Efficient and Novel Distributed Packet Buffers and High-Bandwidth Switches and Routers

G. Manjunath Reddy [1], P. Namratha [2]

[1]*M.Tech(CSE), Intell Engineering College, Ananthapur, Andhra Pradesh, India*
[2]*Assistant Professor, Dept.of CSE, Intell Engineering College, Ananthapur, Andhra Pradesh, India*

Abstract

The phenomenal growth of the Internet has been fueled by the rapid increase in the communication link bandwidth. Internet routers play a crucial role in sustaining this growth by being able to switch packets extremely fast to keep up with the growing bandwidth (line rate). This demands sophisticated packet switching and buffering techniques. Packet buffers need to be designed to support large capacity, multiple queues, and provide short response times. In this paper we studied the fundamental issues to be addressed for distributed packet buffering during the communication over the internet.

1. Introduction

Packet buffers are an essential part of routers. In high- end routers these buffers need to store a large amount of data at very high speeds. To satisfy these requirements, we need a memory with the speed of SRAM and the density of DRAM. A typical solution is to use hybrid packet buffers built from a combination of SRAM and DRAM, where the SRAM holds the heads and tails of per-flow packet FIFOs and the DRAM is used for bulk storage. The main challenge then is to minimize the size of the SRAM while providing reasonable performance guarantees.

High speed Internet routers and switches require fast packet buffer to hold packets during times of congestion. These buffers usually use a memory hierarchy that consist of expensive but fast SRAM and cheap but slow DRAM to meet both, speed and capacity requirements. A challenge building these packet buffers is to provide deterministic bandwidth guarantee under any traffic condition. We propose a novel hybrid packet buffer architecture with parallel DRAMs. Our approach reduces the amount of required SRAM compared to state-of-the-art architectures significantly, e.g., the tail SRAM by 47% for a 100Gbps line card using DDR3 SDRAM. Our architecture also applies packet aggregation and thereby minimizes the required DRAM and SRAM bandwidth and eliminates fragmentation. We are currently implementing the architecture on an FPGA and provide first results.

In order to support fine-grained IP quality of service(QoS) requirements, nowadays, a packet buffer usually maintains thousands of queues. For example, the Juniper E-series routers maintain as many as 64,000 queues. Given the increasing popularity of Open Flow, a packet buffer that supports millions of queues is always desired. Furthermore, a packet buffer should be capable of sustaining continuous data streams for both ingress and egress. With the ever-increasing line rate, current available memory technologies, namely SRAM or DRAM alone cannot simultaneously satisfy these three requirements. This prompted researchers to suggest hybrid SRAM/ DRAM (HSD) architecture with a single DRAM, interleaved DRAMs, or parallel DRAMs sandwiched between SRAMs.

In the following, we address the packet buffer requirements and the available memory technologies to meet these. Then we introduce the state-of-the-art hybrid memory approach to build packet buffers, before we outline our contribution.

- Packet Buffer Requirements
- Available Memory Types
- Hybrid SRAM/DRAM Memory Architecture

2. Related Work (Optical Buffer)

In this section we are going to address several questions such as

- Why do Routers have Buffers?
- How big should the Buffers be?
- How can Optical Data be stored?

There are three main reasons that routers have buffers.

1) Congestion: Congestion occurs when packets for a switch output arrive faster than the speed of the outgoing line. For example, packets might arrive continuously at two different inputs, all destined to the same output. If a switch output is constantly overloaded, their buffers will eventually overflow, no matter how large it is; it simply cannot transmit the packets as fast as they arrive. Short-term congestion is common due to the statistical arrival time of packets. Long-term congestion is usually controlled by an external mechanism, such as the end-to-end congestion avoidance mechanisms of TCP, the XON/XOFF mechanisms of Ethernet, or by the end-host application. In practice, we have to decide how big to make the congestion buffers. The decision is based on the congestion control mechanism—if it responds quickly to reduce congestion, then the buffers can be small; else, they have to be large. The congestion buffers are the largest buffers in a router, and so will be our main focus in this paper. A typical Internet router today holds millions of packet buffers for congestion.

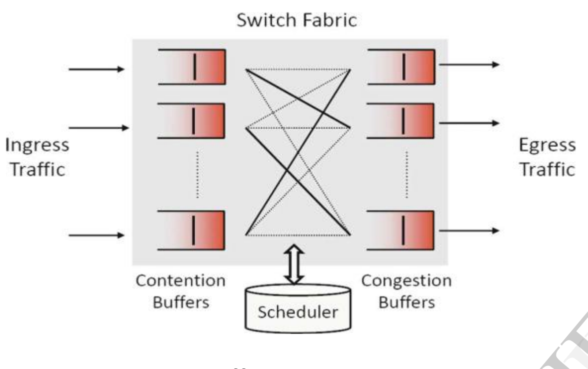


Figure 1: Buffering in a CIOQ router.

2) Internal Contention: Even when the external links are not congested, most packet switches can experience internal contention because of imperfections in their data paths and arbitration mechanisms. The amount of contention, and therefore the number of buffers needed, is determined by the switch architecture. For example, an output-queued switch has no internal contention and needs no contention buffers. At the other extreme, an input-queued switch can have lots of internal contention, as typified in the seminal paper of Karol that shows contention can limit the throughput of an input-queued switch to just 58% of its maximum. Between the two extremes, it is possible to build input-queued switches with 100% throughput. These switches need large internal buffers (theoretically, of infinite depth) to hold packets during times of contention. Some architectures can precisely emulate output queuing through careful arbitration and a combination of input and output queues (CIOQ). These switches still need contention queues (at their inputs) to hold packets while the arbitration algorithm decides when to deliver each to its output queue. Most switches today use CIOQ or multiple stages of CIOQ. As we will see in the next section, CIOQ switches typically need very small contention buffers. Fig. 1 shows the generic architecture of a CIOQ switch.

3) Staging: Packet switches also have staging buffers for pipelining and synchronization. Most designs have hundreds of pipeline stages, each with a small fixed-delay buffer to hold a fixed amount of data. Most designs also have multiple clock domains, with packets crossing several domains between input and output; each transition requires a small fixed-size FIFO. In this paper, we will not be considering staging buffers. Their sheer number means they cannot be ignored, but because they are of fixed size and delay, they can be implemented in various ways using small optical delay lines.

3. Semi Parallel Hybrid SRAM/DRAM (SPHSD) Packet Buffer Architecture

A. Architecture

Semi Parallel Hybrid SRAM/DRAM (SPHSD) architecture is depicted in Figure 3. Its core consists of k parallel DRAMs (or DRAM banks), one tail buffer and one head buffer. Each DRAM provides 1/k of the required bandwidth and contains Q FIFO flow queues, i.e. each logical flow queue is spread over all k DRAMs. The packet buffer aggregates packet data per-flow to constant size blocks. As always full blocks are written to and read from DRAM the total DRAM bandwidth is dimensioned to 2R, which is the minimum possible. So each DRAM provides a bandwidth of 2R/k, i.e. R/k for reading and R/k for writing. The random access time of a DRAM is T and so each DRAM performs one read and one write every 2T. Access time (2T) and bandwidth (R/k) of a DRAM define the block size of b = 2TR/k.

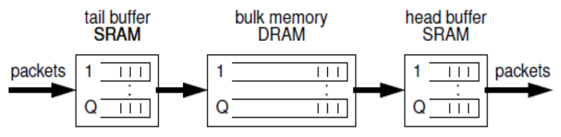


Figure 2: Basic Hybrid SRAM/DRAM Architecture

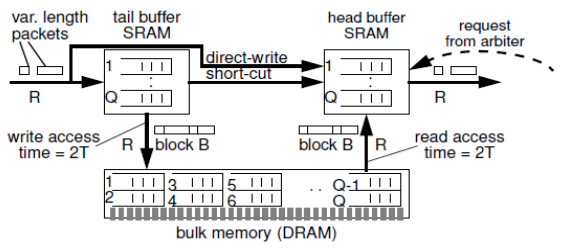


Figure 3: Parallel Hybrid SRAM/DRAM Architecture

B. Dimensioning of Parallelism

The parallelism of the architecture is controlled by the value of k. We will show in Section IV that with dynamic allocation the tail buffer size decreases with increasing k. The block size b = 2TR/k is inversely proportional to k. For k = 1 block size and basic architecture are equal. The minimum block size is determined by the used DRAM technology. E.g. with standard DDR3 SDRAM DIMMs (Dual Inline Memory Module) the smallest reasonable block size is 64byte. At a given line rate this determines the upper limit of k, e.g., for R = 100Gbps and T = 49ns, $\lceil k \rceil$ = 20. The organization overhead required for dynamic memory allocation increases towards smaller block size. This defines another upper limit for k.

C. Tail Round Robin Memory Management Algorithm

This section describes the round robin MMA utilized in the tail part. The MMA consist of two components: the per-flow round robin dispatcher and the tail transferor. Dispatcher and transferor work independently.

D. Head Round Robin Memory Management Algorithm

Our SPHSD architecture is symmetric. So a similar MMA can be used on the head side. The per-flow round robin requester behaves identical to the dispatcher except that it operates on packet requests instead of packets. Therefore, also the properties of the request queues are identical to that of the DRAM queues on tail side. As requests are negligible in size compared to blocks, the request buffer is not considered further. For the head transferor we have two options. First, the transferor can implement a trivial algorithm and process every packet request as soon as possible. As the head buffer acts as a reorder buffer this maximizes the head buffer size. Second, the transferor can behave inversely and wait before processing a request as long as possible, while still guaranteeing a constant read latency. With this, the head buffer has to reorder fewer packets. Utilizing dynamic memory allocation reduces head buffer size.

4. Packet Buffers for Router Linecards

In this section we study the packet buffer for router linecards with trail cache and head cache.

4.1 A Tail-Cache that never over-runs

Theorem 1: If dynamically allocated, the tail cache must contain at least bytes.

Proof: If there are bytes in the tail cache, then at least one queue must have or more bytes in it, and so a block of b bytes can be written to DRAM. If blocks are written whenever there is a queue with b or more bytes in it, then the tail cache can never have more than bytes in it.

4.2 A Head-Cache that never under-runs

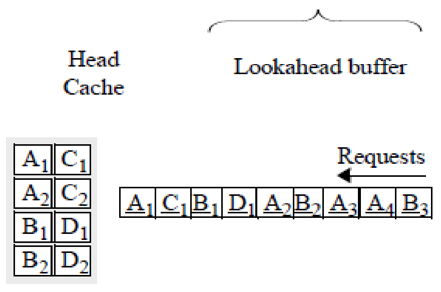
This section identified the implementation of head-cache algorithms with and without pipelining.

If we assume the head cache is statically divided into Q different memories of size, the following theorem tells us how big the head cache has to be (i.e. Qw) so that packets are always in the head cache when the packet processor needs them.

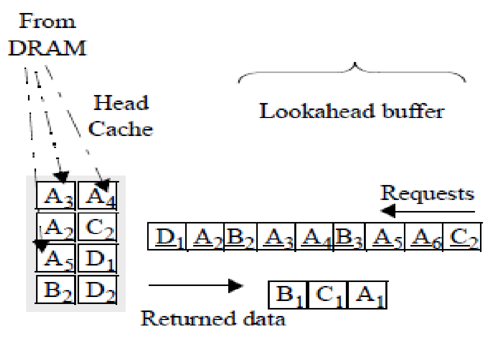
Theorem 2: To guarantee that a byte is always available in head cache when requested, the head cache must contain at least bytes.

Proof: It is one thing to know the theoretical bound; it is another matter to actually design the cache so as to achieve the bound. We need to find an algorithm that will decide when to refill the head cache from the DRAM; which queue should it replenish next? The most obvious algorithm would be shortest queue first; i.e. refill the queue in the head cache with the least data in it. It turns out that a slight variant does the job.

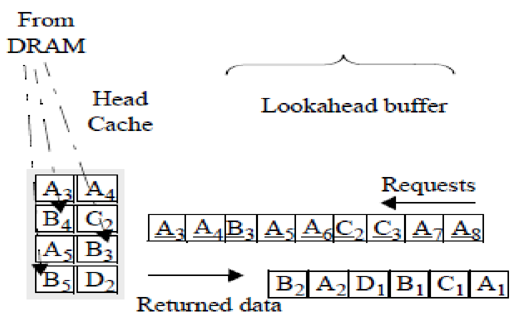
High-performance routers use deep pipelines to process packets in tens or even hundreds of consecutive stages. So it is worth asking if we can reduce the size of the head cache by pipelining the reads to the packet buffer in a look ahead buffer. The read rate is the same as before, it is just that the algorithm can spend longer processing each read. Perhaps it can use the extra time to get a "heads-up" of which queues need refilling, and start fetching data from the appropriate queues in DRAM sooner. We will now describe an algorithm that does exactly that; and we will see it needs a much smaller head cache.



(a) State at t = 0.
Q (A, 0) = 2; Q(A,6) critical.
Q (B, 0) = 2; Q(B,8) critical.
Queue A is requested.



(b) State at t = 3.
Q(A,3) = 4; Q(A,11) critical.
Q(B,3) = 1; Q(B, 9) critical.
Bytes A_3, A_4, A_5 arrive.
Queue B is requested.



(c) State at t = 6.
Q(A,6) = 3; Q(A, 11) critical.
Q(C,6) = 2; Q(C,13) critical.
Bytes B_3, B_4, B_5 arrive.
Queue A is requested.

Figure 4: ECQF with Q = 4 and b = 3 bytes.

When the packet processor issues a read, we are going to put it into the look ahead buffer shown in Figure 4. While the requests make their way through the look ahead buffer, the algorithm can take a "peek" at which queues are receiving requests. Instead of waiting for a queue to run low (i.e. for a deficit to build), it can anticipate the need for data and go fetch it in advance. As an example, Figure 4 shows how the look ahead buffer advances every time slot. The first request in the look ahead buffer at time slot request A1 in Figure 4 is processed at time slot as shown in Figure 4. A new request can arrive to the tail of the look ahead buffer every time slot request C2 in Figure 4b.

5. Packet Buffer with Statistical Guarantees

Packet buffers in high-performance routers are challenging to design because of two factors: memory speed and memory size. Packets belonging to different flows (for example, these flows may correspond to different IP source-destination pairs) arrive and depart at line rate, and are typically stored in per-flow queues. Consecutive pack- ets may belong to different flows in an unpredictable manner. This requires that the buffer be able to store as well as retrieve packets at line rates in an unpredictable memory access order. Thus, the buffer has to match a raw bandwidth (in bits/s) as well as a memory random access speed (in packets/s) of at least twice the line rate. In addition, a rule of thumb indicates that, for TCP to work well, the buffer should be able to store an amount of data equal to the product of the line rate and the average round-trip-time. While it has been recently challenged, this rule of thumb is still widely used. Therefore, both the speed and size of the memory grow linearly with the line rate. As an example, consider a 40Gbits/s linecard. This requires the buffer to match a raw bandwidth of 80Gbits/s. In addition, assuming a constant stream of 40-byte packets, which corresponds to minimum size IP packets containing TCP ACKs, the 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. Finally, assuming an average round-trip time of 0.25s, the buffer must hold 10Gbits. We now investigate the properties of two popular commercially avail- able memories - SRAM and DRAM - to see if they match these requirements.

We note that state-of-the-art SRAMs meet the raw bandwidth requirement of 80Gbits/s as well as the random access time requirement of 4ns. However, these SRAMs can only hold a maximum of 32Mbits per device. Thus, an SRAM-only solution would require over 300 SRAM devices, and therefore be very costly in terms of board real estate. In addition, these SRAMs consume approximately 1.6W per device. This means a total power consumption of 480W - more than the power budget typically allocated to the whole linecard. On the other hand, state-of-the-art DRAMs can hold up to 1Gbits per device, while consuming 2W per device. So, a DRAM-only solution would require only 10 DRAM devices, while consuming 20W, and therefore easily meet the real estate and power requirements. However, DRAM access times haven't kept up with the line rates - with today's DRAM technology, the random access times are in the range 20ns-40ns, and barely meet the requirements for even a 10Gbits/s line card. This shortfall is not going to be solved anytime soon since DRAMs are optimized for size rather than random access times, and the random access times improve by only 10% every 18 months. On the other hand, the line rate doubles in the same time period. Thus, this problem will get worse rather than better over time. Thus, an SRAM-only or a DRAM-only solution cannot meet both the speed and size requirements simultaneously. Since, overall, we would like to have a fast and large memory with the speed of SRAM and the density of DRAM, a solution would be to use both, in a manner very similar to computer systems where fast SRAMs are used as caches whereas dense DRAMs hold bulk of data.

6. Evaluation

The two main metrics of a hybrid packet buffer are (1) the required head and tail buffer size and (2) the read latency. In the following the upper bound for the tail and head buffer size as well as the read latency are derived.

For the following proofs we will assume that the minimal packet size P_{min} that can arrive or depart from the packet buffer is ≈ 0. This is a worst-case approximation that will slightly raise our bounds but simplify the proofs.

A. Tail Buffer Size

Theorem 1. If the tail buffer is statically divided in k partitions (one for each DRAM queue) the upper bound for the tail buffer size in blocks is Qk.

Proof: We know from Lemma 1 that no more than Q blocks can accumulate per DRAM queue. With k DRAM queues the upper bound is Qk blocks. This is nearly equal to the tail buffer size require. The difference originates from the different assumptions for P_{min}. Due to the per-flow round robin dispatching not all DRAM queues can be full at the same time. This allows a significant buffer size reduction with dynamic memory allocation.

Theorem 2. If dynamically allocated, the upper bound for the tail buffer size in blocks is Q(k + 1)/2

Proof: We assume that packets arrive at the packet buffer continuously with full line rate R. This represents the worst- case if we want to show that the buffer size is bounded. The proof consists of four steps leading to Lemma 2, 3, 4 and 5. We make the following observation: as long as any DRAM is idle because its DRAM queue contains no full blocks, tail buffer size will grow. The worst-case traffic pattern maximizes DRAM idle time and by this define the upper bound for the tail buffers size. In the following we define a traffic pattern and proof that it's the worst case traffic pattern, as it maximizes required buffer size.

B. Read Latency

We derive the read latency before the head buffer size, as the head buffer size depends on this value. The read latency is the time between issuing a read request to the packet buffer and receiving the packet. This corresponds to the minimum delay that a packet buffer introduces to every packet.

Theorem 3. The packet buffer has a constant read latency of Qk time slots.

Proof: The read latency is the sum of the maximum latencies introduced by head and tail part.

C. Head Buffer Size

Theorem 4. If the head buffer utilizes dynamic memory allocation and the head transferor processes every request as early as possible, then the upper bound for the head buffer size in blocks is Q(k + 1).

Proof: The head buffer (a) stores blocks to ensure in order delivery and a constant read latency and (b) stores not yet requested packet segments. Memory size for (a) is maximized, when Qk blocks of a single flow are requested consecutively starting from an empty request buffer. The head transferor processes each request immediately. After the read latency of Qk time slots (Q−1)k blocks are completely received from DRAM and k blocks are partly received. Memory size for (b) is maximized, when the Q − 1 other flows each have one segment of nearly the size of a full block available in the head buffer. Rounded up, the upper bound for the head buffer size is Q(k + 1) blocks.

7. References

[1]. Dong Lin, Mounir Hamdi, and Jogesh K. Muppala – "Distributed Packet Buffers for High-Bandwidth Switches and Routers", IEEE Transactions on Parallel and Distributed Systems, Vol. 23, No. 7, July 2012.
[2]. Arthur Mutter – "A Novel Hybrid Memory Architecture with Parallel DRAM for Fast Packet Buffers", 2010 IEEE.
[3]. Neda Beheshti, Emily Burmeister, Yashar Ganjali, John E. Bowers, Daniel J. Blumenthal, and Nick McKeown – "Optical Packet Buffers for Backbone Internet Routers", IEEE/ACM Transaction on Networking, Vol. 18, No. 5, October 2010, pg.no 1599 – 1609.
[4]. Hao Wang Bill Lin – "Block-Based Packet Buffer with Deterministic Packet Departures", Department of Electrical and Computer Engineering, University of California, San Diego, pg.no 38-43.
[5]. Sundar Iyer, Ramana Rao Kompella, Nick McKeown – "Designing Packet Buffers for Router Linecards", Stanford HPNG Technical Report TR02-HPNG-031001.

[6]. Gireesh Shrimali, Isaac Keslassy, Nick McKeown – "Designing Packet Buffers with Statistical Guarantees", IEEE 2004.