Reverie: Low Pass Filter-Based Switch Buffer Sharing for Datacenters with RDMA and TCP Traffic

Vamsi Addanki
TU Berlin

Wei Bai
Microsoft Research

Stefan Schmid
TU Berlin

Maria Apostolaki
Princeton University

Abstract

The switch buffers in datacenters today are dynamically shared by traffic classes with different loss tolerance and reaction to congestion signals. In particular, while legacy applications use loss-tolerant transport, e.g., DCTCP, newer applications require lossless datacenter transport, e.g., RDMA over Converged Ethernet. Unfortunately, as we analytically show in this paper, the buffer-sharing practices of today's datacenters pose a fundamental limitation to effectively *isolate* RDMA and TCP while also maximizing *burst absorption*. We identify two root causes: *(i)* the buffer-sharing for RDMA and TCP relies on two independent and often conflicting views of the buffer, namely ingress and egress; and *(ii)* the buffer-sharing scheme micromanages the buffer and overreacts to the changes in its occupancy during transient congestion.

In this paper, we present REVERIE, a buffer-sharing scheme, which, unlike prior works, is suitable for both lossless and loss-tolerant traffic, providing isolation and better burst absorption than state-of-the-art buffer-sharing schemes. At the core of REVERIE lies a unified (consolidated ingress and egress) admission control that jointly optimizes the buffers for both RDMA and TCP. REVERIE allocates buffer based on a low-pass filter that naturally absorbs bursty queue lengths during transient congestion within the buffer limits. Our evaluation shows that REVERIE can improve the performance of RDMA as well as TCP in terms of flow completion times by up to 33%.

1 Introduction

Network devices contain a buffer that can temporarily store excessive packets during congestion events. As the link speeds increase, maintaining a constant buffer-bandwidth ratio would require buffer memory to evolve faster than Moore's law and is hence impractical [33]. As a result, we observe buffer-per-Gbps to constantly shrink [12, 26] making performance problems rooted in buffer sharing more evident. Indeed, our expert survey, which included experts from six companies, revealed that buffer sharing is causing performance problems in most large-scale datacenters.

At a high level, the goal of a buffer-sharing scheme is to provide isolation between traffic classes, while maximizing the benefit of the buffer e.g., by absorbing bursts and achieving high throughput. Existing buffer management schemes (even recent ones) [1, 8, 15, 25] were designed considering exclusively loss-tolerant traffic (e.g., TCP variants). However, modern datacenters host traffic classes with different loss tolerance. Concretely, along with traditional loss-tolerant transport protocols, many clouds, e.g., Azure [11], Alibaba [22] and OCI [38], deploy RDMA over Converged Ethernet which requires lossless transport. In order to guarantee zero packet loss for RDMA, production datacenters enable Priority Flow Control (PFC) at the switches [11].

The co-existence of TCP and RDMA traffic in the switch buffer makes sharing the buffer particularly challenging. While, in principle, TCP and RDMA traffic have the same performance objectives (e.g., high throughput, low latency), their reaction to network events such as congestion is vastly different in terms of speed and granularity. A PFC pause proactively throttles RDMA traffic at *per-hop* granularity *before* the buffer fills up in order to prevent packet loss due to congestion. On the contrary, a packet drop throttles TCP at *per-flow* granularity once the buffer is filled up due to congestion. Moreover, the effect of PFC pause (in RDMA) on the buffer is not immediately evident as all incoming packets after the PFC has been triggered must be admitted in the buffer further increasing its occupancy. On the contrary, the effect of a packet drop (in TCP) on the buffer is immediately evident in the buffer as current packets do not further increase the buffer occupancy which can decrease proportionately to the aggregate port bandwidth. Since RDMA and TCP share the same switch buffer, congestion caused by TCP can result in excessive PFC pauses for RDMA and similarly the buffer occupied by RDMA (especially when it is paused) can result in excessive packet drops for TCP; leading to throughput degradation and poor burst absorption.

A naive approach for isolating RDMA and TCP in the shared buffer is to statically partition it e.g., dedicate 50% of the buffer to each class. However, such an approach will result in suboptimal burst absorption; and (in the worst case) poor throughput

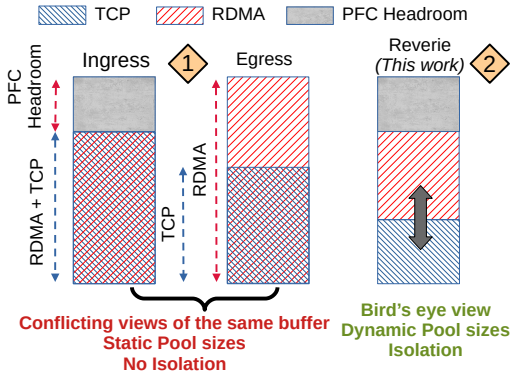


Figure 1: 1 Current buffer-sharing practices maintain two independent and at times conflicting views of the buffer i.e., ingress and egress, which are subdivided into various pools; posing a challenge to achieve isolation across RDMA and TCP traffic. 2 REVERIE maintains a bird's eye view of the buffer; effectively unifying ingress and egress admission control *without statically partitioning the buffer* to achieve isolation.

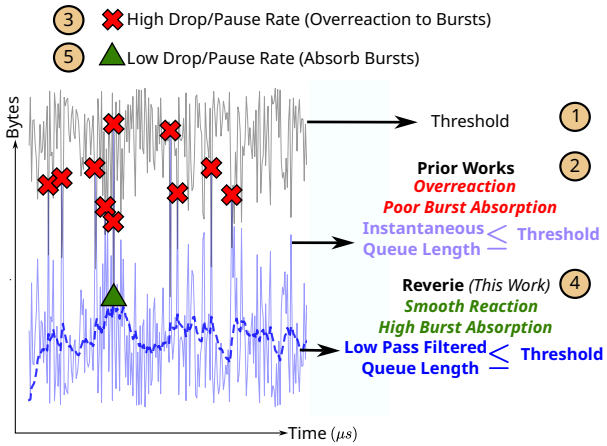


Figure 2: Prior works calculate 1 thresholds and compare against 2 instantaneous queue lengths, which leads to 3 overreaction to bursts and a high loss rate. REVERIE takes a different approach and compares the thresholds against 4 low pass filtered queue lengths. This allows REVERIE to 5 smoothly react to congestion while absorbing transient bursts.

when one of the two classes is not using its dedicated portion of the buffer. On the one hand, production-grade buffer-sharing schemes do not significantly depart from static partitioning among TCP and RDMA. The root cause of this pitfall is the unnecessary complex buffer model together with the use of pre-configured buffer pools i.e., pieces of buffer dedicated to certain queues only. On the other hand, research-grade buffer-sharing schemes such as ABM [1] can –at best– achieve steady-state isolation across traffic priorities only for loss-tolerant traffic, but would fail to isolate lossless and lossy traffic, even if we extend them to work in such settings, as we show in §2.3. Our goal in this paper is to formally navigate the trade-off between isolation and burst absorption in a setting where lossless and lossy traffic co-exist. Two key insights allow us to do so.

Our first key insight –after thoroughly studying the current buffer-sharing practices– is that although lossless and lossy traffic are, in practice, independently managed, the available buffer for both depends on each other's occupancy. Concretely, today's switches maintain two views of the buffer (i.e., ingress and egress); each of these views is virtually further split into pools i.e., buffer pieces configured to serve a subset of the queues. Figure 1 summarizes these views. The complexity of the buffer design stems from its evolution over the years from serving lossy traffic to serving both lossy and lossless. Unfortunately, as we show in this paper, this complex buffer design of today's datacenter switches leads to unexpected buffer issues e.g., lossy traffic gets more buffer allocation than lossless traffic when they compete, against the high-level objective of the configuration. Our analysis shows that buffer pools and the *independent* views of the buffer at the ingress and egress are the root causes of such issues. To tackle this problem, we propose a simple buffer-sharing scheme in which both RDMA and TCP are managed *jointly* with a bird's eye view of the buffer. Such an allocation scheme facilitates novel admission control schemes that can efficiently isolate RDMA and TCP without statically partitioning the buffer.

Our second key insight is that absorbing RDMA bursts is extremely challenging because the decisions of a buffer-sharing scheme are on a per-packet basis but PFC pause (required for RDMA) is a per-hop signal affecting not just the incoming packet but all the future arrivals from the previous hop. Moreover, sudden and concurrent fluctuations in multiple queue lengths can rapidly change the buffer occupancy. Worse yet, the incoming rate at today's link speeds can very rapidly fill in a buffer. Existing schemes [1, 8, 25] that were designed specifically for loss-tolerant traffic apply large thresholds to those packets that are classified as short flows or incast flows. Unfortunately, as we later show in this paper, these techniques cannot fundamentally achieve better burst absorption for lossless traffic since PFC works at *per-hop* or per-queue granularity and not per-packet i.e., burst absorption for RDMA requires prioritizing a queue experiencing burst (not just specific packets). To address this problem, we show that instead of increasing thresholds at per-packet granularity under bursty scenarios, it is sufficient to dampen the queue statistic (e.g., by a low-pass filter) against which thresholds are compared, when taking buffer decisions. Figure 2 illustrates our main idea. This essentially prioritizes queues experiencing bursts and improves the burst absorption capabilities of the buffer.

We present REVERIE, a buffer-sharing scheme suitable for modern datacenters hosting traffic with different loss tolerance. REVERIE jointly optimizes the buffer allocation for lossless and lossy traffic with a bird's eye view over the buffer; essentially unifying ingress and egress admission control as shown in Figure 1. Further, REVERIE significantly improves the burst absorption capabilities of the buffer by

comparing low-pass-filtered queue lengths against thresholds as illustrated in Figure 2.

Our extensive evaluation of REVERIE based on large-scale simulations in NS3 shows that REVERIE effectively isolates RDMA and TCP, reduces the overall number of PFC pauses by 60% on average and improves the flow completion times for bursty workloads by up to 33% compared to the state-of-the-art approaches.

In summary, our key contributions in this work are:

- The first analysis of a production-grade buffer model (exemplified by the open source SONiC [44]) that includes both ingress and egress admission control. This analysis generates multiple insights including the conflicting buffer views of ingress and egress that prevent effective isolation between lossless and lossy traffic.
- REVERIE, the first buffer-sharing scheme that can isolate lossless and lossy traffic while improving burst absorption for both.
- As a contribution to the research community and to facilitate future work, all our artifacts have been made publicly available at https://github.com/inet-tub/ns3-datacenter.

2 Motivation

We first present unintuitive outcomes (issues) that arise under typical configurations (§2.1). To explain the root cause of those issues (§2.3), we first describe in detail a representative buffer-sharing architecture of an open-source and widely-used switch OS, i.e., SONiC [44] (§2.2). Our buffer model has been endorsed by major Ethernet switch ASIC vendors, including Broadcom, Cisco and NVIDIA.

2.1 Buffer Issues in Datacenters

In this section, we walk through three issues that operators of large-scale RDMA deployments [11] can face while debugging buffer problems. We have verified that these issues are *(i)* possible by showing them analytically (§2.3); and *(ii)* realistic by direct communications with operators of large-scale RDMA deployments. Consider an operator who wants lossless traffic to get as much buffer as it needs, i.e., lossless is prioritized over lossy traffic under buffer contention. This is a typical use case in datacenters with large RDMA deployments. Although the operator closely follows the "best practices" (i.e., a set of heuristics) to configure the buffer, which we explain in §2.2.2, they observe the following issues.

Issue 1. *Lossy traffic gets more buffer allocation than lossless traffic when they both compete for buffer space.*

Issue 2. *Lossless traffic yields to the increase in buffer occupancy of lossy traffic, while the opposite is not true, i.e., the allocation for lossy traffic is not affected by the buffer occupancy of lossless traffic.*

Issue 3. *The buffer is more efficient in absorbing bursts of lossy traffic than bursts of lossless traffic.*

2.2 Buffer Sharing Practices

To understand the root cause of the issues (§2.3), we need to understand the buffer model used in today's datacenters and the "best practices" for configuring it. To the best of our knowledge, we are the first to present a detailed and up-to-date description of a buffer model of a datacenter switch that serves both lossless (e.g., RDMA) and lossy (e.g., TCP) traffic. We use SONiC [44], an open-source network operating system that is the closest we can get to the modus operandi for buffer management. SONiC runs on switch ASICs from multiple vendors, e.g., Broadcom, NVIDIA, Cisco and Intel, and has been widely deployed in Microsoft [11], Alibaba [48], LinkedIn [51] and Tencent. Importantly, our buffer model aligns with that of NVIDIA Onyx [47]. Hence, we believe our buffer model is representative of a broad range of scenarios and settings. We next describe the terminology and the configurable parts according to the buffer model of SONiC. We tabulate the important notations we use in Table 1.

Hereafter, we denote lossless by $\bullet$ and lossy by $\circ$. The switch uses a memory management unit (MMU) to manage the packet buffer.

Ingress and Egress Counters (Queues): The MMU maintains two types of *counters*[1], ingress denoted by $\leftarrow$ and egress denoted by $\rightarrow$ that serve admission control purposes. We henceforth refer to these counters as *queues*. Let Q be the set of all queues maintained by the MMU. Once a packet arrives at the switch, the packet is mapped to an ingress queue $(s,p) \in \overleftarrow{Q}$ based on the source port s and the packet's priority p[2]; and an egress queue $(d,p) \in \overrightarrow{Q}$ based on the destination port d. Ingress (egress) admission control acts over ingress (egress) queues. Each packet is admitted to the buffer if and only if the corresponding ingress and egress queues pass the ingress and egress admission controls. An arriving and admitted packet increases both the corresponding ingress and egress queues, while a departure packet decreases the queues. A packet is only buffered once regardless of the number of counters it is accounted by. Note that once a packet is admitted, it cannot be *pushed out* by new packet arrivals. A queue carrying lossless (lossy) traffic is known as a lossless (lossy) queue. Overall, the MMU maintains four sets of queues i.e., ingress lossless $\overleftarrow{Q}^{\bullet}$, ingress lossy $\overleftarrow{Q}^{\circ}$, egress lossless $\overrightarrow{Q}^{\bullet}$ and egress lossy $\overrightarrow{Q}^{\circ}$.

Buffer Size and Pools: The packet buffer has a total size of b. Current datacenter practices define pools that can intuitively be viewed as the buffer available for certain types of queues. In other words, the pool is a group of queues. The user can configure the buffer allocation policy, including the allocation algorithm, per-queue limit, and total size, for this group of queues. SONiC defines the following four pools:

- Ingress pool of size $\overleftarrow{b}$ shared by *both* ingress lossless and lossy queues, with an occupancy of $\overleftarrow{q}(t)$ at time t.

[1] Not to be confused with ingress and egress pipelines. Throughout this paper, ingress and egress are merely counters (referred to as queues).

[2] Most of the switch ASICs support 8 priorities. Operators typically map a packet to a priority based on its DSCP value.

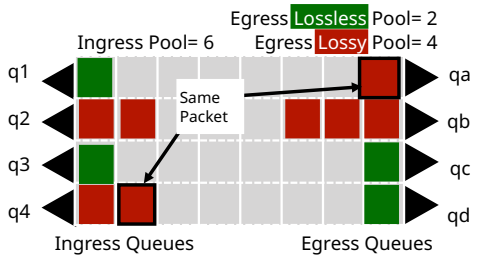


Figure 3: Buffer bookkeeping: packets are stored once but accounted twice; once in the ingress and once at the egress for admission control purposes.

- PFC headroom pool of size b_h used *only* by ingress lossless queues upon PFC pause (described next).
- Egress lossless pool of size $\overset{\bullet}{\overrightarrow{b}}$ used by egress lossless queues, with an occupancy of $\overset{\bullet}{\overrightarrow{q}}(t)$ at time t.
- Egress lossy pool of size $\overset{\circ}{\overrightarrow{b}}$ used by egress lossy queues, with an occupancy of $\overset{\circ}{\overrightarrow{q}}(t)$ at time t.

Note that pool sizes and occupancy are also *counters*. A packet can be counted in multiple pools while being buffered once (pools may overlap), thus the sum of all pools may exceed the actual buffer occupancy.

Figure 3 illustrates an example of the buffer bookkeeping. Packets are physically stored only once, but are accounted twice. For example, packet 5 is accounted in the ingress queue (counter) $q4$ and in the egress queue (counter) qa. In essence, all packets are accounted in the ingress pool, but packets of lossless queues are also accounted in the egress lossless pool, while packets of lossy queues are also accounted in the egress lossy pool.

Admission Control: Each queue i.e., counter $(i,p) \in Q$ at an input or output port denoted by i, corresponding to priority p is associated with a *threshold* $\Gamma_i^p(t)$ at time t. The admission control scheme compares the instantaneous length $q_p^i(t)$ of a queue against its threshold to make buffering decisions. Thresholds can be intuitively viewed as the maximum size of a queue. Once the queue hits the threshold, the switch will drop the incoming packet or send PFC pause frames to throttle the queue build-up. We emphasize that the switch cannot push out existing packets in the buffer to make room for the incoming packet.

2.2.1 Journey of a Packet in the Switch MMU

We walk through the various counters that are incremented and decremented during a packet's journey in the MMU. Recall that a packet can travel through the switch if and only if it satisfies both ingress and egress admission control.

Ingress Admission Control: The admission control in the ingress is different for lossy and lossless queues since as TCP (lossy) tolerates packet loss whereas RDMA (lossless) requires PFC and does not tolerate packet loss. The admission for ingress lossy queues is straightforward. If the ingress lossy queue hits the threshold (meaning if the length of the queue equals or exceeds the corresponding threshold devised by the buffer-sharing logic), the packet is dropped. Otherwise, both the ingress lossy queue and ingress pool counters are incremented upon admission and decremented as the packet departs to its destination.

In contrast, the admission control for ingress lossless queues is more complex and designed to achieve zero packet loss. If the ingress queue hits the threshold, the switch moves the queue to "paused" state and keeps sending PFC pause frames to the peer device. Then the arriving packet is admitted, but it increments the PFC headroom pool occupancy rather than the ingress pool occupancy. In other words, once an ingress lossless queue uses up its limit in the ingress pool, it starts to consume (or be accounted in) the PFC headroom pool. As the buffer drains, an ingress lossless queue under "paused" state first decrements its headroom pool occupancy, and then its ingress pool occupancy. When the headroom buffer occupancy is zero and the ingress pool occupancy is below the threshold, the switch moves the "paused" ingress lossless queue back to "resumed" state and sends PFC resume frames.

Egress Admission Control: Egress counters are straightforward. Egress queue length and pool occupancy based on the class of packet (lossy or lossless) are incremented upon admission and decremented as the buffer drains. The switch drops packets if egress queues hit thresholds.

2.2.2 Buffer Management

The MMU of the switch uses a buffer management algorithm that assigns thresholds to all ingress and egress queues. Dynamic Thresholds [15] (DT) is the state-of-the-art buffer management algorithm widely adopted by ASIC vendors [37, 45].

In a nutshell, DT calculates *dynamic* thresholds for each queue $(i,p) \in Q$ as the product of a configurable parameter α_p^i and the *remaining* buffer space in the corresponding pool. We refer the reader to Table 1 for the list of notations we use. In the following, we summarize DT's buffer management for lossless and lossy traffic, at ingress and egress queues.

$$\Gamma_p^i(t) = \alpha_p^i \times \begin{cases} \overleftarrow{b} - \overleftarrow{q}(t) & \textit{Ingress Lossless: } (i,p) \in \overleftarrow{Q}^{\bullet} \\ \overleftarrow{b} - \overleftarrow{q}(t) & \textit{Ingress Lossy: } (i,p) \in \overleftarrow{Q}^{\circ} \\ \overset{\bullet}{\overrightarrow{b}} - \overset{\bullet}{\overrightarrow{q}}(t) & \textit{Egress Lossless: } (i,p) \in \overset{\bullet}{\overrightarrow{Q}} \\ \overset{\circ}{\overrightarrow{b}} - \overset{\circ}{\overrightarrow{q}}(t) & \textit{Egress Lossy: } (i,p) \in \overset{\circ}{\overrightarrow{Q}} \end{cases} \quad (1)$$

In addition to the above threshold checks, the switch also uses the physical packet buffer limit as the last defense.

Having a better understanding of how the buffer works, we can go back to the operator's goal to prioritize lossless over lossy and explain how they would in practice configure the buffer. To avoid lossless packet drops, they would want to control lossless traffic *only* at the ingress with PFC thresholds. To make the problem easier to debug and more intuitive, they would want to control lossy traffic *only* at the egress with drop

Notation	Description
$\leftarrow$	Ingress
$\rightarrow$	Egress
$\bullet$	Lossless
$\circ$	Lossy
$*$	Shared
b	Total buffer size
b_h	Headroom pool size
Q	Set of all queues
$\overleftarrow{Q}^{\bullet}$	Set of ingress lossless queues
(i,p)	Queue at input or output port i, priority p
$q_i^p(t)$	Length of queue (i,p) at time t
$\Gamma_i^p(t)$	Threshold of queue (i,p) at time t
$\overrightarrow{q}^{\circ}(t)$	Occupancy of egress lossy pool
α_i^p	Parameter for queue (i,p)
$\overleftarrow{\alpha}^{\bullet}$	α for ingress lossless queues (for simplicity)
$\overrightarrow{\alpha}^{\circ}$	α for egress lossy queues (for simplicity)
$\overleftarrow{n}^{\bullet}$	# ingress lossless queues using buffer
$\overrightarrow{n}^{\circ}$	# egress lossy queues using buffer

Table 1: Important notations used in this paper.

thresholds. To this end, they would use the following buffer configuration heuristics. For simplicity, in this section, we set α_p^i to $\overleftarrow{\alpha}^{\bullet}$, $\overleftarrow{\alpha}^{\circ}$, $\overrightarrow{\alpha}^{\bullet}$ and $\overrightarrow{\alpha}^{\circ}$ for all ingress lossless, ingress lossy, egress lossless and egress lossy queues respectively.

Heuristic 1. *To avoid packet drops for lossless traffic at ingress, the sum of the ingress pool size $\overleftarrow{b}$ and headroom pool size b_h should be equal to (or smaller than) the total size of the switch buffer b i.e., $\overleftarrow{b}+b_h \leq b$.*

Heuristic 2. *To bypass egress admission control for lossless traffic and to allow fully utilizing the buffer space i.e., to avoid lossless packet drops at egress, we should set egress lossless pool size $\overrightarrow{b}^{\bullet}$ to total switch buffer size b and use an infinitely large egress lossless threshold i.e., $\overrightarrow{\alpha}^{\bullet} \gg 1$.*

Heuristic 3. *To avoid packet drops for lossy traffic at ingress, we should set ingress lossy threshold $\overleftarrow{\alpha}^{\circ}$ to an infinitely large value, and ensure that egress lossy pool size $\overrightarrow{b}^{\circ}$ is not larger than ingress pool size $\overleftarrow{b}$ i.e., $\overrightarrow{b}^{\circ} \leq \overleftarrow{b}$.*

2.3 Root Causes of the Buffer Issues

To systematically analyze the problems, we consider a fluid flow model with deterministic packet arrivals and analyze the steady state[3] of the buffer, similar to prior works [1, 15]. In the following, we explain our key findings intuitively. Our model and complete analysis can be found in Appendix A.

First, based on Heuristic 1, lossless traffic is allowed to use the entire buffer at the ingress i.e., $\overleftarrow{b}+b_h \leq b$. Further, lossless traffic is allowed to use the entire buffer at the egress i.e., $\overrightarrow{b}^{\bullet}=b$ based on Heuristic 2. Still, we end up with Issue 1.

[3] A steady state is achieved when the queue lengths stabilize i.e., packet arrival rate equals departure rate.

Root cause of Issue 1: To shed light on this issue, we analytically derive the aggregate buffer allocation $\overleftarrow{q}^{\bullet}$ to lossless queues and the aggregate buffer allocation $\overrightarrow{q}^{\circ}$ to lossy queues. Let $\overleftarrow{n}^{\bullet}$ and $\overrightarrow{n}^{\circ}$ denote the number of ingress lossless queues and egress lossy queues using the buffer respectively. We have:

$$\overleftarrow{q}^{\bullet} = \overleftarrow{b} \cdot \left(\frac{\overleftarrow{n}^{\bullet} \cdot \overleftarrow{\alpha}^{\bullet}}{1+\overleftarrow{n}^{\bullet} \cdot \overleftarrow{\alpha}^{\bullet}} \right) - \overrightarrow{b}^{\circ} \cdot \left(\frac{\overleftarrow{n}^{\bullet} \cdot \overleftarrow{\alpha}^{\bullet}}{1+\overleftarrow{n}^{\bullet} \cdot \overleftarrow{\alpha}^{\bullet}} \cdot \frac{\overrightarrow{n}^{\circ} \cdot \overrightarrow{\alpha}^{\circ}}{1+\overrightarrow{n}^{\circ} \cdot \overrightarrow{\alpha}^{\circ}} \right) \quad (2)$$

$$\overrightarrow{q}^{\circ} = \frac{\overrightarrow{n}^{\circ} \cdot \overrightarrow{\alpha}^{\circ} \cdot \overrightarrow{b}^{\circ}}{1+\overrightarrow{n}^{\circ} \cdot \overrightarrow{\alpha}^{\circ}} \quad (3)$$

Figure 4 illustrates the ratio of buffer allocated to lossy and lossless. Notice that for a sufficiently large number of lossless queues $\overleftarrow{n}^{\bullet}$ and lossy queues $\overrightarrow{n}^{\circ}$, the buffer allocation to lossless queues tends to $\overleftarrow{b} - \overrightarrow{b}^{\circ}$ (based on Equation 2) and the allocation for lossy queues tends to $\overrightarrow{b}^{\circ}$ (based on Equation 3). Unless $\overleftarrow{b} \geq 2 \times \overrightarrow{b}^{\circ}$, we end up with Issue 1, caused by the buffer pools. Specifically, although lossless queues are allowed to fully utilize the buffer, the egress lossy pool *effectively* overlaps with both ingress pool and egress lossless pools as shown in Figure 1, leading to Issue 1.

■ ***Takeaway.*** *The buffer is pre-fragmented in a way that makes enforcing high-level objectives through low-level configuration impossible. Current buffer sharing practices cannot prevent Issue 1 unless the ingress pool is at least twice as large as the egress lossy pool i.e., the buffer is statically partitioned.*

Second, according to Heuristic 2 and Heuristic 3, since lossless (lossy) bypasses egress (ingress) admission control, we would expect that lossless and lossy traffic are isolated in the buffer. While Issue 1 already suggests that lossy traffic may effectively get more buffer allocation than lossless traffic, we find yet another issue that lossy and lossless traffic interact in a surprisingly unfair manner: lossy traffic is effectively prioritized over lossless traffic (Issue 2) although our expert heuristics are intended otherwise.

Root cause of Issue 2: Notice that the buffer occupancy of lossy traffic at the egress equals its occupancy at the ingress i.e., $\overrightarrow{q}^{\circ} = \overleftarrow{q}^{\circ}$, since every packet is accounted both in the ingress and egress as shown in Figure 3. As a result, the overall ingress pool occupancy is the sum of egress lossy occupancy and the ingress lossless occupancy i.e., $\overleftarrow{q} = \overrightarrow{q}^{\circ} + \overleftarrow{q}^{\bullet}$. Using Equations 2 and 3, as well as the above relation, we derive the steady-state thresholds for ingress lossless (PFC thresholds) and egress lossy (drop thresholds) based on Equation 1.

Figure 5a shows how PFC thresholds for lossless queues vary depending on the number of ingress lossless queues using the buffer and the number of egress lossy queues. Interestingly, we find that the drop thresholds vary only according to the number of egress lossy queues (affected by the own buffer occupancy), but remain unchanged as the number of ingress lossless queues increases, see Figure 5b. We observe that current buffer sharing practices allow buffering lossy packets mostly independently of lossless traffic, but lossless traffic is

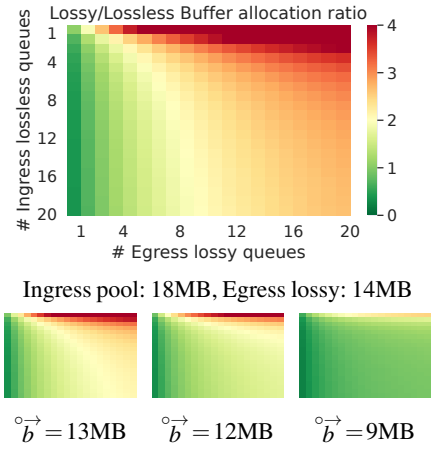
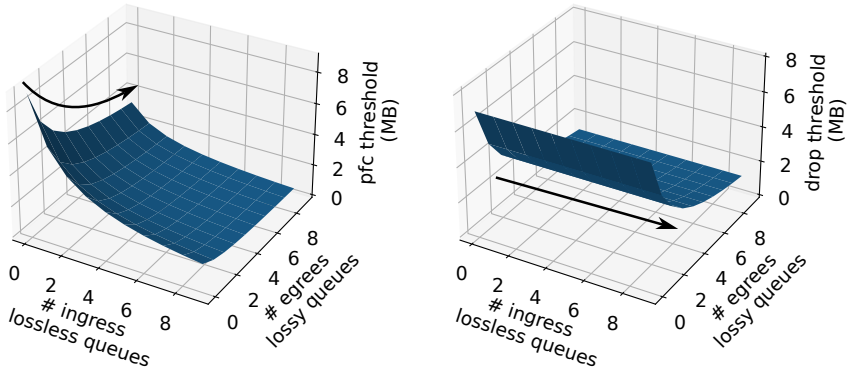


Figure 4: Isolation cannot be achieved with SONiC unless the ingress pool size $\overleftarrow{b}$ = 18MB is at least twice as large as egress lossy pool size $\overrightarrow{b}$ i.e., statically partitioning the buffer.



(a) PFC thresholds (for ingress lossless queues) are affected by both the number of ingress lossless and egress lossy queues.

(b) Drop thresholds (for egress lossy queues) are affected only by lossy queues.

Figure 5: Contrary to the expectation that lossless and lossy traffic are admitted independently in the buffer, lossless traffic is throttled due to the buffer occupancy of lossy, whereas lossy is admitted independent of the presence of lossless i.e., seemingly prioritizing lossy over lossless.

suppressed due to lossy traffic i.e., effectively prioritizing lossy over lossless. Our analysis reveals two root causes of Issue 2: *(i)* the egress lossy pool occupancy $\overrightarrow{q}$ which is used to calculate egress lossy drop thresholds (see Equation 1) does not account for lossless traffic whereas *(ii)* the ingress pool occupancy $\overleftarrow{q}$ which is used to calculate ingress lossless PFC thresholds (see Equation 1) accounts for both lossless and lossy traffic.

■ ***Takeaway.*** *Although lossless and lossy traffic are admitted seemingly independently by ingress and egress, the admission control for lossless traffic depends on both lossy and lossless occupancy, whereas the admission control for lossy traffic depends only on its own occupancy.*

It is natural to ask here whether these issues are due to the underlying buffer management scheme DT [15], and hence whether recent proposals such as ABM [1] should be able to avoid them. Although ABM can address isolation across various traffic priorities, it only works for a buffer-sharing architecture that supports lossy traffic but not lossless traffic. While, in theory, one could extend ABM to calculate the thresholds accordingly (by replacing Equations 1 within the same buffer architecture with the corresponding pools described in §2.2), both Issue 1 and Issue 2 would still hold. Indeed, if ABM operates in the default buffer architecture, it will (similar to DT) control lossless and lossy independently since lossless (lossy) bypasses egress (ingress) admission control. As a result, ABM cannot jointly impose fairness and isolate lossless and lossy traffic. Importantly, the technique used for burst absorption in ABM and other recent proposals [8, 25], i.e., prioritizing burst packets by using a high α parameter for thresholds, also results in Issue 3.

Root cause of Issue 3: Indeed, several recent works [1, 8, 25] rely on selectively taking action on burst and non-burst packets i.e., if the queue length exceeds its threshold, the buffer management scheme still accepts burst packets selectively by increasing the thresholds using a higher α parameter value only for those packets identified as burst. Figure 13 in Appendix A intuitively summarizes our key points. For example, if a non-burst (e.g., long flow) packet interleaves burst packets, the non-burst packet would be dropped since the queue length exceeds its threshold. However, a key property of PFC is that once PFC is triggered due to a non-burst packet, it affects all arriving traffic (including bursts) to the queue due to PAUSE frames. Selectively accepting packets does not apply for PFC.

■ ***Takeaway.*** *Optimally prioritizing bursts involves preferentially treating packets belonging to the burst only; this is possible for lossy but not for lossless traffic where congestion is signaled at a per-queue granularity.*

3 REVERIE

Based on the lessons learned from our analysis in §2, we design a buffer-sharing scheme REVERIE which prevents harmful interactions between lossless and lossy traffic (isolation) while absorbing bursts of both. We first describe the two pillars on which REVERIE relies: *(i)* consolidated admission control; and *(ii)* a low pass filter. Next, we explain how they fit together to form REVERIE. Finally, we discuss REVERIE's properties and its practicality.

3.1 Single Buffer Pool for Isolation

We argue that the first step towards achieving isolation is to have full visibility and control over the state of the buffer. Yet, current buffer-sharing practices maintain independent views and admissions at the ingress and egress, prohibiting global visibility and control. To address this, REVERIE uses a single shared buffer pool as shown in Figure 1, in

addition to a headroom pool dedicated to lossless queues. Further, REVERIE uses a single admission control that jointly optimizes the buffer allocation for lossless and lossy queues. The single shared buffer pool and a single admission control offer REVERIE a bird's eye view over the buffer.

Specifically, upon a packet arrival, REVERIE first determines the packet's class (lossless or lossy). If the packet belongs to lossless • class, REVERIE maps the packet to an ingress lossless queue $(s,p) \in \overleftarrow{Q}^{\bullet}$, where s is the source port that received the packet and p is the packet priority. Similarly, if the packet belongs to the lossy class, REVERIE maps the packet to an egress lossy queue $(d,p) \in \overrightarrow{Q}^{\circ}$, where d is the port to which the packet is destined and p is the packet priority.

As illustrated in Figure 6, REVERIE maintains only two types of queues (counters) i.e., ingress lossless and egress lossy, as opposed to the four types of queues in SONiC (see Figure 3). Further, REVERIE accounts for each packet only once as opposed to twice (once at ingress and once at egress) in the current buffer-sharing practices. In essence, all the lossless and lossy queues are mapped to the same (single) shared buffer pool as shown in Figure 1.

Using a single shared buffer pool and a single admission control leaves REVERIE solely responsible for fairly allocating the buffer across all queues to ensure isolation. In §3.3, we show how REVERIE's allocation achieves isolation across lossless and lossy.

3.2 Low-Pass Filter for Burst Absorption

A vast majority of prior works, including DT [15], FAB [8], ABM [1], TDT [25], calculate a threshold and compare it against *instantaneous* queue lengths in order to take buffer decisions. To improve burst absorption, prior works [1,8,25] selectively prioritize certain packets (e.g., short flows) by assigning them a larger threshold compared to the default threshold for other packets (e.g., long flows). However, as we explained in §2.3, selective packet prioritization cannot improve burst absorption for lossless traffic leading to Issue 3. It is essential to identify a queue experiencing a burst and prioritize all incoming traffic to the queue under bursty scenarios. A natural indicator for a queue that is experiencing a burst is its queue gradient, i.e., the rate of change of queue length. Thus, one could increase thresholds proportionally to the queue gradient. While intuitive, queue gradient is hard to monitor/calculate in practice in hardware, especially at microsecond granularity. To address this, we show an equivalence between an admission control based on queue gradient and an admission control based on first-order low-pass filtered queue lengths (Property 1). Our full proof can be found in Appendix B. Leveraging this equivalence, REVERIE uses an exponential weighted moving average which is an easy-to-implement first-order low-pass filter. In essence, REVERIE compares *average* queue lengths against a threshold, unlike prior works that use instantaneous queue lengths.

Property 1 (Relationship of low pass filter and gradient)**.** *Let Ψ be an admission control scheme that compares first order low pass filtered queue length $\hat{q}(t-\delta t)$ against a threshold $\Gamma(t)$ i.e., $\hat{q}(t-\delta t) \leq \Gamma(t)$, where $t-\delta t$ denotes the previous time instance. Let Φ be an admission control that compares instantaneous queue length $q(t)$ against Ψ's threshold $\Gamma(t)$ incremented proportionally based on the average queue gradient $\frac{d\hat{q}(t)}{dt}$ i.e., $q(t) \leq \Gamma(t) + K \cdot \frac{d\hat{q}(t)}{dt}$; where K is a constant and $\frac{d\hat{q}}{dt}$ is the gradient. Then, there exists a constant K such that Ψ and Φ are equivalent.*

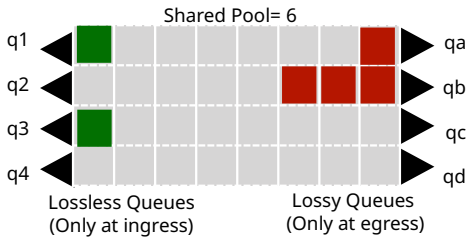


Figure 6: REVERIE's buffer bookkeeping: packets are stored once and accounted for once; lossless and lossy packets are accounted in the ingress and egress queues, respectively. All the queues are managed by a single admission control scheme.

3.3 The Workings of REVERIE

In this subsection, we put all the pieces together, to describe REVERIE's buffer-sharing architecture, admission control, and the underlying buffer management scheme.

Buffer sharing architecture: Let b be the total buffer space. REVERIE dedicates a headroom pool of size b_h for lossless traffic similar to the existing architecture. The rest of the shared buffer space denoted by $\overset{*}{b} = b - b_h$ is shared by both lossless and lossy traffic dynamically. A lossless packet is mapped to a lossless queue $(s,p) \in \overleftarrow{Q}^{\bullet}$ and a lossy packet is mapped to a lossy queue $(d,p) \in \overrightarrow{Q}^{\circ}$ based on the source port s, destination port d and the packet priority p. In total, REVERIE maintains a set $\overset{*}{Q}$ of queues, where $\overset{*}{Q} = \overleftarrow{Q}^{\bullet} \cup \overrightarrow{Q}^{\circ}$ consists of only two types of queues i.e., lossless and lossy instead of the four types maintained by SONiC.

Admission control: REVERIE calculates a threshold $\Gamma^i_p(t)$ for each queue (i,p) and compares it against the moving averaged queue length $\hat{q}^i_p(t)$ of the corresponding queue at time t i.e.,

$$\hat{q}^i_p(t) \leq \Gamma^i_p(t) \tag{4}$$

where $\hat{q}^i_p(t)$ is given by,

$$\hat{q}^i_p(t) = \overbrace{\gamma \cdot \hat{q}^i_p(t-\delta t)}^{\textit{Capture steady congestion}} + \overbrace{(1-\gamma) \cdot q^i_p(t)}^{\textit{Capture transient bursts}} \tag{5}$$

Here γ is a constant and a parameter for REVERIE, $q^i_p(t)$ is the instantaneous queue length, and $t-\delta t$ denotes the previous time instance. γ can be intuitively viewed as the degree of burst absorption. Since REVERIE's admission control (Equation 4) compares the threshold of a queue against its average queue

length (Equation 5), a higher γ masks the impact of transient bursts on the average queue lengths and, as a result, allows the admission control to absorb more transient bursts as illustrated in Figure 2. However, setting γ arbitrarily close to 1 makes admission control more oblivious to persistent or steady-state congestion. We leave it for future work to study the optimal parameter setting.

Any arriving packet that satisfies Equation 4 is buffered in the shared pool. If a lossy packet does not satisfy the above condition, the packet is simply dropped. However, if a lossless packet does not satisfy the above condition, the queue enters pause state (sends a PFC pause frame to its peer) and all the subsequent packets to the queue are buffered in the dedicated headroom irrespective of Equation 4. The queue sends out a PFC resume frame once the queue's headroom drains out completely and when Equation 4 is satisfied again.

Buffer management: The threshold $\Gamma^i_p(t)$ for each queue $(i,p) \in \overset{*}{Q}$ calculated by REVERIE at time t, depends on *(i)* a configurable parameter α_p for each priority, *(ii)* the number of congested queues n_p of priority p and the remaining shared buffer space $\overset{*}{b} - \overset{*}{q}(t)$ where $\overset{*}{b}$ is the size of shared buffer pool and $\overset{*}{q}(t)$ is the pool occupancy at time t. The threshold $\Gamma^i_p(t)$ is given by,

$$\Gamma^i_p(t) = \alpha_p \cdot \frac{1}{n_p} \cdot (\overset{*}{b} - \overset{*}{q}(t)) \quad \forall (i,p) \in \overset{*}{Q} \tag{6}$$

The thresholds used by REVERIE are similar to ABM [1] but we drop the dequeue rate factor due to the complexity of measuring it, especially when queues pause in the case of lossless traffic. However, our design does not prevent using the dequeue rate factor as well if it can be systematically measured.

3.4 The Properties of REVERIE

REVERIE inherits the steady-state isolation properties of ABM's thresholds for lossy as well as lossless traffic. Unlike ABM, though, which can only achieve isolation across priorities *within lossy* traffic, REVERIE can also achieve isolation *across lossless and lossy* priorities. In the following, for simplicity, we consider that all lossless and lossy queues are configured with the parameter value $\overset{\bullet}{\alpha}$ and $\overset{\circ}{\alpha}$ respectively. Our full proofs can be found in Appendix B. Next, we discuss our results intuitively.

In Theorem 1, we state the ratio in which buffer is allocated in the steady-state across lossless and lossy traffic in aggregate when both traffic classes compete for buffer. The ratio turns out to be the ratio of the configured α parameter. This makes it very intuitive and flexible to configure the buffer required for each traffic class, rather than the complicated pool sizes in the current practices.

Theorem 1 (Isolation)**.** *Under contention,* REVERIE *allocates buffer across lossless and lossy in the ratio of the corresponding* α *parameters i.e.,*

$$\frac{\overset{\bullet}{q}}{\overset{\circ}{q}} = \frac{\overset{\bullet}{\alpha}}{\overset{\circ}{\alpha}}$$

where $\overset{\bullet}{q}$ *and* $\overset{\circ}{q}$ *denote the steady-state shared buffer occupancy of lossless and lossy traffic respectively;* $\overset{\bullet}{\alpha}$ *and* $\overset{\circ}{\alpha}$ *denote the parameter values for lossless and lossy queues respectively.*

Based on Theorem 1, it is sufficient that the α parameter for lossless is greater than lossy in order to prevent issue 1. Further, since the thresholds are calculated with a bird's eye view of the buffer, the thresholds for both lossless and lossy depend on the overall buffer occupancy (see Equation 6). Hence, given that the α parameter for lossless is greater than lossy, REVERIE assigns a larger threshold for lossless compared to lossy i.e., prioritizing lossless over lossy. Essentially, REVERIE solves both issue 1 and issue 2 without statically partitioning the buffer.

When a single traffic class utilizes the buffer, REVERIE allocates $\frac{\alpha}{1+\alpha}$ fraction of the shared buffer, where α corresponds to the parameter value of the traffic class using the buffer. Notice that REVERIE allocates more buffer to a traffic class when it is not competing with the other class e.g., REVERIE allocates $\frac{\overset{\bullet}{\alpha} \cdot \overset{*}{b}}{1+\overset{\bullet}{\alpha}}$ amount of buffer when only lossless traffic is using the buffer compared to $\frac{\overset{\bullet}{\alpha} \cdot \overset{*}{b}}{1+\overset{\bullet}{\alpha}+\overset{\circ}{\alpha}}$ amount of buffer allocation for lossless when both traffic classes are competing for buffer space. Intuitively, REVERIE dynamically adapts the buffer allocation to lossless and lossy according to their load as opposed to the static pool sizes in the current practices. However, REVERIE keeps some buffer idle.

Theorem 2 (Buffer waste)**.** REVERIE *keeps idle a certain amount of buffer in the steady-state denoted by* b_w *given by,*

$$\frac{\overset{*}{b}}{1+\overset{\bullet}{\alpha}+\overset{\circ}{\alpha}} \leq b_w \leq \frac{\overset{*}{b}}{1+min(\overset{\bullet}{\alpha},\overset{\circ}{\alpha})}$$

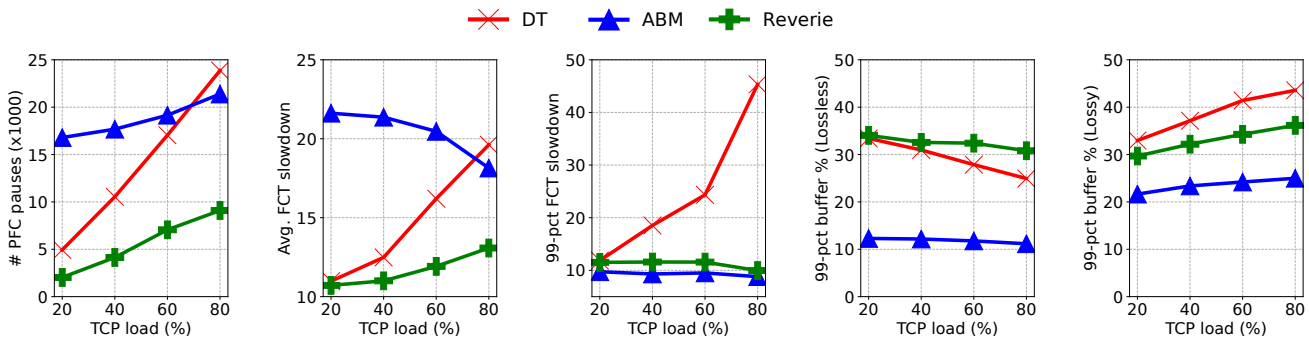
where $\overset{*}{b}$ *is the shared buffer pool size;* $\overset{\bullet}{\alpha}$ *and* $\overset{\circ}{\alpha}$ *are the parameter values for lossless and lossy queues correspondingly.*

Although REVERIE keeps a tiny buffer portion idle in the steady state, this helps in absorbing transient bursts. REVERIE effectively absorbs transient bursts even for lossless traffic since it compares *average* queue lengths against the threshold. Indeed, upon burst arrival, the average queue length hits the threshold slower than the instantaneous queue length, essentially absorbing transient bursts even for lossless queues. As a result, REVERIE finally solves issue 3.

3.5 Implementation Feasibility

A prototype implementation of REVERIE is beyond the scope of this paper and is part of our future work. Our discussions with NVIDIA already confirm that an approximation of the shared buffer pool model of REVERIE is feasible in hardware. In fact, we are currently discussing with a major Ethernet switch vendor on implementing REVERIE using their latest ASIC with programmable admission control features.

REVERIE is within reach of today's hardware because it does not significantly depart from commodity ASICs' buffer-sharing architecture and admission control mechanisms.



(a) PFC Pauses (b) RDMA incast flows (c) TCP short flows (d) RDMA buffer (e) TCP buffer

Figure 7: Buffer sharing under incast RDMA (DCQCN) workload and across various loads of websearch TCP (Cubic) workload. ABM achieves better performance for TCP but heavily penalizes RDMA, while REVERIE balances the two.

REVERIE introduces two primary changes. First, REVERIE uses a simpler buffer-sharing architecture with a single shared pool (excluding PFC headroom) and two types of queues. We have confirmed with NVIDIA that this architecture is supported by existing NVIDIA Ethernet switch ASICs[4]. Second, REVERIE uses a first-order low pass filter to obtain the moving averaged queue lengths. We believe this is practical as moving averages are used by common AQM like RED [20]. As previous works (e.g., ABM [1], [42]) have noted, various queue-length statistics are available and already used by the native buffer management of the MMU (although switch vendors do not open up the API for operators to do so on their own).

4 Evaluation

We evaluate the performance of REVERIE and compare it against the state-of-the-art approaches in the datacenter. Our evaluation aims to answer four main questions:

(Q1) *Can* REVERIE *protect RDMA from TCP?*

We find that REVERIE shields the performance of RDMA from TCP under various loads. At increased loads, REVERIE reduces the number of PFC pauses by 60% on average compared to DT and by 71.2% compared to ABM, with DCQCN as the transport protocol. When using advanced congestion control for RDMA, REVERIE reduces the number of PFC pauses by up to 100% compared to ABM.

(Q2) *Can* REVERIE *improve burst absorption of any class?*

We show that REVERIE significantly improves the burst absorption for RDMA and for TCP. With background TCP traffic (websearch), REVERIE improves the incast performance of RDMA by up to 33.3% compared to DT and by 50.4% compared to ABM. Under background RDMA traffic REVERIE improves the incast performance of TCP by up to 46.8% compared to DT and by up to 2.1% compared to ABM.

(Q3) *Does* REVERIE *penalize TCP?*

REVERIE does not penalize TCP. We find that REVERIE also improves the 99-percentile flow completion times (FCT) for short flows of TCP by 42.7% on average across various loads compared to DT. On this front, REVERIE is on par with ABM.

(Q4) *How sensitive is* REVERIE *to its parameters?*

We find an interesting characteristic of the parameter γ in REVERIE: increasing γ arbitrarily close to 1 dramatically reduces the number of PFC pauses, and improves the FCT for incast flows. However, beyond a certain value of γ, the infrequent PFC pauses negatively affect FCT. Finding the optimal γ value for a given switch remains an open question.

4.1 Setup

Our evaluation is based on network simulator NS3 [36].

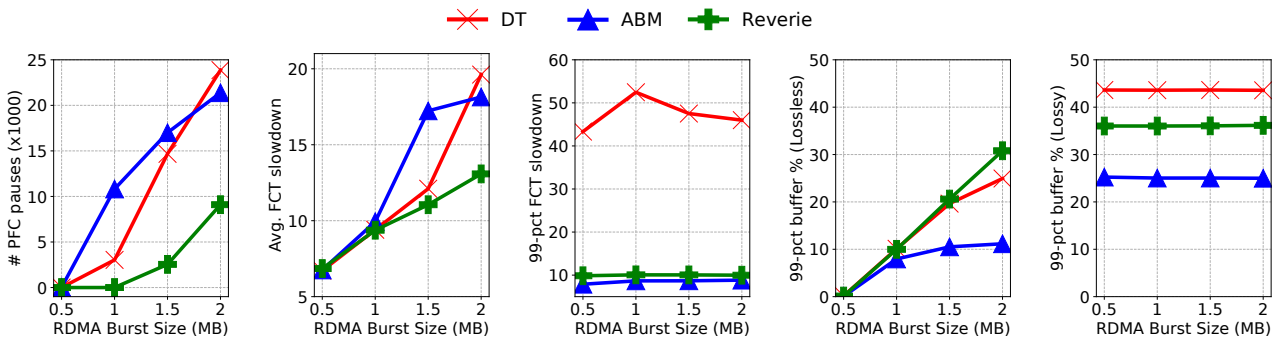
Topology: We consider a leaf-spine datacenter topology with 256 hosts organized into 4 spines and 16 leaves with 25Gbps links; link delay to 2μs (thus 17.28μs base RTT and 54KB bandwidth-delay product) and an oversubscription of 4, similarly to previous work [1, 2, 41]. All switches have 5.12KB buffer-per-port-per-Gbps similar to Broadcom Tomahawk [14][5]. All server NICs and switches are PFC enabled.

Traffic mix: We launch two types of workloads in our evaluation: *(i)* background and *(ii)* incast workloads. First, we generate background traffic across 20%-80% loads using websearch [5] flow size distribution, which is based on real-world datacenter measurements. Second, similar to prior works [1,2,4], we generate incast traffic using a synthetic workload that simulates the query-response behavior of a distributed file system. Specifically, each server in our topology sends out requests (queries) to all servers connected to a different leaf switch, chosen uniformly at random. These servers respond by sending a fraction of the file. We generate requests from each server based on a poisson process and we set the average request rate to 2 per second. We vary the file size (referred to as burst size). We use DCQCN [53] and PowerTCP [2] for RDMA congestion control; and Cubic [23] for TCP.

Baselines & metrics: We compare REVERIE with the SONiC [44] buffer model which is the state-of-the-art buffer sharing architecture widely deployed in today's datacenters.

[4]To implement this buffer-sharing architecture, we just need to map both egress lossy queues and egress lossless queues to a single egress buffer pool, and use an infinitely large egress lossless threshold.

[5]While Tomahawk splits the buffer across 4 MMUs, for simplicity, we assume a single MMU manages the entire buffer.



(a) PFC Pauses (b) RDMA incast flows (c) TCP short flows (d) RDMA buffer (e) TCP buffer

Figure 8: Buffer sharing under websearch TCP (Cubic) 80% load and across various burst sizes of incast RDMA (DCQCN) workload. As the burst size increases, the inability of DT and ABM to absorb RDMA bursts becomes more.

SONiC uses Dynamic Thresholds (DT) [15] as the buffer management scheme. The vast majority of the schemes in the literature are tailored for loss-tolerant traffic; thus it is unclear how those schemes can be evaluated in a fair manner for lossless traffic. To address this, we extend ABM [1] to support lossless traffic within the SONiC buffer model (after discussing with the authors) by accounting for the drain rate as well as the number of saturated queues in the ingress, and use it as a baseline. We report the following metrics: *(i)* number of PFC pauses triggered, *(ii)* average FCT slowdown for incast traffic, *(iii)* 99-percentile FCT slowdown for short flows of background traffic *(iv)* 99-percentile buffer occupancy of RDMA and *(v)* 99-percentile buffer occupancy of TCP, as a percentage of the total shared buffer.

Switch buffer configuration: We set the headroom pool size based on the NIC bandwidth and link delay, according to [49]. The remaining buffer is configured as ingress pool size. We set the egress lossless pool to the total switch buffer size and the egress lossy pool to 80% of the ingress pool size. For REVERIE, the headroom pool configuration is the same as stated earlier and the remaining buffer is configured as shared pool size. We set $\alpha = 1$ for all the schemes and set $\gamma = 0.999$ for REVERIE. We configure DCQCN according to [31], which is based on industry experience, and PowerTCP according to [2]. We set TCP minRTO to $1ms$.

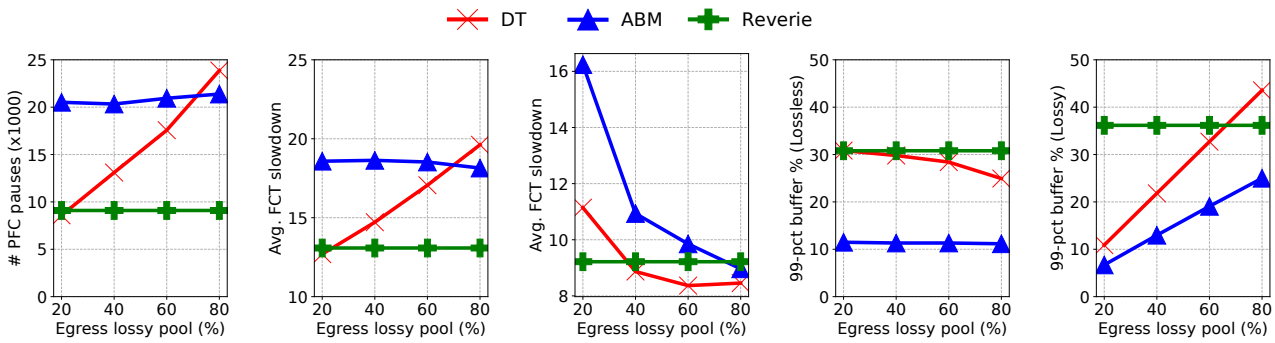
4.2 Results

REVERIE significantly reduces PFC pause rate: We generate TCP traffic using websearch workload and RDMA traffic using the incast workload in Figure 7 and Figure 8. RDMA uses DCQCN for congestion control. Across various loads of TCP and a burst size of 2MB for RDMA traffic, we observe from Figure 7a that REVERIE reduces the number of PFC pauses by 60% on average compared to DT and by 71.2% on average compared to ABM. Specifically, even at 20% TCP load, REVERIE reduces the number of PFC pauses by 58.9% compared to DT and by 87.9% compared to ABM. Further, across various burst sizes of RDMA with 80% TCP load, REVERIE reduces the number of PFC pauses by 61.8% on average compared to DT and by 57.4% on average compared to ABM, as shown in Figure 8a. In Figure 11 and Figure 12, we generate RDMA traffic using websearch workload and TCP traffic using the incast workload. We use PowerTCP as the congestion control for RDMA. From Figure 11a showing various RDMA loads at 1.5MB TCP burst size, and Figure 12a showing various TCP bursts at 80% RDMA load, we observe that REVERIE significantly reduces the PFC pauses by 100% compared to ABM, while REVERIE and DT perform similarly in this case. This confirms our observations in §2 on SONiC that lossy severely interacts with lossless traffic even though they are controlled independently by ingress and egress. REVERIE drastically reduces TCP's interference with RDMA.

REVERIE improves burst absorption for RDMA & TCP: Figure 7b, shows that across various TCP loads, REVERIE significantly reduces the average FCT for incast flows by 18.5% on average compared to DT and by 18.2% on average compared to ABM. At 80% TCP load, across various RDMA burst sizes, Figure 8b shows that REVERIE improves the average FCT for incast flows by 10% on average compared to DT and by 17% compared to ABM. This shows that REVERIE improves the overall burst absorption for RDMA. Although REVERIE's thresholds are similar to ABM's, REVERIE achieves better performance for lossless traffic due to its LPF-based admission control which favors transient bursts.

Across various loads of RDMA and 1.5MB TCP burst size, in Figure 11b, we see that REVERIE significantly reduces the average FCT for incast flows by 33.7% on average compared to DT and by 1.08% compared to ABM. From Figure 12b, across various TCP burst sizes, we observe that REVERIE reduces the average FCT for incast flows of TCP by up to 30% for large bursts compared to DT while REVERIE performs similarly to ABM. Overall, REVERIE's LPF-based admission control scheme improves burst absorption for RDMA as well as for TCP.

REVERIE protects RDMA from TCP in the buffer: Given the better burst absorption of REVERIE and significantly fewer PFC pauses even in the presence of TCP as seen in Figures 7-



(a) PFC pauses (b) RDMA incast flows (c) TCP long flows (d) RDMA buffer (e) TCP buffer

Figure 9: Buffer sharing under websearch TCP (Cubic) 80% load and incast RDMA (DCQCN) workload across various egress lossy pool sizes (% of ingress pool size) available for lossy TCP traffic. By changing the size of lossy pool, ABM and DT can only decide which traffic class will be prioritized against the other.

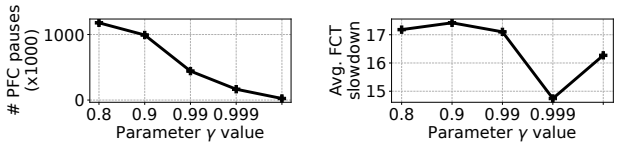


Figure 10: As the parameter γ value approaches 1, REVERIE becomes oblivious to the changes in the buffer and drastically reduces the number of PFC pauses. However, beyond a certain value, the lack of PFC pauses negatively impacts FCTs.

12, already shows that REVERIE successfully prevents TCP's interactions with RDMA. Moreover, from Figure 7d, we see that DT *reduces* its buffer allocation significantly for lossless traffic as the load of TCP increases, while *increasing* buffer allocation for TCP (Figure 7e). However, REVERIE gives lossless traffic its fair share in the buffer even at high TCP loads. Further, as the burst size of RDMA increases, from Figure 8d, we see that REVERIE opportunistically allocates increasingly buffer to RDMA whereas DT and ABM fail to allocate more buffers to RDMA, thus significantly increasing PFC pauses (Figure 8a). REVERIE's isolation properties allow RDMA to get its fair share of buffer even at high TCP loads. Reducing TCP's buffer share (egress lossy pool size) for DT and ABM improves the number of PFC pauses (Figure 9a) and flow completion times for RDMA incast flows, but TCP long flows suffer (Figure 9c) due to the reduced overall buffer available for TCP (Figure 9e). In contrast, REVERIE dynamically utilizes the entire shared buffer space in a fair manner and protects RDMA from TCP in the buffer as seen in Figures 7, 8, 9.

With advanced congestion control for RDMA (PowerTCP) under websearch workload, across various RDMA loads and TCP incasts (Figures 11d, 12d), REVERIE and DT as well as ABM occupy a significantly small portion of buffer and achieve similar FCTs for short flows of RDMA (Figures 11c, 12c). However, REVERIE triggers much lower PFC pauses than ABM as we observe in Figures 11a, 12a even with PowerTCP.

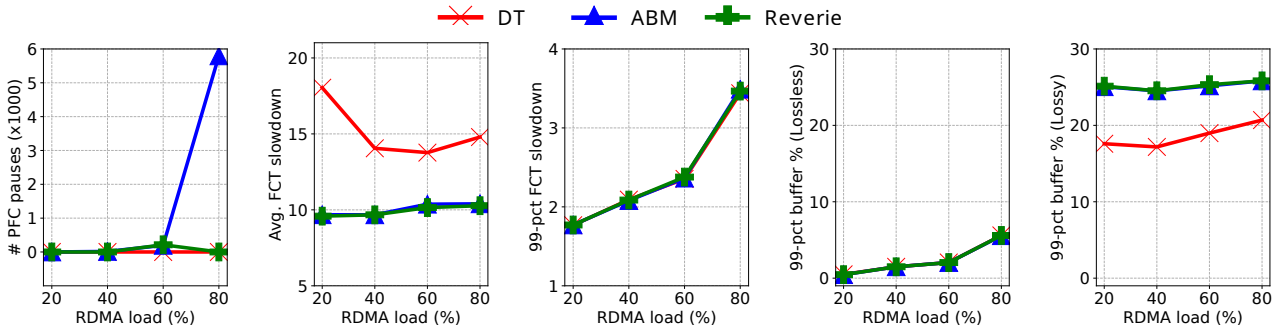
REVERIE also protects TCP in the buffer: Under websearch workload for TCP and RDMA incasts, from Figures 7c and 8c, we see that REVERIE and ABM achieve similar FCTs for short flows of TCP whereas DT severely penalizes TCP. DT penalizes TCP short flows even though it allocates more buffer to TCP compared to REVERIE as seen in Figures 7c, 8e. This excessive buffering results in increased queueing delays for DT. Further, under incast workload for TCP and websearch workload for RDMA, while REVERIE and ABM achieve similar FCTs for incast TCP flows, DT suffers from poor FCTs for TCP incasts (Figures 11b, 12b). Unlike DT, REVERIE and ABM protect TCP in the buffer.

Impact of LPF filtering: As discussed in §3.3, the parameter γ balances the capturing of steady-state (long-term) congestion against transient-state (short-term) congestion is captured by the admission control scheme. To better understand the impact of γ, we generate RDMA traffic using websearch workload at 80% load along with incast workload at 2MB burst size. In Figure 10, we show the number of PFC pauses and the average FCT for incast flows as a function of γ value. We observe that PFC pauses dramatically reduce as γ increases. Average FCT for incast flows decreases as γ increases until $\gamma = 0.999$. Yet, for $\gamma = 0.999999$ (close to 1), the average FCT increases by 9%. Naturally, a small γ value makes the admission control scheme highly sensitive to instantaneous queue lengths, which triggers PFC more frequently upon transient bursts. Similarly, a high γ value makes the admission control scheme insensitive to queue length and PFC is not triggered even when the queues *steadily* grow. In such cases, the excessive buffer occupied by steady-state traffic leaves less buffer to absorb transient bursts. Finding an optimal γ value is not required for reaping the benefits of REVERIE, as long as we avoid the extreme values that are easy to distinguish.

5 Related Work

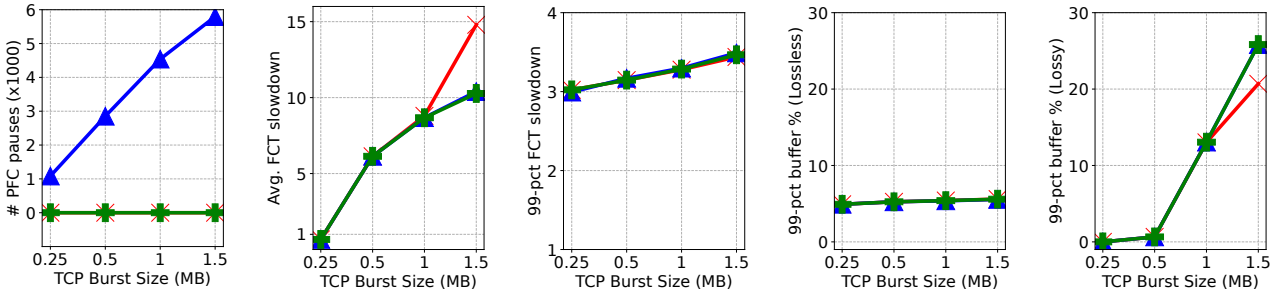
Our work relates to *(i)* buffer management; and *(ii)* RDMA.

Multiple works focus on sharing the on-chip buffer across queues of the same switch [1, 7–10, 13, 15, 18, 19, 29] and on sharing bandwidth across queues of the same port e.g.,



(a) PFC pauses (b) TCP incast flows (c) RDMA short flows (d) RDMA buffer (e) TCP buffer

Figure 11: Buffer sharing under incast TCP (with Cubic) and across various loads of websearch RDMA (with PowerTCP). ABM can only deal with low loads of RDMA traffic as it cannot distinguish or priortize it.



(a) PFC pauses (b) TCP incast flows (c) RDMA short flows (d) RDMA buffer (e) TCP buffer

Figure 12: Buffer sharing under websearch RDMA (with PowerTCP) at 80% load varying burst sizes of incast TCP (with Cubic). As TCP traffic increases, ABM further penalizes RDMA to protect TCP traffic.

AQM and scheduling [6, 20, 21, 35, 39]. In fact, there are also proposals to combine the two [16]. Further, augementing buffer sharing algorithms with machine-learned predictions has been shown to improve performance [3]. While useful, such works are designed exclusively for lossy traffic (i.e., TCP variants) and often with loss-based congestion control in mind. As a result, they are orthogonal to this work.

Many cloud providers have deployed RDMA over Ethernet to accelerate storage [11, 22], HPC, and ML [38]. To the best of our knowledge, all of these deployments [11, 22, 38] rely on PFC. Other research efforts related to RDMA include congestion control [2, 31, 53], efficient loss recovery [34], deadlock prevention [24], high performance RDMA applications [17, 26, 27, 30], testing [28], security [40, 46, 50] and performance isolation [52]. Among them, the most related topic is congestion control, but also in those works the buffer is only used by RDMA traffic [31, 53] (i.e., no TCP).

Coexistence of RDMA and TCP is an emerging new problem. Several recent parallel works proposed alternative solutions e.g., dynamically sharing the headroom buffer space under extremely shallow buffers [43]; using average occupancy time of packets to allocate buffers for each queue [32]. Yet, unlike REVERIE, these works do not address the fundamental issues that arise due to the static buffer pool configurations in today's switches. We leave it for future work to evaluate how the emerging alternative approaches fare against REVERIE's allocation.

6 Conclusion

This paper addresses the tension in buffer sharing between lossy traffic (e.g., TCP variants) and lossless traffic (e.g., RDMA) on datacenter switches. To this end, we first uncover, and explain analytically three particular unexpected buffer behaviors (issues) that today's buffer-sharing scheme can cause. Next, we find the root cause of these inefficiencies, and design a new buffer sharing scheme, REVERIE that can provide both isolation and high burst absorption to lossy and lossless traffic. In future work, we will try to closely work with a switch ASIC vendor to incorporate REVERIE into an ASIC's programmable admission control features.

Acknowledgements

We would like to thank our shepherd, Amy Ousterhout, as well as the anonymous reviewers for their useful feedback. We would like to thank Alex Shpiner, Eddy Kvetny, Matty Kadosh and Liron Mula from NVIDIA for many helpful discussions on the implementation feasibility of REVERIE. This work is part of a project that has received funding from the Austrian Science Fund (FWF), grant I 5025-N (DELTA), with Gabor Retvari, 2020-2024.

References

[1] 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*, SIGCOMM '22, page 36–52, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3544216.3544252.

[2] Vamsi Addanki, Oliver Michel, and Stefan Schmid. PowerTCP: Pushing the performance limits of datacenter networks. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 51–70, Renton, WA, April 2022. USENIX Association. URL: https://www.usenix.org/conference/nsdi22/presentation/addanki.

[3] Vamsi Addanki, Maciej Pacut, and Stefan Schmid. Credence: Augmenting datacenter switch buffer sharing with ml predictions. In *21th USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, Santa Clara, CA, April 2024. USENIX Association. URL: https://www.usenix.org/conference/nsdi24/presentation/addanki-credence.

[4] Mohammad Alizadeh and Tom Edsall. On the data path performance of leaf-spine datacenter fabrics. In *2013 IEEE 21st Annual Symposium on High-Performance Interconnects*, pages 71–74, 2013. doi:10.1109/HOTI.2013.23.

[5] 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*, SIGCOMM '10, page 63–74, New York, NY, USA, 2010. Association for Computing Machinery. doi:10.1145/1851182.1851192.

[6] Mohammad Alizadeh, Shuang Yang, Milad Sharif, Sachin Katti, Nick McKeown, Balaji Prabhakar, and Scott Shenker. Pfabric: Minimal near-optimal datacenter transport. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, page 435–446, New York, NY, USA, 2013. Association for Computing Machinery. doi:10.1145/2486001.2486031.

[7] Maria Apostolaki, Vamsi Addanki, Manya Ghobadi, and Laurent Vanbever. Fb: A flexible buffer management scheme for data center switches. *arXiv preprint arXiv:2105.10553*, 2021. URL: https://arxiv.org/abs/2105.10553.

[8] Maria Apostolaki, Laurent Vanbever, and Manya Ghobadi. Fab: Toward flow-aware buffer sharing on programmable switches. BS '19, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3375235.3375237.

[9] Mutlu Arpaci and John A. Copeland. Buffer management for shared-memory atm switches. *IEEE Communications Surveys & Tutorials*, 3(1):2–10, 2000. doi:10.1109/COMST.2000.5340716.

[10] James Aweya, Michel Ouellette, and Delfin Y Montuno. Buffer management scheme employing dynamic thresholds, September 7 2004. US Patent 6,788,697. URL: https://patents.google.com/patent/US6788697B1/en.

[11] Wei Bai, Shanim Sainul Abdeen, Ankit Agrawal, Krishan Kumar Attre, Paramvir Bahl, Ameya Bhagat, Gowri Bhaskara, Tanya Brokhman, Lei Cao, Ahmad Cheema, Rebecca Chow, Jeff Cohen, Mahmoud Elhaddad, Vivek Ette, Igal Figlin, Daniel Firestone, Mathew George, Ilya German, Lakhmeet Ghai, Eric Green, Albert Greenberg, Manish Gupta, Randy Haagens, Matthew Hendel, Ridwan Howlader, Neetha John, Julia Johnstone, Tom Jolly, Greg Kramer, David Kruse, Ankit Kumar, Erica Lan, Ivan Lee, Avi Levy, Marina Lipshteyn, Xin Liu, Chen Liu, Guohan Lu, Yuemin Lu, Xiakun Lu, Vadim Makhervaks, Ulad Malashanka, David A. Maltz, Ilias Marinos, Rohan Mehta, Sharda Murthi, Anup Namdhari, Aaron Ogus, Jitendra Padhye, Madhav Pandya, Douglas Phillips, Adrian Power, Suraj Puri, Shachar Raindel, Jordan Rhee, Anthony Russo, Maneesh Sah, Ali Sheriff, Chris Sparacino, Ashutosh Srivastava, Weixiang Sun, Nick Swanson, Fuhou Tian, Lukasz Tomczyk, Vamsi Vadlamuri, Alec Wolman, Ying Xie, Joyce Yom, Lihua Yuan, Yanzhao Zhang, and Brian Zill. Empowering azure storage with RDMA. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 49–67, Boston, MA, April 2023. USENIX Association. URL: https://www.usenix.org/conference/nsdi23/presentation/bai.

[12] Wei Bai, Shuihai Hu, Kai Chen, Kun Tan, and Yongqiang Xiong. One more config is enough: Saving (dc)tcp for high-speed extremely shallow-buffered datacenters. *IEEE/ACM Transactions on Networking*, 29(2):489–502, 2021. doi:10.1109/TNET.2020.3032999.

[13] Andreas V Bechtolsheim and David R Cheriton. Per-flow dynamic buffer management, February 4 2003. US Patent 6,515,963. URL: https://patents.google.com/patent/US6515963B1/en.

[14] Broadcom Tomahawk. https://people.ucsc.edu/~warner/Bufs/tomahawk.

[15] A.K. Choudhury and E.L. Hahne. Dynamic queue length thresholds for shared-memory packet switches. *IEEE/ACM Transactions on Networking*, 6(2):130–140, 1998. doi:10.1109/90.664262.

[16] Cisco. Intelligent buffer management on cisco nexus 9000 series switches. URL: https://www.cisco.com/c/en/us/products/collateral/switches/nexus-9000-series-switches/white-paper-c11-738488.pdf.

[17] Aleksandar Dragojević, Dushyanth Narayanan, Miguel Castro, and Orion Hodson. FaRM: Fast remote memory. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, pages 401–414, Seattle, WA, April 2014. USENIX Association. URL: https://www.usenix.org/conference/nsdi14/technical-sessions/dragojevi%C4%87.

[18] F. Ertemalp, D.R. Cheriton, and A. Bechtolsheim. Using dynamic buffer limiting to protect against belligerent flows in high-speed networks. In *Proceedings Ninth International Conference on Network Protocols. ICNP 2001*, pages 230–240, 2001. doi:10.1109/ICNP.2001.992903.

[19] Ruixue Fan, A. Ishii, B. 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 1B, pages 631–637 vol. 1b, 1999. doi:10.1109/GLOCOM.1999.830130.

[20] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, 1993. doi:10.1109/90.251892.

[21] Sally Floyd, Ramakrishna Gummadi, Scott Shenker, et al. Adaptive red: An algorithm for increasing the robustness of red's active queue management, 2001.

[22] Yixiao Gao, Qiang Li, Lingbo Tang, Yongqing Xi, Pengcheng Zhang, Wenwen Peng, Bo Li, Yaohui Wu, Shaozong Liu, Lei Yan, Fei Feng, Yan Zhuang, Fan Liu, Pan Liu, Xingkui Liu, Zhongjie Wu, Junping Wu, Zheng Cao, Chen Tian, Jinbo Wu, Jiaji Zhu, Haiyong Wang, Dennis Cai, and Jiesheng Wu. When cloud storage meets RDMA. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, pages 519–533. USENIX Association, April 2021. URL: https://www.usenix.org/conference/nsdi21/presentation/gao.

[23] Sangtae Ha, Injong Rhee, and Lisong Xu. Cubic: A new tcp-friendly high-speed tcp variant. *SIGOPS Oper. Syst. Rev.*, 42(5):64–74, jul 2008. doi:10.1145/1400097.1400105.

[24] Shuihai Hu, Yibo Zhu, Peng Cheng, Chuanxiong Guo, Kun Tan, Jitendra Padhye, and Kai Chen. Tagger: Practical pfc deadlock prevention in data center networks. CoNEXT '17, page 451–463, New York, NY, USA, 2017. Association for Computing Machinery. doi:10.1145/3143361.3143382.

[25] 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. doi:10.1109/TNET.2022.3173930.

[26] Anuj Kalia, Michael Kaminsky, and David Andersen. Datacenter RPCs can be general and fast. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, pages 1–16, Boston, MA, February 2019. USENIX Association. URL: https://www.usenix.org/conference/nsdi19/presentation/kalia.

[27] Anuj Kalia, Michael Kaminsky, and David G. Andersen. Design guidelines for high performance RDMA systems. In *2016 USENIX Annual Technical Conference (USENIX ATC 16)*, pages 437–450, Denver, CO, June 2016. USENIX Association. URL: https://www.usenix.org/conference/atc16/technical-sessions/presentation/kalia.

[28] Xinhao Kong, Yibo Zhu, Huaping Zhou, Zhuo Jiang, Jianxi Ye, Chuanxiong Guo, and Danyang Zhuo. Collie: Finding performance anomalies in RDMA subsystems. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 287–305, Renton, WA, April 2022. USENIX Association. URL: https://www.usenix.org/conference/nsdi22/presentation/kong.

[29] S. Krishnan, A.K. Choudhury, and F.M. Chiussi. Dynamic partitioning: a mechanism for shared memory management. In *IEEE INFOCOM '99. Conference on Computer Communications. Proceedings. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. The Future is Now (Cat. No.99CH36320)*, volume 1, pages 144–152 vol.1, 1999. doi:10.1109/INFCOM.1999.749262.

[30] Bojie Li, Tianyi Cui, Zibo Wang, Wei Bai, and Lintao Zhang. Socksdirect: Datacenter sockets can be fast and compatible. In *Proceedings of the ACM Special Interest Group on Data Communication*, SIGCOMM '19, page 90–103, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3341302.3342071.

[31] Yuliang Li, Rui Miao, Hongqiang Harry Liu, Yan Zhuang, Fei Feng, Lingbo Tang, Zheng Cao, Ming Zhang, Frank Kelly, Mohammad Alizadeh, and Minlan Yu. Hpcc: High precision congestion control. In *Proceedings of the ACM Special Interest Group on*

Data Communication, SIGCOMM '19, page 44–58, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3341302.3342085.

[32] Yi Liu, Jiangping Han, Kaiping Xue, Ruidong Li, and Jian Li. L2bm: Switch buffer management for hybrid traffic in data center networks. In *2023 IEEE 43rd International Conference on Distributed Computing Systems (ICDCS)*, pages 1–11, 2023. doi:10.1109/ICDCS57875.2023.00076.

[33] Matt Mathis and Andrew McGregor. Buffer sizing: a position paper. URL: https://buffer-workshop.stanford.edu/papers/paper16.pdf.

[34] Radhika Mittal, Alexander Shpiner, Aurojit Panda, Eitan Zahavi, Arvind Krishnamurthy, Sylvia Ratnasamy, and Scott Shenker. Revisiting network support for rdma. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '18, page 313–326, New York, NY, USA, 2018. Association for Computing Machinery. doi:10.1145/3230543.3230557.

[35] Kathleen Nichols and Van Jacobson. Controlling queue delay: A modern aqm is just one piece of the solution to bufferbloat. *Queue*, 10(5):20–34, may 2012. doi:10.1145/2208917.2209336.

[36] ns-3. Network simulator. URL: https://www.nsnam.org/.

[37] Eugene Opsasnick. Buffer management and flow control mechanism including packet-based dynamic thresholding. *US patent US7953002B2*. URL: https://patents.google.com/patent/US7953002B2/en.

[38] Oracle Cloud Infrastructure Blog: First principles: Building a high-performance network in the public cloud. URL: https://blogs.oracle.com/cloud-infrastructure/post/building-high-performance-network-in-the-cloud.

[39] Rong Pan, Preethi Natarajan, Chiara Piglione, Mythili Suryanarayana Prabhu, Vijay Subramanian, Fred Baker, and Bill VerSteeg. Pie: A lightweight control scheme to address the bufferbloat problem. In *2013 IEEE 14th International Conference on High Performance Switching and Routing (HPSR)*, pages 148–155, 2013. doi:10.1109/HPSR.2013.6602305.

[40] Benjamin Rothenberger, Konstantin Taranov, Adrian Perrig, and Torsten Hoefler. ReDMArk: Bypassing RDMA security mechanisms. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 4277–4292. USENIX Association, August 2021. URL: https://www.usenix.org/conference/usenixsecurity21/presentation/rothenberger.

[41] Ahmed Saeed, Varun Gupta, Prateesh Goyal, Milad Sharif, Rong Pan, Mostafa Ammar, Ellen Zegura, Keon Jang, Mohammad Alizadeh, Abdul Kabbani, and Amin Vahdat. Annulus: A dual congestion control loop for datacenter and wan traffic aggregates. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '20, page 735–749, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3387514.3405899.

[42] 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, 2015. doi:10.1109/INFOCOM.2015.7218374.

[43] Danfeng Shan, Yuqi Liu, Tong Zhang, Yifan Liu, Yazhe Tang, Hao Li, and Peng Zhang. Less is more: Dynamic and shared headroom allocation in pfc-enabled datacenter networks. In *2023 IEEE 43rd International Conference on Distributed Computing Systems (ICDCS)*, pages 591–602, 2023. doi:10.1109/ICDCS57875.2023.00019.

[44] SONiC. Software for Open Networking in the Cloud. URL: https://sonic-net.github.io/SONiC/.

[45] Tim Stevenson. Nexus 9000 architecture, 2020. URL: https://www.ciscolive.com/c/dam/r/ciscolive/emea/docs/2020/pdf/BRKDCN-3222.pdf.

[46] Konstantin Taranov, Benjamin Rothenberger, Adrian Perrig, and Torsten Hoefler. sRDMA – efficient NIC-based authentication and encryption for remote direct memory access. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, pages 691–704. USENIX Association, July 2020. URL: https://www.usenix.org/conference/atc20/presentation/taranov.

[47] NVIDIA Onyx User Manual v3.10.4302 (LTS). Shared Buffers. URL: https://docs.nvidia.com/networking/display/Onyxv3104302/Shared+Buffers.

[48] Haiyong Wang. OCPUS18 – SONiC Development and Deployment at Alibaba, 2018. URL: https://www.youtube.com/watch?v=aSd3R3gnQtw.

[49] CISCO white paper. Priority flow control: build reliable layer-2 infrastructure. 2009.

[50] Jiarong Xing, Kuo-Feng Hsu, Yiming Qiu, Ziyang Yang, Hongyi Liu, and Ang Chen. Bedrock: Programmable network support for secure RDMA systems. In *31st USENIX*

Security Symposium (USENIX Security 22), pages 2585–2600, Boston, MA, August 2022. USENIX Association. URL: https://www.usenix.org/conference/usenixsecurity22/presentation/xing.

[51] Zhenggen Xu. OCPSummit19 - EW: SONiC - LinkedIn Adoption of OCP SONiC, 2019. URL: https://www.youtube.com/watch?v=skUnjqPOvXs.

[52] Yiwen Zhang, Yue Tan, Brent Stephens, and Mosharaf Chowdhury. Justitia: Software Multi-Tenancy in hardware Kernel-Bypass networks. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 1307–1326, Renton, WA, April 2022. USENIX Association. URL: https://www.usenix.org/conference/nsdi22/presentation/zhang-yiwen.

[53] Yibo Zhu, Haggai Eran, Daniel Firestone, Chuanxiong Guo, Marina Lipshteyn, Yehonatan Liron, Jitendra Padhye, Shachar Raindel, Mohamad Haj Yahia, and Ming Zhang. Congestion control for large-scale rdma deployments. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, SIGCOMM '15, page 523–536, New York, NY, USA, 2015. Association for Computing Machinery. doi:10.1145/2785956.2787484.

A Analysis of SONiC

In this section, we formally model the buffer sharing scheme of SONiC [44] as well as REVERIE and analyze their buffer allocation properties. Our analysis aims to understand the steady-state behavior of the buffer. We build upon the model and the notation introduced in §2. Specifically, we analyze a shared buffer switch architecture whose buffer sharing scheme supports both lossless traffic and lossy traffic. Our analysis is based on a fluid flow model with deterministic packet arrival rates, extending [1, 15].

We begin by analyzing the drop thresholds for lossy traffic. As we study the steady-state of the buffer, we drop the time variables in our notation for ease of presentation. Since every packet is accounted both in the ingress and egress, the following relations hold at all times: *(i)* lossy traffic buffer occupancy $\overset{\circ}{\rightarrow}{q}$ at egress equals its occupancy $\overset{\circ}{\leftarrow}{q}$ at ingress; *(ii)* lossless traffic buffer occupancy at egress $\overset{\bullet}{\rightarrow}{q}$ equals its occupancy at the ingress pool $\overset{\bullet}{\leftarrow}{q}$ plus headroom occupancy q_h; *(iii)* ingress pool occupancy $\overset{}{\leftarrow}{q}$ equals the sum of occupancy of lossless $\overset{\bullet}{\leftarrow}{q}$ (without PFC headroom) and lossy $\overset{\circ}{\leftarrow}{q}$ traffic occupancy at ingress.

$$\overset{\circ}{\rightarrow}{q} = \overset{\circ}{\leftarrow}{q} \tag{7}$$

$$\overset{\bullet}{\rightarrow}{q} = \overset{\bullet}{\leftarrow}{q} + q_h \tag{8}$$

$$\overset{}{\leftarrow}{q} = \overset{\circ}{\leftarrow}{q} + \overset{\bullet}{\leftarrow}{q} \tag{9}$$

Based on the egress admission control for lossy traffic i.e., Dynamic Thresholds (see Equation 1 in §2), if $\overset{\circ}{\rightarrow}{n}$ egress lossy queues are in the steady state, then the total buffer occupancy is $\overset{\circ}{\rightarrow}{q} = \overset{\circ}{\rightarrow}{n} \cdot (\overset{}{\rightarrow}{b} - \overset{\circ}{\rightarrow}{q})$. By rearranging the terms, we obtain the egress lossy pool occupancy:

$$\overset{\circ}{\rightarrow}{q} = \frac{\overset{\circ}{\rightarrow}{n} \cdot \overset{}{\rightarrow}{\alpha} \cdot \overset{}{\rightarrow}{b}}{1 + \overset{\circ}{\rightarrow}{n} \cdot \overset{}{\rightarrow}{\alpha}} \tag{10}$$

Similarly, based on the ingress admission control for lossless traffic, if $\overset{\bullet}{\leftarrow}{n}$ lossless queues are in the steady state, then the total buffer occupancy is $\overset{\bullet}{\leftarrow}{q} = \overset{\bullet}{\leftarrow}{n} \cdot \overset{\bullet}{\leftarrow}{\alpha} \cdot (\overset{}{\leftarrow}{b} - \overset{\bullet}{\leftarrow}{q})$. Substituting in Equation 9 and using Equation 10 for $\overset{\circ}{\rightarrow}{q}$, we obtain the following:

$$\overset{}{\leftarrow}{q} = \overset{}{\leftarrow}{b} \cdot \left(\frac{\overset{\bullet}{\leftarrow}{n} \cdot \overset{\bullet}{\leftarrow}{\alpha}}{1 + \overset{\bullet}{\leftarrow}{n} \cdot \overset{\bullet}{\leftarrow}{\alpha}} \right) + \overset{\circ}{\rightarrow}{b} \cdot \left(\frac{1}{1 + \overset{\bullet}{\leftarrow}{n} \cdot \overset{\bullet}{\leftarrow}{\alpha}} \cdot \frac{\overset{\circ}{\rightarrow}{n} \cdot \overset{}{\rightarrow}{\alpha}}{1 + \overset{\circ}{\rightarrow}{n} \cdot \overset{}{\rightarrow}{\alpha}} \right)$$

Finally, substituting $\overset{}{\leftarrow}{q}$ from above in $\overset{\bullet}{\leftarrow}{q} = \overset{\bullet}{\leftarrow}{n} \cdot \overset{\bullet}{\leftarrow}{\alpha} \cdot (\overset{}{\leftarrow}{b} - \overset{}{\leftarrow}{q})$, we obtain $\overset{\bullet}{\leftarrow}{q}$, the buffer occupied by lossless traffic at the ingress pool.

$$\overset{\bullet}{\leftarrow}{q} = \overset{}{\leftarrow}{b} \cdot \left(\frac{\overset{\bullet}{\leftarrow}{n} \cdot \overset{\bullet}{\leftarrow}{\alpha}}{1 + \overset{\bullet}{\leftarrow}{n} \cdot \overset{\bullet}{\leftarrow}{\alpha}} \right) - \overset{\circ}{\rightarrow}{b} \cdot \left(\frac{\overset{\bullet}{\leftarrow}{n} \cdot \overset{\bullet}{\leftarrow}{\alpha}}{1 + \overset{\bullet}{\leftarrow}{n} \cdot \overset{\bullet}{\leftarrow}{\alpha}} \cdot \frac{\overset{\circ}{\rightarrow}{n} \cdot \overset{}{\rightarrow}{\alpha}}{1 + \overset{\circ}{\rightarrow}{n} \cdot \overset{}{\rightarrow}{\alpha}} \right) \tag{11}$$

Overall, our steady-state analysis gives the amount of buffer occupied by lossless traffic (Equation 11) and lossy traffic (Equation 10) based on the buffer configuration and the state of the buffer i.e., the number of active queues of each class. Following the admission control of Dynamic Thresholds from Equation 1, we could compute the drop thresholds for egress lossy and PFC thresholds for ingress lossless by using each pool occupancy from above.

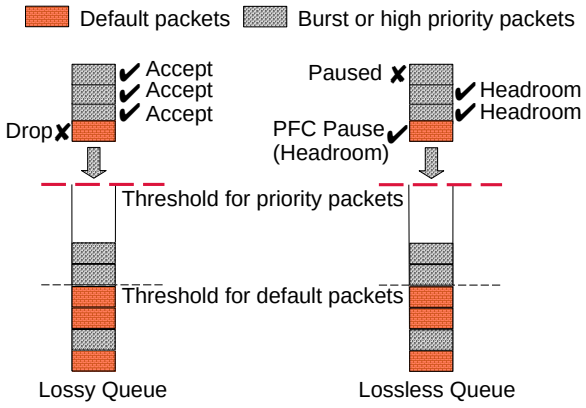


Figure 13: Per-packet prioritization cannot increase burst absorption for lossless traffic since PFC works at per-queue granularity.

B Analysis of REVERIE

In this section, we formally prove the properties of REVERIE stated in this paper. Before analyzing the steady-state behavior of REVERIE, we first show the relation between low-pass filters and gradient which builds the intuition for our low-pass filter-based approach.

Property 1 (Relationship of low pass filter and gradient)**.** *Let Ψ be an admission control scheme that compares first order low pass filtered queue length $\hat{q}(t-\delta t)$ against a threshold $\Gamma(t)$ i.e., $\hat{q}(t-\delta t) \leq \Gamma(t)$, where $t-\delta t$ denotes the previous time instance. Let Φ be an admission control that compares instantaneous queue length $q(t)$ against Ψ's threshold $\Gamma(t)$ incremented proportionally based on the average queue gradient $\frac{d\hat{q}(t)}{dt}$ i.e., $q(t) \leq \Gamma(t) + K \cdot \frac{d\hat{q}(t)}{dt}$; where K is a constant and $\frac{d\hat{q}}{dt}$ is the gradient. Then, there exists a constant K such that Ψ and Φ are equivalent.*

Proof. We consider exponentially weighted moving average for the first order low pass filter in this context. Let $q(t)$ denote the instantaneous queue length and let $\hat{q}(t)$ denote the average queue length. We denote the moving average parameter by ν. The moving average of the instantaneous queue lengths is then as follows:

$$\hat{q}(t) = \nu \cdot q(t) + (1-\nu) \cdot \hat{q}(t-\delta t)$$

where δt denotes the previous time when the average was updated. By rearranging the terms and dividing by δt,

$$\frac{\hat{q}(t) - \hat{q}(t-\delta t)}{\delta t} = \frac{\nu}{\delta t} \cdot (q(t) - \hat{q}(t-\delta t))$$

Let $K = \frac{\delta}{\nu}$. Using Euler's approximation method, we obtain the following:

$$K \cdot \frac{d\hat{q}(t)}{dt} = q(t) - \hat{q}(t-\delta t) \tag{12}$$

Using this relation, we now prove that Ψ and Φ admission control schemes are equivalent. We begin with Ψ which compares average queue lengths against a threshold $\Gamma(t)$ at time t.

$$\hat{q}(t-\delta t) \leq \Gamma(t)$$

Using Equation 12, we convert the above inequality as follows:

$$q(t) - K \cdot \frac{d\hat{q}(t)}{dt} \leq \Gamma(t)$$

By rearranging the terms, we obtain the the admission control scheme Φ. Hence Ψ and Φ are equivalent.

$$q(t) \leq \Gamma(t) + K \cdot \frac{d\hat{q}(t)}{dt}$$

□

Property 1 underlies the design of REVERIE's admission control based on LPF. We next prove the steady-state properties of REVERIE. Notice that by definition, the average queue length equals instantaneous queue length in steady-state. Due to this, REVERIE's steady-state properties are largely inherited from ABM. REVERIE differs from ABM in that, REVERIE's admission control is based on a single shared pool, where as ABM in the SONiC model would assign thresholds based on the pool size and occupancies for each priority (within a class). ABM cannot dynamically allocate buffer across RDMA and TCP since the pool sizes are fixed in the SONiC buffer model.

We next analyze the buffer allocation ratio for lossless and lossy.

Theorem 1 (Isolation)**.** *Under contention,* REVERIE *allocates buffer across lossless and lossy in the ratio of the corresponding α parameters i.e.,*

$$\frac{\overset{\bullet}{q}}{\overset{\circ}{q}} = \frac{\overset{\bullet}{\alpha}}{\overset{\circ}{\alpha}}$$

where $\overset{\bullet}{q}$ and $\overset{\circ}{q}$ denote the steady-state shared buffer occupancy of lossless and lossy traffic respectively; $\overset{\bullet}{\alpha}$ and $\overset{\circ}{\alpha}$ denote the parameter values for lossless and lossy queues respectively.

Proof. Since under steady-state, average and instantaneous converge, we simply use instantaneous values to prove our claim. Let $\overset{\bullet}{n}$ and $\overset{\circ}{n}$ be the number of congested queues of lossless and lossy. REVERIE allocates a total of $\overset{\bullet}{q} = \overset{\bullet}{n} \cdot \overset{\bullet}{\alpha} \cdot \frac{1}{\overset{\bullet}{n}} \cdot (\overset{*}{b} - \overset{*}{q}) = \overset{\bullet}{\alpha} \cdot (\overset{*}{b} - \overset{*}{q})$ to $\overset{\bullet}{n}$ lossless and a total of $\overset{\circ}{q} = \overset{\circ}{n} \cdot \overset{\circ}{\alpha} \cdot \frac{1}{\overset{\circ}{n}} \cdot (\overset{*}{b} - \overset{*}{q}) = \overset{\circ}{\alpha} \cdot (\overset{*}{b} - \overset{*}{q})$ to $\overset{\circ}{n}$ lossy queues. Since both lossy and lossless are mapped to the shared pool, we have that $\overset{\bullet}{q} + \overset{\circ}{q} = \overset{*}{q}$. But substituting the previous relations, we obtain:

$$\overset{*}{q} = \frac{(\overset{\bullet}{\alpha} + \overset{\circ}{\alpha}) \cdot \overset{*}{b}}{1 + (\overset{\bullet}{\alpha} + \overset{\circ}{\alpha})} \tag{13}$$

$$\overset{\bullet}{q} = \frac{\overset{\bullet}{\alpha} \cdot \overset{*}{b}}{1 + (\overset{\bullet}{\alpha} + \overset{\circ}{\alpha})} \tag{14}$$

$$\overset{\circ}{q} = \frac{\overset{\circ}{\alpha} \cdot \overset{*}{b}}{1 + (\overset{\bullet}{\alpha} + \overset{\circ}{\alpha})} \tag{15}$$

From the above relations, it is easy to see that the ratio $\frac{\overset{\bullet}{q}}{\overset{\circ}{q}} = \frac{\overset{\bullet}{\alpha}}{\overset{\circ}{\alpha}}$. □

Theorem 2 (Buffer waste)**.** REVERIE *keeps idle a certain amount of buffer in the steady-state denoted by b_w given by,*

$$\frac{\overset{*}{b}}{1 + \overset{\bullet}{\alpha} + \overset{\circ}{\alpha}} \leq b_w \leq \frac{\overset{*}{b}}{1 + min(\overset{\bullet}{\alpha}, \overset{\circ}{\alpha})}$$

where $\overset{*}{b}$ is the shared buffer pool size; $\overset{\bullet}{\alpha}$ and $\overset{\circ}{\alpha}$ are the parameter values for lossless and lossy queues correspondingly.

Proof. Our proof follows from the proof of Theorem 2. Specifically, for $\overset{\bullet}{n}$ lossless queues and $\overset{\circ}{n}$ lossy queues, REVERIE allocates $\overset{*}{q}$ in aggregate given by Equation 13. The remaining buffer $\overset{*}{b} - \overset{*}{q}$ which is wasted in the steady state is then given by,

$$b_w \geq \overset{*}{b} - \overset{*}{q} = \frac{\overset{*}{b}}{1 + (\overset{\bullet}{\alpha} + \overset{\circ}{\alpha})}$$

However, if a traffic class eg., lossless does not use the buffer, we can derive the remaining shared pool buffer similar to above. Depending on the smallest α value across all traffic

classes, when such a class uses the buffer alone, then in this case the buffer waste is given by,

$$b_w \leq \overset{*}{b} - \overset{*}{q} = \frac{\overset{*}{b}}{1 + (min(\overset{\bullet}{\alpha}, \overset{\circ}{\alpha}))}$$

□

We believe that REVERIE not only has interesting steady-state properties but its low pass filter based admission control find its best benefit under transient state analysis. We plan to analyze more such properties in the future.