LossPass: Absorbing Microbursts by Packet Eviction for Data Center Networks

Gyuyeong Kim, *Member, IEEE,* and Wonjun Lee, *Senior Member, IEEE*

Abstract—A bursty traffic pattern, called the microburst, is a key hurdle to achieve low latency for user-facing applications because it causes excessive packet losses in shallow buffered switches. Explicit Congestion Notification (ECN) can absorb microbursts by reserving buffer headroom, but the existence of headroom results in a fundamental trade-off between latency and throughput. To this end, we present LossPass, a buffer sharing mechanism that absorbs microbursts as many as possible while maintaining line-rate throughput. Specifically, LossPass evicts buffered large flow packets to make free buffer space on demand for arriving small flow packets. Our solution is inexpensive to implement on hardware. We implement a LossPass prototype and evaluate its performance through extensive testbed experiments and large-scale simulations. Our evaluation results show that LossPass reduces the FCT of small flows while maintaining line-rate throughput. For example, in testbed experiments, LossPass outperforms ECN by up to $3.20\times$ in the 99th percentile FCT of small flows.

Index Terms—Data center switches, Low latency, Packet loss.

1 Introduction

Modern data center applications, such as key-value stores and data analytics, require low latency for small messages and high throughput for bulk transfers. Those applications generate the *microburst*, which is a bursty traffic pattern consisting of many small flows [2], [3], [4]. Microbursts cause a sudden spike in queue length on the switch buffer. This often leads to excessive packet losses, hurting the flow completion time (FCT) of small flows, which is crucial to application message response times. Unfortunately, modern switches are vulnerable to microbursts because most of them have a shallow on-chip packet buffer.

Over past years, Explicit Congestion Notification (ECN) has garnered significant attention [5], [6]. ECN keeps queue length low around the ECN marking threshold K, and this naturally leaves buffer headroom where microburst traffic can be absorbed. However, despite the efficiency, ECN has a fundamental trade-off between latency and throughput due to buffer headroom. To provide line-rate throughput on a congested link, the switch requires a buffer size larger than the Bandwidth Delay Product (BDP) [7]. Therefore, when a high marking threshold ($K \geq$ BDP) is configured, the switch can fully utilize link capacity. However, it may result in unacceptable packet losses due to small headroom size. On the other hand, a low marking threshold ($K <$ BDP) enables the switch to absorb microbursts sufficiently but degrades throughput by causing overreaction of senders to ECN signals.

This motivates us to ask the following question: *can we absorb microbursts as many as possible while maintaining line-rate throughput?* To answer this question affirmatively, we present LossPass, a switch buffer sharing mechanism that absorbs microburst traffic as many as possible without buffer headroom. Specifically, when the buffer is full, LossPass evicts buffered large flow packets to avoid dropping of arriving small flow packets. This enables the switch to make free buffer space on request without reserving buffer headroom, providing line-rate throughput for bottleneck links. The key insight behind the idea is that the impact of packet loss on the FCT decreases rapidly as the flow size increases. For example, a timeout of $10ms$ [5] is fatal to a 10KB flow requiring only one RTT to finish while it is trivial to a 10MB flow lasting for many RTTs. Therefore, with LossPass, small flows can avoid lethal timeouts while causing only marginal impacts on large flows. Packet eviction is different from packet dropping in terms of target packets. Packet dropping targets unbuffered arriving packets whereas packet eviction handles already buffered packets.

To make the idea truly effective, we should address several technical challenges because switch hardware provides only restricted action primitives and prohibits non-deterministic operations. The first challenge is how to reduce the complexity of the victim search operation that finds a large flow packet in the port buffer. In general, finding a packet in a queue requires the $O(n)$ complexity. However, switch ASICs do not allow loop operations. We address this challenge by leveraging multiple class queues in the port. We classify small and large flows into different class queues so that the switch can directly find the victim packet by pointing the tail packet of the large flow queue. Second, when the size of the arriving packet is larger than that of the victim packet, multiple packets should be evicted.

- *This research was partly sponsored by the National Research Foundation of Korea (NRF) grant funded by the Ministry of Science and ICT (No. 2019R1A2C2088812), Next-Generation Information Computing Development Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science and ICT (No. 2017M3C4A7083676).*

- *G. Kim and W. Lee are with Network and Security Research Lab., School of Cybersecurity, Korea University, Seoul 02841, South Korea. E-mail: gykim08@korea.ac.kr; wlee@korea.ac.kr.*

Manuscript received 20 May 2020; revised DD Oct. 2020.
(Corresponding author: Wonjun Lee.)
A preliminary version of this paper appears in [1] as a letter.

This operation is also non-deterministic similar to the victim search using linear search. To address this, we evict the victim only if the victim packet is larger than or equal to the arriving packet. This does not limit the efficiency of LossPass because most of the buffered packets in the large flow queue are MTU-sized. We measure the number of missed chances on our testbed to demonstrate it. Third, large flows may be starved due to repetitive packet evictions. To prevent starvation, we assign a minimum guaranteed buffer size to the large flow queue, which can not be taken by small flows. We also measure the 99th percentile FCT of large flows to show that starvation does not occur. Lastly, LossPass cannot be effective when an ECN-based transport protocol is employed due to collision between the ECN behavior and packet eviction. To make LossPass compatible with ECN, we design a selective per-queue ECN scheme.

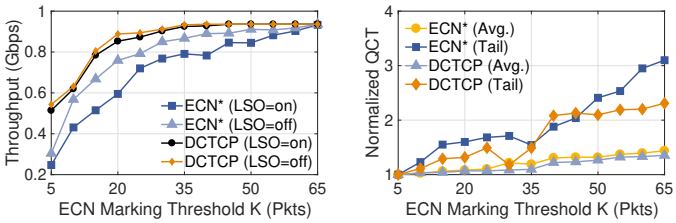
LossPass can be implemented on hardware inexpensively using existing action primitives with 5 clock cycles. To evaluate the efficiency of LossPass, we have implemented a LossPass prototype as a Linux `qdisc` module on a server-emulated switch. We first evaluate the performance of LossPass on a small-scale testbed. We conduct a microbenchmark using a targeted microburst scenario using `memcached`. The results show that LossPass improves the 99th percentile query completion time (QCT) by up to $22.24\times$ compared to ECN while maintaining line-rate throughput. We also perform dynamic flow experiments using realistic workloads from production data centers running web search [5] and data mining [8]. Our experiment results demonstrate that LossPass provides the lower latency than comparisons, ECN and PIAS [9]. For example, LossPass achieves the better average and 99th percentile FCTs of small flows than PIAS by up to $1.31\times$ and $3.20\times$, respectively.

To complement small-scale testbed experiments, we conduct large-scale packet-level simulations using ns-2 [10]. Our results show that LossPass can reduce the average and 99th percentile FCTs of small flows in large-scale data center networks. In addition, by analyzing packet loss rate and the impact on the FCT, we demonstrate that reducing the packet loss rate brings a significant improvement in the FCT of small flows while affecting the FCT of large flows slightly. We also demonstrate that LossPass is robust to different network conditions like ECN and transport protocols.

This paper makes the following contributions:

- We show that packet eviction enables the switch to absorb microbursts sufficiently without buffer headroom while sustaining line-rate throughput.
- We present LossPass, a buffer sharing mechanism that implements packet eviction by addressing various practical challenges.
- We show that LossPass can be implemented inexpensively and evaluate the performance of LossPass though extensive testbed experiments and large-scale simulations.

The remainder of the paper is organized as follows. In Section 2, we describe the background and motivation. Section 3 provides the design of LossPass. We present the implementation of a prototype and evaluation results in Section 4 and Section 5, respectively. We discuss related work in Section 6. Lastly, we conclude our work in Section 7.



(a) Aggregate Throughput (b) Query Latency

Fig. 1. [Experiment] Trade-off between latency and throughput in ECN.

2 Background and Motivation

2.1 Microbursts in Data Center Networks

Microbursts are a typical traffic pattern consisting of many small flow packets, and this is the primary cause of transient congestion events in data center networks [2], [3], [4]. The communication pattern of applications is the main reason of microbursts. In modern distributed applications, the aggregator node parallelizes tasks across worker nodes, and the workers return computation results to the aggregator. Because the processing time between servers is generally similar, the results often arrive at the switch output port almost simultaneously. This causes a sudden spike in queue length and often leads to buffer overflow. System batching schemes, such as Large Send Offload (LSO) and Large Receive Offload (LRO), are also responsible for microbursts [3], [5]. For example, TCP Segmentation Offload (TSO) is a typical hardware LSO technique. It forms a batch larger than MTU and transmits the batch-sized packet to the NIC. Since packets are transmitted back-to-back, the switch observes bursty packet arrivals. In this work, we focus the microburst caused by many small flows rather than the system batching schemes.

It is important to handle microbursts for the following reasons. First, packet loss caused by the microburst aggravates user experience seriously. The tail FCT of small flows, which is crucial to application response times, can be lengthened multiple times by only a single timeout. This is because the small flow lasts for a few RTTs whereas the timeout is hundreds of the RTT. Second, data center switches are prone to buffer overflow. The switch stores packets by dividing into multiple fixed-sized cells for memory management, and they are generally shallow-buffered. For example, in Broadcom Tomahawk ASIC-based switches, a 16MB of on-chip SRAM buffer is shared by 4 Memory Management Units (MMUs) where each MMU has 8×100Gbps ports. In this case, with static buffer allocation, per-port buffer size is only 512KB. In addition, even with dynamic buffer allocation, many switches limit the buffer occupancy of a single port to ensure per-port fairness.

2.2 Latency-Throughput Trade-off of ECN

ECN is a packet marking scheme that makes the sender decrease sending rates before that the queue length reaches the port buffer size B to avoid congestion proactively. ECN-enabled switches mark packets if the instantaneous queue length exceeds the ECN marking threshold K. This maintains queue length around K and naturally forms

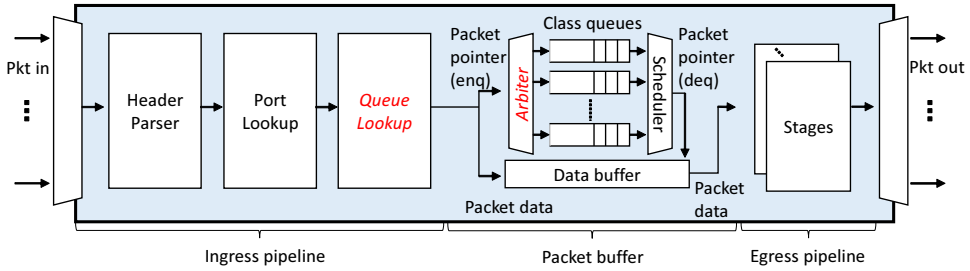


Fig. 2. Packet processing pipeline with LossPass. The arbiter performs packet enqueueing decisions including packet eviction based on the buffer occupancy of class queues, which are assigned by the queue lookup stage in the ingress pipeline.

$\approx B-K$ packets of buffer headroom where microbursts can be absorbed. Many transport protocols [5], [11] are designed to work in conjunction with ECN to decrease packet loss rate. Note that ECN in data centers uses the instantaneous queue length to react microbursts quickly, not the average queue length in the Internet environment.

Unfortunately, the existence of headroom causes a fundamental trade-off between latency and throughput. The switch requires at least $C \times RTT$ (i.e. the BDP) of buffer size to provide line-rate throughput on a bottleneck link where C and RTT denote link capacity and the base RTT [7][1]. If $K \geq C \times RTT$, the switch can maintain high link utilization. However, as the headroom is small, microbursts cannot be absorbed fully, leading to excessive packet losses. In contrast, if we configure $K < C \times RTT$, the switch can absorb bursts sufficiently, but cannot provide line-rate throughput due to overreaction of senders to the ECN congestion signal.

We conduct a series of experiments to show the trade-off of ECN. We build a `memcached` cluster with 7 nodes and a gigabit switch. A client node sends read queries over four server nodes with pre-populated key-value pairs. The other two nodes generate background traffic. By varying the ECN marking threshold, we measure the receiver-side aggregate throughput with and without LSO. We also measure QCTs of 1K 16-threaded queries with 32KB items. Fig. 1 shows the achieved throughput and the normalized average and 99th percentile QCTs. We use ECN* (ECN-enabled TCP) [11] and DCTCP [5] for the transport protocols. We omit the QCT results with LSO because the results with and without LSO are similar. We can clearly observe that the throughput and latency increase as K grows. Although DCTCP achieves better throughput than ECN*, it still loses 393Mbps when K=5. LSO makes throughput worse that, compared to the results without LSO, it degrades throughput by $1.13\times$ and $1.03\times$ on average for ECN* and DCTCP, respectively. We also find that the 99th percentile QCT increases rapidly as K grows because the headroom size goes small.

3 LossPass Design

3.1 Deisgn Goals and Basic Idea

Our goal is to design a solution that absorbs microbursts as many as possible while maintaining line-rate throughput. We stipulate that a good solution should satisfy the following requirements simultaneously:

- *Minimized tail latency:* Our solution should minimize the tail FCTs of small flows for low latency.
- *Line-rate throughput:* The link capacity of the switch should be fully utilized anytime for high throughput.
- *No headroom:* Our solution should not reserve buffer headroom to absorb microbursts.
- *Being practical:* Our solution should be inexpensive to implement.

To achieve the goal, LossPass passes packet loss of small flows to large flows to avoid timeout of small flows. Specifically, when the port buffer has no vacancy, the switch evicts a buffered large flow packet if the arriving packet is a small flow packet. The switch can buffer the arriving packet since free buffer space is obtained by packet eviction. This brings the following benefits:

- Since the switch can absorb microbursts as many as possible by evicting buffered packets, the tail FCT of small flows can be minimized.
- The switch does not have to limit queue length less than the BDP to absorb microbursts, hence the link capacity can be fully utilized.
- The switch can make free buffer space on demand irrelevant to the buffer occupancy of large flows. Therefore, the switch does not have to reserve buffer headroom for microburst traffic.
- Packet eviction requires memory read and write, which are already available memory operations in switch ASICs. Therefore, LossPass can be implemented inexpensively using commodity hardware.

The above benefits come at costs. Since the switch evicts buffered packets of large flows, this approach potentially degrades the FCT of large flows. However, this is a reasonable trade-off because we can reduce the FCT of small flows significantly while lengthening the FCT of large flows slightly. The reason why the expected benefits for small flows are larger than the expected loss for large flows is that the impact of packet loss decreases as the flow size grows. Not surprisingly, the timeout is fatal to small flows consisting of a few packets, but it is only a small portion of the entire FCT of large flows.

3.2 Detailed Mechanisms

Fig. 2 shows the packet processing pipeline of a switch ASIC with LossPass. At a high-level, LossPass consists of

1. When end-hosts use DCTCP as the transport protocol, the switch requires only $C \times RTT \times 0.17$ to achieve line-rate throughput though higher K is required in practice due to various factors [3], [5], [12].

Algorithm 1 Enqueueing decision in arbiter

```
      − P : current active packet
      − B : port buffer size determined by buffer management policy
      − Q : queue index metadata assigned by queue lookup table
      − Q_s : small flow queue
      − Q_l : large flow queue
      − B_l : reserved buffer size for Q_l
      − P_t^l : tail packet in Q_l
 1: if P.len + ΣQ_i.len > B then                          ▷ Buffer overflow
 2:     if P.Q == Q_s then                          ▷ § 3.2.1 Small flow packet?
 3:         if P.len ≤ P_t^l.len then               ▷ § 3.2.2 Enough victim size?
 4:             if P.len + Q_s.len ≥ B − B_l then    ▷ § 3.2.3 No Starve?
 5:                 Evict(P_t^l)                          ▷ Evict buffered packet
 6:                 Enqueue(P,Q)                          ▷ Enqueue current packet
 7:             end if
 8:         end if
 9:     end if
10:     Drop(P)                                          ▷ Drop current packet
11: else                                                 ▷ Buffer is not full
12:     Enqueue(P,Q)                                     ▷ Enqueue current packet
13: end if
14: if ECN is enabled then
15:     PerQ_ECN_mark(P)   ▷ § 3.2.4 Marks packet if ECN is enabled
16: end if
```

the queue lookup stage and the arbiter. The queue lookup stage determines the index of class queues in the output port where the packet will be buffered. The arbiter, the core component of LossPass, performs packet enqueueing decisions. Unlike traditional switch ASICs, the arbiter with LossPass can evict buffered packets to make free buffer space for arriving packets.

Alg. 1 describes the pseudocode of packet enqueueing decision of the arbiter. The arbiter in existing switch ASICs, if the buffer has no vacancy (Line 1), drops the current packet (Line 10). However, with LossPass, the arbiter performs additional operations. Specifically, the switch refers to queue index metadata to check whether the current packet belongs to a small flow. If the packet is a part of the small flow (Line 2), the arbiter refers to the tail packet size at the large flow queue to ensure that only one eviction will happen (Line 3). Next, the arbiter ensures that the small flow queue will not exceed the allowed buffer occupancy (Line 4). The arbiter then evicts the tail packet of the large flow queue (Line 5) and enqueues the current packet to the obtained free buffer space (Line 6). Lastly, regardless of buffer overflow, the arbiter marks the packet as congestion experienced if the enqueue ECN marking is enabled (Lines 14-16). Lines 2-4 and Lines 14-16 translate the idea of packet eviction to a practical solution by addressing various design challenges. In the following, we clarify how we handle the challenges in detail.

3.2.1 Reducing Searching Overhead

To perform packet eviction, we must find the victim packet that belongs to large flows first. Since a queue is an unsorted list consisting of a mixture of small and large flows, the complexity to find a large flow packet using linear search is $O(n)$ where n is the queue length. Unfortunately, the current switch ASICs only allow deterministic operations to guarantee a low processing delay. Therefore, we cannot find the victim packet using linear search.

LossPass finds the victim packet with $O(1)$ complexity by leveraging two class queues in the switch port. Modern switches typically provide 4-8 class queues per port, which we can assign different traffic classes to each class queue. We assign different queues Q_s and Q_l to small and large flows, respectively. Since Q_l contains only large flow packets, we can find the large flow packet directly by referring to the pointer of the tail packet at Q_l.

To classify packets into the class queues, we add the queue lookup stage in the ingress pipeline. The lookup stage reads the DSCP field at the IP header and updates the queue metadata Q with the matched queue index. The DSCP value can be tagged at the end-host using a Linux `NETFILTER` hook that tags the DSCP value based on socket metadata including process IDs, IP addresses, and port numbers. We can also use Heavy-Hitter Filter (HHF) `qdisc` [13] or PIAS [9] to classify flows based on a size threshold T. The first T bytes are tagged with the DSCP value of Q_s and the remaining bytes are tagged with the value of Q_l. T is determined by considering flow size distributions. It has no dependency on the packet processing logic of switches. Therefore, we can use T of PIAS for LossPass as well.

Although we leverage multiple queues, LossPass does not enforce a specific packet scheduler since the LossPass operation happens before packet buffering. Therefore, the network operator can use generic packet schedulers like strict priority queueing and round-robin. The recommended packet scheduler is strict priority queueing, which can accelerate small flows.

3.2.2 Dealing with Multiple Evictions

The switch may require multiple packet evictions because the arriving packet size may be larger than the victim packet size. Suppose that the arriving packet is 1500 bytes and the victim is a 64-byte packet. For this case, at least more than two packets must be evicted to enqueue the arriving packet. If every packet in Q_l is 64 bytes, $\lceil 1500/64 \rceil = 24$ rounds are required. This can increase processing delay excessively. More importantly, similar to the victim search operation, we cannot add non-deterministic operations in the packet processing pipeline.

To address the above issue, we perform packet eviction only if the arriving packet size is less than or equal to that of the victim packet. This means that multiple evictions never occur. The switch may miss chances that the arriving packet can be buffered by evicting multiple victims. However, the efficiency of LossPass does not decrease much because such cases are rare in practice. This is because Q_l contains mostly MTU-sized packets due to the large flow size. LSO mechanisms, which generate batches larger than the MTU size, do not have effects on the buffered packet size because the NIC splits the batch into MTU-sized packets before transmitting them to the wire. This also guarantees that the switch has enough free cells for the arriving packet when the target switch architecture divides packets into multiple fixed-sized cells for memory management.

3.2.3 Preventing Starvation

Since large flow packets are evicted, one might concern that large flows may be starved due to excessive eviction. Fortunately, since small flows comprising microbursts last only for a short-time scale and their size is small, starvation is uncommon. Furthermore, many application workloads have heavy-tailed flow size distributions where most of

bytes are from large flows [5], [8], [14], [15], [16]. In testbed experiments, we measure the 99th percentile FCT of large flows to show that starvation does not occur in production workloads. However, this does not guarantee that starvation never occur because workloads can change over time.

To mitigate potential risks of starvation, we assign a guaranteed buffer size B_l to Q_l. This ensures that B_l of buffer space cannot be taken by Q_s. Therefore, B_l prevents cases that large flows cannot occupy any buffer space, which can lead to starvation. With B_l, the queue length of Q_s is limited to $B - B_l$. Network operators should configure B_l by considering the flow size distribution of their workloads.

3.2.4 Providing Compatibility with ECN

LossPass should be compatible with ECN-based protocols because many data centers employ ECN-based transport protocols [5], [17]. However, with ECN, LossPass may not be effective. ECN tries to decrease congestion window size of arriving small flows, not buffered large flows. This is because the standard per-port ECN marking regards microbursts as the cause of congestion, which make the queue length exceed the marking threshold K.

To provide compatibility with ECN, LossPass uses selective ECN marking by leveraging per-queue ECN marking. Per-queue ECN marking allows the switch to mark packets using different ECN marking threshold K_i where i denotes the class queue index. We set K_s to the port buffer size while K_l is set to the recommended value of the employed ECN protocol. Therefore, the switch does not mark packets of Q_s and marks only packets in Q_l. Thus, when microburst traffic arrives, the switch can absorb microbursts and decrease the congestion window size of large flows at the same time. Note that synthesizing LossPass and ECN does not mean that our solution depends on buffer headroom to absorb microbursts. LossPass can make free buffer space regardless of ECN marking. ECN only controls the queueing delay, and LossPass takes charge of microburst absorption. In addition, although the senders of flows in Q_s do not receive ECN signals, they still can adjust congestion window size using triple duplicate ACKs and TCP retransmission timer.

3.3 Analysis of LossPass

We show how LossPass works through theoretical analysis.

Absorbing Microbursts without Headroom. Consider a switch output port whose link capacity is C. The port is shared by N infinitely long-lasting large flows with the same RTT in the compute rack. At time t, the queue length in the port can be given by

$$q(t) = \sum_{i=1}^{N} w_i(t) - C \times RTT \tag{1}$$

where $w_i(t)$ denotes the window size of ith sender at time t and RTT is the base RTT. Since the number of concurrent large flows on a link is small [5], we can assume that the flows are synchronized. Thus,

$$w_1(t) = w_2(t) = \cdots = w_N(t) = w(t) \tag{2}$$

$q(t)$ can be simplified as

$$q(t) = Nw(t) - C \times RTT \tag{3}$$

For ECN, $q(t)$ reaches the marking threshold K when $w(t) = (K + C \times RTT)/N$. Since it takes RTT to notify congestion to the sender, $q(t)$ can reach $K + 1$. Recall that we have N flows. Therefore, the maximum queue length in the port for ECN is

$$q_{max}^{\text{ECN}} = K + N \tag{4}$$

Let B denote the port buffer size where $B \geq C \times RTT$. Modern switches satisfy $B \geq C \times RTT$. For example, Broadcom Trident II ASIC can assign 192KB of buffer per port equally while base RTT is less than $100\mu s$ in 10Gbps networks. With q_{max}^{ECN}, we can obtain the maximum absorbable burst size of ECN that

$$\mu_{max}^{\text{ECN}} = B - q_{max}^{\text{ECN}} = B - (K + N) \tag{5}$$

Eq. (5) shows the buffer headroom of ECN. Since $K \gg N$, the headroom size depends on K.

LossPass has no buffer headroom, hence $q(t)$ can reach B. Thus, we have the maximum queue length in the port for LossPass as $q_{max}^{\text{LP}} = B$. However, LossPass can evict buffered large flow packets. Therefore, the maximum absorbable burst size of LossPass can be given by

$$\mu_{max}^{\text{LP}} = B - \left(q_{max}^{\text{LP}} - B\tau\right) = B\tau \tag{6}$$

where $\tau > 0$ is the fraction of large flows in the port.

We now compare ECN and LossPass. Eq. (6) can be rewritten as $B - B(1 - \tau)$. Since we have considered a case where only large flows exist in the port, $\tau = 1$. Since $K + N < B$, it is obvious that $B - B(1-\tau) > B - (K + N)$ when $\tau = 1$. Thus, we can conclude that $\mu_{max}^{\text{LP}} > \mu_{max}^{\text{ECN}}$. Let us consider a case where small and large flows are mixed in the port. Still, τ is generally high due to the heavy-tailed nature of flow size distribution in data centers. For example, 95%($\tau = 0.95$) of bytes are from large flows in a production web search workload [5], [18]. Thus, the inequality $\mu_{max}^{\text{LP}} > \mu_{max}^{\text{ECN}}$ still holds in practice. This suggests that LossPass can absorb microbursts as many as possible without headroom.

We now consider a case where only small flows exist in the buffer. LossPass is designed to make small flows occupy the port buffer as many as possible by evicting buffered large flow packets. Therefore, LossPass does not perform since the objective of LossPass is already fulfilled.

Line-Rate Throughput. We show that LossPass can achieve line-rate throughput. To achieve 100% throughput on the bottleneck link, the inequality $K \geq C \times RTT$ must be satisfied. Since $q_{max}^{\text{LP}} = B$, we can say that LossPass is equivalent to ECN with $K = B$. Recall that $B \geq C \times RTT$. Thus, LossPass satisfies the inequality.

3.4 Alternative Designs

Selective Packet Dropping. Cisco AFD [19] reserves a buffer to absorb microbursts by limiting the buffer share of large flows. AFD is in the same vein with ECN since it reserves buffer headroom to absorb microbursts. AFD results in worse performance than ECN because when the queue length reaches the threshold, AFD drops the packets whereas ECN marks the packet. Since AFD also uses buffer headroom, the trade-off between latency and throughput still exists. Aeolus [20] is a recent solution that uses selective

packet dropping. The switch drops low priority packets when the port queue length exceeds a given threshold. However, high priority packets are dropped only when buffer overflow occurs. Similar to AFD, Aeolus depends on buffer headroom that limits burst tolerance efficiency.

Deep Buffered Switches. Some hardware vendors provide deep buffered data center switches (e.g. Arista 7280R and Dell EMC Networking S4200-ON). However, most switches use a on-chip SRAM buffer because the deep buffer is too slow to meet latency requirements of modern user-facing applications. The deep buffer is provided by an off-chip DRAM. DRAM is less expensive than SRAM, but does not support cut-through switching and high switching capacity because of limited bandwidth. External memory also requires additional clock cycles for packet lookup operations. It can also cause excessive buffering of packets, leading to a large queueing delay.

Shared Buffer Allocation. Some buffer management schemes [2], [21] allow an output port to grab up excessive buffer space. For example, Arista 7050 series [21] allocate up to 56% of buffer space to a single port. However, it harms short-term fairness between ports because a single port can take away excessive buffer space that can be allocated to the other ports. The violation of short-term fairness can be harmful when many ports are busy and concurrent microbursts appear. LossPass is free from the fairness issue and does not require additional buffers since we make free buffer space by evicting buffered packets in the same port. LossPass is actually complementary to such buffer management schemes. For instance, the switch can absorb microbursts first using shared buffers and perform packet eviction for the remaining packets. We also use the dynamic threshold algorithm [22] for large-scale simulations in performance evaluation.

Dedicated Buffer Allocation. The switch can reserve a dedicated buffer size to absorb microburst traffic. For example, we can assign 50% of port buffer to Q_s. Since Q_s is for small flows, the switch can abosrb microbursts regardless of buffer occupancy of large flows. However, similar to ECN, the dedicated buffer approach also depends on buffer headroom. The switch cannot absorb traffic more than the dedicated buffer size when the rest buffer is full.

4 Implementation

In this section, we first discuss the implementation of LossPass on hardware. Next, we describe a software prototype of LossPass in detail.

4.1 Fixed-function Switch ASIC

The feasibility of hardware implementation as fixed-function switch ASICs can be demonstrated by analyzing the required action primitives and the number of clock cycles. As shown in Section 3, LossPass does not require prohibited operations like loops, and the queue lookup stage and the arbiter can be implemented using existing action primitives like comparators and memory read/write.

We now analyze the consumed clock cycles by LossPass. We consider hardware running at a typical clock frequency of 1 Ghz and the queue data structure is a doubly-linked list. The queue lookup stage can be implemented as a TCAM table. The switch first finds the DSCP field value of the IP header in the table entries (1 cycle). Next, the switch updates queue metadata with the matched queue index (1 cycle). For the arbiter, all the conditions (Lines 2-4 in Alg. 1) can be pipelined because they have no read/write dependency (1 cycle). Next, the switch frees memory of the victim packet by shifting the tail pointer of the large flow queue (1 cycle). Lastly, the switch enqueues the packet into the buffer by linking the packet to the small flow queue (1 cycle).

Overall, LossPass consumes 2 clock cycles for the queue lookup stage and 3 clock cycles for the arbiter. This overhead is small because switch ASICs generally require hundreds of clock cycles to process a packet. For example, Broadcom Trident 3 ASIC offers a minimum per-packet processing delay of 800 clock cycles. For this case, the overhead of LossPass is only 0.63% (5/800).

A challenge emerges when the reference queue structure is a singly-linked list. In the singly-linked list queue, evicting the tail packet requires n clock cycles because we must find the penultimate packet to update the tail pointer. In this case, evicting the head packet can be an alternative option because it requires no pointer update. The head eviction increases the FCT as much as the packet sojourn time, but this is tolerable because retransmission timer is generally tens of the packet sojourn time.

4.2 Programmable Switch ASIC

Recently, programmable switch ASICs like Barefoot Tofino [23] are emerged. These ASICs allow us to customize packet processing pipelines by adopting the Reconfigurable Match-action Table (RMT) architecture [24].

The queue lookup module can be implemented as a match-action table in the ingress pipeline easily. The DSCP field is the match key and the class queue index is the action parameter. It is challenging to implement the arbiter because the packet buffer is still a fixed-function block even in programmable switch ASICs. Therefore, we must implement the arbiter indirectly. Note that this is not an issue in fixed-function ASIC implementation. Since LossPass addresses packet enqueueing decisions, the arbiter should be located at the end of the ingress pipeline. One idea is to leverage a register array that stores only packet metadata. Every packet records its unique ID as metadata in register slots. Similar to head packet eviction, the switch may drop victim packets at the egress pipeline if the recorded metadata becomes invalid if other packets overwrite the register slot. However, this may cause an idle time on the link, degrading throughput. Another idea is to bounce microburst traffic to the switch control plane, which has off-chip DRAM. Note that modern baremetal switches have a CPU module including a x86 CPU and DRAM. The control plane returns packets to the data plane upon receiving. Although packet latency increases, the switch can prevent packet loss of microbursts. The correct implementation of LossPass using programmable switch ASICs is our future work.

4.3 Software Prototype

We have implemented a LossPass prototype as a Linux `qdisc` module on a server-emulated switch as illustrated

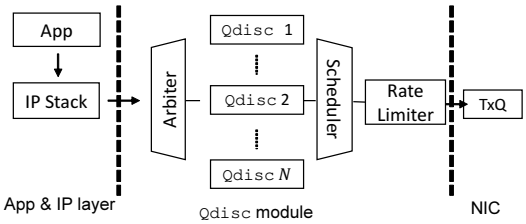


Fig. 3. LossPass software prototype as a Linux `qdisc` module.

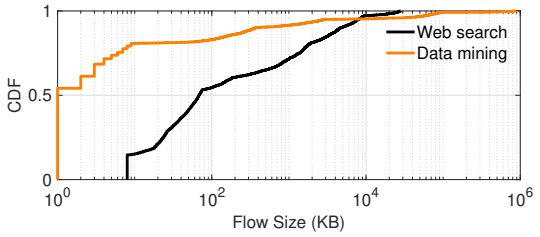


Fig. 4. Used workloads in dynamic flow experiments and simulations.

in Fig. 3. The module implements multiple priority queues by creating multiple FIFO `qdiscs` for each NIC. The key stages are as follows:

Enqueueing. When the packet from TCP/IP stack arrives to the `qdisc` layer, the module first returns the index of the corresponding priority queue by referring to the DSCP field. Next, the switch checks whether the port buffer has enough space to enqueue the packet. Basically, the switch enqueues/drops the packet based on the remaining buffer size. With LossPass, the switch performs packet eviction if conditions are met. To evict the buffered tail packet, we call `qdisc_dequeue_tail()` in the `qdisc`. When the switch evicts a buffered packet, we need to know the size of tail packet in the large flow queue. The size can be easily obtained using `skb_peek_tail()` supported in the `qdisc`. The switch also performs ECN marking.

Dequeueing. Packet dequeueing is done by the packet scheduler. The dequeued packet goes to NIC driver and NIC hardware before it is transmitted to the wire. Our `qdisc` module uses a Token-Bucket rate limiter to shape outgoing traffic at 99.5% of the NIC capacity to monitor queue length accurately and to avoid excessive buffering on TX ring.

5 Performance Evaluation

In this section, we evaluate the performance of LossPass to answer the following questions:

- **How does LossPass perform in practice?** We first conduct a series of testbed experiments using `memcached` and realistic workloads derived from production data centers to demonstrate the efficiency of LossPass.
- **Does LossPass scale to large-scale data center networks?** We build a large leaf-spine topology and evaluate LossPass against comparisons to show that LossPass performs well in the large network.
- **How LossPass brings performance gains in detail?** By analyzing packet loss rate, we clarify how LossPass decreases the FCT of small flows significantly.
- **How is LossPass robust to different network environments?** We show that LossPass is robust to network conditions like ECN and transport protocols.

Compared Schemes. We basically compare LossPass with ECN. We also compare our scheme against PIAS [9] in dynamic flow experiments because we use the DSCP tagging module of PIAS to classify small and large flows in dynamic flow experiments. PIAS is the state-of-the-art non-clairvoyant flow scheduling solution that implements multilevel feedback queueing.

Workloads. For dynamic flow experiments, we use realistic workloads derived from production data centers running a web search [5] and a data mining [8], which are shown in Fig. 4. The workloads have flows with heavy-tailed size distributions whose mean flow sizes are 1.58MB for the web search and 7.12MB for the data mining.

5.1 Testbed Experiments

Testbed Setup. We have built a small-scale testbed to emulate a compute rack. The testbed consists of 7 servers connected to a server-emulated switch. Each server is with Intel Gigabit NIC. The switch is equipped with multiple Intel I350-T4 v2 Gigabit NICs. For the switch, we have disabled LSO to emulate switch hardware behaviors more correctly. The base RTT is roughly $500us$.

Switch and Transport. Each port in the switch has the 128KB of buffer completely shared by queues. For the transport protocol, we use DCTCP [5] with K=45 packets (≈66KB) by default. We enable enqueue ECN marking to make LossPass compatible with ECN-based protocols. We set RTO_{min} to 10ms as suggested in existing works [5]. For static flow experiments, we use a `NETFILTER` module to tag DSCP values using port numbers. In dynamic flow experiments, we employ two-level PIAS [9] to classify small and large flows. We use 1MB for the demoting threshold T as suggested in the paper [9]. We set a guaranteed buffer of Q_l to 0 for maximized burst tolerance. We use strict priority queueing as the packet scheduler that the small flow queue has the highest priority.

Performance Metrics. Our primary performance metric is the FCT. We breakdown the FCT across different flow sizes to analyze the impact on small ($\leq$ 100KB), medium ($>$ 100KB & $\leq$ 10MB), and large flows ($>$ 10MB). For clear comparison, the results are normalized by the values of LossPass. Due to space limit, we omit the results of medium flows in testbed experiments.

5.1.1 Static Flow Experiments

We begin with a static flow experiment.

Methodology. For static flow experiments, in addition to DCTCP, we consider ECN* [11] (ECN-enabled TCP) because it shows the net effect of ECN. We first inspect throughput on the bottleneck link. Two senders transmit flows for 10 seconds using `iperf` and the receiver calculates average aggregate goodput. Next, we measure the 99th percentile QCT of `memcached`. `Memcached` is a widely used key-value store for user-facing services like web search engines and video streaming services. We pre-populate 4 server

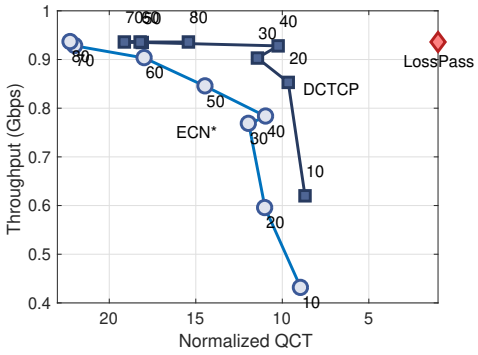


Fig. 5. [Experiment] Aggregate throughput and the QCT of `memcached`.

TABLE 1
[Experiment] Packet Statistics of Large Flows

	Total	Dropped	Evicted	Missed
w/ ECN	4.4M	2400	4522	26
w/o ECN	4.4M	6657	6939	15

TABLE 2
[Experiment] Improvement Degree by Packet Eviction

	Packet loss rate		99th percentile FCT	
	Small	Large	Small	Large
No-ECN	41.55×	0.54×	12.38×	0.99×
Per-Port ECN	10.61×	0.24×	1.13×	1.01×
Per-Queue ECN	11.27×	0.42×	1.32×	1.01×

instances with 1K key-value pairs with a 32KB item using `PUT` operation. Two senders generate background flows to the receiver using `iperf`. The client sends a 16-threaded `GET` query over the servers and the servers response with the requested item. Each query can be completed only if all response flows are finished. The client issues the subsequent query when the current query is completed. We generate 1K queries and obtain the QCT of the queries. We normalize the QCT by the values of LossPass.

Results. Fig. 5 shows the aggregate throughput and the 99th percentile QCT for ECN*, DCTCP, and LossPass. Note that the x-axis is reversed so that lower QCTs are to the right. Each label on data points of ECN* and DCTCP denotes the ECN marking threshold K. It is easy to see that LossPass is the best in both throughput and the tail QCT, which suggests that our solution successfully absorbs microbursts sufficiently without harming line-rate throughput. LossPass provides nearly 100% of throughput and outperforms the comparisons by up to $22.24\times$ in the tail QCT. LossPass achieves 100% throughput because it does not have to limit queue length to maximize burst tolerance. Unlike LossPass, each of ECN-based transport protocols forms an achievable trade-off region with Ks. Although DCTCP requires lower K than ECN* to achieve line-rate throughput, the throughput achieved by DCTCP with K=10 is only 620Mbps.

5.1.2 Dynamic Flow Experiments

We inspect the FCT using realistic data center workloads.

Methodology. We use a client-server application to generate traffic [25]. One server acts as the receiver and the other servers are senders. The receiver opens 5 persistent TCP connections to each of the senders. The receiver sends requests based on a Poisson process to senders with data size generated by the workloads through available connections. If there is no available connection, the receiver establishes a new connection. The senders respond with the requested data. We generate 5K flows by varying traffic loads from 50% to 80%.

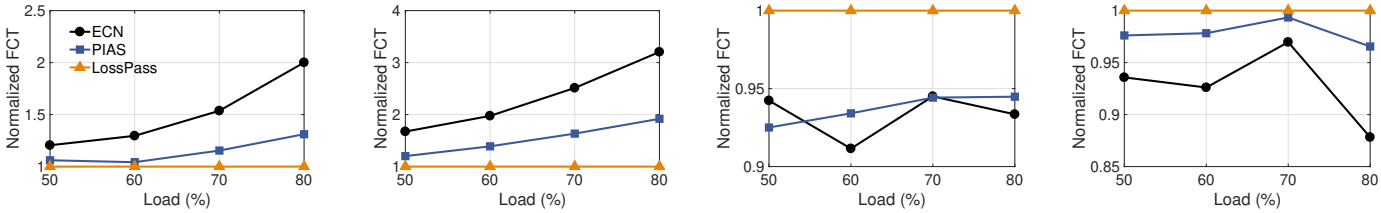
Small Flows. We examine the average and 99th percentile FCTs of small flows. Fig. 6 (a) and Fig. 7 (a) report the average FCT. As expected, LossPass works the best for both workloads. Our solution outperforms ECN by up to $2.00\times$ and $1.63\times$ for the web search workload and the data mining workload, respectively. Compared to PIAS, LossPass delivers the better performance by up to $1.31\times$ and $1.09\times$ for each of workloads, respectively. For the 99th percentile FCT, LossPass significantly improves the performance as shown in Fig. 6 (b) and Fig. 7 (b). Compared to PIAS, our scheme offers the lower 99th percentile FCT whose gap ranges are within $1.20\times \sim 1.92\times$ in the web search workload. The gaps are decreased in the data mining workload. However, LossPass is still better than PIAS by up to $1.66\times$. The difference between the two workloads is due to that the data mining workload has a more heavy-tailed flow size distribution.

Large Flows. We now inspect the results of large flows. The main concern of LossPass is how much the FCT of large flows is affected by being evicted instead of small flows. Fig. 6 (c) and Fig. 7 (c) show the average FCT of large flows for both workloads. LossPass performs worse than ECN and PIAS slightly. Our scheme underperforms ECN and PIAS by $0.93\times$ and $0.94\times$ on average in the web search workload. This is the expected because LossPass trades the FCT of large flows for that of small flows. By considering the performance gain on the FCT of small flows as shown in Fig. 6 (a), (b), we gracefully argue that the FCT degradation on large flows is tolerable. The results in the data mining workload show that LossPass has similar performance to ECN and PIAS due to its flow size distribution. The performance gap ranges are within $1.00\times \sim 1.02\times$ and $0.99\times \sim 1.05\times$ for ECN and PIAS, respectively. Fig. 6 (d) and Fig. 7 (d) show the 99th percentile FCT of large flows. If packet eviction of LossPass starved large flows, the tail FCT should be degraded significantly. However, the results are similar to those of the average FCT. This suggests that LossPass does not cause starvation of large flows. The primary reason is that the size of small flows is not enough to starve large flows.

5.1.3 Deep Dive

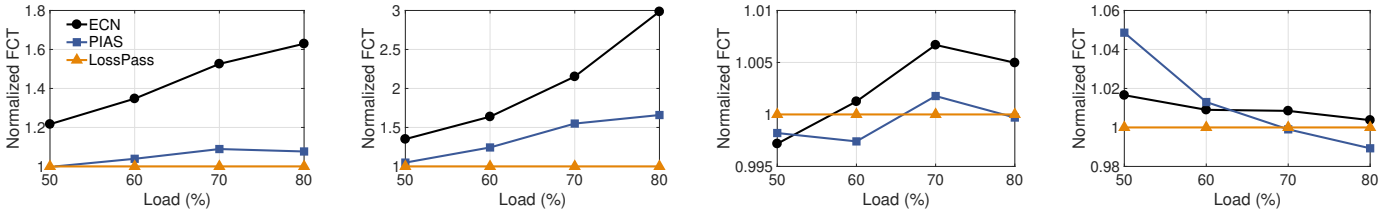
We now analyze the results of the web search workload with 70% of load for deep understanding.

How Many Evictions Occur? We examine how many evictions occur by LossPass with and without ECN. We also measure the number of missed eviction changes which the arriving packet for the small flow queue is larger than the buffered tail packet in the large flow queue. Table 1 shows the packet statistics of large flows. Overall, 4.4M of packets are generated as parts of large flows. We can find that more dropping and eviction occur when ECN is disabled



(a) (0, 100KB]: Avg (b) (0, 100KB]: 99th Percentile (c) (10MB, ∞]: Avg (d) (10MB, ∞]: 99th Percentile

Fig. 6. [Experiment] FCT statistics across different flow sizes in the web search workload.



(a) (0, 100KB]: Avg (b) (0, 100KB]: 99th Percentile (c) (10MB, ∞]: Avg (d) (10MB, ∞]: 99th Percentile

Fig. 7. [Experiment] FCT statistics across different flow sizes in the data mining workload.

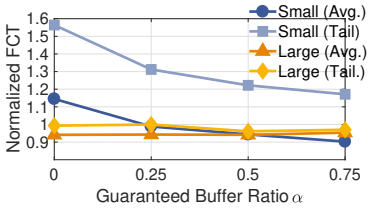


Fig. 8. [Experiment] Impact of guaranteed buffer size.

because queue length is not controlled. We also observe that LossPass does not cause excessive evictions. The number of missed chances is rare that we observe only 26 and 15 cases. This suggests that it is reasonable to limit the number of eviction to prevent the increase of processing delay.

Net Effect of Packet Eviction. We now investigate the net effect of packet eviction. Table 2 shows how packet eviction changes the packet loss rate and the tail FCT. The results are normalized to the values of LossPass. With No-ECN, the packet loss rate of small flows is reduced by $41.55\times$, which lead to $12.38\times$ of tail FCT reduction. The huge gap is due to poor performance of No-ECN. The packet loss rate of large flows changes by $0.54\times$, but the tail FCT of large flows does not change much. This is because large flows are robust to packet loss. The experiments with per-port and per-queue ECN schemes show similar results. Since ECN limits queue length to avoid packet loss, the improvement degree in the packet loss rate and the corresponding tail FCT for small flows are less than those of No-ECN. However, LossPass still improves the tail FCT of small flows and does not degrade the tail FCT of large flows. The results of 99th percentile FCT of large flows also suggest that LossPass does not cause starvation.

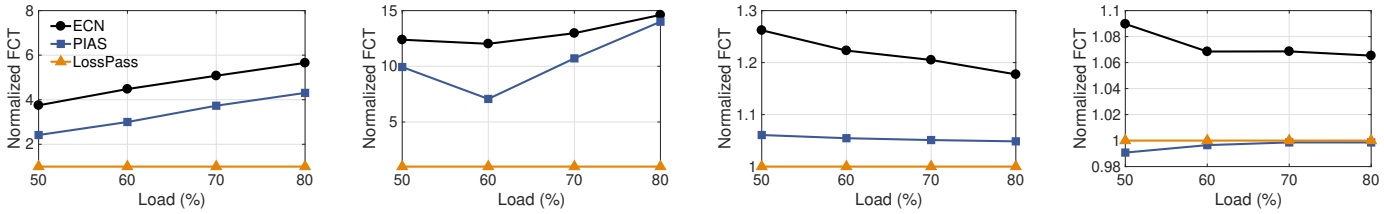
Impact of Minimum Guaranteed Buffer. Fig. 8 shows the FCTs of LossPass with different guaranteed ratios when load is 70%. The FCTs are normalized to the results of PIAS. We set the guaranteed buffer for Q_l to $K_l \times \alpha$ where $\alpha \leq 1$ is the guaranteed ratio. We find that the FCT of large flows is improved as the α grows. However, since large flows are relatively insensitive to packet loss and the workload does not generate extremely large bursts, the degree of change is marginal unlike small flows. This suggests that starvation is rare in production workloads and the minimum guaranteed buffer can prevent starvation by reserving a dedicated buffer for large flows.

5.2 Large-scale ns-2 Simulations

We also perform large-scale simulations using ns-2 [10]. **Methodology.** We build a leaf-spine topology, which has 192 servers, 8 leaf switches, and 2 spine switches. Each leaf switch has 24×10Gbps downlinks and 2×40Gbps. The base RTT is $85.2\mu s$. We use ECMP as the load balancing scheme. We generate 5K flows across 192×191 communication pairs by varying traffic load from 50% to 90%. We use the same transport settings as in the testbed experiments except the following settings. We set RTO_{min} to 5ms, the lowest stable value in `jiffy` timer [26]. Leaf and spine switches have 9MB and 16MB shared buffers, respectively. For buffer management, we use the dynamic threshold algorithm [22].

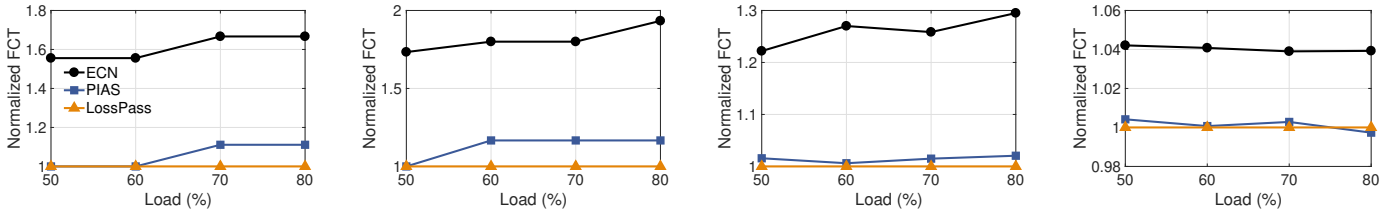
5.2.1 *Dynamic Flow Experiments*

We observe that our simulation results are similar to the experiment results. LossPass works the best on small flows as expected. Fig. 9 (a) and Fig. 10 (a) report the average FCT for small flows. Our scheme performs better than the comparisons within $2.42\times \sim 5.91\times$ in the web search workload. In the data mining workload, the improvement degree is less than that in the web search workload, but LossPass is generally better. For the 99th percentile FCT, LossPass offers more improved performance as shown in Fig. 9 (b) and Fig. 10 (b). For example, in the web search workload, LossPass achieves the 99th percentile FCT better than PIAS by up to $14.03\times$.



(a) (0, 100KB]: Avg (b) (0, 100KB]: 99th Percentile (c) (100KB, 10MB]: Avg (d) (10MB, ∞]: Avg

Fig. 9. [Simulation] FCT statistics across different flow sizes in the web search workload.



(a) (0, 100KB]: Avg (b) (0, 100KB]: 99th Percentile (c) (100KB, 10MB]: Avg (d) (10MB, ∞]: Avg

Fig. 10. [Simulation] FCT statistics across different flow sizes in the data mining workload.

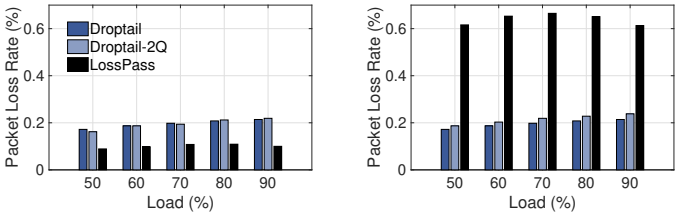
Fig. 9 (c) and Fig. 10 (c) report the results for the average FCT of medium flows. LossPass shows the best performance among comparisons in both workloads. Fig. 9 (d) and Fig. 10 (d) show the average FCT of large flows for both workloads. We find that LossPass performs better than ECN, but has similar performance to PIAS. In the web search workload, compared to ECN, our scheme achieves the average FCT better by $1.07\times$ on average. LossPass and PIAS has similar performance and the maximum gap is only $0.99\times$. The results in the data mining workload are similar to those in the web search workload.

5.2.2 *Root of Performance Gains*

To clarify how does LossPass bring performance gains, we compare LossPass with Droptail and Droptail-2Q. Droptail-2Q uses a droptail scheme using two-level SPQ. We also use Droptail-2Q since LossPass employs two-level SPQ by default. The only difference between LossPass and Droptail-2Q is packet enqueueing decision at the end of ingress pipeline. All of the schemes use TCP without ECN to see the net effect of LossPass.

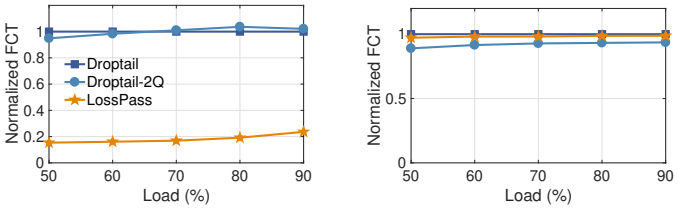
Packet Loss Rate. Fig. 11 shows the packet loss rate of small and large flows in the web search workload. For small flows, LossPass reduces the packet loss rate by as much as half. The range of packet loss rate of Droptail-2Q is between 0.16% and 0.22% whereas that of LossPass is within $0.09\% \sim 0.10\%$. Droptail is similar to Droptail-2Q, but Droptail-2Q results in slightly higher packet loss rate. Not surprisingly, LossPass increases the packet loss rate of large flows. For Droptail and Droptail-2Q, the average packet loss rates of large flows across traffic loads are 0.22% and 0.20%, respectively. In contrast, LossPass results in the average packet loss rate of 0.64%, which is $\approx 2.90\times$ higher than that of Droptail-2Q. The above result indicates that more large flow packets are dropped to avoid timeouts of small flows in LossPass.

Impact on FCT. Fig. 12 shows the average FCT, which is the result of packet loss rate and flow loss rate. Across the traffic loads, LossPass outperforms Droptail-2Q in the average FCT of small flows by $5.57\times$ on average. This shows that reducing the packet loss rate by half results in a significant improvement in the FCT of small flows. For large flows, our scheme performs worse than Droptail-2Q by $0.94\times$ on average while outperforming Droptail by $1.02\times$ on average. This explains why our scheme outperforms Droptail in the average FCT of large flows although LossPass is expected to harm the FCT of large flows. LossPass increases the FCT of large flows slightly, but two-level SPQ compensates the loss.



(a) (0, 100KB] (b) (10MB, ∞]

Fig. 11. [Simulation] Packet loss rate. We can see that LossPass reduces packet loss rate of small flows and increases that of large flows as expected.



(a) (0, 100KB]: Avg (b) (10MB, ∞]: Avg

Fig. 12. [Simulation] The average FCT of small and large flows. We can observe that LossPass significantly reduces the FCT of small flows while degrading the FCT of large flows slightly.

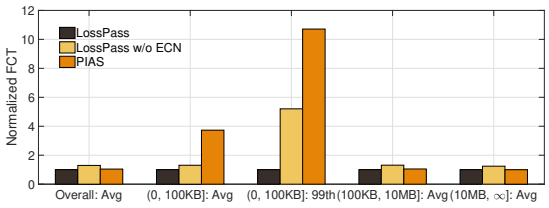


Fig. 13. [Simulation] LossPass with/without ECN and PIAS. ECN-disabled LossPass can offer the better performance than PIAS.

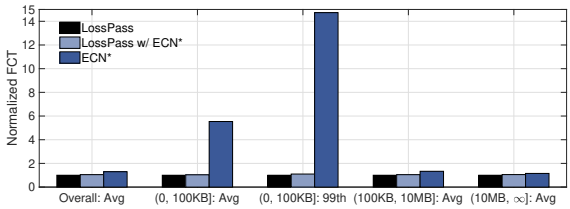


Fig. 14. [Simulation] FCT summary for LossPass, LossPass with/without ECN*, and ECN*. LossPass maintains similar performance regardless of the transport protocol.

5.2.3 Performance Under Different Environments

Impact of ECN. We have enabled ECN for performance evaluation because LossPass is a complementary to ECN. Therefore, one may wonder what if we disable ECN. Fig. 13 shows the FCT results in the web search workload. We breakdown the FCT results with 70% traffic load across different flow sizes. We find that ECN-disabled LossPass shows competitive performance to ECN-enabled LossPass except the 99th percentile FCT of small flows. We also see that ECN-disabled LossPass still beats PIAS in the FCT of small flows whose maximum performance gap is roughly $2.10\times$ in the 99th percentile FCT. However, the average FCTs of overall, medium, and large flows are worse than PIAS.

Impact of Transport Protocol. We now evaluate the performance of LossPass using a different data center transport protocol. While we use DCTCP [5] for the transport protocol, we here employ ECN* [11], another ECN-based TCP variant that uses the standard TCP congestion avoidance algorithm with instantaneous ECN marking. We use K=84 for line-rate throughput. Fig. 14 shows the results. We breakdown the FCT in the web search workload with 70% traffic load across different flow sizes. We find that LossPass and LossPass with ECN* shows similar performance. In addition, both LossPass and LossPass with ECN* outperform ECN* significantly as expected. The results suggest that LossPass works well regardless of transport protocols.

6 Related Work

Microburst Absorption. There exist variants of ECN to absorb microbursts [3], [12], [27], [28], [29]. CEDM [3] alleviates throughput loss caused by LSO using an additional ECN marking threshold at dequeueing. However, the FCT of small flows increases because senders decrease sending rates of large flows more conservatively. BCC [12] adaptively uses two ECN marking schemes, per-port ECN marking for high throughput and shared buffer ECN marking for low latency. However, BCC achieves either low latency or high throughput at a time based on the remaining shared buffer. S-ECN [27], QAECN [29], MBECN [28] are also variants of ECN, which still have the latency-throughput tradeoff. LossPass is complementary to the solutions when ECN is enabled because our solution space is not packet marking but buffer sharing. MATCP [30] is a TCP variant that can adjust congestion window size rapidly to avoid packet losses of small flows. Specifically, MATCP reduces congestion window size in proportion to the slope of queue length evolution, making senders react to microbursts quickly. MATCP is basically a transport protocol and LossPass is a buffer sharing solution. Therefore, LossPass is complementary to MATCP. In a similar vein, LossPass is complementary to existing end-host works that aim at reducing the FCT of small flows (e.g., Qjump [31]) because LossPass adds packet eviction functionality to the switch, not replaces existing switch functionality like packet scheduling or modifies end-host network stacks.

DT [22] and EDT [2] absorb microburst traffic by allowing the port to occupy many buffer space. However, it harms short-term fairness of buffer sharing among ports. LossPass does not harm short-term fairness since we make free buffer space on request. FAB [32] is a dynamic buffer sharing solution that makes a difference in the maximum queue length between small flows and large flows using a weight parameter. Specifically, FAB assigns higher weights to small flows than large flows so that small flows can occupy more buffer space than large flows. FAB determines the port buffer size based on flow information whereas LossPass operates when there is no free buffer space for a port, which means that both the works are orthogonal.

Packet Eviction. Packet eviction has been discussed in various network environments including the Internet, ATM-enabled networks, and data center networks [33], [34], [35], [36], [37]. For example, pFabric [37], a clean-slate approach rearchitecting the whole packet transport, drops a packet with the highest remaining flow size to approximate the optimal flow scheduling for data center networks. Unfortunately, it is hard to implement the above solution at line-rate because of expensive complexity to find a victim packet. pFabric $O(\log n)$ complexity with a binary tree. In addition, pFabric requires non-trivial modifications on end-hosts and switches with impractical assumptions, which include prior knowledge of flow size, μs-scale RTO, NIC modifications, a customized TCP, and millions of hardware queue support. LossPass is distinguished from the existing packet eviction works that our work addresses various design challenges without impractical assumptions. In addition, the problem space is different that we address microburst absorption, not per-flow fairness for the Internet [36] or rearchitecting the whole packet transport [37].

7 Conclusion

We studied how we can absorb microbursts as many as possible while maintaining line-rate throughput. We proposed LossPass, a buffer sharing solution that passes packet loss to large flows to avoid packet loss of small flows. To implement packet eviction, we have addressed essential technical challenges like reducing the search complexity and preventing

multiple packet evictions. We have implemented a LossPass prototype and evaluated its performance through extensive testbed experiments and large-scale simulations. Our evaluation results demonstrated that LossPass can minimize the FCTs of small flows while affecting large flows slightly.

REFERENCES

[1] G. Kim and W. Lee, "Absorbing microbursts without headroom for data center networks," *IEEE Communications Letters*, vol. 23, no. 5, pp. 806–809, May 2019.

[2] D. Shan, W. Jiang, and F. Ren, "Absorbing micro-burst traffic by enhancing dynamic threshold policy of data center switches," in *Proc. of IEEE INFOCOM*, April 2015, pp. 118–126.

[3] D. Shan and F. Ren, "Improving ecn marking scheme with micro-burst traffic in data center networks," in *Proc. of IEEE INFOCOM*, May 2017, pp. 1–9.

[4] Q. Zhang, V. Liu, H. Zeng, and A. Krishnamurthy, "High-resolution measurement of data center microbursts," in *Proc. of ACM IMC*, 2017, pp. 78–85.

[5] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data center tcp (dctcp)," in *Proc. of ACM SIGCOMM*, 2010, pp. 63–74.

[6] Y. Lu, G. Chen, L. Luo, K. Tan, Y. Xiong, X. Wang, and E. Chen, "One more queue is enough: Minimizing flow completion time with explicit priority notification," in *Proc. of IEEE INFOCOM*, May 2017, pp. 1–9.

[7] C. Villamizar and C. Song, "High performance tcp in ansnet," *SIGCOMM Comput. Commun. Rev.*, vol. 24, no. 5, pp. 45–60, Oct. 1994.

[8] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "Vl2: A scalable and flexible data center network," in *Proc. of ACM SIGCOMM*, 2009, pp. 51–62.

[9] W. Bai, K. Chen, H. Wang, L. Chen, D. Han, and C. Tian, "Information-agnostic flow scheduling for commodity data centers," in *Proc. of USENIX NSDI*, May 2015, pp. 455–468.

[10] "The network simulator ns-2," http://www.isi.edu/nsnam/ns/.

[11] H. Wu, J. Ju, G. Lu, C. Guo, Y. Xiong, and Y. Zhang, "Tuning ecn for data center networks," in *Proc. of ACM CoNEXT*, 2012, pp. 25–36.

[12] W. Bai, S. Hu, K. Chen, K. Tan, and Y. Xiong, "One more config is enough: Saving (dc)tcp for high-speed extremely shallow-buffered datacenters," in *Proc. of IEEE INFOCOM*, 2020.

[13] "net-qdisc-hhf: Heavy-hitter filter (hhf) qdisc," https://lwn.net/Articles/577208.

[14] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, "Inside the social network's (datacenter) network," in *Proc. of ACM SIGCOMM*, 2015, pp. 123–137.

[15] M. Park, S. Sohn, K. Kwon, and T. T. Kwon, "Maxpass: Credit-based multipath transmission for load balancing in data centers," *Journal of Communications and Networks*, vol. 21, no. 6, pp. 558–568, 2019.

[16] G. Kim and W. Lee, "Protocol-independent service queue isolation for multi-queue data centers," in *Proc. of IEEE ICDCS*. USA: IEEE, 2020.

[17] G. Judd, "Attaining the promise and avoiding the pitfalls of TCP in the datacenter," in *Proc. of USENIX NSDI*, 2015, pp. 145–157.

[18] H. Xu and B. Li, "Repflow: Minimizing flow completion times with replicated flows in data centers," in *Proc. of IEEE INFOCOM*, April 2014, pp. 1581–1589.

[19] "Intelligent buffer management on cisco nexus 9000 series switches," White Paper, 2017.

[20] S. Hu, W. Bai, B. Qiao, K. Chen, and K. Tan, "Augmenting proactive congestion control with aeolus," in *Proc. of APNet*. New York, NY, USA: ACM, 2018, pp. 22–28.

[21] "Arista 7050x3 series switch architecture," White Paper, 2018.

[22] A. K. Choudhury and E. L. Hahne, "Dynamic queue length thresholds for shared-memory packet switches," *IEEE/ACM Transactions on Networking*, vol. 6, no. 2, pp. 130–140, Apr 1998.

[23] "Tofino programmable switch," https://www.barefootnetworks.com/.

[24] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz, "Forwarding metamorphosis: Fast programmable match-action processing in hardware for sdn," in *Proc. of ACM SIGCOMM*, 2013, pp. 99–110.

[25] W. Bai, L. Chen, K. Chen, and H. Wu, "Enabling ecn in multi-service multi-queue data centers," in *Proc. of USENIX NSDI*, 2016, pp. 537–549.

[26] V. Vasudevan, A. Phanishayee, H. Shah, E. Krevat, D. G. Andersen, G. R. Ganger, G. A. Gibson, and B. Mueller, "Safe and effective fine-grained tcp retransmissions for datacenter communication," in *Proc. of ACM SIGCOMM*, 2009, pp. 303–314.

[27] D. Shan, F. Ren, P. Cheng, R. Shu, and C. Guo, "Micro-burst in data centers: Observations, analysis, and mitigations," in *Proc. of IEEE ICNP*, Sep. 2018, pp. 88–98.

[28] K. Kang, J. Zhang, J. Jin, D. Shen, J. Luo, W. Li, and Z. Wu, "Mbecn: Enabling ecn with micro-burst traffic in multi-queue data center," in *Proc. of IEEE CLUSTER*, Sep. 2019, pp. 1–12.

[29] K. Kang, J. Zhang, J. Jin, D. Shen, R. Xiong, and J. Luo, "Qaecn: Dynamically tuning ecn threshold with micro-burst in multi-queue data centers," in *Proc. of IEEE 23rd International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, May 2019, pp. 398–403.

[30] D. Shan, F. Ren, P. Cheng, R. Shu, and C. Guo, "Observing and mitigating micro-burst traffic in data center networks," *IEEE/ACM Transactions on Networking*, vol. 28, no. 1, pp. 98–111, 2020.

[31] M. P. Grosvenor, M. Schwarzkopf, I. Gog, R. N. M. Watson, A. W. Moore, S. Hand, and J. Crowcroft, "Queues don't matter when you can jump them!" in *Proc. of USENIX NSDI*. USA: USENIX Association, 2015, p. 1–14.

[32] M. Apostolaki, L. Vanbever, and M. Ghobadi, "Fab: Toward flow-aware buffer sharing on programmable switches," in *Proc. of the 2019 Workshop on Buffer Sizing*. New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: https://doi.org/10.1145/3375235.3375237

[33] A. Thareja and A. Agrawala, "On the design of optimal policy for sharing finite buffers," *IEEE Transactions on Communications*, vol. 32, no. 6, pp. 737–740, 1984.

[34] A. K. Choudhury and E. L. Hahne, "Space priority management in a shared memory atm switch," in *Proc. of IEEE GLOBECOM*, 1993, pp. 1375–1383 vol.3.

[35] I. Cidon, L. Georgiadis, R. Guerin, and A. Khamisy, "Optimal buffer sharing," *IEEE Journal on Selected Areas in Communications*, vol. 13, no. 7, pp. 1229–1240, 1995.

[36] D. D. D. Clark, G. Minshall, L. Zhang, S. Shenker, D. C. Partridge, L. Peterson, D. K. K. Ramakrishnan, J. T. Wroclawski, J. Crowcroft, R. T. Braden, D. S. E. Deering, S. Floyd, D. B. S. Davie, V. Jacobson, and D. D. Estrin, "Recommendations on Queue Management and Congestion Avoidance in the Internet," RFC 2309, Apr. 1998.

[37] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker, "pfabric: Minimal near-optimal datacenter transport," in *Proc. of ACM SIGCOMM*, 2013, pp. 435–446.

Gyuyeong Kim received his B.S. and Ph.D. degrees in computer science from Korea University, South Korea, in 2012 and 2020, respectively. He is currently a research professor with Future Network Center at Korea University. His research interests include networked systems, cloud computing systems, and programmable hardware.

Wonjun Lee received his B.S. and M.S. degrees in computer engineering from Seoul National University, South Korea, in 1989 and 1991, respectively; another M.S. degree in computer science from the University of Maryland, College Park, in 1996; and his Ph.D. degree in computer science and engineering from the University of Minnesota, Minneapolis, in 1999. In 2002, he joined the faculty of Korea University, Seoul, where he is currently a professor with the School of Cybersecurity. He has authored or coauthored more than 180 papers in refereed international journals and conferences. He has served as a TPC member of numerous international conferences including IEEE INFOCOM 2008-2021. His research interests include communication and network protocols, optimization techniques in wireless communication and networking, security and privacy in mobile computing, and radio-frequency powered computing and networking.