# PABO: Congestion Mitigation via Packet Bounce

Xiang Shi*‖, Lin Wang†, Fa Zhang‡, Kai Zheng§ and Zhiyong Liu*¶
*Beijing Key Laboratory of Mobile Computing and Pervasive Device,
Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China
†Technische Universität Darmstadt, Germany
‡Key Laboratory of Intelligent Information Processing, ICT, CAS, Beijing, China
§Huawei Technologies, Beijing, China
¶State Key Laboratory for Computer Architecture, ICT, CAS, Beijing, China
‖University of Chinese Academy of Sciences, Beijing, China

***Abstract*—Today's data center applications can generate a diverse mix of short and long flows. However, switches used in a typical data center network are usually shallow buffered in order to reduce queueing delay and deployment cost. As a result, the buildup of the queues by long flows can block short flows, leading to frequent packet losses and retransmissions, which translates to crucial performance degradation. While multiple end-to-end TCP-based solutions have been proposed, none of them have tackled the real challenge: reliable transmission in the network. In this paper, we fill this gap by presenting PABO – a novel link-layer design that can mitigate congestion by temporarily bouncing packets to upstream switches. PABO's design fulfills the following demands: $i$) providing per-flow based flow control on the link layer, $ii$) handling transient congestion without the intervention of end devices, and $iii$) gradually back propagating the congestion signal to the source when the network is not capable to handle the congestion. We complete a proof-of-concept implementation, and experiments under different severities of congestion show that PABO outperforms the standard unreliable link-layer protocol by guaranteeing zero packet loss while introducing only a reasonable stretch on packet delay.**

## I. Introduction

Today's modern large-scale data centers are equipped with a dedicated network built with commodity switches. This network (a.k.a. Data Center Network, DCN) is required to support high-speed communications between servers with high availability in data centers. On the one hand, data center applications such as web search and online social networks usually generate a diverse mix of short and long flows, demanding high utilization for long flows, low latency for short flows, and high burst tolerance [1]. On the other hand, switches used in a DCN are usually shallow buffered due to the considerations on queueing delay and deployment cost. It has been demonstrated by [1] that the greedy fashion of the traditional TCP and its variants cannot satisfy all those demands and thus, various TCP-like protocols such as DCTCP [1] and ICTCP [2] have recently been proposed dedicatedly for DCN environments. However, none of the proposals can guarantee one hundred percent prevention of packet losses or timeouts [3] – the main culprit for performance degradation in DCNs, leaving poor performance in most cases.

The need of end-to-end solutions (on the transport layer) for reliable data transmission comes from the fact that reliable point-to-point transmission mechanisms (on the data link layer) are not available in the current protocol stack. When excessive packets that are destined to the same next-hop arrive at a switch, the corresponding output buffer will be saturated. If packet processing in the link is slower than packet arrival, the output buffer will overflow and the subsequently arrived packets will be dropped. The source will not be explicitly notified about this congestion and thus will have no knowledge of when and where packet losses have happened. The retransmission of the dropped packets will be handled by upper-layer congestion control protocols (e.g., TCP, DCCP), typically through a timeout-based pathway.

As one of the very few proposals toward the goal of providing reliability on the link layer, PAUSE Frame [4] allows a switch to send a special frame (namely PAUSE frame) to its upstream switches, which results in a temporary halt of the data transmission. However, all the flows on the same link will be affected without considering their contributions on the congestion. To alleviate this situation, PFC (Priority Flow Control) [5] further extends the idea to support eight service classes and consequently, PAUSE frames can be sent independently for each service class. Despite the lack of per-flow control, the parameters in PFC have to be carefully tuned individually according to each network circumstance in order to guarantee congestion-free [3].

In this paper, we propose PAcket BOunce (PABO), a novel link-layer protocol design that can provide reliable data transmission in the network. Instead of dropping the excess packets when facing buffer saturation, PABO chooses to bounce them to upstream switches. On the one hand, transient congestion can be mitigated at a per-flow granularity, which can help achieve significant performance gain for short-lived flows as in the typical incast scenario [6] in a DCN. On the other hand, the congestion is gradually back propagated toward the source and can finally be handled by the source if the congestion cannot be solved right in the network. To the best of our knowledge, PABO is among the first solutions for reliable transmission and in-network congestion mitigation.

The Corresponding Author is Zhiyong Liu (zyliu@ict.ac.cn). This work has been supported partially by the National Natural Science Foundation of China (NSFC) Major International Collaboration Project 61520106005, NSFC Project for Innovation Groups 61521092, and the Collaborative Research Center 1053 (MAKI) of the German Research Foundation (DFG).

Our contributions can be summarized in three aspects: $i$) We propose a novel link-layer protocol, PABO, that supports full reliability and can handle transient congestions in the network. $ii$) We present the design of PABO and explain its components in detail. $iii$) We carry out extensive simulations to validate the basic properties of PABO and evaluate its performance.

The rest of the paper is organized as follows: Section II summarizes related work. Section III describes the design rationale of PABO and details its components one by one. Section IV discusses the implementation of PABO and Section V presents the experimental results. Section VI concludes the paper and discusses future work.

## II. Related Work

We summarize some representative works on congestion control in data center networks and make a comparison with PABO in this section.

**Transport layer.** As the most frequently used transport layer protocol, TCP provides reliable end-to-end communication on unreliable infrastructures. Despite several variants of the traditional TCP protocol, the reactive fashion to congestion (i.e., timeout) and the slow-start nature in adjusting the sending window size cannot satisfy the growing requirements for small predictable latency and large sustained throughput in data center environments [1]. ICTCP [2] aims at preventing incast congestion through adjusting the advertised windows sizes at the receiver side by estimating the available bandwidth and RTT. DCTCP [1], another TCP variant developed for data center environment, take advantage of the Explicit Congestion Notification (ECN) feature [12] on switches to predict the extent of the congestion and provide smooth adjustments on the sending window size accordingly. These end-to-end solutions do not assume any reliability in the network and thus are orthogonal to PABO.

**Network layer.** AQM (Adaptive Queue Management) is an intelligent probabilistic packet dropping mechanism designed for switch buffer to avoid global synchronization among flows as can be frequently seen in traditional drop-tail queue settings (e.g., RED [13] and PI [14]). ECN (Explicit Congestion Notification) [12] allows end-to-end congestion notification and ECN-enabled switches can set a marker (i.e., CE) in the IP header of the packet to signal impending congestion. This information will be echoed back to the sender with the ACK for this packet. While both can provide packet drop prevention at some degree, there is no guarantee on that no packet will be dropped. Fastlane [15] is an agile congestion notification mechanism, which aims at informing the sender as quickly as possible to throttle the transmission rate by sending explicit, high-priority drop notifications to the sender immediately at the congestion point.

**Link layer.** This research line aims at providing hop-by-hop reliability inside the network through a backpressure-alike feedback loop. Among them, PAUSE Frame [4] is one of the flow control mechanisms for Ethernet, basing on the idea of sending PAUSE frame to the upstream switch to avoid buffer overflow. However, PAUSE frame is per-link based and cannot differentiate among flows. PFC (Priority Flow Control) [5] further extends to provide individual flow control for several pre-defined service class. While it brings about some mitigation on inter-flow interference, the number of service class is still not enough in many circumstances. Moreover, the parameters of both PAUSE frame and PFC are very difficult to tune to ensure full reliability, making them unpractical [16].

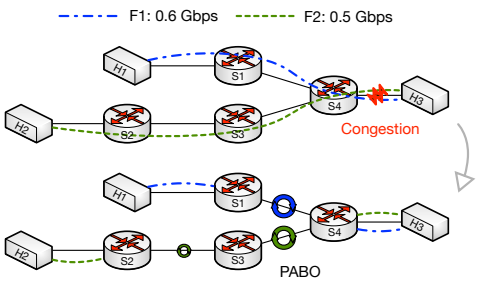



Fig. 1. A motivating example to demonstrate the idea of PABO, i.e., solving network congestion without dropping packets. We assume GigE links in the network.

DIBS [3] solves local and transient congestions by detouring packets via a random port at the same switch when congestion occurs on a link. While adopting a similar idea of sharing switch buffers to mitigate transient congestion, PABO has two major additional merits: $i$) Packets are detoured by bouncing to upstream switches, which is more manageable and can minimize inter-flow interference; $ii$) The bounced packets can also serve as a congestion notification for upstream nodes (switches or end-hosts).

## III. PABO's Design

PABO is a link-layer solution for congestion mitigation based on back-propagation. Particularly, it is well suited for data center environments, where transient congestions (e.g., in-cast congestion) are commonly presented [7], [8]. In addition, upper-layer congestion control protocols can be incorporated to achieve smooth congestion handling in all circumstances by taking advantage of the back propagation nature of PABO.

### A. An Example

We provide a motivating example and explain why PABO is superior in handling congestion by providing reliable transmission in communication networks. Assume a GigE network and consider the scenario in Figure 1, where two short flows F1 (with rate 0.6 Gbps) and F2 (with rate 0.5 Gbps) that consist only tens of packets are destined to the same host H3. When congestion occurs on the link S4 - H3, PABO will bounce some of the packets to upstream switches (i.e., S1 and S3, respectively). The number of bounced packets will depend on the congestion level. Packet bouncing can be contagious reversely along the forwarding path. e.g., from S3 to S2. The bounced packets will then be forwarded to H3 again. Consequently, there will be packets bouncing back and forth on the few links in the network until the congestion vanishes.

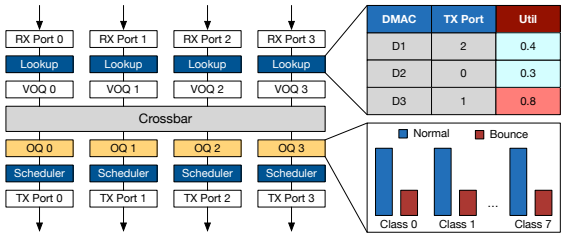



Fig. 2. An overview of PABO's design based on a Combined Input and Output Queue (CIOQ) switch model.

### B. Overview

We assume a general Combined Input and Output Queue (CIOQ) switch model [9] as depicted in Figure 2. This simplified switch model contains following modules: the FIB (Forwarding Information Base, also known as forwarding table) which contains MAC address-to-port mappings obtained from MAC address learning, the lookup unit, the virtual output queues, the output queues, and the output schedulers. Upon the arrival of a packet from any input interface, the packet will first go through the lookup unit. The lookup unit decides the output port for the packet by querying the FIB. Next, the packet will be passed into the corresponding virtual output queue (illustrated as VOQ in Figure 2). The packet will then be sent to the corresponding output queue through the crossbar and finally to the output interface by the output scheduler. Our design of PABO involves modifications on the following components of traditional switches: the lookup unit, the FIB, the output queue and output scheduler, and packet structure. We will discuss them one by one in the following subsections.

### C. Lookup Unit

By querying the FIB, the lookup unit obtains the output port number for a packet corresponding to the packet's destination. Instead of forwarding the packet directly to its destined port, PABO introduces a probabilistic decision making process to decide where to forward the packet. Upon packet arrival, the lookup unit calculates a probability with which the packet will be bounced to its previous-hop switch. The calculation is based on a probability function $P$, which follows the principles:

**PRCP-1:** The probability should be zero when the queue is almost free and should be one when the queue is full;

**PRCP-2:** The probability should increase super-linearly with the queue utilization to prevent the queue from fast buildup;

**PRCP-3:** Packets that have already been returned should receive smaller probability to be bounced back again.

To satisfy PRCP-3, we first define a base probability $P_b$ for each packet according to the distance that the packet has been bounced away from the congestion point, i.e., $n_p$. When $n_p$ grows, the base probability should decrease dramatically to intentionally reduce the chance of a packet being persistently bounced, as it could result in a large delay. To this end, we use an exponential decay function in the following form for the base probability:

$$P_b(n_p) = \frac{1}{e^{\lambda \cdot (n_p+1)}} \tag{1}$$

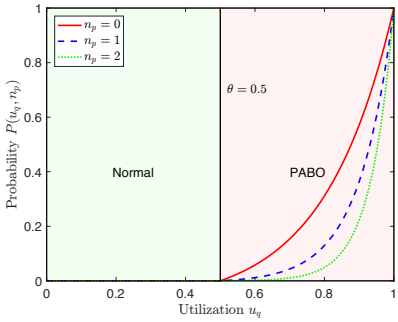



Fig. 3. Probability function $P(u_q, n_p)$ for packet bounce decision making. We set the threshold $\theta$ to 0.5, the constant $\lambda$ to 5, and we show the curves in cases of $n_p = 1, 2, 3$, respectively. When $u_q \leq 0.5$, switches forward packets normally; PABO is only involved when $u_q > 0.5$.

where $\lambda$ is the exponential decay constant. It is always true that $P_b(n_p) \in (0, 1]$ for any $n_p \in \mathbb{Z}_0^+$.

We introduce a lower threshold $\theta \in [0, 1)$ for the output queue utilization $u_q$ and define $P(u_q, \cdot) = 0$ if $0 \leq u_q \leq \theta$ and $P(u_q, \cdot) = 1$ if $u_q = 1$. The first equation means that if the queue utilization is under the predefined threshold, i.e., the queue is underutilized, there is no need to bounce packets; the second equation guarantees that when the queue is full or overflows, all the upcoming packets need to be bounced to avoid packet drop. These two equations ensure the validity of PRCP-1. To satisfy PRCP-2, we define for $\theta < u_q < 1$,

$$P(u_q, \cdot) = \alpha \cdot P_b^{-u_q} + \beta \tag{2}$$

where $\alpha$ and $\beta$ are constants. Noting that $P(u_q, \cdot)$ also satisfies $(\theta, 0)$ and $(1, 1)$, we have

$$\alpha = \frac{P_b^{\theta+1}}{P_b^{\theta} - P_b}, \beta = \frac{-P_b}{P_b^{\theta} - P_b}. \tag{3}$$

Substituting $P_b$ with Equation (1) and combining all the above cases, we have the closed form of $P$ as

$$P(u_q, n_p) = \begin{cases} 0 & 0 \leq u_q \leq \theta, \\ \dfrac{e^{\lambda(n_p+1)(u_q-\theta)} - 1}{e^{\lambda(n_p+1)(1-\theta)} - 1} & \theta < u_q \leq 1. \end{cases} \tag{4}$$

The curves of function $P$, with $\theta = 0.5$, $\lambda = 5$, and $n_p = 0, 1, 2$, under different utilization $u_q$ are illustrated in Figure 3. Note that $\theta$ has a major impact on the proportion of packets that will be bounced, while $\lambda$ dictates the total number of hops each bounced packet will traverse. We will verify these correlations in Section IV.

### D. FIB

The FIB maintains mappings between the packet destination MAC address and the corresponding output port (i.e., interface) that the packet should be forwarded to. In PABO's design, the lookup unit relies on two parameters, $u_q$ and $n_p$, to make the forwarding decision for each packet as we just discussed. While $n_p$ can be obtained from the packet as we will describe in Section III-F, $u_q$ needs to be available after the inquiry to the FIB from the lookup unit. To achieve this, we introduce an extra column named "Util" in the forwarding

table, as shown in Figure 2. In addition to maintaining the MAC-port mappings, the FIB also monitors and updates the output queue utilization $u_q$ for each output port. When a packet is forwarded to an output queue, the utilization $u_q$ of the queue will be updated by the following equation $u_q \leftarrow u_q + 1/C_q$, where $C_q$ is the maximum capacity of the queue. When a packet is expelled by the scheduler at an output queue, the corresponding value of $u_q$ in the FIB is also updated according to the following equation $u_q \leftarrow u_q - 1/C_q$.

### E. Output Queue

We separate the bounced packets from the normal packets by assigning the bounced packets higher priority at the output queue (illustrated as OQ in Figure 2). This is due to the observation that compared to normal packets, packets that have already been bounced should be processed earlier as they have already been delayed during the bouncing process. To this end, we introduce two virtual sub-queues for the output queue, namely bounce queue and normal queue (for every service class). The packets in the bounce queue will enjoy higher priorities when being scheduled by the output scheduler. For simplicity we adopt a straightforward scheduling strategy for the moment and we modify the output scheduler such that packets from the normal queue will be transmitted only if the bounce queue is empty. In a sequel, the bouncing delay will be compensated by reducing their queueing delay during their retry process.

### F. Packet

Each packet in the network will carry a counter $n_p$ to indicate how far (measured by the number of hops) it has been bounced from its last normally reached switch before it was first bounced. Take again the example in Figure 1. When congestion occurs at $S_4$-$H_3$, suppose the last normally reached switch for the packets from both flows $F_1$ and $F_2$ is $S_4$. For packets from flow $F_2$ that are bounced by $S_4$ to $S_3$, the value of $n_p$ will be increased by one and will become two if there are bounced packets further arriving at $S_2$. When the bounced packets are forwarded out normally, the value of $n_p$ will be decreased and will finally become 0 when the packets arrive at $S_4$ again. This counter $n_p$ is used by the lookup unit as an input for the probability function.

### G. End-host Support

When persistent congestion occurs, there will be packets bounced back to the source. While end-host involvement is required to support this circumstance, the bounced packets can also serve as a congestion notification for upper-layer congestion control protocols. When receiving a bounced packet, the source will be notified that the congestion has happened along the forwarding path, and exceeds the ability of the network to handle it. In such cases, the source would reduce its sending rate. The extent of this rate adjustment will depend on the number of bounced packets the source has received during a certain amount of time. We claim that more sophisticated transport layer congestion control protocols can also be incorporated to further handle the congestion smoothly. The bounced packets received by the source will be stored into the source's output queue again for further retransmission, which also ensures that no packets will be dropped.

## IV. Implementation

To validate the effectiveness of PABO, we completed a proof-of-concept implementation based on the INET framework for OMNeT++ [10]. The code is open-sourced at [11]. By overriding the corresponding link-layer modules, we created Ethernet switch and host models that can support PABO. The detailed modifications made to each module will be explained in the following.

**EtherSwitch.** We mainly modify the implementation of its submodules including `MACTable`, `MACRelayUnit`, `EtherQosQueue`. We introduce a float-point variable for each entry in the `MACTable` module to keep track of the output queue utilization at the corresponding output interface. In `MACRelayUnit`, We alter the implementation of the forwarding strategy (i.e., function `handleAndDispatchFrame`) by applying our probability-based forwarding decision making mechanism. The utilization variable in the `MACTable` is also updated accordingly after a forwarding decision has been made. Furthermore, we disable the MAC address self-learning process when receiving bounced packets, as the destination addresses of bounced packets are already in the forwarding table. `EtherQosQueue` is a typical buffer module type, which is composed of classifier, queue, and scheduler. In addition to the default `dataQueue` (as `normalQueue`), we introduce another queue called `bounceQueue`. We alter the classifier in order to separate bounced packets from normal ones. Bounced packets are stored in `bounceQueue`, while normal packets are sent into `normalQueue`. Finally, we modify the output scheduler `PriorityScheduler` where we give priorities to the packets in the `bounceQueue` to reduce delay.

**EtherFrame.** This is a message type representing link-layer frame. It contains the common header fields and payloads. To keep track of the value of $n_p$, i.e., the distance between the packet and the first congested link on its forwarding path, we add a non-negative integer counter `bouncedHop` in the header. This counter will increase by one if the packet is bounced and will decrease by one if the packet is normally forwarded. Then we introduce a parameter `maxBouncedHop` for each packet to record the farthest distance it has been bounced from the congestion point. We also add a non-negative integer counter `totalHop` to record the total number of hops (normal plus bounced, including the sender) that the packet has traversed in the network, which will be used for analysis.

**EtherHost.** The senders and receivers are all `EtherHost` type – an example host with an Ethernet traffic generator `EtherTrafGen` module, an `EtherEncap` module, a buffer queue, and an Ethernet interface. We modify the implementation of the `EtherHost` to generate our sender model. Ideally, PABO is so far only used for mitigating transient congestion in the network without the involvement of end-hosts. However, it is possible that the congestion condition persists too long and

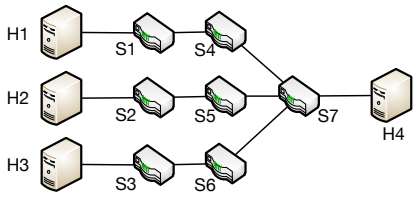



Fig. 4. The network topology used for evaluating the performance of PABO.

the bounced packets will finally reach the sender. To handle this situation, we modify the `EtherEncap` module to check whether there is bounced packet or not. If so, the bounced packets will be sent to the sender's buffer for resending. Note that the type of sender's buffer is also `EtherQosQueue`. As a result, the same modifications we made for switches will also be applied for `EtherHost`.

## V. Experiments

We conduct simulation studies to evaluate the performance of PABO and we report the experimental results in this section.

### A. Simulation Setup

We use a tree-based network topology consisting of three senders, one receiver, and seven switches, as depicted in Figure 4. Each sender is three-hop away from the receiver and the data from all the senders will be aggregated at the last-hop switch connected to the receiver. We consider this topology as partition/aggregate traffic patterns, which can be easily emulated and conveniently monitored. The duration of all the simulations is set to one second, which is sufficient to cover multiple appearances of periodic congestions. The links in the network are assumed to have the same rate of 1Gbps.

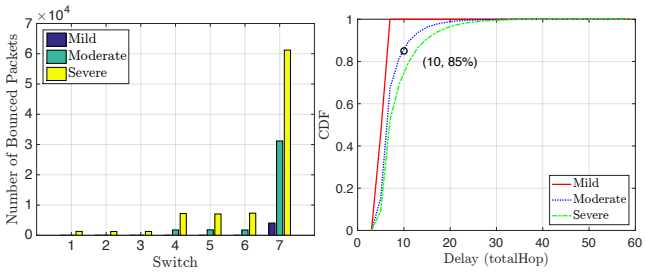
**Traffic.** We imitate a periodic uniform flow by altering the traffic generating module. We fix the burst number to be one and set the `sendInterval` to small values. Then, we introduce a new parameter `numPacketsPerGenerate` to specify the number of packets to be sent in each generating. The duration of each generating can be calculated by `numPacketsPerGenerate` $\times$ `sendInterval`. As the overlap of two consecutive generatings may bring persistent congestions which will overflow the sender's buffer, we modify the `EtherTrafGen` module to enable *pause* after every generating. To guarantee fine-grained control over the sending rate, we fix the value of `sendInterval` to be $10\mu$s. Then, we control the rate by tuning `numPacketsPerGenerate` to simulate congestions in different severities. We set `pauseInterval` to be 0.2s, which is sufficient to avoid overlaps between two consecutive congestions.

**Queue.** The input and output queues of both hosts and switches are with the `DropTailQueue` type. The capacities of `normalQueue` and `bounceQueue` in switches are set to be 500 by default. And we specially allocate larger buffers of size 1500 to both `normalQueue` and `bounceQueue` in senders to avoid packet loss at the sender side.

**Packets.** All the packets generated are in the IEEE 802.3 frame formats with the payload size set to be 1500 bytes.

TABLE I
Distribution of `maxBouncedHop`

| Scenario | # of `maxBouncedHop` | | | |
|---|---|---|---|---|
| | **0** | **1** | **2** | **3** |
| Mild | 45.91% | 54.09% | – | – |
| Moderate | 15.26% | 84.74% | – | – |
| Severe | 9.20% | 85.14% | 5.66% | – |




(a) (b)

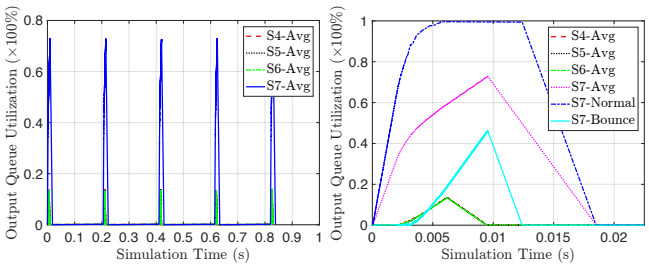
Fig. 5. Performance results of PABO: (a) Packet bounce frequency under different severities of congestion, and (b) the CDF of the end-to-end delay measured by `totalHop`.

### B. Effectiveness under Different Congestion Severities

We validate the effectiveness of PABO by comparing it to the standard link-layer protocol under three different severities of congestion. Parameters of probability function $P$ are fixed as $\lambda = 2$, $\theta = 0.6$. We set `numPacketsPerGenerate` to 500, 1500, 2500 to simulate different severities of congestion, which are respectively referred to as *mild*, *moderate*, and *severe*. We also measure the cases without PABO under the same traffic conditions and experimental results show a packet drop rate of $0.13\%$, $44.46\%$, $53.34\%$ at S7, respectively. Note that *the retransmission of those lost packets by upper-layer protocols could generally result in an order of magnitude increase on packet delay due to the timeout-based fashion [6].*

When PABO is involved, packet losses can be prevented. The number of bounced packets at each switch in all the three scenarios is illustrated in Figure 5(a) and the proportion of `maxBouncedHop` is shown in Table I. As we can observe that, only switch S7 has bounced $54.09\%$ of all the packets in the mild scenario. When the extent of the congestion becomes larger, as in the moderate scenario, switches that are one hop from the receiver (i.e. H4), have bounced packets and there are in total $84.74\%$ of the packets have been bounced one hop away from the congestion point. When the congestion becomes very severe, all the switches will be activated for bouncing packets, while there are still up to $9.2\%$ of the packets being successfully transmitted without any interference.

The zero packet loss guarantee is achieved at the sacrifice of delay, as bouncing a packet would inevitably increases its `totalHop`. To measure the delay stretch brought by PABO, we collect the values for `totalHop` from all the packets in three scenarios and presented the CDFs in Figure 5(b). We can observe that almost all packets experience a delay no more than 7 hops in the mild scenario, and up to $85\%$ of the packets traverse no more than 10 hops in the moderate scenario, which are still no more than three times the delay in the normal case. This is quite acceptable compared to the orders of magnitude delay increases in retransmission-based approaches.




(a) (b)

Fig. 6. Switch output queue utilization under the mild scenario: (a) over the whole duration of the simulation, and (b) over the first traffic peak.

We also monitor the output queue utilization of switch S4, S5, S6, and S7 in the mild scenario, and the results are depicted in Figure 6. We monitor the utilization levels of `normalQueue` and `bounceQueue` separately, and calculate the average utilization of the two queues at the output. Figure 6(a) depicts the average utilization of the relevant switches over the whole duration of the simulation which includes five appearances of transient congestion. We then focus on the first traffic peak as illustrated in Figure 6(b), where we notice that when the `normalQueue` utilization of S7 becomes high, packets are bounced to upstream switches S4, S5, and S6 and thus, the `bounceQueue` at S4, S5, S6, as well as S7, will be used instead of overflowing the `normalQueue` of S7. When the traffic volume declines, the queues at S4, S5, and S6 will be firstly cleared up and then finally the congestion vanishes with the drop of the average (first bounce and then normal) queue utilization of S7. This verifies our claim that PABO can avoid packet loss and handle congestion by temporarily utilizing the buffers of upstream switches.

### C. Impact of Parameters

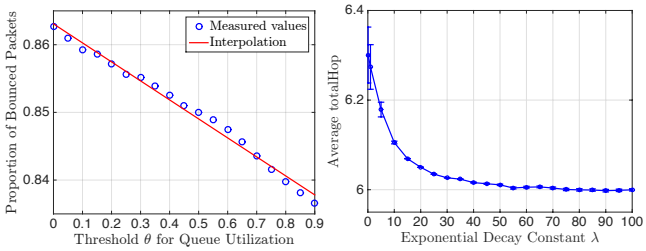
We also explore the impact of the parameters on the effectiveness of PABO in the moderate scenario. We focus mainly on two parameters in the probability function $P$ and the experimental results are shown in Figure 7. We measure the impact of threshold $\theta$ for queue utilization on the proportion of packets being bounced and the exponential decay constant $\lambda$ on the average number of total hops for all packets, respectively. When setting $\lambda$ to a fixed value 2, we notice a clear trend that the proportion of bounced packets decreases linearly with the increase of $\theta$, as depicted in Figure 7(a). Similarly, we fix $\theta$ to be 0.6 and observe that the average number of total hops, i.e., `totalHop`, decreases dramatically with the increase of $\lambda$ from 0 to 30. Thereafter, it remains stable with only a negligible variation.

The values for the parameters should be determined according to the needs of the network operator. The general principle is: larger values for both $\theta$ and $\lambda$ would result in better absorption of congestion in the network, while smaller values for both would indicate a faster signaling of congestion to the sender.

## VI. Conclusion and Future Work

In this paper, we proposed a reliable data transmission protocol – PABO, for the link layer. When facing buffer saturation, PABO bounces the excess packets to upstream switches to avoid packet loss. On the one hand, PABO can mitigate transient congestion in network at a per-flow granularity, while on the other hand, it can back propagate the congestion gradually toward the sender as a signal. A proof-of-concept implementation and extensive simulations proved the effectiveness of PABO, showing that PABO can guarantee zero packet loss in all cases, with reasonable delay introduced.

While PABO is still at the early stage of its development, we have already witnessed its high potential for network congestion control and congestion mitigation. As to future work, we will explore different packet scheduling algorithms at the output queue, incorporating rate control mechanisms in the upper-layer to achieve complete congestion handling, and finally evaluating PABO's performance in more complicated network environments.




(a) (b)

Fig. 7. Impact of the parameters (a) $\theta$ and (b) $\lambda$ on the proportion of bounced packets and the average end-to-end delay measured by `totalHop`, respectively.

## References


[1] M. Alizadeh, A. G. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data center TCP (DCTCP)," in *SIGCOMM*, 2010, pp. 63–74.

[2] H. Wu, Z. Feng, C. Guo, and Y. Zhang, "ICTCP: incast congestion control for TCP in data center networks," in *CoNEXT*, 2010, p. 13.

[3] K. Zarifis, R. Miao, M. Calder, E. Katz-Bassett, M. Yu, and J. Padhye, "DIBS: just-in-time congestion mitigation for data centers," in *EuroSys*, 2014, pp. 6:1–6:14.

[4] IEEE Std 802.3x-1997, pp. 1–324, 1997.

[5] O. Feuser and A. Wenzel, "On the effects of the IEEE 802.3x flow control in full-duplex ethernet lans," in *LCN*, 1999, pp. 160–161.

[6] Y. Chen, R. Griffith, J. Liu, R. H. Katz, and A. D. Joseph, "Understanding TCP incast throughput collapse in datacenter networks," in *WREN*, 2009, pp. 73–82.

[7] S. Kandula and R. Mahajan, "Sampling biases in network path measurements and what to do about it," in *IMC*, 2009, pp. 156–169.

[8] S. Kandula, J. Padhye, and P. Bahl, "Flyways to de-congest data center networks," in *HotNets*, 2009.

[9] A. Kesselman and A. Rosén, "Scheduling policies for CIOQ switches," *J. Algorithms*, vol. 60, no. 1, pp. 60–83, 2006.

[10] OMNeT++, https://inet.omnetpp.org/.

[11] PABO, https://github.com/Xiang-Shi/PacketBounce.

[12] Explicit Congestion Notification, https://tools.ietf.org/rfc/rfc3168.txt.

[13] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Trans. Netw.*, vol. 1, no. 4, pp. 397–413, 1993.

[14] C. V. Hollot, V. Misra, D. F. Towsley, and W. Gong, "On designing improved controllers for AQM routers supporting TCP flows," in *INFOCOM*, 2001, pp. 1726–1734.

[15] D. Zats, A. P. Iyer, G. Ananthanarayanan, R. Agarwal, R. H. Katz, I. Stoica, and A. Vahdat, "Fastlane: making short flows shorter with agile drop notification," in *SOCC*, 2015, pp. 84–96.

[16] D. Zats, T. Das, P. Mohan, D. Borthakur, and R. H. Katz, "Detail: reducing the flow completion time tail in datacenter networks," in *SIGCOMM*, 2012, pp. 139–150.