USENIX

THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

Themis: Scheduling-Aware Buffer Management for HBM-Based Hybrid Buffers

Zhiyu Zhang, Minkun Xue, Kan Yu, Ruyi Yao, Shili Chen, and Hao Mei, *Fudan University;* Zixuan Chen, *China Telecom;* Weiyi Chen, Yibo Fan, and Yang Xu, *Fudan University*

https://www.usenix.org/conference/nsdi26/presentation/zhang-zhiyu

This paper is included in the Proceedings of the 23rd USENIX Symposium on Networked Systems Design and Implementation.

May 4–6, 2026 • Renton, WA, USA

ISBN 978-1-939133-54-0

Open access to the Proceedings of the 23rd USENIX Symposium on Networked Systems Design and Implementation is sponsored by

جامعة الملك عبدالله للعلوم والتقنية

King Abdullah University of Science and Technology

Themis: Scheduling-Aware Buffer Management for HBM-Based Hybrid Buffers

Zhiyu Zhang
Fudan University

Minkun Xue
Fudan University

Kan Yu
Fudan University

Ruyi Yao
Fudan University

Shili Chen
Fudan University

Hao Mei
Fudan University

Zixuan Chen
China Telecom

Weiyi Chen
Fudan University

Yibo Fan
Fudan University

Yang Xu*
Fudan University

Abstract

Packet buffers are critical for absorbing congestion and sustaining throughput in high-speed routers. As link rates escalate, on-chip SRAM alone can no longer provide sufficient capacity. To address this, modern routers widely adopt hybrid buffer architectures that augment limited on-chip SRAM with large off-chip DRAM. Despite this architectural promise, existing hybrid Buffer Management (BM) schemes severely undermine router performance. They simply redirect packets that would otherwise be dropped into DRAM, which leads to priority inversion and head-of-line blocking: as packets buffered in DRAM age into the highest-priority packets, SRAM must stall until they are retrieved, wasting bandwidth and degrading throughput. Worse still, existing BM schemes ignore DRAM's access characteristics, further constraining its limited bandwidth and reducing overall performance dramatically.

We present Themis, a hybrid buffer management scheme that fully exploits SRAM's high bandwidth and DRAM's large capacity. Its core principle is scheduling-aware packet placement, which ensures packets with the earliest departure time are preferentially stored in SRAM to maximize its bandwidth utilization. To achieve this, Themis proactively migrates buffered packets between SRAM and DRAM, reserving SRAM space for imminent, high-priority traffic. Themis is compatible with diverse scheduling algorithms and supports dynamic changes to the scheduling policy. It also organizes DRAM storage according to scheduling order, mapping consecutively departing packets to contiguous addresses. This unlocks effective off-chip bandwidth and accelerates packet migration. Themis is hardware-friendly, and its FPGA and ASIC prototypes achieve low overhead and high frequency. Large-scale simulations show that Themis improves end-to-end performance by up to 2.6×.

*Corresponding author: xuy@fudan.edu.cn

1 Introduction

In recent years, network bandwidth has surged to 100 Gbps and beyond [2], with commercial chips now supporting 400 Gbps ports [25]. Accordingly, packet buffers must provide sufficient speed and capacity to keep up with line rates. On-chip SRAM is fast but limited in capacity, expensive, and power-hungry, whereas off-chip DRAM provides large capacity but suffers from high latency. No single storage medium can satisfy all the requirements of packet buffers. To bridge this gap, modern routers adopt hybrid buffer architectures that combine SRAM and DRAM to leverage the strengths of both. Hybrid buffering has become the dominant trend in high-performance routers for campus and wide-area networks, as seen in Broadcom Jericho [15], Cisco Silicon One [18], and Huawei CloudEngine [24].

For hybrid buffers, several Buffer Management (BM) schemes have been proposed [27, 28, 31–34, 44, 49]. Lately, mainstream solutions—those adopted in chip designs [15, 24]—primarily rely on thresholds to decide whether packets are placed in SRAM or DRAM, differing only in how these thresholds are calculated. Their central idea is to use DRAM as overflow space for packets that SRAM would otherwise drop. While this philosophy is straightforward and simple to implement, it often fails to realize the performance advantage of hybrid buffers. Specifically, high-priority packets may end up in off-chip DRAM, thus constraining the egress bandwidth, while low-priority packets remain in on-chip SRAM, occupying scarce on-chip resources that cannot be released in time.

Such a *priority inversion* problem leads us to reflect: BM essentially treats all flows uniformly, penalizing long queues and diverting their subsequent packets to DRAM. However, packet scheduling pursues the opposite goal: differentiated service to ensure Quality of Service (QoS), as evidenced by the wide variety of scheduling algorithms [8, 14, 19, 42]. This brings a fundamental contradiction: **BM decides admission, scheduling decides departure, whereas fairness-based admission cannot adequately support differentiation-based**

departure. As a result, flows that are prioritized by the scheduler receive no preference at admission, and their additional packets are forced to be served from a lower-priority storage medium. Yet these high-priority flows are often the ones most sensitive to delay.

While some recent approaches [4] recognize this issue and attempt to approximate scheduling awareness by allocating thresholds based on drain rates, these are reactive and limited. In dynamic conditions, they often fail to protect high-priority flows at the right moment. At the same time, fundamentally solving this problem requires considering another limitation: **BM is an online algorithm that must decide a packet's storage location immediately upon arrival. However, the impact of this decision persists until the packet is transmitted**. This means that a high-priority packet admitted to SRAM may occupy space needed later by an even higher-priority packet, forcing the latter into DRAM. Since the future cannot be predicted, such misplacements appear unavoidable, ultimately leading to the *head-of-line blocking* problem.

At the execution level, another problem in hybrid BM becomes apparent: **DRAM, as the bandwidth bottleneck of the entire hybrid buffer, exhibits a strong dependence of its performance on the access pattern**. On the latest DRAM technology, High-bandwidth Memory (HBM) [1], this dependence is even more pronounced—without favorable access patterns, the achievable bandwidth can fall far short of its nominal bandwidth. However, in packet buffer scenarios, consecutively arriving packets are not transmitted in sequence, making it difficult to sustain efficient access patterns.

To address the three critical problems outlined above and unlock the true advantages of hybrid buffers—offering both low latency and large capacity—we present Themis, a scheduling-aware buffer management design for hybrid buffers. Themis leverages the notion of *rank* in the programmable scheduling model PIFO [43] to elegantly unify the semantics of BM and scheduling. In Themis, every buffer management action is guided by packet rank, ensuring that earlier-departing packets are stored in low-latency storage. Moreover, Themis actively adjusts the placement of already buffered packets, enabling proactive correction and avoiding the traditional predicament where blocking is resolved only by natural draining. At the same time, by exploiting rank, Themis groups consecutively departing packets into contiguous memory locations and performs HBM accesses in batch mode, maximizing effective HBM bandwidth.

More importantly, Themis is not just a theoretical proposal. Beyond demonstrating strong theoretical performance, Themis delivers a complete and high-performance hardware implementation. Specifically, it analyzes the operations already supported in the Traffic Manager (TM) and the new operations required by Themis, identifies their overlap, and reuses existing hardware modules to realize the full functionality of Themis at minimal additional cost. We have implemented the Themis-enabled TM framework on FPGA and conducted testbed experiments on this prototype. In addition, large-scale network simulations have been performed in ns-3. Results show that Themis achieves significant performance gains.

We summarize the contributions of this paper as follows:

- We identify three key problems of existing hybrid buffer management: priority inversion, head-of-line blocking, and inefficient HBM bandwidth utilization.
- We propose Themis, a scheduling-aware buffer management design that unifies BM and scheduling via packet rank, corrects misplacements proactively, and maximizes HBM bandwidth through batch access.
- We deliver a high-performance hardware implementation of Themis at minimal additional cost by introducing the BM-SCH Unified Model in the TM, which eliminates unnecessary memory access overhead.
- Our extensive experiments demonstrate that at the micro level, Themis achieves up to 30% fewer packet drops, 2.1× higher on-chip hit rates for packet dequeues, 43% greater egress utilization, and 46% higher HBM throughput; at the macro level, Themis reduces Flow Completion Time (FCT) by a factor of 2.6. The project is available at https://github.com/Themis-Source/Themis.

2 Background

2.1 Buffer Management

A hybrid buffer consists of on-chip SRAM and off-chip DRAM. On-chip SRAM has limited capacity and cannot hold many packets, while off-chip DRAM has slower access speed and cannot keep up with line rate. Therefore, an appropriate BM policy is required to decide which packets should stay on-chip and which should be placed off-chip. Existing hybrid BM schemes can be broadly categorized into two approaches.

The first approach mandates that all packet dequeues occur from SRAM in order to sustain full link utilization. It mainly has two implementations. The first implementation places the head-of-line packet from every queue into SRAM [27, 28]. As a result, the required SRAM size scales with the number of queues. In systems with a large number of queues, this implementation quickly becomes impractical: for example, in Broadcom's BCM88800 [15], which supports up to 128K VOQs, it would require nearly 5 GB of SRAM—far beyond the capacity of modern silicon. The second implementation directly dequeues packets from each flow according to a fixed pattern (e.g., round-robin) [34, 44].

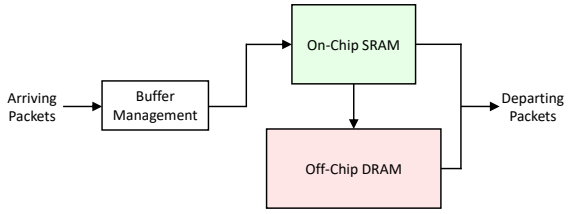


Figure 1: Spillover-Only Hybrid Buffer Architecture.

While this implementation ensures that all dequeued packets come from SRAM with reduced SRAM demand, it completely ignores the scheduling algorithm, and the intended QoS guarantees cannot be achieved. Overall, the first approach has low feasibility in modern router designs.

The second approach, as illustrated in Figure 1, designates on-chip SRAM as the primary buffer and treats off-chip DRAM purely as an overflow region [15, 24, 32]. Packets first enter SRAM, and when the length of a queue in SRAM exceeds a threshold, all subsequent packets arriving at that queue are directly spilled over into DRAM. Packets identified for spillover make only a brief stop in SRAM before being immediately directed to DRAM, and in the following discussion these packets are also described as *packets directed to DRAM* or *packets written to DRAM*. We term this class of methods **spillover-only hybrid BM**. Owing to its design simplicity, predictable behavior, and feasibility for hardware implementation, this spillover-only approach has been adopted in commercial switch chips from leading vendors such as Broadcom [15] and Huawei [24]. Our subsequent analysis will primarily focus on this approach.

2.2 HBM

Modern high-performance network systems, including programmable switches and routers, are fundamentally constrained by memory bandwidth. As port speeds reach 400 Gbps and beyond, packet processing pipelines must buffer and access packets at line rate, placing extreme demands on memory subsystems. Conventional DDR [30], though cost-effective, often becomes a bandwidth bottleneck due to limited pin count and modest data transfer rates.

HBM [1] has emerged as a key enabling technology to overcome these limitations. HBM is a 3D-stacked DRAM standard, integrating multiple memory dies vertically via through-silicon vias [39] (TSVs). Its wide interface—typically 1024-bit or 2048-bit per stack—delivers much higher bandwidth than DDR while operating at lower frequencies, improving energy efficiency [29, 46]. In networking, HBM has been adopted in cutting-edge routing chips as a high-performance off-chip packet buffer [15].

3 Motivation

In this section, we expose the limitations of existing hybrid BM schemes, analyze their root causes, and clarify the key challenges in addressing them. Our analysis is presented from two perspectives: the *policy perspective*, which concerns the correctness and suitability of BM decisions, and the *execution perspective*, which concerns the efficiency of enforcing these decisions on SRAM/DRAM in terms of read and write performance.

3.1 Limitations of Existing Schemes: Policy Perspective

As a scheme that has been adopted in commercial router chips, spillover-only hybrid BM is simple and intuitive. However, it suffers from two critical issues: priority inversion and head-of-line blocking. We demonstrate these problems in §3.1.1 and §3.1.2, respectively.

3.1.1 Inter-Flow Priority Inversion

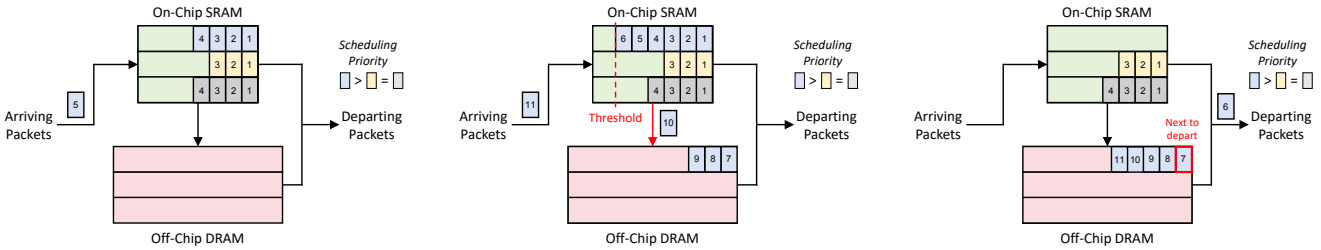
Consider the scenario shown in Figure 2 (a): for a single output port, there are three queues corresponding to blue, yellow, and gray flows. The scheduling algorithm is defined as follows: the blue flow has strict priority over the yellow and gray flows, while the yellow and gray flows equally share the remaining bandwidth. For clarity, we assume all packets have the same size.

Starting from an empty buffer, packets from the three flows arrive in sequence, with the number on each packet indicating its order within the flow. As shown in Figure 2 (b), because the blue flow has more arrivals, it is the first to trigger the threshold under spillover-only hybrid BM. Although different implementations may calculate thresholds differently, they share two common properties: (1) a flow with a larger backlog triggers the threshold earlier, and (2) the threshold reserves a portion of the on-chip buffer to accommodate potential new flows that have not yet arrived. Consequently, subsequent packets of the blue flow are written into off-chip DRAM, while packets of the yellow and gray flows continue to be admitted into on-chip SRAM.

At this point, the scheduler, following the predefined scheduling algorithm, always serves the blue flow first when dequeuing.[1] Thus, as shown in Figure 2 (c), once the blue packets in on-chip SRAM are transmitted, the scheduler continues to fetch blue packets from off-chip DRAM. During this period, the port's effective throughput is limited by off-chip DRAM read bandwidth, which cannot keep up with the line rate of the output port. However, SRAM in fact has sufficient capacity to hold these packets.

We refer to this phenomenon as the **priority inversion problem**. Its immediate cause lies in BM making incorrect

[1] Packet arrival and departure occur simultaneously in real systems. During congestion buildup, the packet arrival rate exceeds the departure rate, leading to a positive queue growth; whereas during queue draining, the departure rate exceeds the arrival rate, leading to a negative queue growth. In the examples of this paper, for clarity, we describe arrival and departure separately. This treatment does not affect the final conclusion.



(a) Initial Setup (b) High-Priority Packets Spill to DRAM (c) Priority Inversion

Figure 2: Inter-Flow Priority Inversion.

decisions about packet placement. In the example, packets 7–11 of the blue flow are mistakenly placed in off-chip DRAM. More fundamentally, regardless of the specific threshold algorithm used, spillover-only hybrid BM essentially assigns and adjusts thresholds based on the observed states of flows, in order to prevent any single flow from occupying excessive buffer space. By "observed states", we mean that flows are not inherently different; only their runtime behavior leads to the assignment of different thresholds.

However, from the perspective of the network as a whole, flows often need to be differentiated a priori, so as to meet their respective QoS requirements and maximize network resource utilization. Such differentiated requirements are common in practice. For example, in multi-tenant scenarios, traffic from VIP tenants is typically given higher priority than that of regular tenants; in distributed machine learning scenarios, inference traffic usually requires higher priority than training traffic. These differentiated requirements are reflected, at the router's micro level, through the scheduling algorithms in the traffic manager. In the traffic manager, the packet scheduler handles packet departures, while BM handles packet arrivals. When BM operates under the premise of inherent fairness across flows, while the scheduler aims to provide differentiated services, conflicts inevitably arise. In the example, the blue flow—highest in priority from the scheduler's perspective—does not receive corresponding preferential treatment in BM decisions. As a result, packets that should have been prioritized during departure are instead placed, at arrival, into a lower-priority storage medium (i.e., off-chip DRAM), ultimately leading to priority inversion.

3.1.2 Intra-Flow Head-of-Line Blocking

In the previous subsection we discussed the problem across flows. We now consider a simpler scenario: as shown in Figure 3 (a), a single output port maintains only one queue corresponding to the blue flow. For simplicity, we again assume all packets have the same size. Starting from an empty buffer, packets of the blue flow arrive sequentially, triggering the threshold. Once the threshold is reached subsequent packets of the blue flow are written into off-chip DRAM.

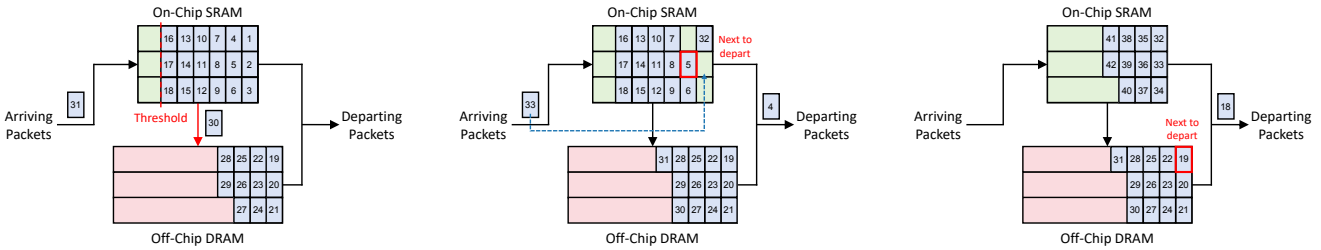
For a single flow, the scheduler serves packets in order of arrival. Consequently, head packets stored in SRAM are gradually transmitted. Since SRAM experiences only departures, as Figure 3 (b) illustrates, the queue length decreases below the threshold, and newly arriving packets are then admitted into SRAM. This process continues: head packets depart to free space, and new packets occupy the freed space. Eventually, as shown in Figure 3 (c), all packets present in SRAM when the threshold was first triggered have departed, and only newly arrived packets remain on-chip.

At this point, the packets that were initially spilled to off-chip DRAM because they were at the tail have now become the head of the flow. As a result, the scheduler continuously fetches packets from off-chip DRAM. We face the same problem again: during this period, because off-chip DRAM bandwidth cannot match the output line rate, the effective throughput of the port is limited by the access speed of off-chip DRAM. Even worse, the packets remaining in SRAM at this moment are all tail packets of the flow. These tail packets are retained on-chip, occupying valuable SRAM space and unable to depart until the head packets in off-chip DRAM have all been transmitted, thereby reducing the buffer's ability to absorb traffic from other flows.

We refer to this phenomenon as the **head-of-line blocking problem**. It is important to note that its cause is different from that of the priority inversion problem. In the example, BM's decisions about packet placement are not incorrect. Throughout the process, whenever on-chip SRAM becomes congested and packets must be written to off-chip DRAM, the packets selected are always those that, according to the scheduling algorithm, should be transmitted last among the current backlog. Choosing any other packets to spill would only cause off-chip transmissions to occur even earlier.

This raises a key question: why can correct decisions still lead to undesirable outcomes? The answer lies in the fact that spillover-only hybrid BM makes one-time placement decisions only when packets arrive, and then does not revisit them, leaving its control ability fundamentally insufficient. BM decisions are made at the moment of arrival, yet their consequences persist into the future. As a result, a choice that appears entirely reasonable at the time may later evolve into an adverse outcome.

Furthermore, the operation of a router requires that its ports continuously receive incoming packets, while the



(a) Tail Packets Spilled to DRAM (b) Later Tail Packets Admitted to SRAM (c) Head-of-Line Blocking

Figure 3: Intra-Flow Head-of-Line Blocking.

router has no knowledge of future arrivals. BM algorithms must therefore make immediate decisions without foresight. Hence, simply refining the decision algorithm alone cannot resolve the head-of-line blocking problem.

3.2 Limitations of Existing Schemes: Execution Perspective

The problems identified in §3.1 ultimately lead to performance degradation, mainly because DRAM bandwidth is limited. Unlike SRAM, whose bandwidth is high and stable, DRAM has lower bandwidth, and its bandwidth varies significantly with access patterns. In the latest DRAM technology, HBM, although bandwidth has been greatly increased, it still cannot fully sustain line-rate processing; moreover, its effective bandwidth exhibits an even stronger dependence on access patterns. Therefore, compared to single BM, which only involves SRAM, studying hybrid BM with HBM requires the *execution perspective*, which focuses on the efficiency of read and write operations on the underlying storage medium.

In the Appendix A, we analyze the read/write performance of HBM. Overall, to fully exploit HBM's bandwidth, accesses to HBM need to satisfy two requirements: (1) spatial continuity, i.e., accessing consecutive addresses, and (2) temporal continuity, i.e., consecutively performing accesses of the same type to avoid frequent switches between reads and writes. However, existing hybrid BM schemes clearly do not consider these two aspects. Consider the scenario with the same configuration as in §3.1.1. As Figure 4 illustrates, according to the scheduling algorithm, the next three packets scheduled to depart are packets 7, 8, and 9 of the blue flow in DRAM. These three packets do not reside at contiguous addresses in DRAM, which means the subsequent three reads are spatially discontinuous. Moreover, if during these three packet departures new packets arrive and are directed to DRAM for storage, then a write access must be interleaved between the reads, so temporal continuity is also not satisfied. Under such an access pattern, DRAM read/write performance degrades further, ultimately significantly reducing the effective throughput at the output port.

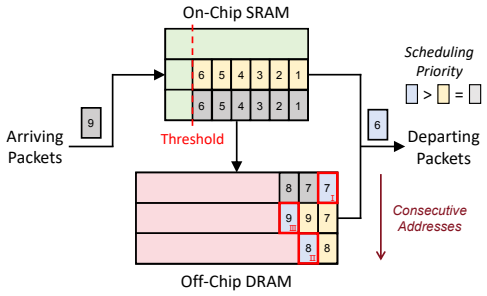


Figure 4: Spatial and Temporal Discontinuous Accesses.

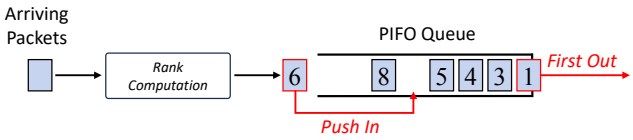


Figure 5: Programmable Scheduling with the PIFO Model.

4 Themis

To address the problems identified in §3, the key is to find an abstraction that captures the semantics of scheduling algorithms. Ideally, such an abstraction should be simple and intuitive, while remaining applicable to different types of scheduling algorithms. The popular programmable scheduler model Push-In First-Out (PIFO) [43] provides such an opportunity. As shown in Figure 5, its key idea is that packets are assigned a rank upon arrival and placed into a priority queue, and departures always take the packet with the smallest rank. By changing the rank computation function, PIFO can implement a variety of scheduling algorithms. For example, by ranking packets with the virtual finish time, PIFO can implement WFQ [19]; by ranking packets with a priority level, PIFO can implement SP.

The rank in the PIFO model naturally serves as the abstraction we need. Building on this abstraction, we design Themis, a scheduling-aware hybrid BM. We begin in §4.1 by describing Themis's design and architecture. In §4.2, we examine its behavior in the representative scenarios introduced earlier. The discussions are summarized in §4.3.

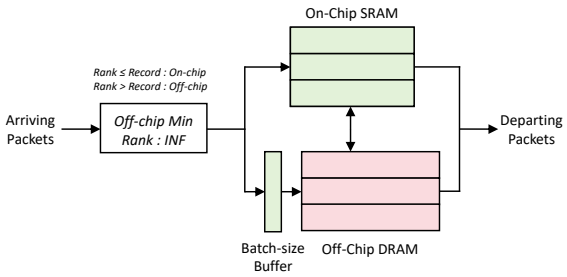


Figure 6: Themis Overall Architecture.

4.1 Design of Themis

Unlike prior approaches that manage buffers at the granularity of queues and assign thresholds to each queue, Themis discards the queue abstraction and instead manages buffers directly at the granularity of individual packets. At this level, Themis's decisions are made directly based on packet ranks. Since packets within the same flow inherently have an inherent rank order, this design does not introduce reordering problems. From an implementation perspective, routers physically contain only a single unified on-chip SRAM; the notion of multiple queues is purely a logical abstraction that requires extra data structures to maintain. By eliminating queues, Themis better matches the actual hardware structure and avoids unnecessary complexity. We revisit this point in more detail in §5 when discussing hardware design.

At a high level, Themis is a unified buffer management scheme, as shown in Figure 6, that operates over a hybrid buffer composed of a small on-chip SRAM and a large off-chip DRAM. Incoming packets are admitted into either SRAM or DRAM based on their rank, and Themis dynamically adjusts their placement between the two tiers. Importantly, all DRAM-related operations in Themis are performed in units of batches, whose size (*batch size*) is defined as the minimum amount of contiguous data required to fully utilize HBM's read/write bandwidth. Specifically, the *batch size* depends on the actual HBM parameters, which we analyze in detail in Appendix A. For ease of illustration, the examples in this section use a *batch size* equal to three packet sizes. Built on this abstraction, Themis provides three key primitives—admission, swap-out, and swap-in—which we describe below.

Admission. Themis maintains a global record of the smallest rank currently stored in DRAM; when DRAM is empty, this value is set to infinity. Upon arrival, a new packet compares its rank with this record: if smaller, it is admitted into SRAM; otherwise, it is designated for DRAM. Packets destined for DRAM are not written immediately. Instead, they are first placed into a temporary buffer that accumulates up to *batch size* packets, implemented in SRAM or registers with line-rate read/write capability. Packets in this buffer are then written to DRAM as a batch and stored in contiguous addresses. Notably, writes are triggered on arrival: when the first packet is being written and the write instruction of the second arrives immediately after, the two are merged into contiguous writes in DRAM; if the first write completes before the second arrives, this indicates that DRAM bandwidth is not the bottleneck at that instant.

Swap-Out. During congestion buildup, as more packets are admitted into SRAM, occupancy increases. To ensure space for future high-priority packets, Themis defines a swap-out watermark in SRAM. When occupancy exceeds this watermark, Themis triggers a swap-out operation, proactively evicting low-priority packets from SRAM. Specifically, Themis selects *batch size* packets with the largest ranks—i.e., the least urgent packets—and forms a batch that is written into DRAM at contiguous addresses. The configuration of this watermark will be discussed in detail in Appendix B. It is worth noting that, unlike prior methods which must reserve a certain amount of SRAM to accommodate potential new flows, Themis only needs to reserve a minimal amount of space. If a new flow is high-priority, its packets naturally enter SRAM and evict lower-priority packets; if it is low-priority, placing its packets in DRAM is precisely the intended behavior.

Swap-In. During congestion draining, as packets depart from SRAM, occupancy decreases. To avoid the scheduler stalling when high-priority packets are only available in DRAM, Themis defines a swap-in watermark. When occupancy falls below this watermark, a swap-in operation is triggered to proactively bring back packets from DRAM. Themis selects the batch with the smallest representative rank—that is, the most urgent batch—and loads it into SRAM in its entirety. Here, rank is a property of packets; since a batch contains multiple packets, Themis assigns the smallest rank within the batch as its representative rank. Only representative ranks are compared across batches, and the batch with the smallest representative rank is chosen for swap-in. The configuration of the swap-in watermark will also be detailed in Appendix B.

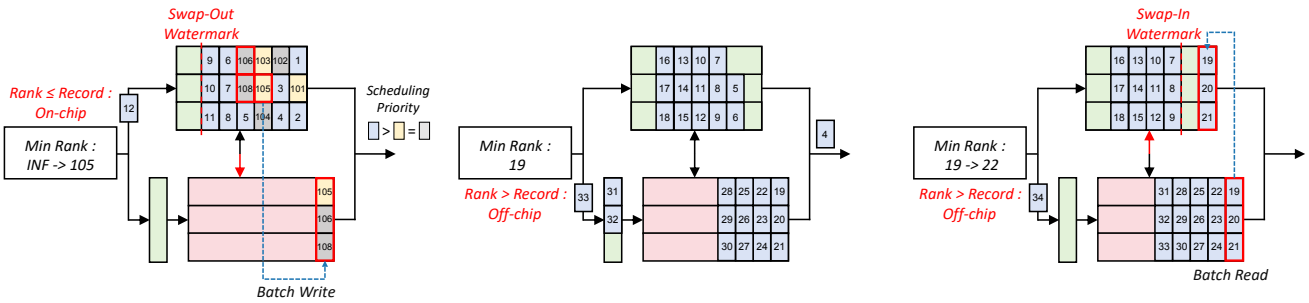
Together, these three primitives define the complete lifecycle of a packet in Themis: on arrival, a packet is admitted into SRAM or DRAM based on its rank; as SRAM occupancy approaches its limit, high-rank packets are swapped out in batches to DRAM; and as SRAM is gradually drained, the most urgent batch in DRAM is swapped back in.

4.2 Behavior of Themis

In the previous section, we introduced the overall design of Themis and its three key primitives. Next, we revisit the representative scenarios discussed in §3 to intuitively demonstrate how Themis performs in them.

4.2.1 Correct Decisions: Rank-Based Admission

In the scenario of §3.1.1, spillover-only hybrid BM, due to queue-based thresholds, places part of the blue flow pack-



(a) Rank-Based Admission (b) Tail Packets Forced to DRAM (c) Swap-In of Middle Packets

Figure 7: Themis Behavior in Challenging Scenarios.

ets into off-chip DRAM, forcing the scheduler to fetch high-priority packets from DRAM. In Themis, as shown in Figure 7 (a), the numbers on the packets no longer indicate their order within the flow but instead represent their rank in the PIFO model. Consequently, the high-priority blue flow is assigned smaller ranks, while the lower-priority yellow and gray flows receive larger ranks. These larger ranks increase in an interleaved manner, indicating that the yellow and gray flows equally share the remaining bandwidth. Packets of the blue flow, having smaller ranks, are always admitted into on-chip SRAM, while packets of the yellow and gray flows are more often directed to DRAM. As a result, the scheduler can continuously obtain blue flow packets from SRAM for transmission.

The core difficulty in addressing priority inversion lies in making BM decisions faithfully reflect scheduling semantics. In the traditional BM framework, this requires designing and maintaining complex threshold algorithms for different queues, and these thresholds are tightly coupled to specific scheduling policies—once the scheduling algorithm changes, the threshold algorithm must also change accordingly. Themis breaks this limitation: it bases admission on packet departure order (rank) and discards the queue abstraction. Since rank is itself the semantic measure used by scheduling algorithms, admission decisions are naturally aligned with the scheduling semantics; meanwhile, discarding queues means that only a single global threshold is needed, which is decoupled from specific scheduling policies. Through this design, Themis ensures that BM is strictly aligned with scheduling semantics while keeping the implementation simple and general.

4.2.2 Active Management: Dynamic Swapping

In the scenario of §3.1.2, spillover-only hybrid BM eventually results in a distribution where, within a single flow, the middle packets are placed in off-chip DRAM while the tail packets remain in on-chip SRAM. As a result, once the head packets in SRAM have been transmitted, the scheduler has to fetch the middle packets from DRAM for an extended period; meanwhile, the tail packets remain stuck in SRAM. In Themis, as shown in Figure 7 (b), newly arriving tail packets are written into DRAM—even when free space is available in SRAM—thereby allowing SRAM space to be gradually released. As shown in Figure 7 (c), when SRAM occupancy falls below the swap-in watermark, the middle packets are proactively brought back from DRAM, enabling the scheduler to continue transmitting directly from SRAM.

The key difficulty in addressing the head-of-line blocking problem lies in the limited control of BM, which confines placement decisions to the moment of packet arrival. Themis overcomes this limitation by introducing dynamic swapping, granting BM the ability of proactive management. The swap-out and swap-in primitives operate continuously to reshape the distribution of packets between SRAM and DRAM: when occupancy exceeds the swap-out watermark, the packets with the largest ranks are evicted in batches to DRAM; when occupancy drops below the swap-in watermark, the packets with the smallest rank are brought back to SRAM. In this way, SRAM content is continuously adjusted with congestion buildup and draining, ensuring it always holds the most urgent packets, while DRAM efficiently stores those deferred for later processing.

4.2.3 Efficient Access: Batch-Based Operations

In the scenarios of §3.1.1 and §3.1.2, DRAM accesses under spillover-only hybrid BM are fragmented and fine-grained, which further aggravates the bandwidth bottleneck. In contrast, as shown in Figure 7, all DRAM accesses in Themis are carried out in batches: on writes, packets are collected into a batch and written sequentially to contiguous addresses; on reads, the entire batch is read out in one operation. The key difficulty in achieving efficient DRAM access lies in the lack of spatial and temporal locality in packet buffering. In Themis, the mechanisms introduced to solve the first two problems—Rank-Based Admission and Dynamic Swapping—also naturally overcome this limitation.

Along the spatial dimension, by leveraging ranks, Themis aligns packet organization with scheduling semantics, grouping packets with similar ranks into batches: for DRAM, a batch written via swap-out consists of the packets with the

largest ranks currently in SRAM, while a batch written via admission consists of packets with ranks greater than the record maintained in Themis. In both cases, packets within the same batch have been classified or filtered so that they are close in scheduling order and placed into contiguous addresses, thereby actively constructing spatial locality.

Along the temporal dimension, through the swap-in and swap-out operations, Themis transforms DRAM accesses from originally random and passive interleaving into ordered and proactive operations. As a result, Themis can bundle read and write operations into batches, thereby actively constructing temporal locality.

4.3 Discussions

Impact of Swap-In/Out. A natural question is whether the swap-in and swap-out primitives in Themis interfere with normal buffer operations (i.e., packet arrivals and departures), and whether they consume SRAM bandwidth that should be used for these operations.

First, in terms of execution priority, Themis ensures that swap-in/out always have strictly lower priority than arrivals and departures. Swapping is performed only when it does not interfere with them. Moreover, the swap process is preemptible: if a new arrival or departure occurs during swap-in/out, the swap is immediately interrupted. This design is consistent with the approach taken in Occamy [41].

Second, regarding SRAM bandwidth usage, the swap primitives exploit only residual bandwidth. Take swap-out, which consumes read bandwidth, as an example: its purpose is to prevent SRAM from becoming overly occupied. When the packet arrival rate exceeds the departure rate, there is spare read bandwidth at the SRAM output, which can be used for swap-out. Conversely, when the packet arrival rate is less than or equal to the departure rate, SRAM occupancy will not keep growing, so swap-out is not urgent and can be paused or delayed without affecting performance. A similar reasoning applies to swap-in.

Packet Loss. Compared to a buffer with only on-chip SRAM, a hybrid buffer is more effective in absorbing bursty traffic. However, due to the limited off-chip DRAM bandwidth, packet drops are unavoidable. In Themis, packet loss only occurs in the following two scenarios: (1) For packets designated for off-chip DRAM upon arrival, Themis first places them into a temporary buffer. The write-in rate of this buffer is determined by the packet arrival speed, while the read-out rate is constrained by DRAM bandwidth and is typically slower. Once the buffer is full, newly arriving packets will be directly dropped. (2) When the swap-out watermark is reached, Themis moves data from SRAM to DRAM. However, since DRAM write speed is relatively low and new packets may continue entering SRAM at a faster rate, SRAM occupancy can keep increasing even during an ongoing swap-out. When SRAM is about to be completely filled while the current swap-out has not yet finished, Themis directly drops a batch of the highest-rank packets to free up space.

Overall, the packet dropping policy of Themis is as follows: from the perspective of SRAM, all dropped packets always have higher rank than any packet that remains in SRAM, ensuring that SRAM consistently retains the most important data. From the perspective of DRAM, the system adopts a best-effort policy for low-priority packets: it stores them whenever possible, but packets that exceed the limited DRAM write bandwidth are inevitably discarded. In this way, the system can still achieve priority-aware trade-offs under resource constraints.

Due to space limitations, we place the remaining discussion in Appendix B. Specifically, it covers: (1) the on-chip dequeue hit rate metric and its implications for output link utilization; and (2) guidance on setting the swap-in and swap-out watermarks, which reflects the trade-off between burst absorption capability and on-chip dequeue hit rate.

5 Implementing Themis in Hardware

In this section, we describe how Themis is implemented within the traffic manager. §5.1 gives background, §5.2 analyzes the operation of Themis primitives, and §5.3 presents the BM-SCH Unified Model enabling cost-efficient hardware implementation.

5.1 Traffic Manager

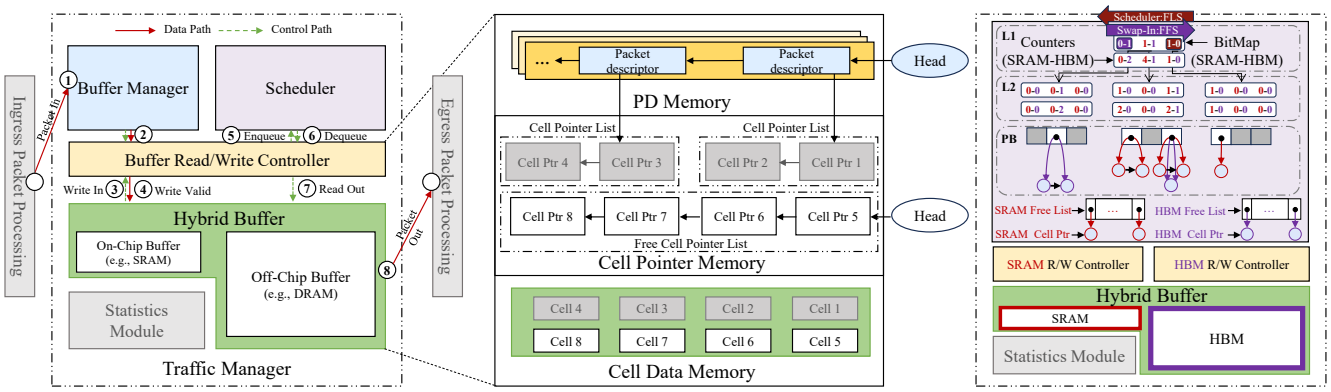
In router chips, BM is a module of the TM. We provide details of the TM in Appendix D. As shown in Figure 8 (a), the TM processes a packet through four main steps: (1) BM admits the packet and determines whether it is stored in on-chip or off-chip memory; (2) the buffer read/write controller stores the packet data and descriptors; (3) the scheduler dequeues the packet; and (4) the buffer read/write controller reads out the packet and releases the associated buffer space.

5.2 Analysis of Themis Primitives

Compared to previous BM schemes, Themis introduces the swap-in and swap-out primitives. The core operation of these primitives is to identify packets with the largest ranks. A similar mechanism exists in the TM: in the programmable scheduler PIFO [43] introduced earlier, scheduling is achieved by selecting the packet with the smallest rank. For PIFO, numerous high-speed implementations [13,21,48] have been developed, making it fully capable of serving as a high-speed port scheduler. This observation motivates the question: **Since finding both the largest and smallest ranks relies on sorting, can we leverage PIFO to help implement Themis?**

5.3 BM–SCH Unified Model

We provide the answer in practice: Themis reuses the scheduler and implements a streamlined design. This design, re-



(a) Traffic Manager (b) Buffer Architecture (c) Themis

Figure 8: Architecture of TM, Buffer and Themis.

ferred to as the BM-SCH Unified Model, unifies BM and scheduler operations within a single hardware structure. Figure 8 (c) gives an overview. Specifically, we take BBQ [13], a state-of-the-art PIFO implementation in terms of scalability and throughput, as the starting point. Details of BBQ are provided in Appendix E. In essence, BBQ implements a bucket-based sorting scheme: each rank is assigned a bucket, and a hierarchical bitmap is used to index the buckets. When a packet is enqueued, its rank determines the corresponding bucket, and the bitmap is updated accordingly. During dequeue, a Find-First-Set (FFS) operation on the bitmap is used to locate the bucket with the smallest rank. Based on BBQ, our design introduces:

Enabling Find-Last-Set (FLS). BBQ locates the packet with the smallest rank using FFS. To identify the packet with the largest rank, we introduce the FLS operation. FLS is realized by simply inverting the bitmap bits and reusing the original FFS, without additional time overhead. With FLS enabled, the lowest-priority batch of packets is swapped from SRAM to HBM, thus realizing swap-out.

Storing and Organizing Cell Pointers in Priority Buckets (PB). In the original BBQ design, Packet Descriptor (PD) addresses are stored in the PB. This means that, for each packet dequeue, the system must first retrieve the descriptor address from the PB, then use it to index the cell pointer list, and finally access the actual cell containing the packet data. We simplify this process by directly storing the cell pointer list in the PB of BBQ. Similarly, the free cell pointer list is maintained using BBQ's existing free list. This approach eliminates unnecessary memory access overhead, thereby saving memory bandwidth.

Hybrid Buffer Support in BBQ. BBQ is originally designed to manage a single buffer. For a hybrid buffer comprising both SRAM and HBM, two BBQ instances would normally be required. We enhance BBQ by adding shadow bitmaps and shadow counters at each hierarchy level: one set tracks packets in the SRAM buffer, while the other tracks packets in the HBM buffer. By reusing the computational logic, this modification enables a single BBQ to efficiently manage a hybrid buffer with minimal additional overhead.

As shown in Figure 8 (c), the packet processing workflow operates as follows: (1) during enqueue, hierarchical bitmap traversal locates the target priority bucket, updates path counters and bitmaps, and stores the packet in SRAM; (2) a unified centralized controller orchestrates swap-in and swap-out between SRAM and HBM, with swap-out realized via FLS and swap-in via FFS; (3) during dequeue, FFS identifies the highest-priority non-empty bucket across both SRAM and HBM, and the corresponding packet is dequeued. All operations are fully pipelined to maximize performance, as detailed in Appendix F.

6 Prototype and Evaluation

In this section, we first present the hardware prototype of Themis, implemented on both FPGA and ASIC, and analyze its resource overhead and maximum frequency. Based on the FPGA prototype equipped with HBM, we conduct testbed experiments to verify its improvement on packet forwarding completion time. We then perform simulations using ns-3 [36]. The simulations include two parts: (1) single-point, viewed from the microscopic perspective of a single router, to assess packet loss, on-chip dequeue hit rate, and output link utilization; and (2) large-scale, viewed from the macroscopic perspective of the entire network, to assess the FCT.

6.1 Hardware Prototype

The hardware prototype of Themis, including the buffer management, scheduler, and hybrid buffer, is written in 3654 lines of SystemVerilog code (with Block Memory Generator IP instances and HBM IP instances) and implemented on Xilinx Alveo U280 datacenter accelerator card with XCU280 FPGA [47], which has 1082K LUTs, 1490 36KLUTRAMs and 2180K Flip-Flops. The U280 is equipped with 8 GB of HBM2 memory, which offers up to 460 GB/s of memory bandwidth. To analyze the overhead of Themis (excludes hy-

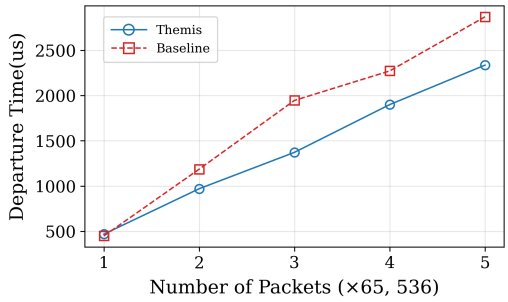


Figure 9: Packet Departure Time Comparison.

brid buffer: 5 MB SRAM + 4 GB HBM), we use Vivado Design Suite [11] to evaluate the FPGA resource overhead. Furthermore, we use Design Compiler [26] to synthesize the Themis in GlobalFoundries 28nm process [22], evaluating the timing, chip area, and power consumption.

Table 1: Hardware Cost.

Module	FPGA Cost		ASIC Cost		
	LUTS	Flip Flops	Meets Timing at 1 GHz	Area (mm^2)	Power (mW)
Baseline	7053	7347	yes	1.622	3.439
Themis	7157	7435	yes	1.827	3.746
Additional overhead	134	88	--	0.205	0.307

Table 1 shows the FPGA resource utilization and the ASIC cost of Themis. Themis costs 7157 LUTs and 7435 Flip-Flops, only introducing 134 LUTs and 88 Flip-Flops compared with a traditional TM baseline. This baseline employs a BBQ scheduler and a common Greedy-Hybrid BM approach, which defaults to SRAM and uses HBM only upon overflow. Besides, the timing report indicates that Themis has a latency of less than 1.0 ns, meaning it can perform swap-in/out per cycle at a 1.0 GHz clock. Furthermore, Themis introduces less than 0.205 mm^2 of ASIC area and 0.307 mW of power consumption, which is negligible for a commercial switch ASIC.

6.2 Testbed Evaluation

We employ Integrated Logic Analyzer (ILA) [3] to capture the average departure time of every 65,536 packets residing in the buffer in real-time, comparing Themis with Greedy-Hybrid BM.

In the experimental setup, we maintain consistent conditions, including the same hybrid buffer size (5MB SRAM + 4GB HBM), identical exit scheduling speed and packet arrival rates, and statistically similar packet sizes. We also implement a packet generator with random ranks on the FPGA, utilizing clock and packet count counters to measure the average departure time of every 65,536 packets residing in the buffer. As shown in Figure 9, Themis shortens the forwarding completion time of all packets by up to 22% compared to Greedy-Hybrid BM.

6.3 Simulation

We conduct our experiments on ns-3, where we provide a HBM model. Specifically, we implement a bank-based state machine, maintain each bank's row-activation state, command queues in parallel, and timing counters. The model strictly enforces HBM timing constraints and accurately computes request latency under different access patterns. We compare Themis with spillover-only hybrid BM. Specifically, we implement three variants of the spillover-only hybrid BM:

1. **Greedy-Hybrid**: Packets are written to SRAM by default upon arrival. Only when SRAM capacity is insufficient are packets written to HBM instead. Once a packet enters the buffer, it is neither moved nor dropped before dequeue.

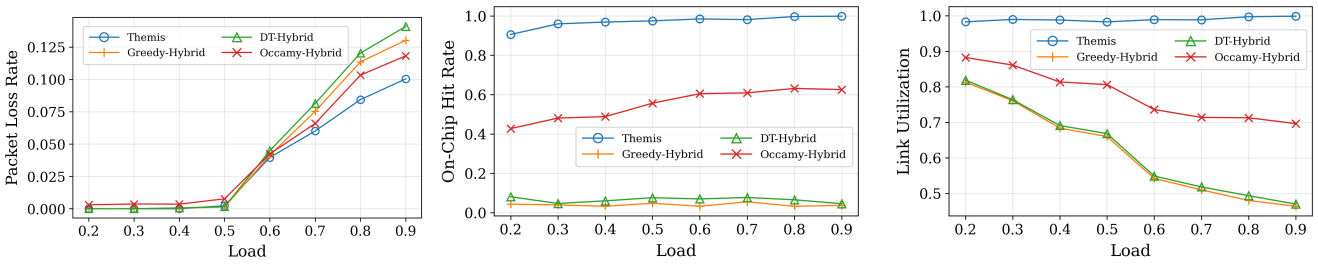
2. **DT-Hybrid**: Dynamic Threshold (DT) [17] is a widely used admission control policy in commodity switches. Upon packet arrival, DT decides whether to admit the packet based on current buffer occupancy: the closer the buffer is to full, the stricter the admission threshold. Packets rejected by DT are dropped, whereas admitted packets remain in the buffer until dequeued. In the hybrid-buffer setting, we adapt DT by redirecting to HBM the packets it would otherwise drop.

3. **Occamy-Hybrid**: Occamy [41] is a state-of-the-art buffer management scheme, which achieves preemptive buffer management long thought impossible. Specifically, Occamy reuses DT for admission control. It preemptively reclaims space by dropping buffered packets from flows that over-occupied the buffer when DT was more permissive. In the hybrid buffer setting, we also adapt Occamy. Packets that would be dropped during admission or preemption are instead written to HBM.

6.3.1 Single-Point Simulation

In the single-point experiments, we compare Themis against three baseline schemes in terms of packet loss rate, on-chip dequeue hit rate, output link utilization, and HBM throughput within a single switch.

Experimental Setup. We evaluate a 32-to-1 aggregation with 50 Gbps per input port. The arrivals follow a Poisson process, and flow sizes are drawn from the WebSearch distribution [7]. We evaluate performance metrics under different total input port loads, a higher input load implies more severe congestion at the buffer. In the hybrid buffer, the SRAM capacity is set to 5 MB, and the HBM timing parameters



(a) Packet Loss Rate (b) On-Chip Dequeue Hit Rate (c) Link Utilization

Figure 10: Performance in Single-Point Experiments.

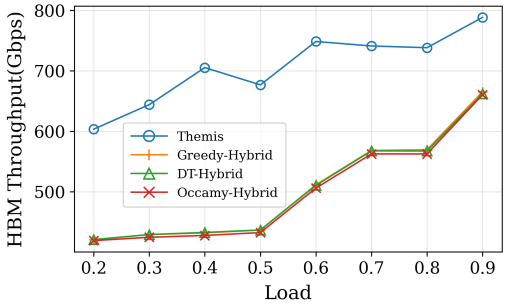


Figure 11: HBM Throughput vs. Load.

follow Samsung HBM2E [38]. We use UDP as the transport layer protocol and adopt Shortest Flow First (SFF) [43] as the scheduling algorithm. SFF can be implemented by a PIFO instance and is observed to perform almost as well as pFabric [8]. We evaluate the performance under varying total input loads, using HBM throughput, packet loss rate, on-chip dequeue hit rate and link utilization as metrics.

Performance. Across all input loads, Themis consistently outperforms the three spillover-only hybrid BM schemes on all metrics. The advantages manifest in these aspects: (1) As shown in Figure 11, the hybrid baselines sustain only about 410 to 650 Gbps of HBM throughput across 0.2 to 0.9 offered load. In contrast, Themis achieves 600 to 790 Gbps by merging accesses, improving performance by up to 46%. (2) The higher HBM throughput enables Themis to accommodate larger bursts. As shown in Figure 10 (a), Themis consistently incurs lower packet loss than the three hybrid baselines, with its advantage more pronounced at the highest load. Compared with other schemes, Themis reduces packet loss by 18% to 30%. (3) By adopting a scheduling-aware placement strategy which preferentially keeps high-priority packets on chip, Themis achieves a higher on-chip dequeue hit rate. As shown in Figure 10 (b), compared with the strongest baseline (Occamy-Hybrid), Themis achieves up to a 2.16× improvement in the on-chip dequeue hit rate. (4) The joint effect of the higher hit rate and increased HBM throughput further yields superior link utilization. Figure 10 (c) shows a 43% improvement over Occamy-Hybrid. Additional results evaluating Themis under varying traffic patterns are provided in the Appendix C.

Watermark. In the preceding experiments, Themis sustains a very high hit rate and a low packet-loss rate. Therefore, the impact of watermark tuning was limited. To further probe this effect, we subject the system to a heavier load by raising each link speed to 100 Gbps, and assess how watermarks influence performance. We consider two setups: (1) fixing either the swap-in or swap-out watermark while modifying the other; and (2) co-tuning both watermarks while maintaining a constant gap between them.

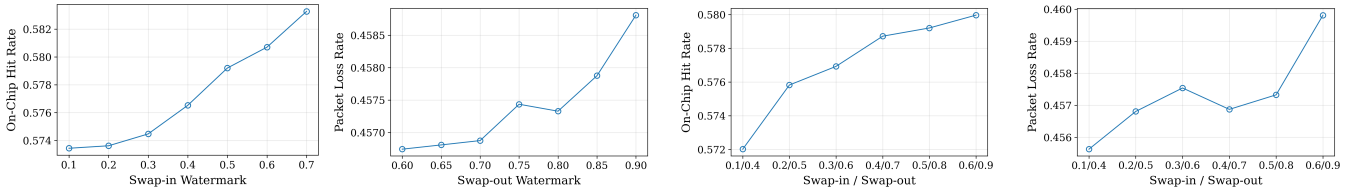
As shown in Figure 12 (a) and 12 (c), higher watermarks retain more packets on chip and thus yield a higher on-chip dequeue hit rate. In addition, Figure 12 (b) and 12 (d) show that lower watermarks reduce packet loss by engaging HBM earlier to share the storage burden. Overall, watermark selection represents a trade-off between ingress drops and egress hit rate. Balancing these objectives, we adopt the swap-in watermark as 0.5 and the swap-out watermark as 0.8 as the default configuration for Themis.

6.3.2 Large-Scale Simulation

In this section, we evaluate and compare Themis and other baselines under large-scale datacenter scenarios with real traffic. Switch configuration and scheduling remain identical to the single-point experiments.

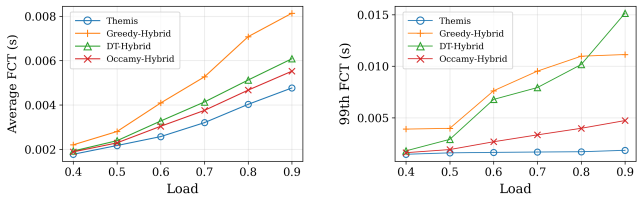
Topology. We implement a canonical Leaf-Spine topology widely deployed in production. The network comprises 9 Leaf switches and 4 Spine switches, interconnecting 144 servers. Each server attaches to a Leaf via a 50 Gbps downlink; each Leaf connects to every Spine via 200 Gbps uplinks. Link propagation delay is 10μs, and the end-to-end RTT through a Spine path is about 80μs. Load balancing uses Equal-Cost Multi-Path (ECMP) routing with five-tuple hashing, selecting the shortest path at flow granularity.

Workload. We use two representative workloads from real datacenter: WebSearch and traffic collected from Facebook's Hadoop cluster [37]. Both exhibit pronounced heavy tails and are dominated by small flows. Hadoop's average flow is smaller, with over 70% of flows within 10 KB, whereas WebSearch flow sizes are mainly concentrated between 10 KB



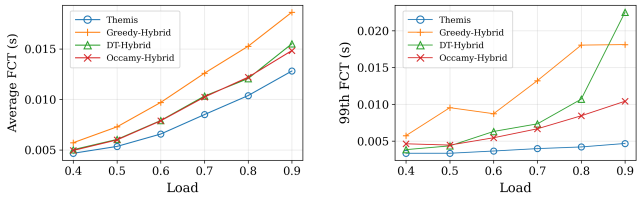
(a) Swap-Out = 0.8 (b) Swap-In = 0.5 (c) Difference = 0.3 (d) Difference = 0.3

Figure 12: Watermark Sensitivity (Total Input Port Load = 0.8).



(a) Overall: Avg (b) (0, 100KB]: 99th prctile

Figure 13: Performance in Hadoop Workload.



(a) Overall: Avg (b) (0, 100KB]: 99th prctile

Figure 14: Performance in WebSearch Workload.

and 1 MB. Source and destination hosts are chosen uniformly at random among the 144 servers. The transport protocol is TCP-NewReno [9]. We set MSS to 1448 bytes, the initial congestion window to 20 segments, and RTOmin to 0.24 ms.

Metrics. In large-scale experiments, we focus on the following performance metrics. First, we report the average FCT over all flows and the 99th percentile FCT for small flows, where small flows denote flows less than 100 KB. We normalize each flow's FCT by its corresponding flow size.

Performance. As shown in Figure 13 and 14, across two production workloads (WebSearch and Hadoop) and loads from 0.4 to 0.9, Themis attains the lowest average FCT and 99th percentile FCT in small flows. The advantage is modest at light load and widens as the network becomes busier. Under high load, the baselines exhibit sharp tail inflation, whereas Themis grows much more gently. Quantitatively, compared with each workload's best-performing baseline, Themis reduces the 99th percentile FCT by a factor of 2.23 on WebSearch and 2.61 on Hadoop.

7 Related Work

Modern buffer management is largely influenced by DT [16, 17, 20], a simple yet effective scheme that adapts queue limits to traffic load. Building on this foundation, some works focus on enhancing burst absorption in buffers [5, 10, 12, 23, 35, 40]. Some works incorporate adaptive control (ABM [4]) or machine learning (NDT [45], Credence [6]) to optimize buffer sharing. The latest work Occamy [41] exploits redundant memory bandwidth to rapidly evict packets from over-allocated queues, significantly improving burst absorption, performance isolation, and mitigation of buffer blockage compared to the non-preemptive BM. While these schemes optimize the use of on-chip SRAM, they are all fundamentally constrained by its limited capacity. To address this, some commercial routers adopt hybrid architectures, using off-chip DRAM as a simple spillover destination for congested queues [15, 24, 32]. However, this spillover-only model, while increasing total buffer size, incurs significant latency for traffic directed to slower off-chip memory, motivating more intelligent hybrid buffer management designs.

8 Conclusion

In this paper, we reveal the priority inversion and head-of-line blocking problems in hybrid buffers under existing buffer management, and we analyze how HBM performance depends on access patterns in packet buffering scenarios. We propose Themis, a scheduling-aware hybrid buffer management scheme that places early-departing packets in fast on-chip memory and organizes off-chip accesses to be temporally and spatially contiguous. Experiment and simulation results show that Themis significantly improves the on-chip dequeue hit rate and output link utilization, ultimately reducing overall flow completion times.

Acknowledgement. We thank the anonymous reviewers and the paper shepherd Ratul Mahajan for their insightful comments and suggestions, which help improve this paper. We also thank Guang Li, Nan Li and Danfeng Shan for all technical discussions and valuable comments.

This work is sponsored by the National Key Research and Development Program of China (No. 2024YFE0203900) and National Natural Science Foundation of China (U25B2032, 62572128).

References

[1] High bandwidth memory (hbm) dram standard, jesd235. Technical report, JEDEC Solid State Technology Association, 2013.

[2] Ieee standard for ethernetamendment 11: Media access control parameters, physical layers, and management parameters for 200 gb/s and 400 gb/s, 2017.

[3] Integrated Logic Analyzer v6.3 LogiCORE IP Product Guide. Technical Report PG172, Xilinx Inc., 2023.

[4] Vamsi Addanki, Maria Apostolaki, Manya Ghobadi, Stefan Schmid, and Laurent Vanbever. Abm: Active buffer management in datacenters. In *Proceedings of the ACM SIGCOMM 2022 Conference*, pages 36–52, 2022.

[5] Vamsi Addanki, Wei Bai, Stefan Schmid, and Maria Apostolaki. Reverie: Low pass {Filter-Based} switch buffer sharing for datacenters with {RDMA} and {TCP} traffic. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pages 651–668, 2024.

[6] Vamsi Addanki, Maciej Pacut, and Stefan Schmid. Credence: Augmenting datacenter switch buffer sharing with {ML} predictions. In *21st USENIX symposium on networked systems design and implementation (NSDI 24)*, pages 613–634, 2024.

[7] Mohammad Alizadeh, Albert Greenberg, David A Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. Data center tcp (dctcp). In *Proceedings of the ACM SIGCOMM 2010 Conference*, pages 63–74, 2010.

[8] Mohammad Alizadeh, Shuang Yang, Milad Sharif, Sachin Katti, Nick McKeown, Balaji Prabhakar, and Scott Shenker. pfabric: Minimal near-optimal datacenter transport. *ACM SIGCOMM Computer Communication Review*, 2013.

[9] M. Allman, V. Paxson, and W. Stevens. Tcp congestion control. RFC 6582, 2012. Updates RFC 2582.

[10] Hamidreza Almasi, Rohan Vardekar, and Balajee Vamanan. Protean: Adaptive management of shared-memory in datacenter switches. In *IEEE INFOCOM 2023-IEEE Conference on Computer Communications*, pages 1–10. IEEE, 2023.

[11] AMD. Amd vivado design suite. https://www.amd.com/ts/software/adaptive-socs-and-fpgas/vivado.html, [n.d.].

[12] Maria Apostolaki, Laurent Vanbever, and Manya Ghobadi. Fab: Toward flow-aware buffer sharing on programmable switches. In *Proceedings of the 2019 Workshop on Buffer Sizing*, pages 1–6, 2019.

[13] Nirav Atre, Hugo Sadok, and Justine Sherry. {BBQ}: A fast and scalable integer priority queue for hardware packet scheduling. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pages 455–475, 2024.

[14] Jon CR Bennett and Hui Zhang. Hierarchical packet fair queueing algorithms. *IEEE/ACM Transactions on networking*, 5(5):675–689, 1997.

[15] Broadcom Inc. Bcm88800 traffic management architecture design guide. Technical Report 88800-DG108-PUB, Broadcom Inc., February 2021. Accessed: 2025-09-08.

[16] Abhijit K Choudhury and Ellen L Hahne. Dynamic queue length thresholds in a shared memory atm switch, July 30 1996. US Patent 5,541,912.

[17] Abhijit K Choudhury and Ellen L Hahne. Dynamic queue length thresholds for shared-memory packet switches. *IEEE/ACM Transactions On Networking*, 6(2):130–140, 2002.

[18] Cisco Systems. Hybrid buffer architecture: A silicon one approach to scalable router design. https://www.cisco.com/c/dam/en/us/solutions/collateral/silicon-one/white-paper-sp-hybrid-buffer-architecture.pdf, 2021. White Paper.

[19] Alan Demers, Srinivasan Keshav, and Scott Shenker. Analysis and simulation of a fair queueing algorithm. *ACM SIGCOMM Computer Communication Review*, 1989.

[20] Ruixue Fan, Alexander Ishii, Brian Mark, G Ramamurthy, and Qiang Ren. An optimal buffer management scheme with dynamic thresholds. In *Seamless Interconnection for Universal Services. Global Telecommunications Conference. GLOBECOM'99.(Cat. No. 99CH37042)*, volume 1, pages 631–637. IEEE, 1999.

[21] Peixuan Gao, Anthony Dalleggio, Jiajin Liu, Chen Peng, Yang Xu, and H Jonathan Chao. Sifter: An {Inversion-Free} and {Large-Capacity} programmable packet scheduler. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pages 75–95, 2024.

[22] GlobalFoundries. Globalfoundries offers new low-power 28nm solution for highperformance mobile and iot applications. GlobalFoundries Press Release, May

2015. 28nm Super Low Power (SLP)RF Process Technology with HKMG, accessed via GF website.

[23] Sijiang Huang, Mowei Wang, and Yong Cui. Traffic-aware buffer management in shared memory switches. *IEEE/ACM Transactions on Networking*, 30(6):2559–2573, 2022.

[24] Huawei Technologies Co., Ltd. Cloudengine 12800 series data center switches, 2025. Accessed: 2025-09-08.

[25] Broadcom Inc. Broadcom announces industrys first 12.8tbps ethernet switch, the strataxgs tomahawk 3 family. https://www.broadcom.com/company/news/product-releases/53003, 2017. Press Release.

[26] Synopsys Inc. Synopsys design compiler. https://www.synopsys.com/implementation-and-signoff/rtl-synthesis-test/dc-ultra.html, [n.d.].

[27] S. Iyer, R.R. Kompella, and N. McKeown. Analysis of a memory architecture for fast packet buffers. In *2001 IEEE Workshop on High Performance Switching and Routing (IEEE Cat. No.01TH8552)*, pages 368–373, 2001.

[28] Sundar Iyer, Ramana Rao Kompella, and Nick McKeown. Designing packet buffers for router linecards. *IEEE/ACM Transactions On Networking*, 16(3):705–717, 2008.

[29] jedec. Jedec standard jesd235: High bandwidth memory (hbm). https://www.jedec.org/standards-documents/docs/jesd235a, 2013.

[30] JEDEC Solid State Technology Association. Ddr4 sdram standard, 2012. JESD79-4 Standard.

[31] Cunlu Li, Dezun Dong, Xiangke Liao, and John Kim. Hybrid memory buffer microarchitecture for high-radix routers. *IEEE Transactions on Computers*, 71(11):2888–2902, 2021.

[32] Cunlu Li, Dezun Dong, Xiangke Liao, John Kim, and Changhyun Kim. Deephir: Improving high-radix router throughput with deep hybrid memory buffer microarchitecture. In *Proceedings of the ACM International Conference on Supercomputing*, pages 403–413, 2019.

[33] Dong Lin, Mounir Hamdi, and Jogesh Muppala. Distributed packet buffers for high-bandwidth switches and routers. *IEEE Transactions on Parallel and Distributed Systems*, 23(7):1178–1192, 2011.

[34] Arthur Mutter. A novel hybrid memory architecture with parallel dram for fast packet buffers. In *2010 International Conference on High Performance Switching and Routing*, pages 44–51. IEEE, 2010.

[35] Hamed Rezaei, Hamidreza Almasi, and Balajee Vamanan. Smartbuf: An agile memory management for shared-memory switches in datacenters. In *2021 IEEE/ACM 29th International Symposium on Quality of Service (IWQOS)*, pages 1–7. IEEE, 2021.

[36] George F Riley and Thomas R Henderson. The ns-3 network simulator. In *Modeling and tools for network simulation*, pages 15–34. Springer, 2010.

[37] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, George Porter, and Alex C Snoeren. Inside the social network's (datacenter) network. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, pages 123–137, 2015.

[38] Samsung Semiconductor. Khaa84901b-jc16 hbm2e flashbolt specifications. https://semiconductor.samsung.cn/dram/hbm/hbm2e-flashbolt/khaa84901b-jc16/.

[39] Sergey Savastiouk, Oleg Siniaguine, and Eugene Korczynski. Through-silicon via (tsv) technology for 3d wafer-level packaging, 2000. Unpublished manuscript.

[40] Danfeng Shan, Wanchun Jiang, and Fengyuan Ren. Absorbing micro-burst traffic by enhancing dynamic threshold policy of data center switches. In *2015 IEEE Conference on Computer Communications (INFOCOM)*, pages 118–126. IEEE, 2015.

[41] Danfeng Shan, Yunguang Li, Jinchao Ma, Zhenxing Zhang, Zeyu Liang, Xinyu Wen, Hao Li, Wanchun Jiang, Nan Li, and Fengyuan Ren. Occamy: A preemptive buffer management for on-chip shared-memory switches. In *Proceedings of the Twentieth European Conference on Computer Systems*, pages 1365–1382, 2025.

[42] Madhavapeddi Shreedhar and George Varghese. Efficient fair queuing using deficit round-robin. *IEEE/ACM Transactions on networking*, 4(3):375–385, 1996.

[43] Anirudh Sivaraman, Suvinay Subramanian, Mohammad Alizadeh, Sharad Chole, Shang-Tse Chuang, Anurag Agrawal, Hari Balakrishnan, Tom Edsall, Sachin Katti, and Nick McKeown. Programmable packet scheduling at line rate. In *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM)*, 2016.

[44] Feng Wang, Mounir Hamdi, and Jogesh K. Muppala. Using parallel dram to scale router buffers. *IEEE Transactions on Parallel and Distributed Systems*, 20(5):710–724, 2009.

[45] Mowei Wang, Sijiang Huang, Yong Cui, Wendong Wang, and Zhenhua Liu. Learning buffer management policies for shared memory switches. In *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*, pages 730–739. IEEE, 2022.

[46] Wiki. High bandwidth memory. https://en.wikipedia.org/wiki/High_Bandwidth_Memory.

[47] Xilinx Inc. Alveo U280 Data Center Accelerator Card. https://www.xilinx.com/products/boards-and-kits/alveo/u280.html, 2018. Product Brief DS963 (v1.0), Accessed: February 26, 2026.

[48] Ruyi Yao, Zhiyu Zhang, Gaojian Fang, Peixuan Gao, Sen Liu, Yibo Fan, Yang Xu, and H Jonathan Chao. Bmw tree: Large-scale, high-throughput and modular pifo implementation using balanced multi-way sorting tree. In *Proceedings of the ACM SIGCOMM 2023 Conference*, pages 208–219, 2023.

[49] Ling Zheng, Zhiliang Qiu, Shiyong Sun, Weitao Pan, Ya Gao, and Zhiyi Zhang. Design and analysis of a parallel hybrid memory architecture for per-flow buffering in high-speed switches and routers. *Journal of Communications and Networks*, 20(6):578–592, 2018.

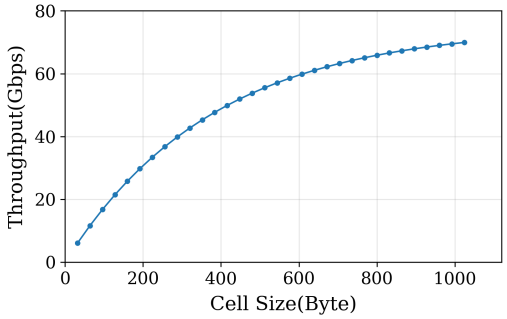


Figure 15: Single-Bank Read Throughput vs. Cell Size.

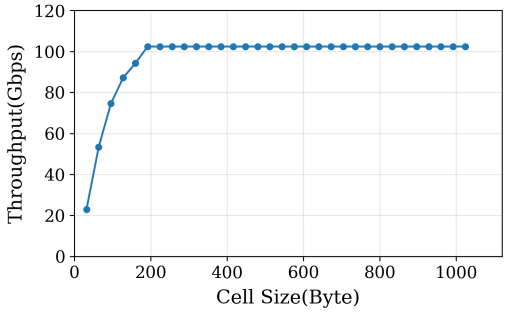


Figure 16: Single-BG Read Throughput vs. Cell Size.

A Impact of Access Patterns on HBM Performance

Compared with DDR, HBM offers substantially higher peak bandwidth. This advantage does not stem from shorter single-access latency; rather, it relies on extremely high degrees of parallelism. Consequently, if an application's access pattern cannot effectively exploit this parallelism, HBM may even underperform DDR. In this section, we use Samsung's HBM2E Flashbolt [38] as a case study—16 GB capacity with a peak throughput of 3.2 Tbps—to analyze how different access patterns shape realized performance.

HBM is hierarchically organized into pseudo channels (PCs), bank groups (BGs), and banks. Within a bank, a row and column identify the target address, and the minimum access granularity is 32 B. For each memory command, the controller must locate the row holding the data, issue precharge and activation, and then access one minimum access granularity via the column address. Accessing different rows within the same bank requires re-activation; by contrast, issuing multiple consecutive accesses within the same open row amortizes that overhead. As illustrated in Figure 15, we quantify row-level effects on HBM performance, where the x-axis denotes the number of bytes transferred after each row activation. Bank and BG placement also matter: Figure 16 and Figure 17 present analogous measurements when the accessed rows lie in different banks within the same BG and in different BGs, respectively. In addition, PCs can operate relatively independently; confining requests to a single PC severely limits overall throughput. As shown in Table 2, putting these factors together, the best-case access pattern—one that fully exploits spatial parallelism—achieves a total time of about 2.64 μs to transfer 1 MB. In the worst case, where all data reside in a single bank and each row holds only one minimum access granularity (thus forcing frequent activations), the transfer time rises to roughly 1.54 ms. Hence, the optimal pattern is 583$\times$ faster than the worst. In networking workloads, data typically arrives as packets. Assuming a packet size of 2,048 bytes, even with an optimal intra-packet data layout for a single packet—while disregarding the relative placement of other packets—the total access time can still reach 24 μs in the worst case; relative to the theoretical best case, this realizes only about $1/9$ of the peak performance.

Beyond address placement, frequent changes between reads and writes can drastically degrade effective bandwidth. In a Samsung 16 Gb HBM2E configuration, for accesses to an already-activated row, two identical commands (read-read or write-write) must wait 4 clock cycles between issuances ($t_{CK} = 0.625$ ns). In contrast, read-write switching requires a longer turnaround: read to write requires a 30-clock-cycles turnaround, while write to read requires about 7.5 ns. Consequently, read/write ping-pong reduces the effective bandwidth to roughly one-fifth of that under continuous reads or continuous writes.

Table 2: HBM Access Latency under Different Access Patterns

Case	Data Layout	Cell Size	#Rows	Total Latency
Best Performance	1MB distributed across 16 PCs, different bank groups	1KB	64	$\approx 2.64\ \mu$s
Worst Performance	1MB in the same PC, same bank, different rows	32B	32K	≈ 1.54 ms
Packet-granularity Worst	1MB in 16 PCs, same bank, packet size = 2048B split	128B	512	$\approx 24\ \mu$s

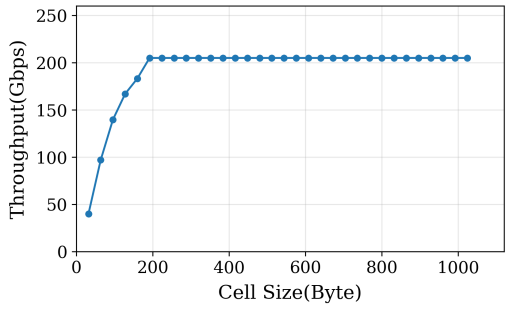


Figure 17: Multi-BG Read Throughput vs. Cell Size.

These observations imply that HBM's headline bandwidth can be realized only when access patterns satisfy all three of the following conditions:

1. Distribute data as evenly as possible across PCs and BGs to maximize spatial parallelism;
2. Read as much useful data as possible per row activation to amortize activation/precharge timing overheads;
3. Batch operations into sustained reads or sustained writes, avoiding frequent direction switches.

Only when these three conditions are met does HBM deliver a decisive bandwidth advantage over DDR.

Implementation in Themis. Themis performs batched accesses to HBM. Based on point 2, the ideal is to read or write the entire row after activation. In Samsung HBM2E, a row holds 1 KB. We stripe each packet across all 16 PCs; consequently, in our experiments we set the *batch size* to 16 KB.

B Design Considerations and Parameter Settings

On-Chip Dequeue Hit Rate. From the dequeue perspective, to better evaluate the performance of hybrid BM, we introduce *on-chip dequeue hit rate*, defined as the fraction of packets dequeued from on-chip SRAM. Since off-chip DRAM read bandwidth is limited, the on-chip dequeue hit rate reflects the utilization of the output link. In Themis, newly arriving packets are differentiated at admission based on their rank, so that the vast majority of dequeue operations can be served from on-chip memory. However, the on-chip dequeue hit rate in Themis cannot always reach 100%, mainly due to two reasons: (1) the swap-in speed from off-chip to on-chip cannot keep up with the on-chip dequeue rate, causing the number of on-chip packets to gradually decrease, possibly reaching zero, at which point dequeues must be served from off-chip; (2) swap-in is performed in units of batches. Each swap-in guarantees that the globally smallest-rank packet in DRAM is brought on-chip, but other packets in the same batch may have higher ranks. As a result, after the batch is swapped in, it is possible to violate the property that all on-chip packets have lower rank than all off-chip packets.

For reason (1), this phenomenon is unavoidable: once a packet is written into DRAM, off-chip dequeues will inevitably occur during certain periods. In a BM scheme that performs only admission control, every packet that enters DRAM must incur an off-chip access when dequeued. By contrast, Themis proactively swaps in packets and can guarantee that a certain number of packets can be dequeued on-chip instead of off-chip, given by

$$\left(\frac{Q_{\text{on}}}{R_{\text{s}}}\right) R_{\text{d}}$$

Here, Q_{on} is the number of remaining on-chip packets, R_{s} is the SRAM read rate, and R_{d} is the DRAM read rate.

For reason (2), as discussed in §4.2.3, the ranks of packets within the same batch are usually close; even if a few packets have particularly large ranks, they will be evicted or dropped once on-chip occupancy is high, thus preventing persistent blocking.

Parameter Settings. Themis contains two parameters: the swap-in watermark and the swap-out watermark, both expressed in terms of the occupancy of on-chip SRAM. These two watermarks characterize the degree to which off-chip DRAM participates in packet buffering. A lower watermark means that DRAM engages in storage earlier, providing higher absorption capability when burst traffic arrives and thus leading to fewer packet drops. However, during the draining phase, since the number of packets stored in SRAM is limited, the on-chip dequeue hit rate and output link bandwidth utilization are lower. In contrast, a higher watermark delays the involvement of DRAM, keeping more packets on-chip, which results in a higher dequeue hit rate but lower burst absorption capability and thus more packet drops. In addition, to avoid frequent migrations of packets between SRAM and DRAM, the gap between the two watermarks should not be too small.

Overall, the setting of the watermarks reflects a trade-off

between burst absorption capability and output link bandwidth utilization. In essence, the optimal configuration depends on the burst length. However, in practical systems, burst lengths are often difficult to predict. To balance burst absorption capability and output link bandwidth utilization across different scenarios, Themis adopts an empirical configuration. In §6, we present experimental results to demonstrate the impact of different parameter choices on performance.

C Additional Simulation Results under Varying Traffic Patterns

To comprehensively characterize the performance of Themis, we conduct simulations under different traffic patterns. Since Themis fundamentally makes decisions based on packet ranks, we predefine the priorities of input flows and vary the ratio of high- to low-priority traffic in the workload. The traffic consists of one high-priority flow and one low-priority flow, which are mixed according to the specified ratio. Packets are evenly spaced both within each flow and across flows. Specifically, when the ratio is n, for every n high-priority packets, one low-priority packet arrives on the link. We adopt strict-priority scheduling, while keeping the rest of the experimental setup identical to that in §6.3.1.

Figure 18 and 19 illustrate the impact of the ratio. As the ratio increases, both the on-chip dequeue hit rate and link utilization increase. This is because denser arrivals of high-priority packets lead to swap-in batches containing more high-priority packets. Consequently, each swap-in results in a greater number of on-chip hits. Themis selects a *batch size* number of the lowest-priority packets currently in SRAM to form a batch when triggering a swap-out. However, the absolute ranks of these packets are not necessarily close to each other, yet they will be swapped in together at some point in the future. Completely resolving this issue would require prior knowledge of the rank distribution of incoming traffic. This is clearly difficult for a general-purpose buffer manager. Nevertheless, even in low-ratio scenarios, Themis still demonstrates a significant improvement over other methods.

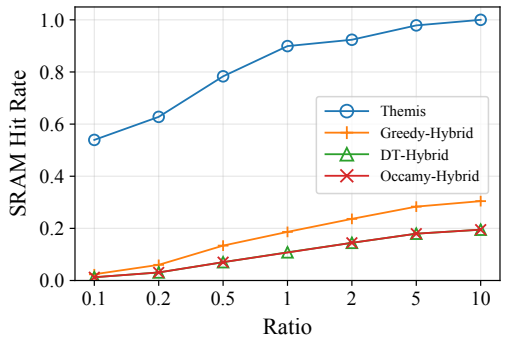


Figure 18: On-Chip Hit Rate vs. High-Priority Ratio.

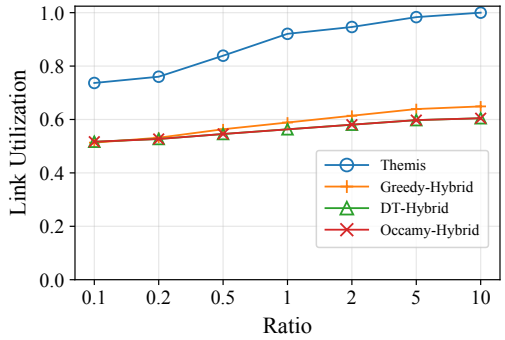


Figure 19: Link Utilization vs. High-Priority Ratio.

D Traffic Manager

Figure 8 (a) illustrates the architecture of TM. The TM is a fundamental component, tasked with managing packet buffering, ensuring scheduling prioritization to meet QoS requirements, and performing traffic statistical analysis. It consists of five primary modules:

- **Buffer Management:** BM is responsible for controlling the admission of incoming packets into the buffer. It determines whether a packet should be stored in on-chip or off-chip memory, based on predefined policies (e.g., per-flow threshold), ensuring optimal use of available resources.
- **Buffer Read/Write Controller:** Figure 8 (b) provides details of how the buffer is efficiently organized by the buffer r/w controller for packet storage and space release. It operates with three main types of memory: cell data memory, cell pointer memory, and PD memory. Cell data memory stores the actual packet data, divided into fixed-size cells. Cell pointer memory holds pointers to the cells in cell data memory. For packets split across multiple cells, the cell pointers are linked in a list. Free cell pointer list also tracks available memory cells by maintaining a list of free cell pointers. PD memory contains packet descriptors, each associated with a packet. A PD stores metadata (e.g., packet length) and the head of the corresponding cell pointer list.
- **Buffer:** The buffer holds the actual packet data, either in on-chip memory (e.g., SRAM) for fast access or off-chip memory (e.g., DRAM) for larger storage capacity.
- **Scheduler:** The scheduler determines the order in which packets are dequeued from the buffer.
- **Statistics:** The statistics module is typically used to collect real-time information (e.g., queue lengths) to assist

BM in making strategy.

E BBQ

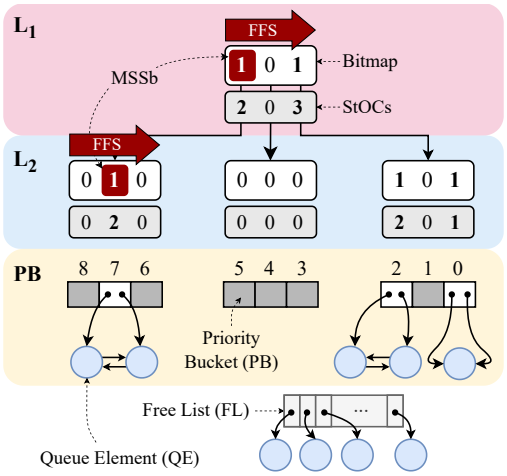


Figure 20: Architecture of BBQ.

BBQ is a hardware-based integer priority queue designed for scalable and efficient packet scheduling. It addresses the challenge of handling large numbers of concurrent flows at high throughput and line rates. BBQ employs a bitmap-based hierarchical priority queue, offering significant improvements in scalability and performance over traditional comparison-based queues.

At its core, BBQ uses a bitmap tree structure to organize priority buckets. Each bit represents a bucket, and the queue leverages a FFS operation to efficiently locate the highest-priority bucket containing a packet, eliminating the need for comparison-based sorting. This hierarchical structure allows for efficient lookups, enabling fast and scalable scheduling.

Packets are enqueued by inserting them into the corresponding priority bucket based on their priority value. The packet is added to a doubly-linked list in the bucket, and the corresponding bitmap bit is updated. This simple operation ensures the queue scales efficiently with increasing flows and priorities.

Dequeueing begins at the root of the bitmap tree. BBQ uses FFS to identify the highest-priority bucket and dequeues the packet from the linked list, updating the bitmap accordingly. This approach allows BBQ to dequeue packets at line rates, ensuring high throughput without compromising accuracy.

BBQ supports logical partitioning, allowing multiple logical queues to share a single physical priority queue. This is done by partitioning the priority range and masking relevant bits in the bitmap tree for each logical queue. Logical partitioning is crucial in multi-tenant environments, such as cloud-based SmartNICs or modern switches, where it enables independent management of different tenants or flows with minimal performance overhead.

BBQ is highly efficient in hardware, supporting up to 100K concurrent flows and 32K priority levels on both FPGAs and ASICs. Its fully pipelined architecture allows enqueue and dequeue operations in a single clock cycle, optimizing throughput and minimizing latency. BBQ is ideal for high-speed networking hardware, such as switches and SmartNICs, handling large volumes of network traffic with scalable and efficient packet scheduling.

F Hardware Pipeline of Themis

In principle, enqueueing, dequeueing, and swap-in/out follow a similar blueprint, yielding an intuitive algorithm for mapping them to a unified datapath: for each level of the tree starting with the root (i.e., L1), compute the bitmap index (for DEQUEUE(t), the index is computed by performing FFS on the bitmap; for ENQUEUE(X, p), it is computed using simple bit manipulations on p; for swap-in/out, the same index computation is applied on the corresponding tier), increment or decrement the corresponding StOC, then update the bitmap; finally, splice at the appropriate end of the doubly linked list corresponding to the target priority bucket.

Maximizing $f_{\max}$. Our first objective is to maximize $f_{\max}$, the maximum clock frequency at which the design can operate. To this end, we reuse the existing BBQ structure and employ a deep pipeline in which stages are balanced to do little and roughly equal amounts of work. Figure 21 shows the cycle-level events in an 11-stage pipeline for a two-level design. By load balancing expensive operations across stages, we minimize the number and severity of same-stage dependencies (depicted by $\rightarrow$ and $\rightsquigarrow$); for example, chaining *FFS* and StOC updates (multi-bit addition or subtraction) would create large combinational delay, so we split this work across stages (e.g., cycles 1 and 2).

Maximizing operations per cycle. High $f_{\max}$ is only useful if we are not rate-limited by pipeline latency. Our second objective is to fully pipeline the design so it can concurrently process as many operations as there are pipeline stages, thereby achieving its maximum rate of 1 op/cycle. Concretely, BBQ adopt a Hierarchical Find-First-Set (HFFS) queue, write forwarding and data forwarding, operation coloring, and pipelined bitmap/StOC updates. Themis reuses this structure and makes specific modifications: during swap-in (migrating the highest-priority packets from HBM to SRAM), Themis performs a dual-tier co-commit—at each level, it simultaneously updates the SRAM-side bitmap/StOC and the HBM-side bitmap/StOC in the same pass—thus eliminating a second top-down walk and avoiding transient cross-tier inconsistencies. Symmetrically, during swap-out (migrating the lowest-priority packets from SRAM to HBM), Themis applies the same dual-tier co-

commit: the SRAM- and HBM-side bitmaps/StOCs are updated in the same top-down pass, preserving per-level atomicity and avoiding a second traversal. In conclusion, Themis can perform one operation per cycle: packet-in, dequeue, swap-in, and swap-out.

Cycle	Description	PHR

0	Register inputs
	If_ENQUEUE:
	$F \leftarrow$ *SRAM*_FreeList.POP()// Pop free list
	If_SWAP_IN:
	$F \leftarrow$ *SRAM*_FreeList.POP()// Pop free list
	If_SWAP_OUT:
	$F \leftarrow$ *HBM*_FreeList.POP()// Pop free list	

1	IF ENQUEUE or DEQUEUE or DROP: Compute *SRAM* L_1 bitmap index ↳Read the corresponding *SRAM* L_1 StOC IF SWAP_IN: Compute *HBM* L_1 bitmap index ↳Read the corresponding *SRAM and HBM* L_1 StOC IF SWAP_OUT: Compute *SRAM* L_1 bitmap index ↳Read the corresponding *SRAM and HBM* L_1 StOC	L1
2	According Action: Compute, Write:*(SRAM | HBM)* L_1 StOC ⇝ L_1 bitmap Read *(SRAM | HBM)* L_2 bitmap	L2
3	// Read delay for L_2 bitmap	
4	IF ENQUEUE or DEQUEUE or DROP: Compute *SRAM* L_2 bitmap index ↳Read the corresponding *SRAM* L_2 StOC IF SWAP_IN: Compute *HBM* L_2 bitmap index ↳Read the corresponding *SRAM and HBM* L_2 StOC IF SWAP_OUT: Compute *SRAM* L_2 bitmap index ↳Read the corresponding *SRAM and HBM* L_2 StOC	
5	// Read delay for L_2 StOC	
6	According Action: Compute, Write: *(SRAM | HBM)* L_2 StOC ⇝ L_2 bitmap Read the corresponding *(SRAM | HBM)* PB	PB
7	// Read delay for PB	
8	If_Not ENQUEUE: (a)Read $X \leftarrow$ *(SRAM or HBM)*QE_{DATA}[PB.TAIL^{new}] (b)Read $Y \leftarrow$ *(SRAM or HBM)*QE_{PREV}[PB.TAIL^{new}]	
9	// Read delay for QE_{DATA} and QE_{PREV}	
10	If_ENQUEUE: // Enqueue at the HEAD (a)*SRAM*_$QE_{DATA}[F] \leftarrow$ Data to enqueue (b)Write *SRAM*_$QE_{NEXT}[F] \leftarrow$ *SRAM*_PB.HEAD (c)Write *SRAM*_QE_{PREV}[PB.HEAD] $\leftarrow F$ (d)Write *SRAM*_PB.HEAD$^{new} \leftarrow F$ If_DEQUEUE or DROP: // Dequeue from TAIL (a)*SRAM*_FreeList.PUSH(PB.TAIL) (b)Write *SRAM*_PB.TAIL$^{new} \leftarrow Y$ (c)Output X (If Drop , No Action) If_SWAP_IN: // Data From HBM To SRAM (a)*SRAM*_$QE_{DATA}[F] \leftarrow$ X (b)Write *SRAM*_$QE_{NEXT}[F] \leftarrow$ *SRAM*_PB.HEAD (c)Write *SRAM*_QE_{PREV}[PB.HEAD] $\leftarrow F$ (d)Write *SRAM*_PB.HEAD$^{new} \leftarrow F$ (e) *HBM*_FreeList.PUSH(PB.TAIL) (f)Write *HBM*_PB.TAIL$^{new} \leftarrow Y$ If_SWAP_OUT: // Data From SRAM To HBM (a)*HBM*_$QE_{DATA}[F] \leftarrow$ X (b)Write *HBM*_$QE_{NEXT}[F] \leftarrow$ *HBM*_PB.HEAD (c)Write *HBM*_QE_{PREV}[PB.HEAD] $\leftarrow F$ (d)Write *HBM*_PB.HEAD$^{new} \leftarrow F$ (e) *SRAM*_FreeList.PUSH(PB.TAIL) (f)Write *SRAM*_PB.TAIL$^{new} \leftarrow Y$	

Figure 21: Pipeline of Themis.