Enabling Load Balancing for Lossless Datacenters

Jinbin Hu[1,2], Chaoliang Zeng[2], Zilong Wang[2], Junxue Zhang[2], Kun Guo[3], Hong Xu[4], Jiawei Huang[5], Kai Chen[2]
[1]Changsha University of Science and Technology, [2]Hong Kong University of Science and Technology
[3]Fuzhou University, [4]Chinese University of Hong Kong, [5]Central South University

***Abstract*—Various datacenter network (DCN) load balancing schemes have been proposed in the past decade. Unfortunately, most of these solutions designed for lossy DCNs do not work well for Priority Flow Control (PFC) enabled lossless DCNs, primarily due to the reason that the individual congestion signals used in these solutions, e.g., link load, queue length, Round Trip Time (RTT) and Explicit Congestion Notification (ECN), may not be able to correctly or timely reflect the hop-by-hop PFC pausing.**

This paper first reveals the above problems via extensive experiments, and then based on the insights learned, we present Proteus, a PFC-aware load balancing scheme that is resilient to PFC pausing by exploring a combination of multi-level congestion signals. At its heart, Proteus leverages RTT-level signals (i.e., RTT and link utilization) to detect path status for initial routing decision, and exploits sub-RTT level signal (i.e., cumulative sojourn time) to reflect instantaneous PFC pausing and make timely rerouting choices based on the idea of better-late-than-never. We have implemented Proteus in the hardware programmable switch. Our testbed experiments as well as large-scale simulations show that Proteus can effectively handle PFC pausing under realistic workloads and achieve up to 35%, 31%, 28%, 22% and 46%, 42%, 34%, 29% better average FCT and 99^{th} percentile FCT than CONGA, DRILL, Hermes and MP-RDMA, respectively.

***Index Terms*—Datacenter, Lossless Networks, Load Balancing**

I. Introduction

Driven by the stringent demands for ultra-low latency and high throughput from datacenter applications such as distributed storage and machine learning training [1]–[10], reliable lossless networks without packet loss are increasingly crucial in Ethernet-based DCNs deployed with Remote Direct Memory Access (RDMA) [11]–[15]. RoCEv2 (RDMA over Converged Ethernet), the de-facto standard protocol for high-speed networks, relies on hop-by-hop PFC to prevent buffer overflow caused by network congestion [16]–[20].

In the meanwhile, to strive to balance the ever increasing traffic over multiple equal-cost paths in DCNs, a rich body of load balancing schemes [21]–[26] have been proposed in the past decade. However, these existing load balancing solutions designed for lossy DCNs with packet loss do not work well for PFC-enabled lossless DCNs. As shown in Table I and experimentally demonstrated in §II, the crux is that the introduction of PFC would invalidate congestion signals in interpreting network congestions as before. In particular, we make the following observations:

- First, link load used in the schemes such as CONGA [21] cannot reflect PFC pausing correctly. A path with low link utilization due to PFC pausing is a bad path, but it may be misinterpreted as a good path without congestion.

979-8-3503-0322-3/23/$31.00 ©2023 IEEE

TABLE I: Issues of congestion signals in the prior load balancing solutions in PFC-enabled DCNs.

Congestion Signals	Existing Schemes	Issues in PFC-enabled Networks
Link load	CONGA [21] HULA [22]	Low link load due to PFC pausing is misinterpreted as good
End-to-end signals (RTT, ECN)	Hermes [23] MP-RDMA [15]	Stale for hop-by-hop PFC pausing due to control loop
Local queue length	DRILL [24]	Cannot sense remote PFC pausing

- Second, schemes such as Hermes [23] using end-to-end signals (RTT and ECN) to decide forwarding path cannot detect PFC pausing timely due to the intrinsic RTT-level feedback loop. In such case, a path with larger delay is not necessarily a worse path, as PFC may have just resumed.
- Third, local queue-based schemes such as DRILL [24] cannot reflect remote PFC pausing on a path, and thus a local egress port with smaller queue length is not necessarily a better path.

We will show that prior load balancing schemes can inflate FCT by up to 49% in PFC-enabled networks due to the well-known PFC's head-of-line (HoL) blocking problem (§II-B).

Based on the above observations, we propose Proteus, a PFC-aware load balancing scheme for lossless DCNs that works with the hop-by-hop PFC pausing (§III). At its heart, Proteus leverages RTT-level signals (RTT and link utilization) to identify the path state for the initial routing decision, and explores sub-RTT level signal (cumulative sojourn time—the cumulative queueing delay of a packet on consecutive switches on the path) for timely rerouting when experiencing unexpected PFC pausing.

Specifically, Proteus divides paths into three categories, i.e., non-congested path, congested path, and undetermined path, and uses RTT-level signals to capture the link status and distinguish the path category. We treat the path suffering PFC pausing as undetermined since the pausing may be short or long lasting. For non-congested paths, both RTT and link utilization are low; for congested paths, RTT is high due to queueing delay and the link is fully utilized; for undetermined paths where PFC has occurred, RTT is high due to queueing delay and the link utilization is less than 100% due to PFC pausing. Thus the combination of these two signals is able to indicate the path state at a coarse granularity in PFC-enabled networks, and guide the initial path selection in Proteus.

Furthermore, Proteus explores sub-RTT level signal, i.e., a packet's cumulative sojourn time (CST), to timely handle hop-

by-hop PFC pausing with the observation that the duration of a PFC pausing is unpredictable and prematurely kicking out the path may lead to a sub-optimal performance especially when the PFC pausing is short (§II-B). Therefore, when experiencing PFC pausing, Proteus follows the idea of better-late-than-never, which will reroute the packet only when the CST of the paused packet exceeds a maximum acceptable delay. Once the packet is rerouted, Proteus removes the paused path from the current available forwarding paths to avoid continuous HoL blocking.

We have implemented a Proteus prototype in a hardware programmable switch (§IV), and built a small-scale testbed to evaluate the effectiveness of Proteus (§V-A). To complement small-scale testbed experiments, we further conducted large-scale NS-3 simulations under realistic workloads to validate the performance of Proteus at scale (§V-B).

Overall, this paper makes the following contributions:

- We are among the first to conduct an extensive simulation study to reveal the issues of three typical congestion signals of existing load balancing schemes in PFC-enabled DCNs (§II-B): 1) link load cannot reflect PFC pausing correctly; 2) end-to-end signals cannot detect PFC pausing timely; 3) local queue cannot sense remote PFC pausing.
- We propose a PFC-aware load balancing scheme Proteus (§III), which makes forwarding decisions based on both RTT and sub-RTT level signals. Proteus chooses the initial best path based on RTT and link utilization. When packets are blocked due to PFC pausing, instead of switching path instantly or never, Proteus carefully considers whether to invalidate the current path according to CST and reroute packets based on the idea of better-late-than-never.
- By using both testbed implementation (§IV) and NS-3 simulations (§V), we demonstrate that Proteus significantly outperforms prior schemes. For example, under the Web Search workload [34], Proteus reduces the average flow completion time (AFCT) and 99^{th} percentile FCT by up to 35%, 31%, 28%, 22% and 46%, 42%, 34%, 29% compared to CONGA, DRILL, Hermes and MP-RDMA, respectively. Especially, Proteus achieves up to 1.8x reduction of AFCT for short flows less than 100KB.

The rest of this paper is organized as follows. §II explores problems. §III describes Proteus design. §IV gives the implementation of Proteus. §V presents testbed and simulation results, and §VI discusses related works before we conclude the paper in §VII.

II. Problem Exploration

A. PFC is Triggered Even with Congestion Control

PFC mechanism. To guarantee lossless transmission, Converged Enhanced Ethernet (CEE) employs the hop-by-hop flow control mechanism PFC defined by IEEE 802.1Qbb [27]. Once the ingress queue length exceeds a specified threshold, the switch sends PFC PAUSE to upstream related egress ports or queues to stop data transmission until receiving PFC RESUME to guarantee lossless transmission. Since PFC pausing is based on port or queue, the coarse reaction to congestion causes the well-known problems such as HoL blocking and congestion spreading [11]–[14], [28].

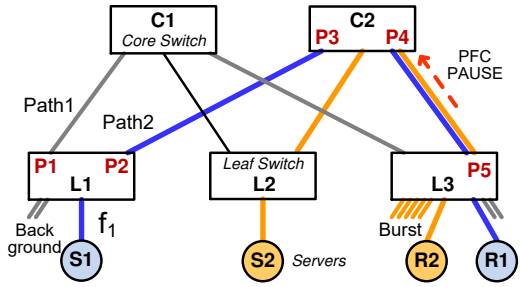


Fig. 1: Typical network scenario. The flows (in yellow) sent to the receiver $R2$ are responsible for the congestion at the ingress port $P5/L3$. The flow f_1 (in blue) sent to the receiver $R1$ is not responsible for the congestion at $P5/L3$.

PFC is triggered under bursty traffic. Recent proposed end-to-end transport protocols can efficiently control non-transient congestion and thus effectively reduce PFC triggering [11], [12], [29]–[31]. However, transient congestion caused by bursty traffic is widespread [9], [10], [32], [33]. Recent research reveals that over 60-90% short-lived flows usually finish within one RTT in modern DCNs [34]–[36]. It is hard for end-to-end transports to control bursy traffic, resulting in inevitable PFC triggering.

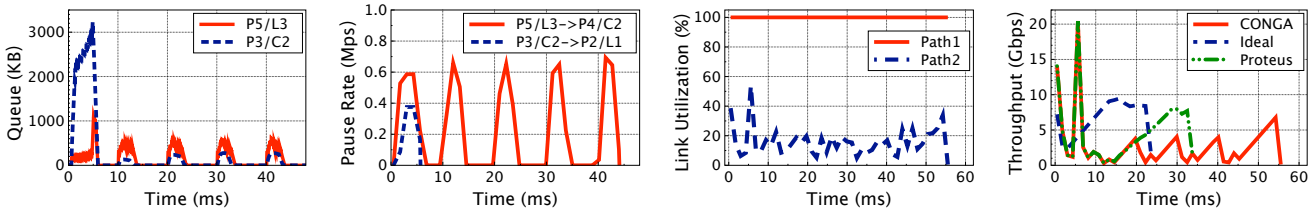
Fortunately, we can employ in-network load balancing to rapidly react to PFC pausing. However, although the existing load balancing schemes work well in lossy DCNs, they are inadequate to handle hop-by-hop PFC pausing in lossless DCNs as we will demonstrate in the following.

B. Problem Demonstration

To explore the drawbacks of three typical congestion signals used by existing load balancing schemes in lossless DCNs, we first conduct NS-3 simulations in a common Clos topology as shown in Fig. 1. The link capacity is 40Gbps, and the switch buffer size is 9MB with PFC enabled. We adopt DCQCN [11] as the underlying transport protocol and set parameters to the default values recommended in [11].

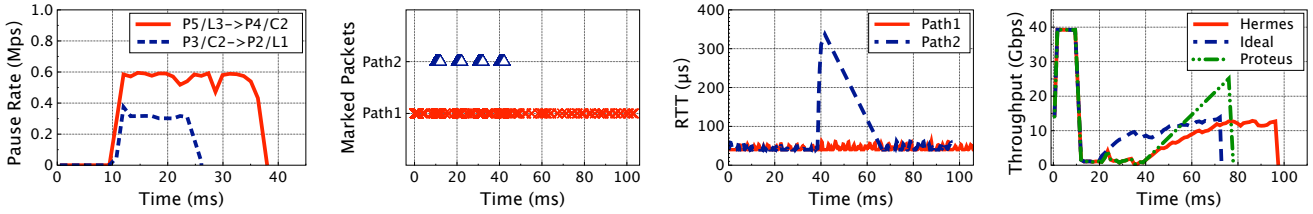
Traffic pattern: As shown in Fig. 1, there are two equal-cost paths between leaf switches $L1$ and $L3$, i.e., $Path1$ $\{P1/L1, C1, L3\}$ and $Path2$ $\{P2/L1, C2, L3\}$. The innocent flow f_1 (in blue) that is not responsible for congestion is sent from the sender $S1$ to the receiver $R1$. The bursty congested flows (in yellow) that really cause congestion are sent intermittently from the sender $S2$ to the receiver $R2$. Meanwhile, 14 hosts under $L3$ also send bursty flows at line rate to the same receiver $R2$. The background flows (in gray) are transmitting on $Path1$ from the source hosts under $L1$ to the destination hosts under $L3$, which makes the existing load balancing treat $Path1$ worse than $Path2$.

*1) **Link load cannot reflect PFC pausing correctly**:* The congestion-sensitive load balancing schemes such as CONGA [21] and HULA [22] detect path conditions by measuring link load. They fare poorly in lossless DCNs, because the path



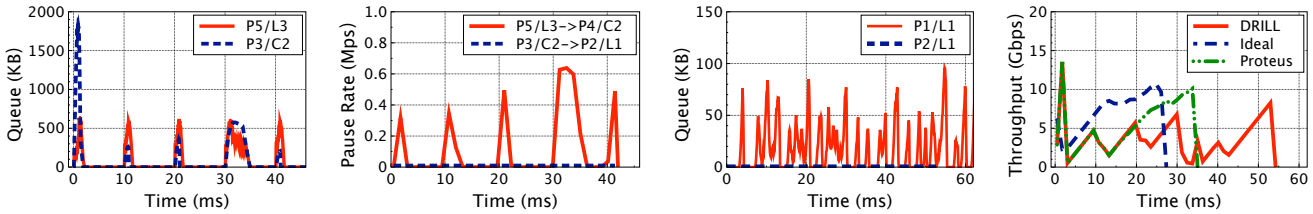
(a) Ingress queue length (b) PFC pause rate (c) Link utilization (d) Throughput of f_1

Fig. 2: Rerouting based on link load (CONGA [21]).



(a) PFC pause rate (b) Marked packets on two paths (c) Round trip time (d) Throughput of f_1

Fig. 3: Rerouting based on RTT and ECN (Hermes [23]).



(a) Ingress queue length (b) PFC pause rate (c) Egress queue length (d) Throughput of f_1

Fig. 4: Rerouting based on local queue (DRILL [24]).

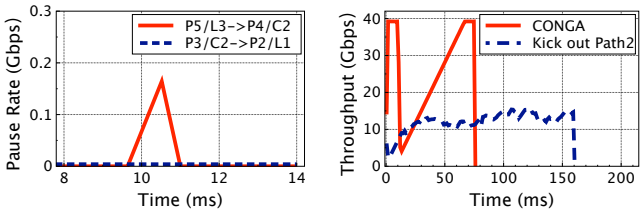
experiencing PFC pausing is actually under-utilized, a path with lower link load is not necessarily a better choice. So link load cannot indicate path congestion correctly and even mislead rerouting.

To quantify this impairment, we conduct simulations with CONGA. The victim flow f_1 is 20MB and bursty flow is 200KB. At time 1ms, 42 bursty flows from $S2$ and 588 bursty flows under $L3$ are sent to $R2$. In Fig. 2a and Fig. 2b, 5 bursty congestion occurred intermittently on $Path2$, and PFC is triggered on the ingress port $P5/L3$ and spread to $P3/C2$. Fig. 2c shows that the link utilization of $Path2$ is less than that of $Path1$ with real congestion. In Fig. 2d, CONGA always allocates the packets of f_1 to $Path2$ with the lower measured link utilization, resulting in HoL blocking. The ideal solution knows in advance that PFC will triggered multiple times on $Path2$ and does not choose $Path2$, achieving the lowest FCT.

*2) **End-to-end signals cannot detect PFC pausing timely**:* For the typical load balancing scheme using end-to-end congestion signals, such as Hermes [23], it leverages RTT and ECN to convey path congestion. Although these two signals capture the queueing delay, they cannot reflect PFC pausing in time due to the intrinsic control loop of at least one RTT, and even these signals blocked by PFC PAUSE can be slowly fed back to the end-hosts only after PFC resuming. When the congestion signals arrive at the source hosts, PFC may have just resumed, so a path with larger delay may be a good one.

To understand such impact, in this experiment, f_1 is 120MB and each bursty flow is 200KB. At time 10ms, 45 bursty flows from $S2$ and 635 bursty flows under $L3$ are continuously sent to $R2$. Fig. 3a shows that PFC PAUSE frames are sent from $P5/L3$ to $P4/C2$ continuously and spread from $P3/C2$ to $P2/L1$ during bursty congestion. Fig. 3b shows packets on $Path2$ are marked by ECN during PFC triggering, while packets on $Path1$ are marked continuously due to real congestion. In Fig. 3c, the RTT of $Path2$ is increased significantly due to PFC pausing. Thus, Hermes preferentially selects $Path2$ with small RTT and ECN at the beginning, and it forwards packets to $Path1$ once the RTT of $Path2$ is much larger than $Path1$ shown in Fig. 3d. However, such a late rerouting based on the staled signals damages performance. On the contrary, FCT with prescient rerouting in the ideal case is better than Hermes by up to 26%.

*3) **Local queue cannot sense remote PFC pausing**:* For a local queue-based scheme DRILL [24], it purely relies on local queue length to make forwarding decisions to deal with microbursts quickly. Since there is no any coordination among switches, it cannot sense the remote PFC pausing, potentially leading to serious HoL blocking at the downstream switches and even inability to reroute. The local queue-based switching is oblivious to remote PFC pausing, a local egress port with



(a) PFC pause rate (b) Throughput of f_1

Fig. 5: Kick out the path with PFC pausing.

smaller queue is not necessarily a better choice.

Next, to illustrate the above issue, we show a case study. At the beginning, f_1 with 20MB starts at line rate. At time 1ms, 26 bursty flows from $S2$ and 56 bursty flows under $L3$ with each flow size of 200KB are intermittently sent to $R2$, with a total of 5 bursts. In Fig. 4a and Fig. 4b, when each bursty traffic starts, the queue length of ingress port $P5/L3$ is increased and PFC PAUSE messages are sent from $P5/L3$ to pause the egress port $P4/C2$. In this process, since the queue length of $P3/C2$ does not exceed the PFC threshold, PFC pausing is not spread to the upstream switch $L1$. The queue length of $P2/L1$ is not increased and is always lower than that of $P1/L1$ as shown in Fig. 4c. Thus, based on the local queue to make routing decision, DRILL always chooses $P2/L1$ to forward packets of f_1 on $Path2$ $\{P2/L1, C2, L3\}$, resulting in f_1 suffering from PFC's HoL blocking and largest FCT, as shown in Fig. 4d. To illustrate the potential performance improvement without HoL blocking, we employ an ideal routing solution, which can predict PFC triggering on $Path2$ and forward packets of f_1 to $Path1$ without any HoL blocking. Fig. 4d shows the ideal solution reduces FCT by up to 49%.

C. Kicking Out the Path with PFC Pausing Directly is Unwise

Intuitively, as long as the path where PFC is triggered, it should be kicked out directly as an unavailable path. However, for a short PFC pausing, it is wise to still choose this path. To quantify this impact, we further conduct simulations with CONGA in the same experiment settings as in the above contrived example. The flow f_1 is 250MB and bursty flow is 200KB. At time 9ms, 2 bursty flows from $S2$ and 28 bursty flows from 14 end-hosts under $L3$ are sent to $R2$.

Fig. 5a shows the duration of PFC pausing is very short. In this experiment, CONGA always chooses $Path2$ with lower link utilization. Even if the flow f_1 experiences the transient PFC pausing and suffers from a short throughput degradation, it still obtains smaller FCT as shown in Fig. 5b. On the contrary, if $Path2$ is kicked out arbitrarily and packets are forwarded on $Path1$, FCT is increased by 53% due to the real congestion on $Path1$. Therefore, it is unreasonable to remove the path with PFC pausing directly from the available valid paths, but it is necessary to carefully decide whether to reroute according to the actual path conditions.

III. Design

Given the above explored problems, the limitations of prior individual congestion signals highlight the properties required

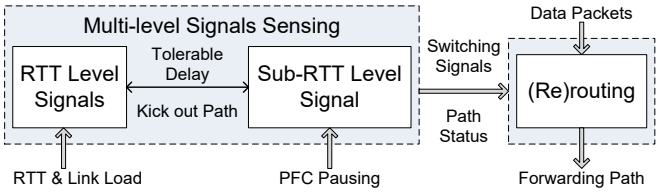


Fig. 6: Proteus overview.

for load balancing solutions in PFC-enabled DCNs. Therefore, we aim to achieve the following three design goals: (1) correctly reflect path states to guide load balancing decisions; (2) timely sense the duration of PFC pausing to reflect the hop-by-hop flow control; (3) flexibly reroute to balance traffic and avoid continuous PFC's HoL blocking.

To this end, we propose Proteus, a simple yet effective load balancer for PFC-enabled DCNs. The key point of Proteus is to make an initial load balancing decision based on the coarse-grained RTT level signals, and decide whether to reroute by using the fine-grained sub-RTT level signal when experiencing PFC pausing. For the short transient PFC pausing, Proteus will benefit more if it tolerates pausing delay without rerouting. On the contrary, for the long continuous PFC pausing, Proteus will decide to switch the path based on the idea of better-late-than-never to avoid further PFC's HoL blocking. Fig. 6 overviews Proteus, which mainly contains two modules including the sensing module and the rerouting module.

- **Multi-level signals sensing (§III-A):** Proteus first uses RTT level signals (i.e., RTT and link utilization) to correctly reflect path states and divide the paths into three categories including non-congested path, real congested path and the undetermined path with PFC pausing. Then Proteus uses sub-RTT level signal (i.e., cumulative sojourn time, CST) to rapidly reflect the hop-by-hop PFC pausing.
- **(Re)routing (§III-B):** Proteus makes load balancing decisions based on the perceived path status detected by the RTT level signals at the edge switches. When data packets transmitting on the initial optimal path experience PFC pausing, Proteus has opportunities to reroute according to the sub-RTT level signal to avoid the packets being suffered from continuous HoL blocking.

A. Multi-level Signals Sensing

RTT level signals: In PFC-enabled networks, in addition to the congested and non-congested path states, there is also an undetermined path state, which showing ON/OFF transmission pattern due to PFC pausing/resuming with under-utilized link. Although the congestion signals such as link utilization, queue length, RTT and ECN used seperately in the previous load balancing schemes provide good performance in lossy DCNs, they are insufficient to reflect different path states simultaneously when used alone in the lossless DCNs with PFC. For example, a path with link utilization less than 100% does not necessarily indicate non-congestion. It may be that PFC is triggered on the path, resulting in paused transmission. Similarly, a small RTT can directly reflect a non-congested path, while a large RTT does not necessarily mean that a path

TABLE II: Outcome of path states using RTT and link utilization.

RTT	Link Utilization	Possible Cause	Characterization
Low	<1	Under-utilized link	Non-congested path
Low	=1	Fully-utilized link with no queueing	
Moderate/High	<1	PFC pausing with queueing	Undetermined path
High	=1	Highly loaded link with queueing	Congested path

is congested. PFC may have resumed after a short pausing and the path is no longer congested. Fortunately, RTT and link utilization used together can nicely complement each other in PFC-enabled DCNs. Therefore, Proteus detects path conditions through the combination of RTT and link utilization to well indicate the aforementioned different path states.

Specifically, based on RTT and link utilization, parallel paths can be divided into the following three categories: (1) non-congested paths with small RTT less than RTT_{low}, which is set to be 20-40% times larger than one-way base RTT to ensure that the path is lightly loaded [23], and small or full link utilization; (2) undetermined paths with moderate or high RTT larger than RTT_{low} and small link utilization; (3) real congested paths with large RTT greater than RTT_{low} and full link utilization. Table II summarizes the outcome of path conditions and the reasons behind them.

However, in addition to the advantages of capturing three different path states correctly in the PFC-enabled networks, RTT and link utilization also have the disadvantage of stale information due to end-to-end feedback delay. Therefore, Proteus further leverages the sub-RTT signal to reflect the transient congestion timely as shown below.

Sub-RTT level signal: To react to the transient congestion timely, Proteus should make a rational decision whether to switch path when encountering PFC pausing on the initial optimal path selected based on the RTT level signals. The high-level idea is that if the current path delay is much less than other paths and PFC pausing is only transient, Proteus prefers to tolerate the delay caused by PFC pausing rather than switching paths to risk larger delay. In contrast, if the PFC pausing lasts for a long time, timely rerouting will bring more benefit to Proteus.

However, it is hard to accurately predict the duration of PFC pausing as we cannot know the future congestion caused by bursty traffic. But we can measure the elapsed time on the path, i.e., the cumulative sojourn time (CST), which refers to the cumulative time consumed by a packet transmitting on a path over multiple switches. Then we can determine whether it is necessary to reroute based on the idea of better-late-than-never to avoid continuous HoL blocking. Therefore, Proteus further tracks the CST of per-packet at sub-RTT timescale for rerouting to handle bursty PFC pausing timely. Specifically, if the CST is less than the tolerate delay, Proteus still chooses the initial optimal path. Otherwise, Proteus kicks out the current path with PFC pausing and triggers rerouting.

B. (Re)routing

Given multi-level signals sensing, how to balance load flexibly and deal with the burst PFC pausing rationally are still non-trivial. The challenges are as follows: First, in case of uncertain duration of PFC pausing, it is difficult to determine whether it is benefit to directly reroute or continue wait to tolerate queueing delay. Second, prior flowlet switching is to passively wait for the rerouting chance based on the fixed flowlet timeout and cannot flexibly balance load due to rare flowlets caused by rate smooth [15]. Third, the finest-grained packet switching is extremely easy to introduce disorder packets, while the simple go-back-N retransmission mechanism will significantly degrade the performance of applications in the PFC-enabled RDMA networks [54]. Proteus addresses the above challenges with careful (re)routing design by employing both RTT level and sub-RTT level signals.

Algorithm 1: Routing based on RTT level signals

```
   Input:
   t_pre, t_cur: The arrival time of previous & current packet;
   t_pd, u_link: The one-way delay & link load of a path;
1  for every packet do
2      Assume its corresponding flow is f and path is p;
3      if f is a new flow then
4          SelectPath(Active Paths Status);
5      else
6          {P'} = all paths with less delay than p;
7          /*∀p' ∈ {P'},0 < p.t_pd − p'.t_pd < f.t_cur − f.t_pre*/
8          if {P'} ≠ ∅ then
9              SelectPath(Paths{P'} Status);   /* Reroute */
10         else
11             p* = p;   /* Do not reroute */
12 Function SelectPath(Active Paths Status) is
13     {P'} = non-congested paths;
14     {P''} = undetermined paths occurred PFC;
15     {P'''} = congested paths;
16     if {P'} ≠ ∅ then p*=Argmin_{p∈{P'}}(p.t_pd) ;
17     else
18         if {P''} ≠ ∅ then p*=Argmax_{p∈{P''}}(p.u_link) ;
19         else
20             p* = Argmin_{p∈{P'''}}(p.t_pd);
21         end
22     end
23     return p* ;
24 end
```

Routing based on RTT level signals: Proteus selects the initial best forwarding path from the three types of paths (i.e., non-congested path, undetermined path, and real congested path) in order of priority based on the RTT and link utilization at the edge switches. The routing logic of Proteus based on RTT level signals is illustrated in Algorithm 1, which is triggered at edge switches for the cases of a new flow packet arriving or the current path is congested (lines 3-11). When Proteus makes a load balancing decision, it selects the best path from three path sets (lines 13-15) in turn. That is, Proteus first tries to select an available (p_{active}==1) non-congested

path with the minimum RTT (line 16). If it fails, Proteus chooses an available undetermined path with the maximum link utilization (line 18), which is likely resumed transmission quickly. The lowest priority choice of Proteus is the path with the minimum RTT among the remaining available real congested paths (line 20).

To preserve orderly transmission, the initial selected path should ensure that the current packet arrives at the receiver later than the previous packets with smaller sequence number belonging to the same flow. For non new flow packets, Proteus calculates the time interval between the current packet and the last forwarded one in the same flow, and then selects the rerouting path only in the set of paths with the path delay difference less than the time interval (lines 6-7). In this way, the current packet is rerouted to a better path with smaller delay to reach the receiver faster, but it will arrive in an orderly manner after the previous packets with smaller sequence number belonging to the same flow. Unlike existing schemes, there is no preset threshold.

The initial selected path ID is recorded in the packet header, and then the packet is forwarded to the designated path by using source routing [37]. Moreover, the packet also carries multiple alternative sub-optimal path IDs with the priority order for efficient and orderly rerouting at downstream switches, which will discuss later.

Algorithm 2: Rerouting based on sub-RTT level CST

```
   Input:
   t_td: Tolerable delay; t_enqueue: The enqueue time;
1  for every packet do
2      Assume its forwarding path is p;
3      Predict the minimum queueing delay t_qd;
4      if PFC pausing then
5          if t_td < t_qd then
6              Wait to be forwarded at time t_dequeue;
7              t_td = t_td - (t_dequeue - t_enqueue);
8          else
9              Trigger rerouting;
10             p_active = 0;   /* kick out path p */
11     return p_active
```

Rerouting based on sub-RTT level signal: Each packet has opportunity to be rerouted at the downstream switches based on sub-RTT level signal CST. When a packet encounters a long enough PFC pausing on the current path, Proteus reroutes the packet to the next alternative available path carried in the packet header at downstream switches. As a result, all rerouted packets are sent to the same path, so that it can preserve an orderly transmission. The Algorithm 2 illustrates the detailed process of sensing PFC pausing by using sub-RTT level signal CST. Specifically, each packet header carries a tolerable delay t_{td} spending on switches over the path p, which is the delay difference between the optimal and sub-optimal paths. After arriving the ingress port, the packet first predicts the minimum queuing delay t_{qd} based on the current destination egress port queue. Then, if t_{td} is less than t_{qd}, the packet waits to be forwarded. The tolerable delay t_{td} is updated by subtracting the sojourn time [38]–[40] at the current switch (lines 5-7), which is the time it takes from arriving at the switch to leaving the switch. Otherwise, if the packet's sojourn time t_{qd} exceeds t_{td}, Proteus triggers rerouting and kicks out the path p (P_{active}=0) (lines 9-10) until PFC resuming.

Theoretically, piggybacking enough path IDs can ensure rerouting to an available path, but it introduces more bytes overhead. To limit the extra overhead in the packet header, each packet in Proteus carries 3 alternative path IDs, which can effectively support rerouting with affordable cost in the evaluation (§IV). In the extreme cases, if there is no effective alternative path for rerouting, the packet will not switch path, which rarely occurs (§V-B).

C. Theoretical Analysis

Proteus makes rerouting decisions on the fly without knowing in advance how long the data transmission can be resumed after PFC pausing. We use the classical rent-or-buy problem[1] with online decision-making [41] as the context of Proteus.

To theoretically analyze the rerouting strategy of Proteus, we assume that the path delay of the initial optimal path selected according to RTT level signals is D_{po}, and the suboptimal path delay is D_{ps}. After the packets are forwarded on the initial optimal path, if they experience the bursty hop-by-hop PFC pausing due to transient congestion, and it is hard to predict how long PFC pausing will last, they need to decide whether and when to reroute. This rerouting problem is the same as rent-or-buy where rerouting is like buy, waiting on the initial optimal path with PFC pausing is like renting, and PFC resuming is like the last skiing. Therefore, the optimal strategy is: if the packets know the PFC pausing duration will exceed the delay difference between the initial optimal path and the suboptimal one, they should reroute at the beginning. If the packets know PFC pausing time is very short, they should just waiting on the initial optimal path to tolerate the pausing delay. But, what if the packets do not know? How long should the packets wait until they give up and decide to reroute?

We use the competitive ratio to discuss the quality of the online algorithm for Proteus. The competitive ratio of an online algorithm ALG is the ratio of the cost of ALG on the instance I (i.e., $ALG(I)$) to the minimum possible cost of ALG on the instance I (i.e., $OPT(I)$), that is, $\frac{ALG(I)}{OPT(I)}$.

As demonstrated in the above scenario, we only consider that PFC pausing will always end and PFC resuming frames will always be received. For such a deterministic algorithm when D_{ps} is larger than D_{po}, the algorithm better-late-than-never (i.e., wait until the packets realize they should have rerouted at the start, then reroute) is a nice strategy. Specifically, the packets wait for $D_{ps} - D_{po}$ on the initial optimal path at most, and then they decide to switch path. If PFC pausing is finished in less than $D_{ps} - D_{po}$, the competitive ratio is 1. Otherwise, $OPT(I)$ is D_{ps}, and the

[1]Suppose you are going to start skiing. You can either rent skis at a low price or buy them at a high price. Since you don't know how many times you will go skiing in the future, you just decide to rent at the start. After going skiing many times, you realize that the cost of renting is about to exceed that of buying. You wish you had bought right at the beginning.

competitive ratio is $\frac{D_{ps}-D_{po}+D_{ps}}{D_{ps}} = 2-\frac{D_{po}}{D_{ps}}$, which is less than 2. If the packets wait longer than better-late-than-never, the numerator of the competition ratio $ALG(I)$ increases, while the denominator $OPT(I)$ remains unchanged, thus the ratio is worse. Therefore, the algorithm better-late-than-never has the best competitive ratio in this case.

IV. Implementation

Proteus dataplane: We have implemented a Proteus prototype on a hardware programmable switch with the type of Wedge 100BF-32X by using programming protocol-independent packet processors (P4) language [42]. The packet processing ingress pipeline primarily contains a series of exact match-action tables and stateful ALUs to implement the logic of delay difference comparison, rerouting decision and forwarding port selection. The challenges of implementing Proteus on hardware P4 platform is to ensure that the limited resources are accessed exclusively in all stages of the pipeline.

Specifically, in the flow table, each entry consists of a 16-bit flow ID calculated by a cyclic redundancy check (CRC 16) hash function based on the unique five tuples, 24-bit timestamp for the last forwarded packet, 12-bit selected path ID [31], three 12-bit alternative path IDs and 2-bit path status. For each packet arriving at the switch, if the packet matches an existing flow ID, the delay difference between the packet interval and path delay is compared by using blackbox stateful ALUs. Then the packet takes action to select forwarding path according to the path status read from the flow table. The 12-bit selected path ID, three 12-bit alternative path IDs and 24-bit tolerable delay are inserted in the metadata, so the total byte overhead is 9 bytes, which is only 0.9% in a 1KB packet. If the sojourn time exceeds the tolerable delay, the packet reroutes and set the initial forwarding path to invalid with the path state value of 2-bit 00. Otherwise, the matching action for the packet is direct forwarding without rerouting.

Resource consumption: Table III shows the hardware resource usage of the packet processing pipeline at the programmable switch under four load balancing schemes. Proteus uses more match-action tables and stages in the pipeline to implement the logic of comparison and calculation. These operations require more stateful ALUs and SRAM, results in higher resource consumption such as match crossbar, gateway, SRAM and ALU instruction. In brief, compared with ECMP [43], DRILL [24] and CONGA [21], Proteus consumes more hardware resources due to fine-grained rationally rerouting. However, the limited resource consumption is acceptable compared with the benefits of load balancing.

Practical deployment: Proteus is deployed on each switch working for RoCE traffic with PFC and not affecting other non RoCE traffic. To reduce deployment overhead in the multi-tier topology, Proteus only maintains the path status of each end-to-end path specified by a unique path ID [44] at the edge switches. The delay and link utilization (the ratio of throughput to link speed) of the paths are measured at the destination edge switches over a certain period (e.g., RTT) and piggybacked by the packets passing through the reverse path [21].

TABLE III: Resource consumption of load balancing schemes.

Resource	ECMP	DRILL	CONGA	Proteus
Match Crossbar	2.41%	4.24%	6.6%	7.05%
Hash Bits	3.08%	4.05%	6.04%	6.34%
Gateway	1.39%	3.13%	6.96%	8.18%
SRAM	1.56%	3.74%	4.72%	5.17%
VLIW Actions	1.56%	2.52%	4.21%	4.93%
ALU Instruction	2.6%	5.16%	6.72%	8.4%

V. Evaluation

In this section, we evaluate Proteus using a combination of testbed experiments and large-scale NS-3 simulations to answer the following three questions:

- **How does Proteus perform in practice?** Testbed experiments (§V-A) show that Proteus can achieve low latency with varying traffic loads and link speeds. Specifically, Proteus outperforms ECMP, CONGA and DRILL in terms of AFCT by up to 3x, 65% and 30%, respectively. Moreover, Proteus achieves up to 1.8x AFCT for short flows compared with other schemes.
- **How effective is Proteus in large-scale DCNs?** In large-scale NS-3 simulations (§V-B), we show that Proteus generally achieves the lowest AFCT and tail latency. For example, compared to CONGA, DRILL, Hermes and MP-RDMA, Proteus reduces up to 35%, 31%, 28%, 22% and 46%, 42%, 34%, 29% for AFCT and 99^{th} percentile FCT under Web Search, respectively.
- **How robust is Proteus to network settings?** Deep dive experiments (§V-C) show that Proteus maintains persistent good performance with varying piggybacked alternative paths, and is robust to transport protocols.

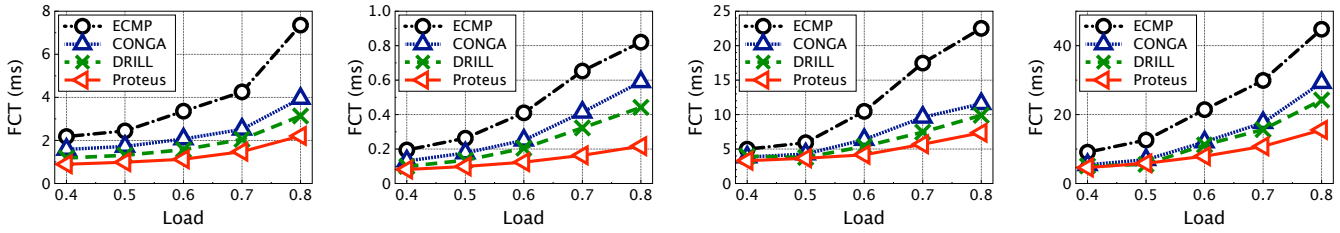
Baseline: We compare ECMP [43], CONGA [21], Hermes [23], DRILL [24] and MP-RDMA [15] with Proteus. DCQCN [11] is deployed as the default congestion control scheme, which is implemented by using DPDK 20.08 [45] on the testbed. All the parameters are set as recommended in the corresponding paper. We use reordering buffer to mask packet reordering[2] [23], [25].

Benchmark traffic: We use four widely-used realistic workloads including Web Server (WSv), Cache Follower (CF), Web Search (WSc) and Data Mining (DM) with heavy-tailed distribution in production datacenters [47]–[51]. The average flow sizes are 64KB, 701KB, 1.6MB and 7.41MB, respectively. Specifically, about 81%, 53%, 62% and 83% flows are less than 100KB, and 0%, 29%, 20% and 9% flows are larger than 1MB, respectively. All flows are generated between random sources and destinations following Poisson process.

A. Testbed Experiments

Testbed setup: We built a small testbed that consists of 8 servers (Dell PRECISION TOWER 5820 desktop) connected

[2]Although there is a trend to support reordering buffer in RDMA NICs [15], [54], most commodity RDMA NICs still use go-back-N retransmission scheme to deal with out-of-order packets. In this scenario, the baseline performance will be worse due to the severe penalty incurred by out-of-order packets, while Proteus can remain the same performance due to its order-preserving rerouting.



(a) Overall avg FCT (b) Small flow ($\leq$ 100KB) avg (c) Large flow ($>$ 1MB) avg (d) Overall 99^{th} percentile

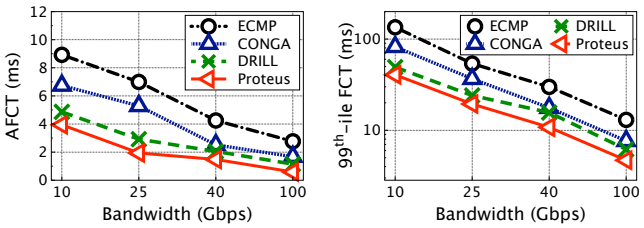
Fig. 7: Average and 99^{th} percentile flow completion time for Web Search.

to the Wedge 100BF-32X hardware P4 switches for a 2×8 leaf-spine topology as used in prior work [21], [23]. Each server has 10 cores Intel Xeon W-2255 CPU, 64GB memory and Mellanox ConnectX5 100GbE NICs. All servers run Ubuntu 20.04.1 with Linux version 5.4.0-42-generic. Each switch is equipped with 32 full duplex 100Gbps ports supporting DPDK and 22MB shared buffer. PFC is enabled with dynamic threshold. The default link capacity is 40Gbps.

Basic results: Fig. 7 shows the average and 99^{th} percentile FCT for Web Search workload. First, we find that Proteus performs increasingly better (2x-3x) FCT compared to ECMP [43] as the load increases as shown in Fig. 7a. This is as expected because ECMP is insensitive to path congestion and the hash collisions make PFC trigger more frequently, resulting in serious HoL blocking. Moreover, Proteus achieves up to 37% lower overall FCT at 0.7 load compared to CONGA. This is because CONGA incorrectly selects the path with low link utilization caused by PFC pausing, while Proteus can correctly choose the initial optimal path according to both RTT and link utilization signals. Also, we observe that Proteus outperforms DRILL by up to 30% in terms of AFCT with varying load. This is because DRILL cannot detect PFC pausing at the downstream switches and potentially selects worse path, while Proteus can further reroute based on the sub-RTT level signal cumulative sojourn time.

Another observation is that Proteus improves the performance of short flows less than 100KB by up to 1.8x, which is better than that of long flows larger than 1MB (by up to 51%) in Fig. 7b and Fig. 7c. The reason is that PFC pausing delay experienced by small flows accounts for a larger proportion in FCT. Proteus also achieves the lowest 99^{th} percentile FCT as shown in Fig. 7d. This is because Proteus not only performs PFC-sensitive routing based on RTT and link utilization, but also can further make rerouting decisions based on CST to avoid continuous blocking.

Impact of link speed: We repeat the above experiments under 70% load with varying link speeds. The results are shown in Fig. 8 (a) and Fig. 8 (b). First, we observe that the performance of CONGA deteriorates under the lower link speeds such as 10Gbps and 25Gbps. This is because PFC pausing time is longer, and the packets are mistakenly forwarded on the path with lower link utilization caused by PFC pausing, resulting in HoL blocking and large tail FCT. Moreover, we observe that Proteus achieves up to 20% and 23% lower average FCT



(a) Average FCT (b) 99^{th} percentile FCT

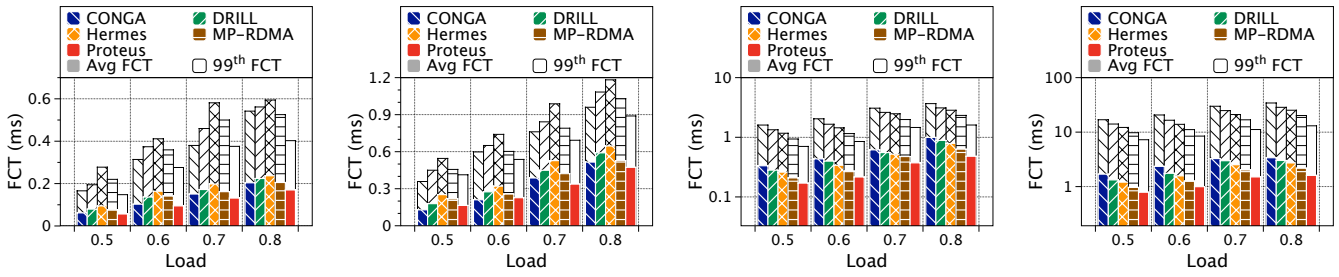
Fig. 8: FCT with varying link speeds.

and 99^{th} percentile FCT than DRILL at 10Gbps, respectively. The performance of DRILL under lower bandwidth is close to that of Proteus than CONGA. The reason is that when PFC pausing triggered at the downstream switches is passed back to the upstream switches in a hop-by-hop manner, DRILL can make correct routing decisions. In brief, Fig. 8 (a) and Fig. 8 (b) show that Proteus outperforms other schemes, because Proteus can choose appropriate forwarding path initially and rationally decide to reroute in the better-late-than-never way to avoid the negative impact of PFC pausing.

B. Large-scale NS-3 Simulations

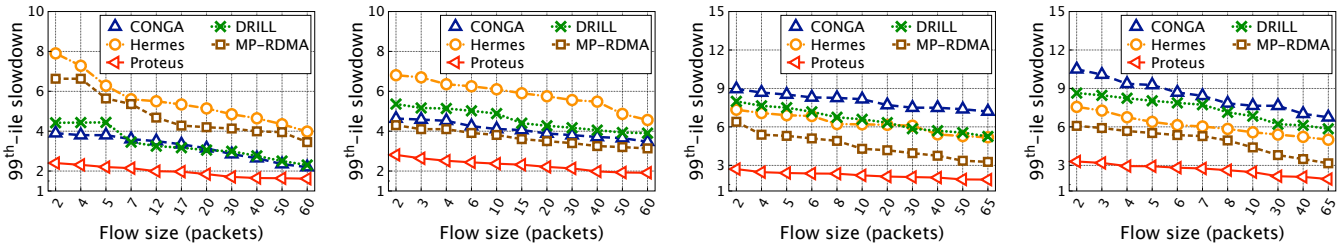
Simulation setup: Same as [23], we use a 8×8 leaf-spine topology with 128 hosts connected by 100Gbps links. The switch buffer size is set to 12MB and PFC is enabled with dynamic threshold as suggested in [11]. We further run large-scale NS-3 simulations to evaluate the performance of Proteus.
Overall: In Fig. 9, the results indicate that Proteus senses and reacts to path congestion effectively in PFC-enabled networks. Because after Proteus selects initial path based on RTT level signals, Proteus carefully considers whether to reroute according to sub-RTT level signal once experiencing burst PFC pausing, rather than rerouting arbitrarily or staying on the paused path all the time. For example, in Web Search workload, as the traffic load increases, Proteus outperforms CONGA, DRILL, Hermes and MP-RDMA on AFCT and 99^{th} percentile FCT by up to 35%, 31%, 28%, 22% and 46%, 42%, 34%, 29%, respectively.

Another observation is that CONGA and Hermes behave very differently across four workloads. The reason is that PFC pausing time is relatively short under the Web Server and Cache Follower workloads. Hermes uses stale end-to-end congestion signals to guide routing, resulting in many unnecessary rerouting and missing many good paths after PFC resuming. In Web Search and Data Mining workloads, if PFC pausing lasts



(a) Web Server (b) Cache Follower (c) Web Search (d) Data Mining

Fig. 9: Overall average and 99^{th} percentile FCT under realistic workloads with varying traffic load.



(a) Web Server (b) Cache Follower (c) Web Search (d) Data Mining

Fig. 10: The 99^{th} percentile slowdown as a function of flow size for small flows (0 - 100KB] under realistic workloads.

for a long time due to more congested large flows, the low link utilization misguides CONGA to always choose the paused path without timely rerouting, resulting in more HoL blocking flows and reordering packets. Although DRILL cannot sense the remote PFC pausing, it can reroute when PFC PAUSE frames are back pressured to the local switch. In comparison, Proteus can make rerouting decisions rationally and timely to adapt to different PFC pausing scenarios, the victim flows are not blocked across different workloads.

In comparison, Proteus can make rerouting decisions rationally and timely to adapt to different PFC pausing scenarios, the victim flows are not blocked across different workloads.

The performance improvement of Proteus is also shown in the motivation examples in Fig. 2 (d), Fig. 3 (d) and Fig. 4 (d) in Section II-B. Specifically, the FCT of the victim flow f_1 is reduced by 39%, 21% and 34% over CONGA, Hermes and DRILL, respectively.

Small flows: Next, we test the 99^{th} percentile FCT slowdown with varying flow size under 0.6 traffic load, as shown in Fig. 10. The X-axis are linear in the total number of small flows less than 100KB, and each tick is increased by 5%.

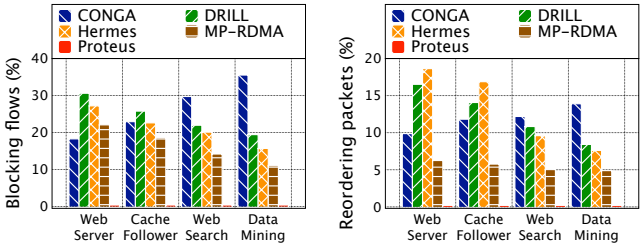
For small flows, as shown in Fig. 10, Proteus achieves greater performance improvement at the 99^{th} percentile FCT as expected, since the tail latency will be several times higher than the ideal FCT once small flows suffering from HoL blocking problem. For example, in Web Server workload, Proteus achieves up to 1.7x, 2x, 3.3x, 2.8x lower 99^{th} percentile FCT compared to CONGA, DRILL, Hermes and MP-RDMA, respectively. The reason is that the rerouting signals in the existing load balancing schemes cannot correctly or timely sense the PFC pausing before the small flows finish transmission, resulting in large tail FCT. Under bursty PFC pausing, Proteus further decides whether to reroute rationally based on the tolerable delay, which will avoid to be blocked earlier than the other schemes. In brief, Proteus provides more opportunities for small flows to reroute when they are blocked by PFC PAUSE, especially for tiny flows that would be completed in less than one RTT.

C. Proteus Deep Dive

We further inspect the effectiveness of Proteus in terms of HoL blocking flows and reordering packets under realistic workloads in the same large-scale simulation in Section V-B. Then we look into the sensitivity of Proteus to different parameter settings including RTT_{low} for path classification and the number of alternative paths for rerouting.

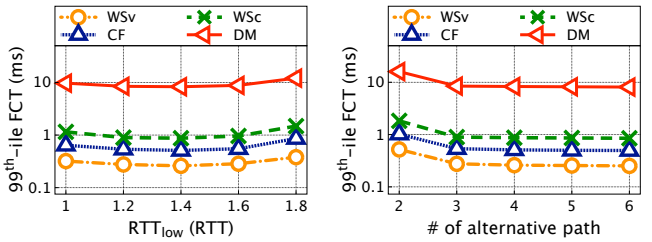
Effectiveness of rerouting: We first observe the ratios of HoL blocking flows and reordering packets under 60% traffic load. Fig. 11a shows that Proteus successfully avoids HoL blocking without victim flows under four workloads. However, the blocking flows of CONGA, DRILL, Hermes and MP-RDMA under Data Ming are as high as 35%, 19%, 15%, and 11%, respectively. This is because Proteus leverages multi-level routing signals to allocate traffic, it can make rational rerouting decisions based on the cumulative sojourn time. Proteus can not only prune unnecessary rerouting, but also make the better-late-than-never rerouting decisions.

We also observe that Proteus effectively avoid packet reordering in Fig. 11b. The reason is that Proteus can preserve orderly transmission both at RTT and sub-RTT level rerouting. For CONGA, the ratios of blocked and out-of-order packets increase as more paths with lower link utilization due to transient PFC pausing are mistakenly selected for rerouting. DRILL is heavily affected by PFC pausing because it makes rerouting decision at packet granularity and can only switch path after the local queueing buildup caused by PFC PAUSE.



(a) Ratio of blocking flows (b) Ratio of reordering packets

Fig. 11: Blocking flows and reordering packets.



(a) Sensitivity to RTT_{low} (b) Sensitivity to alternative paths

Fig. 12: Sensitivity to parameters.

Hermes cannot make accurate rerouting decisions only based on the stale RTT-level congestion signals. MP-RDMA introduces relatively few reordering packets since it can control the degree of out-of-order packets.

Sensitivity of parameter settings: We next investigate how different parameter settings affect the performance of Proteus. Fig. 12 (a) and Fig. 12 (b) show the sensitivity analysis for RTT_{low} and the number of alternative paths. First, we observe that Proteus obtains relatively stable 99^{th} percentile FCT when we set these two parameters around the recommended values. Another observation is that the tail FCT increases as RTT_{low} is set to a more conservative or aggressive value. This is because Proteus will miss some uncongested paths with smaller RTT_{low}, while Proteus will choose some congested paths with larger RTT_{low}. We also observed that Proteus achieves almost good performance, although the number of alternative paths increases from 3. The reason is that as long as there are paths where PFC does not trigger, traffic can be shifted to alternative paths to avoid the continuous blocking.

VI. Related Work

Congestion control: In recent years, several RDMA congestion control mechanisms [11], [12], [15], [19], [29]–[31], [52]–[54] have been proposed. QCN [52] quantizes congestion notification at link layer to control congestion for Ethernet. DCQCN [11] relies on ECN marking to alleviate congestion and reduce PFC triggering. TIMELY [29] and Swift [30] leverage RTT as the congestion signal to reduce queuing delay. MP-RDMA [15] proposes a multi-path ACK-clocking scheme to reduce memory footprint. HPCC [31] leverages in-network telemetry technology to detect congestion for rate adjusting. PCN [12] only regulates the rate of identified congested flows. TCD [14] detects congested flows for existing transports. BFC [53] designs a per-hop per-flow control to reduce HoL blocking. Dart [19] proposes a divide-and-specialize approach to reduce receiver and in-network congestion. The above efforts effectively reduce congestion, but they still cannot avoid PFC pausing especially under bursty scenario. IRN [54] explores loss recover scheme for RDMA without PFC.

Load balancing: To alleviate congestion by making full use of the overwhelming multiple parallel paths, there is a large body of load balancing schemes [21]–[26], [55]–[59] originally designed for lossy DCNs. CONGA [21] uses global congestion information to allocate flowlets among paths in a distributed manner. HULA [22] balances traffic load effectively by using link utilization to sense path congestion. DRILL [24] forwards packets purely based on the local queue length to alleviate micro-burst congestion. Hermes [23] leverages comprehensive sensing to detect path status and makes timely yet cautious rerouting to be resilient to various uncertainties. Clove [55] makes routing decision based on end-to-end signals such as RTT and ECN. LetFlow [26], Presto [25] and RPS [56] choose forwarding path randomly in a congestion insensitive manner. The paths selected by flowcell-based Presto and packet-based RPS are prone to PFC pausing, leading to a lot of out-of-order packets. TLB [57] adjusts the switching granularity dynamically for elephant flows. Hedera [58] and MicroTE [59] schedule flows by using network controller. However, these schemes do not work well in PFC-enabled networks due to unreliable congestion signals.

VII. Conclusion

This paper presented Proteus, a load balancing scheme for PFC-enabled networks, which is an extension of the previous workshop paper [20]. Proteus leverages RTT level signals (i.e., RTT and link utilization) to detect path conditions and divide paths into three categories (i.e., non-congested paths, undetermined paths and congested paths) to guide initial path selection. Then Proteus utilizes sub-RTT level signal (i.e., cumulative sojourn time) to make rerouting decisions rationally in a better-late-than-never way. In this way, Proteus can react to PFC pausing timely and alleviate head-of-line blocking. We have implemented Proteus with hardware programmable switches and evaluated it through testbed experiments and large-scale simulations. The test results show that Proteus outperforms CONGA, DRILL, Hermes and MP-RDMA on the average FCT and 99^{th} percentile FCT by up to 35%, 31%, 28%, 22% and 46%, 42%, 34%, 29%, respectively.

Acknowledgment

We thank the anonymous reviewers and our shepherd Prof. Eric Rozner for their constructive suggestions. This work is supported by the National Natural Science Foundation of China (62102046, 62072056, 62132022), the Natural Science Foundation of Hunan Province (2022JJ30618, 2020JJ2029, 2021JJ30867), the Hunan Provincial Key Research and Development Program (2022GK2019), the Scientific Research Fund of Hunan Provincial Education Department (22B0300), and the Hong Kong RGC TRS T41-603/20-R and GRF 16213621. Jiawei Huang and Kai Chen are the corresponding authors.

REFERENCES

[1] W. Bai, A. Agrawal, A. Bhagat, et al. Empowering Azure Storage with 100×100 RDMA. In Proc. USENIX NSDI, 2023.

[2] Y. Gao, Q. Li, L. Tang, et al. When Cloud Storage Meets RDMA. In Proc. USENIX NSDI, 2021.

[3] W. Cao, Y. Liu, Z. Cheng, et al. POLARDB Meets Computational Storage: Efficiently Support Analytical Workloads in Cloud-Native Relational Database. In Proc. USENIX FAST, 2020.

[4] A. Singhvi, A. Akella, D. Gibson, et al. 1RMA: Re-envisioning Remote Memory Access for Multi-tenant Datacenters. In Proc. ACM SIGCOMM, 2020.

[5] C. Hu, B. Liu, H. Zhao, et al. DISCO: Memory Efficient and Accurate Flow Statistics for Network Measurement. In Proc. IEEE ICDCS 2010.

[6] H. Li, Y. Zhang, Z. Zhang, et al. Ursa: Hybrid Block Storage for Cloud-Scale Virtual Disks. In Proc. ACM EuroSys 2019.

[7] W. Bai, K. Chen, S. Hu, K. Tan, Y. Xiong. Congestion Control for High-speed Extremely Shallow buffered Datacenter Networks. In Proc. ACM APNet 2017.

[8] Y. Wang, W. Wang, D. Liu, et al. Enabling Edge-Cloud Video Analytics for Robotic Applications. IEEE Transactions on Cloud Computing, 2023, 11, 1500-1513.

[9] M. Abadi, P. Barham, J. Chen, et al. TensorFlow: A System for Large-Scale Machine Learning. In Proc. USENIX OSDI, 2016.

[10] Y. Jiang, Y. Zhu, et al. A Unified Architecture for Accelerating Distributed DNN Training in Heterogeneous GPU/CPU Clusters. In Proc. USENIX OSDI, 2020.

[11] Y. Zhu, H. Eran, D. Firestone, et al. Congestion Control for Large-Scale RDMA Deployments. In Proc. ACM SIGCOMM, 2015.

[12] W. Cheng, K. Qian, W. Jiang, T. Zhang, F. Ren. Re-architecting Congestion Management in Lossless Ethernet. In Proc. USENIX NSDI, 2020.

[13] K. Qian, W. Cheng, T. Zhang, F. Ren. Gentle Flow Control: Avoiding Deadlock in Lossless Networks. In Proc. ACM SIGCOMM, 2019.

[14] Y. Zhang, Y. Liu, Q. Meng, F. Ren. Congestion Detection in Lossless Networks. In Proc. ACM SIGCOMM, 2021.

[15] Y. Lu, G. Chen, B. Li, et al. Multi-Path Transport for RDMA in Datacenters. In Proc. USENIX NSDI, 2018.

[16] C. Guo, H. Wu, Z. Deng, et al. RDMA over Commodity Ethernet at Scale. In Proc. ACM SIGCOMM, 2016.

[17] H. Lim, W. Bai, Y. Zhu, Y. Jung, D. Han. Towards Timeout-less Transport in Commodity Datacenter Networks. In Proc. ACM EuroSys, 2021.

[18] C. Tian, B. Li, L. Qin, et al. P-PFC: Reducing Tail Latency with Predictive PFC in Lossless Data Center Networks. IEEE TPDS, 31(6), 2020, pp.1447–1459.

[19] J. Xue, M. U. Chaudhry, B. Vamanan, T. Vijaykumar, M. Thottethodi, Dart: Divide and Specialize for Fast Response to Congestion in RDMA-based Datacenter Networks. IEEE/ACM ToN, 28 (1), 2020, pp.322–335.

[20] J. Hu,C. Zeng, Z. Wang, H. Xu, J. Huang, K. Chen. Load Balancing in PFC-Enabled Datacenter Networks. In Proc. ACM APNet, 2022.

[21] M. Alizadeh, T. Edsall, S. Dharmapurikar, et al. CONGA: Distributed Congestion-aware Load Balancing for Datacenters. In Proc. ACM SIGCOMM, 2014.

[22] N. Katta, M. Hira, C. Kim, A. Sivaraman, J. Rexford. HULA: Scalable Load Balancing Using Programmable Data Planes. In Proc. ACM SOSR, 2016.

[23] H. Zhang, J. Zhang, et al. Resilient Datacenter Load Balancing in the Wild. In Proc. ACM SIGCOMM, 2017.

[24] S. Ghorbani, Z. Yang, P. B. Godfrey, et al. DRILL: Micro Load Balancing for Low-Latency Data Center Networks. In Proc. ACM SIGCOMM, 2017.

[25] K. He, E. Rozner, K. Agarwal, et al. Presto: Edge-based Load Balancing for Fast Datacenter Networks. In Proc. ACM SIGCOMM, 2015.

[26] E. Vanini, R. Pan, M. Alizadeh, et al. Let it Flow: Resilient Asymmetric Load Balancing with Flowlet Switching. In Proc. USENIX NSDI, 2017.

[27] IEEE 802.1 Qbb - Priority-based Flow Control. https://1.ieee802.org/dcb/802-1qbb/.

[28] S. Hu, Y. Zhu, P. Cheng, et al. Tagger: Practical PFC Deadlock Prevention in Data Center Networks. In Proc. ACM CoNEXT, 2017.

[29] R. Mittal, V. T. Lam, N. Dukkipati, et al. Timely: RTT-based Congestion Control for the Datacenter. In Proc. ACM SIGCOMM, 2015.

[30] G. Kumar, N. Dukkipati, K. Jang, et al. Swift: Delay is Simple and Effective for Congestion Control in the Datacenter. In Proc. ACM SIGCOMM, 2020.

[31] Y. Li, R. Miao, H. H. Liu, et al. HPCC: High Precision Congestion Control. In Proc. ACM SIGCOMM, 2019.

[32] J. Zheng, Z. Du, Z. Zha, et al. Learning to Configure Converters in Hybrid Switching Data Center Networks. IEEE/ACM Transactions on Networking (TNET), in press.

[33] W. Wei, H. Gu, K. Wang, et al. Multi-Dimensional Resource Allocation in Distributed Data Centers Using Deep Reinforcement Learning. IEEE Transactions on Network and Service Management, 2023, 20(2),1817-1829.

[34] S. Hu, W. Bai, G. Zeng, et al. Aeolus: A Building Block for Proactive Transport in Datacenters. In Proc. ACM SIGCOMM, 2020.

[35] G. Zeng, W. Bai, G. Chen, K. Chen, D. Han, Y. Zhu, L. Cui. Congestion control for cross-datacenter networks. In Proc. IEEE ICNP 2019.

[36] W. Bai, S. Hu, K. Chen, K. Tan, Y. Xiong. One more config is enough: Saving (DC) TCP for high-speed extremely shallow-buffered datacenters. IEEE/ACM Transactions on Networking, 2020, 29(2), 489-502.

[37] P. Wang, G. Trimponias, H. Xu, Y. Geng. Luopan: Sampling-based Load Balancing in Data Center Networks, IEEE TPDS, 30(1), 2018, pp.133–145.

[38] K. Nichols, V. Jacobson. Controlling Queue Delay, Communications of the ACM, 55(7), 2012, pp.42–50.

[39] W. Bai, K. Chen, et al. Enabling ECN over Generic Packet Scheduling. In Proc. ACM CoNEXT, 2016.

[40] J. Zhang, W. Bai, K. Chen. Enabling ECN for Datacenter Networks with RTT Variations. In Proc. ACM CoNEXT, 2019.

[41] A. R. Karlin, M. S. Manasse, et al. Competitive Snoopy Caching. Algorithmica, 3(1), 1988, pp.79–119.

[42] P. Bosshart, D. Daly, G. Gibb, et al. P4: Programming Protocol-independent Packet Processors. In Proc. ACM SIGCOMM Computer Communication Review, 44(3):87-95, 2014.

[43] C. Hopps. Analysis of an Equal-cost Multipath Algorithm. RFC 2992, Internet Engineering Task Force. 2000.

[44] S. Hu, K. Chen, H. Wu, et al. Explicit Path Control in Commodity Data Centers: Design and Applications. In Proc. USENIX NSDI, 2015.

[45] DPDK Plane Development Kit, Intel DPDK, 2019.

[46] B. Montazeri, Y. Li, M. Alizadeh, and J. Ousterhout. Homa: A receiver-driven low-latency transport protocol using network priorities. In Proc. ACM SIGCOMM, 2018.

[47] J. Hu, J. Huang, Z. Li, et al. RPO: Receiver-driven Transport Protocol Using Opportunistic Transmission in Data Center. In Proc. IEEE ICNP, 2021.

[48] Z. Li, W. Bai, K. Chen, et al. Rate-aware flow scheduling for commodity data center networks. In Proc. IEEE INFOCOM 2017.

[49] Y. Zhao, Y. Huang, K. Chen, et al. Joint VM placement and topology optimization for traffic scalability in dynamic datacenter networks. Computer Networks, 2015, 80, 109-123.

[50] C. Hu, B. Liu, H. Zhao, et al. Discount counting for fast flow statistics on flow size and flow volume. IEEE/ACM Transactions on Networking, 2014, 22 (3), 970-981.

[51] W. Bai, L. Chen, K. Chen, D. Han, C. Tian, and H. Wang. Information-agnostic Flow Scheduling for Commodity Data Centers. In Proc. USENIX NSDI, 2015.

[52] IEEE. 802.1Qau – Congestion Notification. http://www.ieee802.org/1/pages/802.1au.html.

[53] P. Goyal, P. Shah, K. Zhao, et al. Backpressure Flow Control. In Proc. USENIX NSDI, 2022.

[54] R. Mittal, A. Shpiner, A. Panda, et al. Revisiting Network Support for RDMA. In Proc. ACM SIGCOMM, 2018.

[55] N. Katta, A. Ghag, M. Hira, et al. Clove: Congestion-aware Load Balancing at the Virtual Edge. In Proc. ACM CoNEXT, 2017.

[56] A. Dixit, P. Prakash, Y. C. Hu, R. R. Kompella. On the Impact of Packet Spraying in Data Center Networks. In Proc. IEEE INFOCOM, 2013.

[57] J. Hu, J. Huang, W. Lyu, et al. Adjusting Switching Granularity of Load Balancing for Heterogeneous Datacenter Traffic. IEEE/ACM ToN, 29(5), 2021, pp.2367–2384.

[58] M. Al-Fares, S. Radhakrishnan, B. Raghavan, et al. Hedera: Dynamic Flow Scheduling for Data Center Networks. In Proc. USENIX NSDI, 2010.

[59] T. Benson, A. Anand, A. Akella, M. Zhang. Microte: Fine Grained Traffic Engineering for Data Centers. In Proc. ACM CoNEXT, 2011.