A Novel Hybrid Memory Architecture with Parallel DRAM for Fast Packet Buffers

Arthur Mutter
University of Stuttgart, Institute of Communication Networks and Computer Engineering (IKR)
Pfaffenwaldring 47, 70569 Stuttgart, Germany
mutter@ikr.uni-stuttgart.de

Abstract—**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 100 Gbps 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.**

I. Introduction

Internet routers and switches need buffers to store packets in case of congestion, e. g., when an output port is currently busy transmitting another packet. With ever increasing line rates packet buffers are becoming major bottlenecks in high speed routers and therefore significantly impact their performance. Packet buffers are typically located on the individual line cards of the router, each maintaining several first-in-first-out (FIFO) queues. E. g., a packet buffer on an input line card maintains *number of service classes* times *number of output ports* separate FIFO queues (virtual output queues, VOQ). The number of VOQs common today is in the order of hundreds to thousands [1]. All data of one queue belongs to one flow. In the following, the queues maintained by a packet buffer are named *flow queues*.

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.

A. Packet Buffer Requirements

A packet buffer has to satisfy requirements concerning three major properties: bandwidth, capacity, and random access time.

Bandwidth – The packet buffer has to both, store and retrieve packets at line rate. Therefore, the minimal necessary memory bandwidth is 2R, where R is the aggregate line rate of all ports on a line card. It is common for network nodes to segment packets into fixed size chunks to simplify memory management. 64 byte is a common choice as it is the first power of two able to hold a minimal Ethernet or IP packet. Nevertheless, in the worst-case the packet buffer must sustain a stream of 65 byte packets, which all consume two 64 byte chunks, leaving the second nearly empty. This is called the 65-byte-problem [1]. Because of this, memory bandwidth is usually over-provisioned by a factor of two, i. e. to $4R$. A high memory bandwidth can be achieved by using many memory chips in parallel.

Capacity – It is common to use a rule-of-thumb that says, for TCP to work well, the buffer should be dimensioned to $RTT \times R$, where RTT is the round trip time between active end hosts [2]. Assuming an RTT of 200 ms and a line rate of $R = 100\,$Gbps the buffer size is 2.5 Gbyte. While it has been challenged in [2] this rule-of-thumb is still widely used. The 65-byte-problem also affects capacity as it leads to fragmentation. Memory capacity can also be increased by the number of parallel operated memory chips.

Random Access Time – The random access time T (or T_{RC} in memory parlance) is the maximum time to write to or read from any memory location and limits the possible number of read/write operations. For example, as it takes 5.12 ns to receive a 64 byte packet at 100 Gbps, a stream of 64 byte packets requires an access time of $T = 2.56\,$ns since packets have to be written to and read from memory.

B. Available Memory Types

Two principle types of memories can be utilized to meet the mentioned requirements. First, static random access memory (SRAM) has a relatively short access time (e. g., $T = 3.33\,$ns for QDRII-SIO SRAM), a small capacity in the range of few MByte (e. g., 72 Mbit per QDRII-SIO SRAM chip) and a very high price per bit. Second, dynamic random access memory (DRAM) has a long access time (e. g., $T = 49\,$ns for a DDR3-1600 SDRAM), a large capacity (e. g., 2048 Mbit per DDR3 SDRAM chip) and a very low price per bit. This makes it very attractive to router manufacturers. DRAMs consist of several independent banks that can be accessed interleaved to reduce access time. However, this allows reducing T only on average as the unknown packet departure order may require unbalanced access to the individual banks.

C. Hybrid SRAM/DRAM Memory Architecture

To simultaneously meet the packet buffer requirements researchers propose a hybrid SRAM/DRAM (HSD) architec-

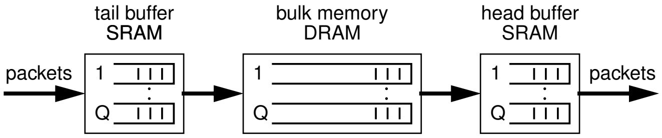


Fig. 1. Basic hybrid SRAM/DRAM architecture

ture [3]–[5]. SRAM meets the random access time and DRAM the capacity requirement, while both meet the bandwidth requirement. For this approach we assume that an SRAM is available that meets the required random access time. Fig. 1 shows the basic architecture. The tail buffer (SRAM) contains the tails of all flow queues, i. e. the packets that arrived last. The head buffer (SRAM) contains all the heads of the flow queues, i. e. the packets that are going to be requested soon. The bulk memory (DRAM) in the middle holds all the other packets in the queues. A memory management algorithm (MMA) controls the data transfers between SRAM and DRAM. Its challenge is to never let the tail buffer overflow and to make sure that packets are always present in the head buffer when needed. When the required SRAM for head and tail buffer are small enough they can be realized on the same chip as the MMA. This simplifies board layout and reduces cost.

When designing a hybrid SRAM/DRAM packet buffer like shown in Fig. 1 we have to choose between providing *statistical or deterministic bandwidth guarantee*. This is a major question, as this impacts the interface of the packet buffer.

When bandwidth guarantee is *statistical* packets are occasionally lost [6], [7]. This can be acceptable depending on the system. When bandwidth guarantee is *deterministic*, then the packet buffer always behaves like an SRAM under any packet arrival and departure pattern, i. e. it delivers the requested packets in-order after a constant read latency. This leads to a very narrow and simple packet buffer interface, simplifying the realization of subsequent components as they do not have to deal with variable latencies or not delivered packets. Our approach implements deterministic bandwidth guarantee.

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. When a requested packet is delivered immediately, then we have no read latency. Therefore the head SRAM has to buffer always enough packets as there is no time to access the DRAM. However, the necessary MMAs are not practical for a larger number of queues [1]. In contrast, if we accept a pipeline delay, i. e. a read latency > 0, less SRAM is required as the MMA starts requesting packets from DRAM only after it received a packet request. Therefore, in our approach we assume, that a pipeline delay, i. e. a read latency > 0, is acceptable.

D. Paper Targets and Result Summary

Our target is to design a hybrid SRAM/DRAM packet buffer architecture with deterministic bandwidth guarantee under any traffic condition while minimizing the required SRAM size. Requested packets are delivered in-order after a constant read latency.

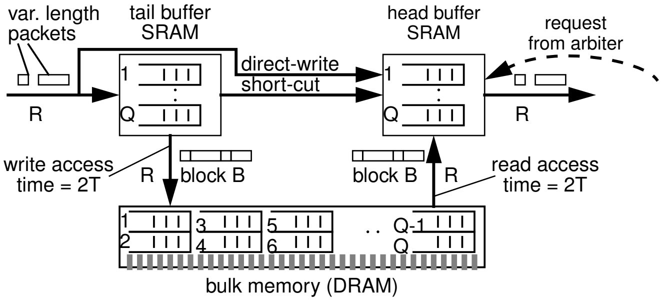


Fig. 2. Hybrid SRAM/DRAM architecture of [3]

In this paper we propose a novel hybrid architecture with parallel DRAMs. We prove that utilizing dynamic memory allocation the tail buffer size can be reduced significantly. E. g., for a 100 Gbps line card using DDR3 SDRAM we reduce tail buffer size by 47 % compared to [3], [5] and by 28 % compared to [4]. We prove also, that the read latency is equal to [3] and is not raised for any reason.

Further, our architecture aggregates packet data per-flow. This eliminates the 65-byte-problem and so we require no bandwidth over-provisioning and have no fragmentation in both, SRAM and DRAM.

We are currently implementing a prototype to work on an FPGA and present results for the tail part.

E. Paper Organization

The rest of this paper is organized as follows. In Section II, we introduce the state-of-the-art hybrid packet buffer architectures. In Section III, we introduce our proposed architecture including the MMAs used on head an tail side. In Section IV, we evaluate our architecture by proving the required head and tail buffer size as well as the read latency. Section V presents realization results for the tail part of the architecture and Section VI concludes the paper.

II. Related Work

Three basically different HSD packet buffer architectures have been proposed that provide deterministic bandwidth guarantee [3]–[5]. These will be introduced in the following.

[3] initially proposes a HSD packet buffer (cf. Fig. 2) and details it in [1]. It maintains Q FIFO flow queues and stores the heads and tails of all queues in the corresponding buffer. The remaining part is stored in bulk DRAM. The tail buffer aggregates variable length packets per flow to constant size blocks of $B = 2TR$ byte. As only full blocks are transferred to and from DRAM its bandwidth can be dimensioned to $2R$. The authors prove, that the required tail buffer size is $Q(B-1)$ byte. The smallest required head buffer is $Q(B-1)$ byte if a read latency of $Q(B-1)+1$ time slots is accepted, while they define a time slot as $\frac{1\text{ byte}}{R}$. The drawbacks of this architecture are the large tail and head buffer size and that it does not exploit bank interleaving although banks are available in all DRAMs.

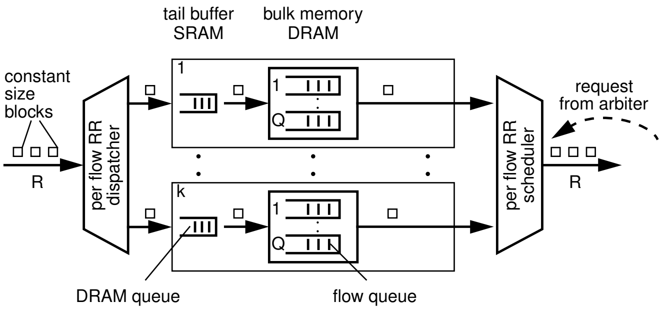


Fig. 3. Parallel Hybrid SRAM/DRAM Architecture of [5]

The architecture in [4] exploits bank interleaving to reduce tail and head buffer sizes. On the abstraction level discussed here, the architecture is equal to that in Fig. 2. The main idea is to decrease T by bank interleaving to reduce the block size B. As head and tail buffer sizes and read latency are proportional to B, they are decreased when B decreases.

The price they pay is additional effort for DRAM bank management and heavy fragmentation of the DRAM for special traffic patterns. In contrast to today's DRAMs, they assume DRAMs with several hundred banks, which do not exist now and in near future. According to their formula, they reduce the head and tail buffer size compared to [3] by each 26 % for a realistic system with $R = 100\,\text{Gbps}$ and DDR3 SDRAM (8 banks).

Finally, the authors in [5] present a parallel hybrid SRAM/DRAM architecture and detail it in [8]. They assume constant size blocks (packet segments) arriving to and departing from the packet buffer. Fig. 3 shows its architecture built from k parallel subsystems. Each subsystem consists of a tail buffer (SRAM) and a DRAM. Each tail buffer holds blocks destined to the corresponding DRAM. They do not use a head buffer as they solve in-order delivery algorithmically. A round robin MMA distributes blocks per-flow equally to the k subsystems. Each subsystem provides $\frac{1}{k}$ of the total required bandwidth. When providing deterministic bandwidth guarantee, the summed tail buffer size is equal to the size in [3]. The main advantage of this architecture is that it can be implemented distributed and that it requires no reorder buffer, i. e. head buffer.

However, there are several drawbacks. In-order delivery of requested packets is only guaranteed per-flow and read latency is variable. If this is acceptable depends on the subsequent component receiving these blocks. Further, the algorithms proposed to achieve per-flow in-order delivery require DRAM bandwidth over-provisioning. Finally, as they do not apply aggregation they will face the 65-byte-problem.

Our novel architecture combines the strengths of the introduced architectures and additionally decreases the tail buffer size significantly. We utilize a tail and a head buffer in combination with parallel DRAMs (or DRAM banks) for deterministic bandwidth guarantee and in-order delivery. As our MMA accesses DRAMs in a strict deterministic way, we can replace them by DRAM banks saving lots of data pins. With our MMA and dynamic memory allocation in the tail buffer our architecture requires a significantly smaller tail buffer size. Finally, by utilizing aggregation we require no memory bandwidth over-provisioning and eliminate fragmentation in both, SRAM and DRAM.

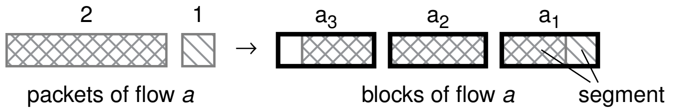


Fig. 5. Packet segmentation and subsequent aggregation to blocks for one flow

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

A. Architecture

Our proposed Semi Parallel Hybrid SRAM/DRAM (SPHSD) architecture is depicted in Fig. 4. 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 $b = 2TR/k$.

Definition 1. *A time slot is the time to receive a block of* b *byte at line rate* R.

From this follows that $2T = k$ time slots. In k time slots all DRAMs together write k blocks and read k blocks.

The tail side aggregates packet data per-flow and writes it in granularity of blocks into the DRAMs. In Fig. 4 from left to right, the segmenter receives variable length packets, divides them into segments and forwards them to the dispatcher. The size of a segment is always chosen in a way that a block gets full. In the example in Fig. 5 the first packet is segmented into one segment of same size, while the second packet is segmented into three segments. The dispatcher distributes segments over k DRAM queues (one for each DRAM, cf. Fig. 4). For each flow, each time after one DRAM queue received segments with a total size of b byte, the dispatcher chooses the next DRAM queue. E. g., in Fig. 5 the first two segments are dispatched to the same DRAM queue so they can be aggregated to block a_1. a_i denotes the i^{th} block of flow a.

The tail buffer maintains the k DRAM queues. A DRAM queue serves two purposes. First, it holds data of non-full blocks during aggregation. Second, it holds full blocks in case the corresponding DRAM is temporarily overbooked. The tail transferor transfers full blocks from the DRAM queues to the DRAMs.

The head side receives requests from an external arbiter, retrieves the data from DRAM and delivers the packets in-order with constant read latency. In Fig. 4 from right to left,

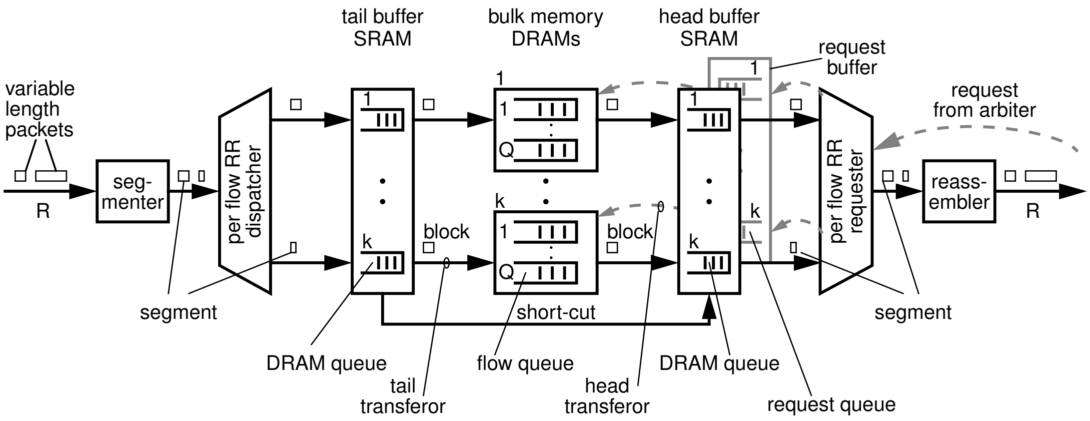


Fig. 4. Semi Parallel Hybrid SRAM/DRAM (SPHSD) Packet Buffer Architecture

an external arbiter sends packet requests to the requester. The requester determines the DRAMs that contain a block with a segment of the packet. If the corresponding blocks were not requested yet, block requests for the corresponding DRAMs are generated and forwarded to the request buffer. The request buffer maintains one request queue per DRAM. Its purpose is to hold the requests in case the corresponding DRAM is overbooked. The head transferor sends the requests from the request buffer to DRAM and delivers the received block to the head buffer.

The head buffer maintains the k DRAM queues, i. e. one per DRAM. A DRAM queue in the head buffer serves two purposes. First, it stores data until it can be delivered because the head buffer serves as a reorder-buffer. Second, it holds segments that were not yet requested by the arbiter. After the constant read latency the per-flow RR requester reads the requested segments from the head buffer and forwards them to the reassembler which reassembles the original packets.

A side effect of aggregation is that non-full blocks are never written to DRAM. But for flows with light traffic it may happen, that such a block is requested by the head part but is not available in the DRAM. A short-cut path from tail to head buffer solves the problem.

Switch fabrics usually operate with fixed size cells. When the packet buffer serves a switch fabric and the block size is dimensioned equal to the cell size then the reassembler can be omitted.

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 to those in [3].

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 64 byte. At a given line rate this determines the upper limit of k, e. g., for $R = 100\,\text{Gbps}$ and $T = 49\,\text{ns}$, $\lceil k \rceil = 20$. 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

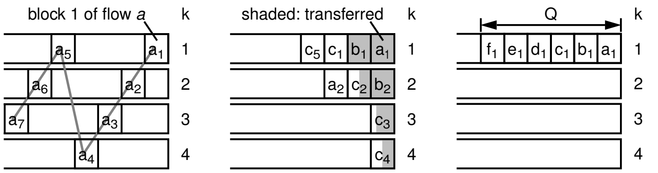
This section describes the round robin MMA utilized in the tail part. The MMA consist of two components: the *per-flow round robin dispatcher* and the *tail transferor*. Dispatcher and transferor work independently.

Per-flow round robin dispatcher – The dispatcher distributes packet data of each flow block-wise in a round robin manner over all k DRAMs, i. e. every k^{th} block of each flow is dispatched to the same DRAM. As the dispatcher actually dispatches segments, for each flow it chooses the next DRAM if the current already received segments with a total size of b byte. The dispatcher completes writing b byte into the tail buffer in one time slot. To simplify the description we will say in the following, that the dispatcher dispatches blocks. Further, we will call the arrival of b byte for one flow *block arrival*.

Dispatching per-flow allows us to write/read blocks of each flow to/from DRAM with line rate. This is mandatory in order to provide deterministic bandwidth guarantee. The architectures in [4], [5] also use per-flow round robin dispatching. Fig. 6(a) shows the consecutive dispatching of blocks from a single flow to the DRAMs.

Tail transferor – The task of the tail transferor is trivial. It transfers full blocks from tail buffer to DRAM as fast as possible in order to keep tail buffer occupation low. Each DRAM completes writing a block in $2T = k$ time slots. The transferor performs parallel writes to all k DRAMs if all DRAM queues have full blocks. With full blocks in each DRAM queue the tail buffer fill level cannot raise, as per time slot, b byte leave and at most b byte arrive.

Fig. 6(b) shows the state of the DRAM queues after the arrival of blocks from several flows. In the example we assume $k = 4$ DRAMs and $Q = 6$ flows named a to f. To simplify the example we assume that the received packets size is an integer multiple of the block size. The blocks arrive at the tail buffer one by one in the following order: a_1, b_1, b_2, c_1, c_2, c_3, c_4, c_5, a_2. In DRAM queues 1 and 2 blocks accumulate, as due to the round robin dispatching these DRAMs are temporarily



(a) RR principle (b) example with three flows (c) flows synchronized to RR scheme

Fig. 6. RR-MMA: Status of the DRAM queues in the tail buffer

overbooked. The grey shaded area marks blocks that have been in the queue before, but were transferred to DRAM meanwhile. E. g., after block c_4 two more blocks arrived. During these two time slots the block c_4 could be transferred to DRAM by half, as it takes $k = 4$ time slots to transfer it completely.

Lemma 1. *No more than Q blocks can accumulate per DRAM queue for $P_{min} \approx 0 \ll b$, where P_{min} is the minimal packet size in byte.*

Proof: Assume all Q flows have a nearly full block in tail buffer. Then each flow receives a minimal size packet completing aggregation of all Q blocks. If all flows are synchronized to the round robin sequence, then all Q blocks are in the same DRAM queue. Fig. 6(c) shows the case after the sequential completion of the blocks a_1 to f_1.

The per-flow round robin dispatcher assigns every k^{th} block of a flow to the same DRAM queue. However, in k time slots the transferor can remove one block from every DRAM queue and write it to DRAM. So no more than Q blocks can accumulate. ■

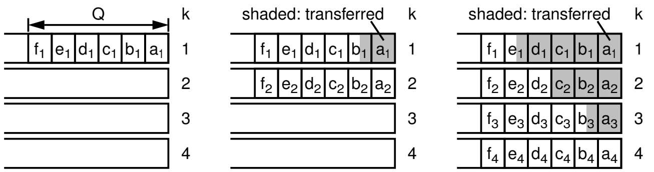
D. Head Round Robin Memory Management Algorithm

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.

IV. 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.



(a) $t = Q$ time slots (b) $t = 2Q$ time slots (c) $t = kQ$ time slots

Fig. 7. Arrival of traffic pattern *P1*: tail buffer status

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 required in [3] and [5]. 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*

$$Q\frac{(k+1)}{2}$$

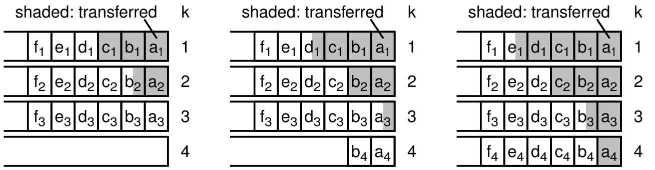
Proof: We assume that packets arrive at the packet buffer continuously with full line rate R. This represents the worst-case if we want to show that the buffer size is bounded. The proof consists of four steps leading to Lemma 2, 3, 4 and 5.

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 defines 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.

Definition 2. *Traffic pattern P1 has the following properties:*

- *Q blocks accumulate to one DRAM queue according to Lemma 1.*
- *This consecutively happens $\geq k$ times*

Fig. 7 gives an example for *P1* assuming $Q = 6$ and $k = 4$. Starting from an empty tail buffer at $t = 0$ after $Q - \varepsilon$ time slots no block is fully aggregated yet. However, at $t = Q$ time slots all Q blocks are full in the first DRAM queue (see Fig. 7(a)). Until now all DRAMs are idle. At $t = 2Q$ time slots Q blocks get full in DRAM queue 2 (see Fig. 7(b)). Up to now DRAM queue 1 transferred $Q/k = 1.5$ blocks to DRAM 1. We visualize transferred blocks by shading them.



(a) $t = 3Q$ time slots (b) $t = 3Q + 2$ time slots (c) $t = kQ$ time slots

Fig. 8. Arrival of traffic pattern *P2*: tail buffer status

Up to now all other DRAMs are still idle. Fig. 7(c) shows tail buffer status at $t = kQ$ time slots.

Lemma 2. *With P1, starting from an empty tail buffer the maximal required tail buffer size is* $Q \cdot \frac{(k+1)}{2}$ *blocks.*

Proof: The non-shaded area in the Fig. 7(c) represents the required tail buffer size. Based on this we can calculate the buffer size for arbitrary k and Q.

Accumulation of Q blocks in each of the DRAM queues takes Q time slots. The transferor removes in Q time slots Q/k blocks from any DRAM queue with full blocks. At $t = kQ$ DRAM queue i has $Q\frac{i}{k}$ full blocks, i. e. DRAM queue $k-1$ has $Q\frac{k-1}{k}$ full blocks. Summing up the blocks of all DRAM queues gives us

$$S_{P1} = Q \cdot \sum_{i=1}^{k} \frac{i}{k} = Q \cdot \frac{(k+1)}{2}$$

At any time $t = xQ$ time slots, with $x > k$ the buffers size is equal to S_{P1} because the fill levels just rotate through the DRAM queues. E. g., starting from Fig. 7(c), at $t = (k+1)Q$ DRAM queue 1 will have the fill level of DRAM queue k, 2 that of 1 and so on. As all DRAM queues have full blocks all the time, data leaves from tail buffer with full line rate and leaves space for arriving segments. ■

Now we show that starting from an empty tail buffer any traffic pattern different from *P1* leads to a lower bound for the tail buffer size.

Definition 3. *Traffic pattern P2 includes all possible traffic patterns except P1.*

Lemma 3. *With P2, starting from an empty tail buffer the maximal required tail buffer size is always less than with P1.*

Proof: Starting from an empty tail buffer, with *P2* some blocks get full earlier than with *P1*. Consequently, the transferor also starts writing blocks to DRAM earlier. This always leads to a smaller maximal required tail buffer size than *P1*.

Fig. 8 gives an example for *P2*. We assume $Q = 6$ and $k = 4$. Starting from an empty tail buffer, up to $t = 3Q$ time slots in this example there is no difference to *P1* (cf. Fig. 8(a)). At $t = 3Q + 2$ time slots already 2 blocks get full in DRAM queue 4 (cf. Fig. 8(b)). At $t = kQ$ time slots 4 further blocks get full in DRAM queue 4 (cf. Fig. 8(c)). The main difference to *P1* is, that the transferor could start writing blocks to DRAM 4 earlier. This leads to a smaller total tail buffer size compared to *P1*. ■

Now we consider starting from a *non-empty tail buffer*.

Lemma 4. *Starting from any valid non-empty tail buffer status P1 does not raise the upper bound from Lemma 2.*

Proof: The proof requires two definitions.

Definition 4. *System A, is a system that receives from the beginning only P1.* F_{Ai} *denotes the length of DRAM queue* i *in its tail buffer. Fig. 7 depicts the tail buffer status of such a system.*

Considering System A, independent how long *P1* is received, the required buffer size is bounded (cf. Lemma 2).

Definition 5. *System B, is a system that receives from the beginning only P2. Then packet arrivals synchronize to the round robin scheme so that it can receive P1 in the following.* F_{Bi} *denotes the length of DRAM queue* i *in its tail buffer. Fig. 8 depicts the tail buffer status of such a system.*

We compare now the tail buffer status of systems A and B. We consider the status of system A depicted in Fig. 7(c) and the status of system B depicted in Fig. 8(c). System B is at the point in time before it starts receiving *P1*. Comparison shows that $F_{Ai} \geq F_{Bi}, \forall i \in \{1, 2, \ldots, k\}$. If system B starts now receiving *P1* then it cannot raise the upper bound of Lemma 2 as it has a better starting position than system A. ■

Lemma 5. *Starting from any valid non-empty tail buffer status introduced by P1, P2 does not raise the upper bound from Lemma 2.*

Proof: We start from a system receiving *P1* (cf. Fig. 7(c)). Receiving packets at full line rate R, the required tail buffer size of this system cannot decrease as the DRAM bandwidth available for writing is also R. As long as each DRAM queue has full blocks the required tail buffer size can also not increase.

We show now, that at full line rate none of the DRAM queues can run empty of full blocks and therefore the required tail buffer size cannot increase. The tail buffer status in Fig. 7(c) is the start point. To see if a DRAM queue i can run empty of full blocks we need a traffic pattern that maximizes the waiting time of DRAM queue i for full blocks. *P1* has this property. E. g., the maximum time for DRAM queue 1 until a new block gets full is Q time slots. The Q/k blocks in DRAM queue 1 are exactly enough not running empty in these Q time slots. From this we conclude, that here with *P1* no DRAM queue can run empty. Consequently, with *P2* also no DRAM queue can run empty, because with *P2* blocks get full earlier compared to *P1*. We conclude, that *P2* does not raise the upper bound from Lemma 2. ■

From Lemma 2, 3, 4 and 5 it is clear that independent of the received traffic pattern (i. e. only *P1*, only *P2*, first *P1* then *P2* or first *P2* then *P1*) the upper bound of the required tail buffer size is that given in Lemma 2. According to the definitions of *P1* and *P2* these cover together all existing traffic patterns. *P1* introduces the upper bound in Lemma 2 and is therefore the worst case traffic pattern. ■

We compare now our required tail buffer size to the size required in other systems. In realistic systems the value of k is large, e. g., $k = 20$ for a system with $R = 100\,\text{Gbps}$ and DDR3 SDRAM (cf. Section III-B). This leads to a buffer size of $S_1 = Q\frac{20+1}{2}$ blocks. In [3] the tail buffer size is $S_2 = Q(B-1)$ byte. With $B = kb$, $S_2 \approx Qkb$ byte $= Qk$ blocks. The tail buffer size in [5] is equal to that in [3]. Concluding, our SPHSD architecture reduces the tail buffer size by 47 % compared to [3], [5].

For the given example, according to the formula in [4], the authors reduce the tail buffer size by 26 % compared to [3], [5]. Concluding, our SPHSD architecture reduces the tail buffer size by 28 % compared to [4].

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.

Lemma 6. *The tail part introduces no latency.*

Proof: The short-cut path can be used to transfer any block from tail to head buffer. So no latency is introduced. ■

Lemma 7. *The head part introduces a maximum latency of Qk time slots.*

Proof: A DRAM queue in the tail part can accumulate up to Q blocks (cf. Lemma 1). Due to symmetry, a request queue can accumulate the same amount of block requests. The head transferor can read from one DRAM one block every k time slots. Therefore, the maximum latency for a request is Qk time slots. ■

From Lemma 6 and 7 we conclude, that the read latency of the packet buffer is Qk time slots. ■

This is nearly equal to the read latency in [3] when the architecture in [3] uses the minimal possible head buffer size (cf. Section II). The difference originates from the different assumptions for P_{min}.

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 (cf. Lemma 7) $(Q-1)k$ blocks are completely received from DRAM and k blocks are partly received.

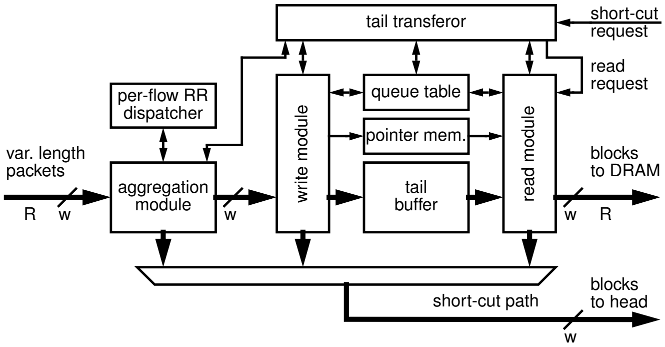


Fig. 9. Prototype architecture of tail part

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. ■

Compared to [3], where the head buffer size is $Q(B-1)$ byte, our SPHSD architecture requires a head buffer that is $\frac{1}{k}$ larger. In a realistic system k is around 20 (cf. Section III-B) and so the difference to [3] is quite small. However, therefore the architecture in [3] requires a direct write path from the packet buffer input to the head buffer (cf. Fig. 2). This increases the required bandwidth of their head buffer. The same holds also compared to [4] with the difference, that the head buffer size in [4] is 26 % below that in [3] (cf. Section II).

We believe that due to the symmetric property of our SPHSD system, a more sophisticated head transferor that process requests as late at possible (cf. Section III-D) the head buffer size can be reduced by the same amount (approx. 50 %) like the tail buffer. The corresponding proof is ongoing work.

V. Realization

We are currently implementing a packet buffer prototype in VHDL to run on an FPGA. In the following we deliver results for the tail part. Fig. 9 shows its architecture. Realization allows us quantifying the price for the available features: (1) aggregation, (2) dynamic memory allocation, and (3) short-cut. These features are also available in other architectures [3], [4], but their implementation complexity is not evaluated in this detail by the authors.

The prototype has a data bus width of $w = 512\,\text{bit}$, while the design can be synthesized for any $w = 2^n$. We implement dynamic memory allocation by utilizing linked lists and organize the tail buffer with a granularity of one bus word. The block size can be any integer multiple of one word. To minimize tail buffer size we set the block size to the smallest possible value, i. e. one bus word. The tail buffer and all other memories are realized with on-chip dual-port SRAM that allows one read and one write access per clock cycle.

In Fig. 9 from left to right the aggregation module segments the packets, aggregates them to full words and retrieves the target DRAM queue for each block from the dispatcher. It forwards full words to the write module which writes them to the tail buffer. Further, the write module accesses pointer memory and queue table to update the data structures for the linked lists and the DRAM queue status. The transferor checks periodically in a round robin manner the status of each DRAM queue. If in a DRAM queue at least one block is full, it generates a read request for the read module. The read module reads the block from tail buffer and forwards it towards DRAMs. Further, the read module accesses pointer memory and queue table to update the data structures for the linked lists and the DRAM queue status.

If the head part wants to read a block that is not in DRAM, it generates a short-cut request for the tail transferor. The transferor checks in parallel the blocks location, which can be anywhere in the tail part as this depends on the incoming traffic pattern. Then it triggers the corresponding module to forward the block via the short-cut path.

In the following we discuss the complexity and resource consumption added by the individual features.

Aggregation – We perform aggregation to full words in a separate module to keep the operating frequency of the tail buffer low. Thereby, the tail buffer receives only full words and has to perform only one write per incoming bus word to the tail part – instead otherwise two due to segmentation. The price for aggregation is mainly the additional required memory of Q words.

Dynamic Memory Allocation – This adds complexity to the read and write module which have to maintain the linked lists in the pointer memory. To keep the pointer memory neglectable in size we organize the tail buffer in a granularity of one word. As the tail buffer contains only full words, it is not fragmented at all. However, Theorem 2 assumed an infinite fine grained organization. The unavoidable memory fragmentation occurs in the aggregation module.

Short-cut – All components have to support the short-cut. This has major impact on two modules. First, the aggregation module has to support one additional read to its aggregation memory per clock cycle. We solve this by doubling this memory instead of doubling its clock frequency. Second, instead of k we maintain Qk queues in the tail buffer, i. e. Q queues for each logical DRAM queue. A short-cut always requests the oldest block of a flow from a given DRAM queue. Now the read module can easily retrieve the requested block.

Concluding, the complexity and resources for these features are small compared to the delivered benefits, e. g., no required bandwidth over-provisioning for both, SRAM and DRAM.

VI. Conclusion

High speed routers and switches utilize hybrid SRAM/ DRAM packet buffers to meet both, speed and capacity requirements.

In this paper, we propose a novel hybrid architecture providing deterministic bandwidth guarantee under any traffic condition as well as constant read latency. Our architecture utilizes a tail and a head buffer (SRAM) in combination with parallel DRAMs (or DRAM banks). It has the two following main characteristics.

First, it combines the strengths of other architectures in a single architecture. These are namely per-flow aggregation and banking. Aggregation eliminates the 65-byte-problem, so we require no bandwidth over-provisioning and have no fragmentation in both, SRAM and DRAM. To enable banking we replace the individual DRAMs in our architecture by DRAM banks. Thereby we still provide deterministic bandwidth guarantee as our memory management algorithm accesses these banks in strict deterministic order. This saves many I/O pins and still allows usage of cheap commodity DRAM like DDR3 SDRAM.

Second, it reduces the tail buffer size (SRAM) significantly compared to other architectures, e. g., for a 100 Gbps system with DDR3 SDRAM we decrease tail buffer size by 47 % compared to [3], [5] and by 28 % compared to [4]. We achieve this with help of the parallel DRAMs (banks) and dynamic memory allocation in the tail buffer. The former reduces the data block size transferred from and to DRAM and so data blocks are earlier ready to be written to DRAM. The latter allows us to utilize the tail buffer efficiently.

Our head buffer size is marginally above the size required in [3] if we utilize a trivial memory management algorithm. However, due to the symmetry of our architecture, we believe, that with a more sophisticated memory management algorithm in the head part we can decrease the head buffer size by the same amount as the tail buffer, i. e. by approx. 50 %. The formal proof is ongoing work.

We prove also, that the read latency is equal to [3] and so is not raised by any of the characteristics mentioned above. Finally, we present the FPGA prototype architecture for the tail part.

VII. Acknowledgments

This work was funded by Alcatel Lucent Bell Labs Germany, Stuttgart.

References

[1] S. Iyer, R. Kompella, and N. McKeown, “Designing packet buffers for router linecards,” *Networking, IEEE/ACM Transactions on*, vol. 16, no. 3, pp. 705–717, June 2008.

[2] G. Appenzeller, I. Keslassy, and N. McKeown, “Sizing router buffers,” *SIGCOMM Comput. Commun. Rev.*, vol. 34, no. 4, pp. 281–292, 2004.

[3] S. Iyer, R. R. Kompella, and N. Mckeown, “Analysis of a memory architecture for fast packet buffers,” in *IEEE High Performance Switching and Routing*, 2001, pp. 368–373.

[4] J. García, J. Corbal, L. Cerdà, and M. Valero, “Design and implementation of high-performance memory systems for future packet buffers,” in *MICRO 36*. Washington, DC, USA: IEEE Computer Society, 2003, p. 373.

[5] F. Wang and M. Hamdi, “Scalable router memory architecture based on interleaved dram,” in *High Performance Switching and Routing, 2006 Workshop on*, 0-0 2006, pp. 6 pp.–.

[6] S. Kumar, P. Crowley, and J. Turner, “Design of randomized multichannel packet storage for high performance routers,” in *HOTI ’05*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 100–106.

[7] G. Shrimali and N. McKeown, “Building packet buffers using interleaved memories,” in *High Performance Switching and Routing, 2005. HPSR. 2005 Workshop on*, May 2005, pp. 1–5.

[8] F. Wang, M. Hamdi, and J. K. Muppala, “Using parallel dram to scale router buffers,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 20, pp. 710–724, 2009.