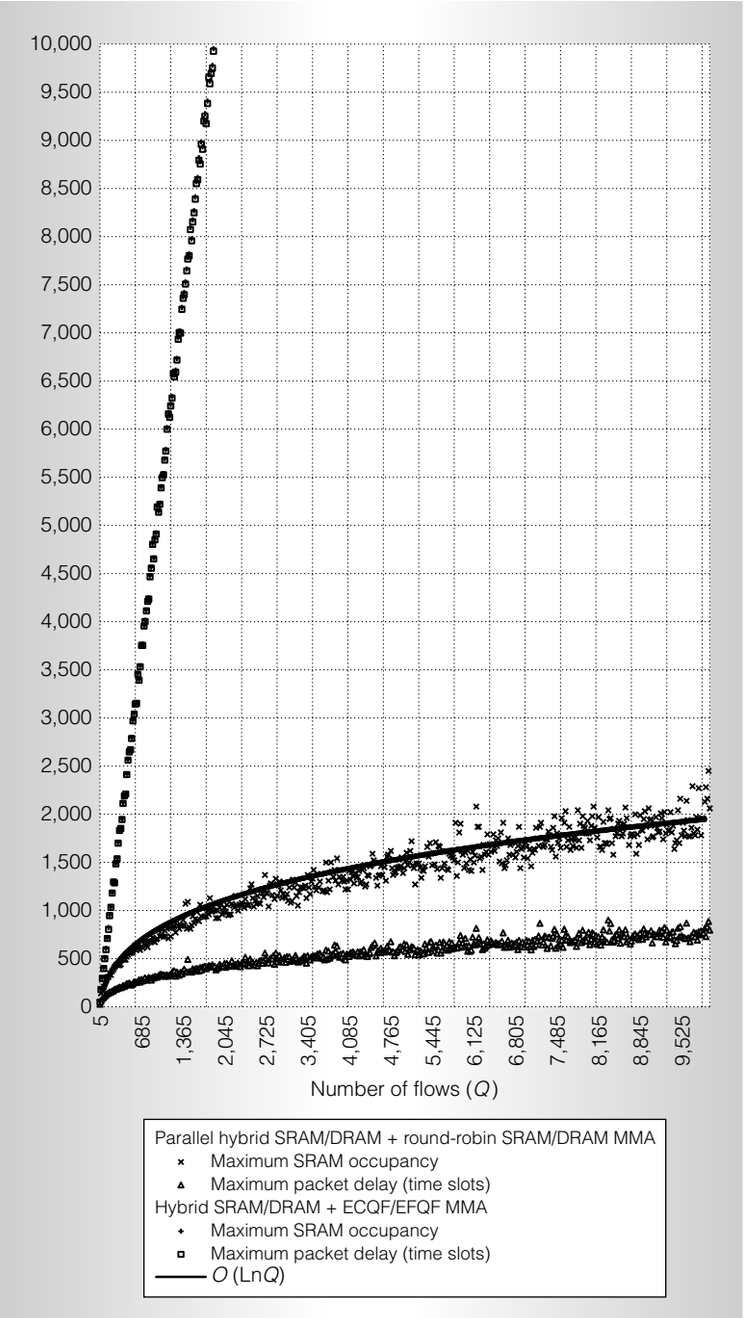


Figure 6. Performance of hybrid SRAM/DRAM and parallel hybrid SRAM/DRAM with an increasing number of $Q$ flows.

Figure 8 compares the performance of hybrid SRAM/DRAM and parallel hybrid SRAM/DRAM under different traffic loads. As Figure 8a shows, in parallel hybrid SRAM/DRAM, the SRAM occupancy decreases dramatically when the traffic loads decreases. However, as Figure 8b shows, in hybrid SRAM/DRAM, the SRAM occupancy keeps unchanging even under lighter traffic loads.

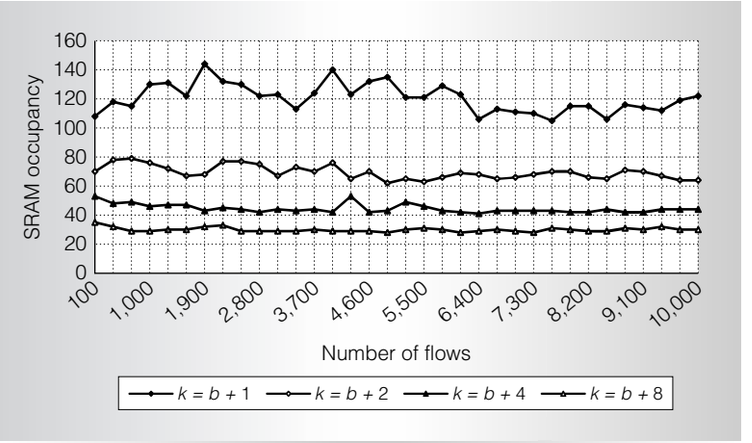


Figure 7. Maximum SRAM occupancy for parallel hybrid SRAM/DRAM with the round-robin SRAM/DRAM (RRSD) MMA when $k$ increases.

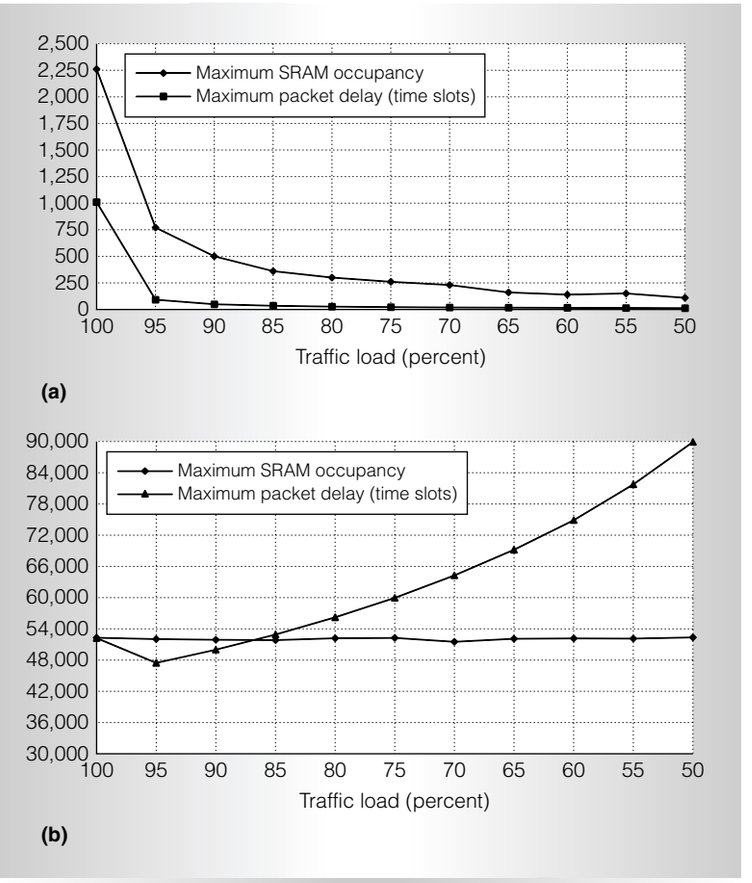


Figure 8. Performance under decreasing traffic loads: parallel hybrid SRAM/DRAM (a) and hybrid SRAM/DRAM (b).

The parallel hybrid SRAM/DRAM system's advantages don't come for free. Rather, they come at the cost of a possible packets-out-of-order problem. That is, in a given flow, a packet might respond to the arbiter earlier than preceding packets, because they're dispatched into different subsystems and might experience different delays. In the hybrid SRAM/DRAM, however, all packets follow the same route to the final arbiter, so packet sequence is strictly maintained. Packet order in a flow (for example, TCP) isn't a fundamental requirement in the Internet. However, when designing routers, we typically try to maintain packet order within the router, if we can achieve this at a reasonable cost, even though it might not be a strict requirement. This would prove beneficial for more advanced router features, such as QoS and various active-queue-management techniques. Future work will address this out-of-order problem in the parallel hybrid SRAM/DRAM system. MICRO

## Acknowledgments

The research described in the article was supported under Hong Kong University of Science and Technology grant HKUST RP07/08.EG09.

## References

1. S. Iyer, R. Kompella, and N. McKeown, *Designing Buffers for Router Line Cards,* tech. report TR02-HPNG-031001, Stanford Univ., 2002.
2. M. March and J. Corbal, ''A DRAM/SRAM Memory Scheme for Fast Packet Buffers,'' *IEEE Trans. Computers,* vol. 55, no. 5, May 2006, pp. 588-602.
3. J.L. Hennessy and D.A. Patterson, *Computer Architecture: A Quantitative Approach,* Morgan Kaufmann, 2006.
4. C. Villamizar and C. Song, ''High Performance TCP in ANSNET,'' *ACM Computer Comm. Rev.,* vol. 24, no. 5, Oct. 1994, pp. 45-60.
5. G. Appenzeler, I. Keslassy, and N. McKeown, ''Sizing Router Buffers,'' *ACM SIGCOMM Computer Comm. Rev.,* vol. 34, no. 4, Oct. 2004, pp. 281-292.

6. M. Kabra, S. Saha, and B. Lin, ''Fast Buffer Memory with Deterministic Packet Departures,'' *Proc. 14th IEEE Symp. High-Performance Interconnects* (HOTI 06), IEEE CS Press, 2006, pp. 67-72.
7. Y. Tamir and G.L. Frazier, ''High-Performance Multi-Queue Buffers for VLSI Communications Switches,'' *ACM SIGARCH Computer Architecture News,* vol. 16, no. 2, May 1988, pp. 343-354.
8. A. Demers, S. Keshav, and S. Shenker, ''Analysis and Simulation of a Fair Queuing Algorithm,'' *ACM SIGCOMM Computer Comm. Rev.,* vol. 19, no. 4, Sept. 1989, pp. 1-12.
9. S. Iyer, R.R. Kompella, and N. McKeown, ''Analysis of a Memory Architecture for Fast Packet Buffers,'' *Proc. IEEE Workshop High-Performance Switching and Routing,* IEEE Press, 2001, pp. 368-373.
10. B. Agrawal and T. Sherwood, ''Virtually Pipelined Network Memory,'' *Proc. Int'l Symp. Microarchitecture* (MICRO 06), IEEE CS Press, 2006, pp. 197-207.
11. G. Shrimali, I. Keslassy, and N. McKeown, ''Designing Packet Buffers with Statistical Guarantees,'' *Proc. 12th Ann. IEEE Symp. High Performance Interconnects* (HOTI 04), IEEE CS Press, 2004, pp. 54-60.

**Feng Wang** is a financial quantitative analyst in a Chinese investment bank. His academic research interests are generally in computer networks. In particular, he is focusing on the design and analysis of high performance switches and routers. Wang has a PhD in computer science from the Hong Kong University of Science and Technology.

**Mounir Hamdi** is a full professor and head of the Department of Computer Science and Engineering at the Hong Kong University of Science and Technology. He is also director of the Master of Science program in Information Technology and director of the Computer Engineering and Networking Lab. His main research area of interest is high-speed wired and wireless networking. Hamdi has a PhD in electrical engineering from the University of Pittsburgh.

Direct questions and comments about this article to Feng Wang, Flat F, 11/F, Chuk Lam Court, Lucky Plaza, Shatin, N.T., Hong Kong; franklin.f.wang@gmail.com.