FLASHPASS: Proactive Congestion Control for Shallow-buffered WAN

Gaoxiong Zeng[1], Jianxin Qiu[1], Yifei Yuan[2], Hongqiang Liu[2], Kai Chen[1]
[1]*iSING Lab@Hong Kong University of Science and Technology* [2]*Alibaba*

***Abstract*—In recent years, large enterprises (e.g., Google, Alibaba, etc.) have been building and deploying their wide-area routers based on shallow-buffered switching chips. However, with legacy reactive transport (e.g., TCP Cubic), shallow buffer can easily get overwhelmed by large BDP wide-area traffic, leading to high packet losses and degraded throughput. To address it, we ask: can we design a transport to simultaneously achieve high throughput and low loss for shallow-buffered WAN?**

We answer this question affirmatively by employing proactive congestion control (PCC). However, two issues exist for existing PCC to work on WAN. Firstly, wide-area traffics have diverse RTTs, leading to what we called imperfect scheduling issue (e.g., data crash in time). Secondly, there is one RTT delay for credits to trigger data sending, which may degrade network performance. Therefore, we propose a novel PCC design - FLASHPASS. To address the first issue, FLASHPASS adopts sender-driven emulation process with send time calibration to avoid the data packet crash. To address the second issue, FLASHPASS enables early data transmission in the starting phase, and incorporates an over-provisioning with selective dropping mechanism for efficient credit allocation in the finishing phase. Our evaluation with production workload demonstrates that FLASHPASS reduces the overall flow completion times of TCP Cubic and ExpressPass by up to 32% and 11.4%, and the 99-th tail completion times of small flows by up to 49.5% and 38%, respectively.

I. INTRODUCTION

With the prevalence of the geo-distributed web applications and services, wide-area network (WAN) is becoming an increasingly important cloud infrastructure [1]–[5]. For example, Google [3] reveals that its inter-datacenter wide-area traffic has been growing exponentially with a doubling of every 9 months in recent 5 years. This pushes the WAN facility to evolve much faster than the rest of its infrastructure components.

To scale the wide-area network cost-effectively and flexibly, large enterprises such as Google and Alibaba have been building and deploying their customized wide-area routers based on shallow-buffered commodity switching chips [1], [6]–[9] (§II-A). However, conventional wisdom [10] dictates that buffer size of one bandwidth-delay product (BDP) is required to achieve full link utilization in the worst (synchronized) case. Thus, the cheap shallow-buffered WAN gear imposes stringent requirements on the underlying transport protocols.

We revisit the buffer sizing problem with the newly evolved TCP-style reactive congestion control (RCC) [11]–[15] algorithms, and find that shallow buffer can easily get overwhelmed by the wide-area traffic (§II-B). Specifically, running legacy reactive transport protocols over shallow-buffered WAN may lead to either high packet losses or degraded throughput

978-1-6654-4131-5/21/$31.00 © 2021 IEEE

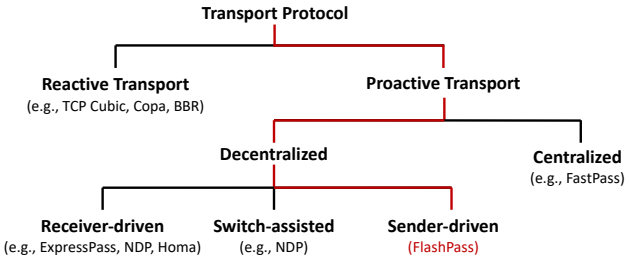


Fig. 1: Design space of transport protocols.

or both. To mitigate these problems, current practice seeks help from global traffic engineering [1]–[3], end-host rate limiting [4], or traffic scheduling with differentiated services.

Instead, we ask the question: can we design a transport to simultaneously achieve low loss rate and high throughput under shallow-buffered WAN? Inspired by the emerging proactive congestion control (PCC) [18]–[23] design (Figure 1), we answer this question affirmatively by taking the initiative to extend the PCC idea for shallow-buffered WAN.

However, while PCC has been proven to work well in datacenter network (DCN), we find several practicality issues to make it work on WAN (§II-C). First of all, centralized PCC [18] and switch-assisted PCC [19] protocols impose high requirements on network facilities (e.g., cutting payload [24]) and hence are either unscalable or impractical. Besides, some receiver-driven protocols like Homa [21] are based on the assumption of single bottleneck between the top-of-rack (ToR) switch and receiver, which does not hold on WAN.

While credit-emulated PCC like ExpressPass [20] seems to work, we find it may suffer from efficiency issues on WAN. Firstly, unlike homogeneous DCN, wide-area traffics have much diverse RTTs. The well scheduled credits on the reverse path may still trigger data packet crush on the data forward path due to different turn-around times. Secondly, there is one RTT delay for credits to trigger data sending for both starting phase (to start data sending quicker) and finishing phase (to stop credit generation in time) in PCC. Such an overhead is prohibitive given much higher RTT on WAN.

Therefore, we propose a novel PCC solution - FLASHPASS to address these challenges (§III). Firstly, to address the imperfect credit scheduling issue, FLASHPASS leverages a sender-driven emulation process together with send time calibration. Unlike receiver-driven protocols, sender-driven FLASHPASS can exactly rehearse the future data sending in the same direction on the emulation network. With the addition of the timestamp information on the emulation and credit packets,

TABLE I: Buffer size for commodity switching chips.

Switching Chips	BCM Trident+	BCM Trident2	BCM Trident3	BCM Trident4	BF Tofino	BF Tofino2
Capacity (ports × BW)	48 × 10Gbps	32 × 40Gbps	32 × 100Gbps	32 × 400Gbps	64 × 100Gbps	32 × 400Gbps
Total Buffer	9MB	12MB	32MB	132MB	22MB	64MB
Buffer per port	192KB	384KB	1MB	4.125MB	344KB	2MB
Buffer per port per Gbps	19.2KB	9.6KB	10.2KB	10.56KB	3.44KB	5.12KB

[a]BCM is short for Broadcom. BF is short for Barefoot (acquired by Intel in 2019). They are two dominant switching chip manufacturers.

FLASHPASS can strictly schedule data sending in the time space. Secondly, to mitigate the impact of one RTT delay for credits to trigger data sending, FLASHPASS leverages Aeolus [23] to enable early data transmission in the starting phase, and further incorporates a credit over-provisioning mechanism together with a selective dropping discipline for efficient credit or bandwidth allocation in the flow finishing phase.

We measured the realistic workload on a production wide-area network of Alibaba, and experimented FLASHPASS with NS2 simulation to demonstrate its superior performance (§IV). In static workload experiments, FLASHPASS achieves near full throughput (9.14Gbps) with zero packet loss persistently across various settings under the shallow-buffered WAN. The throughput is up to 55.9% higher than that of TCP Cubic. While comparing to ExpressPass, FLASHPASS achieves the similar throughput with zero packet loss rate (up to 0.12% losses for ExpressPass). In realistic dynamic workload experiments, FLASHPASS (with Aeolus enhancement) reduces the overall flow completion times of TCP Cubic and ExpressPass (also with Aeolus enhancement) by up to 32% and 11.4%; and the reduction of small flow 99-th tail completion times can get up to 49.5% and 38%, respectively.

We also presented a practical deployment analysis (§V). Specifically, multiple queues and ECN marking/dropping should be configured across the WAN (e.g., to coexist with legacy TCP traffic), and an efficient transport implementation is required. However, building a fully functional prototype is beyond the scope of this paper. Our hope is that the design, analysis, and extensive simulations conducted in this paper will pave the way for the next step of prototyping and deployment.

II. BACKGROUND AND MOTIVATION

A. Shallow-buffered Network

The last decade has witnessed an exponential growth of web applications and services (e.g., web search, cloud computing, social networking, etc.). This drives the large Internet companies (e.g., Google [25], Microsoft [26], Facebook [27], and Alibaba [28], etc.) to build the modern data centers (DCs) at an unforeseen speed and scale across the globe. With the ever-increasing communication demand, traditional network infrastructure built with commercial switches [29] fall short to meet the scale, management, and cost requirements.

To address it, inspired by the then-emerging merchant switching silicon industry [6], large enterprises start to build and deploy their own customized networking hardware both on WAN [1], [8] and DCN [6], [7]. However, while the cutting-edge merchant silicon provides the highest bandwidth density in a cost effective way, the shallow buffer (Table I [9]) shipped with it can degrade the network performance to a great extent. For example, Google has reported its experience of high packet losses on the shallow-buffered WAN [1] and DCN [7].

The buffer pressure is especially high for WAN communication. Conventional wisdom on buffer sizing problem [10] dictates that one bandwidth-delay product (BDP) buffering is required to achieve full link utilization in the worst case (i.e., with synchronized flows). However, the commodity switching chips provide shallow buffer of less than 20KB per port per Gbps according to Table I. That is even lower than 0.1% of WAN BDP (25MB per Gbps assuming 200ms RTT). Thus, it is extremely challenging to deliver low loss rate and high throughput simultaneously on shallow-buffered WAN.

B. Reactive congestion control (RCC) is insufficient

We revisit the buffer sizing [10] problem, and see if the state-of-art reactive congestion control (RCC) protocols [11]–[15] perform well for shallow-buffered WAN. The theoretical analysis is summarized in Table II:

- **TCP NewReno** [11] is the seminal loss-based congestion control algorithm. It follows the additive-increase and multiplicative-decrease (AIMD) control rule. The conventional buffer sizing theory [10] indicates one BDP buffer is required for full link utilization.
- **TCP Cubic** [12] is loss-based congestion control and enabled by default in Linux system. It increases window size based on a cubic function of time and decreases multiplicatively by a fraction of β=0.2 by default on loss. The resulting buffer requirement is BDP/4.
- **TCP Vegas** [13] reacts to both delay and loss signal. It applies additive-increase and additive-decrease (AIAD) control rule based on delay signal to control the lower and upper bound of excessive packets in flight, and also performs multiplicative decrease on loss signal. The buffer requirement is 5×flow# packets.
- **Copa** [14] is based on delay signal. It adjusts sending rate towards $1/(\delta d_q)$ by AIAD control rule, where d_q is the measured queueing delay. With default $\delta = 0.5$, the buffer requirement is 5×flow# packets.
- **BBR** [15] is model-based congestion control. It tries to drive the transport to the theoretical optimal point [30] based on accurate bandwidth and RTT estimation. BBR bounds the inflight packets to cwnd_gain×BDP by default. However, accurate bandwidth estimation is difficult to achieve, often leading to high buffering in practice.

Based on the analysis, we find that the reactive congestion control protocols all require non-negligible buffering for high throughput and low loss rate. Even worse, the buffer requirement is unscalable (in proportion to either network capacity or flow number) as network evolves and traffic demand grows.

TABLE II: Reactive congestion control (RCC) protocols for WAN.

Protocol	Signal	Algorithm	Buffer requirement for high throughput and low loss
TCP NewReno [11]	Loss	AIMD	β / (1 - β) × BDP = BDP (β = 0.5)
TCP Cubic [12]	Loss	AIMD	β / (1 - β) × BDP = BDP/4 (β = 0.2)
TCP Vegas [13]	Delay & loss	AIAD (MD on loss)	(β + 1) × n = 5n pkts (n = flow#, α = 2 pkts, β = 4 pkts)
Copa [14]	Delay only	AIAD	2.5n / δ = 5n pkts (n = flow#, δ = 0.5 / pkt)
BBR [15]	No direct signal	Not incremental	(cwnd_gain - 1) × BDP in Probe_BW phase (cwnd_gain = 2)

[a]Copa and BBR are both insensitive to loss signal, i.e., they do not throttle their sending rates during fast loss recovery.

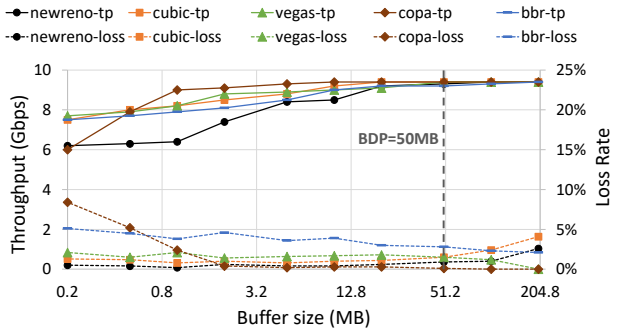


Fig. 2: Performance of reactive transports under shallow buffer. The solid lines indicate throughput (or tp) on the left y-axis, and the dash lines indicate loss rate on the right y-axis.

Note that this problem is fundamental to RCC protocols, because they can detect congestion only after the formation of queues and require one RTT delay before taking reaction[1]

We run NS3 simulation to illustrate the performance degradation of reactive protocols under shallow buffer. We generate 200 parallel flows from different senders to single receiver sharing the same 10Gbps bottleneck link with 40ms RTT (thus BDP=50MB) for 5 seconds. We experiment buffer size of 0.2, 0.5, 1, 2, 5, 10, 20, 50, 100, and 200 MB. Commodity switching chips should have shallow buffer of no more than 0.2MB per port in 10Gbps network (according to Table I). We enable selective acknowledgement (SACK) for efficient loss detection and retransmission.

Experimental results are shown in Figure 2. We find that the total throughput drops by 18%-37% as buffer decreases to the shallow size (i.e., 200KB under 10Gbps). The loss rate also increases dramatically for some protocols (e.g., 8.4% for Copa and 5.1% for BBR). We observe significant losses at the end of slow start (time at ∼0.5 second). This is because in the slow start phase, window sizes are doubling every RTT, resulting in packet losses of roughly half a window size at the end. Therefore, we conclude that with RCC protocols, shallow buffer can easily get overwhelmed by large BDP wide-area traffic, leading to high packet losses and degraded throughput.

C. Proactive congestion control (PCC) as a solution

Inspired by the emerging proactive congestion control (PCC) [18]–[23] design (Figure 1), we now explore the possibility of employing PCC protocols for shallow-buffered WAN. Unlike RCC that uses a "try and backoff" approach, PCC operates in a "request and allocation" style. The key

[1]We do not consider RCC protocols using advanced switch features such as in-band network telemetry (INT) that are often unavailable on WAN (§VI).

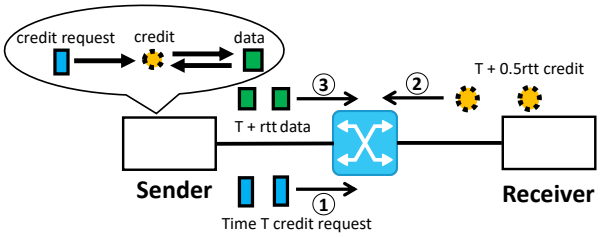


Fig. 3: Receiver-driven proactive congestion control (PCC).

conceptual idea is to explicitly allocate the bandwidth of bottleneck link(s) among active fows and proactively prevent congestion. As a result, ultra-low buffer occupancy and (near) zero packet loss can be achieved. Furthermore, PCC reaches peak rate in roughly one RTT, avoiding the long convergence time and high losses of RCC slow start.

However, as most of existing PCC protocols are designed for DCN, many of them cannot work practically on WAN. Centralized PCC [18] and switch-assisted PCC [19] protocols impose high requirements on network facilities (e.g., cutting payload [24]) and hence are either unscalable or impractical. Some receiver-driven protocols like Homa [21] adopt simple credit scheduling on the receiver side, assuming that there is single bottleneck link between the top-of-rack (ToR) switch and receiver. However, this assumption does not hold on WAN.

There are other receiver-driven protocols like ExpressPass [20] that leverage sophisticated credit emulation process on a separate queue for credit allocation. Unlike simple scheduling, emulation-based approach works for core-congested network (more practical in WAN). Figure 3 shows an overview of receiver-driven credit-emulated PCC protocols. Specifically, they generate credit packets on the reverse rate-limited path to emulate data sending. Each minimum-sized credit packet (e.g., 84B) passing through the network triggers the sender to transmit a MTU-sized data packet (e.g., 1538B). Thus, the credit queue is rate-limited to 84/(84+1538) ≈ 5% of the link capacity, and the remaining 95% is used for transmitting data packets. This makes them possible to work on WAN without assuming non-congested network core.

Therefore, we select ExpressPass [20] as our baseline PCC design for WAN. We validate its superior performance over RCC protocols following the same static workload experiment in §II-B with the open-sourced NS2 simulation code [31]. The results show that, ExpressPass achieves 9.5Gbps throughput and zero packet loss when running static workload with same RTTs on WAN (see more results in Figure 10).

However, we find that existing PCC protocols suffer from several efficiency issues that are particularly serious on WAN with different RTTs, especially under dynamic workload.

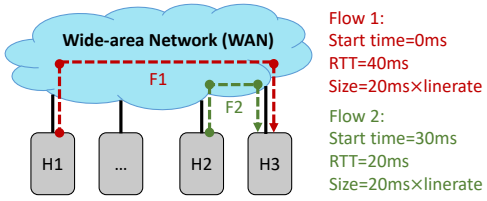


Fig. 4: Competing flows have different RTTs on WAN.

TABLE III: Data crush of receiver-driven PCC (Figure 4).

	Start time	Credit send	Data send	Data arrive
Flow 1	0ms	20-40ms	40-60ms	~60-80ms
Flow 2	30ms	40-60ms	50-70ms	~60-80ms

[a]Simultaneous data arrival leads to high buffering or packet loss.

Problem 1: Unlike homogeneous DCN, wide-area traffics have much diverse RTTs (not as simple as cases in §II-B). The well scheduled credit on the reverse path may still trigger data packet crush or bandwidth under-utilization on the data forward path due to different turn-around times of flows.

We illustrate the problem with a case shown in Figure 4. There are two flows from host H1 and host H2 to host H3, competing on the same bottleneck link that connects down to H3. The RTTs are 40ms and 20ms, respectively. Both have a traffic demand size of 20ms×linerate. Table IV shows the running process of receiver-driven PCC protocols. Flow 1 from H1 starts at 0ms, i.e., the credit request send time. After half a RTT, i.e., at time 20ms, the credit request reaches the receiver side H3 and triggers the credit generation. The credit generation lasts for 20ms and stops at 40ms, which is based on the traffic demand size. Flow 2 from H2 starts at 30ms and triggers credit generation from H3 during time 40-60ms. Thus, the credit generation of Flow 1 and Flow 2 are well interleaved. All credits then pass through the network successfully. They trigger out the real data sending, and the data arrival time are approximately 1 RTT after the credit send time. Ideally (implicitly assuming the same RTT for different flows in DCN), we expect the data packets to get through the network without crush with each other. However, in this case, giving different RTTs for Flow 1 and Flow 2 on WAN, we find the data arrival times of two flows are the same, i.e., 60-80ms. This leads to severe congestion, causing to large queueing and packet losses in the shallow-buffered WAN.

In theory, maximum queueing Q_{max} can be calculated as:

$$Q_{max} = \Delta RTT \times linerate \tag{1}$$

where ΔRTT is the maximum RTT difference among flows.

We run simulation under the Figure 4 scenario with 10Gbps links. We set a very large buffer size in the bottleneck link down to the host H3. We set initial credit rate to maximum and terminate credit generation as soon as the flow demand is reached. Results show that the maximum queueing reaches 8532 1.5KB-MTU-sized packets or 12.5MB, which exactly matches our theoretical analysis (i.e., Equation 1). This will lead to severe packet losses on the shalow-buffered WAN. While in another simulation, we change the RTT of Flow 2 to be 40ms (the same RTT as Flow 1) and vary the start time to be 0ms, 10ms, 20ms or 30ms; and repeat the same experiment.

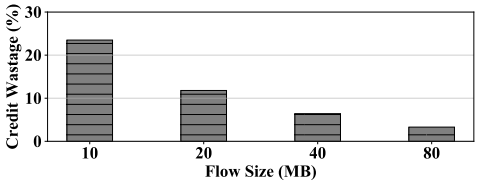


Fig. 5: Credit wastage of ExpressPass.

We find that the maximum queueing length drops dramatically to no more than 6 packets. This explains why ExpressPass does not work well on WAN with large RTT difference, even if it may work for homogeneous network such as DCN.

Problem 2: There is one RTT delay for credits to trigger data sending. This impacts both starting phase (to start data sending quicker) and finishing phase (to stop credit generation in time) of a flow in PCC. Such an overhead is prohibitive given large RTT on WAN. While we find that recent work (e.g., Aeolus [23]) has addressed the problem of starting phase and can be extended on WAN similarly, the problem of finishing phase remains unsolved.

For the flow finishing phase, the delayed data sending may either lead to low network utilization with default aggressive credit generation, or possibly increase the flow completion time by early termination of credit generation. As mentioned earlier, to work for the core-congested WAN, existing PCC protocols should leverage receiver-driven credit emulation. Then, receiver cannot determine the exact amount of credit successfully granted to each flow. This is problematic as it means that the receiver also cannot determine the exact time to stop credit generation. The receiver has two choices. On one hand, if it keeps sending credits until getting the last data packets, then there will be roughly one RTT wastage of credits, leading to network under-utilization. On the other hand, if it stops credit generation immediately when in-flight credits are enough to cover the flow traffic demand, the flow may need to request for more credits when some credits are dropped in the emulation process, leading to higher network latency.

Figure 5 shows our experimental result of running realistic workload on a network with an average flow RTT of 60ms and bottleneck link of 1Gbps (same setup as §IV-C2). We generate synthetic workload [32]. Flow sizes are varied from 10MB to 80MB (based on our WAN measurement in §IV-A). The average network load is set to 0.8 of the full network capacity. We find that the credit wastage[2] of ExpressPass can get up to 23.5% of the total successfully received credits. Notice that the ExpressPass paper [20] shows even higher credit wastage up to 60% in a workload with many small flows.

III. FlashPass Design

A. Design overview

In this work, we aim to design a practical and efficient transport protocol to simultaneously achieve high throughput and low loss rate for shallow-buffered WAN (§II-A). Our investigation over reactive transport shows its inherent insufficiency, requiring non-negligible buffering (§II-B). We then

[2]The credit wastage is measured by the received but not used credits.

turn to the emerging proactive transport. While revealing the promising potential of the receiver-driven PCC protocols, we also find two key technical challenges (§II-C):

1) How to schedule credits effectively without triggering data crush even if network traffic has diverse RTTs?
2) How to generate credits sufficiently while not wasting credits even if the granted amount is unpredictable?

The first challenge is fundamental to the receiver-driven protocols. This is because receiver-driven protocols uses the credit sending on the reverse path to emulate forward path data sending. The network delay between receiver and bottleneck link is different from that between sender and bottleneck. Thus, interleaved credits passing through the reverse path cannot guarantee well interleaved incoming data at the bottleneck. To address it, we propose to adopt a sender-driven emulation mechanism. This ensures that the emulation follows the same direction as the real data sending. Besides, strict timing [3] should be enforced to the data sending. This ensures that the delay for emulation/credit packet to trigger data packet out keeps constant, instead of dependent on the different RTTs.

To address the second challenge, we should keep generating credits even if the expected incoming credits are enough to cover the flow traffic demand (we call it *over-provisioning*). This is to ensure that flows can finish quickly with sufficient credits, even if the amount of credits successfully passing through the emulation network and granted to the sender is unknown. However, we should also make sure these over-provisioned credits waste no bandwidth, i.e., they should only occupy the leftover bandwidth by the ordinary traffic. To this end, we incorporate a selective dropping mechanism to grant the over-provisioned credits in a "best-effort" manner.

Therefore, we propose a novel PCC solution - FLASHPASS to simultaneously deliver high throughput and low loss rate for shallow-buffered WAN. FLASHPASS is the first sender-driven PCC protocol. It leverages a sender-driven emulation process together with send time calibration to effectively allocate credits for bandwidth and eliminate the data packet crush in time (§III-B). It enables early data transmission in the flow starting phase, and incorporates credit over-provisioning together with a selective dropping mechanism to efficiently utilize network bandwidth in the flow finishing phase (§III-C).

B. Sender-driven emulation mechanism

FLASHPASS leverages a sender-driven emulation mechanism (Figure 6) to allocate network bandwidth without data packet crush. The emulation process requires to separate the network into two parts: an emulation network running on the emulation queue, and a real data communication network on the data communication queue. Emulation packet is set to the minimum size, i.e., 84B Ethernet frame, including the preamble and inter-packet gap. Each emulation packet passing through the network triggers a credit return from receiver and then a sender data transmission up to a maximum size Ethernet frame (e.g., 1538B). Thus, in Ethernet, the emulation network

[3]Existing PCCs have no strict timing on data sending in the received credits.

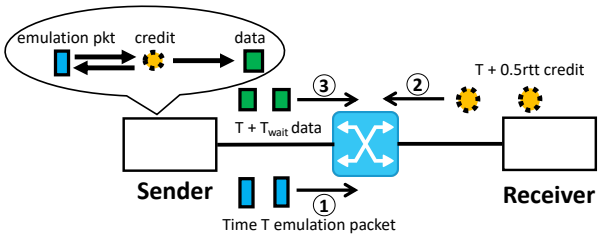


Fig. 6: Sender-driven emulation mechanism.

TABLE IV: Interleaved data arrival of FLASHPASS (Figure 4).

	Start time	Credit send	Data send	Data arrive
Flow 1	0ms	20-40ms	40-60ms	~60-80ms
Flow 2	30ms	40-60ms	70-90ms	~80-100ms

[a]Interleaved data arrival avoids high buffering and packet loss.

is rate-limited to 84/(84+1538) ≈ 5% of the link capacity, and the remaining 95% is used for the data communication. Unlike existing PCCs that send data packets as soon as credits arrive, FLASHPASS encodes timestamps (T) in its emulation packets and credit packets, and grants data sending at exact time of $T + T_{wait}$ to eliminate data crush at the bottleneck link. This send time calibration mechanism effectively helps in avoiding data packet crush in time as shown below.

Figure 6 shows the running process of the sender-driven emulation mechanism. ① At time T, a new flow arrives and immediately generates emulation packet at full rate of the emulation network. The emulation packet will compete on the emulation network and get dropped if queues form. The emulation network is configured with a low buffer of several packets. ② At around time $T+0.5rtt$, emulation packet passes through the network and triggers the credit feedback[4] from the receiver side. And around half a RTT later, the sender receives and records the credit packet. ③ At time $T + T_{wait}$, the sender injects the data packet correspondingly into the data communication network. Note that T_{wait} should be larger than the maximum RTT of the network[5]. In this way, emulation network exactly mimics the bandwidth competition of the future data communication network. If network is over-utilized, only emulation packets get dropped on the emulation network and data communication network remains zero buffering.

To further illustrate the effectiveness of sender-driven emulation mechanism, we compare the result of FLASHPASS (Table IV) with that of the receiver-driven PCC (Table III) - the running process both in the case of Figure 4. In FLASHPASS, the T_{wait} is set to the maximum RTT of the network, i.e., 40ms. We find that the Flow 2 adjusts its data send time to 70-90ms, and thus lead to ~80-100ms data arrival time at the receiver side. This effectively avoids the data crush with Flow 1 during time 60-80ms as with the receiver-driven solution. We should also notice that receiver-driven solution cannot resolve the data crush problem in general by simply adding a constant waiting time T_{wait}, even if it seems to work in this case (bottleneck at the ToR-to-receiver link). This is

[4]Credit packets are seldomly lost (not in our simulation with perfect timing). And we can put it on a high priority queue to avoid the unexpected losses.

[5]Given a private WAN with shallow buffering, the maximum RTT of the network can be obtained based on simple measurement of the baseline RTT.

Algorithm 1: Emulation Feedback Control at Sender.

Input : New Incoming Credit Packet
Output : Emulation Packet Sending Rate cur_rate
Initialize: $cur\_rate \leftarrow max\_emulation\_rate$

```
/* Update the emulation loss rate */
loss_rate = 1 − #_credit_in/#_emulation_out ;
/* Update sending rate every RTT */
if loss_rate > max_target_loss then
    cur_rate = cur_rate ×
      (1 − loss_rate) × (1 + min_target_loss) ;
else if loss_rate < min_target_loss then
    cur_rate = cur_rate × (1 + max_target_loss) ;
```

because in practice, bottleneck link may reside in any hop of the network, and the time or distance to reach the bottleneck is different and unknown for both senders and receivers. Thus, it is fundamentally difficult to avoid data crush without faithfully mimic the data sending from a sender-driven approach.

Emulation feedback control: FLASHPASS uses an emulation feedback control to regulate the sending rate of emulation packets. This is to ensure high bandwidth utilization and fairness, which is similar to ExpressPass[6]. For example, in the parking lot scenario, naively sending out emulation packets at linerate will lead to link under-utilization. To address it, a feedback control is required for the emulation process.

Unlike data forwarding, the emulation feedback control has low cost in its emulation packet losses and thus can be aggressive in probing for more bandwidth. Observing this characteristic, FLASHPASS adopts a simple yet effective loss-based feedback control as shown in Algorithm 1. The emulation packet loss rate is controlled between min_target_loss and max_target_loss (by default 1% and 10%, respectively). The rate adjustment follows a multiplicative-increase and multiplicative-decrease (MIMD) control rule. With the record of current loss rate and the target loss range, the increase and decrease adjustment can be calculated precisely. In this way, FLASHPASS emulation process can achieve fast convergence to full link utilization. Notice that FLASHPASS emulation process allows fast start for new flows while still guaranteeing low credit waste with the selective dropping mechanism. We also show that the feedback control algorithm here helps in calculating the expected credits inflight (see Equation 2 and more details in §III-C).

Sender-driven vs receiver-driven? There are both pros and cons for sender-driven approach. On one hand, sender-driven approach can emulate the data forwarding with precisely interleaved arrival time at the bottleneck link, and thus can practically achieve near zero buffering. This is the most desirable feature in our targeting shallow-buffered WAN scenario, and explains why we choose the sender-driven approach in our design. Besides, it does not need to ensure path symmetry like the receiver-driven protocols, where data packet must exactly follow the reverse path of the credit emulation.

[6]More detailed illustration can be found in Figure 4 of ExpressPass [20]

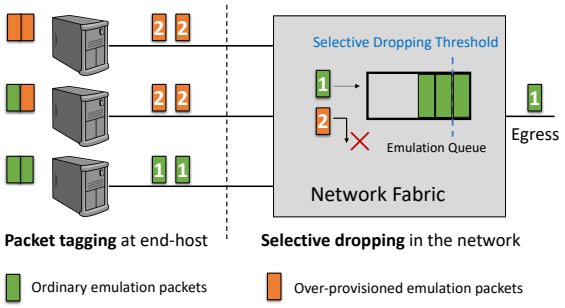


Fig. 7: Over-provisioning with selective dropping mechanism.

On the other hand, sender-driven approach introduces half a RTT longer delay between emulation and data sending. Such a longer delay can have negative impact without careful consideration. In FLASHPASS, we have well handle this problem for both the starting phase and finishing phase. Besides, it also adds one-way more control packets. To reduce this overhead, FLASHPASS adopts delayed credit sending. Specifically, it waits for more credits of the same flow and feeds them back in single packet if timestamp info indicates a long waiting time for data sending. The delayed feedback credit packets should include timestamps of all corresponding emulation packets.

C. Over-provisioning with selective dropping mechanism

To handle the credit delay problem in the flow starting phase, FLASHPASS adopts similar idea as in Aeolus [23]. Specifically, new flows start at line rate in the first RTT before receiving any credit (i.e., *pre-credit phase*). The first RTT packets are marked as "unscheduled". The unscheduled data packets in the pre-credit phase can then get selectively dropped if meeting the (non-first-RTT) scheduled packets in the network (similar to the selective dropping mechanism described below). In case of first-RTT packet losses, a tail loss probing is employed together with credit-scheduled retransmission.

To handle the credit wastage problem in the flow finishing phase (unsolved by Aeolus), FLASHPASS incorporates a credit over-provisioning mechanism with selective dropping discipline (a major contribution in this paper). Specifically, FLASHPASS keeps generating emulation packets and triggering out credit packets even if the expected incoming credits are enough to cover the flow traffic demand (we call it *over-provisioning*). While during the packet emulation process, the over-provisioned emulation packets are selectively dropped if conflicting with the ordinary emulation packets in the network. This ensures that the over-provisioned credits only occupy the leftover bandwidth not used by the ordinary traffic demand.

However, it is also challenging to determine the right time to switch emulation packets from ordinary to over-provisioned. To this end, we need to maintain an estimation of the incoming credits. If the estimate cannot cover the remaining traffic demand, emulation packets are set to be ordinary. Otherwise, emulation packets are over-provisioned.

Figure 7 illustrates the detailed credit over-provisioning and selective dropping mechanism of FLASHPASS.

Packet tagging at the end-host. The end-host maintains a per-flow estimation of the incoming credit packets ($C_{expected}$).

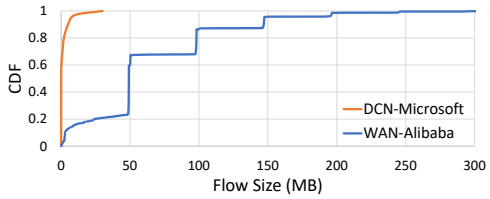


Fig. 8: Flow size distribution.

TABLE V: Traffic characteristic of a production WAN.

	Small Flow	Large Flow	Average
Flow Size (MB)	0-10	>10	65
Flow Percentage	15.8%	84.2%	-

Specifically, the sender end-host records the in-flight emulation packets (E_{out}). With the regulation of the emulation feedback control loop, the actual emulation packet loss rate is expected to be lower than max_target_loss most of the time. Thus, the expected credits can be calculated following Equation (2):

$$C_{expected} >= E_{out} \times (1 - max\_target\_loss) \quad (2)$$

When the expected credit amount $C_{expected}$ exceeds the remaining traffic demand[7] of the flow, the sender end-host starts marking the emulation packets as over-provisioned, i.e., with high dropping priority label. Otherwise, emulation packets are marked as ordinary, i.e., with low dropping priority label.

Selective dropping on the network fabric. Based on the dropping priority label in the emulation packets, network switches selectively drop the high dropping priority packets before low priority ones when network bandwidth is not enough. Commodity switching chips cannot push out packets that are already stored in the switch buffers. Thus, we can only selectively drop packets at the ingress queue. To this end, we adopt a feature of RED/ECN function, which is widely supported by commodity switches [33], [34]. Specifically, when the switch queue length exceeds the ECN marking threshold, the switch will mark the arrival ECN-capable packets and drop the non-ECN-capable packets. Therefore, FLASHPASS can repurpose this function to achieve selective dropping. Senders can simply set ordinary emulation packets as ECN-capable and over-provisioned emulation packets as non-ECN-capable, to enable selective dropping of the over-provisioned packets with high priority on the network fabric.

IV. EVALUATION

In this section, we present the detailed FLASHPASS evaluation setup in §IV-A, and conduct extensive experiments to answer the following questions:

- *Can* FLASHPASS *achieve high throughput and low loss rate under static workload?* In static workload experiments (§IV-B1), FLASHPASS achieves near full throughput (9.14Gbps) with zero packet loss persistently across various settings under the shallow-buffered WAN. The throughput is up to 55.9% higher than that of TCP Cubic.
- *Can* FLASHPASS *reduce flow completion time (FCT) under realistic dynamic workload?* In realistic dynamic workload experiments (§IV-B2), FLASHPASS with Aeolus enhancement achieves 28.5%-32% and 3.4%-11.4% smaller overall flow completion times compared to Cubic, and Aeolus-enhanced ExpressPass, respectively.
- *How do different parameters and components of* FLASHPASS *impact its network performance?* We show the impact of parameter settings of FLASHPASS in §IV-C1, and validate the effectiveness of the over-provisioning with selective dropping mechanism in §IV-C2.

[7]We assume traffic demand information is available. If not, we simply use the send buffer occupancy to calculate the remaining traffic demand [56].

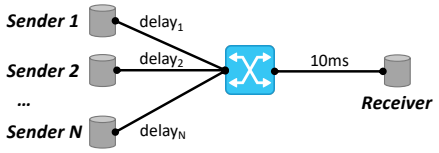


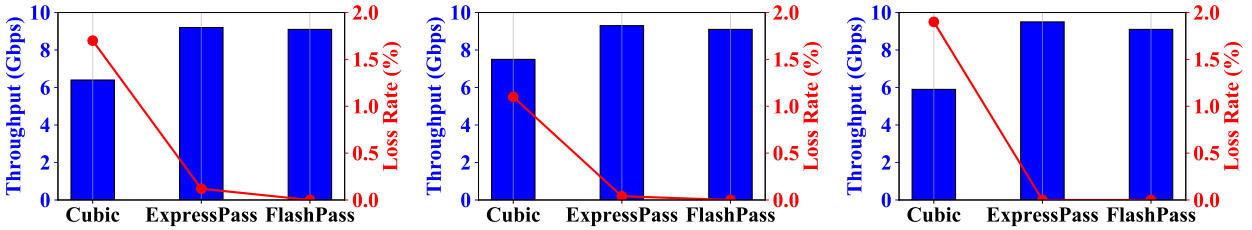
Fig. 9: Dumbbell network topology.

A. Evaluation Setup

Schemes Compared: We mainly compare the performance of FLASHPASS with TCP Cubic [12] and ExpressPass [20]. TCP Cubic is a loss-based reactive congestion control (RCC) protocol that is enabled by default in Linux. It is serving for the majority of the real-world wide-area traffic nowadays. ExpressPass is one of the seminal proactive congestion control (PCC) protocols. Based on our knowledge, it is the only existing PCC that can practically work on WAN. We use the default parameter settings in ExpressPass. We have evaluated both FLASHPASS and ExpressPass with and without Aeolus [23]. Aeolus is a building block for PCC solutions that improves their performance in the pre-credit phase (i.e., first RTT).

Experiment Configuration: We build and run our NS2 simulation based on the open-sourced code from ExpressPass [31]. In the static workload experiments (§IV-B1), we mainly use a dumbbell network topology as shown in Figure 9. In the dynamic workload experiments (§IV-B2), we mainly use a regional wide-area network as shown in Figure 13. By default, the switch data packet buffer is set to 20KB per port per Gbps according to the Table I. The emulation/credit network queue buffer is set to 8 packets for both FLASHPASS and ExpressPass. The selective dropping threshold is set to 2 packets for both Aeolus and over-provisioning mechanism of FLASHPASS. The retransmission timeout is set to 200ms. The packet MTU is set to 1.5KB.

Realistic Workload: We measured the flow size from a regional wide-area network of Alibaba. The data are collected on links between two data centers. The traffic running on those links are mainly from a data storage service. The flow size distribution is shown in Figure 8 and summarized in Table V. Compared with the datacenter workload [35]–[37], wide-area workload has much larger flow size (~65MB) on average. Based on our classification, there are more than 84.2% flows that are large-sized with more than 10MB traffic volume. The largest flows on WAN can get up to GBs (not shown in the figure), which again are much larger than those on DCN. In our realistic workload experiments, workloads are generated based on traffic patterns measured from the production WAN. Flows



(a) $delay_i = Uniform(0, 20ms)$, $i = 1, 2, ..., N; N = 20$.

(b) $delay_i = Uniform(0, 20ms)$, $i = 1, 2, ..., N; N = 200$.

(c) $delay_i = 10ms, i = 1, 2, ..., N$; $N = 20$.

Fig. 10: Static workload experiment results running on a dumbbell network topology shown in Figure 9. The blue bar indicates the throughput (Gbps) performance on the left y-axis. The red line indicates the packet loss rate (%) on the right y-axis.

arrive by the Poisson process. The source and destination is chosen uniformly random from different DCs.

Performance Metrics: In the static workload experiments (§IV-B1), we mainly measure the throughput or network utilization, packet loss rate, and buffer occupancy. In the dynamic workload experiments (§IV-B2), we use flow completion time (FCT) as the major performance metric, which can directly reflect the data transfer speed of network applications.

B. Evaluation Results

1) Static Workload Experiments: In this part, we evaluate the performance of FLASHPASS under static workload experiments. We mainly use the dumbbell network topology as shown in Figure 9. All links have 10Gbps capacity. There are N senders that simultaneously transfer data to the same receiver. The network delay from sender to the bottleneck switch is set to a uniformly random value between 0 and 20ms. We also test a case with identical delay of 10ms (i.e., 40ms RTT). The sender number is set to 20 or 200. Flows start randomly in the initial 0.2 second and run for 5 seconds.

Figure 10 shows the experimental results. Specifically, Figure 10a shows the results under 20 long flows with different RTTs. The throughput is 6.4Gbps, 9.2Gbps, 9.1Gbps for TCP Cubic, ExpressPass, and FLASHPASS. And the packet loss rate is 1.7%, 0.12%, and 0 for TCP Cubic, ExpressPass, and FLASHPASS, respectively. In general, FLASHPASS achieves the best performance compared to Cubic and ExpressPass. It is able to maintain near full throughput with zero packet loss throughout all experiments.

As a reactive protocol, TCP Cubic achieves the lowest throughput and worst loss rate throughout the experiments. Compared to FLASHPASS, the throughput can get up to 35.9% lower in the worst case (Figure 10c). This is because the "try and backoff" nature of RCC leads to inherently high data packet losses, and as well periodical under-utilization after rate "backoff". Besides, Cubic uses slow start to ramp up its sending rate at the initial phase, which can lead to high losses at the end of the slow start (significant losses are observed during time at ∼0.5 second in our experiments).

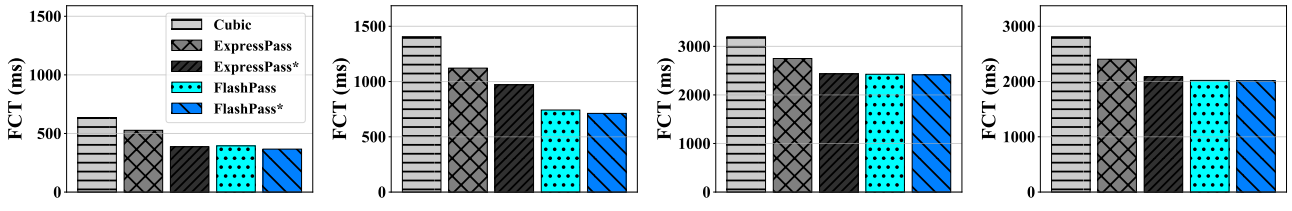
While ExpressPass achieves similar throughput with FLASHPASS, it has observable packet losses in the scenario with different RTTs (i.e., Figure 10a and 10b). While the packet losses only slightly reduce the throughput of large data transfer in this case, it has much larger negative impact on small flows as we will see in the dynamic workload experiments. FLASHPASS can effectively resolve the imperfect scheduling issue with its sender-driven emulation process. Notice that FLASHPASS has a slightly lower throughput due to the emulation packet overhead. In the static experiments, there is one-way traffic only, which hides the reverse path credit overhead of ExpressPass.

When comparing the case of 20 flows (Figure 10a) with that of 200 flows (Figure 10b), the throughput is better while the packet loss is worse for both Cubic and ExpressPass with smaller concurrent flows. This is because more flows can lead to better statistical multiplexing [10] and thus more stable throughput performance in general. When comparing the case of different RTTs (Figure 10a) with that of identical RTTs (Figure 10c), we find that the packet losses of ExpressPass are indeed caused by the different RTTs on WAN. Identical RTTs also leads to lower throughput for Cubic because it increases the probability of synchronized "sawtooth" (i.e., larger variation in congestion window or sending rate).

2) Dynamic Workload Experiments: In this part, we evaluate the performance of FLASHPASS under realistic dynamic workload experiments. We mainly use the wide-area network (WAN) topology as shown in Figure 13. There are three datacenters (DCs) in the wide-area region, connecting with each other by WAN links of 20ms, 30ms, and 40ms one-way delay, respectively. All wide-area links have 10Gbps capacity. There are 20 hosts in each DC connecting directly to the wide-area border router each with a 1Gbps link. The DC link delay is set to 10 microseconds, which is negligible compared to that of WAN. The workloads are generated based on traffic patterns measured from the production WAN (Figure 8). Flows arrive by the Poisson process. The source and destination is chosen uniformly random from different DCs. Therefore, the WAN links and DC links should have equal load on average. We vary the load from 0.4 to 0.8 of the full network capacity.

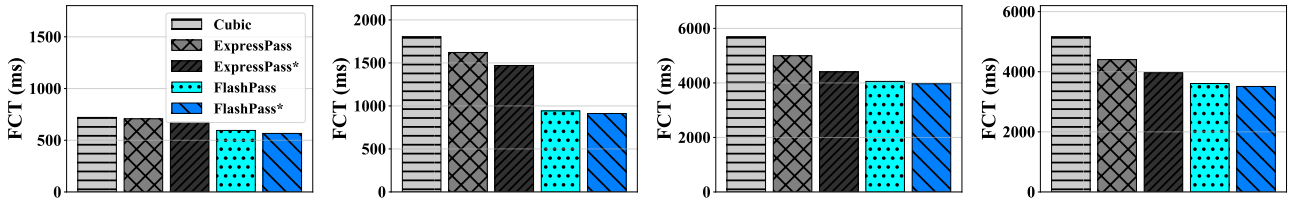
Figure 11 and Figure 12 show the experimental results. In general, FLASHPASS performs the best for flow completion times (FCTs) of both small flows (<10MB) and large flows (>10MB) compared to Cubic and ExpressPass. Specifically, for the best version (i.e., enhanced by Aeolus), FlashPass* reduces the overall FCT by 28.5%-32%, and 3.4%-11.4% when comparing to Cubic and ExpressPass*, respectively.

For small flows, reactive congestion control like Cubic adds time delay for slow start and its inherent high packet loss also



(a) Small Flow - Average. (b) Small Flow - 99th Tail. (c) Large Flow - Average. (d) All Flow - Average.

Fig. 11: Flow completion times (FCTs) of various transport protocols under realistic dynamic workload (average load = 0.4). ExpressPass* indicates the Aeolus-enhanced ExpressPass version. FlashPass* indicates the Aeolus-enhanced FlashPass version.



(a) Small Flow - Average. (b) Small Flow - 99th Tail. (c) Large Flow - Average. (d) All Flow - Average.

Fig. 12: Flow completion times (FCTs) of various transport protocols under realistic dynamic workload (average load = 0.8). ExpressPass* indicates the Aeolus-enhanced ExpressPass version. FlashPass* indicates the Aeolus-enhanced FlashPass version.

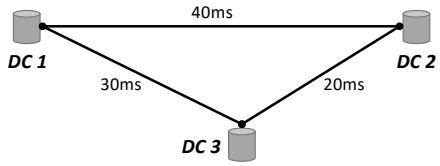


Fig. 13: Wide-area network (WAN) topology.

introduces high retransmission timeout overhead, leading to much larger completion times than those of proactive transports. While for ExpressPass, it probes for more bandwidth with much more aggressive credit control algorithm while still keeping low packet losses and thus low timeout. FLASHPASS improves the emulation process and achieves zero losses, thus reducing the small flow completion times to a great extent. When enhanced by Aeolus with linerate start (i.e., ExpressPass* and FlashPass*), one more RTT is saved for credit packets to start data sending, resulting in even lower small flow FCT. This is especially important when the average network load is low. For example, FlashPass* reduces the small flow FCT by 5.1%-16.4% on average and 24.3%-38.0% at 99-th tail when comparing to ExpressPass*.

For large flows, due to shallow buffer and high packet losses, Cubic suffers from low throughput and thus achieves relatively high FCTs for large flows. While there are also some losses for ExpressPass data sending due to imperfect scheduling in time, the negative impact is very limited because it does not reset its sending rate when loss or timeout happens. However, it does waste some bandwidth due to the last RTT credit scheduling. Such a credit wastage introduces severe negative impact when the network load is high (e.g., when average load = 0.8). FLASHPASS handles the last RTT credit scheduling issue with over-provisioning and selective dropping mechanism. Thus, the credit wastage is effectively avoided, leading to roughly 10.2% reduction on large flow completion times when comparing FlashPass* with ExpressPass*.

C. Deep Dive

1) How do parameters of FLASHPASS *affect its network performance:* In this part, we evaluate the FLASHPASS performance under various parameter settings. First of all, FLASHPASS sets the initial credit rate to the maximum of the emulation network for new flows. This is enabled by the low loss penalty of credit packets as well as the negligible credit wastage with the help of selective dropping mechanism of FLASHPASS. The linerate credit start helps to reduce short flow completion times, especially when the network is mostly idle. Secondly, the FLASHPASS emulation feedback control loop uses two parameters, i.e., min_target_loss and max_target_loss, to control the target credit loss rate. We repeat the same experiments in §IV-B2 and Figure 14 shows the results. In general, higher target loss rates lead to faster convergence under traffic dynamics and thus higher efficiency in utilizing network bandwidth, while lower ones have smaller emulation packet overhead. Based on the results, we recommend default parameter settings of $min\_target\_loss = 1\%$ and $max\_target\_loss = 10\%$.

2) How effective is the over-provisioning with selective dropping mechanism of FLASHPASS *in avoiding bandwidth or credit wastage:* In this part, we compare the performance of FLASHPASS with and without the over-provisioning with selective dropping mechanism. We again use the wide-area network topology shown in Figure 13. We generate synthetic workload [32]. Flow sizes are varied from 10MB to 80MB. The average network load is set to 0.8 of the full network capacity. Figure 15 shows the experimental results. We find that the over-provisioning with selective dropping mechanism of FLASHPASS helps to save credits in the last RTT by 3.4%-27% of the total traffic volume. This leads to a reduced flow completion time by up to 19% comparing to FLASHPASS without the over-provisioning and selective dropping mechanism.

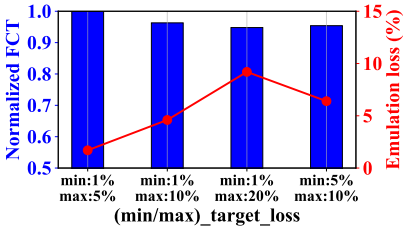


Fig. 14: FCT and credit loss rate of FLASHPASS. The average FCT is normalized by the result of FLASHPASS with $min\_target\_loss = 1\%$ and $max\_target\_loss = 5\%$.

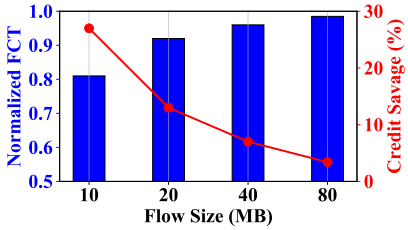


Fig. 15: FCT and credit savage of FLASHPASS with over-provisioning mechanism. The average FCT is normalized by the result of FLASHPASS without over-provisioning.

V. PRACTICAL DEPLOYMENT ANALYSIS

For practical deployment on the enterprise WAN, there are some more requirements for FLASHPASS than the pure end-to-end transport. Firstly, a separate emulation queue should be reserved and rate limited on network switches for emulation process. Secondly, to enable selective dropping (§III-C), single-threshold ECN marking/dropping should be configured on the emulation queue. These requirements can be achieved with commodity switches, but need non-trivial configuration across the whole network. Thirdly, we also notice that PCC solutions cannot co-exist with the legacy TCP protocols. A straightforward workaround is to separate different traffic with multiple queues, but may bring about some overheads.

Lastly, an efficient implementation of proactive congestion control logic is required. Recent work [38], [39] presents a Linux kernel implementation (based on Homa) that achieves magnitudes lower latency than TCP. Specifically, a variety of issues such as batching, load balancing, and realtime processing, have been well addressed in the work. There are some other efforts on realizing PCC protocols on user-space DPDK [23], or on congestion control plane (CCP) [40]. While these efforts have validated the feasibility of building an efficient PCC network stack, prototyping a fully functional FLASHPASS is our next step effort and is beyond the scope of this paper. Our hope is that the design, analysis, and extensive simulations conducted in this paper will pave the way for the next step of prototyping and deployment.

VI. RELATED WORK

Reactive Congestion Control (RCC): The seminal work of TCP congestion control [11] works in a reactive manner. It detects congestion based on a delayed feedback signal (e.g., packet loss) from the network and reacts passively. Many variants have emerged since then, e.g., Cubic [12], Compound [41], etc. Vegas [13] is the first delay-based protocol to avoid intrinsic high loss and queueing delay of loss-based transport. After that, many protocols [14], [15] are proposed to use delay signal (either explicitly or implicitly). More recently, PCC-Allegro [42] proposes to react to congestion based on the measured performance. It is followed by PCC-Vivace [43] and PCC-Proteus [44] that extend the framework to be more efficient and TCP-friendly (e.g., on Internet). However, these protocols are essentially reactive, and may hardly meet our need on shallow-buffered WAN (e.g., TCP variants in §II-B).

There are other RCC protocols designed in particular for DCN. DCTCP [35] detects the network congestion with explicit congestion notification (ECN) signal. Following that, many ECN-based protocols [45]–[53] are proposed. More recently, HPCC [54] leverages In-band Network Telemetry (INT) to detect rate mismatch, and adjust sending rates accordingly. These protocols require advanced congestion signals that are not available or well supported [55] on WAN.

Proactive Congestion Control (PCC): PCC protocols propose to allocate bandwidth proactively. Centralized PCC [18] regulates traffic rate with a centralized controller. Switch-assisted PCC [19] leverages switch assistance in bandwidth allocation. Both impose high requirements on network facilities (e.g., cutting payload [24]) and hence are either unscalable or impractical. Some receiver-driven protocols like Homa [21] employ simple credit scheduling on the receiver side. They assume single bottleneck link between the top-of-rack (ToR) switch and receiver, which does not hold on WAN. Other receiver-driven protocols like ExpressPass [20] that leverage credit emulation on a separate queue for bandwidth allocation. However, they suffer from efficiency problems on WAN. In contrast, FLASHPASS addresses these problems to fully unleash the power of PCC on shallow-buffered WAN.

VII. CONCLUSION

In this paper, we reveal the trend of adopting shallow-buffered commodity switching chips on wide-area networks (WAN). We then investigate its impact on network performance, and find the insufficiency of the TCP-style reactive congestion control (RCC). To address it, we turn to the emerging proactive congestion control (PCC), and propose FLASHPASS. FLASHPASS is the first attempt to employ PCC on WAN. It is also the first sender-driven credit-scheduled PCC protocol. It leverages the sender-driven emulation process and over-provisioning with selective dropping mechanism to work practically and effectively for the shallow-buffered WAN. Extensive experiments are conducted and validate the superior performance of FLASHPASS under realistic workload.

ACKNOWLEDGMENT

We thank our shepherd Fahad Dogar and the anonymous reviewers for their constructive feedback and suggestions. This work is supported in part by the Hong Kong RGC TRS T41-603/20-R, GRF-16215119 and the Alibaba Research Grant. Kai Chen is the corresponding author of the paper.

REFERENCES

[1] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, and A. Vahdat. "B4: experience with a globally-deployed software defined wan," in SIGCOMM. ACM, 2013, pp. 3–14.

[2] C. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer. "Achieving high utilization with software-driven WAN," in SIGCOMM. ACM, 2013, 15–26.

[3] C. Hong, S. Mandal, M. Al-Fares, M. Zhu, R. Alimi, K. Naidu B., C. Bhagat, S. Jain, J. Kaimal, S. Liang, K. Mendelev, S. Padgett, F. Rabe, S. Ray, M. Tewari, M. Tierney, M. Zahn, J. Zolla, J. Ong, and A. Vahdat, "B4 and after: managing hierarchy, partitioning, and asymmetry for availability and scale in google's software-defined WAN," in SIGCOMM. ACM, 2018, pp. 74–87.

[4] A. Kumar, S. Jain, U. Naik, A. Raghuraman, N. Kasinadhuni, E. C. Zermeno, C. S. Gunn, J. Ai, B. Carlin, M. Amarandei-Stavila, M. Robin, A. Siganporia, S. Stuart, and A. Vahdat. "BwE: Flexible, Hierarchical Bandwidth Allocation for WAN Distributed Computing," in SIGCOMM. ACM, 2015, pp. 1–14.

[5] H. Zhang, K. Chen, W. Bai, D. Han, C. Tian, H. Wang, H. Guan, and M. Zhang. "Guaranteeing Deadlines for Inter-Datacenter Transfers," in EuroSys. ACM, 2015.

[6] N. Farrington, E. Rubow and A. Vahdat. "Data Center Switch Architecture in the Age of Merchant Silicon," in 17th IEEE Symposium on High Performance Interconnects. IEEE, 2009, pp. 93-102.

[7] A. Singh, J. Ong, A. Agarwal, G. Anderson, A. Armistead, R. Bannon, S. Boving, G. Desai, B. Felderman, P. Germano, A. Kanagala, J. Provost, J. Simmons, E. Tanda, J. Wanderer, U. Hölzle, S. Stuart, and A. Vahdat. "Jupiter Rising: A Decade of Clos Topologies and Centralized Control in Google's Datacenter Network," in SIGCOMM. ACM, 2015, pp. 183–197.

[8] S. Supittayapornpong, B. Raghavan, and R. Govindan. "Towards highly available clos-based WAN routers," in SIGCOMM. ACM, 2019, pp. 424–440.

[9] Packet buffers. https://people.ucsc.edu/ warner/buffer.html. Accessed on March 2021.

[10] N. McKeown, G. Appenzeller, and I. Keslassy. "Sizing router buffers," in SIGCOMM. ACM, 2004, pp. 281–292.

[11] V. Jacobson. "Congestion avoidance and control," in SIGCOMM. ACM, 1988, pp. 314–329.

[12] S. Ha, I. Rhee, and L. Xu. "CUBIC: a new TCP-friendly high-speed TCP variant," in SIGOPS. ACM, 2008, pp. 64–74.

[13] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson. "Tcp vegas: New techniques for congestion detection and avoidance," in SIGCOMM. ACM, 1994.

[14] V. Arun and H. Balakrishnan. "Copa: Practical Delay-Based Congestion Control for the Internet," in NSDI. USENIX, 2018, pp. 329-342.

[15] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson. "BBR: congestion-based congestion control," in Communications of the ACM. ACM, 2017, pp. 58–66.

[16] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. "TCP Selective Acknowledgment Options," in RFC 2018. IETF, 1996.

[17] Y. Cheng, N. Cardwell, N. Dukkipat, and P. Jha. "The RACK-TLP Loss Detection Algorithm for TCP," in RFC 8985. IETF, 2021.

[18] J. Perry, A. Ousterhout, H. Balakrishnan, D. Shah, and H. Fugal. "Fastpass: a centralized "zero-queue" datacenter network," in SIGCOMM. ACM, 2014, pp. 307–318.

[19] M. Handley, C. Raiciu, A. Agache, A. Voinescu, A. W. Moore, G. Antichi, and M. Wójcik. "Re-architecting datacenter networks and stacks for low latency and high performance," in SIGCOMM. ACM, 2017, pp. 29–42.

[20] I. Cho, Keon Jang, Dongsu Han. "Credit-Scheduled Delay-Bounded Congestion Control for Datacenters," in SIGCOMM. ACM, 2017, pp. 239-252.

[21] B. Montazeri, Y. Li, M. Alizadeh, and J. Ousterhout. "Homa: a receiver-driven low-latency transport protocol using network priorities," in SIGCOMM. ACM, 2018, pp. 221–235.

[22] S. Hu, W. Bai, B. Qiao, K. Chen, K. Tan. "Augmenting Proactive Congestion Control with Aeolus," in APNet, 2018.

[23] S. Hu, W. Bai, G. Zeng, Z. Wang, B. Qiao, K. Chen, K. Tan, and Y. Wang. "Aeolus: A Building Block for Proactive Transport in Datacenters," in SIGCOMM. ACM, 2020, pp. 422–434.

[24] P. Cheng, F. Ren, R. Shu, and C. Lin. "Catch the whole lot in an action: rapid precise packet loss notification in data centers," in NSDI. USENIX, 2014, pp. 17–28.

[25] Data Centers - Google. https://www.google.com/about/datacenters. Accessed on March 2021.

[26] Datacenter Infrastructure - Microsoft. https://www.microsoft.com/en-us/cloud-platform/global- datacenters. Accessed on March 2021.

[27] Data Center Engineering - Facebook Engineering. https://engineering.fb.com/category/data-center-engineering. Accessed on March 2021.

[28] Alibaba Cloud's Global Infrastructure. https://www.alibabacloud.com/global-locations. Accessed on March 2021.

[29] L. A. Barroso, J. Clidaras, U. Hoelzle. "The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines," Morgan & Claypool, 2013.

[30] L. Kleinrock. "Power and deterministic rules of thumb for probabilistic problems in computer communications," in Proceedings of the International Conference on Communications. 1979.

[31] kaist-ina/ns2-xpass. https://github.com/kaist-ina/ns2-xpass. Accessed on March 2021.

[32] A. Saeed, V. Gupta, P. Goyal, M. Sharif, R. Pan, M. Ammar, E. Zegura, K. Jang, M. Alizadeh, A. Kabbani, and A. Vahdat. "Annulus: A Dual Congestion Control Loop for Datacenter and WAN Traffic Aggregates," In SIGCOMM. ACM, 2020, pp. 735–749.

[33] H. Wu, J. Ju, G. Lu, C. Guo, Y. Xiong, and Y. Zhang. "Tuning ECN for data center networks," in CoNEXT. ACM, 2012, pp. 25–36.

[34] G. Judd. "Attaining the promise and avoiding the pitfalls of TCP in the datacenter," in NSDI. USENIX, 2015, pp. 145–157.

[35] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan. "Data center tcp (dctcp)," in SIGCOMM, 2010.

[36] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta. "VL2: a scalable and flexible data center network," in SIGCOMM. ACM, 2009, pp. 51–62.

[37] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren. "Inside the Social Network's (Datacenter) Network," in SIGCOMM. ACM, 2015, pp. 123–137.

[38] J. Ousterhout. "A Linux Kernel Implementation of the Homa Transport Protocol," in ATC. USENIX, 2021.

[39] PlatformLab/HomaModule. https://github.com/PlatformLab/HomaModule. Accessed on March 2021.

[40] A. Narayan, F. Cangialosi, D. Raghavan, P. Goyal, S. Narayana, R. Mittal, M. Alizadeh, and H. Balakrishnan. "Restructuring endpoint congestion control," In SIGCOMM. ACM, 2018, pp. 30–43.

[41] K. Tan, J. Song, Q. Zhang and M. Sridharan. "A Compound TCP Approach for High-Speed and Long Distance Networks," in INFOCOM. IEEE, 2006, pp. 1-12.

[42] M. Dong, Q. Li, D. Zarchy, P. B. Godfrey and M. Schapira. "PCC: Re-architecting Congestion Control for Consistent High Performance," in NSDI, 2015.

[43] M. Dong, T. Meng, D. Zarchy, E. Arslan, Y. Gilad, B. Godfrey and M. Schapira. "PCC Vivace: Online-Learning Congestion Control," in NSDI, 2018.

[44] T. Meng, N. R. Schiff, P. B. Godfrey and M. Schapira. "PCC Proteus: Scavenger Transport And Beyond," in SIGCOMM, 2020.

[45] M. Alizadeh, A. Kabbani, T. Edsall, B. Prabhakar, A. Vahdat, and M. Yasuda. "Less is more: trading a little bandwidth for ultra-low latency in the data center," in NSDI, 2012.

[46] B. Vamanan, J. Hasan, and T. Vijaykumar. "Deadline-aware datacenter tcp (d2tcp)," in SIGCOMM, 2012.

[47] A. Munir, I. A. Qazi, Z. A. Uzmi, A. Mushtaq, S. N. Ismail, M. S. Iqbal, and B. Khan. "Minimizing flow completion times in data centers," in INFOCOM, 2013.

[48] Y. Zhu, H. Eran, D. Firestone, C. Guo, M. Lipshteyn, Y. Liron, J. Padhye, S. Raindel, M. H. Yahia, and M. Zhang. "Congestion control for large-scale rdma deployments," in SIGCOMM, 2015.

[49] W. Bai, L. Chen, K. Chen, D. Han, C. Tian, and H. Wang. "Information-Agnostic Flow Scheduling for Commodity Data Centers," in NSDI, 2015.

[50] W. Bai, K. Chen, L. Chen, C. Kim, and H Wu. "Enabling ECN in Multi-Service Multi-Queue Data Centers," in CoNEXT, 2016.

[51] W. Bai, K. Chen, L. Chen, C. Kim, and H Wu. "Enabling ECN over Generic Packet Scheduling," in CoNEXT, 2016.

[52] J. Zhang, W. Bai, and K. Chen. “Enabling ECN for datacenter networks with RTT variations,” in CoNEXT, 2019.
[53] G. Zeng, W. Bai, G. Chen, K. Chen, D. Han, Y. Zhu, and L. Cui. “Combining ECN and RTT for Datacenter Transport,” in APNet, 2017.
[54] Y. Li, R. Miao, H. H. Liu, Y. Zhuang, F. Feng, L. Tang, Z. Cao, M. Zhang, F. Kelly, M. Alizadeh, and M. Yu. “HPCC: high precision congestion control,” in SIGCOMM. ACM, 2019, pp. 44–58.
[55] G. Zeng, W. Bai, G. Chen, K. Chen, D. Han, Y. Zhu, and L. Cui. “Congestion Control for Cross-Datacenter Networks,” in ICNP. IEEE, 2019, pp. 1-12.
[56] V. Dukic, S. Abdu J., B. Karlas, M. Owaida, C. Zhang, and A. SinglaIs. “Is advance knowledge of flow sizes a plausible assumption?” in NSDI, 2019, pp. 565-580.