PABO: A Link-Layer Congestion Mitigation Mechanism Based on Packet Bounce

Xiang Shi, Lin Wang, *Member, IEEE*, Fa Zhang, Kai Zheng, *Senior Member, IEEE*, Max Mühlhäuser, and Zhiyong Liu

Abstract

In today's data center, a diverse mix of throughput-sensitive long flows and delay-sensitive short flows are commonly presented. However, commodity switches used in a typical data center network are usually shallow-buffered for the sake of reducing queueing delay and deployment cost. The direct outcome is that the queue occupation by long flows could potentially block the transmission of delay-sensitive short flows, leading to degraded performance. Congestion can also be caused by the synchronization of multiple TCP connections for short flows, as typically seen in the partition/aggregate traffic pattern. The congestion is usually transient and any end-device intervention through the timeout-based pathway would result in suboptimal performance. While multiple end-to-end transport-layer 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 goals: 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 present the detailed design of PABO and complete a proof-of-concept implementation. We discuss the impact of system parameters on packet out-of-order delivery and conduct extensive experiments to prove the effectiveness of PABO. We examine the basic properties of PABO using a tree-based topology, and further evaluate the overall performance of PABO using a realistic fattree topology for data center networks. Experiment results show that PABO can provide prominent advantage of mitigating transient congestions and can achieve significant gain on end-to-end delay.

I. Introduction

Nowadays, organizations like large companies or universities built various sizes of data centers for different purpose today. To interconnect all of the data center resources together, a dedicated network (a.k.a Data Center Network, DCN) is used to support high-speed communications between servers with high availability in data centers. For reasons of queueing delay and deployment cost, DCN are usually composed of shallow-bufferd commodity switches at low

A preliminary version of the paper has appeared at IEEE ICC 2017 [1]. 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).

X. Shi, F. Zhang, and Zhiyong Liu are with the State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China. This work was done while X. Shi was visiting Technische Universität Darmstadt.

L. Wang and M. Mühlhäuser are with the Telecooperation Lab at Technische Universität Darmstadt, Germany.

K. Zheng is with Huawei Technologies, Beijing, China.

costs. However, data center applications such as web search, recommendation systems and online social networks can generate a diverse mix of short and long flows, demanding high utilization for long flows, low latency for short flows, and high burst tolerance [2]. Long flows lead to queue buildup in switches, which reduces the amount of buffer space available for delay-sensitive short flows and burst traffic, leading to frequent packet losses and retransmissions. Meanwhile, data center applications such as real-time applications and data intensive applications produce traffic that follows the partition/aggregate pattern, overflowing the bottleneck switch buffer in a short period of time. The final performance of the partition/aggregate pattern is determined by the slowest TCP connection that suffers timeout due to packet losses. Therefore, for short flows and bursty traffic which are delay-sensitive, even a few lost packets can trigger window reduction and retransmission timeout, causing crucial performance degradation and high application latencies.

It has been demonstrated by [2] that the greedy fashion of the traditional TCP and its variants fail to satisfy the performance requirements of these flows and thus, various TCP-like protocols such as DCTCP [2] and ICTCP [3] have been proposed dedicatedly for DCN environments. However, none of the proposals can guarantee one hundred percent prevention of packet losses or timeouts [4] – 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. If packet processing in a link is slower than packet arrival, excessive packets competing for the same output port of the switch will lead to queue buildup. Continuous queue buildup will overflow the output buffer, then the subsequently arrived packets will be dropped. Neither the source nor the destination will be explicitly notified about this congestion and thus will have no knowledge of when and where packet losses have happened. The dropped packets will be retransmitted 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 [5] 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) [6] 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 [4].

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 back 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 and burst flows as in the typical incast scenario [7] 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.

Our contributions can be summarized in five 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 complete a proof-of-concept implementation of PABO in OMNeT++ [8]. *iv*) We investigate into the impact of PABO on the level of packet out-of-order, based on which we provide some insights for configuring PABO. *v*) We carry out extensive experiments to validate the basic properties of PABO using a tree topology and to evaluate PABO's overall performance using a realistic fattree topology.

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. Section V analyzes the relationship between PABO and the level of packet out-of-order. Section VI and VII present the experimental results. Section VIII concludes the paper.

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 [2]. ICTCP [3] aims at preventing incast congestion through adjusting the advertised windows sizes at the receiver side by estimating the available bandwidth and RTT. DCTCP [2], another TCP variant developed for data center environment, take advantage of the Explicit Congestion Notification (ECN) feature [9] 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 [10] and PI [11]). ECN (Explicit Congestion Notification) [9] 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 [12] 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 [5] 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) [6] 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 [13].

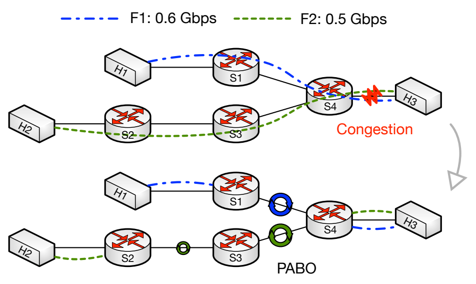


Fig. 1. A motivating example to demonstrate the idea of PABO, i.e., solving network congestion without dropping packets. We assume GigE links in the network.

DIBS [4] 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., incast congestion) are commonly presented [14], [15]. 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 for 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.

B. Overview

We assume a general Combined Input and Output Queue (CIOQ) switch model [16] 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

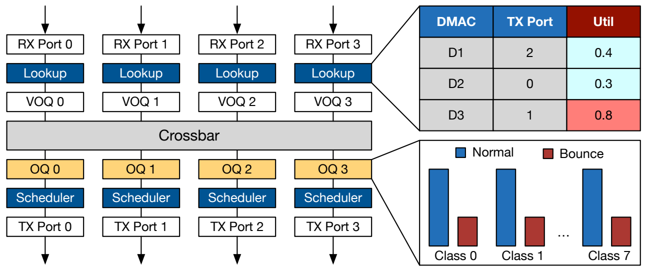


Fig. 2. An overview of PABO's design based on a Combined Input and Output Queue (CIOQ) switch model.

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 at the early stage of bouncing back to prevent the queue from fast buildup;

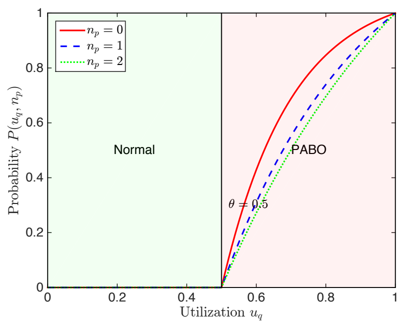
PRCP-3: Packets that have already been returned should receive smaller probability to be bounced back again.

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

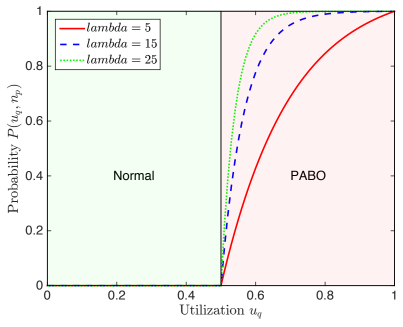
$$P_b(n_p) = e^{\frac{\lambda}{n_p+1}} \tag{1}$$

where $\lambda > 0$ is the exponential decay constant. It is always true that $P_b(n_p) > 1$ 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 output queue utilization is under the predefined threshold, i.e., the queue is underutilized, there



(a)



(b)

Fig. 3. Probability function $P(u_q, n_p)$ for packet bounce decision making. We set the threshold θ to 0.5: (a) we set the constant λ to 5, and we show the curves in cases of $n_p = 0, 1, 2$, respectively, and (b) we show the curves with $lambda = 5, 15, 25$ in the case of $n_p = 0$. When $u_q \leq 0.5$, switches forward packets normally; PABO is only involved when $u_q > 0.5$.

is no need to bounce packets; the second equation guarantees that when the output queue is full or overflows, all the upcoming packets have to be bounced back in order to avoid packet drops. These two equations ensure the validity of PRCP-1. Then we define for $\theta < u_q < 1$,

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

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

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

Substituting P_b with Equation (1) and combining all the above cases, we have the closed form of P satisfying PRCP-2 as

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

The curves of function P, with $\theta = 0.5$, under different utilization u_q are illustrated in Figure 3. Note that θ has a major impact on the proportion of packets that will be bounced back at a given queue, while λ can be used to roughly control the maximum number of hops each bounced packet will traverse. We will further verify these correlations in Section VI.

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 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 many times the packet has been bounced back. 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 back to S_3 by S_4, the value of n_p will be increased by one. When the bounced packets are forwarded out normally to S_4, the value of n_p will stay the same. If these packets are bounced from S_4 to S_3 again, n_p will increase to record the new bounce behaviors. This counter n_p will be used by the lookup unit in switches as an input for the bounce probability calculation. By carefully setting the probability functions as we have already discussed, the probability that a packet is bounced back consistently for multiple times will be significantly reduced.

G. End-host Support

When persistent congestion occurs, there will be packets bounced straight back to the source along the reverse direction of the forwarding path. 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 severity of the congestion, measured by for example 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 injected 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++ [8]. Our implementation code is open-sourced at [17]. By overriding the corresponding link-layer modules, we created Ethernet switches 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`. The `normalQueue` and `bounceQueue` are with the `DropTailQueue` type. 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 number of times the packet has been bounced back, we add a non-negative integer counter `bouncedHop` in the header. This counter will increase by one every time the packet is bounced back by one hop and will stay the same if the packet is forwarded normally. Then we introduce three other parameters for further analysis. We introduce another counter `bouncedDistance` for each packet 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. This counter will increase by one if the packet is bounced and will decrease by one if the packet is normally forwarded. For each packet, to record the farthest distance it has been bounced, we introduce the parameter `maxBouncedDistance`. Meanwhile, 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.

Host. The host is `StandardHost` type – an example host contains modules related to link layer, network layer, transport layer and application layer. We mainly make modifications to the link layer of the `StandardHost` to generate our host 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 the bounced packets will finally reach the sender. To handle this situation, we modify the `EtherMAC` module in the `StandardHost` to avoid dropping bounced packets that are not destined for this host. Then 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 retransimission. Note that the type of sender's buffer is also `EtherQosQueue`. As a result, the same modifications we made for switches can also be applied for `EtherHost`.

In addition to the PABO implementation, we make some special modifications to the transport layer of the `StandardHost` for the purpose of measuring the level of packet out-of-order. In the TCP sender, we record the sending order of each packet into a special queue called `sentSeqQueue`. We maintain a send counter in the TCP sender. Every time a new packet is sent, the packet is labeled with a unique sending order s_i, and this mapping information (packet i, s_i) will be recorded into `sentSeqQueue` until the packet i is ACKed. If packet i is retransmitted, the sender will look up in `sentSeqQueue` to find the corresponding s_i, and label the retransmitted packet with it. In the TCP receiver, we maintain a receive counter. Each received packet is assigned a receiving order r_i (loss and duplicate packets are ignored). For each packet, we calculate the difference between its s_i and r_i. Finally, we can get a distribution of displacement of packets, which will be used for further analysis.

V. Packet Out-of-Order Analysis

Packets arriving at receiver disobeying their sending order is called packet out-of-order delivery [18]. TCP requires the in-order of packets, which means the receiving order of the packets is supposed to be the same as their sending order. Out-of-order packets will generate sequence holes on the receiver side and then, TCP will duplicate the ACK to request the missing packets. Three continuous duplicate ACKs will trigger spurious fast retransmission [19], bringing unnecessary packet delay as well as reduced congestion window. Studies have shown that packet out-of-order delivery is not rare [20], [21] and it can be caused by multiple factors such as packet loss, parallelism within routers or switches, different path delays in packet-level multi-path routing, route fluttering and so on [18]. In our case, PABO allows packets to be retuned on the forwarding path in order to prevent packet loss. On the other hand, this bouncing behavior will disturb the packet forwarding direction, thus inevitably leading to increased number of out-of-order packets. In this section, we will dissect the impact of each of the system parameters (e.g., θ and λ in the bounce probability function (4)) to packet out-of-order. From the analysis, we show how the parameters could be tuned in order to balance the efficiency of PABO and its negative effect on packet out-of-order.

A. Measuring Packet Out-of-Order

To measure the level of packet out-of-order delivery in a packet stream, various methods have been proposed. Among them, Reorder Density (RD) captures packet out-of-order delivery by a weighted distribution of the displacement of packets [22]. Consider a sequence of packets sent in the order of $[1, 2, ..., N]$, which is referred to as sending index s_i for packet i. When arriving at the receiver side, each packet i will be assigned a receiving index recording the order of reception, denoted by r_i. The difference between the sending index and the receiving index, denoted by d_i for each packet i, is calculated as

$$d_i = r_i - s_i. \tag{5}$$

If $d_i > 0$, packet i is considered to be late; $d_i < 0$ means that packet i arrives earlier than expected; $d_i = 0$ means there is no out-of-order event occurred. RD also introduces a threshold $D_T > 0$ on $|d_i|$, beyond which an early or a late packet is deemed lost. Lost or duplicated packets will not be assigned any receive index. Then, we define distribution vector $S[k]$ which contains the number of packets with a displacement of k. By normalizing it to the total number of non-duplicated received packets N, we obtain the following weighted distribution of displacements, denoted by $RD[k]$.

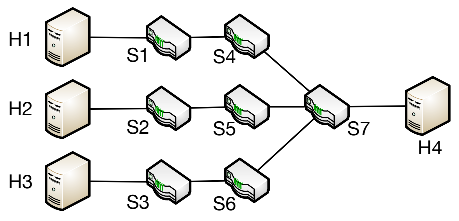


Fig. 4. The network topology used for investigating the impact of PABO on the level of packet out-of-order, and also for conducting the hop-by-hop evaluation of PABO.

$$RD[k] = \frac{S[k]}{N}, -D_T \leq k \leq D_T. \tag{6}$$

Based on RD, we now derive a new metric called *reorder entropy* to quantitatively analyze the properties of the distribution [23]. Reorder entropy uses a single value to characterize the level of out-of-order in a packet stream, reflecting the fraction of packets displaced and the severity of packet displacement. The formal definition of reorder entropy is given by

$$E_R = (-1) \cdot \sum_{i=-D_T}^{i=D_T} (RD[i] \cdot \ln RD[i]). \tag{7}$$

It can be verified that larger reordering entropies represent a more dispersed distribution of packet displacement, translating into a more severe packet out-of-order event. If there is no packet out-of-order at all, the reorder entropy should be equal to zero.

B. Impact of PABO on Reorder Entropy

To explore the impact of PABO on packet out-of-order, we conduct some experiments to investigate how the reorder entropy changes with different PABO parameters. In particular, we tune the values for parameters θ and λ in the bounce probability function as in Equation (4) and we report our major observations, based on which we discuss possible ways to improve PABO in terms of packet out-of-order.

Simulation Setup. We adopt a tree-based network topology consisting of three servers (i.e., H1, H2 and H3) and one client (i.e., H4) connected by seven switches, as depicted in Figure 4. All the links in the network are assumed to have the same rate of 1Gbps. We consider a scenario where the client establishes and maintains three concurrent TCP connections with the three servers, respectively. The topology shows that the client is three-hop away from each of the servers and the data streams from all the servers will be aggregated at the last-hop switch directly connected to the client. All the experimental results presented in this section will be combined measurement results of the three concurrent TCP connections.

Traffic. We create communication patterns in the considered scenario to simulate the expected congestion conditions. In our experiment, we set up one TCP session for each of the TCP connections during the whole simulation time. Each of the TCP session includes four TCP requests, in which the hosts behave in a request-reply style: The client sends a request (200B) with the expected reply length (1MiB) to the server, then the server responds immediately with

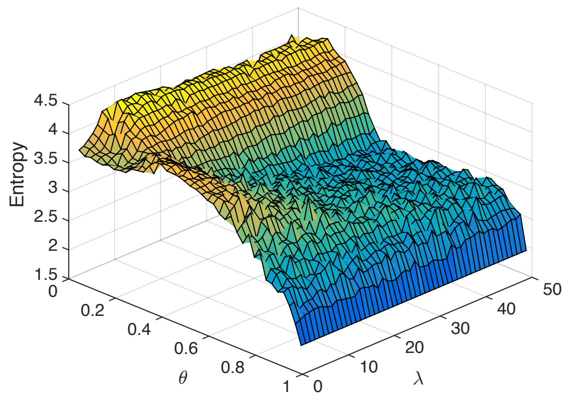


Fig. 5. The relationship between θ, λ in bounce probability function and `entropy`.

the requested length of data. Each TCP request represent an appearance of transient congestion at switch S7 due to the fact that all the three servers will send data to the client in a synchronized fashion. This setting simply emulates the partition/aggregate traffic pattern that is very popular in a data center network and can be conveniently monitored. In the rest of the paper, we will refer to it as the partition/aggregate traffic for ease of expression. The time gap between TCP requests is set to one second, which is enough to avoid overlaps between the periodic congestion appearances. To simulate a transient congestion (e.g., incast congestion), we set buffer sizes to be small and flow rates to be bursty. More specifically, the queue capacities of both `dataQueue` and `bouncedQueue` in all the switches and hosts are set to 100. To remove the limit on flow rate by the congestion control mechanism in TCP, we set `ssthresh` to be arbitrarily high so that the servers will perform slow start without the limit of `ssthresh`. At the same time, the advertised window of the client is set to 45535 bytes to enable the growth of flow rate. We also disable fast retransmission in TCP and set the retransmission timeout (RTO) to its upper bound 240s [24]. This way, the packet out-of-order caused by packet retransmission is eliminated so we can observe the packet out-of-order brought only by the bouncing behavior of PABO.

Overview of Packet Out-of-Order Delivery. We first present a brief overview of packet out-of-order delivery under different parameter settings. We use reorder entropy (referred to as entropy hereafter for the ease of expression) to quantify the level of packet out-of-order. By tuning the values for the parameters, namely θ and λ, we make observations on how the entropy changes accordingly. The experimental results are depicted in Figure 5. As we can observe that, the entropy declines along the θ axis, e.g., when θ increases, the threshold for PABO to bounce back packets becomes higher and thus, less packets will be bounced under a certain traffic condition, as a result of which packet out-of-order is less severe. Along the λ axis, the variation of entropy is less significant. Note that the entropy converges to a minimal point in cases that the switch will only bounce back packets when the buffer is full.

Separate Impact of Parameters θ and λ. We now focus on analyzing how θ and λ affect on packet out-of-order separately. In addition to the reorder entropy which measures the level of packet out-of-order, we calculate the *variance* of the buffer utilization of all the switches in the network. Note that the variance can be used to roughly characterize the effectiveness of PABO as PABO utilizes the buffer of upstream switches to avoid packet loss, leading to a more even distribution of packets among the switch buffers in the network. Meanwhile, we define *timeRatio* to measure the scope of affected packets under different pairs of θ and λ. For those utilization of the buffers that will be considered during the probabilistic decision making process, we calculate the average time ratio of them over θ.

Figure 6 depicts the separate impact of θ and λ on the reorder entropy, the variance, and the timeRatio. We select three representative values for θ and λ respectively to conduct further analysis. It can be generated observed that the reorder entropy in all the six figures is highly correlated to timeRatio, which confirms that PABO can affect the level of packet out-of-order.

Figure 6 (a)-(c) illustrate the impact of parameter θ on packet out-of-order. According to Equation (4), larger θ means the less effectiveness of PABO. We can observe in the three figures that timeRatio reflecting the affected scope of PABO decreases gradually to a steady point. This is because most switches are under low utilization, thus the change of θ in a lower range can affect more switches. As PABO is gradually losing its influence, we can observe that the variance in the three figures grows with similar tendency. However, the decrease tendency is different for the reorder entropy. In Figure 6(a), the entropy changes relatively stable comparing to the other two figures. This can be explained by the change of affected scope of packets: when λ is at small value (e.g. $\lambda = 1$), the change of θ has much less impact on timeRatio. In Figure 6(b) and (c), the entropy drops sharply at first, then remains stable in the middle area, followed by a further decrease in the end. This is basically in accordance with the change of timeRatio. The final decline is reasonable, as the switches are trying to avoid packet bounces as much as possible.

Figure 6 (d)-(f) depict the influence of parameter λ on packet out-of-order. It is known that larger λ means the more effectiveness of PABO. In pace with the growth of λ, timeRatio increases and the variance decreases with a reasonable range of fluctuation. Moreover, we can observe from the ordinate range of the above figures that larger θ leads to the reduced influence of λ on timeRatio. When θ is very close to 1 (e.g. in Figure 6 (f)), the influence of λ can be ignored, since it hardly affects the bounce probability. As is demonstrated in Figure 6 (d) that the `entropy` rises rapidly at first and then remains a steady point. However, Figure 6 (e) illustrates an opposite tendency of `entropy`. This is because when $\theta = 0.5$, S7 is the only switch among the topology in Figure 4 that satisfies the condition for bouncing back packets. As λ rises, the percentage of bounced packets continues to grow util it is infinitely close to 1. In this case, higher bounce back percentage at S7 leads to lower `entropy`, as the bouncing back process of the packets is comparable with experiencing an equally extended path.

C. Discussion

As stated above, the effectiveness of PABO can be a tradeoff when deciding the values of θ and λ. The main idea of PABO is to utilize the buffer of upstream switches to avoid packet loss, making the packets more evenly distributed in the network. Consequently, the transient congestion (e.g. the incast problem) is relieved. Higher influence of PABO brings the stronger ability to mitigate transient congestion in the network, which can also result in more severe packet out-of-order. Severe packet out-of-order can lead to the performance collapse of the related TCP connection. Therefore, we should balance between the variance and the reorder entropy by carefully choosing the parameters for target networks. Focusing on this tradeoff, the recommended solution is to control the bounce back scope around the congestion point. This means when there is a congestion, θ should be set to just exclude the switches which are not

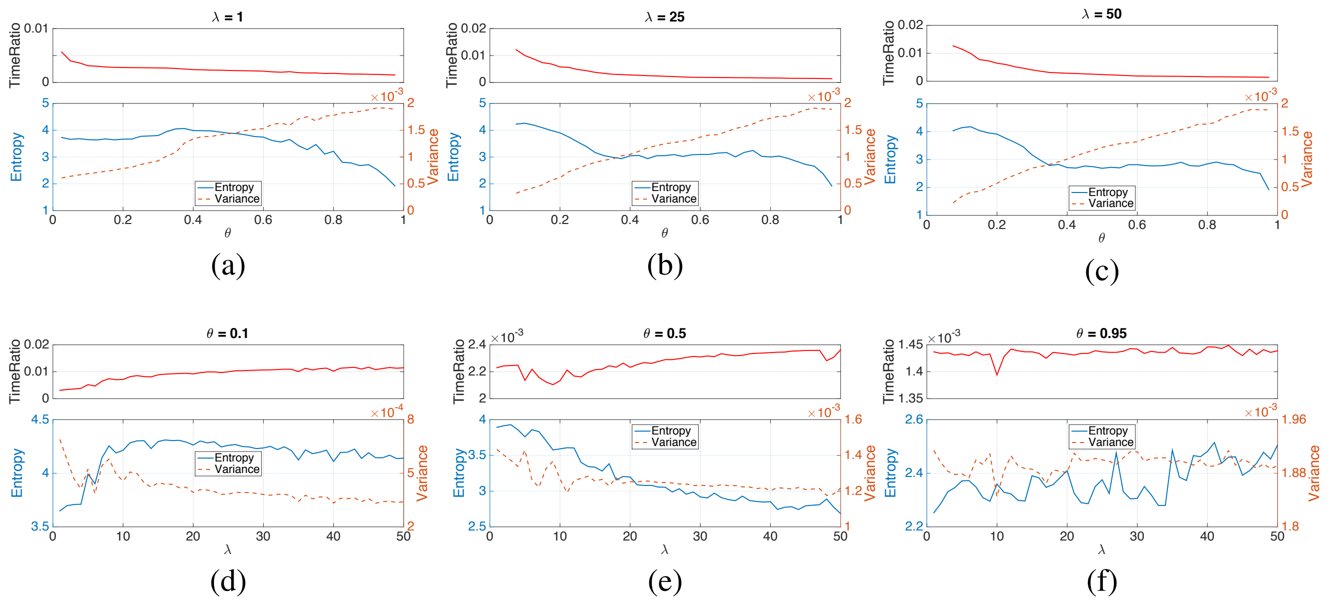


Fig. 6. Separate impact of parameter θ, λ in bounce probability function on reorder entropy, variance and timeRatio.

faced with the danger of packet loss. There is no need to set the θ too large, for the reduced reorder entropy is not worth to compensate for the increased variance. If the bounce back scope is limited to be just around the congestion point by θ, larger λ representing higher bounce back percentage is preferred to reduce both the reorder entropy and the variance.

VI. Hop-by-Hop Evaluation

We first conduct simulation studies at a hop-by-hop level to evaluate the performance of PABO, and we report the experimental results in this section.

A. Simulation Setup

We adopt the same topology as Section V in Figure 4, except that the hosts don't contain any high layer protocols (i.e., IP, TCP). We focus on only one direction data retransmission for the moment, where three senders (i.e., H1, H2 and H3) send data simultaneously to a single receiver (i.e., H4). The data from all the senders will be aggregated at the last-hop switch (i.e., S7), in which the congestion appears. The duration of all the simulations is set to one second 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 roughly calculated by `numPacketsPerGenerate` × `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

TABLE I
DISTRIBUTION OF `maxBouncedDistance`

Scenario	**# of** `maxBouncedDistance`			
	0	**1**	**2**	**3**
Mild	43.45%	56.55%	–	–
Moderate	14.73%	85.27%	–	–
Severe	8.82%	84.14%	7.04%	–

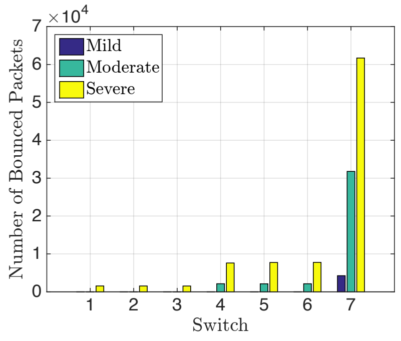
of `sendInterval` to be 10μs. Then, we control the rate by tuning `numPacketsPerGenerate` to simulate congestions in different severities. We set `pauseInterval` to be 0.2s, which is sufficient to avoid overlaps between two consecutive congestions.
Queue. The input and output queues of both hosts and switches are with the `DropTailQueue` type. When using PABO, the capacities of both `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. In the cases without PABO where `bounceQueue` are not used, we double the capacities of `normalQueue` in both switches and senders for fairness concerning.
Packets. All the packets generated are in the IEEE 802.3 frame formats with the payload size set to be 1500 bytes.

B. Effectiveness under Different Congestion Severities

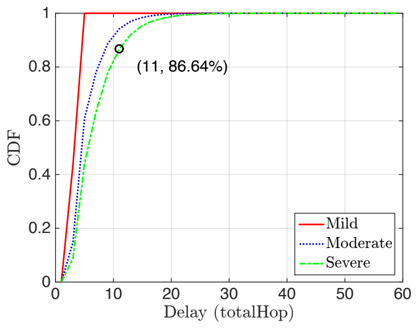
We validate the effectiveness of PABO by comparing it to the standard link-layer protocol under three different severities of congestion. Parameters of bounce probability function P are fixed as $\lambda = 50$, $\theta = 0.8$. We set `numPacketsPerGenerate` to 500, 1500, 2500 to simulate different severities 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 [7].*

When PABO is involved, packet loss can be prevented. The number of bounced packets at each switch in all the three scenarios is illustrated in Figure 7(a) and the proportion of `maxBouncedDistance` is shown in Table II. As we can observe that, only switch S7 has bounced 56.55% 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 85.27% of the packets have been bounced one hop away from its last normally reached switch. When the congestion becomes very severe, all the switches will be activated for bouncing packets, while there are still up to 8.82% 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 7(b). We can observe that almost all packets experience a delay no more than 5 hops in the mild scenario, and up to 86.64% of the packets traverse no more than 11 hops in the

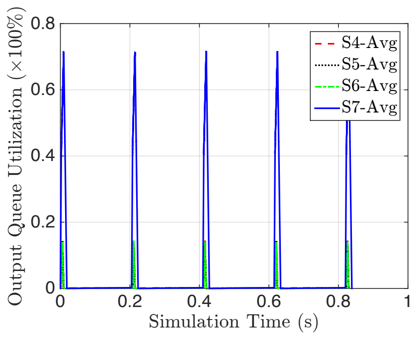


(a)

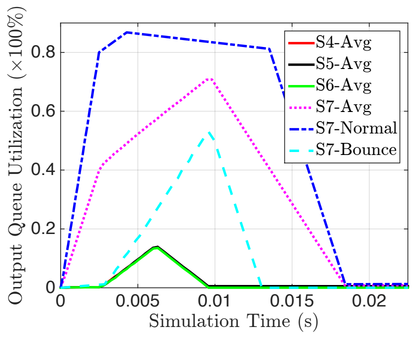


(b)

Fig. 7. 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`.



(a)



(b)

Fig. 8. Switch output queue utilization under the mild scenario: (a) over the whole duration of the simulation, and (b) over the first traffic peak.

severe scenario, which are still no more than four times the delay in the normal case. This is quite acceptable compared to the orders of magnitude delay increases in retransmission-based approaches.

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 8. We monitor the utilization levels of `normalQueue` and `bounceQueue` separately, and calculate the average utilization of the two queues at the output. Figure 8(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 8(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.

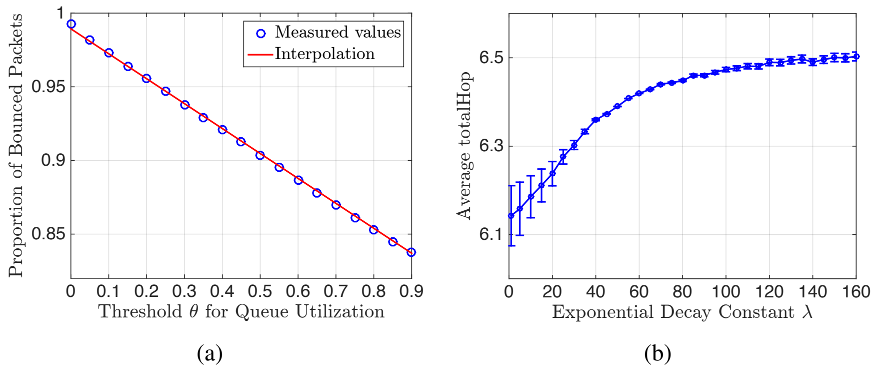


Fig. 9. Impact of the parameters (a) θ and (b) λ on the proportion of bounced packets and the average end-to-end delay measured by `totalHop`, respectively.

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 bounce probability function P and the experimental results are shown in Figure 9. We measure the impact of threshold θ for queue utilization on the proportion of packets being bounced and the exponential decay constant λ on the average number of total hops for all packets, respectively. When setting λ to a fixed value 50, we notice a clear trend that the proportion of bounced packets decreases linearly with the increase of θ, as depicted in Figure 9(a). Similarly, we fix θ to be 0.8 and observe that the average number of total hops, i.e., `totalHop`, increases gradually stable with the increase of λ from 0 to 160. 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: smaller θ and larger λ improve the effectiveness of PABO, which tends to avoid the congestion at a earlier time; while larger θ and smaller λ would prefer to reduce the sensitivity of PABO, thus delay the absorption of the congestion.

VII. End-to-End Evaluation in Realistic Topology

We further evaluate the performance of PABO in the Fattree network topology [25] and we present the experimental results in this section. We simulate different traffic patterns (many-to-one, many-to-many) to observe the performance of PABO, then we present the parameter study of θ about the impact of PABO on reorder entropy, per-packet delay and end-to-end delay.

A. Two-level Routing Table Implementation

Based on `EtherSwitch` with PABO mechanism, we created a `FattreeSwitch` implementing the two-level routing table of Fattree. Owing to the fact that `EtherSwitch` module does not contain network layer, we implement the two-level routing table on the link layer by applying the following modifications.

Addressing. Firstly, in line with the Fattree addressing scheme, we make a one-to-one mapping between an IP address and a MAC address. To illustrate, the IP address 10.0.0.1 corresponds to the MAC address 0A-AA-0A-00-00-01. All MAC addresses share the same first two octets 0A-AA, and the rest are transformed equivalently. Then we assign the transformed MAC addresses to each switch and host. Note that all the MAC ports of a switch share the same MAC address for simplicity, which will not effect any experimental results.

Structure. Secondly, we modify the structure of `MACTable` to allow entries containing prefixes and suffixes (i.e., /m prefixes are the masks used for left-handed matching, /m suffixes are the masks used for right-handed matching).

Lookup. Thirdly, we modify the lookup unit of `MACTable` to allow two-level route lookup. Prefixes are basically intended for route matching of intra-pod traffic, while suffixes for inter-pod traffic. The value of prefix or suffix is simply used to check the number of octets required for comparison. To be more specific, if we want to match an entry in the `MACTable` with a left-handed prefix of N (e.g. 24), we should find from left to right at least N/8 (e.g. 3) identical octets between this entry and the destination MAC address, excluding the same first two octets. This also applies to the match of a right-handed suffix except that the matching direction is from right to left.

Routing Example. Here we explain the Fattree two-level routing algorithm implemented in Figure 10. For the hosts connected to each lower-level switch (e.g. S1) in this Fattree structure, the last octets of the left hosts are 02, and the last octets of the right hosts are 03. Based on the last octet of the destination MAC address, the algorithm uses the prefix/suffix matching to disperse different traffic, which we will explain in a simplified way. In the fattree topology, each pod follows similar rules on packet routing. We take pod 0 as an example to explain inter-pod routing and intra-pod routing separately. For ease of expression, we refer to S1, S3 as the left-side switches, and S2, S4 as the right-side switches. For inter-pod traffic, the left-side switches route a packet destined for another pod to the port number same as the last octet of its destination MAC address (e.g. packet addressed to 02 forwarded to port 2 and 03 forwarded to port 3). The right-side switches work in an opposite way (e.g. packet addressed to 02 forwarded to port 3 and 03 forwarded to port 2). We give an example to explain the route decisions taken for a packet from the inter-pod traffic: source H2 to destination H10, which is illustrated in Figure 10. Marked in the picture are the port numbers of the switches. As the last octet of its destination is 03, the packet first take the port 3 of S1, then goes out at the port 2 of S4 to C3, after which there is only one path to take: to be transmitted to the destination pod, then the destination subnet switch where it is finally switched to its destination host. For intra-pod traffic, the first-hop switch follows the same rules as in inter-pod routing, then the second-hop switch route the packet to its destination subnet switch, and finally the destination host.

B. Simulation Setup

We use the Fattree network topology depicted in Figure 10 to evaluate the performance of PABO. We initialize the `MACTable` of all the `FattreeSwitch` using input files that give all the prefixes and suffixes, and turn off the update function. Meanwhile, to avoid the broadcast storm brought by ARP request of the hosts, we statically initialize all the hosts with the IP-address-to-MAC-address mapping information. The duration of all the simulations is set to ten second, which can cover multiple appearances of periodic congestions. The links in the network are assumed to have the same rate of 1Gbps.

Traffic. We simulate different traffic patterns by changing the number of servers and clients. For each TCP connection, we use the same traffic and buffer setup as Section V, except that the advertised window is set to 50000 bytes. When using PABO, we disable all the retransmissions (both fast retransmit and retransmission timeout) as well as skipping the related window reduction

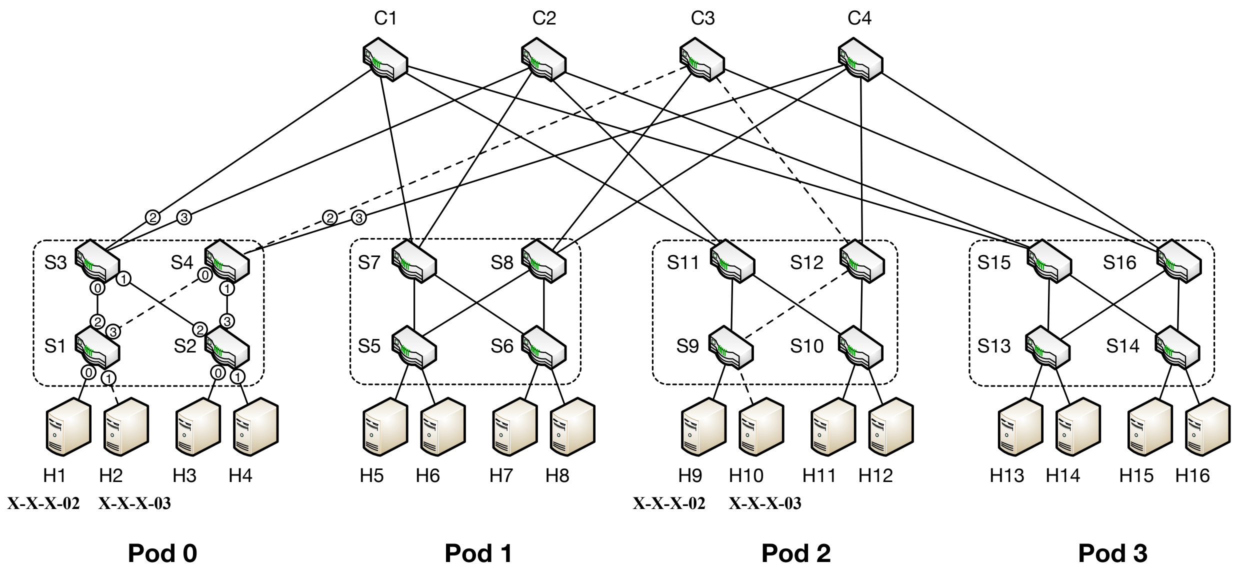


Fig. 10. The fattree topology used for evaluating the performance of PABO.

TABLE II
DISTRIBUTION OF `maxBouncedDistance`

Scenario	# of `maxBouncedDistance`		
	0	1	2
3 to 1	50.17%	49.83%	–
6 to 1	53.24%	46.76%	–
9 to 1	56.71%	43.29%	–
12 to 1	57.68%	42.32%	–
m to m	38.37%	61.63%	–

intended for congestion control, as the retransmissions are unnecessary due to the reliability of PABO. In the cases without PABO, we adopt the TCP Reno protocol to provide network congestion control.

PABO Configuration. We set the system parameters $\theta = 0.95$, $\lambda = 50$ in both many-to-one and many-to-many scenario experiment.

C. Many-to-One Scenario

First, we mimic the partition/aggregate traffic by specifying multiple servers responding to one client (i.e., H9) in a synchronized fashion. We simulate different severities of congestion by changing the number of servers (i.e., 3 to 1, 6 to 1, 9 to 1, 12 to 1). And we observe how PABO performs in different congestions comparing to cases without PABO.

Figure 11 demonstrates the packet bounce frequency of each switch under different many-to-one scenario. In the 3 to 1 congestion scenario, we choose one host from each pod to be the servers except the pod with the client H9, i.e., H1, H5, H13. The first aggregate switch of the three connections is C1. When PABO is not working, the core switch C1 will be the only switch

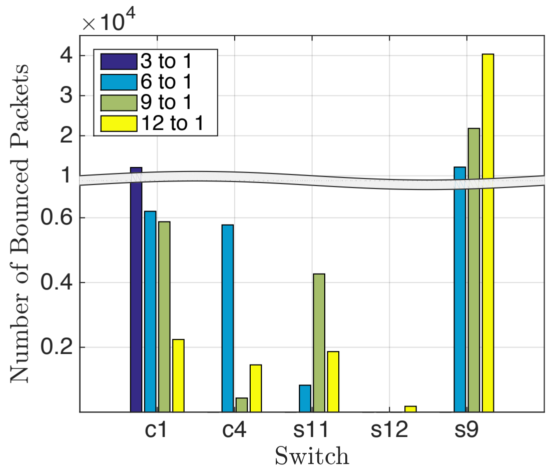


Fig. 11. Packet bounce frequency under different many-to-one scenario.

to experience packet losses with an overall drop rate of 0.33%. As PABO participates, only C1 bounce back packets to avoid packet loss at a bounce percentage of 49.83%.

In the 6 to 1 congestion scenario, we add one server from each pod except pod 2, i.e., H3, H7, H15, then S9 is congested with a drop rate of 0.51%. There is no significant increase in drop rate and the number of drop location, as the congestion control mechanism of TCP Reno is taking effect. For PABO, we can see that as the first aggregate switches each gathering traffic flows of three connections, C1 and C4 bounce back packets to mitigate congestion. Since most of the congestion resulted from aggregating three connections is mitigate by the bouncing back of C1, C4. There is only a small scale of bounce back in S11 and no bounce back in S12. The highest proportion of bouncing back is in S9 as it is the last hop that gathers all the traffic flows to H9.

In the 9 to 1 congestion scenario,we further add a server from each pod except pod 2, i.e., H2, H8 and H14. In this scenario, pod switch S9 is congested with a drop rate of 0.45% without PABO. When PABO is involved, as aggregate switches of five traffic flows from H1, H2, H5, H13 and H14, C1 and S11 bounce back packets. For traffic flows from H3, H7, H8 and H15, the aggregate switch C4 bounces back packets. As the only path to H9, S9 still account for the highest percentage of bouncing back. In total, 43.29% of all the packets are bounced back.

In the 12 to 1 congestion scenario, all of the hosts in the first two pod and three out of four hosts (i.e., H13, H14, H15) in the last pod are the servers, together with a special intra-pod traffic brought by H11. Without PABO, both pod switch S9 and core switch C1 are congested with a total drop rate of 0.52%, while 32.6% of the drop event happen in C1 and 67.4% happen in S9. With PABO taking effect, C1, C4, S11, S12 and S9 bounce back 42.32% of the packets.

For all the many-to-one scenarios, Table II shows that as the number of servers increases, the percentage of bounced packet decreases although the congestion is getting more severe. This is reasonable because though the percentage of bounced packets is smaller, each bounced packet is bounced back and forth more frequently around the congestion point.

Figure 12 shows the CDF of absolute packet displacement measured by RD in the 12 to 1 scenario. In cases with PABO, because of the severe congestion, approximately 50% of the packets arrive at their destination out of order. The max absolute value of displacement is 95.

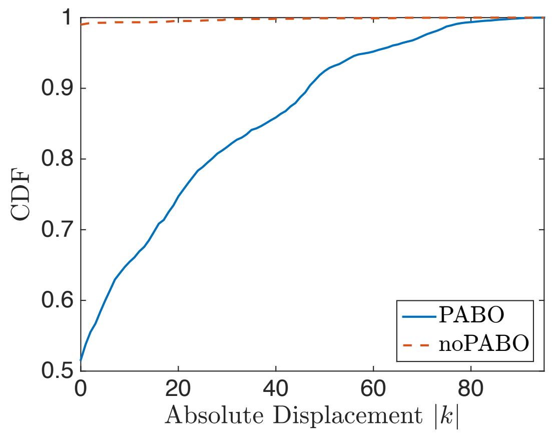


Fig. 12. The CDF of absolute packet displacement in the 12 to 1 scenario.

In cases without PABO, packet loss is the main reason of packet out-of-order. Although the level of out-of-order is much less severe, the max absolute value of displacement is up to 76. This means there are still packets with large displacement values which are enough to trigger retransmission timeout.

We also focus on the time delay comparison between cases with PABO and cases without PABO. Figure 13 illustrates the average time delay per packet, regarding the time spent of every received packet from its source to destination. It shows that the average per-packet delay of PABO is slightly higher than cases without PABO, which is because some of the packets are bounced back to avoid packet loss. Moreover, with the growth of the server number in the many-to-one scenario, the standard deviation of per-packet delay increases as the congestion is becoming more severe. This confirm our statement that smaller percentage of bounced packets in more severe congestion means that bounced packets are bounced back and forth more frequently. For all the many-to-one despite the 12 to 1 scenario, the basic trend for per-packet delay is very related to the degree of congestion. The exception is because the path of the special intra-pod traffic existed only in the 12 to 1 scenario is much shorter than the others.

Figure 14 depicts the average end-to-end delay of all TCP connections in each scenario, from which we can see that cases with PABO has obvious advantage over cases without PABO in all the scenarios, for retransmission-based approaches of TCP Reno bring orders of magnitude delay increases. Moreover, the advantage of PABO on end-to-end delay in the many-to-one scenario grows increasingly evident with the congestion becoming more severe (excluding the 12 to 1 scenario for the influence of intra-pod traffic).

D. Many-to-Many Scenario

Second, we evaluate PABO under the many-to-many scenario, which consists of two 3 to 1 partition/aggregate traffic. In this scenario, H1, H5 and H13 are the servers which respond to the request of client H9. In the meantime, H4, H8 and H16 respond to the request of client H10. For each 3 to 1 traffic, the first aggregate switches are the core switch C1 and C2 respectively. Then the two 3 to 1 traffic aggregate at S11 and go separately at the output ports of S9. Without PABO, the drop rate is 0.54%, with all the packet losses occur in S11. By using PABO, 61.63% of the packets are bounced back, with 18.35% bounced at C1 , 18.77% at C2 and 62.88% at S11.

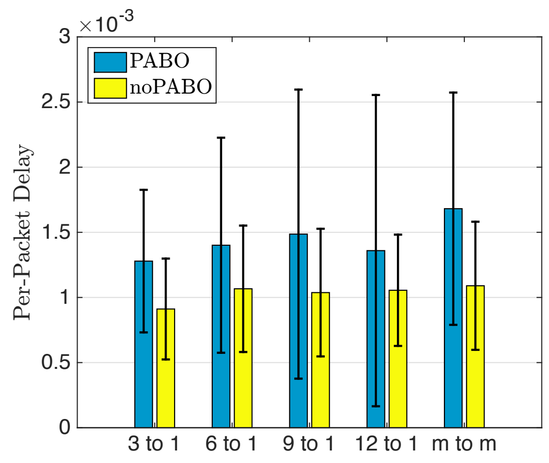


Fig. 13. Average time delay per packet under different congestion scenario.

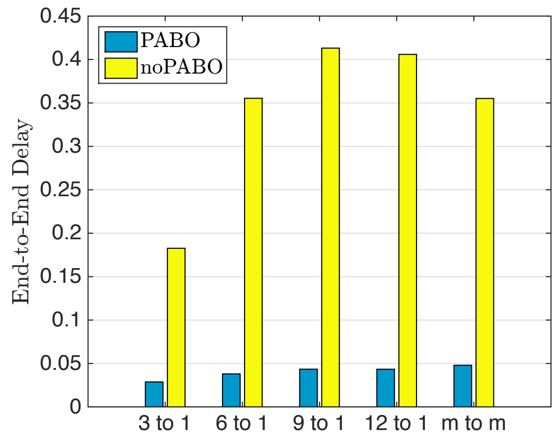


Fig. 14. Average end-to-end delay of all TCP connections in each congestion scenario.

In the many-to-many scenario, the per-packet delay of PABO is still higher than cases without PABO, for over half of all the packets are bounced back. As for the end-to-end delay results, PABO still has great advantage over the normal case.

E. Impact of Parameter θ

We make evaluations to observe how the parameter θ in the bounce probability function of PABO affects on the experiment results under the many to many scenario. We neglect the small values of θ, because it's unnecessary to bounce back too early. Therefore, we only focus on the domain of $\theta \geq 0.5$. The results are illustrated in Figure 15. It shows that with the increase of θ, `entropy` remains stable then drops when θ is close to 1, which is similar to the previous result in Section V. As to the effect of θ on the end-to-end delay, it shows no obvious regularity. For sooner or later PABO starts to bounce back packets, the bounced packets are absorbed inside the network to queue up for being finally handled by the destination end host. Therefore when using PABO, the arrival time of the last packet is determined by the limit of the last hop switch connected to the destination, rather than the bouncing back threshold θ. For results of per-packet

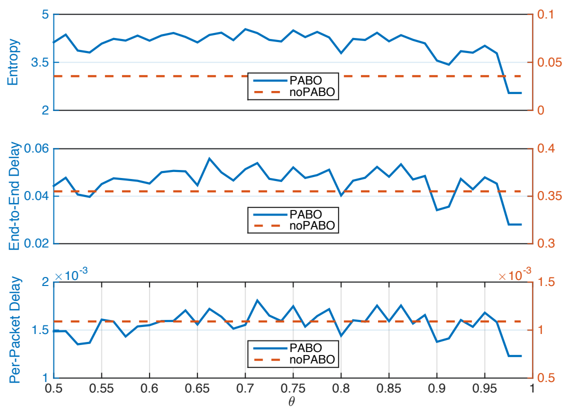


Fig. 15. The relationship between θ, `entropy`, per-packet delay and end-to-end delay in the many-to-many scenario.

delay, there is also no obvious regularity. Smaller values of θ avoid congestion at an earlier time, which result in larger percentage of bounced packets and relatively smaller bounced frequency for each packet. Similarly, larger θ values tend to avoid packet loss as well as maintaining high utilization of switches, thus the percentage of bounced packets is smaller and the bounced packets are bounced back and forth more frequently.

VIII. Conclusion

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, which can mitigate transient congestion in network at a per-flow granularity. We complete a proof-of-concept implementation, and investigate into the impact of PABO on the level of packet out-of-order, then we provide useful insights for configuring PABO. Extensive simulations have proved the effectiveness of PABO, showing that PABO has obvious superiority of end-to-end delay over the traditional protocol stack by guaranteeing zero packet loss in all cases.

References

[1] X. Shi, L. Wang, F. Zhang, K. Zheng, and Z. Liu, "Pabo: Congestion mitigation via packet bounce," in *Communications (ICC), 2017 IEEE International Conference on Communications*. IEEE, 2017, pp. 1–6.
[2] 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.
[3] H. Wu, Z. Feng, C. Guo, and Y. Zhang, "ICTCP: incast congestion control for TCP in data center networks," in *CoNEXT*, 2010, p. 13.
[4] 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.
[5] IEEE Std 802.3x-1997, pp. 1–324, 1997.
[6] 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.
[7] 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.
[8] OMNeT++, https://inet.omnetpp.org/.
[9] Explicit Congestion Notification, https://tools.ietf.org/rfc/rfc3168.txt.
[10] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Trans. Netw.*, vol. 1, no. 4, pp. 397–413, 1993.

[11] 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.
[12] 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.
[13] 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.
[14] S. Kandula and R. Mahajan, “Sampling biases in network path measurements and what to do about it,” in *IMC*, 2009, pp. 156–169.
[15] S. Kandula, J. Padhye, and P. Bahl, “Flyways to de-congest data center networks,” in *HotNets*, 2009.
[16] A. Kesselman and A. Rosén, “Scheduling policies for CIOQ switches,” *J. Algorithms*, vol. 60, no. 1, pp. 60–83, 2006.
[17] PABO, https://github.com/Xiang-Shi/PacketBounce.
[18] K.-C. Leung, V. O. Li, and D. Yang, “An overview of packet reordering in transmission control protocol (tcp): problems, solutions, and challenges,” *IEEE transactions on parallel and distributed systems*, vol. 18, no. 4, 2007.
[19] R. Ludwig and R. H. Katz, “The eifel algorithm: making tcp robust against spurious retransmissions,” *ACM SIGCOMM Computer Communication Review*, vol. 30, no. 1, pp. 30–36, 2000.
[20] J. C. Bennett, C. Partridge, and N. Shectman, “Packet reordering is not pathological network behavior,” *IEEE/ACM Transactions on Networking (TON)*, vol. 7, no. 6, pp. 789–798, 1999.
[21] V. Paxson, “End-to-end internet packet dynamics,” *IEEE/ACM Transactions on Networking (TON)*, vol. 7, no. 3, pp. 277–292, 1999.
[22] A. Jayasumana, “Improved packet reordering metrics,” 2008.
[23] B. Ye, A. P. Jayasumana, and N. M. Piratla, “On monitoring of end-to-end packet reordering over the internet,” in *Networking and Services, 2006. ICNS’06. International conference on*. IEEE, 2006, pp. 3–3.
[24] R. Braden, “Rfc-1122: Requirements for internet hosts-communication layers,” vol. 11, no. 3, pp. 82–89, 1989.
[25] M. Al-Fares, A. Loukissas, and A. Vahdat, “A scalable, commodity data center network architecture,” in *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 4. ACM, 2008, pp. 63–74.