PABO: Mitigating Congestion via Packet Bounce in Data Center Networks

Shi, Xiang; Wang, Lin; Zhang, Fa; Zheng, Kai; Mühlhäuser, Max; Liu, Zhiyong

published in
Computer Communications
2019

DOI (link to publisher)
10.1016/j.comcom.2019.04.002

document version
Publisher's PDF, also known as Version of record

document license
Article 25fa Dutch Copyright Act

Computer Communications 140–141 (2019) 1–14

Contents lists available at ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

PABO: Mitigating congestion via packet bounce in data center networks[☆]

Xiang Shi[a,b,1], Lin Wang[c,d,2], Fa Zhang[a], Kai Zheng[e,3], Max Mühlhäuser[d], Zhiyong Liu[a,*]

[a] High Performance Computer Research Center, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100095, China
[b] University of Chinese Academy of Sciences, Beijing, China
[c] Vrije Universiteit Amsterdam, The Netherlands
[d] Technische Universität Darmstadt, Germany
[e] Huawei Technologies, Beijing, China

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 has 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 flow completion time.

1. 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-buffered commodity switches at low 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 that 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

☆ A preliminary version of the paper has appeared at IEEE ICC 2017 Shi et al. (2017) [1]. This work was partially supported by the National Key Research and Development Program of China (grant number 2017YFB1010001), the National Natural Science Foundation of China (grant numbers 61520106005, 61761136014). L. Wang was funded by the German Research Foundation (DFG), Germany project 392046569 and by the subproject C7 within the DFG Collaborative Research Center 1053 (MAKI).

* Corresponding author.
E-mail address: zyliu@ict.ac.cn (Z. Liu).

[1] This work was done while X. Shi was visiting Technische Universität Darmstadt.
[2] Member, IEEE.
[3] Senior Member,IEEE.

https://doi.org/10.1016/j.comcom.2019.04.002
Received 28 July 2018; Received in revised form 3 April 2019; Accepted 3 April 2019
Available online 16 April 2019

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) [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 [4].

In this paper, we propose PAcket BOunce (PABO) [1], 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 per-flow granularity, which can help achieve significant performance gain for short-lived flows and burst 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.

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++ [7]. (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 2 describes the design rationale of PABO and details its components one by one. Section 3 discusses the implementation of PABO. Section 4 analyzes the relationship between PABO and the level of packet out-of-order. Sections 5 and 6 present the experimental results. Section 7 summarizes related work and Section 8 concludes the paper.

2. 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 [8,9]. 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.

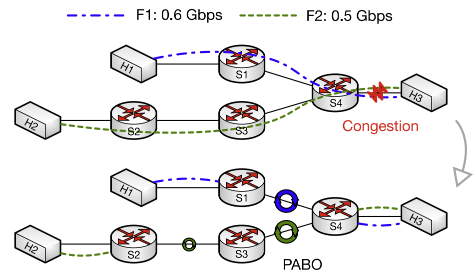

Fig. 1. A motivating example to demonstrate the idea of PABO, i.e., solving network congestion without dropping packets. We assume GigE links in the network.

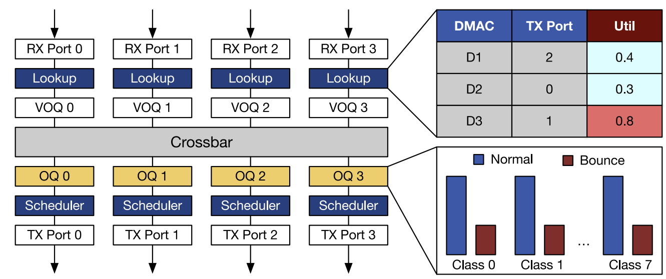

Fig. 2. An overview of PABO's design based on a Combined Input and Output Queue (CIOQ) switch model.

2.1. 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 Fig. 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.

2.2. Overview

We assume a general Combined Input and Output Queue (CIOQ) switch model [10] as depicted in Fig. 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 Fig. 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.

2.3. 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 $0 \le u_q \le \theta$ 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 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 Eq. (1) and combining all the above cases, we have the closed form of P satisfying PRCP-2 as

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

The curves of function P, with $\theta = 0.5$, under different utilization u_q are illustrated in Fig. 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 6.

2.4. 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 that we will describe in Section 2.6, 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 Fig. 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 $u_q \leftarrow u_q - 1/C_q$.

2.5. 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 Fig. 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 reduced queueing delay during their retry process.

2.6. 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 Fig. 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 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.

2.7. 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.

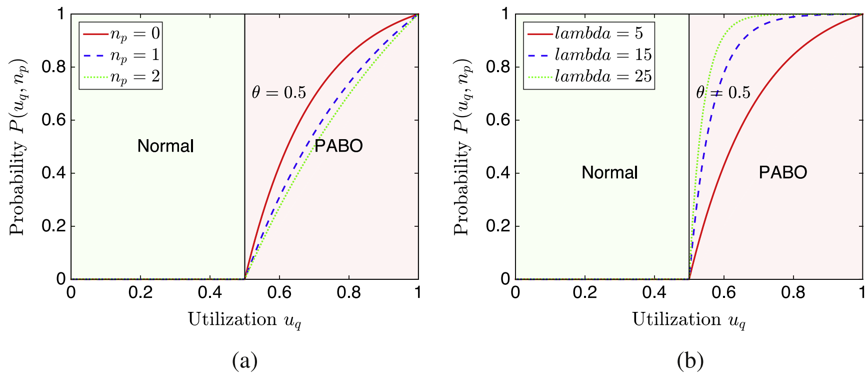

Fig. 3. Probability function $P(u_q, n_p)$ for packet bounce decision-making. We set the threshold θ to 0.5: (a) we set the constant λ to 5, and we show the curves in cases of $n_p = 0, 1, 2$, respectively, and (b) we show the curves with $lambda = 5, 15, 25$ in the case of $n_p = 0$. When $u_q \leq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

3. Implementation

To validate the effectiveness of PABO, we completed a proof-of-concept implementation based on the INET framework for OMNeT++ [7]. Our implementation code is open-sourced at [11]. 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 `MACRelayUnit`, `EtherQosQueue`. In `MACRelayUnit`, We alter the implementation of the forwarding strategy (i.e., function `handleAndDispatchFrame`) by applying our probability-based forwarding decision-making mechanism. During the decision-making process, the required queue utilization of the corresponding output interface is obtained by dividing instantaneous queue length by queue capacity. 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 IEEE 802.3 Ethernet frame [12]. 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 Ethernet header of an Ethernet frame [12]. 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 retransmission. 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 `StandardHost`.

In addition to the PABO implementation, we make some special modifications to the transport layer of the `StandardHost` to measure 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 reception 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.

The real-world deployment of PABO includes packet structure, switch function, and end-host support. PABO requires a counter to be added as a new field in the header of an Ethernet frame, indicating the number of times the packet has been bounced. This counter adds two octets to the head. PABO is designed in the scope of a CIOQ switch, which is commonly used in switches today [13,14]. Our implementation of an EtherSwitch module in OMNet++ resembles a CIOQ switch structure. Therefore, it can be used for reference in the actual deployment. The modification to a CIOQ switch includes two parts: the lookup unit and the output queue. In our implementation in OMNet++, the `MACRelayUnit` module corresponds to the lookup unit; and the `EtherQosQueue` corresponds to the output queue structure. In the end-hosts, modifications should be made to support the function of PABO, including link layer and transport layer modification. The link layer modification can refer to the implementation of `StandardHost`. The retransmission in the transport layer can be largely avoided as PABO improves the reliability of the network since packets will not be dropped because of buffer overflow. Meanwhile, the threshold that triggers fast retransmit should be carefully set so that fast retransmit only happen when there is really a need, e.g., packets are corrupted due to reasons such as hardware failures.

The overhead of PABO includes the traffic overhead brought by packet structure modification, and time overhead of the probabilistic decision-making. To record the number of bounced times for each packet, n_p is introduced as a new header field of 2 bytes in the Ethernet frame for simplicity. In the probabilistic decision-making process, the required PABO operations for each packet arriving at a switch are listed as follows: Firstly, calculate outbound buffer utilization based on the instantaneous queue size; Secondly, if the utilization is higher than the bounce threshold θ, calculate the bounce probability based on n_p of the packet and decide the forwarding port based on the probability; Finally, forward the packet the decided port. At nanosecond granularity provided by OMNET++, we did not observe any noticeable time overhead brought by PABO.

4. Packet out-of-order analysis

Out-of-order delivery [15] is the delivery of packets arriving at receiver disobeying their sending order. 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 brought by out-of-order packets can trigger spurious fast retransmit [16], bringing unnecessary packet delay as well as reduced congestion window. Studies have shown that packet out-of-order delivery is not rare [17,18] 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 [15]. To deal with this problem, [19] increases the number of duplicate acknowledgments required to trigger fast retransmit and [20] disables fast retransmit entirely.

In our case, PABO allows packets to be retuned on the forwarding path in order to guarantee no 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. This effect on packet out-of-order is also a congestion indication that PABO tries to convey to TCP. If the number of duplicate ACKs grows to a certain threshold, TCP can finally handle the congestion by suppressing the sending rate. However, window reduction should be avoided in transient congestions as it can result in unnecessary performance damage, while things can be solved right by using PABO instead. 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 provide insights focusing on PABO configuration. Meanwhile, we observe the performance of PABO on how packet out-of-order affects the flow completion time.

4.1. Measuring packet out-of-order

To measure the level of packet out-of-order delivery in a packet flow, 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 [21]. Consider a sequence of packets sent in the order of $[1, 2, \dots, N]$, 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 duplicate 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 $RD[k]$.

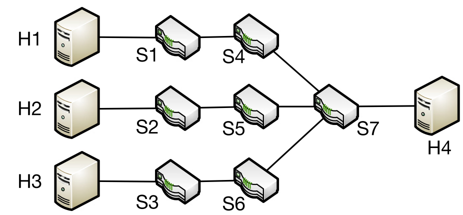

Fig. 4. The network topology used for investigating the impact of PABO on the level of packet out-of-order, and also for conducting the hop-by-hop evaluation of PABO.

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

Based on RD, we now derive a new metric called *reorder entropy* to quantitatively analyze the properties of the distribution [22]. Reorder entropy uses a single value to characterize the level of out-of-order in a packet flow, 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.

4.2. 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 Eq. (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 Fig. 4. All the links in the network are assumed to have the same rate of 1 Gbps. 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 flows 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. *The related simulation parameters are summarized in Table 1.* 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 (1 MiB) to the server, then the server responds immediately with the requested length of data. Each TCP request represents 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 the above traffic as the partition/aggregate traffic for ease of expression. We repeat this partition/aggregate traffic ten times and analyze the average results. The time gap between TCP requests is set

Table 1
Parameters for packet out-of-order analysis.

Parameters	Value
Link rate	1 Gbps
TCP request length	200 B
TCP reply length	1 MiB
Queue size	100 pkt
Advertised window	45,535 B
Gap between requests	1 s

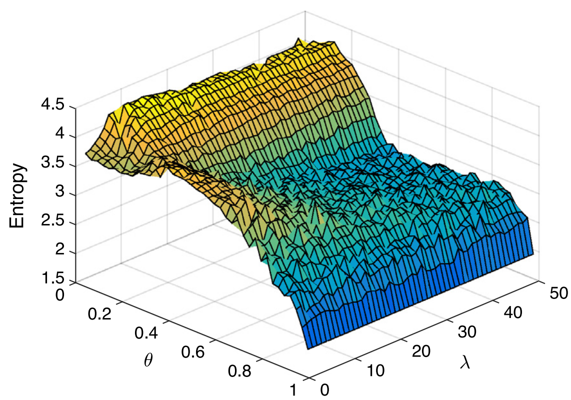

Fig. 5. The relationship between entropy and θ, λ in bounce probability function. Darker color of a grid means smaller entropy. In general, the change of θ has greater influence on entropy than that of λ.

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 up small buffer sizes and bursty flow rates. 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 45,535 bytes to enable the growth of flow rate. We also disable fast retransmit in TCP and set the retransmission timeout (RTO) to its upper bound 240 s [23]. 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 Fig. 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 entropy that 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.

Fig. 6 depicts the separate impact of θ and λ on the 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 entropy in all the six figures is highly correlated to timeRatio, which confirms that PABO can affect the level of packet out-of-order.

Fig. 6(a)–(c) illustrate the impact of parameter θ on packet out-of-order. According to Eq. (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 entropy. In Fig. 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 Fig. 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 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.

Fig. 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 Fig. 6(f)), the influence of λ can be ignored, since it hardly affects the bounce probability. As is demonstrated in Fig. 6(d) that the entropy rises rapidly at first and then remains a steady point. However, Fig. 6(e) illustrates an opposite tendency of entropy. This is because when $\theta = 0.5$, S7 is the only switch among the topology in Fig. 4 that satisfies the condition for bouncing back packets. As λ rises, the percentage of bounced packets continues to grow until it is infinitely close to one. 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.

4.3. Impact of entropy on flow completion time

Following the experiments in 4.2, we also observe how the level of packet out-of-order affects the performance of PABO in terms of flow completion time (FCT). FCT is determined by the time taken from the client sends a request to its reception of the last packet of the corresponding response. FCT includes the network transmission time and the data processing time (e.g. data reordering) at the hosts. In our experiment setting, the network transmission time accounts for a large proportion of the FCT. Fig. 7 illustrates the relationship between entropy and FCT. We can observe that with the variation of λ and θ, the corresponding entropies are more densely distributed on the right part of Fig. 7. Results show that with the increase of entropy, there is no obvious trend in FCT. This is reasonable because the network transmission time is determined by the arrival time of the last packet. The parameters of PABO decide when to start bounce back packets and how many packets are bounced. However, the variation of parameters will not affect the traffic amount and the packet departure rate at the output queue of the bottleneck switch. And for this reason, the arrival time of the last packet remains more or less the same despite the change of PABO parameters. However, under the condition that the data processing time accounts for the majority of the FCT, we acknowledge that FCT should have some linear correlations with the packet out-of-order level.

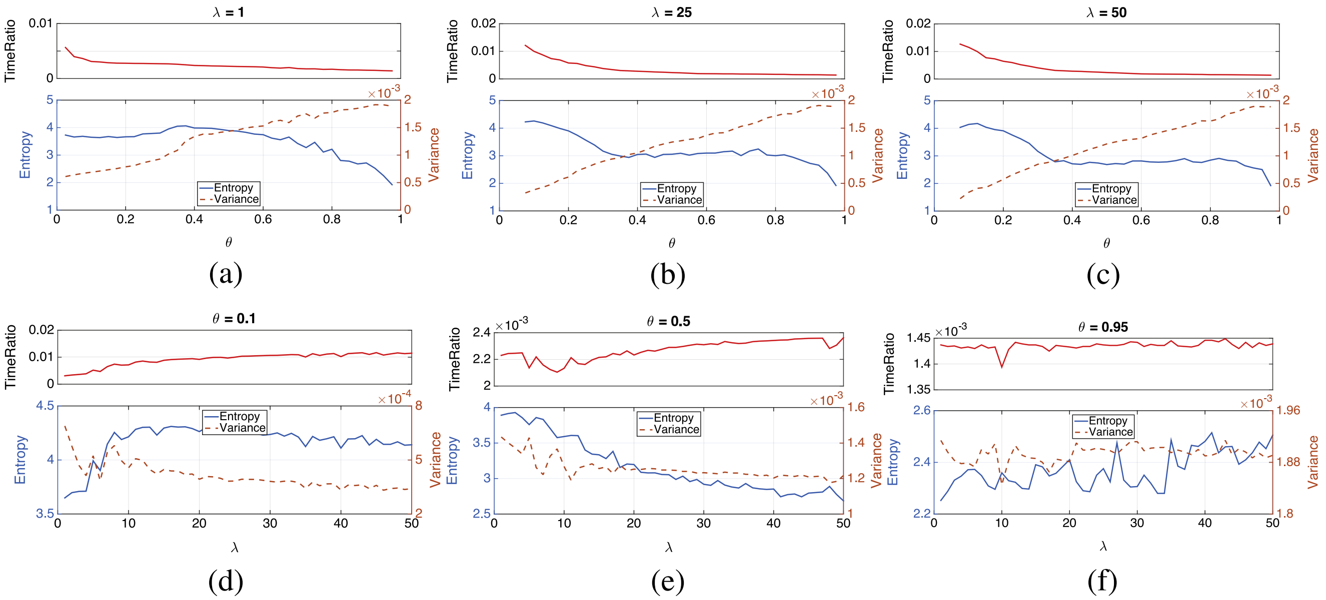

Fig. 6. Separate impact of parameter θ, λ in bounce probability function on entropy, variance and timeRatio: (a)–(c) the influence of fixed λ and change of θ, and (d)–(f) the influence of fixed θ and change of λ.

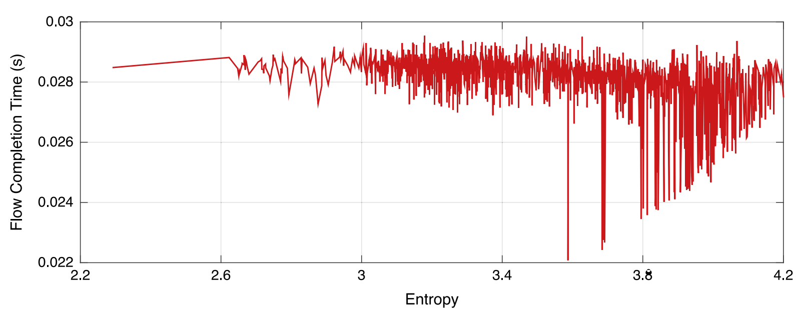

Fig. 7. The relationship between the flow completion time and entropy reflecting the level of packet out-of-order. The entropy is derived from the experiment results with the variation of λ from 1 to 50 and θ from 0 to 0.975.

4.4. Discussion

Although the change of PABO parameters does not obviously affect the flow completion time, the setting of θ and λ is still a trade-off issue. The main idea of PABO is to utilize the buffer of upstream switches to relieve traffic burdens on the bottleneck switch, making the packets more evenly distributed in the network. Consequently, the transient congestion (e.g. the incast problem) is relieved. Smaller θ and larger λ make PABO more aggressive and effective. With smaller θ, PABO tends to bounce back packets earlier, and larger λ makes the bounce probability grow faster. In this case, the utilization of each switch is maintained at a relatively low level, but the level of packet out-of-order will be more severe. Larger θ and smaller λ make PABO more conservative and less effective. PABO tends to bounce late and the grow rate of bounce probability is slower. This way, the level of packet out-of-order is less severe, but the utilization of the bottleneck switch will be maintained at a relatively high level, which can render the buffer to be saturated easily. Therefore, we should balance between variance (representing the effectiveness of PABO) and the entropy (reflecting the extent of out-of-order), by carefully choosing the parameters for target networks. The settings of parameters can be categorized into different cases in terms of traffic characteristics. If traffic condition is bursty, it can quickly ramp up to high network utilization. In this case, if θ is set to a large value, latency can increase because packet bursts are absorbed in the network buffer. Therefore, it is suggested that we bounce back early with a small θ value. If traffic condition is light and with few burst traffic, θ should be set to a large value to largely eliminate congestion in the network. For λ, we suggest a large value to reduce the level of packet out-of-order. In general, the ideal objective is to keep maximally the packet bouncing scope right around the congestion point.

5. 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.

5.1. Simulation setup

We adopt the same topology as Section 4 in Fig. 4, except that the hosts do not 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 related simulation parameters are illustrated in Table 2.* The duration of all the simulations is set to one second to cover multiple appearances of periodic congestions and the experiment is repeated ten times. The links in the network are assumed to have the same rate of 1 Gbps.

Traffic. We imitate a periodic uniform flow by altering the traffic-generating module. We define *uniFlow* as a unit of the periodic uniform

Table 2
Parameters for hop-by-hop evaluation.

Parameters	Value
λ	50
θ	0.8
Link rate	1 Gbps
sendInterval	12 μs
pauseInterval	0.2 s
numPacketsPerGenerate	500 1500 2500
PABO sender queue size	1500 pkt
PABO switch queue size	500 pkt
Payload size	1500 bytes

Table 3
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%	–

flow. As the overlap of two consecutive uniFlow may bring persistent congestions that will overflow the sender's buffer, we modify the `EtherTrafGen` module to enable setting of the periodic interval, which is the `pauseInterval`. In a uniFlow, the packets are sending out at a uniform rate. We make some modifications to the existing modules in order to maintain a uniform sending rate. First, we fix the sending interval between packets and define it as `sendInterval`. Then, we introduce a new parameter `numPacketsPerGenerate` to specify the number of packets to be sent in each uniFlow. The duration of each uniFlow can be roughly calculated by `numPacketsPerGenerate` × `sendInterval`. To specify the sending rate, we fix the value of `sendInterval` to be 12 μs. Then, we control the traffic amount of each uniFlow by tuning `numPacketsPerGenerate`. With a stable sending rate, different `numPacketsPerGenerate` brings different severities of traffic burdens to the network. We set `pauseInterval` to be 0.2 s, 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 concern.

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

5.2. 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% and 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 loss can be prevented. The number of bounced packets at each switch in all the three scenarios is illustrated in Fig. 8(a) and the proportion of `maxBouncedDistance` is shown in Table 3. 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 Fig. 8(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 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 Fig. 9. We monitor the utilization levels of `normalQueue` and `bounceQueue` separately, and calculate the average utilization of the two queues at the output. Fig. 9(a) depicts the average utilization of the relevant switches over the whole duration of the simulation that includes five appearances of transient congestion. We then focus on the first traffic peak as illustrated in Fig. 9(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.

5.3. 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 Fig. 10. 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 Fig. 10(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.

6. End-to-end evaluation in realistic topology

We further evaluate the performance of PABO in the Fattree network topology [24] 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, and then we present the parameter study of θ about the impact of PABO on entropy, per-packet delay and flow completion time.

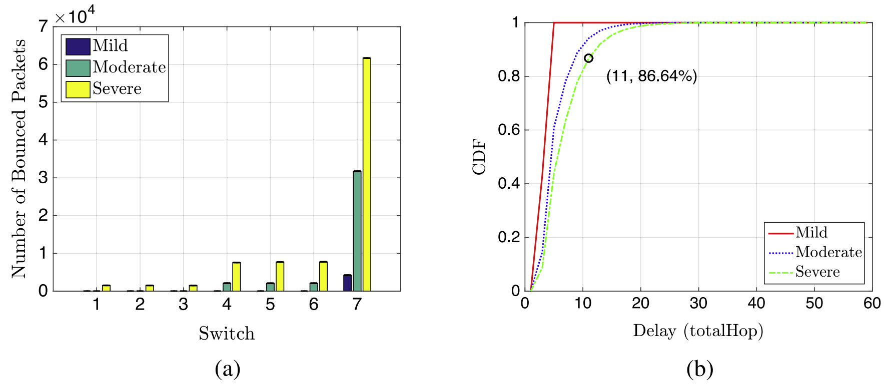

Fig. 8. Performance results of PABO: (a) Packet bounce frequency under different severities of congestion, and (b) the CDF of the packet delay measured by `totalHop`.

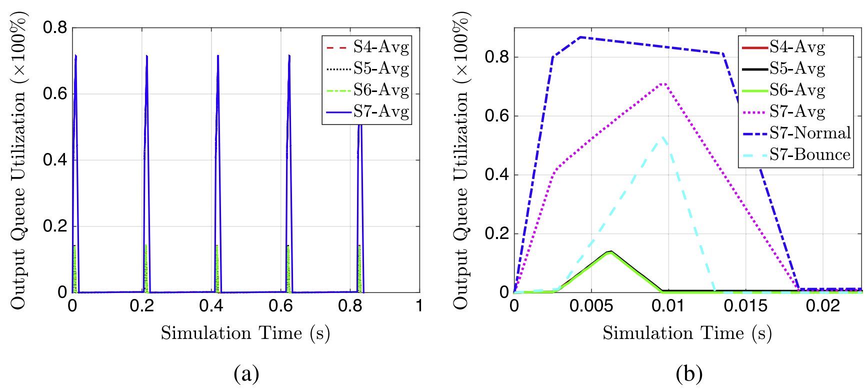

Fig. 9. Switch output queue utilization under the mild scenario: (a) over the whole duration of the simulation, and (b) over the first traffic peak.

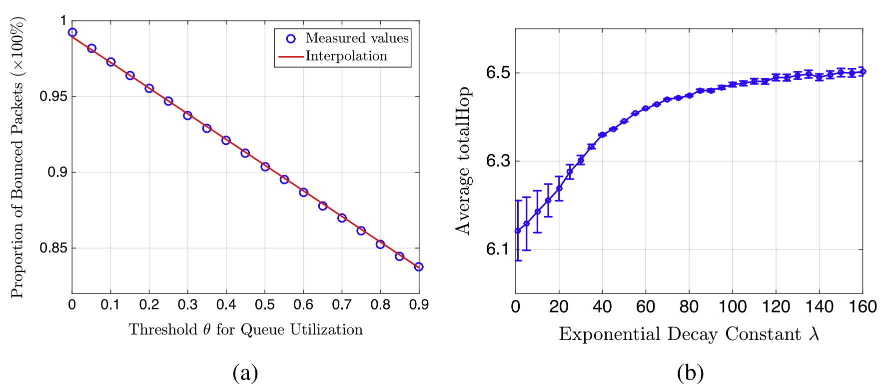

Fig. 10. Impact of the parameters (a) θ and (b) λ on the proportion of bounced packets and the average packet delay measured by `totalHop`, respectively. The proportion of bounced packets decreases linearly with the increase of θ, as θ is the threshold for PABO begins to work. The average `totalHop` of each packet increases gradually stable with the increase of λ, for the growth of λ means the increasing effectiveness of PABO.

6.1. Fattree implementation

We implement the Fattree routing algorithm (used in two-level routing table) on `EtherSwitch`, and calling it the `FattreeSwitch`. For the reason that this 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 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.

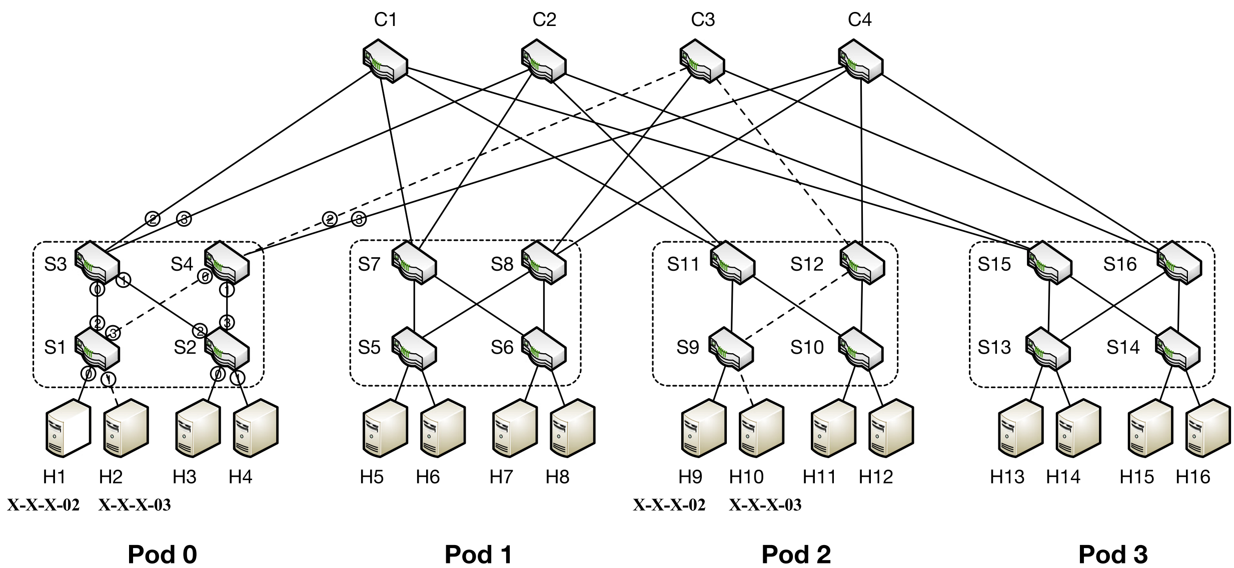

Fig. 11. The Fattree topology used for evaluating the performance of PABO. Using the two-level routing tables described in Section 6.1, packets from source H2 to destination H10 would take the dashed path.

Routing Example. Here we explain the Fattree two-level routing algorithm implemented in Fig. 11. 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 Fig. 11. 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.

6.2. Simulation setup

We use the Fattree network topology depicted in Fig. 11 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 simulation parameters are presented in Table 4.* 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 1 Gbps.

Traffic. We simulate different traffic patterns by changing the number of servers and clients. For each TCP connection, we use the same partition/aggregate traffic and buffer setup as Section 4, except that the advertised window is set to 50,000 bytes, and we repeatedly perform the experiment ten times. When using PABO, we disable all the retransmissions (both fast retransmit and retransmission timeout) as well as skipping the related window reduction 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.

Table 4
Parameters for end-to-end evaluation.

Parameters	Value
λ	50
θ	0.95
Link rate	1 Gbps
TCP request length	200 B
TCP reply length	1 MiB
Queue size	100 pkt
Advertised window	50,000 B

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

6.3. 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). Then we observe how PABO performs in different congestions comparing to cases without PABO.

Fig. 12 demonstrates the packet bounce frequency of each switch under different many-to-one scenarios. 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 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

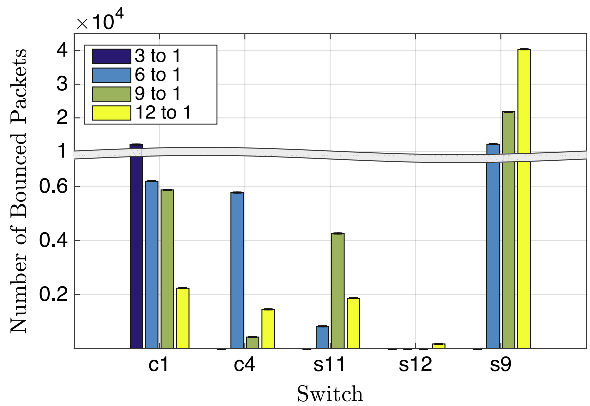

Fig. 12. Packet bounce frequency of each switch under different many-to-one scenarios, measured by number of bounced packets at each switch. The total frequency of a scenario increases with the growth of the number of servers. The middle part is omitted using a break axis for the convenience of showing data with a wide range.

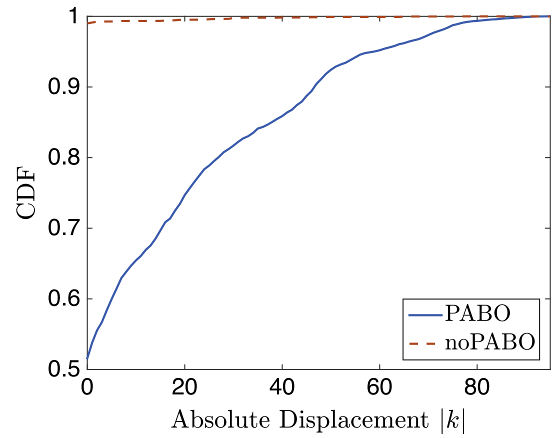

Fig. 13. The CDF of absolute packet displacement in the 12 to 1 scenario. In cases without PABO, the percentage of out-of-order packets is much smaller, but there are still packets with large displacement values, which can trigger multiple times of retransmission timeout.

Table 5
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%	–

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 5 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.

Fig. 13 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. 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 can trigger multiple times of retransmission timeout, resulting in a substantial delay stretch.

We also focus on the delay comparison between cases with PABO and cases without PABO. Fig. 14 illustrates the average delay per packet, regarding the time spent from source to destination. Per-packet delays are recorded when packets successfully arrive at the receiver side, thus the delays of dropped packets are not recorded in cases without PABO. We can see that the average per-packet delay of PABO is slightly higher than cases without PABO. This is because some of the packets are bounced back by PABO to avoid packet loss, leading to per-packet delay stretched. 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.

Fig. 15 depicts the average flow completion time in each scenario, which refers to the time taken from the client sends a request to its reception of the last packet of the corresponding response, neglecting the reordering process in the receive buffer. Therefore, the flow completion time is determined by the arriving time of the last packet. From the figure, we can see that cases with PABO have obvious advantage over cases without PABO in all the scenarios. Though bouncing slightly increases per-packet delay, packet drops and retransmissions are avoided, which can bring orders of magnitude delay increases. Therefore, the arriving time of the last packet can be improved by PABO. Moreover, the advantage of PABO on flow completion time 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).

6.4. Many-to-many scenario

Second, we evaluate PABO under the many-to-many scenario, which consists of two partition/aggregate traffic. In this scenario, H1, H5 and H13 are the servers that 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%

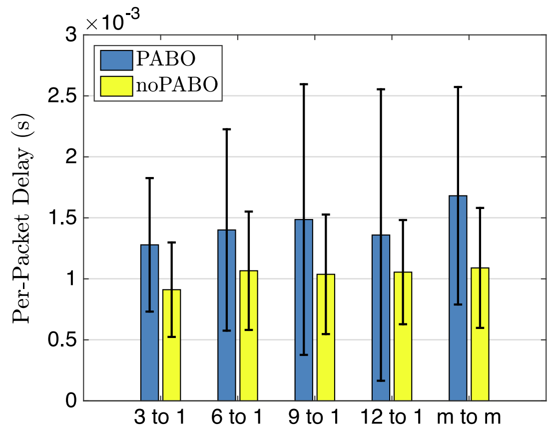

Fig. 14. Average per-packet delay under different congestion scenarios. The results with PABO is slightly higher than the results without PABO as PABO bounces back some of the packets.

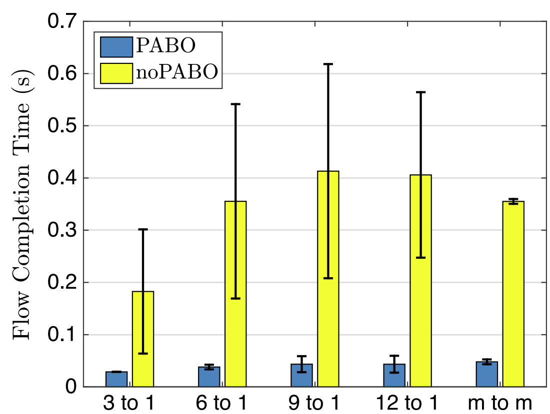

Fig. 15. Average flow completion time in each congestion scenario. The results with PABO has obvious advantage over the results without PABO, for packet losses leading to the orders of magnitude delay increases in retransmission-based approaches of TCP Reno.

at C2 and 62.88% at S11. 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 results of flow completion time, PABO still has great advantage over the normal case.

6.5. 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 is unnecessary to bounce back too early. Therefore, we only focus on the domain of $\theta \geq 0.5$. The results are illustrated in Fig. 16. 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 4. As to the effect of θ on the flow completion time, it shows no obvious regularity. For no matter when 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 bounce threshold θ. For results of per-packet 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.

7. 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 [25] on switches to predict the extent of the congestion and provide smooth adjustments on the sending window size accordingly. Clove and ALB are solutions running in end-hosts and focusing on traffic load balancing on multiple paths. Clove [26] makes use of the virtual switch in the end-host hypervisor to control packet routing by changing the packet header. It relies on Equal-cost multi-path routing (ECMP) on switches to reroute flowlets over multiple paths, for the purpose of getting high link utilization and low packet reordering. ALB [27] further improves performance by eliminating inaccurate congestion feedbacks under asymmetric topologies. 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 [28] and PI [29]). ECN (Explicit Congestion Notification) [25] 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 [30] 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 halt the transmission of the sender for a specified period of time. However, PAUSE frame is per-link based and cannot differentiate among flows, which leads to performance collapse of all the flows on the link. 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 [31]. BCN (Backward Congestion Notification) [32] and FECN (Forward Explicit Congestion Notification) [33] are queue-based mechanisms that provide congestion notification for rate control mechanisms. BCN monitors the queue length of the switch, and it will send messages to the source for rate control when the queue length is above some predefined level. FECN probes the congestion condition proactively and periodically along the path to the destination, then it will use feedbacks from the network to adjust sending rate of the source. However, flows are not differentiated in BCN and FECN, thus the rate control mechanism will affect all the flows on the link.

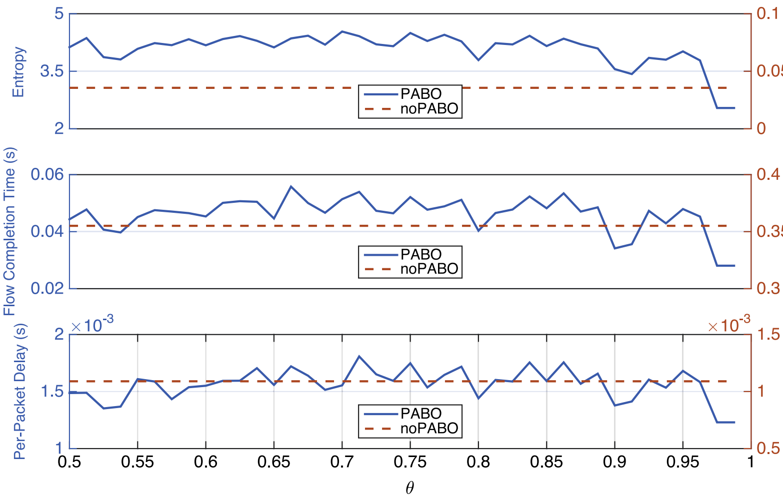

Fig. 16. The relationship between θ, entropy, per-packet delay and flow completion time in the many-to-many scenario.

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 three major additional merits: (i) Packets are detoured by bouncing to upstream switches, which can minimize inter-flow interference on other paths, and is more manageable in maintaining packet order: bounced packets of a flow that queue up in the bounce queue are transmitted following First In First Out (FIFO) order along the same path. (ii) The bounced packets can also serve as a congestion notification for upstream nodes (switches or end-hosts). (iii) For considerations on reducing per-packet delay, the bounced packets are differentiated from normal packets and are limited in the number of times to be bounced.

8. 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, which can mitigate transient congestion in network at 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, and then we provide useful insights for configuring PABO. Extensive simulations have proved the effectiveness of PABO, showing that PABO has obvious superiority of flow completion time over the traditional protocol stack by guaranteeing zero packet loss in all cases. The appropriate configuring of PABO can vary with different network conditions. For network operators, we provide insights on how to configure PABO under known network conditions. However, network conditions vary greatly within different target networks. We claim that decision-making frameworks like reinforcement learning can be used to assist in monitoring and predicting network dynamics in the future. After the prediction, the predicted network dynamic results can be used to make PABO more properly configured.

Conflict of interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] X. Shi, L. Wang, F. Zhang, K. Zheng, Z. Liu, PABO: congestion mitigation via packet bounce, in: IEEE International Conference on Communications, ICC 2017, Paris, France, May 2017 21-25, 2017, pp. 1–6. [Online]. Available: https://doi.org/10.1109/ICC.2017.7996995.

[2] M. Alizadeh, A.G. Greenberg, D.A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, M. Sridharan, Data center TCP (DCTCP), in: Proceedings of the ACM SIGCOMM 2010 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, New Delhi, India, August 30 - September 3, 2010, 2010, pp. 63–74. [Online]. Available: http://doi.acm.org/10.1145/1851182.1851192.

[3] H. Wu, Z. Feng, C. Guo, Y. Zhang, ICTCP: incast congestion control for TCP in data center networks, in: Proceedings of the 2010 ACM Conference on Emerging Networking Experiments and Technology, CoNEXT 2010, Philadelphia, PA, USA, November 30 - December 03, 2010, 2010, p. 13. [Online]. Available: http://doi.acm.org/10.1145/1921168.1921186.

[4] K. Zarifis, R. Miao, M. Calder, E. Katz-Bassett, M. Yu, J. Padhye, DIBS: just-in-time congestion mitigation for data centers, in: Ninth Eurosys Conference 2014, EuroSys 2014, Amsterdam, The Netherlands, April 2014 13-16, 2014, pp. 6:1–6:14. [Online]. Available: http://doi.acm.org/10.1145/2592798.2592806.

[5] O. Feuser, A. Wenzel, On the effects of the IEEE 802.3x flow control in full-duplex ethernet lans, in: Proceedings 26th Conference on Local Computer Networks, Lowell, Massachusetts, USA, 17-20 October, 1999, 1999, pp. 160–161. [Online]. Available: https://doi.org/10.1109/LCN.1999.802012.

[6] Y. Chen, R. Griffith, J. Liu, R.H. Katz, A.D. Joseph, Understanding TCP incast throughput collapse in datacenter networks, in: Proceedings of the 1st ACM SIGCOMM 2009 Workshop on Research on Enterprise Networking, WREN 2009, Barcelona, Spain, August 21, 2009, 2009, pp. 73–82. [Online]. Available: http://doi.acm.org/10.1145/1592681.1592693.

[7] OMNeT++. [Online]. Available: https://inet.omnetpp.org/.

[8] S. Kandula, R. Mahajan, Sampling biases in network path measurements and what to do about it, in: Proceedings of the 9th ACM SIGCOMM Internet Measurement Conference, IMC 2009, Chicago, Illinois, USA, November 2009 4-6, 2009, pp. 156–169. [Online]. Available: http://doi.acm.org/10.1145/1644893.1644912.

[9] S. Kandula, J. Padhye, P. Bahl, Flyways to de-congest data center networks, in: Eight ACM Workshop on Hot Topics in Networks (HotNets-VIII), HOTNETS '09, New York City, NY, USA, October 2009 pp. 22-23, 2009. [Online]. Available: http://conferences.sigcomm.org/hotnets/2009/papers/hotnets2009-final112.pdf.

[10] A. Kesselman, A. Rosén, Scheduling policies for CIOQ switches, J. Algorithms 60 (1) (2006) 60–83, [Online] Available: https://doi.org/10.1016/j.jalgor.2004.09.003.

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

[12] Ethernet frame. [Online]. Available: https://en.wikipedia.org/wiki/Ethernet_frame.

[13] Cisco ASR 9900 Series. [Online]. Available: https://www.cisco.com/c/en/us/products/collateral/routers/asr-9000-series-aggregation-services-routers/datasheet-c78-741129.html.

[14] QFX10008 and QFX10016 Ethernet Switches. [Online]. Available: https://www.juniper.net/uk/en/products-services/switching/qfx-series/qfx10000/.

[15] K. Leung, V.O.K. Li, D. Yang, An overview of packet reordering in transmission control protocol (TCP): problems, solutions, and challenges, IEEE Trans. Parallel Distrib. Syst 18 (4) (2007) 522–535, [Online]. Available: https://doi.org/10.1109/TPDS.2007.1011.

[16] R. Ludwig, R.H. Katz, The eifel algorithm: making TCP robust against spurious retransmissions, Comput. Commun. Rev. 30 (1) (2000) 30–36, [Online]. Available: http://doi.acm.org/10.1145/505688.505692.

[17] J.C.R. Bennett, C. Partridge, N. Shectman, Packet reordering is not pathological network behavior, IEEE/ACM Trans. Netw. 7 (6) (1999) 789–798, [Online]. Available: https://doi.org/10.1109/90.811445.

[18] V. Paxson, End-to-end internet packet dynamics, IEEE/ACM Trans. Netw. 7 (3) (1999) 277–292, [Online]. Available: https://doi.org/10.1109/90.779192.

[19] M. Zhang, B. Karp, S. Floyd, L.L. Peterson, RR-TCP: A reordering-robust TCP with DSACK, in: 11th IEEE International Conference on Network Protocols (ICNP 2003), 4-7 2003, Atlanta, GA, USA, 2003, pp. 95–106. [Online]. Available: https://doi.org/10.1109/ICNP.2003.1249760.

[20] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, S. Shenker, pfabric: minimal near-optimal datacenter transport, in: ACM SIGCOMM 2013 Conference, SIGCOMM'13, Hong Kong, China, August 2013 12-16, 2013, pp. 435–446. [Online]. Available: http://doi.acm.org/10.1145/2486001.2486031.

[21] A.P. Jayasumana, N.M. Piratla, T. Banka, A.A. Bare, R. Whitner, Improved packet reordering metrics, RFC 5236 (2008) 1–26, [Online]. Available: https://doi.org/10.17487/RFC5236.

[22] B. Ye, A.P. Jayasumana, N.M. Piratla, On monitoring of end-to-end packet reordering over the internet, in: 2006 International Conference on Networking and Services (ICNS 2006), 16-21 2006, Silicon Valley, California, USA, 2006, p. 3. [Online]. Available: https://doi.org/10.1109/ICNS.2006.79.

[23] R.T. Braden, Requirements for internet hosts - communication layers, RFC 1122 (1989) 1–116, [Online]. Available: https://doi.org/10.17487/RFC1122.

[24] M. Al-Fares, A. Loukissas, A. Vahdat, A scalable, commodity data center network architecture, in: Proceedings of the ACM SIGCOMM 2008 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, Seattle, WA, USA, August 2008 17-22, 2008, pp. 63–74. [Online]. Available: http://doi.acm.org/10.1145/1402958.1402967.

[25] Explicit Congestion Notification. [Online]. Available: https://tools.ietf.org/rfc/rfc3168.txt.

[26] N.P. Katta, A. Ghag, M. Hira, I. Keslassy, A. Bergman, C. Kim, J. Rexford, Clove: Congestion-aware load balancing at the virtual edge, in: Proceedings of the 13th International Conference on emerging Networking Experiments and Technologies, CoNEXT 2017, Incheon, Republic of Korea, December 2017 12-15, 2017, pp. 323–335. [Online]. Available: https://doi.org/10.1145/3143361.3143401.

[27] Q. Shi, F. Wang, D. Feng, W. Xie, Alb: Adaptive load balancing based on accurate congestion feedback for asymmetric topologies.

[28] S. Floyd, V. Jacobson, Random early detection gateways for congestion avoidance, IEEE/ACM Trans. Netw. 1 (4) (1993) 397–413, [Online]. Available: https://doi.org/10.1109/90.251892.

[29] C.V. Hollot, V. Misra, D.F. Towsley, W. Gong, On designing improved controllers for AQM routers supporting TCP flows, in: Proceedings IEEE INFOCOM 2001, The Conference on Computer Communications, Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies, Twenty years into the communications odyssey, Anchorage, Alaska, USA, April 2001 22-26, 2001, pp. 1726–1734. [Online]. Available: https://doi.org/10.1109/INFCOM.2001.916670.

[30] D. Zats, A.P. Iyer, G. Ananthanarayanan, R. Agarwal, R.H. Katz, I. Stoica, A. Vahdat, Fastlane: making short flows shorter with agile drop notification, in: Proceedings of the Sixth ACM Symposium on Cloud Computing, SoCC 2015, Kohala Coast, Hawaii, USA, August 2015 27-29, 2015, pp. 84–96. [Online]. Available: http://doi.acm.org/10.1145/2806777.2806852.

[31] D. Zats, T. Das, P. Mohan, D. Borthakur, R.H. Katz, Detail: reducing the flow completion time tail in datacenter networks, in: ACM SIGCOMM 2012 Conference, SIGCOMM '12, Helsinki, Finland - August 2012 13-17, 2012, pp. 139–150. [Online]. Available: http://doi.acm.org/10.1145/2342356.2342390.

[32] J. Jiang, R. Jain, Analysis of backward congestion notification (BCN) for ethernet in: datacenter applications, in INFOCOM 2007. 26th IEEE International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies, 6-12 2007, Anchorage, Alaska, USA, 2007, pp. 2456–2460. [Online]. Available: https://doi.org/10.1109/INFCOM.2007.300.

[33] J. Jiang, R. Jain, C. So-In, An explicit rate control framework for lossless ethernet operation, in: Proceedings of IEEE International Conference on Communications, ICC 2008, Beijing, China, 19-23 2008, 2008, pp. 5914–5918. [Online]. Available: https://doi.org/10.1109/ICC.2008.1105.