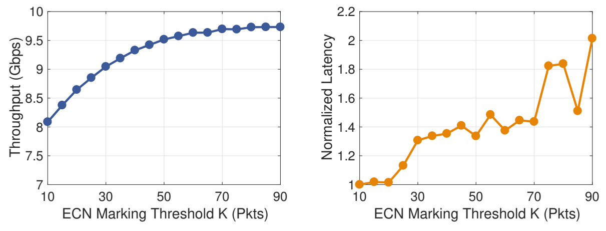
Absorbing Microbursts without Headroom for Data Center Networks

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

***Abstract*—In data center networks, handling microbursts is crucial to user experience because the microburst causes excessive packet losses in shallow buffered switches. Explicit Congestion Notification (ECN) enables the switch to absorb microbursts by leaving buffer headroom, but has a fundamental trade-off between latency and throughput. In this letter, we propose LossPass, a switch port management scheme that achieves low latency and high throughput simultaneously. LossPass breaks the trade-off of ECN by absorbing microbursts without headroom. Intensive benchmark results demonstrate that LossPass reduces the average and tail latency of small flows while maintaining line-rate throughput.**

***Index Terms*—Data center networks, Microbursts, ECN.**



(a) Switch Throughput (b) Query Completion Times

Fig. 1. The trade-off between latency and throughput in ECN. Through ns-2, we simulate a compute rack consisting of 32 servers and a ToR switch. Overall, we generate 10K queries where each of 32 senders responds with 32KB of data. C = 10Gbps, RTT = 100μs, BDP $\approx$ 84pkts.

I. Introduction

MODERN data center applications like web search and cloud storage increasingly require low latency for small flows and high throughput for large flows. The microburst refers to a many-to-one traffic pattern generated by these distributed applications. The aggregator node parallelizes tasks across many worker nodes and the workers return results to the aggregator. Because the processing time between servers is similar, the results arrive at the same output port almost simultaneously [1], [2]. It causes a sudden spike in buffer occupancy and often results in excessive packet losses, which jeopardize the flow completion time (FCT) of small flows and user experience. Unfortunately, today's data center switches are vulnerable to microburst traffic due to their shallow buffers.

ECN is widely employed by data center transport solutions [2]. ECN maintains port queue length low around the ECN marking threshold K, and this leaves buffer headroom to absorb microbursts that is $\approx B$-K where B is the port buffer size. Similarly, Cisco AFD [3] reserves buffer headroom to absorb microbursts. However, the existence of headroom causes a fundamental trade-off between latency and throughput. To achieve line-rate throughput, the switch requires a buffer larger than the Bandwidth Delay Product (BDP), $C \times RTT$ where C is link capacity. When $K \geq$ BDP, the switch can fully utilize link capacity, but cannot absorb microbursts sufficiently due to small headroom size. On the other hand, $K <$ BDP degrades throughput by causing overreaction of senders to congestion signals although the switch can absorb more packets. Fig. 1 summarizes the trade-off of ECN.

We ask the following question: *How can we absorb microbursts while achieving low latency and high throughput simultaneously?* Although there exist variants of ECN [1], [4], no work achieves the requirements simultaneously. CEDM [1] trades the FCT of small flows for high throughput using double marking thresholds. BCC [4] offers either low latency or high throughput at a time, depending on the number of active ports.

In this letter, rather than finding the balance, we break the trade-off of ECN to achieve low latency and high throughput simultaneously. To this end, we propose LossPass, a port management scheme that absorbs microbursts without headroom by shifting the damage of buffer overflow onto large flows.

We make the following contributions:

- With LossPass, we show that it is possible to absorb microbursts without buffer headroom, thereby breaking the fundamental trade-off of ECN. Furthermore, the port buffer can be fully utilized to absorb microbursts and the switch can achieve line-rate throughput.
- LossPass is inspired by the observation that the impact of packet loss on FCT decreases as the flow size increases. Therefore, the FCT of small flows is significantly decreased while large flows are marginally affected. Our intensive benchmark results demonstrate that LossPass achieves the better average and 99th percentile FCT of small flows than compared schemes while keeping link utilization high.

II. The LossPass Design

A. Passing Packet Loss to Large Flows

The core idea of LossPass is shifting the damage of buffer overflow (i.e. packet loss) onto large flows. In the current regime, the switch drops the arriving packet if there is no vacancy in the port buffer. However, with LossPass, the switch

Manuscript received December 19, 2018; revised February 11, 2019; accepted March 19, 2019. Date of publication MM DD, 2019; date of current version March 20, 2019. This research was partly sponsored by the National Research Foundation of Korea (NRF) grant funded by the Ministry of Science and ICT (No. 2017R1A2B2004811), Next-Generation Information Computing Development Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science and ICT (No. 2017M3C4A7083676). The associate editor coordinating the review of this letter and approving it for publication was Ala Al-Fuqaha. *(Corresponding author: Wonjun Lee)*

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

Digital Object Identifier XX.XXXX/LCOMM.2019.XXXXXXX

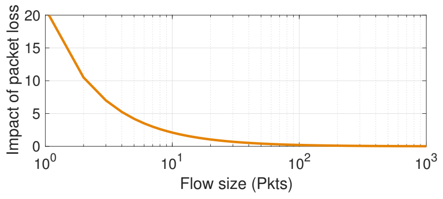


Fig. 2. The impact of packet loss on the FCT. The impact rapidly decreases as the flow size increases, which means that passing packet loss onto large flows causes only marginal FCT degradation for large flows.

drops enqueued large flow packets to enqueue arriving small flow packets. This simple idea brings the following benefits:

- *The switch can absorb microburst traffic without buffer headroom:* Switches do not have to reserve a portion of the port buffer as headroom because LossPass can make rooms for arriving packets by dropping enqueued packets.
- *The port buffer can be fully utilized to absorb microbursts:* With ECN, the upper limit of tolerable burst size is the headroom size. On the other hand, LossPass can absorb the burst up to the port buffer size.
- *The switch can keep link utilization high:* LossPass does not limit the queue length since there is no headroom. The switch can utilize sufficient buffer space to achieve high throughput.

Since large flow packets are dropped instead of small flow packets, it is easy to expect that the FCT would be degraded. Surprisingly, the damage to large flows is marginal because of the following observation: *As the flow size increases, the impact of packet loss on the FCT decreases.*

Let us derive the expected impact of packet loss on the FCT to show the observation. A flow with P packets requires P/W RTTs to finish where W represents the congestion window size. For the sake of simplicity, we assume that the sender uses the constant congestion window size W_c. Then, the FCT of a flow with P packets can be expressed as $FCT_P = \sum_{i=1}^{P/W_c} T_i$ where T_i is the RTT that each window sees. The expected FCT of a flow consisting of P packets can be expressed as

$$E[FCT_P] = \frac{P}{W_c} \cdot E[T] \tag{1}$$

where $E[T]$ denotes the expected RTT.

By considering per-window granularity for packet loss, the expected additional delay caused by packet loss is

$$E[T'] = RTO_{min} + E[T] \tag{2}$$

where RTO_{min} and $E[T]$ denote the loss detection delay by the minimum retransmission timeout and the retransmission delay, respectively. The loss detection delay should be RTO in practice. However, RTO is not constant. Therefore, we approximate it to RTO_{min} for the simplicity. Note that RTO and RTO_{min} are similar because they are in milliseconds while the RTT is in microseconds. From Eq. 1 and Eq. 2, we can derive the expected impact of packet loss on the FCT of a flow with P packets as

$$E[I_P] = \frac{E[T']}{E[FCT_p]} = \frac{RTO_{min} + E[T]}{\frac{P}{w_c} \cdot E[T]} \tag{3}$$

Algorithm 1 The pseudocode of LossPass

```
Require: ⟨Packet⟩$P$                                        ▷ Active packet at a time
 1: ⟨Queue⟩$Q_H, Q_L$                                        ▷ Priority queues
 2: ⟨Packet⟩$P_t$                                            ▷ Tail packet in queue
 3: if $P$.bytes + $\Sigma Q_i$.bytes > Buffer.bytes then     ▷ Buffer Overflow
 4:     if $P$.Priority = $H$ then                            ▷ Small flow packet
 5:         while $P$.bytes + $\Sigma Q_i$.bytes > Buffer.bytes do
 6:             if $P$.bytes < $Q_L$.bytes then
 7:                 Evict($Q_L.P_t$)                          ▷ Evict buffered packet
 8:             else                                         ▷ No more packets to be evicted
 9:                 return Drop($P$)
10:             end if
11:         end while
12:     else                                                 ▷ Large flow packet
13:         return Drop($P$)
14:     end if
15: end if
16: return Enqueue($P$)                                      ▷ Enqueue the packet
```

Fig. 2 shows $E[I_P]$ as P increases where $w_c = 1$, $RTO_{min} = 5ms$ [5], and $E[T] = 250\mu s$ [2]. In this setting, $E[T'] = 21E[T]$. We can find that the impact of packet loss rapidly decreases as the flow size increases. For example, $E[I_1] = 21$ for a small flow with $P = 1$. It means that a single loss can slow down the small flow by 22×. However, for a large flow whose $P = 1000$, the impact is almost negligible that $E[I_{1000}] = 0.021$. The implication of the above analysis is obvious that when packet loss occurs, shifting $E[T']$ onto large flows causes only slight impact on the FCT of large flows, but significantly decreases the FCT of small flows.

B. LossPass as a practical solution

1) Reducing Complexity: Today's data center switches should process billions of packets per second at line-rate. Therefore, complex and expensive operations are undesirable. A key operation of LossPass is packet search operation because we must find large flow packets in the queue before dropping. The queue is an unsorted list consisting of a mixture of small and large flow packets. Therefore, the complexity for searching a packet is $O(n)$ where n is the maximum queue length in number of packets. However, it is burdensome to perform the expensive $O(n)$ operation every dropping decision.

To this end, we approximate the packet search operation by employing two-level strict priority queueing (SPQ). Data center switches provide SPQ as a built-in function and also have 4-8 priority queues per port. We assign the high priority queue Q_H for small flows and the low priority queue Q_L for large flows. In this regime, Q_H and Q_L contain only small flow packets and large flow packets, respectively. Thus, by pointing the tail packet at Q_L, we can instantly find the large flow packet without the expensive search operation. Alg. 1 describes the pseudocode of LossPass. We note that LossPass operation performs only if the following conditions are met: 1) The buffer has no vacancy (Line 3, 5); 2) The priority of arriving packet is high (i.e. small flow packet) (Line 4); 3) The low priority queue is not empty (Line 6).

2) Priority Assignment: We need to know flow size information to classify flows into priority queues. The flow size in many applications is not known in advance. To assign the priority without prior knowledge of flow sizes, we employ an end-host based flow size estimation method [6]. Each connection accumulates the number of bytes sent and tags the corresponding priority at DSCP field in the IP header for each outgoing packet. If the bytes sent is less than the threshold T, the packet is tagged with the high priority and vice versa. The switch can distinguish small flows from large flows by checking the DSCP field value before enqueueing.

3) Implementation: LossPass can be implemented inexpensively in existing hardware. Since we avoid the expensive packet search operation by utilizing SPQ, the only additional operation caused by LossPass is dropping enqueued large packets. Removing the enqueued packet is a simple and stateless operation, which requires no additional registers. This operation is exactly the same as dequeuing operation except that ours targets the tail packet, not the head packet. Therefore, the complexity of removing a packet at tail from the queue is just $O(1)$, which is also same as that of dequeueing. We leave hardware implementation of LossPass as a future work.

We have implemented a software prototype of LossPass using a server-emulated switch with multiple Network Interface Cards (NICs). LossPass is implemented on the top of a Linux `qdisc` kernel module [7]. The key stages of the `qdisc` kernel module are as follows.

Classification: The module implements multiple priority queues by creating multiple transmission queues for each NIC. When the packet sent from IP layer arrives, the packet classifier returns the index of the corresponding priority queue by referring to the DSCP field in the IP header.

Enqueueing: 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 port buffer size. In LossPass, if the queue index of the packet points the high priority queue and there is no free space, the packet eviction performs until there is enough free space for the arriving packet or the low priority queue is empty. To evict the buffered tail packet, we call `qdisc_dequeue_tail()` function. When the arriving packet is finally enqueued, ECN marking also performs if we have enabled ECN.

Dequeueing: The scheduler dequeues the packet from the highest priority queue. The dequeued packet goes to NIC driver and NIC hardware before it is transmitted to the wire. The `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 avoid excessive buffering on NIC driver TX ring buffer and NIC hardware.

4) Comparison to pFabric: pFabric [8] is a clean-slate transport design that uses priority scheduling and priority dropping. Basically, the switch serves a packet with the lowest remaining flow size first and drops a packet with the highest remaining flow size when the buffer is full. While pFabric provides near-optimal performance, but it is difficult to implement because of impractical assumptions and non-trivial modifications. LossPass is practical that has no assumptions and requires no NIC modification or a customized TCP. Table I summarizes the differences between LossPass and pFabric.

TABLE I
DIFFERENCES BETWEEN LOSSPASS AND PFABRIC

Assumptions and Requirements	pFabric [8]	**LossPass**
Prior knowledge of flow size	Yes	No
NIC modification	Yes	No
Customized TCP	Yes	No
Microsecond-scale RTO	Yes	No
Required # of queues	# of flows	2
Enqueueing/dequeueing complexity	$O(log_2N)$	$O(1)$

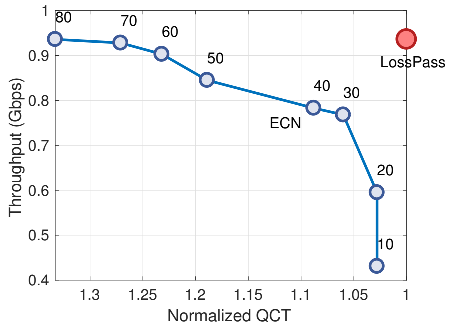


Fig. 3. [Experiment] Switch throughput and the average QCT for each of the schemes. Each number on data points of ECN denotes a marking threshold K in packets. The x-axis is reversed so that lower QCTs are to the right.

III. PERFORMANCE EVALUATION

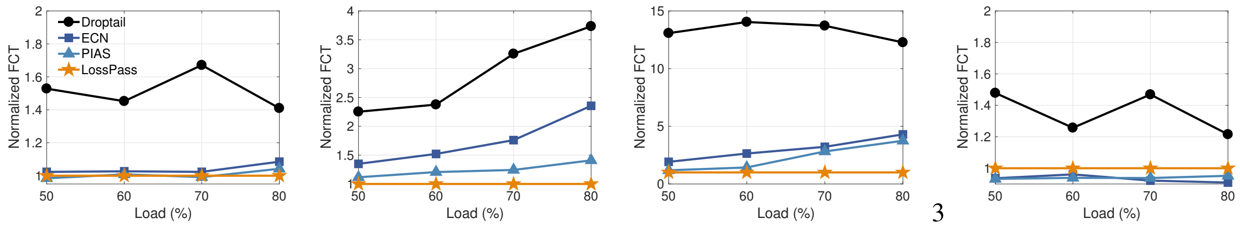
We evaluate the performance of LossPass using testbed experiments and ns-2 simulations. We compare LossPass with the following schemes: Droptail, ECN, and PIAS [6]. Our primary performance metric is the FCT. We also consider throughput and query completion times (QCT). For clear comparison, the results are normalized by the values of LossPass.

A. Testbed experiments

Testbed Setup: We have built a small-scale testbed consisting of 5 servers connected to a server-emulated switch. The servers run Ubuntu 14.04 LTS with Linux 3.18.11 kernel. The switch is equipped with multiple Intel Gigabit NICs. We have disabled the NIC offloading techniques to emulate switch hardware behaviors more correctly. The base RTT is ≈500us.

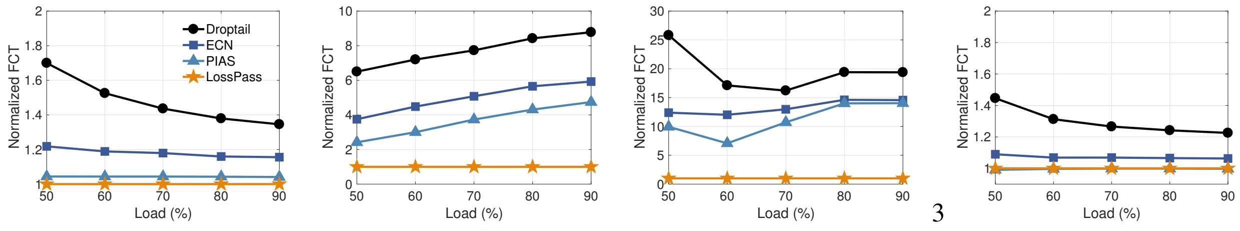
Switch and Transport: Each port in the switch has the 128KB of buffer shared by queues. We use 1MB for the priority assignment threshold as suggested in PIAS paper [6]. We use DCTCP [2](K=45 for line-rate throughput) for macro-benchmarks except Droptail that uses TCP. In micro-benchmarks, we use TCP to see the performance gain of passing packet loss to large flows clearly. We set RTO_{min} to 10ms as suggested in an existing work [2].

1) Micro-benchmarks: We first inspect achievable throughput on the bottleneck link. Using `iperf`, two senders transmit flows to a receiver for 10 seconds. We measure aggregate goodput per second at the receiver. Next, we measure the average QCT, which represents application-level latency. We first generate two long-lasting background flows to the receiver using `iperf`. Next, using a client-server application, the receiver requests a 1MB query to each sender, and each of senders immediately responds with 1MB of data simultaneously. Each query can be completed only if all response flows



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

Fig. 4. [Experiment] FCT statistics across different flow sizes in small-scale testbed experiments.



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

Fig. 5. [Simulation] FCT statistics across different flow sizes in large-scale simulations.

are finished. The receiver issues the subsequent query when the current query is completed. We generate 1K queries and calculate the average completion time of all queries. Fig. 3 shows throughput and the average QCT for the different schemes. It is easy to see that LossPass is the best in both throughput and the average QCT, which suggests that our solution successfully breaks the trade-off of ECN.

2) Macro-benchmarks: In this experiment, the receiver requests a data to the four senders and each of senders responds with data whose flow size follows a workload derived from production data centers running web search [2]. We generate 10K flows by varying traffic loads from 0.5 to 0.8.

Fig. 4 shows that our experiment results. For the average and 99th percentile FCT of small flows, LossPass achieves the best performance among comparisons. For example, LossPass outperforms PIAS by up to 1.41× and 3.75× in the average and 99th percentile FCT, respectively. The FCT results of overall and large flows are generally similar. A notable point is that LossPass underperforms ECN and PIAS. This is not surprising because LossPass trades the FCT of large flows to protect small flows. The maximum performance gap is 0.93×.

B. Large-scale simulations

To examine how LossPass works in large-scale data centers, we build a leaf-spine topology having 192 servers, 8 leaf switches, and 2 spine switches using ns-2. Leaf and spine switches have 9MB and 16MB shared buffers, respectively. Each leaf switch has 24 10Gbps downlinks and 2 40Gbps uplinks yielding oversubscription ratio of 3:1. The base RTT is approximately $85.2\mu s$. Load balancing is done by ECMP. We set the ECN marking threshold K to 65 packets. The other settings are the same as those in the macro-benchmark in the testbed experiment. We generate 5K flows across 192×191 communication pairs by varying traffic load from 50% to 90%.

Fig. 5 reports the simulation results. we can observe that its tendency observed in Fig. 4 still holds in large-scale data centers. LossPass is better than ECN and PIAS by up to 5.91× and 4.74× in the average FCT of small flows. For the 99th percentile FCT of small flows, LossPass achieves the better performance than PIAS by up to 14.03×. For the average FCT of large flows, the difference between PIAS and LossPass is very marginal. The maximum gap is 0.99× when load is 0.5.

IV. Conclusion

In this letter, we broke the fundamental trade-off of ECN with LossPass. LossPass shifts packet loss onto large flows, thereby absorbing microbursts without headroom. Our evaluation results demonstrated that LossPass reduces the average and 99th percentile FCTs of small flows while maintaining line-rate throughput.

References

[1] D. Shan and F. Ren, "Improving ecn marking scheme with micro-burst traffic in data center networks," in *Proc. of IEEE INFOCOM*, pp. 1–9, May 2017.
[2] 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.
[3] "Intelligent buffer management on cisco nexus 9000 series switches." White Paper, 2017.
[4] W. Bai, K. Chen, S. Hu, K. Tan, and Y. Xiong, "Congestion control for high-speed extremely shallow-buffered datacenter networks," in *Proc. of the First Asia-Pacific Workshop on Networking (APNet)*, pp. 29–35, 2017.
[5] 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*, pp. 303–314, 2009.
[6] 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*, (Oakland, CA), pp. 455–468, May 2015.
[7] W. Bai, K. Chen, L. Chen, C. Kim, and H. Wu, "Enabling ecn over generic packet scheduling," in *Proc.of ACM CoNEXT*, pp. 191–204, ACM, 2016.
[8] 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*, pp. 435–446, 2013.