PayDebt: Reduce Buffer Occupancy Under Bursty Traffic on Large Clusters

Kexin Liu, Chen Tian, Qingyue Wang, Yanqing Chen, Bingchuan Tian, Wenhao Sun, Ke Meng, Long Yan, Lei Han, Jie Fu, Wanchun Dou, and Guihai Chen

Abstract—The average/tail Flow Completion Times (FCTs) are critical to many datacenter applications. Congestion control plays a central role in optimizing FCT. Inappropriate congestion control can exacerbate buffer occupancy, thus hurting the flow performance. Our observations are that current approaches are too aggressive in injecting packets into underlying networks. Instead of handling buffer explosion afterward, we reduce buffer occupancy in the first place. We propose PayDebt, a novel and readily-deployable proactive congestion control protocol. At its heart, a *debt* mechanism provides bandwidth coordination between the already-buffered and the forthcoming packets. We evaluate PayDebt both in a testbed and large-scale simulations. The buffer occupancy can be decreased by up to 8.0×-35.9× compared to DCQCN and Homa.

Index Terms—Token, proactive congestion control, datacenter networks

1 Introduction

In a datacenter cluster [1], [2], [3], [4], [5], the network traffic represents the mix of its applications [6], [7], [8], [9]. Connecting tens of thousands of nodes, such a cluster usually simultaneously carries small and large flows. Small flows can be triggered by key-value stores [9], [10], Remote Procedure Calls (RPCs) [11], and application control messages. Large flows can be generated by various applications, including Hadoop/Spark shuffle [12], [13], [14], [15], [16], [17], data replication [18], and machine learning parameter updates [19], [20], [21]. The average/tail Flow Completion Times (FCTs) are critical to many applications. For example, a lower FCT directly contributes to a higher service throughput for RPC systems [11], [22], [23].

Congestion control plays a central role in optimizing FCT. Base Round-Trip Times (RTTs) in datacenters are relatively short (e.g., 10 μs), hence small flows are dominated by network queuing delay, and large flows are usually dominated by network goodput. State-of-the-art datacenter congestion control protocols can be classified into two categories: *reactive* and *proactive*. After first injecting a portion of traffic for each new flow, reactive protocols (e.g., DCTCP [24], DCQCN [25], HPCC [26], PINT [27],PowerTCP [28], Timely [29], Swift [30], and On-Ramp [31]) react to congestion signals from the underlying networks or hosts. For proactive protocols (e.g., ExpressPass [32]), a flow's sender transmits *scheduled* packets after receiving bandwidth allocation instructions (i.e., token packets) from the receiver. Instead of waiting for allocation in the first RTT, recent proactive protocols (e.g., Homa [33], NDP [34], pHost [35], and Aeolus [36]) transmit a portion of each new flow as *unscheduled* packets to improve small flows' performance. This paper also focuses on proactive protocols.

Inappropriate congestion control can exacerbate buffer occupancy, thus hurting the flow performance. Many datacenter networks are oversubscribed (Section 2.1) [37], [38], [39], [40], [41]. Meanwhile, datacenter traffic can be bursty with an on/off behavior [7], hence flows may quickly fill up switches' buffers at the network bottleneck. Large buffers add queuing delay and significantly increase small flows' average FCT. Even worse, buffer overflow causes packet drops and hurts flows' tail FCT. Existing drop remedies (i.e., timed out, fast retransmission [42], and selective drop [36]) are either hard to tune or compromise the performance of specific traffic patterns.

Current approaches are too aggressive in injecting packets into underlying networks. Ignoring already buffered packets in switches, a flow driven by an existing proactive protocol can still transmit scheduled packets. This lack of coordination between scheduled and unscheduled packet injections severely exacerbates buffer occupancy (Section 2).

Subsequently, we take one step back and ask: *instead of handling buffer explosion afterward, can we reduce buffer occupancy in the first place?* PayDebt is a novel proactive congestion control protocol to answer the question. At its heart, a *debt* mechanism provides bandwidth coordination between

- *Kexin Liu, Chen Tian, Qingyue Wang, Yanqing Chen, Bingchuan Tian, Wanchun Dou, and Guihai Chen are with the State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210093, China. E-mail: {kxliu, mg1933055}@smail.nju.edu.cn, {tianchen, douwc, gchen}@nju.edu.cn, yqchen19@outlook.com, bingchuantian@gmail.com.*
- *Wenhao Sun, Ke Meng, Long Yan, Lei Han, and Jie Fu are with the Huawei Technologies Co., Ltd, Shenzhen 518063, China. E-mail: {sam.sunwenhao, mengke6, yanlong20, phoebe.han, fujie}@huawei.com.*

Manuscript received 28 February 2022; revised 11 July 2022; accepted 12 August 2022. Date of publication 29 August 2022; date of current version 14 September 2022.
This work was supported in part by the Key-Area Research and Development Program of Guangdong Province under Grant 2020B0101390001, the National Natural Science Foundation of China under Grants 92067206, 62072228 and 61972222, the Fundamental Research Funds for the Central Universities, the Collaborative Innovation Center of Novel Software Technology and Industrialization and the Jiangsu Innovation and Entrepreneurship (Shuangchuang) Program.
(Corresponding authors: Chen Tian and Lei Han.)
Recommended for acceptance by K. Gopalan.
Digital Object Identifier no. 10.1109/TPDS.2022.3202504

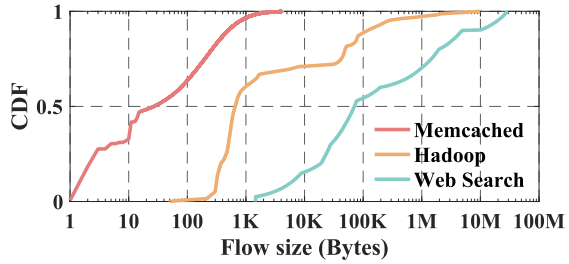


Fig. 1. Flow size distributions of typical workloads [33].

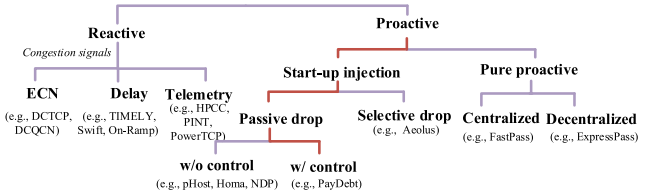


Fig. 2. Design space.

unscheduled and scheduled packets. This mechanism can minimize superfluous injections of scheduled packets (Section 3). Opposite to reactive congestion controls facing *data incast* problem, there is a *token incast* problem for proactive congestion control protocols, where several receivers send tokens to the same sender simultaneously. PayDebt solves it by applying token congestion control. In addition, a number of practical challenges are solved (Section 4). For a practical solution, PayDebt meets two design requirements as follows: (i) effective for a wide range of topologies and traffic patterns, and (ii) readily-deployable.

We implement a running PayDebt system with Linux hosts and commodity switches. We evaluate PayDebt both in a testbed and large-scale simulations (Section 5). The results demonstrate that PayDebt performs well in its targeted scenarios, i.e., mainstream oversubscribed networks. In other kinds of networks (e.g., non-blocking), PayDebt also demonstrates better or at least comparable performance than state of the art. The buffer occupancy can be decreased by up to 8.0×-35.9× compared to DCQCN and Homa. Consequently, compared to state-of-the-art approaches except for Homa (since it benefits from SRPT scheduling strategies), the average FCT of small flows can be decreased by up to 50.2% - 85.4%. For all flows, the average FCT can be decreased by up to 13.0% - 38.1%. The tail FCT can be 1.06× - 7.4× lower. At last, we conclude the paper (Section 7). *This work does not raise any ethical issues.*

2 Motivation

2.1 Background

Datacenter Network Topologies. Non-blocking networks have been proposed for years [43]. While, on the one hand, many operators tend to consider that full connectivity is rarely worthwhile due to traffic locality [44]. On the other hand, non-blocking networks are considered to be costly [38]. In practice, many datacenter clusters enforce a low degree of over-subscription [6], typically ranging from 8:1 to 2:1 [37], [38], [39], [40], [41].

Traffic Characteristics. Datacenter traffic is usually bursty with an on/off behavior [7]. Microbursts in today's datacenters [7], [45], [46], [47], [48] can quickly accumulate packets at the network bottleneck. Given that many network topologies are oversubscribed, the network bottleneck usually occurs at the fan-in point of topologies, e.g., ToR switches or spine switches [8], [49], [50].

Datacenter Workloads. Fig. 1 shows the flow size distributions of widely accepted datacenter workloads (drawn from existing work [33]) used to evaluate different approaches. Data traffic in Memcached is composed of small flows, where more than 85% of flows are smaller than 1KB. In Hadoop and Web search, large flows are mixed with small flows where a small ratio of large flows contributes most bytes.

2.2 Existing Congestion Control Protocols

Scheduled packets denote packets that are transmitted according to rate adjustment (i.e., in reactive protocols) or bandwidth allocation (i.e., in proactive protocols). And *unscheduled* packets denote those transmit upon new flow arrives, without waiting for rate adjustment or bandwidth allocation.

Reactive Approaches. We revisit state-of-the-art congestion control protocols shown in Fig. 2. For reactive designs, a sender first sends a number of start-up packets and then reacts to congestion signals (i.e., ECN bits [24], [25], [51], network delay [29], [30], [31], [52], or switch measurement [26], [27]). Only flows that last longer than one RTT can react to congestion signals; hence, these approaches can deplete switch buffers when bursty flow arrivals occur. Both DCQCN [25] and HPCC [26] send a new flow's *unscheduled* packets at line rate. DCQCN controls its sending rate based on ECN. Using in-network telemetry (INT), an HPCC sender can obtain precise link load information and calculate a large flow's rate. However, it can still lead to inaccurate bandwidth allocation due to stale measurements (see Appendix in [53]). PINT [27] is similar to HPCC, except that it uses probabilistic INT to reduce the cost of INT. PowerTCP [28] combines both absolute network state (e.g., queue length or RTT) and its variations through INT. Both DCQCN and HPCC enable the Priority Flow Control (PFC) feature of Enhanced Ethernet to prevent packet drops. The PFC mechanism faces severe scalability challenges because of PFC storm, Head-Of-Line (HOL) blocking, and deadlock [54], [55]. It also hurts small flows' FCTs. A transport protocol without the need for PFC is more preferable to industry [56]. Swift [30] is an RTT-based solution, leveraging an additional congestion window also to track endpoint congestion besides in-network congestion. On-Ramp [31] handles the transient state of the network by leveraging accurate measurements of one-way delay and leaves the equilibrium state for current reactive approaches. For that PayDebt is a proactive protocol, these efforts are orthogonal to PayDebt.

Proactive Approaches. A typical pure proactive design such as ExpressPass [32] proceeds as follows. On the arrival of a new flow, a control packet is sent from the sender to the receiver as a notification. A receiver schedules the tokens (i.e., *credit* in ExpressPass, *PULL* packet in NDP [34], and *grant* packet in Homa [33], [57]) to allocate bandwidth for different flows. Upon the reception of a token, the sender can correspondingly send a full-length *scheduled* packet.

ExpressPass generates tokens based on a rate. Each receiver automatically generates tokens based on a sending

rate to fetch the remaining scheduled packets for each flow. Congestion points (i.e., receivers and switches) limit tokens' bandwidth to around 5% of link bandwidth to guarantee drop-less of data in the reverse direction. In turn, token packets are dropped at the network bottleneck by setting the token queue length to a relatively small value (e.g., eight-token length). This rate is calculated by the length of a token *versus* a full-length data packet (i.e., $\frac{84}{1538+84} \approx 5\%$). Thus, ExpressPass can achieve low queuing latency and zero data packet drop together. The main problem of a pure proactive protocol is its waste of the first RTT. For workloads where most flows are smaller than one Bandwidth-Delay Product (BDP) worth of bytes (e.g., the Memcached workload), their FCTs could be tripled [36]. In addition, it could result in the last-RTT waste of tokens (see discussion later in Section 4.1.1).

For most recent proactive protocols (i.e., pHost [35], NDP [34], and Homa [33]), a sender can send up to one BDP worth of *unscheduled* data for a new flow. The rest packets (if there are any) are scheduled by the receiver. A non-blocking network core is sometimes assumed to provide full bi-sectional bandwidth. Thus, the receiver and the sender are two scheduling points for each flow. In general, these protocols have reasonable performance, but flaws exist at the same time (verified in our evaluations).

Homa [33] mainly targets RPC-like scenarios, where most bytes are from small flows with several hundreds/kilos of bytes. Homa uses SRPT (shortest remaining processing time) scheduling strategies on senders, prioritizing smaller flows over larger ones. In addition, in-network priority queues are leveraged to enforce priorities among unscheduled/scheduled packets according to their flows' sizes. It mitigates the head-of-line (HOL) blocking caused by queuing, i.e., when a short flow gets stuck behind a long flow in the same queue. Homa assumes that packet drops are rare, so it relies on a timeout mechanism (e.g., a few milliseconds) to retransmit. NDP [34] trims the payload of a dropped packet and uses the header to precisely inform the receiver about the drop. A dropped data packet needs to be *pulled* by the receiver again. By limiting the length of packet queues, NDP can constrain one-way delay. As a clean-slate solution, NDP significantly changes switch behaviors; hence it is not readily-deployable. This paper also focuses on proactive protocols.

2.3 Buffer Occupancy Should be Controlled

A large buffer, by adding queuing delay, is a curse to small flows' performance. If the situation exacerbates and the buffer overflows, the packets are dropped. Packet drops can severely hurt small flows. A single packet drop (followed by detection/retransmission procedures) can push a small flow's completion time to tens or even hundreds of times the base RTT. Furthermore, in private discussions with us, some leading Network Interface Controller (NIC) providers raise their practical concerns: a large number of packet drops pose challenges to RDMA NICs, including (but not limited to) a tremendous buffering demand for re-ordered packets and complicated receive processing logic.

The number of in-network priority queues is not always adequate to handle queuing delays. Homa leverages eight in-network priority queues, which could significantly mitigate the queuing delay of small flows by prioritizing small flows over large flows. Generally, traffic from different applications (e.g., storage, computing, outbound applications, *etc.*) needs to be isolated by different queues [38], [58]. A single application can not use up priority queues on switches. The effectiveness of in-network prioritization could be compromised as the number of available in-network priority queues shrinks. There is a greater possibility that small flows are queuing behind large flows when the buffer occupancy is relatively large (Section 5). In this article, we use Homa as our upper bound, and we mainly focus on practical scenarios where priority queues should be used conservatively.

Existing drop-remedies are hard to tune. Homa relies on a rough timeout mechanism to retransmit. As demonstrated by previous work [36], a large timeout value (i.e., a few milliseconds) results in a large tail latency. In contrast, a small timeout value (i.e., tens of microseconds) results in redundant retransmission, downgrading goodput.

NDP trims the payload of data packets when the buffer exceeds a small threshold and uses packet headers to notify receivers for fast retransmission. Its shallow buffer setting causes packet drops much earlier. Control packets (i.e., cut-payload headers) can also get dropped. Too many control packets and retransmission waste bandwidth under heavy congestion scenarios. Our evaluations (Section 5) demonstrate that these control packets could overflow their belonging queues when a large incast is faced, which results in performance collapse.

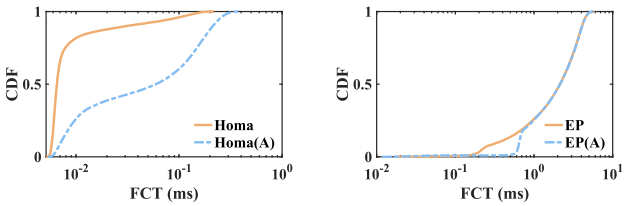
Selective drop is not a cure. Aeolus [36] is a state-of-the-art patch focusing on solving start-up injection challenges for existing proactive protocols. Aeolus leverages the Active Queue Management (AQM) feature of commodity switches to achieve a selective drop of unscheduled packets. They are transmitted blindly at line rate and are dropped when buffer occupancy exceeds a relatively small threshold (i.e., 8KB). Thus, scheduled packets are guaranteed to be delivered without loss.

The selective drop of unscheduled packets may unnecessarily increase small flows' FCT when a workload contains a large fraction of small flows. We demonstrate this by conducting an NS-3 simulation, where workloads are composed with small flows with an 80% load [33] (i.e., Memcached). Flow arrival intervals follow the Poisson process. For Homa, the same non-blocking topology in its paper is used. For ExpressPass, links connected to core switches are decreased to 10 Gbps to construct an oversubscribed topology with a 4:1 over-subscription ratio (see Section 5 for details).

Fig. 3 shows the results. For Homa/Homa(A), a significant performance downgrade has been observed. The average/99th-tail FCTs are increased by 5.5×/2.0× compared with pure Homa, respectively. A large ratio (i.e., 1.38%) of packet drops leads to a large number of retransmissions. Although Aeolus makes full use of the first RTT, small flows' performance is still hurt. With the large amount (i.e., 16.5%) of packet loss, the performance of 17.4% flows significantly downgrades, and EP(A) quickly back-offs to pure ExpressPass.

2.4 State-of-the-Art Injections are Aggressive

In prior works, transmission decisions of scheduled packets are made independently of in-network buffered packets' current status. For illustration, a strawman protocol (i.e.,



(a) Homa w/ and w/o Aeolus. (b) ExpressPass w/ and w/o Aeolus.

Fig. 3. Selective drop hurts FCTs of small flows. Homa(A) denotes Homa-patched-with-Aeolus, and EP/EP(A) denotes ExpressPass/ExpressPass-patched-with-Aeolus, respectively.

simplified Homa without priority queues) is depicted in the first row of Fig. 4. Note that the packet drop in the following discussions is irrelevant to priority usage. Here, we assume three hosts, S0, S1, and S2 send to a host R via a Top-of-Rack (ToR) switch. For clarity, the base RTT is set to two-time units (see an extended version where the base RTT is four-time units [53]). The transmission delays are ignored. The link delay between a sender and the switch is one time unit, while the link delay between the switch and the receiver is negligible. Thus, a packet can traverse one link per time unit. The switch-to-receiver output queue status is demonstrated, which denotes the exact buffer state of the time on its right. The BDP and queue length are set to 2 MTU and 4 MTU, respectively.

Three flows A/B/C sent from S0/S1/S2 arrive at the system at $T0/T2/T4$, respectively. Each flow consists of 6 MTU packets, i.e., two unscheduled and four scheduled. $Au0/At0/As0$ are used to represent the first unscheduled/token/scheduled packets of flow A. At $T0$, flow A arrives at S0, and $Au0$ is transmitted. After receiving $Au0$ at $T1$, R generates a token $At0$. $At0$ is forwarded to S0 and arrives at $T2$. Triggered by token $At0$, S0 sends a scheduled packet $As0$, which arrives at the switch at $T3$, along with an unscheduled packet $Bu0$ sent by S1. The same thing happens to $Au1$, $Bu0$, $Bu1$, ..., *etc.*, and generates tokens $At1$, $Bt0$, $Bt1$, ..., *etc.*, respectively. The length of the queue is increasing, which finally leads to unscheduled packet $Cu1$ drop at $T6$.

Ignoring bandwidth consumed by unscheduled packets, existing protocols send tokens to allocate full bandwidth for the following scheduled packets. Therefore, scheduled packets are unnecessarily injected into the underlying network and accumulated in switches. The queue length never decreases as long as scheduled packets continually arrive.

3 PayDebt Intuition

Unscheduled packets are the cause of network bursts. As the name of unscheduled packets suggests, they consume bandwidth without reservation. Inspired by the observations in Section 2.4, we need a coordination mechanism between unscheduled and scheduled packets so they can harmoniously share the bandwidth.

Insight. Although an unscheduled packet can not be controlled until it is transmitted, it could *announce* its used bandwidth *after its transmission*. This announcement could work as a back pressure to the underlying network when congestion exists. Network bottlenecks along the unscheduled packet's path can subtract the same amount of bandwidth from their following bandwidth allocation to scheduled packets, i.e., tokens can only reserve the rest of the bandwidth. Then unnecessarily injected scheduled packets can be suppressed. In this way, scheduled packets could be aware of the exact bandwidth usage of unscheduled packets and back-off accurately.

Challenges. For a practical solution to this ambition, we list two design requirements as follows:

- *Effective for a wide range of topologies and traffic patterns.* Traffic patterns can vary significantly across both time domains and different regions. This work focuses on heavy load scenarios in typical oversubscribed networks. Meanwhile, the performance of other scenarios (e.g., non-blocking fabrics) should not be compromised.
- *Readily-deployable.* The solution must work with existing commodity switches in datacenter networks. Bandwidth coordination must be performed at the data-plane instead of the control-plane.

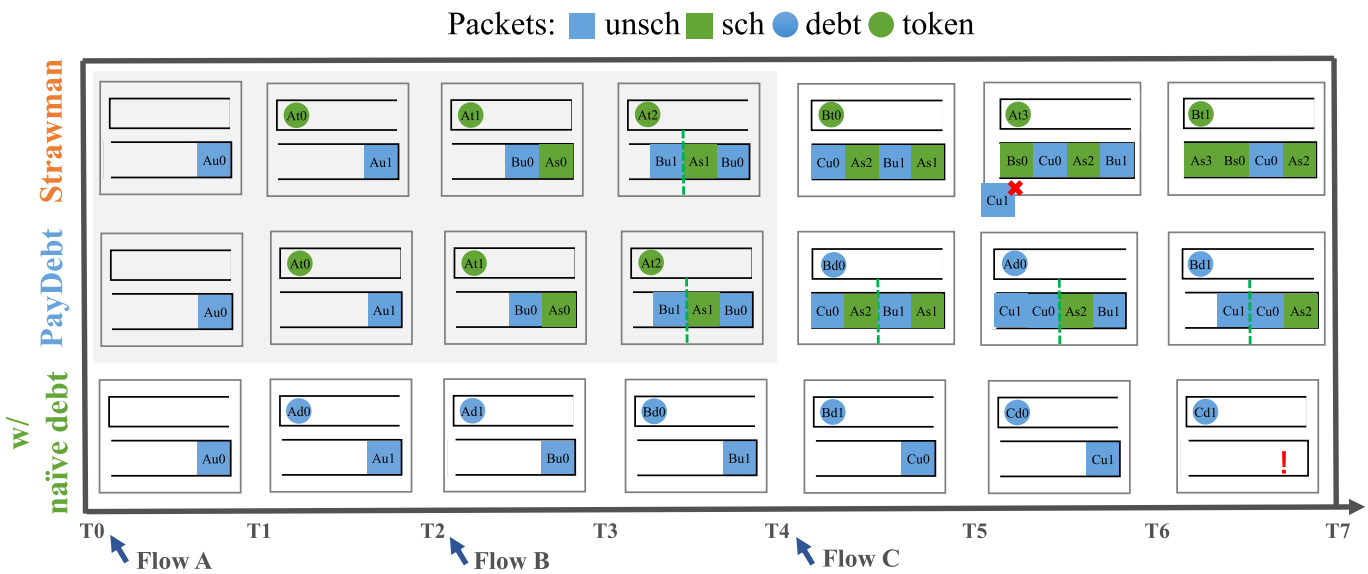


Fig. 4. How debt works, illustrated by a strawman receiver-driven example. Here the x-axis stands for time units, and the y-axis stands for three different variants of the strawman protocol.

Fig. 5. PayDebt framework.

3.1 Pay Your Debt

We propose *debts*, a special kind of control packets sent from receivers to senders, for coordination between unscheduled and scheduled packets. Let us say the token mechanism works like a debit card, where links deposit (i.e., reserve) bandwidth before a scheduled packet uses it. The debt mechanism works like a credit card, where an unscheduled packet spends (i.e., consume) bandwidth first and links reclaim it back later.

In the third line of Fig. 4, we illustrate how debt works by adding a naïve debt mechanism to the strawman protocol. After receiving $Au0$, debt $Ad0$ is sent instead of $At0$. When a debt packet is received, the sender does not send a scheduled packet. The same thing happens to $Au1$, $Bu0$, ..., *etc.*, and replies debt $Ad1$, $Bd0$, ..., *etc.*, respectively. Therefore, the queue length does not increase, making unscheduled packets $Cu1$ successfully received by the receiver. After all the debts are transmitted, the remaining tokens can be sent to allocate bandwidth for the scheduled packets.

Thus, unscheduled packets can pay back pre-consumed bandwidth in a readily-deployable way. Following scheduled packets' transmission is delayed to avoid buffer built-up. Likewise, if a scheduled packet collides with an unscheduled packet already buffered in a congestion point, it can drain from the bottleneck when a corresponding debt takes effect.

4 PayDebt Design

The debt mechanism, which cannot be simply added to current proactive approaches, should be co-designed with tokens. In this section, we propose PayDebt, a novel proactive congestion control protocol. As shown in Fig. 5, there are three major components of PayDebt: sender logic, receiver logic, and switch configuration. There are many remaining challenges, e.g., how tokens are generated? When should debts and tokens be sent? In this part, we present the debt designs in detail (Section 4.1.1). Its limitations are discussed (Section 4.1.2). We introduce the token incast problem and token congestion control designs (Section 4.1.3), followed by the receiver logic, the switch configuration, and miscellaneous detailed designs (Section 4.4).

4.1 Receiver Logic

4.1.1 The Debt Mechanism

Debt Generation. PayDebt uses a data-driven mechanism to generate debts. The meaning behind debt packets is to pay back the unallocated bandwidth used by unscheduled packets. Therefore, a debt packet is generated if an unscheduled packet is received.

Token Generation. Tokens can be generated based on a rate (i.e., rate-based) or when a data packet is received (i.e., data-driven). PayDebt uses a data-driven mechanism to generate tokens. When a data packet is received, a token packet is generated if necessary.

PayDebt generates tokens based on a data-driven manner for that it is more compatible with the unscheduled-driven debt mechanism. With a rate-based mechanism, the massive number of tokens could reduce the effect of debts. At the same time, data-driven token generation can provide better performance on tail latency. With a data-driven token mechanism, there is no necessity for switches to set a small queue for tokens and drop them actively since the number of tokens is under control. Hence, the token loss is assumed to be rare. PayDebt's receiver can transmit the exact number of tokens needed by the sender, without token waste. While the rate-based mechanism has a severe problem, i.e., *last-RTT waste of tokens*. The token queue is set to a small value and tokens can be dropped at the network bottleneck to reduce the excessive tokens (Section 2.2). A receiver cannot stop sending tokens until receiving the last data packet or the notification from the sender. At least one RTT worth of tokens sent by a receiver can never trigger any data packet. These useless tokens compete for bandwidth with other useful tokens at both receivers' sides and networks, resulting in redundant drops of useful tokens. It could downgrade the performance of small flows and hurt bandwidth utilization. For workloads comprised of a large fraction of small flows, FCTs could be prolonged (Section 5).

In PayDebt, if the remaining data is larger than one BDP, PayDebt's sender sets a `token_request` flag in the packet header. When a data packet with a `token_request` flag is received, a corresponding token packet is generated (see online Appendix [53] for example illustration). With this design, no excessive tokens will be generated. Meanwhile, the flow size is not required to be known a-prior. Instead, the sender needs to check whether the appending data of a flow exceeds the BDP when a scheduled packet is transmitted.

Bandwidth Coordination. PayDebt uses symmetric paths and sets a reverse-path rate limiter (supported by IEEE 802.1Qaz) to achieve bandwidth coordination. Rate-limiter can be supported by mainstream datacenter switches [59], [60], [61]. A debt has precisely the same size as that of a token (i.e., 84 bytes as that in ExpressPass). Debts and tokens are rate-limited together to 5% of bandwidth in each port of switches and hosts (Section 2.2).

Rate-limiters on tokens ensure that bandwidth consumed by data packets never exceed 95% of the network bottleneck bandwidth, making PayDebt robust under oversubscribed networks. Moreover, by rate-limiting debts and tokens as a whole, the delivery of tokens can be delayed when debts share the same bottleneck with tokens. In this way, the transmission of scheduled packets can be delayed precisely; therefore, the buffer occupancy can be reduced. It is worth noticing that even if unscheduled and scheduled packets are from different sender hosts, debts can still take effects, i.e., compete with tokens sent to other sender hosts.

Send Debt/Token at the Right Time. A question is whether it is necessary to send debt upon generation, i.e., to pay back any bandwidth used by unscheduled packets as soon as possible. Let us revisit the naïve debt approach in Fig. 4. At $T6$, if no unscheduled packet arrives at the switch, bandwidth is wasted. Consider a simple example where there is only one flow transmitting packets, the bandwidth used by its unscheduled packets does not interfere with others.

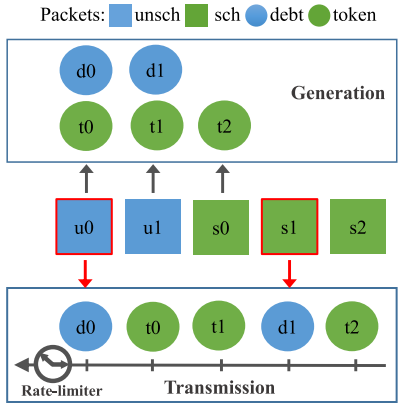


Fig. 6. Debt/token transmission on end-hosts.

Prioritizing the transmission of debts over tokens actually increases its FCT.

Debts should be sent when congestion exists. PayDebt leverages ECN to identify congestion. Data packets are ECN-marked when the queue occupancy exceeds a given threshold K_d. When an unscheduled packet with ECN-marking is received, the receiver transmits a debt immediately. Otherwise, this debt is transmitted until either a following scheduled packet is ECN-marked or the flow finishes. This is because a flow's unscheduled packets which compete for bandwidth with other flows can escape from ECN-marked when leaving the switch queue. Its following scheduled packets are then responsible for triggering the transmission of remaining debts, i.e., pay back bandwidth. Debts are prioritized to be sent over tokens at end-hosts, i.e., when debts and tokens both exist, send debts first. It can pay back the bandwidth used by unscheduled packets when necessary.

We use the second line of Fig. 4 for illustration. Debt is transmitted rather than token when the queue length exceeds a threshold, i.e., 2 MTU. When $Bu0$ leaves the switch, the queue length exceeds the threshold, and $Bu0$ is ECN-marked. Therefore, when $Bu0$ is received, $Bd0$ is sent instead of $Bt0$. The same thing happens to $As1$, $Bu1$, ..., *etc.*, and replying debt $Ad0$, $Bd1$, ..., *etc.* In this way, unscheduled packet $Cu1$ is successfully received by the receiver; meanwhile, bandwidth is not wasted at $T6$. Debt is sent as long as the queue length exceeds the given threshold; therefore, the queue length continues to decrease at $T7$. The queue length can be reduced to zero after debts finish transmitting.

Fig. 6 summarizes the generation and transmission logic of debts/tokens, where $u0$ and $s1$, which have red borders, are ECN-marked. The arrival of unscheduled/scheduled packets triggers the generation of debt/token. Then, PayDebt's receiver paces debt/token waiting for transmission at 5% of port bandwidth. The network congestion determines the transmission order of debt/token. If the received data is ECN-marked, send a queuing debt first. Otherwise, send a token instead.

4.1.2 Debt Discussions

Effectiveness of Debt. Now we discuss the effects of debt in reducing buffer occupancy. In Fig. 4, although the topology fan-in ratio is 3:1, the maximum workload (not topology) fan-in ratio is only 2:1, i.e., at most two data packets per time unit. The debt mechanism can reduce the input-output ratio to 1:1. For oversubscribed networks with uniformly distributed workloads, the maximum workload fan-in ratio is at most its topology fan-in ratio. Therefore, for mainstream oversubscribed networks with topology fan-in ratio, e.g., 2:1, 4:1, debt performs well.

When tokens are generated based on a rate, the last-RTT waste of tokens can decrease the buffer occupancy to a certain extent. However, it cannot coordinate the usage of unscheduled and scheduled packets on time as the debt mechanism in PayDebt can do (Section 5.5).

Other Scenarios. In non-blocking networks, workload fan-in could occur with imperfect load balance, which results in in-network congestion. Debt has a positive effect of reducing the buffer built-up, as demonstrated in our evaluations (Section 5.3).

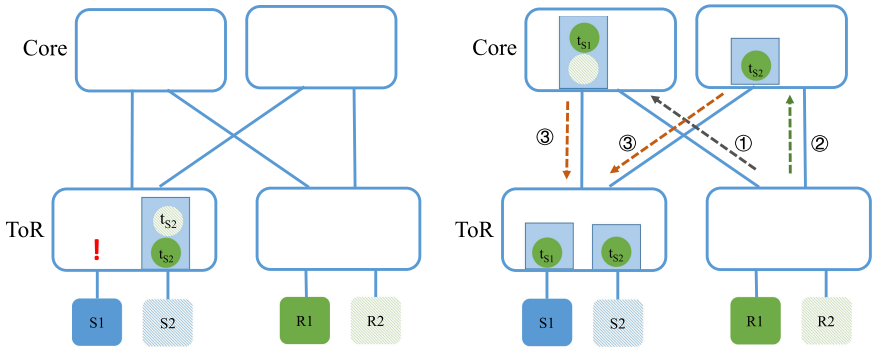
4.1.3 Token Incast and Congestion Control

Token Incast Problem. Tokens are designed to pace the transmission of scheduled packets. A receiver cannot know whether other receivers are requesting data from the same sender simultaneously. Hene, opposite to reactive congestion controls facing *data incast* problem, there is a *token incast* problem for PayDebt (as well as proactive protocols, e.g., ExpressPass and NDP). When several receivers send tokens to a single sender simultaneously, token incast occurs. Different from data incast facing buffer overflow, token incast could result in buffering of tokens and under-utilization of bandwidth. Fig. 7a illustrates the token incast problem, where two receivers R_1 and R_2 both send tokens to a sender S_2. Tokens queue up at the last-hop switch connected to S_2 while S_1 receives no tokens. This scenario wastes the bandwidth of S_1 and hurts performance.

Token Queuing at In-Network Points. In addition, token queuing could occur at in-network points when competing for the bandwidth. Token queuing could result in different RTT values of tokens. These tokens may arrive at different senders at the same time, then corresponding scheduled packets will be sent simultaneously, which in turn results in transmission collision of scheduled packets. Fig. 7b illustrates the scenarios where R_1 sends tokens t_{s1} and t_{s2} to S_1 and S_2 in order. Token t_{s1} endures one-packet queuing delay while t_{s2} does not. Because of the different queuing delays on core switches, these two tokens arrive at senders simultaneously.

Token Congestion Control. PayDebt uses token congestion control to minimize the impact of a token incast and reduce the token queue. A token is ECN-marked if the tokens' queue length in a switch port exceeds a given threshold K_t. A sender forwards the congestion notification back to the receiver by attaching it with a data packet. To distinguish the congestion notification from ECN-marking of the data packet, PayDebt's header should leverage a dedicated bit to carry it. The receiver then updates the maximum inflight tokens, i.e., limits the transmission of tokens to the sender. Compared with the traditional ECN-marking scheme, data packets in PayDebt can carry the congestion information in token packets.

Algorithm 1 presents the pseudo code of the token congestion control. Each receiver maintains `maxInflight`, the



(a) Token incast problem. (b) Tokens with different RTTs.

Fig. 7. It is necessary to leverage token congestion control. Sequence numbers with circles of (b) denote the time order.

maximum in-flight tokens that can be sent and w, a weighted parameter ($0 < w \leq 0.5$) to adjust it. Similar to Additive Increase Multiplicative Decrease (AIMD), it updates maximum in-flight tokens. A token is ECN-marked in switches if the queue length of the token exceeds a given threshold K_t. A PayDebt sender conveys the congestion notification back to the receiver by data packets. When data contains congestion notification arrives the receiver, the receiver decreases w and `maxInflight` (Line 7-9). w is decreased by half. `maxInflight` is decreased by multiple `(w+0.5)`. Otherwise, the receiver increases w and `maxInflight` (Line 11-12). `maxInflight` is increased to a weighted average of `(1-w)` × `maxInflight` and `w` ∗ `BDP` , where `BDP` is the value of base BDP. The value of w estimates the congestion degree of the token queue. Essentially, a value of w closes to 0 indicates a high degree of token congestion. And a value closes to 1 indicates a low degree of token congestion. To avoid overreaction, `maxInflight` is updated per RTT (Line 6).

Algorithm 1. Update Max Inflight Tokens per Flow

Input: parameter $w_{min} \leftarrow 0.1, w_{max} \leftarrow 0.5$
1: $w \leftarrow w_{init}$ // default 0.5
2: $t \leftarrow RTT$ // update timer
3: $maxInflight \leftarrow BDP$ // max in-flight tokens
4:
5: **when** *a scheduled packet is received* **do** ▷ EVENT
6: **if** *t has passed since last update* **then**
7: **if** *scheduled packet is carried with ECN of token* **then**
8: $w = max(w/2, w_{min})$
9: $maxInflight = (w + 0.5) \times maxInflight$
10: **else**
11: $w = (w + w_{max})/2$
12: $maxInflight = (1 - w) \times maxInflight + w \times BDP$

4.1.4 Token Congestion Control Discussions

We regard PayDebt as a proactive protocol. Although PayDebt is inspired by reactive protocols to handle congestion of tokens and unscheduled packets, its core logic follows a proactive way. It schedules the transmission of tokens to allocate bandwidth for scheduled packets, avoiding congestion caused by scheduled packets. The insight behind a proactive protocol is letting tokens queuing instead of scheduled packets. Hence, PayDebt leverages ECN marking on tokens to reduce the queuing of token packets. In addition, the bandwidth used by unscheduled packets is unallocated, hence the congestion induced by them is relieved by debts.

4.2 Sender Logic

PayDebt adopts a simple start-up injection control at the sender hosts. All flows are initialized with one BDP of unscheduled packets. When a sender receives a debt packet, do nothing. When a sender receives a token packet, a scheduled packet is sent to reply to the token.

4.3 Switch Configuration

PayDebt's switch uses a rate limiter to rate limit debts and tokens as a whole, i.e., sharing the 5% bandwidth (Section 4.1.1). Therefore, debts compete with tokens to share the 5% bandwidth at the network bottleneck. And bandwidth consumed by scheduled packets will not exceed the network bottleneck, i.e., incast point or network fan-in point. The switch uses two queues, one for data packets, the other one for debts/tokens.

4.4 Miscellaneous Detailed Designs

In Sections 4.1.1 and 4.1.3, two essential functions on receivers are introduced. We summarize sender and switch logic in this section.

Handling Tiny Flows. In some scenarios, most flows are smaller even than one MTU (e.g., the Memcached workload), raising a possible *fractional debt* problem. If the receiver generates a debt upon receiving every small unscheduled packet, the bandwidth can be wasted. In PayDebt, the sender host maintains a `debt_to_receive` counter. Each time a sender sends an unscheduled packet, `debt_to_receive` is increased by its size. When its value reaches one MTU, the `debt_request` flag bit is set in the unscheduled packet. The receiver generates a debt if the `debt_request` bit is true. This feature avoids transmissions of unnecessary debts.

Handling (Rare) Packet Loss. PayDebt significantly reduces buffer occupancy; thus, we expect packet loss to be rare. In switches, relatively large buffer size is reserved for debt/token queues. Occasional data/debt/token loss can be promptly detected by Packet Sequence Numbers (PSNs).

There could be extraordinary cases where PSNs cannot detect packet loss, such as the loss of a flow's last packet or even the loss of an entire flow. These scenarios are detected via minimum-sized probe packets. A probe packet

is transmitted when either timeout is triggered or at the end of a flow. The probe packet uses the same priority as data packets; therefore, it will not arrive out-of-order. Each probe is attached with the PSN of the last data packet and the PSN of the last received token packets. When the probe is received, the receiver can check the sequence of un-received packets and generate the tokens with corresponding PSN to inform the sender, i.e., the same logic as that of re-ordered data packets are received. Then, the sender can retransmit corresponding data packets.

Handling Incast. Incast could happen when multiple senders transmit data simultaneously to the same receiver. This results in buffer built-up at the incast point. PayDebt bounds the maximum inflight unscheduled packets of a sender-receiver host pair. In this way, the incast scale is significantly reduced. The maximum buffer occupancy is determined by topology rather than proportional to the incast flows.

5 Evaluation

In this section, we use testbed experiments and large-scale NS3 simulations to evaluate PayDebt. We compare PayDebt with state-of-the-art datacenter transport protocols such as Homa, Homa(A), ExpressPass, EP(A), NDP, DCQCN, and HPCC. Similar to HPCC [26], a sending window is added to DCQCN to limit its in-flight data packets. The author-contributed simulation codes, if available, are used in our evaluations [62], [63], [64]. As for Aeolus, we reproduced the results in Aeolus' paper and communicated with the authors to ensure that our implementation is correct. Besides performance comparison, we also validate major design points and parameter selection.

Homa leverages SRPT scheduling and priority queues to benefit small flows (Section 2.2). Hence, in our evaluations, we choose Homa (8q, i.e., 8 priority queues) as our upper bound. And we mainly focus on practical scenarios where priority queues should be used conservatively. Therefore, we also use a variant of Homa (2q) to compare PayDebt with Homa's other design points. Although Homa is not designed for oversubscribed topology, it performs reasonably well in many such scenarios. Therefore we also compare it with PayDebt under oversubscribed topologies. NDP is designed for non-blocking networks. Under oversubscribed networks, it suffers from a large amount of packet loss and can even result in the drop of control packets. Therefore, we only compare it with PayDebt in non-blocking topologies.

In summary, PayDebt reduces buffer occupancy in typical oversubscribed networks under Poisson arrival scenarios, without a significant packet drop or bandwidth waste, therefore speeding up FCT. PayDebt's performance is at least comparable to state of the art under non-blocking topologies.

Workloads. The flow size distributions of Memcached [9], [10], Hadoop [6], and Web search [24] are shown in Fig. 1. Unless otherwise specified, we generate flows following a Poisson arrival process with a load of 0.8, i.e., the same way as Homa. A wide range of network loads is also used to investigate PayDebt's performance (Section 5.3). Incast traffic is also generated for further evaluation.

Parameters. For PayDebt evaluations, we have a set of default settings. Here, $K_d = 1 * \text{BDP}$ and $K_t = 1 * \text{BDP}$. A dedicated subsection later discusses why these values are used (Section 5.6). Round-Robin (RR) is used for sender/receiver scheduling. For that RR is commonly used in industry as it is easy to implement. In addition, flow sizes are not required to be known a-prior when RR is used. We also evaluate PayDebt's performance under SRPT strategy and eight priority queues [53]. Results show that PayDebt achieves comparable performance compared to Homa.

Metrics. We have three major performance metrics: (i) maximum switch buffer usage, (ii) average/99th-tail FCTs, and (iii) packet drop ratio.[1]

5.1 Testbed Experiments

Prototype Implementation. We implement PayDebt in BESS [65] with about 2500 lines of C++ code. Built on DPDK [66], BESS allows us to run PayDebt entirely in userspace and bypass the kernel. In our testbed, each server has two Intel Xeon E5-2650 2.2GHz 12-core CPUs, 256 GB RAM, one 10 Gbps Intel X710 NIC, and OS of Ubuntu 18.04 LTS with 4.15.0 Linux kernel. For comparison, we port Homa open-source codes [67], [68] to the same platform. Due to time constraints, we have not implemented other approaches yet. We are currently in the process of implementing PayDebt with a vendor-specific smart NIC so that the prototype can support 100 Gbps networks.

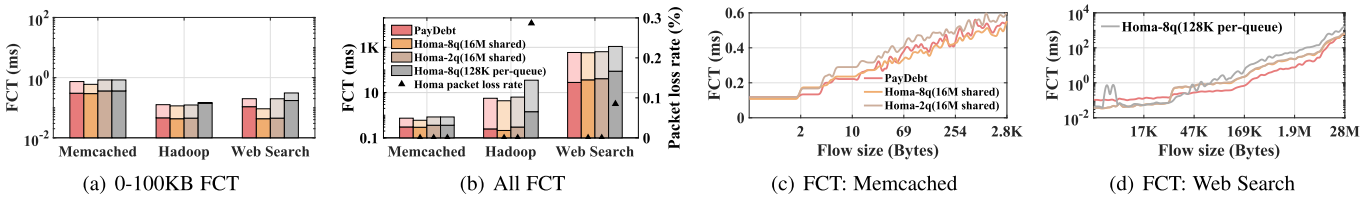
Topology. The testbed includes two commercial ToR switches connected via a 10 Gbps link. Each switch connects two servers via 10 Gbps links. This topology is a single-bottleneck dumbbell topology with a 2:1 over-subscription ratio. The base RTT is 11 μs.

There are two scenarios: *drop-less* and *drop*. In the drop-less scenario, all 16MB shared switch buffer is used. No packet is dropped for both approaches. In the drop scenario, we change the shared buffer mode to 128 KB per-queue buffer to produce packet drops to evaluate PayDebt and Homa.

Drop-Less Scenario. We evaluate PayDebt and Homa with all three workloads. The results are shown in Figs. 8a and 8b. As shown in Fig. 8, Homa achieves good performance, benefiting from SRPT scheduling and in-network priority queues. PayDebt achieves comparable performance under Memcached and Hadoop. A relatively smaller buffer occupancy of PayDebt is beneficial. PayDebt does not allocate in-network priority queues for unscheduled packets, for Web search which mixed with small and large flows, FCTs for small flows can be prolonged. While PayDebt reduces the FCTs for medium and large flows, as shown in Fig. 8d.

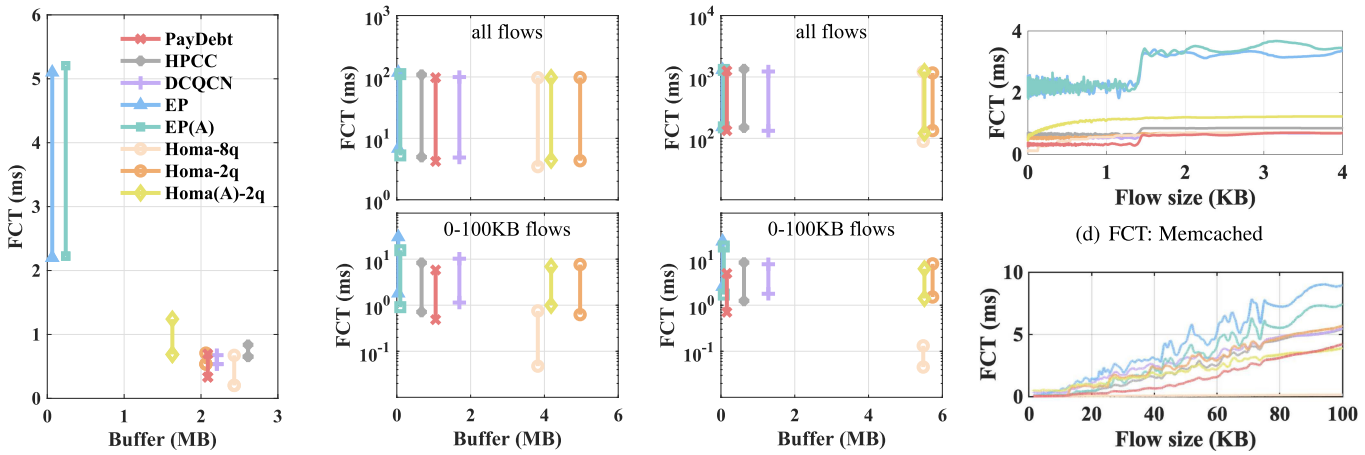
When Homa uses only two priority queues, its performance is downgraded. Homa is designed for non-blocking networks, and it does not address congestion in the core. In an oversubscribed topology, the aggressive injection of unscheduled packets can accumulate packets in the fan-in points. In-network prioritization is not always a cure to buffer built-up. When the number of in-network priority queues is restricted to two, the HOL-blocking problem (i.e.,

1. A result point is an average of over ten runs. Usually, these runs show similar outcomes, and a point's standard deviation is relatively small. We omit deviations for clarity.



(a) 0-100KB FCT (b) All FCT (c) FCT: Memcached (d) FCT: Web Search

Fig. 8. Testbed results with Poisson workloads. The deep/light color of (a) and (b) represents the average/99th-tail value for each bar, respectively. The left/right y-label of (b) is the FCT/the drop ratio (triangles in the figures). The breakdown FCT by message size of Memcached and Web search are shown in (c) and (d).



(a) Memcached (b) Hadoop (c) Web Search (d) FCT: Memcached (e) FCT: Web Search

Fig. 9. Poisson workloads in 4:1 oversubscribed topology. The two points on each bar represent the average FCT and the 99th-tail latency, respectively. (a) has only one subgraph because Memcached is almost made up of flows smaller than 100KB. The upper/lower subgraphs of (b) and (c) depict the average/99th tail FCT of all/0-100KB flows. Note that the y-axis is a log scale for Hadoop and Web search. (d) and (e) decompose the flows' FCT of Memcached and Web search.

small flows get stuck behind large flows) could intensify when buffer increases.

Drop Scenario. PayDebt gains benefit from reducing buffer occupancy. With a small per-queue 128 KB buffer, PayDebt keeps the same performance due to zero packet drop. With Memcached, Homa's FCT is almost the same as that in the drop-less scenario since there is rarely any packet drop. While with Hadoop and Web search, Homa suffers a packet drop ratio of 0.28% and 0.10%, respectively. As shown in Fig. 8b, its performance significantly deteriorates. The breakdown by message size of Web search is shown in Fig. 8d. Suffering from timeout retransmission, small flows' FCTs are increased by an order of magnitude. Compared with Homa, the average FCT of small flows is reduced by 63.5% and 37.6% under Hadoop and Web search, respectively. The tail latency is 3.1× and 1.9× lower, respectively.

5.2 Simulations Under Poisson Workloads

Topologies. To show the performance variation of PayDebt among different over-subscription ratios, several topologies are used in NS3 simulations. The first topology is the same as that used in Homa. It is a 2-level network that contains 4 core switches, 9 ToRs, and 144 hosts (i.e., 16 hosts per ToR). As a non-blocking topology, each ToR connects its hosts and cores with 10/40 Gbps links, respectively. We also build oversubscribed topologies by changing the ToR-to-Core links to 20 Gbps/10 Gbps/5 Gbps, i.e., with a 2:1/4:1/8:1 over-subscription ratio, respectively. The switch buffer is 8 MB. We use the 4:1 oversubscribed topology by default unless otherwise specified. The base RTT and BDP are 10 μs and 12.5KB, respectively.

PayDebt Improves Small Flows' Performance. We evaluate PayDebt with all three workloads when there is no buffer overflow. Fig. 9 shows the performance of maximum buffer occupancy (x-axis) and FCT (y-axis) across different workloads. The closer the bar is to the x-axis, the smaller FCT the protocol achieves. Similarly, the closer the bar to the y-axis, the less buffer the protocol occupies.

These protocols present distinctly different characteristics. Again, we use Homa(8q) as an upper bound. Small flows' performance of Homa(8q) is outstanding, especially under workloads mixed with small and large flows. This is because small flows can be prioritized over large flows both on end-hosts (SRPT scheduling) and in-network (8 priority queues), significantly mitigating the queuing delay of small flows. In addition, Homa's large flows also benefit from the run-to-completion behavior brought by SRPT. PayDebt benefits small flows across three workloads by leveraging the debt mechanism to reduce the buffer occupancy. As expected, debt plays a more effective part in reducing the buffer occupancy in workloads where small flows mix with large flows, i.e., Hadoop and Web search.

(i) Memcached workload. For this workload, scheduled packets are rare; therefore, debt plays a smaller part in reducing buffer occupancy. Although ExpressPass and EP (A) achieve the smallest buffer occupancy across all workloads, their FCTs are relatively high. The main reason is ExpressPass wastes the first RTT. Although EP(A) attempts

to utilize the first RTT, selectively dropping unscheduled packets makes it suffer a large amount of retransmission. Besides, ExpressPass and EP(A) credits should compete for bandwidth at the network bottleneck. It could result in redundant drops of credits which in turn prolongs flows' FCT, especially small flows'. The results are consistent with Section 2.3, while the downgrade of average/tail FCT is less obvious than the distribution of FCT. Similarly, Homa(A) downgrades FCTs of Homa because selective drop hurts small flows. PayDebt achieves almost the same buffer occupancy with Homa (2q) and DCQCN. Buffer occupancy of HPCC is 15.6% higher than DCQCN as In-Network Telemetry (INT) header costs much for workloads composed of a large fraction of small flows.

Compared with state-of-the-art approaches, PayDebt reduces the average FCT of small flows by 3.6% to 85.4%. And the 99th-tail latency is up to 7.4× lower. Buffer built-up influences the effects of Homa's mechanism (Section 5.1). Besides, control packets, i.e., acknowledgments, of Homa share the same queue with data, while PayDebt control packets share the same queue with debt and token. This makes ACKs suffer a smaller queuing time. Therefore, PayDebt achieves a relatively smaller average FCT than Homa (2q). For clarity, Fig. 9d decomposes the flows' FCT of Memcached.

(ii) Hadoop workload. This workload consists of large flows mixed with small flows, PayDebt benefits from debt to achieve a modest buffer occupancy; therefore, it speeds up small flows. Compared with state of the art, the buffer occupancy of PayDebt is reduced by up to 4.8×. Therefore, small flows' average and tail latency is reduced by 20.7% and 35.2%, respectively. The main reason is that Homa transmits unscheduled and scheduled packets without coordination, which results in large buffer occupancy (Section 2.4). Homa(A) selectively drops unscheduled packets, downgrading small flows' performance. Homa(A) transmits tokens triggering retransmission of lost packets. However, it does not use in-network rate-limiters for tokens. Tokens received by senders could exceed network bottleneck bandwidth, resulting in scheduled packets piling up. Therefore, the buffer occupancy of Homa(A) is also relatively large.

The Hadoop workload consists of around 64% of messages whose size is smaller than one MTU. PayDebt handles these tiny flows by leveraging *fractional* debt, which may not perfectly pay back the pre-consumed bandwidth of unscheduled packets. This results in a larger buffer occupancy than HPCC. However, compared with HPCC, PayDebt reduces small flows' average and tail latency by 30.7% and 40.5%, respectively. HPCC leverages INT measurements to adjust the sending window but could mismatch the network state as the buffer occupancy are instantaneously and result in bandwidth waste [53]. Benefit from a smaller buffer occupancy, PayDebt speeds up small flows' average and tail latency, compared to DCQCN, by 56.6% and 51.6%, respectively. The performance of ExpressPass is not good, suffering from the waste of the first-RTT. Compared with ExpressPass, PayDebt reduces small flows' average and tail latency by 72.7% and 83.6%, respectively. Improvement has been made by EP(A) compared with ExpressPass, but the retransmission of unscheduled packets still hurts small flows' performance.

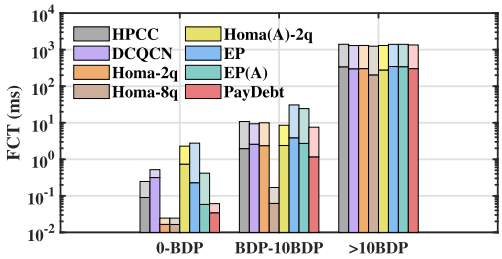


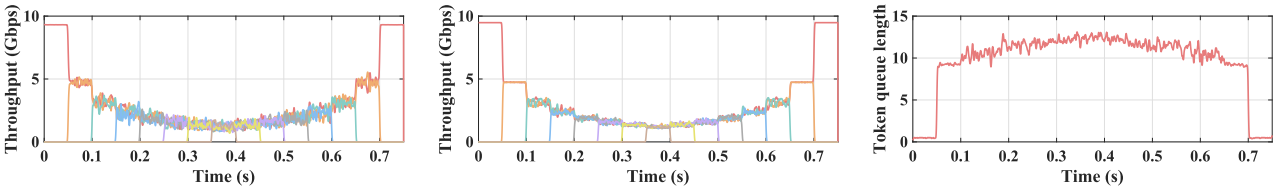
Fig. 10. FCTs of flows with different size (Web Search).

(iii) Web search workload. The trend is a little different. This workload is made up of packets with full-length MTU. Almost no *fractional* debt exists. Therefore, PayDebt achieves a smaller buffer occupancy than HPCC. The buffer occupancy of Homa reaches 5.7 MB, where the value is 163.7 KB for PayDebt. For clarity, Fig. 10 depicts the FCT of flows smaller than 100 KB. PayDebt achieves a relatively small variance among different flows, indicating that the performance of PayDebt is stable. The FCT variance of ExpressPass indicates that with the rate-based token generation mechanism, tokens could be dropped repeatedly. In addition, Fig. 9e decomposes the flows' FCT of Web search. Benefiting from a relatively small buffer occupancy, PayDebt reduces the FCTs of small flows whose size is smaller than 10 BDP compared with state-of-the-art approaches except for Homa. At the same time, the tail latency of PayDebt is not compromised.

PayDebt Does not Hurt the Throughput of Large Flows. Now we focus on the average FCT and tail latency of Hadoop and Web search workload. The average FCT is reduced by up to 38.1% compared with state of the art.

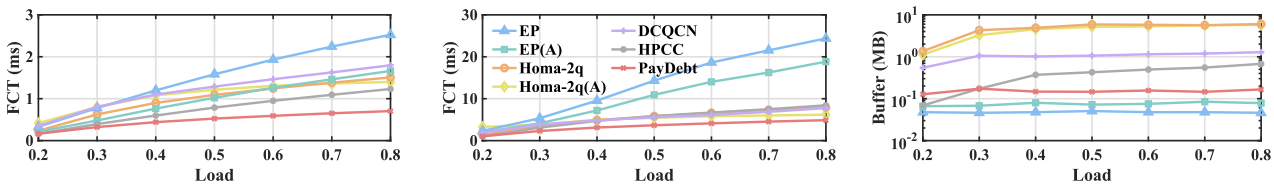
The debt mechanism does not hurt the throughput of large flows, i.e., the tail latency is not prolonged. The tail FCT is reduced by 10.3% - 17.3% compared with ExpressPass/EP(A) and HPCC. They both face the last-RTT waste of tokens (Section 4.1.1). Instead, PayDebt uses a data-driven token generation method, which utilizes bandwidth. HPCC uses accurate in-network information, i.e., queue length and link bandwidth capacity, to adjust sending window. On the one hand, small intermittent flows result in the transiency of queue length. Therefore the adjustment of sending window may mismatch the current network state. On the other hand, HPCC explicitly controls the bottleneck links to have a 5% bandwidth headroom; at the same time, it uses INT headers, which also wastes bandwidth. For Web search workload, the tail FCT of PayDebt is slightly downgraded by 2.4% and 2.9% compared with DCQCN and Homa (drop-less scenarios). Homa and DCQCN benefit from a relatively large buffer occupancy to utilize the bandwidth. Besides, PayDebt rate limits tokens, therefore reversing data path uses up to 95% bandwidth. This can slightly downgrade the performance of tail latency.

PayDebt leverages debt to coordinate between unscheduled and scheduled packets. The buffer occupancy is significantly reduced to benefit small flows without significant throughput loss of large flows. Although ExpressPass and EP(A) achieve the smallest buffer occupancy across all workloads, FCTs of both small flows and tail latency are relatively high. The buffer occupancy of DCQCN and Homa is relatively high. The tail latency is good, but the small flows' FCT is relatively large.



(a) ExpressPass fairness (b) PayDebt fairness (c) Token queue length of PayDebt

Fig. 11. Fairness behavior of PayDebt and ExpressPass.



(a) 0-100KB average FCT (b) 0-100KB 99th tail FCT (c) Buffer occupancy

Fig. 12. Performance of Web Search workloads across different load.

5.3 Deep Dive of PayDebt's Performance

PayDebt Achieves Good Fairness. To test the fairness and convergence speed of PayDebt, we conduct simulations where eight flows pass through the same network bottleneck one by one and leave afterward. Note that for PayDebt, a proactive congestion control whose receiver hosts use a Round-Robin strategy to schedule tokens, flows belonging to the same receiver host can achieve perfect fairness. Therefore, to test PayDebt's token congestion control mechanism (Section 4.1.3), we choose a scenario where flows are destined to different receiver hosts. Figs. 11a and 11b show the fairness of ExpressPass and PayDebt, respectively. Both PayDebt and ExpressPass converge quickly. They both benefit from the characteristics of proactive protocols, allocating bandwidth by receivers. PayDebt provides better fairness than ExpressPass. ExpressPass suffers from redundant credit drops at the network bottleneck, which could be unfair. PayDebt does not drop tokens, and debts do not interfere with the fairness since every flow should generate debt packets.

Fig. 11c demonstrates the token length accordingly. PayDebt sets the ECN-marking threshold K_t of token to $1*$BDP. When the third flow is injected, the queue length of the token exceeds the threshold, i.e., PayDebt's token congestion control starts to take effect. The token queue length converges to around the threshold K_t. It also indicates that token packets do not consume much shared buffer on switches.

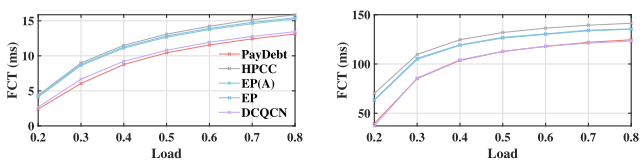
Performance Across Different Loads. We evaluate PayDebt under a wide range of network loads from 20% to 80%. Fig. 12 shows that PayDebt performs well under heavy loads and is at least comparable to state of the art under light loads. Under light loads, e.g., 20%, the FCT of small flows are nearly the same except for ExpressPass/DCQCN/Homa(A). The main reason is that lightweight does not put much pressure on the network and the buffer occupancy. ExpressPass wastes the first-RTT. DCQCN faces a relatively larger buffer because of a relatively large ECN-marking threshold.

Along with load increasing, PayDebt stands out gradually. DCQCN/HPCC/Homa faces a larger buffer occupancy, and EP(A)/Homa(A) faces a larger packet loss rate. The buffer occupancy of PayDebt is almost stable across different loads, i.e., around 160KB to 200KB, indicating that PayDebt achieves good coordination among unscheduled and scheduled packets.

Performance Under Higher Bandwidth Links. Fig. 13 demonstrates the performance under 100 Gbps links across a wide range of network loads from 20% to 80%. The trends of performance among different loads are relatively the same. PayDebt reduces the average FCTs as well as the tail latency, indicating that PayDebt is robust to high bandwidth links[2].

Performance of Topologies With Other Fan-In Ratios. Besides, to investigate PayDebt's performance of Poisson arrival flows across different topologies, a non-blocking topology and topologies with 2:1 and 8:1 over-subscription ratios are used for verification, respectively.

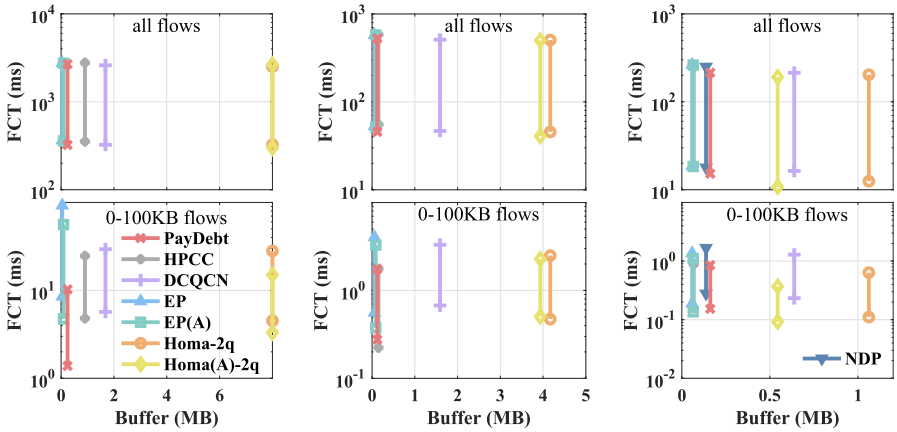
Figs. 14a and 14b show the performance variation of PayDebt under Web search workloads following a Poisson process among topologies with different over-subscription ratio. Recall that Fig. 9c shows the performance of PayDebt under over-subscription ratio 4:1 (Section 5.2). For the 2:1 over-subscription ratio topology, PayDebt achieves good performance on small flows and competitive performance on tail latency as expected. For the 8:1 ratio, a more notable improvement on small flows is achieved. The main reason is that other protocols struggle under a high fan-in ratio, and a large buffer occupancy is observed. It means PayDebt is robust to different fan-in ratios to some extent.



(a) Average FCT (b) 99th tail FCT

Fig. 13. Performance of Web Search workloads under 100 Gbps links.

2. Homa suffers from a large number of packet loss and timeout retransmission under 100 Gbps links. Hence, we omit its results.



(a) Web Search (8:1) (b) Web Search (2:1) (c) Web Search (non-blocking)

Fig. 14. Performance of different topologies.

Fig. 14c shows the performance of PayDebt under Web search workloads under non-blocking topology. Homa is designed for small RPCs in non-blocking topology, and achieves better performance on small flows. Aside from Homa, PayDebt achieves at least comparable performance with other approaches. NDP trims packet payload, therefore achieving a smaller buffer than PayDebt. However, it hurts the performance of small flows because of the aggressive drop (Section 2.2).

5.4 Further Optimization of PayDebt

When Buffer Overflows. In prior simulations, 8MB shared buffer is used, where buffer occupancy of PayDebt/Homa reaches up to 1.03MB/6MB, respectively. We wonder about their performance with a smaller switch buffer, where packet loss occurs. Fig. 15 illustrates the results.

We compare PayDebt with Homa/Homa(A) under both 8MB and 4MB switch buffer. Because PayDebt achieves a relatively small buffer, no packet loss occurs under both scenarios. Under the 4MB shared buffer scenario, Homa faces 0.03% and 0.01% packet loss rate for Hadoop and Web search, respectively. The average FCT of small flows of Homa is downgraded by 10.6% - 19.0%, and tail latency is 1.8× larger compared with Homa (8MB buffer) without packet loss.

Further, the switch buffer is reduced to 800KB to investigate the performance of PayDebt under lossy scenarios. PayDebt starts to drop packets in the Hadoop workload. The timeout of the probe packet is set to a relatively large value, i.e., 4ms, to avoid unnecessary transmission of probing. PayDebt faces 0.01% packet loss rate. Packet loss can hurt the performance of small flows. The average FCT of PayDebt's small flows is downgraded by 3.5% compared with PayDebt (8MB/4MB switch buffer). No explicit downgrade has been observed for small flows' tail latency. Benefit from selective retransmission mechanism, considering all flows, performance downgrade is negligible, i.e., average and tail FCT are increased by 0.9% and 1.4%, respectively. The performance of PayDebt is better than in Homa, even under lossy scenarios. To summarize, the performance of PayDebt does not downgrade severely when packet loss occurs.

Performance Under Incast Scenarios. We evaluate PayDebt's design of bounding the maximum inflight unscheduled packets of the same sender-receiver host pair. Fig. 16a shows the buffer occupancy performance under incast scenarios. Incast flows are composed of one BDP packets. ExpressPass, EP(A), and Homa(A) achieve a small buffer occupancy. This is expected because ExpressPass has no unscheduled packets, while Homa(A) and EP(A) benefit from setting a small threshold to drop unscheduled packets. Along with the number of incast flows increasing, the buffer occupancy of PayDebt remains relatively small. PayDebt benefits from bounding the maximum inflight packets of the same host pair. The buffer occupancy of DCQCN and HPCC reach above 6MB, and PFCs are triggered. The buffer occupancy of Homa reaches 8MB, and Homa starts to drop.

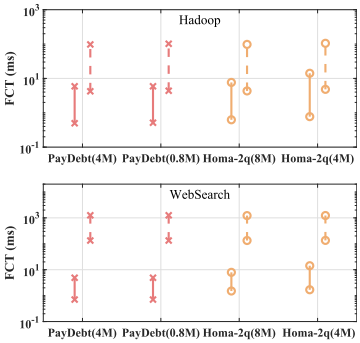
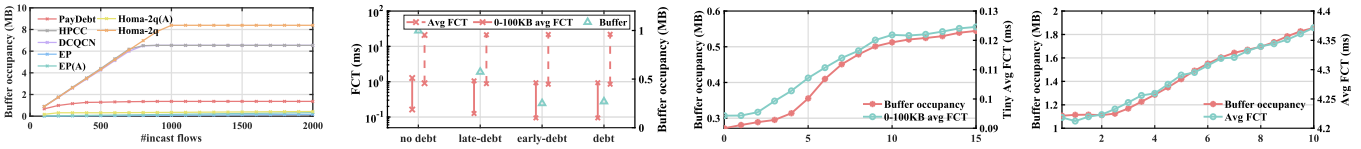


Fig. 15. When packet drop occurs. The solid/dashed line denotes avg/99th-tail latency of flows smaller than 100KB/all flows, respectively.

5.5 Mechanism Validation

This section digs into the debt mechanism of PayDebt to validate its effects. Fig. 16b depicts the results.

The Timing of Debt is Essential. Fig. 16b shows the performance of PayDebt by comparing it with three other variants of PayDebt, i.e., no debt, w/ late-debt, and w/ early-debt when running workload Hadoop following a Poisson process. No debt means that no debt mechanism is used. Late-debt stands for debts are sent after data packets of flows are all received. Contrarily, early-debt stands for a debt is sent as soon as an unscheduled packet is received (i.e., naïve debt in Section 3.1). PayDebt outperforms w/o debt and w/ late-debt. Compared with PayDebt w/o debt and w/ late-debt, the performance of flows smaller than 100KB is



(a) Buffer occupancy under incast. (b) Debt improves small flows. (c) ECN marking K_d. (d) Token congestion control K_t.

Fig. 16. Mechanism validation and parameter selection. In (b), late-debt stands for debts are sent after data packets of flows are all received. Contrarily, early-debt stands for a debt is sent as soon as an unscheduled packet is received. In (c) and (d), the left y-label indicates the value of buffer and the right y-label indicates FCTs.

greatly improved. This is expected because the buffer occupancy is reduced by 3.72× and 2.14×, respectively. It has been observed that improvement is made by PayDebt w/ late-debt, compared with PayDebt w/o debt. The main reason is that the late-debts also compete for bandwidth with tokens and reduce buffer occupancy. However, the timing of late-debt deviates; therefore, the performance is not as well as PayDebt. PayDebt w/ early-debt achieves almost the same performance as PayDebt. Under a heavy load scenario, sending a debt as soon as an unscheduled packet is received does not hurt network throughput.

ECN-Based Token Congestion Control. PayDebt deals with the token incast problem with ECN-based token congestion control. Fig. 16d shows the buffer usage and average FCT with different ECN marking thresholds in the Hadoop workload. A small ECN marking threshold contributes to a small buffer occupancy and a small average FCT. It indicates that token congestion control is vital to provide better performance.

5.6 Parameter Selection

ECN Marking Threshold K_d. K_d determines how conservative it is for flows to transmit scheduled packets. The smaller K_d is, the more conservative PayDebt is to transmit scheduled packets. Fig. 16c shows the buffer usage and tail FCT with different K_d in the Hadoop workload. Threshold 0 means sending a debt upon receiving an unscheduled packet, i.e., early-debt. It achieves the lowest buffer occupancy but not the smallest average FCT for small flows. As K_d increases, the buffer increases. It shows that $1*$ BDP can achieve good performance among different thresholds.

Token Congestion Control K_t. ECN marking threshold of tokens K_t determines how aggressively PayDebt deals with the token incast problem. Fig. 16d shows that a too-small ECN threshold, i.e., $0.5*$BDP, causes under-utilization as in-flight tokens might be insufficient. A too-large threshold is not sensitive to token queuing and leads to the token incast problem (Section 4.1.3). We find $1*$BDP achieves great performance.

6 Discussion

Overhead of PayDebt. The bandwidth overhead of debts/tokens is 5%, as other proactive protocols do. Actually, for reactive protocols (e.g., HPCC), per-packet ACK is necessary for precise congestion signals, which also consume at least 5% bandwidth. PayDebt uses a dedicated queue for rate-limiting. It is necessary for oversubscribed networks to ensure that bandwidth consumed by scheduled packets will not exceed the network bottleneck. One additional queue is generally available.

Compatible With Multiple Applications. Different applications are isolated by different queues (Section 2.3). Their bandwidth can be allocated statically or dynamically. For static allocation, PayDebt's rate-limiters can be simply configured to a static proportion (i.e., 5%) of the allocated bandwidth. For dynamic allocation, given that the bandwidth used on bi-direction paths is not symmetric, data packets should go through the same direction as the corresponding tokens to ensure that the rate-limiting is correct. It can be achieved by leveraging forwarding tokens, i.e., tokens are sent by senders. When the sender receives the ACKs of tokens, corresponding scheduled packets can be transmitted.

When no Scheduled Packet Exists. Debts do not take effect when all flows are made up of unscheduled packets. This is the limitation for almost all the end-to-end congestion control protocols, i.e., flows finish transmitting too fast to take congestion control. To handle all-unscheduled-packets scenarios, per-hop flow control should be involved [69], [70], which is complementary to end-to-end congestion control protocols.

7 Conclusion

The central idea of PayDebt is that buffer occupancy could be reduced significantly in the first place instead of handling buffer explosion afterward. At the core of PayDebt, it proposes a debt mechanism to coordinate unscheduled and scheduled packets so that they can share bandwidth harmoniously. Consequently, packet queuing delay and drops can be reduced significantly. The average FCTs under mainstream oversubscribed networks can be greatly reduced. PayDebt's performance is at least comparable to state of the art under non-blocking networks. This design also alleviates the switch vendors' pressure to further increase the capacity of on-chip buffers.

Acknowledgments

The authors would like to thank anonymous reviewers for their valuable comments.

References

[1] F. Ahmad, S. Chakradhar, A. Raghunathan, and T. N. Vijaykumar, "ShuffleWatcher: Shuffle-aware scheduling in multi-tenant MapReduce clusters," in *Proc. USENIX Annu. Tech. Conf.*, 2014, pp. 1–12.

[2] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, "Large-scale cluster management at Google with Borg," in *Proc. 10th Eur. Conf. Comput. Syst.*, 2015, Art. no. 18.

[3] M. Chowdhury, S. Kandula, and I. Stoica, "Leveraging endpoint flexibility in data-intensive clusters," in *Proc. ACM SIGCOMM Conf.*, 2013, pp. 231–242.

[4] R. Grandl, G. Ananthanarayanan, S. Kandula, S. Rao, and A. Akella, "Multi-resource packing for cluster schedulers," in *Proc. ACM Conf. SIGCOMM*, 2014, pp. 455–466.

[5] D. Gibson et al., "Aquila: A unified, low-latency fabric for datacenter networks," in *Proc. 19th USENIX Symp. Netw. Syst. Des. Implementation*, 2022, pp. 1249–1266.

[6] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, "Inside the social network's (datacenter) network," in *Proc. ACM Conf. Special Int. Group Data Commun.*, 2015, pp. 123–137.

[7] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proc. ACM SIGCOMM Conf. Internet Meas.*, 2010, pp. 267–280.

[8] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The nature of data center traffic: Measurements & analysis," in *Proc. ACM SIGCOMM Conf. Internet Meas.*, 2009, pp. 202–208.

[9] B. Atikoglu, Y. Xu, E. Frachtenberg, S. Jiang, and M. Paleczny, "Workload analysis of a large-scale key-value store," in *Proc. ACM SIGMETRICS/PERFORMANCE Joint Int. Conf. Meas. Model. Comput. Syst.*, 2012, pp. 53–64.

[10] B. Fitzpatrick, "Memcached: A distributed memory object caching system," 2011. [Online]. Available: http://www.memcached.org/

[11] A. Kalia, M. Kaminsky, and D. Andersen, "Datacenter RPCs can be general and fast," in *Proc. 16th USENIX Conf. Netw. Syst. Des. Implementation*, 2019, pp. 1–16.

[12] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[13] M. Zaharia et al., "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *Proc. USENIX Conf. Netw. Syst. Des. Implementation*, 2012, Art. no. 2.

[14] M. Zaharia, M. Chowdhury, M. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets," in *Proc. USENIX Conf. Hot Topics Cloud Comput.*, 2010, Art. no. 10.

[15] G. Ananthanarayanan et al., "Reining in the outliers in map-reduce clusters using Mantri," in *Proc. 9th USENIX Conf. Operating Syst. Des. Implementation*, 2010, pp. 265–278.

[16] M. Chowdhury, M. Zaharia, J. Ma, M. I. Jordan, and I. Stoica, "Managing data transfers in computer clusters with orchestra," in *Proc. ACM SIGCOMM Conf.*, 2011, pp. 98–109.

[17] M. Zaharia et al., "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *Proc. USENIX Conf. Netw. Syst. Des. Implementation*, 2012, Art. no. 2.

[18] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. Long, and C. Maltzahn, "Ceph: A scalable, high-performance distributed file system," in *Proc. 7th Symp. Oper. Syst. Des. Implementation*, 2006, pp. 307–320.

[19] M. Li et al., "Scaling distributed machine learning with the parameter server," in *Proc. 11th USENIX Conf. Oper. Syst. Des. Implementation*, 2014, pp. 583–598.

[20] M. Abadi et al., "TensorFlow: A system for large-scale machine learning," in *Proc. 12th USENIX Conf. Oper. Syst. Des. Implementation*, 2016, pp. 265–283.

[21] Y. Peng et al., "A generic communication scheduler for distributed DNN training acceleration," in *Proc. 27th ACM Symp. Operating Syst. Princ.*, 2019, pp. 16–29.

[22] J. Dean and L. A. Barroso, "The tail at scale," *Commun. ACM*, vol. 56, no. 2, pp. 74–80, 2013.

[23] D. Zats, T. Das, P. Mohan, D. Borthakur, and R. Katz, "DeTail: Reducing the flow completion time tail in datacenter networks," in *Proc. ACM SIGCOMM Conf. Appl. Technol. Archit. Protoc. Comput. Commun.*, 2012, pp. 139–150.

[24] M. Alizadeh et al., "Data center TCP (DCTCP)," in *Proc. ACM SIGCOMM Conf.*, 2011, pp. 63–74.

[25] Y. Zhu et al., "Congestion control for large-scale RDMA deployments," in *Proc. ACM Conf. Special Int. Group Data Commun.*, 2015, pp. 523–536.

[26] Y. Li et al., "HPCC: High precision congestion control," in *Proc. ACM Conf. Special Int. Group Data Commun.*, 2019, pp. 44–58.

[27] R. B. Basat, S. Ramanathan, Y. Li, G. Antichi, M. Yu, and M. Mitzenmacher, "PINT: Probabilistic in-band network telemetry," in *Proc. Annu. Conf. ACM Special Int. Group Data Commun. Appl. Technol. Archit. Protoc. Comput. Commun.*, 2020, pp. 662–680.

[28] V. Addanki, O. Michel, and S. Schmid, "PowerTCP: Pushing the performance limits of datacenter networks," in *Proc. 19th USENIX Symp. Netw. Syst. Des. Implementation*, 2022, pp. 51–70.

[29] R. Mittal et al., "TIMELY: RTT-based congestion control for the datacenter," in *Proc. ACM Conf. Special Int. Group Data Commun.*, 2015, pp. 537–550.

[30] G. Kumar et al., "Swift: Delay is simple and effective for congestion control in the datacenter," in *Proc. ACM Conf. Special Int. Group Data Commun.*, 2020, pp. 514–528.

[31] S. Liu, A. Ghalayini, M. Alizadeh, B. Prabhakar, M. Rosenblum, and A. Sivaraman, "Breaking the transience-equilibrium nexus: A new approach to datacenter packet transport," in *Proc. USENIX Symp. Netw. Syst. Des. Implementation*, 2021, pp. 47–63.

[32] I. Cho, K. Jang, and D. Han, "Credit-scheduled delay-bounded congestion control for datacenters," in *Proc. ACM Conf. Special Int. Group Data Commun.*, 2017, pp. 239–252.

[33] B. Montazeri, Y. Li, M. Alizadeh, and J. Ousterhout, "Homa: A receiver-driven low-latency transport protocol using network priorities," in *Proc. Conf. ACM Special Int. Group Data Commun.*, 2018, pp. 221–235.

[34] M. Handley et al., "Re-architecting datacenter networks and stacks for low latency and high performance," in *Proc. Conf. ACM Special Int. Group Data Commun.*, 2017, pp. 29–42.

[35] P. X. Gao, A. Narayan, G. Kumar, R. Agarwal, S. Ratnasamy, and S. Shenker, "pHost: Distributed near-optimal datacenter transport over commodity network fabric," in *Proc. 11th ACM Conf. Emerg. Netw. Experiments Technol.*, 2015, Art. no. 1.

[36] S. Hu et al., "Aeolus: A building block for proactive transport in datacenters," in *Proc. Annu. Conf. ACM Special Int. Group Data Commun. Appl. Technol. Archit. Protoc. Comput. Commun.*, 2020, pp. 422–434.

[37] A. Andreyev, "Introducing data center fabric, the next-generation Facebook data center network," 2014. [Online]. Available: https://engineering.fb.com/2014/11/14/production-engineering/introducing-data-center-fabric-the-next-generation-facebook-data-center-network/

[38] A. Singh et al., "Jupiter rising: A decade of Clos topologies and centralized control in Google's datacenter network," in *Proc. ACM Conf. Special Int. Group Data Commun.*, 2015, pp. 183–197.

[39] Cisco, "Oversubscription and density best practices," 2015. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/data-center-virtualization/storage-networking-solution/net_implementation_white_paper0900aecd800f592f.html

[40] C. DeCusatis, "Transforming the data center network," in *Handbook of Fiber Optic Data Communication*, 4th ed. New York, NY, USA: Academic, 2013, pp. 3–22.

[41] L. Paraschis and K. Raj, "Chapter 15 - Innovations in DCI transport networks," in *Optical Fiber Telecommunications VII*, A. E. Willner, Ed., New York, NY, USA: Academic, 2020, pp. 673–718.

[42] V. Vasudevan et al., "Safe and effective fine-grained TCP retransmissions for datacenter communication," in *Proc. ACM SIGCOMM Conf. Data Commun.*, 2009, pp. 303–314.

[43] A. Greenberg et al., "Vl2: A scalable and flexible data center network," in *Proc. ACM SIGCOMM Conf. Data Commun.*, 2009, pp. 51–62.

[44] L. A. Barroso and U. Hölzle, "The datacenter as a computer: An introduction to the design of warehouse-scale machines," *Synth. Lectures Comput. Archit.*, vol. 4, no. 1, pp. 1–108, 2009.

[45] W. Cheng, K. Qian, W. Jiang, T. Zhang, and F. Ren, "Re-architecting congestion management in lossless ethernet," in *Proc. USENIX Conf. Netw. Syst. Des. Implementation*, 2020, pp. 19–36.

[46] T. Benson, A. Anand, A. Akella, and M. Zhang, "Understanding data center traffic characteristics," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 1, pp. 92–99, 2010.

[47] R. Kapoor, A. C. Snoeren, G. M. Voelker, and G. Porter, "Bullet trains: A study of NIC burst behavior at microsecond timescales," in *Proc. 9th ACM Conf. Emerg. Netw. Experiments Technol.*, 2013, pp. 133–138.

[48] M. Noormohammadpour and C. S. Raghavendra, "Datacenter traffic control: Understanding techniques and tradeoffs," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 2, pp. 1492–1525, Apr.–Jun. 2018.

[49] M. Al-Fares et al., "Hedera: Dynamic flow scheduling for data center networks," in *Proc. USENIX Conf. Netw. Syst. Des. Implementation*, 2010, pp. 89–92.

[50] X. Zhou et al., "Mirror mirror on the ceiling: Flexible wireless links for data centers," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 42, no. 4, pp. 443–454, 2012.

[51] H. Wu, J. Ju, G. Lu, C. Guo, Y. Xiong, and Y. Zhang, "Tuning ECN for data center networks," in *Proc. 8th Int. Conf. Emerg. Netw. Experiments Technol.*, 2012, pp. 25–36.

[52] C. Lee, C. Park, K. Jang, S. B. Moon, and D. Han, "Accurate latency-based congestion feedback for datacenters," in *Proc. USENIX Conf. Annu. Tech. Conf.*, 2015, pp. 403–415.

[53] PayDebt appendix, 2021. [Online]. Available: https://anonymous.4open.science/r/Paydebt-appendix-8BEC

[54] C. Guo et al., "RDMA over commodity ethernet at scale," in *Proc. ACM SIGCOMM Conf.*, 2016, pp. 202–215.

[55] K. Qian, W. Cheng, T. Zhang, and F. Ren, "Gentle flow control: Avoiding deadlock in lossless networks," in *Proc. ACM Conf. Special Int. Group Data Commun.*, 2019, pp. 75–89.

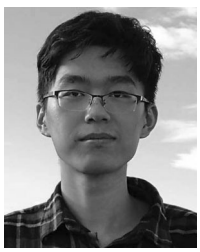
[56] Y. Le, B. Stephens, A. Singhvi, A. Akella, and M. M. Swift, "RoGUE: RDMA over generic unconverged ethernet," in *Proc. ACM Symp. Cloud Comput.*, 2018, pp. 225–236.
[57] J. Ousterhout, "A Linux kernel implementation of the Homa transport protocol," in *Proc. USENIX Annu. Tech. Conf.*, 2021, pp. 99–115.
[58] Y. Gao et al., "When cloud storage meets RDMA," in *Proc. USENIX Conf. Netw. Syst. Des. Implementation*, 2021, pp. 519–533.
[59] Cisco., "Cisco Nexus 7000 series NX-OS security configuration guide, release 6.x," 2019. [Online]. Available: https://www.cisco.com/c/en/us/td/docs/switches/datacenter/sw/6_x/nx-os/security/configuration/guide/b_Cisco_Nexus_7000_NX-OS_Security_Configuration_Guide__Release_6-x.html
[60] Juniper, "Configuring rate limiting and sharing of excess bandwidth on multiservices PICs," 2021. [Online]. Available: https://www.juniper.net/documentation/us/en/software/junos/cos/topics/task/cos-configuring-rate-limiting-and-sharing-of-excess-bandwidth-on-multiservices-pics.html
[61] Broadcom, "Broadcom BCM53154: Low-power five-port GbE time-sensitive networking switch," 2018. [Online]. Available: https://docs.broadcom.com/doc/53154-PB100
[62] Stanford, "Homa simulator," 2018. [Online]. Available: https://github.com/PlatformLab/HomaSimulation
[63] KAIST, "Expresspass simulator," 2017. [Online]. Available: https://github.com/kaist-ina/ns2-xpass
[64] Alibaba, "Hpcc simulator," 2019. [Online]. Available: https://github.com/alibaba-edu/High-Precision-Congestion-Control
[65] Berkeley, "Berkeley extensible software switch," 2018. [Online]. Available: https://github.com/NetSys/bess
[66] Intel, "Data plane development kit," 2011. [Online]. Available: https://www.dpdk.org/
[67] Stanford, "An implementation of the Homa transport protocol as a C++ userspace library," 2019. [Online]. Available: https://github.com/PlatformLab/Homa
[68] Stanford, "A Linux kernel module that implements the Homa transport protocol," 2019. [Online]. Available: https://github.com/PlatformLab/HomaModule
[69] K. Liu et al., "Floodgate: Taming incast in datacenter networks," in *Proc. 17th Int. Conf. Emerg. Netw. Experiments Technol.*, 2021, pp. 30–44.
[70] P. Goyal, P. Shah, N. K. Sharma, M. Alizadeh, and T. E. Anderson, "Backpressure flow control," in *Proc. 19th USENIX Symp. Netw. Syst. Des. Implementation*, 2022, pp. 779–805.

Kexin Liu received the BS degree from the Department of Software Engineering, Sun Yat-sen University, China, in 2017. She is currently working toward the PhD degree in the Department of Computer Science and Technology, Nanjing University, China. Her research interests include congestion control protocols, flow control protocols, and datacenter networks.

Chen Tian received the BS, MS, and PhD degrees from the Department of Electronics and Information Engineering, Huazhong University of Science and Technology, China, in 2000, 2003, and 2008, respectively. He is a professor with the State Key Laboratory for Novel Software Technology, Nanjing University, China. He was previously an Associate Professor with the School of Electronics Information and Communications, Huazhong University of Science and Technology, China. From 2012 to 2013, he was a postdoctoral researcher with the Department of Computer Science, Yale University. His research interests include data center networks, distributed systems, and internet protocols.

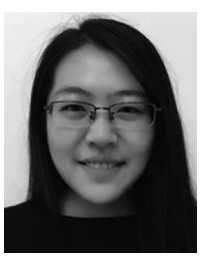
Qingyue Wang received the BS degree from the Department of Computer Science and Technology, Wuhan University, China, in 2019. She is currently working toward the MS degree in the Department of Computer Science and Technology, Nanjing University, China. Her main research interest is datacenter networks.

Yanqing Chen received the BS degree from the Department of Computer Science and Engineering, Southeast University, China, in 2019. He is currently working toward the PhD degree in the Department of Computer Science and Technology, Nanjing University, China. His research interests include programmable switches and datacenter networks.



Bingchuan Tian received the BS degree from the Department of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, China, in 2016, and the PhD degree from the Department of Computer Science and Technology, Nanjing University, China, in 2021. His research interests include network verification and intent-based networking.

Wenhao Sun received the BS degree in information engineering from Southeast University, China, in 2010, and the PhD degree in circuits and systems from Southeast University, China, in 2017. He has been with the Network Technology Lab, 2012 Labs, Huawei since graduation. His current research interests include: Datacenter networks, industrial networks, network reliability, and network science.



Ke Meng received the BE degree in automation control from the University of Science and Technology of China, Hefei, China, in 2013, and the PhD degree in robotics and biomedical engineering from both the University of Science and Technology of China and the City University of Hong Kong, Hong Kong, in 2018. She has been with the Network Technology Lab, 2012 Labs, Huawei since graduation. Her current research interests include: Datacenter networking, congestion control, flow control, and network optimization.

Long Yan received the BS and PhD degrees in computer science and technology from the University of Science and Technology of China, Hefei, China, in 2014 and 2019, respectively. Since 2019, he has been with the Network Technology Lab, 2012 Labs, Huawei, where he is currently a senior engineer. His current research interests include datacenter networks, congestion control protocols, and flow control protocols.

Lei Han is the principal investigator in the datacenter network field with the Network Technology Lab, 2012 Labs, Huawei. He has been working on datacom network technology research since 2005, and started concentrating on datacenter networks in 2009. Many of his research results have been adopted by Huawei DCN products, and he has more than 30 global patents.

Jie Fu is currently the director of the Network Technology Lab, 2012 Labs, Huawei. She leads the research and innovation in network protocols and architecture, fundamental theories, DCN networks, campus networks, etc. She has rich experience in datacom network technology research and product development.

Wanchun Dou received the PhD degree in mechanical and electronic engineering from the Nanjing University of Science and Technology, China, in 2001. He is currently a full professor of the State Key Laboratory for Novel Software Technology, Nanjing University. From April 2005 to June 2005 and from November 2008 to February 2009, he respectively visited the Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Hong Kong, as a visiting scholar. Up to now, he has chaired three National Natural Science Foundation of China projects and published more than 60 research papers in international journals and international conferences. His research interests include workflow, cloud computing, and service computing.

Guihai Chen received the BS degree in computer software from Nanjing University, in 1984, the ME degree in computer applications from Southeast University, in 1987, and the PhD degree in computer science from the University of Hong Kong, in 1997. He is a distinguished professor of Nanjing University. He had been invited as a visiting professor by Kyushu Institute of Technology in Japan, University of Queensland in Australia and Wayne State University in USA. He has a wide range of research interests with focus on parallel computing, wireless networks, data centers, peer-to-peer computing, high-performance computer architecture, and data engineering. He has published more than 350 peer-reviewed papers, and more than 200 of them are in well-archived international journals such as the *IEEE Transactions on Parallel and Distributed Systems*, *IEEE Transactions on Computers*, *IEEE Transactions on Knowledge and Data Engineering*, *IEEE/ACM Transactions on Networking* and *ACM Transactions on Sensor Networks*, and also in well-known conference proceedings such as HPCA, MOBIHOC, INFOCOM, ICNP, ICDCS, CoNext and AAAI. He has won nine paper awards including ICNP 2015 best paper Award and DASFAA 2017 Best Paper Award.