Credit-Scheduled Delay-Bounded Congestion Control for Datacenters

Inho Cho
KAIST
inho00@kaist.ac.kr

Keon Jang*
Google Inc.
keonjang@google.com

Dongsu Han*
KAIST
dongsuh@ee.kaist.ac.kr

ABSTRACT

Small RTTs (~tens of microseconds), bursty flow arrivals, and a large number of concurrent flows (thousands) in datacenters bring fundamental challenges to congestion control as they either force a flow to send at most one packet per RTT or induce a large queue build-up. The widespread use of shallow buffered switches also makes the problem more challenging with hosts generating many flows in bursts. In addition, as link speeds increase, algorithms that gradually probe for bandwidth take a long time to reach the fair-share. An ideal datacenter congestion control must provide 1) zero data loss, 2) fast convergence, 3) low buffer occupancy, and 4) high utilization. However, these requirements present conflicting goals.

This paper presents a new radical approach, called ExpressPass, an end-to-end credit-scheduled, delay-bounded congestion control for datacenters. ExpressPass uses credit packets to control congestion even before sending data packets, which enables us to achieve bounded delay and fast convergence. It gracefully handles bursty flow arrivals. We implement ExpressPass using commodity switches and provide evaluations using testbed experiments and simulations. ExpressPass converges up to 80 times faster than DCTCP in 10 Gbps links, and the gap increases as link speeds become faster. It greatly improves performance under heavy incast workloads and significantly reduces the flow completion times, especially, for small and medium size flows compared to RCP, DCTCP, HULL, and DX under realistic workloads.

CCS CONCEPTS

• **Networks** → Transport protocols;

KEYWORDS

Congestion Control, Datacenter Network, Credit-based

ACM Reference format:
Inho Cho, Keon Jang, and Dongsu Han. 2017. Credit-Scheduled Delay-Bounded Congestion Control for Datacenters. In *Proceedings of SIGCOMM '17, Los Angeles, CA, USA, August 21-25, 2017,* 14 pages.
https://doi.org/10.1145/3098822.3098840

1 INTRODUCTION

Datacenter networks are rapidly growing in terms of size and link speed [52]. A large datacenter network connects over 100 thousand machines using a Clos network of shallow buffered switches [2, 28]. Each server is connected at 10/40 Gbps today with 100 Gbps on the horizon. This evolution enables low latency and high bandwidth communication within a datacenter. At the same time, it poses a unique set of challenges for congestion control.

In datacenters, short propagation delay makes queuing delay a dominant factor in end-to-end latency [3]. Thus, with higher link speeds, fast convergence has become much more important [34]. However, with buffer per port per Gbps getting smaller, ensuring zero loss and rapid convergence with traditional congestion control has become much more challenging. In addition, Remote Direct Memory Access (RDMA), recently deployed in datacenters [40, 41, 58], poses more stringent latency and performance requirements (e.g., zero data loss).

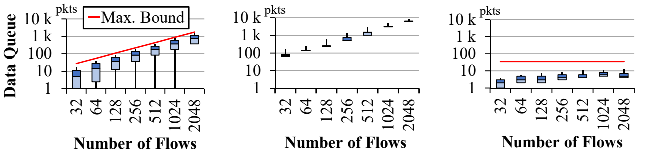
A large body of work addresses these challenges. One popular approach is to react to early congestion signals in a more accurate fashion, using ECN [3, 5, 55, 58] or network delay [38, 41, 47]. These approaches keep queuing lower and handle incast traffic much better than the traditional TCP. However, they are still prone to buffer overflows in bursty and incast traffic patterns. Thus, they rely on priority flow control (PFC) or avoid an aggressive increase to prevent data loss. In fact, small RTTs, bursty flow arrivals, and a large number of concurrent flows in datacenters bring fundamental challenges to this approach because these factors either force a flow to send at most one packet per RTT or induce a large queue build-up. We show in Section 2 that even a hypothetically ideal rate control faces these problems. An alternative is to explicitly determine the bandwidth of a flow or even the packet departure time using a controller or a distributed algorithm [34, 47]. However, this approach incurs signaling latency and is very difficult to scale to large, high-speed (e.g., 100 Gbps) datacenters, and is challenging to make robust against failures and traffic churn [44].

Our approach uses credit packets to control the rate and schedule the arrival of data packets. Receivers send credit packets to senders on a per-flow basis in an end-to-end fashion. Switches then rate-limit the credit packets on each link and determine the available bandwidth for data packets flowing in the reverse direction. By shaping the flow of credit packets in the network, the system proactively controls congestion even *before* sending data packets. A sender naturally learns the amount of traffic that is safe to send, rather than reacting to the congestion signal after sending data. This allows us to quickly ramp up flows without worrying about data loss. In addition, it effectively solves incast because the arrival order of credit packets at the bottleneck link naturally schedules the arrival of data packets in the reverse path at packet granularity.

[*] Co-corresponding authors

SIGCOMM '17, August 21-25, 2017, Los Angeles, CA, USA

ACM ISBN 978-1-4503-4653-5/17/08…$15.00
https://doi.org/10.1145/3098822.3098840



(a) Ideal rate-based (b) DCTCP (c) Credit-based

Figure 1: Data queue length of window/rate-based protocol, DCTCP and credit-based protocol

Realizing the idea of credit-based congestion control, however, is not trivial. One might think that with the credit-based scheme a naïve approach in which a receiver sends credit packets as fast as possible (i.e. the maximum credit rate corresponding to its line rate) can achieve fast convergence, high utilization, and fairness at the same time. In a simple, single bottleneck network, this is true. However, in large networks, the three goals are often at odds, and the naïve approach presents serious problems: (*i*) it wastes bandwidth when there are multiple bottlenecks, (*ii*) it does not guarantee fairness, and (*iii*) the difference in path latency can cause queuing. In addition, in networks with multiple paths, credit and data packets may take asymmetric paths.

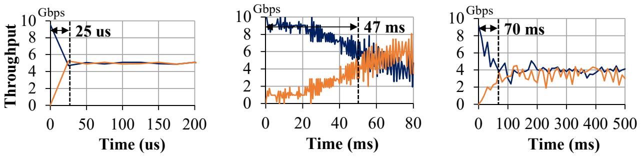
This paper demonstrates credit-based congestion control is a viable alternative for datacenters by addressing the challenges and answers key design questions that arise from a credit-based approach. The resulting design incorporates several components and techniques: (*i*) rate-limiting credits at switches, (*ii*) symmetric hashing to achieve path symmetry, (*iii*) credit feedback control, (*iv*) random jitter, and (*v*) network calculus to determine the maximum queuing.

Our feedback control achieves fast convergence and zero data loss. It effectively mitigates utilization and fairness issues in multi-bottleneck scenarios. We also demonstrate ExpressPass can be implemented using commodity hardware. Our evaluation shows that ExpressPass converges in just a few RTTs when a new flow starts for both 10 Gbps and 100 Gbps, whereas DCTCP takes over hundreds and thousands of RTTs respectively. In all of our evaluations, ExpressPass did not exhibit any single data packet loss. ExpressPass uses up to eight times less switch buffer than DCTCP, and data buffer is kept close to zero at all times. Our evaluation with realistic workload shows that ExpressPass significantly reduces flow completion time especially for small to medium size flows when compared to RCP [23], DCTCP [3], HULL [5], and DX [38], and the gap increases with higher link speeds.

2 MOTIVATION

In large datacenter networks, it is not uncommon for a host to generate more than one thousand concurrent connections [3]. The number of concurrent flows traversing a bottleneck link can be large, and flow arrival is often bursty [13] due to the popularity of the partition/aggregate communication pattern [3]. However, existing congestion control algorithms exhibit fundamental limitations under such workload.

First, partition/aggregate patterns cause bursty packet arrivals that result in packet drops (i.e., incast). The problem only worsens with shallow buffered commodity switches that only provide 100 KB of packet buffer per 10 Gbps port [12] as well as in high-speed network (e.g., 100 Gbps) where switch buffer per port per Gbps decreases as link speeds go up [1]. Second, even if congestion control algorithms estimate each flow's fair-share correctly, bursty flow arrival [3, 13] still causes unbounded queue build-up.



(a) Naïve Credit-based (b) TCP Cubic (c) DCTCP

Figure 2: Convergence Time (testbed experiment)

To demonstrate that even an ideal rate control can result in unbounded queue build-up, we assume a hypothetical but ideal congestion control that instantly determines the exact fair-share rate for each sender using a per-packet timer. We then simulate a partition/aggregate traffic pattern using ns-2. A single master server continuously generates a 200 B request to multiple workers using persistent connections, and each worker responds with 1,000 B of data for each request. We increase the fan-out from 32 to 2,048 in an 8-ary fat tree topology with 10 Gbps links with 5 μs delay, 16 core, 32 aggregator, 32 ToR switches and 128 hosts [2].

Figure 1 (a) shows the queue length at the bottleneck link. The bars represent the minimum and maximum queue, and the box shows 25, 50, and 75%-tile values. Even though the senders transmit data at their fair-share rate and packets within the same flow are perfectly paced, the packet queue builds up significantly. This is because while each flow knows how fast it should transmit its own packets, packets from multiple flows may arrive in bursts. In the worst case, the maximum data queue length grows proportionally to the number of flows (depicted by the red line in Figure 1 (a)). Window or rate-based congestion control is far from the ideal case because the congestion control does not converge to the fair-share rate immediately. Figure 1 (b) demonstrates the queue build-up with DCTCP. The average / maximum data queue lengths are much larger than ideal congestion control because it takes multiple RTTs to react to queue build-up.

The result suggests that existing window- or rate-based protocols have fundamental limitations on queuing, and the problem cannot be solved even with a proactive approach, such as PERC [34]. Existing congestion control will exhibit high tail latency. The unbounded queue also forces a packet drop or the use of flow control, such as PFC. Even worse, large queues interact poorly with congestion feedback; when flows are transmitting at the fair-share rate, ECN [3] or delay-based [38, 41] schemes will signal congestion to some flows, resulting in significant unfairness.

Bounded queue build-up: To overcome the fundamental limitation, ExpressPass uses credit-based scheduling in which a sender transmits a data packet only after receiving a credit from the receiver. When a credit packet reaches a sender, the sender transmits a data packet if it has one to send; otherwise, the credit packet is ignored. ExpressPass controls congestion by rate-limiting the credit packets at the switch but without introducing per-flow state. This effectively

[1] Even a deep buffer Arista 7280R switch provides 15% less buffer per Gbps in their 100 Gbps switches compared to that in 10 Gbps switches [9].

[2] Note multiple workers can share the same host, when the number of workers exceeds the number of hosts.

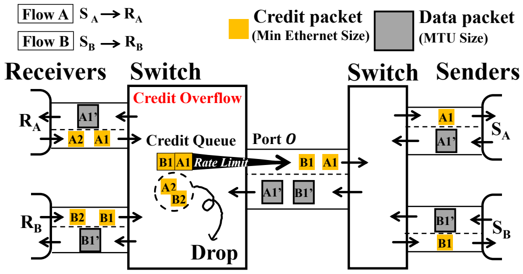


Figure 3: ExpressPass Overview

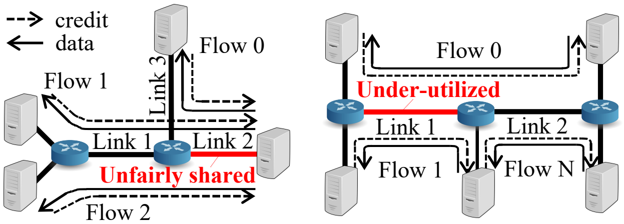
schedules the response packets in the bottleneck link at packet granularity and thus bounds the queue build-up, without relying on a centralized controller [47]. To demonstrate this, we simulate the same partition/aggregate workload using our credit-based scheme. Figure 1 (c) shows the queue build-up. It shows regardless of the fan-out, the maximum queue build-up is bounded.

In a credit-based scheme, the queue only builds up when flows have different round-trip times. Two factors can contribute to RTT differences: 1) the difference in path lengths and 2) the variance in packet processing time at the host. In datacenter environments, both can be bounded. The difference in path lengths is bounded by the network topology. For example, a 3-layered fat tree topology has minimum round-trip path length 4 and maximum 12 between any two pairs of hosts. Note the difference is strictly less than the maximum RTT between any pair of hosts.

The variance in credit processing time can also be bounded. A host discards credit when it does not have any data to send, thus the variance comes only from that of the credit processing delay (e.g., interrupt or DMA latency in a software implementation). In our software implementation on SoftNIC [31], it varies between 0.9 and 6.2 μs (99.99^{th} percentile). Our simulation result in Figure 1 (b) accounts for this variance. The red line shows the maximum queue required considering the two delay factors. Note, a hardware implementation on a NIC can further reduce the variance in credit processing times.

Fast convergence: Another critical challenge for traditional congestion control is quickly ramping up to the fair-share. Fast ramp-up is at odds with low buffer occupancy and risks buffer overflow and packet drops. Thus, in traditional congestion control, it is often a slow, conservative process, which significantly increases the flow completion time. In contrast, credit drop is not as detrimental as data drop, which allows us to send credit packets more aggressively. To demonstrate its potential, we implement a naïve credit-based scheme where a receiver sends credits at its maximum rate. At the bottleneck link, the switch drops excess credit packets using rate-limiting. We use a Pica8 P-3780 10 GbE switch to configure rate-limiting on credit packets. Figure 2 shows the convergence characteristics of a naïve credit-based scheme compared to TCP cubic and DCTCP. It shows the credit-based design can converge to fairness in just one round-trip time, significantly outperforming the TCP variants.

Small RTT and sub-packet regime [16]: Finally, datacenter networks have a small base RTTs around tens of microseconds. Low latency cut-through switches even achieve sub-microsecond latency [8]. Small RTT coupled with a large number of concurrent flows means each flow may send less than one packet per RTT on average [14]—even at 100 Gbps (10 Gbps), 416 (42) flows are enough to reach this point assuming 50 μs RTT. In this regime, window-based protocols break down [14, 16, 32]. Rate-based protocols can increase the probing interval to multiples of RTTs, but this comes at a cost because the ramp-up time also increases significantly. Fundamentally, supporting a high dynamic range of flows efficiently requires a cost-effective mechanism for bandwidth probing. The credit-based approach opens up a new design space in this regard.



(a) Multi-bottleneck topology (b) Parking lot topology

Figure 4: Problems with naïve credit-based approach

3 EXPRESSPASS DESIGN

In ExpressPass, credit packets are sent end-to-end on a per-flow basis. Each switch and the host NIC rate-limit credit packets on a per-link basis to ensure that the returning flow of data does not exceed the link capacity. Symmetric routing ensures data packets follow the reverse path of credit flows (see Section 3.1 for details).

Intuitively, our end-to-end credit-based scheme “schedules” the arrival of data packets at packet granularity, in addition to controlling their arrival rate at the bottleneck link. To show how this mechanism works, we illustrate a simple scenario with two flows in Figure 3, where all links have equal capacity. Consider a time window in which only two packets can be transmitted on the link. Now, receiver R_A and R_B generate credits (A1, A2) and (B1, B2) respectively at the maximum credit rate. All four credit packets arrive at output port O, where credit packets are *rate-limited* to match the capacity of the reverse link capacity. Thus, half the credits are dropped at the output port. In this example, each sender gets one credit and sends one data packet back to the receivers. Note, the senders generate exactly two data packets that can go through the bottleneck link during the time window. In addition, in an ideal case where the RTTs of the two flows are the same, the data packets do not experience any queuing because their arrival at the bottleneck link is well-scheduled.

Design challenges: While the credit-based design shows promising outlook, it is not without its own challenges. One might think that with the credit-based scheme a naïve approach in which a receiver sends credit packets as fast as possible can achieve fast convergence, high utilization, and fairness all at the same time. However, the naïve approach has serious problems with multiple bottlenecks. First, it does not offer fairness. Consider the multi-bottleneck topology of Figure 4 (a). When all flows send credit packets at the maximum rate, the second switch (from the left) will receive twice as many credit packets for Flow 0 than Flow 1 and Flow 2. As a result, Flow 0 occupies twice as much bandwidth on Link 2 than others. Second, multi-bottlenecks may lead to low link utilization. Consider the parking lot topology of Figure 4 (b). When credit packets are sent at full speed, link 1’s utilization drops to 83.3%. This is because, after 50% of Flow 0’s credit passing link 1 (when competing with Flow 1), only 33.3% of credit packets go through Link 2, leaving the reverse path of Link 1 under-utilized by 16.6%. Finally, in large networks, the RTTs of different paths may differ significantly. This may break the scheduling of data packets, which leads to a queue build-up.

Achieving high utilization, fairness, and fast convergence at the same time is non-trivial and requires careful design of a credit feedback loop. In addition, we must limit the queue build-up to ensure zero data loss. Next, we present the design details and how the end-hosts and switches work together to realize the goals.

3.1 Switch and End-Host Design

Credit rate-limiting (switch and host NIC): For credit packets, we use a minimum size, 84 B Ethernet frame, including the preamble and inter-packet gap. Each credit packet triggers a sender to transmit up to a maximum size Ethernet frame (e.g., 1538 B). Thus, in Ethernet, the credit is rate-limited to $84/(84+1538) \approx 5\%$ of the link capacity, and the remaining 95% is used for transmitting data packets. The host and switch perform credit rate-limiting at each switch port. The host also tags credit packets so that switches can classify them and apply rate-limiting on a separate queue. To pace credit packets and limit their maximum burst size, we apply a leaky bucket available on commodity switch chipsets (e.g., maximum bandwidth metering on Broadcom chipsets). At peak credit-sending rate, credits are spaced apart by exactly 1 MTU in time (last byte to the first byte). Because data packets are not always the full MTU in size, two or more data packets can be transmitted back to back, and by the time a credit is sent there may be additional tokens accumulated for a fraction of a credit packet. Increasing the token bucket to the size of 2 credit packets ensures these fractional amounts are not discarded so that credit sending rate becomes nearly the maximum on average.

Finally, to limit the credit queue, we apply buffer carving [19] to assign a fixed buffer of four to eight credit packets to the class of credit packets.

Ensuring path symmetry (switch): Our mechanism requires path symmetry—data packet must follow the reverse path of the corresponding credit packet. Datacenter networks with multiple paths (e.g., Clos networks) often utilize equal-cost multiple paths. In this case, two adjacent switches need to hash the credit and data packets of the same flow onto the same link (in different directions) for symmetric routing. This can be done, in commodity switches, by using symmetric hashing with deterministic Equal Cost Multi-Path (ECMP) forwarding. Symmetric hashing provides the same hash value for bi-directional flows [15, 17, 45], and deterministic ECMP sorts next-hop entries in the same order (e.g., by the next hop addresses) on different switches [21, 51]. Finally, it requires a mechanism to exclude links that fail unidirectionally [52]. Note path symmetry does not affect performance. Even with DCTCP, the utilization and performance on fat tree topology are not affected by path symmetry in our simulations.

Ensuring zero data loss (switch): Rate-limiting credit packets controls the rate of data in the reverse path and makes the network congestion-free. However, the difference in RTT can cause transient queue build-up. Fortunately, the maximum queue build-up is bounded in ExpressPass. We apply network calculus [37] to determine the bound. Note this bound is equivalent to the buffer requirement to ensure zero-data loss.

Topology (link/core-link speed)	ToR down	ToR up	Core
(Core/Aggr./ToR/Server)	(norm. by DCTCP K)		
32-ary fat tree (10/40 Gbps) (16 / 512 / 512 / 8,192)	577.3 KB (5.77)	19.0 KB (0.19)	131.1 KB (0.33)
32-ary fat tree (40/100 Gbps) (16 / 512 / 512 / 8,192)	1.06 MB (2.65)	37.2 KB (0.09)	221.8 KB (0.22)
3-tier Clos (10/40 Gbps) (16 / 128 / 1024 / 8,192)	577.3 KB (5.77)	19.0 KB (0.19)	131.1 KB (0.33)
3-tier Clos (40/100 Gbps) (16 / 128 / 1024 / 8,192)	1.06 MB (2.65)	37.2 KB (0.09)	221.8 KB (0.22)

Table 1: Required buffer size for ToR down ports, ToR up ports, and Core ports with datacenter topology.

Given a network topology, we calculate the bound for each switch port. Let us denote d_p as the delay between receiving a credit packet and observing the corresponding data packet at a switch port p, and Δd_p as the delay spread of d_p (i.e., the difference between maximum and minimum of d_p). Δd_p is determined by the network topology and the queuing capacity of the network. Δd_p represents the maximum duration of data buffering required to experience zero loss because, in the worst case, Δd_p time unit worth of data can arrive at the same time at port p. For a given credit ingress port p, its delay, d_p, consists of four factors: (1) the delay caused by the credit queue, d_{credit}; (2) the switching, transmission, and propagation delay of credit and data packet to/from ingress port q of the next hop, $t(p,q)$; (3) the delay at ingress port q of the next hop, d_q; and (4) returning data packet's queuing delay at port q, $d_{data}(q)$ whose maximum is determined by Δd_q.

$$d_p = d_{credit} + t(p,q) + d_q + d_{data}(q)$$

Then, given an ingress port p and its set of possible next-hop ingress ports $N(p)$, its delay spread Δd_p becomes:

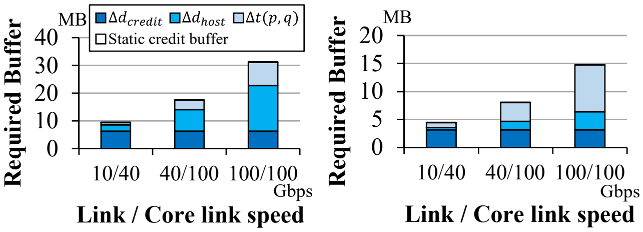
$$\begin{aligned} \Delta d_p =& \max(d_{credit}) + \max_{q \in N(p)} (t(p,q) + d_q + \Delta d_q) \\ &- \min_{q \in N(p)} (t(p,q) + d_q), \end{aligned} \tag{1}$$

In datacenters, switches are often constructed hierarchically (e.g., ToR, aggregator, and core switches). Within a hierarchical structure, the delays can be computed in an iterative fashion. For example, if we know the min/max delay of NIC ports, we can get the min/max delay and the delay spread for uplink ports in a ToR switch. At NIC, Δd_p is the same as the delay spread of host processing delay, Δd_{host}, which is a constant given a host implementation.

We can then compute the min/max delay and the delay spread for uplink ports in ToR and aggregator switches.

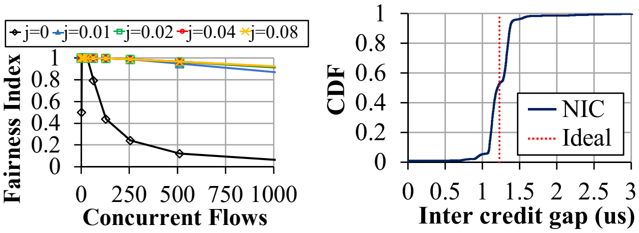
The delay spread accumulates along the path. In hierarchical multi-rooted tree topologies, traffic from a downlink is forwarded both up and down, but traffic from an uplink is only forwarded down. Therefore, for uplink ports, the set of next hop's ingress ports $N(p)$ only includes the switch/NIC ports at lower layers, whereas for downlink ports $N(p)$ includes ports at both lower and upper layers. As a result, uplink ports require smaller buffer than downlink ports. ToR downlink has the largest path length variance, thus has the largest buffer requirement.

Table 1 shows the buffer per port requirement for different topologies. We assume a credit queue capacity of 8 credit packets (see Section 3.2) and propagation delay of 5 μs for core links and 1 μs for



(a) 8 credit queue, $\Delta d_{host} = 5\mu s$ (b) 4 credit queue, $\Delta d_{host} = 1\mu s$

Figure 5: Maximum buffer for ToR switch in 32-ary fat tree



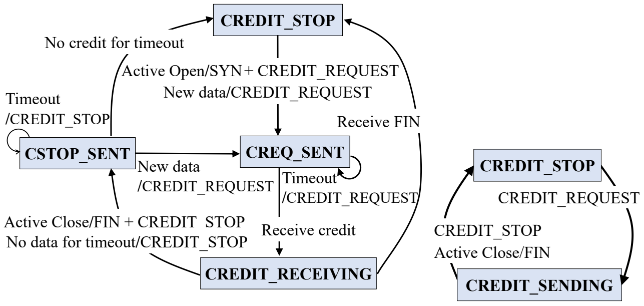
(a) Jitter vs. Fairness (b) Inter-credit Gap at NIC

Figure 6: Required jitter for fair credit drop and measured inter-credit gap at NIC

others. For the host processing delay spread, we use the measurement result from our testbed (shown in Figure 14 (a)). To compare buffer requirement of ExpressPass to DCTCP, we also show the value normalized to DCTCP's ECN marking threshold K as recommended in the DCTCP paper [3]. Note, for ExpressPass, the result shows the buffer requirement for operating the network without any data packet loss, whereas DCTCP's marking threshold represents the average queue size for DCTCP. The maximum buffer requirement is a very conservative bound assuming the worst case.

It is required when a part network that has the longest delay has a full (credit and data) buffer, while the path with the shortest delay has zero buffer. This is an unlikely event that only happens in a network with significant load imbalance. Under realistic workloads, ExpressPass uses only a fraction of the data queue as we show in Section 6. Finally, ExpressPass's correct operation does not depend on zero loss of data packets (i.e., it operates even with smaller buffers).

Three main factors impact the required buffer size for zero data loss: delay spread of host processing, credit buffer size, and link speed. Figure 5 shows the breakdown of maximum buffer for a ToR switch by the each contributing source in 32-ary fat tree (8, 192 hosts) topologies with (10/40), (40/100), and (100/100) Gbps (link/core link) speeds. We use two sets of parameters: a) 8 credit queue and $\Delta d_{host} = 5.1\,\mu s$ reflect our testbed setting; b) 4 credit queue and $\Delta d_{host} = 1\,\mu s$ represent a case where ExpressPass is implemented in NIC hardware. Smaller credit queue capacity and host delay spread results in a smaller data queue. The required buffer space for zero-loss increases sub-linearly with the link capacity. Note, a ToR switch has the largest buffer requirement among all. Even then, its requirement is modest in all cases. Today shallow buffered 10 GbE switches have 9 to 16 MB of shared packet buffers and 100 GbE switches have 16 MB to 256 MB [10, 54], whereas deep buffered switches have up to 24 GB of shared packet buffer [54].



(a) Sender (b) Receiver

Figure 7: State transition of sender and receiver.

Ensuring fair credit drop (end-host): ExpressPass relies on the uniform random dropping of credit packets at the switch to achieve fairness—if n flows are sending credits at the same rate to the shared bottleneck, equal fraction of credit packets must be dropped from each flow. Unfortunately, subtle timing issue can easily result in a skewed credit drop with drop-tail queues. When the credit buffer is (nearly) full, the order of credit packet arrival determines which ones get dropped. Thus, synchronized arrival can result in significant unfairness. It is particularly important because we use a tiny credit buffer (see Section 3.2).

To address this issue, we introduce random jitter in sending credit packets at the end-host, instead of perfectly pacing them. To determine how much jitter is required for fairness, we create a number of concurrent flows (between 1 and 1024) that traverse a single bottleneck link. We vary the jitter level, j, from 0.01 to 0.08 relative to the inter-credit gap (e.g., the jitter is between 13 ns and 104 ns for 10 GbE) and measure the Jain's fairness index [33] over an interval of 1 ms. Figure 6 (a) shows the result. We observe that perfect pacing causes significant unfairness due to exact ordering (fairness index of 1.0 means perfect fairness). However, even small jitter (tens of nanosecond) is enough to achieve good fairness as it breaks synchronization. We also find, in our prototype implementation, the natural jitter at the host and NIC is sufficient enough to achieve fairness. Figure 6 (b) plots the CDF of inter-credit gap measured using an Intel X520 10 GbE NIC, when credit packets are sent at the maximum credit rate with pacing performed in SoftNIC. The inter-credit gap has a standard deviation of 772.52 ns, which provides a sufficient degree of randomization.

However, synchronized credit drop can also arise when credits from multiple switches compete for a single bottleneck. The jitter at the end host is not sufficient when the credit queues at the switches are not drained for a long period of time. In large scale simulations, we observe that some flows experience excessive credit drop while other flows make progress. To ensure fairness across switches, we randomize the credit packet sizes from 84 B to 92 B. This effectively creates jitter at the switches and breaks synchronization. With these mechanisms, ExpressPass provides fair credit drop across flows without any per-flow state or queues at the switches.

Starting and stopping credit flow (end-host): Finally, ExpressPass requires a signaling mechanism to start the credit flow at the receiver. For TCP, we piggyback credit request to either SYN and/or SYN+ACK packet depending on data availability. This allows data

Algorithm 1 Credit Feedback Control at Receiver.

```
1:  $w \leftarrow w_{init}$
2:  $cur\_rate \leftarrow initial\_rate$
3:  repeat per update period (RTT by default)
4:      $credit\_loss = \#\_credit\_dropped / \#\_credit\_sent$
5:      if $credit\_loss \leq target\_loss$ then
6:          (increasing phase)
7:          if previous phase was increasing phase then
8:              $w = (w + w_{max})/2 \quad (w_{max} = 0.5)$
9:          $cur\_rate = (1 - w) \cdot cur\_rate$
                    $+ w \cdot max\_rate \cdot (1 + target\_loss)$
10:     else
11:         (decreasing phase)
12:         $cur\_rate = cur\_rate \cdot (1 - credit\_loss) \cdot (1 + target\_loss)$
13:         $w = \max(w/2, w_{min}) \quad (0 < w_{min} \leq w_{max})$
14: until End of flow
```

transmission to start immediately after the 3-way handshake. For persistent connections or UDP, we send credit requests in a minimum sized packet, but this adds a delay of an RTT. At the end of the flow, if there is no data to send for a small timeout period, the sender transmits a CREDIT_STOP to the receiver, and the receiver stops sending credits after receiving it [3]. Figure 7 depicts the state transition diagram.

3.2 Credit Feedback Control

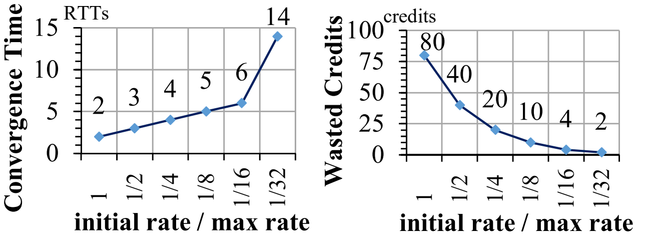
Fast convergence, utilization, and fairness present challenging trade-offs in congestion control. In our credit-based scheme, considering only one (e.g., fast convergence) results in an undesirable outcome as shown in Section 2. To address this, we introduce a credit feedback loop where credit sending rate is dynamically adjusted. Our feedback loop strives to achieve high utilization, fairness, and fast convergence.

One question is: *how does it differ from existing data packet feedback control?* We first illustrate two key differences and describe the feedback design.

First, in the credit-based scheme, one can ramp up a flow's credit sending rate much more aggressively because ramping up the credit sending rate does not result in data loss. In traditional design, ramping up the flow rate had to be done carefully in a conservative fashion because it might introduce data loss, which greatly increases the flow completion time due to retransmission. Our algorithm leverages aggressive ramp-up of credit sending rate to achieve fast convergence.

Second, we fully leverage the fact that the cost of overshooting the credit sending rate is low. At steady state, overshoot targets a small credit loss rate (e.g., 10%). This allows ExpressPass to instantly detect and fill up additional available bandwidth when bottleneck link starts to free up, which achieves high utilization.

Feedback control algorithm: At a high-level, our feedback algorithm uses binary increase, similar to BIC-TCP [56], with the target rate set to the link capacity. The aggressive increase ensures fast convergence. Fairness is achieved because the flow with lower rate increases its rate more than the competing flow with the higher rate. However, the aggressive binary increase does not converge and results in oscillation. To improve the stability, we dynamically adjust the aggressiveness factor, w.



(a) Convergence Time (b) One packet flow credit waste

Figure 8: Convergence time and credit waste trade-offs.

ExpressPass uses the credit loss rate as the congestion indicator. For credit loss rate measurement, each credit packet carries a sequence number, and the data packet echoes back the credit sequence number. A gap in the sequence number indicates credit packets have been dropped in the network. When congestion is detected, ExpressPass reduces the credit sending rate to the credit rates that passed through the bottleneck during the last RTT period.

Algorithm 1 describes the credit feedback control. If *credit_loss* is less than or equal to the target loss rate (*target_loss*), the feedback control concludes that the network is under-utilized and goes through the increasing phase (line 6 to 9). It increases *cur_rate* by computing a weighted average of *cur_rate* and *max_rate*$\cdot(1+$*target_loss*$)$ using an aggressiveness factor w ($0 < w \leq 0.5$) as the weight, where *max_rate* indicates the maximum credit rate for the link. Otherwise (i.e., when *credit_loss* > *target_loss*), the feedback control detects congestion and goes through the decreasing phase, where it reduces the credit sending rate (*cur_rate*) so that the credit loss rate will match its target loss rate at the next update period assuming all flows adjust in the same manner.

The feedback loop dynamically adjusts the aggressiveness factor w (between w_{min} and w_{max} or 0.5) to provide a balance between stability and fast convergence. When w is large (small), the credit sending rate ramps up more (less) aggressively. Thus, when congestion is detected, we halve w in the decrease phase. When no congestion is detected for two update periods, we increase w by averaging its current value and the maximum value, 0.5. At steady state, a flow experiences increase and decrease phase alternatively, and as a result, w decreases exponentially. This achieves stability as we show in Section 4. Finally, w_{min} provides a lower bound, which keeps w from becoming infinitesimally small. In all our experiments, we use w_{min} of 0.01. Setting w_{min} involves in a trade-off between a better steady state behavior and fast convergence, as we show through analysis in Section 4.

3.3 Parameter choices

Initial rate and w_{init} determine how aggressively a flow starts. They decide the trade-off between small flow FCT and credit waste. Setting them high allows flows to use as much bandwidth as possible from the beginning. However, a flow with only a single packet to send will waste all-but-one credits in the first RTT. Setting them low reduces the credit waste, but increases the convergence time

[3] This cause some credits to be wasted. We quantify the overhead and discuss mitigation strategies in Section 7.

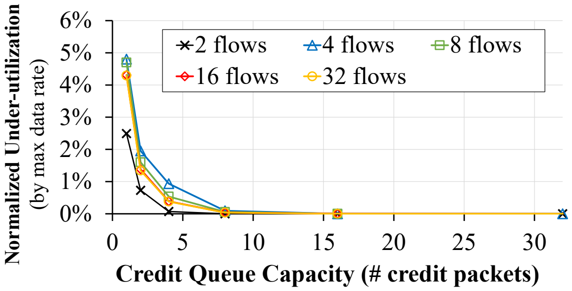
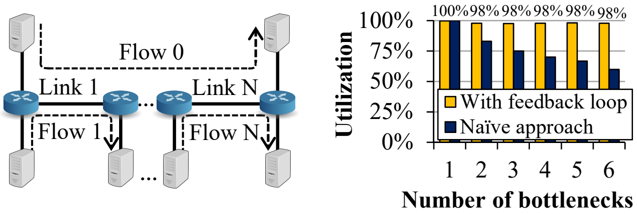


Figure 9: Credit queue capacity vs. Utilization



(a) Scenario

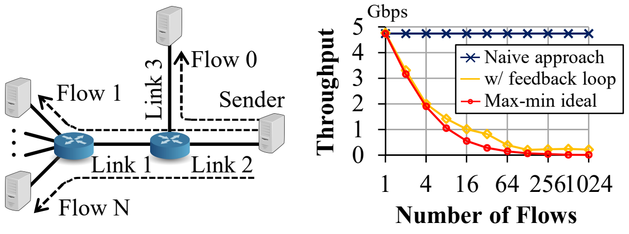
(b) Link Utilization (normalized by max data rate)

Figure 10: Utilization with parking-lot topology

(or ramp-up time). To understand the trade-offs, we measure the convergence time of a new flow competing with an existing flow by varying the initial rate from $\frac{1}{32} \times max\_rate$ to max_rate while w_{init} is fixed to 0.5. Figure 8 (a) shows that as the initial rate decreases, the convergence time increases from 2 RTTs to 14 RTTs. Figure 8 (b) shows the amount of credit waste with single packet flows in an idle network whose RTT is 100 μs. As the initial rate decreases, the amount of wasted credits decline as expected. Note that w_{init} does not affect the credit waste for a single packet flow. We further analyze the trade-offs with realistic workloads later in Section 6.3.

Target loss: One important role of the target loss factor is to compensate for subtle timing mismatch. For example, a receiver could send N credits over prior RTT and only receives N-1 data packets due to subtle timing mismatch, but we do not want our feedback control to overreact to such cases. While targeting some loss may appear to introduce more credit waste, it usually does not because credits delivered to the sender will be used if data is available. However, with multiple bottlenecks, setting the target loss rate higher risks under-utilization. Thus, we use a small target loss rate of 10%.

Credit queue capacity: The size of the credit buffer affects utilization, convergence time, and data queue occupancy. A large credit buffer hurts fast convergence as it delays feedback and increases the delay spread and queue occupancy for data packets. However, too small credit buffer can hurt utilization because credit packets can arrive in bursts across different ports and get dropped. We quantify how much credit queue is necessary to ensure high utilization. For this, we conduct an experiment with our feedback control while varying the number of flows from 2 to 32 which all arrive from different physical ports and depart through the same port. We then vary the credit queue size from 1 to 32 packets in power of 2 and measure the corresponding under-utilization in Figure 9. It shows that credit buffer size of eight is sufficient across the different number of flows. Hence, we set the credit buffer to eight for rest of the paper.



(a) Scenario

(b) Throughput of Flow 0

Figure 11: Fairness in multi-bottleneck topology

3.4 Effect of feedback control

The feedback loop significantly improves utilization with multiple bottlenecks by reducing wasted credits for long flows when flows traverse multiple hops. Here, we quantify this using the topology of Figure 10 (a). We increase the number of bottleneck links, N, from one to six. Figure 10 (b) shows the utilization of the link with the lowest utilization. To isolate the loss due to multiple bottlenecks, we report the utilization normalized to the maximum data rate excluding the credit packets. Our feedback control achieves high utilization even with multiple bottlenecks. With two bottlenecks, it achieves 98.0% utilization, an improvement from 83.3% the naïve case without feedback control; and with six bottlenecks, it achieves 97.8% utilization, an improvement from 60% in the naïve case.

Our feedback control also improves fairness with multiple bottlenecks. To demonstrate this, we use the multi-bottleneck scenario in Figure 11 (a) and vary the number of flows (N) that use Link 1. We then measure the throughput of Flow 0. Note, all flows are long-running flows and Flow 1 through N experience multiple bottlenecks but Flow 0 only has a single bottleneck. Figure 11 (b) shows the throughput of Flow 0 as the number of competing flows increases. With ideal max-min fairness, Flow 0 should get 1/(N+1) of the link capacity (red line in Figure 11 (b)). ExpressPass follows the max-min fair-share closely until four flows. But, as the number of flows increases, it shows a small gap. There are many factors that contribute to fairness. One important factor is when there is less than one credit packets per RTT. ExpressPass can still achieve good utilization and bounded queue, but fairness deteriorates in such cases.

4 ANALYSIS OF EXPRESSPASS

We now provide analysis of our feedback control. We prove that it converges to fair-share and analyze stability using a discrete model. For analysis, we assume N (> 1) flows are sharing a single bottleneck link and their update periods are synchronized as in many other studies [4, 46, 59].

Stability analysis: Let us denote the credit sending rate of flow n at time t by $R_n(t)$, and its aggressiveness factor w by $w_n(t)$. The maximum credit sending rate corresponding to the link capacity is denoted as max_rate. We define C as $C = max\_rate \cdot (1 + target\_loss)$, which is the maximum credit sending rate for a flow.

Without loss of generality, assume the bottleneck link is underutilized. Then, $credit\_loss$ will be zero for all flows, and they will increase their credit sending rates. Eventually, the aggregate credit sending rate ($\sum_{i=1}^{N} R_i(t)$) will exceed the maximum credit sending rate, C. In the next update period $t = t_0$, it will reach C according to the decreasing phase (Algorithm 1 line 12) that reduces cur_rate by

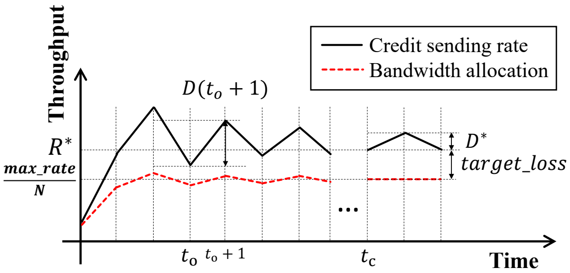


Figure 12: Steady state behavior

$(1 - credit\_loss) \cdot (1 + target\_loss)$. At time $t_0 + 1$, *increasing phase* of feedback control is triggered, and the aggregate credit sending rate becomes:

$$\begin{aligned}\sum_{i=1}^{N} R_i(t_0+1) &= \sum_{i=1}^{N}\{(1-w_i(t_0+1))\cdot R_i(t_0) + w_i(t_0+1)C\} \\ &> \sum_{i=1}^{N}\{(1-w_{min})\cdot R_i(t_0) + w_{min}C\} > C\end{aligned} \tag{2}$$

The aggregate credit sending rate now exceeds C. Then, at $t_0 + 2$, it becomes C again. From the time t_0, all flows will experience increasing phase and decreasing phase alternatively. Thus, we get the following equations from algorithm 1 line 12 and 9:

$$\begin{aligned}R_n(t_0+1) =&(1-w_n(t_0))R_n(t_0) + w_n(t_0)C \\ R_n(t_0+2) =&\frac{C}{\sum_{i=1}^{N} R_i(t_0+1)} R_n(t_0+1) \\ =&\frac{(1-w_n(t_0))R_n(t_0) + w_n(t_0)C}{1 - \frac{1}{C}\sum_{i=1}^{N} w_i(t_0)R_i(t_0) + \sum_{i=1}^{N} w_i(t_0)} \\ =&\frac{1}{A(t_0)}\{(1-w_n(t_0))R_n(t_0) + w_n(t_0)C\}\end{aligned}$$

$$A(t_0) := 1 + \sum_{i=1}^{N} w_i(t_0)\{1 - \frac{R_i(t_0)}{C}\}$$

Solving the recurrence equations yields (for $k > 0$):

$$\begin{aligned}R_n(t_0+2k) \approx& A^{-k}(t_0)(1-w_n(t_0))^k \cdot R_n(t_0) \\ &+ \frac{w_n(t_0)\cdot C}{A(t_0) - (1-w_n(t_0))}\end{aligned} \tag{3}$$

$$\begin{aligned}R_n(t_0+2k+1) \approx& (1-w_n(t_0+2k)) \cdot R_n(t_0+2k) \\ &+ w_n(t_0+2k)\cdot C\end{aligned} \tag{4}$$

The aggressiveness factor $w_n(t)$ halves every two update periods and eventually it converges to w_{min} (Algorithm 1 line 8, 11-12).

$$w_n(t+1) = w_n(t), \quad w_n(t+2) = \max(\frac{1}{2}w_n(t), w_{min})$$

Let us denote the time when $w_n(t)$ have converged to w_{min} by t_c; $w_n(t_c + n) = w_{min}$. Equation 3 and 4 still hold at t_c:

$$R_n(t_c+2k) = A^{-k}(t_c)(1-w_{min})^k \cdot R_n(t_c) + \frac{C}{N}$$

$$R_n(t_c+2k+1) = (1-w_{min})R_n(t_c+2k) + w_{min}C$$

Because $0 < w_{min} \le w_n(t_0) \le 0.5$, $N > 1$ and $C > 0$, $A(t_0) > 1$ and $0.5 \le (1 - w_{min}) < 1$. Thus, we get:

$$R_n(t_c+2k) \rightarrow \frac{C}{N} \tag{5}$$

$$R_n(t_c+2k+1) \rightarrow \frac{C}{N}(1 + (N-1)w_{min}) \tag{6}$$

We have shown that regardless of the initial credit sending rate and the initial aggressiveness factor w_n, $R_n(t + 2k)$ converges to C/N and $R_n(t + 2k + 1)$ is bounded. Thus, our feedback control is stable. Below, we also show the difference between $R_n(t + 2k)$ and $R_n(t + 2k + 1)$ is bounded.

Convergence to fairness: Now, we show the bandwidth allocation B_n converges to fairness. From Equations 5 and 6, we see that all flows have the same credit sending rate. Therefore, the bandwidth allocation converges to fairness.

Figure 12 illustrates the convergence behavior. Let's denote the fair-share rate of a flow as R^* $(=\frac{C}{N})$. Let us denote the difference of credit sending rate at time t and time $t - 1$ as $D(t)$. It follows that:

$$\begin{aligned}D(t_0+2k+1) &= |R_n(t_0+2k+1) - R_n(t_0+2k)| \\ &\approx |\max(2^{-k}w_n(t_0), w_{min}) \cdot \{C - R_n(t_0+2k)\}|\end{aligned}$$

Similarly, we can compute $D(t_0 + 2k + 2)$. $D(t)$ is an exponentially decreasing function that eventually converges to:

$$D^* = C \cdot w_{min}(1 - \frac{1}{N})$$

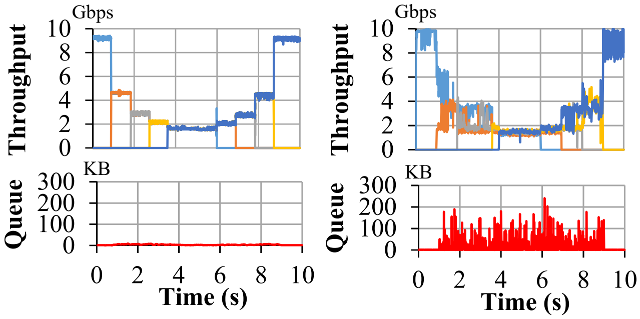
Note $C = max\_rate \cdot (1 + target\_loss)$. Thus, D^* depends on w_{min} and *target_loss*; a small w_{min} and *target_loss* results in a small D^* value, which improves the steady state behavior. However, a small w_{min} can cause delayed convergence by making $(1 - w_{min})$ larger in Equation 4.

5 IMPLEMENTATION

We leverage SoftNIC [31] to implement ExpressPass. SoftNIC is a framework that allows developers to extend NIC functionality by exploiting high-performance kernel bypass I/O framework (DPDK). SoftNIC enables us to pace credits with microseconds level accuracy and react to the credit packets with a small delay of few microseconds. SoftNIC also enables legacy applications and kernel networking stacks to be used without modification. Applications run transparently on top of SoftNIC using TCP/IP. We implement ExpressPass as a module inside SoftNIC. ExpressPass module adjusts the credit sending rate according to the feedback algorithm and paces the transmission of credit packets. Data packets are sent only when the ExpressPass module receives a credit packet. To start and stop the transmission of credit packets, we implement the per-flow state machine in Figure 7. For example, when the NIC receives data to transmit from the kernel, it generates credit requests. Upon reception of a credit packet, it offers a data packet if available.

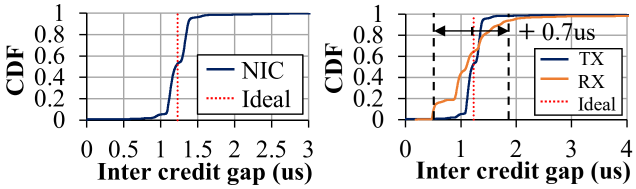
For rate-limiting credit packets, we leverage the scalable rate limiter implementation of SoftNIC. It is able to handle thousands of concurrent flows while introducing only a few microseconds of jitter [31]. For rate-limiting in the switch, we leverage maximum bandwidth metering available in Broadcom chipsets. We create separate per-port queues for credits. Then, we set the maximum burst size as 2 credits and the queue occupancy as 8 credit packets.

Credit processing and rate-limiting: To validate our implementation and setup, we measure 1) the host credit processing time of our implementation and 2) the accuracy of credit rate-limiting at the switch. We connect two hosts using a ToR switch (Quanta T3048-LY2). Figure 14 (a) shows the CDF of credit processing latency.



(a) ExpressPass (b) DCTCP

Figure 13: Convergence behavior (Testbed)



(a) Credit processing dealy measured at host (b) Inter-credit gap before/after passing the switch

Figure 14: Host and switch performance (Testbed)

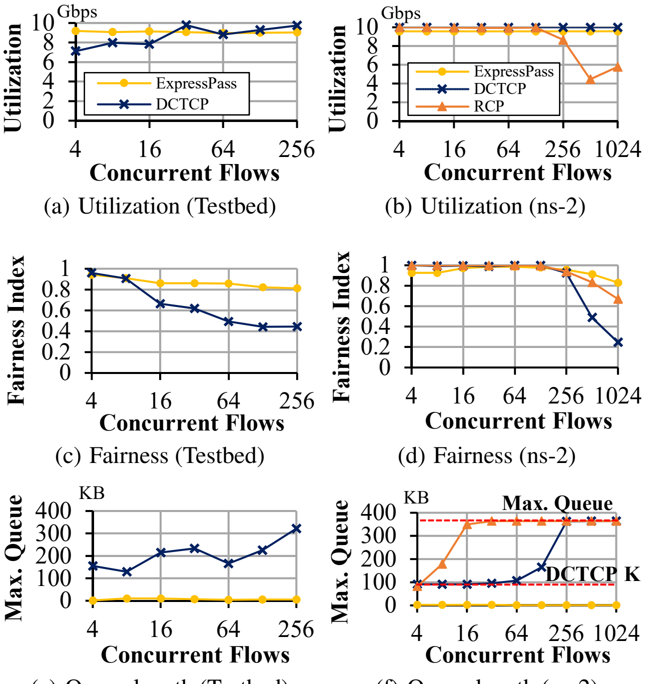
The median is about 0.38 μs, but 99.99th percentile is large at 6.2 μs. We believe a hardware implementation will have a much smaller variance, reducing the buffer requirement for ExpressPass. RDMA NICs implementing iWARP exhibit a maximum delay spread (jitter) of 1.2 μs [18]. In addition, all the core components of ExpressPass, including pacers [5], rate-limiters [48, 49, 58], and credit-based flow control, have been implemented on NIC. Figure 14 (b) shows the inter-credit gap at transmission at the sender and reception at the receiver. We timestamp credit packets at SoftNIC right before transmission and right after the reception. We observe that the jitter of the inter packet gap is relatively small (within 0.7 μs) which implies that software implementation is efficient enough to generate and receive credit packets at 10 Gbps.

Convergence characteristics: To demonstrate ExpressPass works end-to-end, we run a simple test with five flows that arrive and depart over time. Figure 13 shows the throughput averaged over 10 *ms* and the queue size measured at the switch every 30 *ms*. ExpressPass achieves a much more stable steady state behavior and exhibits a very small queue. The maximum ExpressPass data throughput is 94.82% of the link capacity because 5.18% of the bandwidth is reserved for credit. The maximum queue size observed was 18 KB for ExpressPass and 240.7 KB for DCTCP. The maximum credit queue size of ExpressPass was only 672 B (8 packets).

6 EVALUATION

We evaluate three key aspects of ExpressPass using testbed experiments (Section 6.1) and ns-2 simulations [39] (Section 6.1 - 6.3):

(1) We evaluate the flow scalability of ExpressPass in terms of convergence speed, fairness, utilization, and queuing.
(2) We measure the effectiveness of ExpressPass under heavy incast.
(3) We quantify the benefits and the trade-offs of ExpressPass using realistic datacenter workloads.



(a) Utilization (Testbed) (b) Utilization (ns-2)

(c) Fairness (Testbed) (d) Fairness (ns-2)

(e) Queue length (Testbed) (f) Queue length (ns-2)

Figure 15: Queue length / fairness / utilization with many concurrent flows

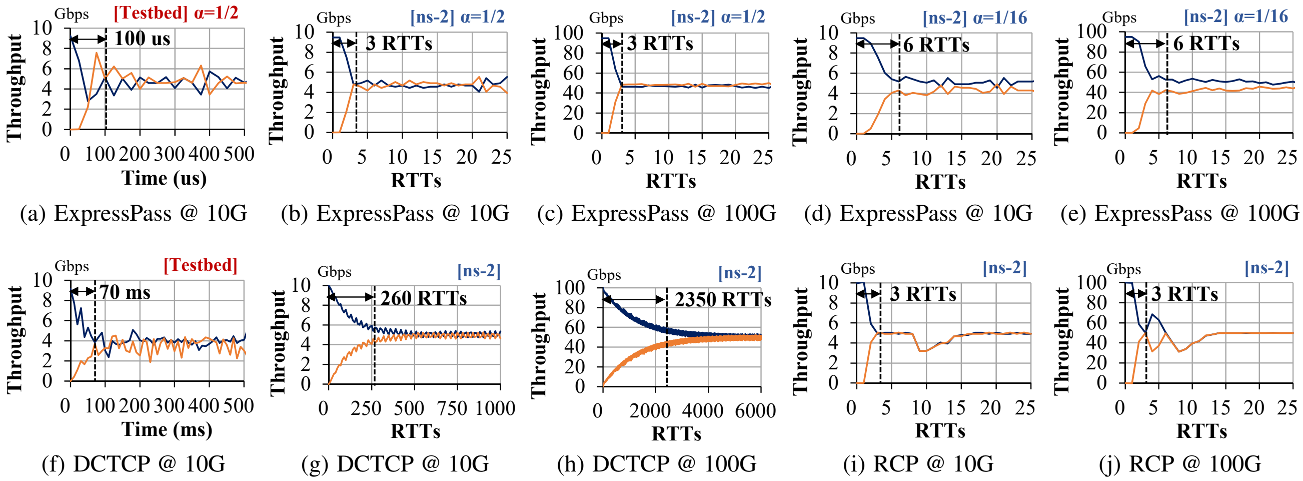
6.1 Microbenchmark

In this section, using simple synthetic microbenchmark we verify whether ExpressPass behaves as designed, quantify its benefits, and compare it against the DCTCP [3] and the RCP [23].

Flow scalability: Flow scalability is an important problem because datacenters have very small BDP on the order of 100 packets. Yet, it needs to support various scale-out workloads that generate thousands of concurrent connections [25]. To test the flow scalability of ExpressPass, we run a simple experiment using a dumbbell topology where N pairs of sender and receiver share the same bottleneck link. We vary the number of flows from 4 to 256 (testbed) and 1024 (ns-2), and measure utilization, fairness and queuing. Note, these flows are long running flows whose arrival times are *not* synchronized. We experiment with both testbed and simulation for cross-validation. For testbed experiments, we use 12 machines to generate traffic, where each sender may generate more than one flow.

First, we measure the utilization. ExpressPass achieves approximately 95% of utilization due to the reserved bandwidth for credits. DCTCP achieves 100% utilization in all cases. RCP has underutilization beyond 256 flows. In the testbed, DCTCP shows slightly lower utilization when the number of flows is small. We suspect high variation in kernel latency as the cause.

Second, we measure the fairness to evaluate how fairly bandwidth is shared across flows. We compute the Jain's fairness index using the throughput of each flow at every 100 *ms* interval and report the average. With a large number of flows DCTCP's fairness drops significantly. Because DCTCP cannot handle a congestion window of less than 1, some flows time out and eventually collapse. In contrast, both ExpressPass and RCP achieve very good fairness.



(a) ExpressPass @ 10G (b) ExpressPass @ 10G (c) ExpressPass @ 100G (d) ExpressPass @ 10G (e) ExpressPass @ 100G

(f) DCTCP @ 10G (g) DCTCP @ 10G (h) DCTCP @ 100G (i) RCP @ 10G (j) RCP @ 100G

Figure 16: Convergence time of ExpressPass, DCTCP and RCP at 10/100 Gbps bottleneck link

Finally, Figure 15 (e) and (f) show the maximum queue occupancy at the bottleneck as the number of flows increases. ExpressPass shows maximum queuing of 10.5 KB and 1.34 KB in the testbed and ns-2 respectively. In contrast, DCTCP's max queue occupancy increases with the number of flows. In our simulation, when the maximum queue hits the queue capacity, packet drop occurs. The experimental results show a similar trend. DCTCP's congestion control starts to break around 64 flows. When the number of concurrent flows is larger than 64, most flows stay at the minimum congestion window of 2 because the queue size is always larger than the marking threshold. However, note the maximum queue length for 16 and 32 flows are higher than that of 64 flows. For 16 and 32 flows, some flows may occasionally ramp up when they do not get an ECN signal, which builds up the queue. RCP exceeds the queue capacity and packet drop occurs even with 32 flows. This is because RCP assigns the same rate for a new flow as existing flows when new flows start.

Fast convergence: Figure 16 shows the convergence behavior of ExpressPass, DCTCP, and RCP over time. First, we use testbed experiments to compare ExpressPass and DCTCP at 10 Gbps. ExpressPass's throughput is averaged over 25 μs and DCTCP is averaged over 100 *ms* due to its high throughput variance. As shown in Figure 16 (a) and (b), ExpressPass converges 700x faster than DCTCP in just 100 μs (four RTTs), while DCTCP took 70 *ms* to converge. Two factors contribute to the difference. First, convergence is much faster in ExpressPass than the DCTCP which performs AIMD. Second, ExpressPass shows RTT of 10 μs at the minimum and 25 μs on average, measured in SoftNIC. On the other hand, DCTCP's feedback loop runs much slower in the Linux kernel, which adds hundreds of microseconds RTT variation [38].

Next, we use simulation to compare the congestion feedback algorithm of ExpressPass and DCTCP in isolation. We compare the convergence time of ExpressPass and DCTCP on two different link speeds (10 Gbps and 100 Gbps). The base RTT is set to 100 μs. Figure 16 (c) - (f) shows the flows' throughput for ExpressPass and DCTCP at each RTT [4]. ExpressPass converges within 3 RTTs which is consistent with 4 RTTs in experiments. DCTCP takes more than 80 times longer than ExpressPass with 10 Gbps link. As bottleneck link capacity increases, the convergence time gap between ExpressPass and DCTCP becomes larger. At 100 Gbps, ExpressPass's convergence time remains unchanged, while that of DCTCP grows linearly to the bottleneck link capacity. Because of DCTCP's additive increase behavior, its convergence time is proportional to the bandwidth-delay product (BDP).

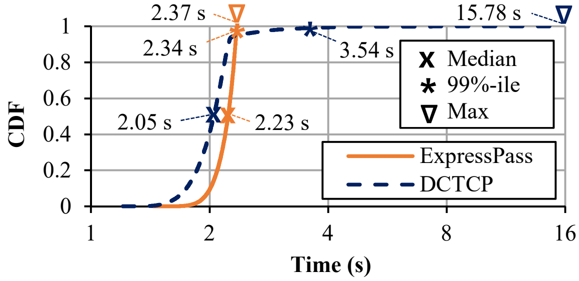


Figure 17: Shuffle workload (ns-2)

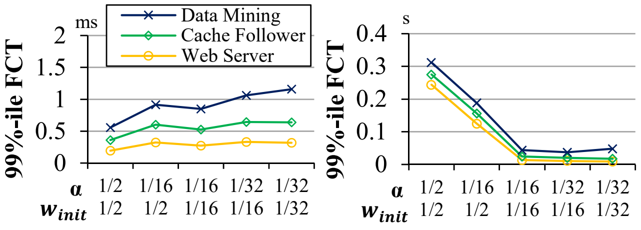
6.2 Heavy incast traffic pattern

One advantage of ExpressPass is robustness against incast traffic patterns. Such traffic patterns commonly happen in the shuffle step of MapReduce [20]. It creates an all-to-all traffic pattern, generating incast towards each host running a task. We simulate 40 hosts connected to single top-of-rack (ToR) switch via 10 Gbps links using ns-2. Each host runs 8 tasks, each of which sends 1 MB to all other tasks. Thus, each host sends and receives 2496 (39 × 8 × 8) flows. Figure 17 shows the CDF of flow completion times (FCTs) with DCTCP and ExpressPass. The median FCT of DCTCP is slightly better (2.0 vs. 2.2 *s*). However, DCTCP has a much longer tail. At 99^{th} percentile and tail, ExpressPass outperforms DCTCP by a factor of 1.51 and 6.65 respectively. With DCTCP, when some faster flows complete, the remaining flows often catch up. However, at the tail end, delayed flows tend to be toward a small set of hosts, such that they cannot simply catch up by using all available bandwidth. This drastically increases the tail latency and contributes to the straggler problem in MapReduce [7]. Our example demonstrates

[4] We set the DCTCP parameter K= 65, g= 0.0625 for 10 Gbps link, and K= 650, g= 0.01976 for 100 Gbps link. For ExpressPass, we report the average throughput for each RTT. For DCTCP, we averaged over 10 RTT due to its high variance.

		Data Mining [28]	Web Search [3]	Cache Follower [50]	Web Server [50]
0 - 10KB	(S)	78%	49%	50%	63%
10KB - 100KB	(M)	5%	3%	3%	18%
100KB - 1MB	(L)	8%	18%	18%	19%
1MB -	(XL)	9%	20%	29%	
Average flow size		7.41MB	1.6MB	701KB	64KB

Table 2: Flow size distribution of realistic workload



(a) 0 - 10KB (S) (b) 100KB - 1MB (L)

Figure 18: 99%-ile FCT of ExpressPass with different α and w_{init} (10 Gbps, load 0.6)

congestion control can contribute to the problem, and ExpressPass effectively alleviates this.

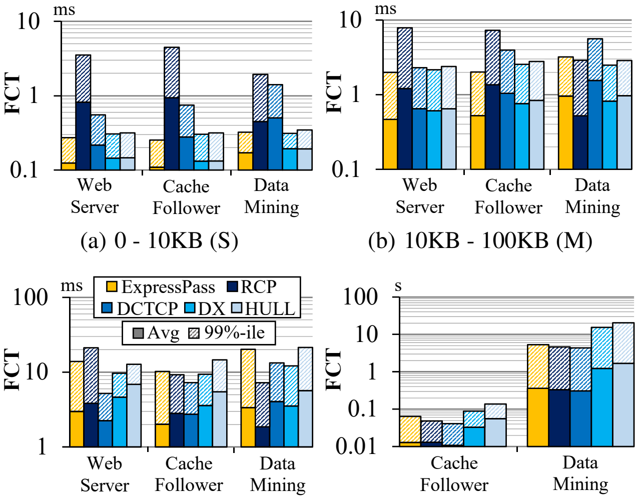
6.3 Performance under realistic workload

To evaluate the performance of ExpressPass in a more realistic scenario, we run ns-2 simulations with four different workloads shown in Table 2. It shows the flow size distribution and the average flow size for each workload. We have chosen the workloads to cover a wide range of average flow sizes ranging from 64 KB to 7.4 MB. While the data mining workload has a smaller fraction of XL size flows compared to web search, it has a larger cap of 1 GB compared to 30 MB for web search, resulting in higher average flow sizes. We generate 100 thousand flows with exponentially distributed inter-arrival time. We simulate three target loads of 0.2, 0.4, and 0.6. There is an over-subscription at the ToR uplinks and most of the traffic traverses through the ToR up-links due to random peer selection. Hence, we set the target load for ToR up-links.

We use a fat tree topology that consists of 8 core switches, 16 aggregator switches, 32 top-of-rack (ToR) switches, and 192 nodes. The topology has an over-subscription ratio of 3:1 at ToR switch layer. We create two networks, one with 10 Gbps links and the other with 40 Gbps links. Maximum queue capacities are set to 384.5 KB (250 MTUs) for the network with 10 Gbps link and 1.54 MB (1, 000 MTUs) for the network with 40 Gbps. All network link delays are set to 4 μs and host delays to 1 μs, which results in maximum RTT of 52 μs between nodes excluding queuing delay. To support multipath routing, Equal Cost Multi Path (ECMP) routing is used.

We measure the flow completion time (FCT) and queue occupancy of ExpressPass and compare them with RCP, DCTCP, DX, and HULL. We set the parameters as recommended in their corresponding papers.

Parameter sensitivity: The initial value of credit sending rate ($\alpha \times$ *max_rate*) and aggressiveness factor (w_{init}) determine the early behavior as described in Section 3.2. To decide appropriate values, we run realistic workloads at target load of 0.6 with different α and w_{init} values. Figure 18 (a) and (b) shows the 99%-ile FCT values for short (S) and large (L) flows respectively. As α and w_{init} decrease, 99%-ile FCT of large flows decreases at the cost of increased FCT for short flows. With $\alpha = w_{init} = 1/16$, large flow FCT decreases significantly, while short flow FCT increases less than 100% compared to using $\alpha = w_{init} = 1/2$. Further reducing the values provides an incremental gain in large flow FCT, but at a larger cost in short flow FCT. $\alpha = w_{init} = 1/16$ provides a sweet spot, and we use the setting in the rest of the experiments.



(a) 0 - 10KB (S) (b) 10KB - 100KB (M)

(c) 100KB - 1MB (L) (d) 1MB - (XL)

Figure 19: Average / 99%-ile flow completion time for realistic workload (10Gbps, load 0.6)

Flow Completion Time: We show the average and 99^{th} percentile FCTs across workloads for a target load of 0.6 in Figure 19. The solid bar at the bottom indicates the average FCT and the upper stripe bar shows the 99^{th} percentile value. One clear pattern is that ExpressPass performs better than others for short flows (S and M) across workloads, and DCTCP and RCP perform better on large flows (L and XL). ExpressPass achieves from 1.3x to 5.14x faster average FCT compared to DCTCP for S and M flows, and the gap is larger at 99^{th} percentile. For L and XL size flows, its speed ranges from 0.37x to 2.86x of DCTCP. This is expected given that two dominant factors for short flow completion time are low queuing and ramp up time which ExpressPass improves at the cost of lower utilization. Between workloads, ExpressPass performs the worst for Web Server workload relative to the others. This is due to the small average flow size of 64 KB causing more credit waste.

Credit Waste: To understand how much credit is wasted, we measure the ratio of credit waste from the sender. Figure 20 shows the result broken down by the workload and the link speed. As the average flow size becomes smaller, the wasted amount of credit increases up to 60% in 40 Gbps and 34% in 10 Gbps in the Web Server workload. Higher link speed also increases the wasted credits. This explains why ExpressPass performs worse than DCTCP for large flows in the Web Server workload. In general, the amount of wasted credit is proportional to the bandwidth delay product (BDP) and inversely proportional to the average flow size. In the worst case, the receiver may send an entire BDP worth of credits to the sender

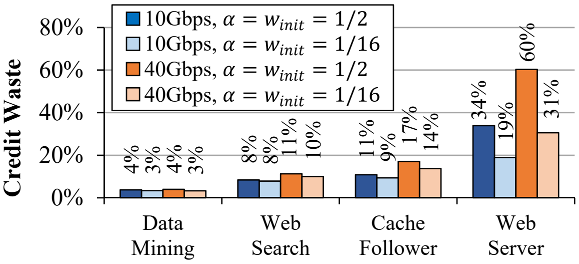


Figure 20: Credit waste ratio with target load of 0.6

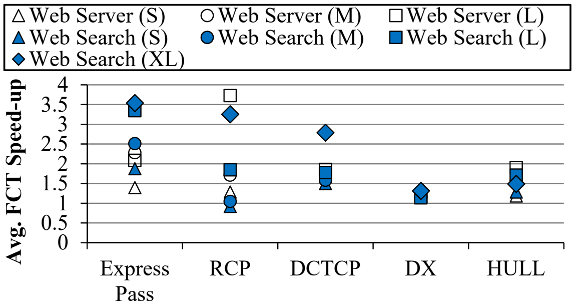


Figure 21: Average speed-up of 40G over 10G with websearch, webserver workload (load 0.6).

		Average Queue (KB)					Max Queue (KB)				
Traffic Type	**Load**	**Express Pass**	**RCP**	**DCTCP**	**DX**	**HULL**	**Express Pass**	**RCP**	**DCTCP**	**DX**	**HULL**
Web Server	0.2	0.14	2.34	0.46	0.56	0.62	30.76	375.3	86.63	15.38	15.46
	0.4	0.32	2.78	0.99	0.62	0.67	29.31	375.3	131.6	22.00	26.15
	0.6	0.41	5.46	2.79	0.77	0.84	20.75	375.3	157.1	42.09	77.63
Cache Follower	0.2	0.07	3.19	0.88	0.56	0.62	34.00	375.3	153.8	10.77	29.22
	0.4	0.27	10.54	2.85	0.65	0.71	41.78	375.3	178.3	23.45	23.24
	0.6	0.50	14.58	6.18	0.77	0.82	32.30	375.3	214.7	39.30	79.28
Web Search	0.2	0.08	3.27	1.84	0.57	0.63	40.16	375.3	253.8	12.40	38.45
	0.4	0.29	12.19	4.77	0.66	0.71	49.30	375.3	296.8	20.18	26.15
	0.6	0.54	18.01	8.65	0.79	0.84	30.93	375.3	343.9	37.76	74.05
Data Mining	0.2	0.06	2.31	2.58	0.61	0.58	38.45	375.3	386.0	9.32	49.22
	0.4	0.12	4.78	5.92	0.60	0.69	40.66	375.3	375.3	17.39	49.22
	0.6	0.47	5.78	9.33	0.64	0.67	46.14	375.3	357.0	19.99	53.83

Table 3: Average/maximum queue occupancy (ns-2) @ 10Gbps

and yet receive only one data packet. The figure also shows credit waste with two different parameter settings to highlight this tradeoff. Setting α to 1/16 reduces the amount of wasted credits significantly to 31% and 19% for 40 Gbps and 10 Gbps link speed respectively.

Link speed scalability: Higher link speed enables flows to push more bits per second but may require longer convergence time, which diminishes the benefit of more bandwidth. To evaluate how well ExpressPass performs, we measure the relative speed-up in FCT when the link speed increases from 10 Gbps to 40 Gbps. Figure 21 shows the average FCT speed-up for the Web Server and Web Search workloads. Data Mining and Cache Follower show similar results as Web Search. For small flows, we observe less speed-up compared to larger flows because RTT dominates the small flow FCT thus increased bandwidth helps less. ExpressPass shows the largest gain (1.5x - 3.5x) across all cases except large flows in Web Server workload. RCP has the largest gain in this case due to its aggressive ramp up. It also maintains high utilization, whereas ExpressPass suffers from increased credit waste when the BDP increases. DCTCP has a 2.8x gain for the XL size flows and less than 2x gains for S, M, and L size flows. DX and HULL benefit the least as they're the least aggressive scheme. Overall, this shows increasing benefit of ExpressPass's fast convergence and low queuing with the higher link speed.

Queue length: Table 3 shows the average and maximum queue occupancy observed during the simulation. On average, ExpressPass uses less buffer than other congestion controls. ExpressPass's max queue is not proportional to the load whereas all other transports use more queue with the increased load. ExpressPass's queue bound is a property of the topology, independent to the workload.

7 DISCUSSION AND LIMITATION

Path symmetry: To ensure path symmetry, we have used symmetric routing. Symmetric routing can be achieved as evidenced by prior work [27] and our simulation also uses symmetric routing on fat tree. However, it incurs increased complexity to maintain consistent link ordering with ECMP in the forward and reverse directions, especially for handling link failures. Packet spraying [22] is a viable alternative because it ensures all available paths get the equivalent load. We believe the bounded queuing property of ExpressPass will also limit the amount of packet reordering.

Presence of other traffic: In real datacenter networks, some traffic, such as ARP packets and link layer control messages, may not be able to send credits in advance. One solution to limit such traffic and apply "reactive" control to account for it. When traffic is sent without credit, we absorb them in the network queue and send credit packets from the receiver, which will drain the queue.

Multiple traffic classes: Datacenter networks typically classify traffic into multiple classes and apply prioritization to ensure the quality of service. Existing schemes use multiple queues for data packets and enforce priority or weighted fair-share across the queues. The same logic can be applied to ExpressPass for credit packets instead of data packets. For example, prioritizing flow A's credits over flow B's credits while throttling the sum of credits from A and B will result in the strict prioritization of A over B. Applying weighted fair-share over multiple credit queues would have a similar effect.

Limitation of our feedback algorithm: The credit-based design opens up a new design space for feedback control. We have explored a single instance in this space, but our feedback algorithm leaves much room for improvement.

Short flows cause credit packets to be wasted. This hurts the flow completion times of long flows that compete with many short flows. One way to overcome this is to use the approach of RC3 [42]. RC3 uses low priority data packet to quickly ramp up flows without affecting other traffic. Similarly, in ExpressPass, one can allow applications to send low priority data packets without credits. Such low priority traffic would then be transmitted opportunistically to compensate for the bandwidth lost due to wasted credits. However, this approach comes at the cost of rather complex loss recovery logic and requires careful design [42]. Credit waste can also be reduced if the end of the flow can be reliably estimated in advance. Currently, we assume senders do not know when the flow ends in advance. However, it is possible for the sender to notify the end of the flow in advance and send the credit stop request preemptively with some margin. Some designs [1] even propose advertising send buffer to the receiver. The sender can then leverage the information to control the amount of credit waste.

Another limitation is that our feedback currently assumes all hosts have the same link capacity. We leverage this assumption in our feedback design for faster convergence. However, when host link speeds are different, the algorithm does not achieve fairness. One could use other algorithms, such as CUBIC [29], to regain fairness by trading-off the convergence time under such a setting, without compromising bounded queuing.

8 RELATED WORK.

Credit-based flow control: Our approach is inspired by credit based flow control [36] used in on-chip networks and high-speed system interconnect such as Infiniband, PCIe, Intel QuickPath, or AMD Hypertransport [53]. However, traditional credit-based flow control is hop-by-hop, which requires switch support, and is difficult to scale to datacenter size. Decongestion control and pFabric pioneered a design where hosts transmit data aggressively and switches allocate bandwidth. The difference is that we allocate bandwidth using credit. Finally, TVA [57] uses a similar idea to rate-limit requests at the router, but it is designed for DDoS prevention considering the size of response rather than congestion control.

Low latency datacenter congestion control: DCQCN [58] and TIMELY [41] are designed for datacenters that have RDMA traffic. DCQCN uses ECN as congestion signal and QCN-like rate control. The main goals of DCQCN alleviate the problems caused by PFC by reducing its use while reducing ramp-up time. TIMELY uses delay as feedback, similar to DX [38], but incorporates PFC to achieve zero loss and lower the 99^{th} percentile latency. PERC [34] proposes a proactive approach to overcome the problems of reactive congestion control. We believe ExpressPass presents an alternative approach and shows promise in the high-speed environment (e.g., 100 Gbps networks).

Flow scheduling in datacenters: A large body of work focuses on flow scheduling [6, 11, 26] in datacenter networks to minimize flow completion times. Although the goals might overlap, flow scheduling is largely orthogonal to congestion control. We believe congestion control is a more fundamental mechanism for network resource allocation. We also note that some flow scheduling schemes [43] have been used in conjunction with congestion control to minimize the flow completion times. pFabric treats the network as a single switch and performs shortest job first scheduling assuming the flow size is known in advance. This requires switch modifications. PIAS [11] makes the approach more practical by approximating shortest-job-first by implementing a multi-level feedback queue using priority queuing in commodity switches and does not require knowledge of individual flow size. pHost [26] shares the idea of using credit (token) packets, but token packets in pHost are used for scheduling packets/flows rather than congestion control. It assumes a congestion-free network by using a network with full bisection bandwidth and packet spraying. In addition, pHost does require knowledge of individual flow size in advance.

Router-assisted congestion control: Some congestion control algorithms require in-network support [24, 30, 35, 42]. These mechanisms introduce a form of in-network feedback with which switches explicitly participate in rate allocation of each flow. To reach fast convergence, PERC [34] and FCP [30] employ mechanisms for end-hosts to signal their bandwidth demand to the switches in the network, which require changes in the switches. In ExpressPass, we use credit packets to signal demand and merely use rate-limiting, which does not require modification of the switches. Finally, RC3 [42] uses in-network priority queues to fill up the available bandwidth in one RTT. We believe this technique can be applied to credit packets to achieve similar benefits, but leave it as future work.

9 CONCLUSION

In this work, we introduce ExpressPass, an end-to-end credit-based congestion control. We use end-to-end credit transfer for bandwidth allocation and fine-grained packet scheduling. We explore the key benefits of a credit-based design and demonstrate it opens up a new design space for more efficient congestion control. In particular, the use of credit enables 1) low-cost bandwidth probing without queue build-up and 2) scheduling the arrival of data packets at packet granularity. We address key challenges in realizing a credit-based congestion control and demonstrate it can be implemented using commodity switches. By shaping the flow of credit packets at the switch, ExpressPass effectively controls congestion even before sending data packets. By achieving fast convergence, it drastically reduces the FCT for small flows. ExpressPass requires a small amount of buffer. Our evaluation shows that ExpressPass (1) outperforms other congestion control algorithms; (2) ensures high utilization and fairness even with many concurrent flows; and (3) the benefits of ExpressPass over other algorithms become even more favorable as link speeds increase.

ACKNOWLEDGEMENT

We thank our shepherd Changhoon Kim and anonymous reviewers for their valuable feedback. We also thank Wei Bai for sharing his expertise in switch configurations, Sangjin Han for his advice on using SoftNIC, Yibo Zhu for providing the DCQCN simulator, and Changhyun Lee for providing the DX simulator. We appreciate valuable feedback from David Wetherall, Nandita Dukkipati, Judson Wilson, Aurojit Panda, Seongmin Kim, and Jaehyeong Ha. This work was supported by IITP grants funded by the Korea government (MSIP) (No.R-20160222-002755 and No.2015-0-00164).

REFERENCES

[1] Alexandru Agache and Costin Raiciu. 2015. Oh Flow, Are Thou Happy? TCP Sendbuffer Advertising for Make Benefit of Clouds and Tenants. In *Proceedings of the 7th USENIX Conference on Hot Topics in Cloud Computing*.

[2] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. 2008. A scalable, commodity data center network architecture. In *ACM SIGCOMM*.

[3] Mohammad Alizadeh, Albert Greenberg, David A Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. 2010. Data center TCP (dctcp). In *ACM SIGCOMM*.

[4] Mohammad Alizadeh, Adel Javanmard, and Balaji Prabhakar. 2011. Analysis of DCTCP: stability, convergence, and fairness. In *Proceedings of the ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*.

[5] Mohammad Alizadeh, Abdul Kabbani, Tom Edsall, Balaji Prabhakar, Amin Vahdat, and Masato Yasuda. 2012. Less is more: trading a little bandwidth for ultra-low latency in the data center. In *USENIX Symposium on Networked Systems Design and Implementation*.

[6] Mohammad Alizadeh, Shuang Yang, Milad Sharif, Sachin Katti, Nick McKeown, Balaji Prabhakar, and Scott Shenker. 2013. pfabric: Minimal near-optimal datacenter transport. In *ACM SIGCOMM*.

[7] Ganesh Ananthanarayanan, Srikanth Kandula, Albert G Greenberg, Ion Stoica, Yi Lu, Bikas Saha, and Edward Harris. 2010. Reining in the Outliers in Map-Reduce Clusters using Mantri. In *USENIX OSDI*.

[8] Arista Networks. 2016. Architecting Low Latency Cloud Networks. https://www.arista.com/assets/data/pdf/CloudNetworkLatency.pdf. (2016). [Online; accessed Jan-2017].

[9] Arista Networks. 2016. Arista 7280R Series Data Center Switch Router Data Sheet. https://www.arista.com/assets/data/pdf/Datasheets/7280R-DataSheet.pdf. (2016). [Online; accessed Jan-2017].

[10] Arista Networks. 2017. 7050SX Series 10/40G Data Center Switches Data Sheet. https://www.arista.com/assets/data/pdf/Datasheets/7050SX-128_64_Datasheet.pdf. (2017). [Online; accessed Jan-2017].

[11] Wei Bai, Li Chen, Kai Chen, Dongsu Han, Chen Tian, and Hao Wang. 2015. Information-agnostic flow scheduling for commodity data centers. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*.

[12] Andreas Bechtolsheim, Lincoln Dale, Hugh Holbrook, and Ang Li. 2016. Why Big Data Needs Big Buffer Switches. https://www.arista.com/assets/data/pdf/Whitepapers/BigDataBigBuffers-WP.pdf. (2016). [Online; accessed Jan-2017].

[13] Theophilus Benson, Aditya Akella, and David A. Maltz. 2010. Network Traffic Characteristics of Data Centers in the Wild. In *Proc. 10th ACM SIGCOMM Conference on Internet Measurement*.

[14] Bob Briscoe and Koen De Schepper. 2015. Scaling tcp's congestion window for small round trip times. *Technical report TR-TUB8-2015-002, BT* (2015).

[15] Broadcom. 2012. Smart-Hash — Broadcom. https://docs.broadcom.com/docs/12358326. (2012). [Online; accessed Jan-2017].

[16] Jay Chen, Janardhan Iyengar, Lakshminarayanan Subramanian, and Bryan Ford. 2011. TCP Behavior in Sub Packet Regimes. In *Proc. ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems*. 2.

[17] Cisco. 2013. Nexus 7000 FabricPath. http://www.cisco.com/c/en/us/products/collateral/switches/nexus-7000-series-switches/white_paper_c11-687554.html. (2013). [Online; accessed Jan-2017; Section 7.2.1 Equal-Cost Multipath Forwarding].

[18] Chelsio Communications. 2013. Preliminary Ultra Low Latency Report. http://www.chelsio.com/wp-content/uploads/2013/10/Ultra-Low-Latency-Report.pdf. (2013). [Online; accessed Jan-2017].

[19] Sujal Das and Rochan Sankar. 2012. Broadcom Smart-Buffer Technology in Data Center Switches for Cost-Effective Performance Scaling of Cloud Applications. https://docs.broadcom.com/docs/12358325. (2012). [Online; accessed Jan-2017].

[20] Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: simplified data processing on large clusters. *Commun. ACM* 51, 1 (2008).

[21] Dell. 2015. Dell Networking Configuration Guide for the MXL 10/40GbE Switch I/O Module 9.9(0.0). http://topics-cdn.dell.com/pdf/force10-mxl-blade_Service%20Manual4_en-us.pdf. (2015). [Online; accessed Jan-2017. Enabling Deterministic ECMP Next Hop (pp.329)].

[22] Advait Dixit, Pawan Prakash, Y Charlie Hu, and Ramana Rao Kompella. 2013. On the impact of packet spraying in data center networks. In *INFOCOM, 2013 Proceedings IEEE*. IEEE.

[23] Nandita Dukkipati. 2008. *Rate Control Protocol (RCP): Congestion control to make flows complete quickly*. Stanford University.

[24] Nandita Dukkipati, Masayoshi Kobayashi, Rui Zhang-Shen, and Nick McKeown. 2005. Processor sharing flows in the internet. In *International Workshop on Quality of Service*.

[25] Michael Ferdman, Almutaz Adileh, Onur Kocberber, Stavros Volos, Mohammad Alisafaee, Djordje Jevdjic, Cansu Kaynak, Adrian Daniel Popescu, Anastasia Ailamaki, and Babak Falsafi. 2012. Clearing the Clouds: A Study of Emerging Scale-out Workloads on Modern Hardware. In *Proceedings of the Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS XVII)*. ACM, New York, NY, USA, 12. DOI: https://doi.org/10.1145/2150976.2150982

[26] Peter X Gao, Akshay Narayan, Gautam Kumar, Rachit Agarwal, Sylvia Ratnasamy, and Scott Shenker. 2015. pHost: Distributed near-optimal datacenter transport over commodity network fabric. In *ACM CoNEXT*.

[27] Rajib Ghosh and George Varghese. 2001. Modifying Shortest Path Routing Protocols to Create Symmetrical Routes. (2001). UCSD technical report CS2001-0685, September 2001.

[28] Albert Greenberg, James R Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A Maltz, Parveen Patel, and Sudipta Sengupta. 2009. VL2: a scalable and flexible data center network. In *ACM SIGCOMM*.

[29] Sangtae Ha, Injong Rhee, and Lisong Xu. 2008. CUBIC: a new TCP-friendly high-speed TCP variant. *ACM SIGOPS Operating Systems Review* 42, 5 (2008).

[30] Dongsu Han, Robert Grandl, Aditya Akella, and Srinivasan Seshan. 2013. FCP: A Flexible Transport Framework for Accommodating Diversity. In *ACM SIGCOMM*.

[31] Sangjin Han, Keon Jang, Aurojit Panda, Shoumik Palkar, Dongsu Han, and Sylvia Ratnasamy. 2015. SoftNIC: A software NIC to augment hardware. In *Technical Report UCB/EECS-2015-155*. EECS Department, University of California, Berkeley.

[32] Jiawei Huang, Yi Huang, Jianxin Wang, and Tian He. 2015. Packet slicing for highly concurrent TCPs in data center networks with COTS switches. In *IEEE ICNP*.

[33] Raj Jain, Dah-Ming Chiu, and William R Hawe. 1984. A quantitative measure of fairness and discrimination for resource allocation in shared computer system. (1984).

[34] Lavanya Jose, Lisa Yan, Mohammad Alizadeh, George Varghese, Nick McKeown, and Sachin Katti. 2015. High speed networks need proactive congestion control. In *Proceedings of the 14th ACM Workshop on Hot Topics in Networks*.

[35] Dina Katabi, Mark Handley, and Charlie Rohrs. 2002. Congestion control for high bandwidth-delay product networks. In *ACM SIGCOMM*.

[36] HT Kung, Trevor Blackwell, and Alan Chapman. 1994. Credit-based flow control for ATM networks: credit update protocol, adaptive credit allocation and statistical multiplexing. In *ACM SIGCOMM*.

[37] Jean-Yves Le Boudec and Patrick Thiran. 2001. *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*. Springer-Verlag, Berlin, Heidelberg.

[38] Changhyun Lee, Chunjong Park, Keon Jang, Sue Moon, and Dongsu Han. 2015. Accurate latency-based congestion feedback for datacenters. In *USENIX Annual Technical Conference*.

[39] Steven McCanne, Sally Floyd, Kevin Fall, Kannan Varadhan, and others. 1997. Network simulator ns-2. (1997).

[40] Microsoft. 2015. Azure support for Linux RDMA. https://azure.microsoft.com/en-us/updates/azure-support-for-linux-rdma. (2015). Online; accessed 12-July-2016.

[41] Radhika Mittal, Nandita Dukkipati, Emily Blem, Hassan Wassel, Monia Ghobadi, Amin Vahdat, Yaogong Wang, David Wetherall, David Zats, and others. 2015. TIMELY: RTT-based Congestion Control for the Datacenter. In *ACM SIGCOMM*.

[42] Radhika Mittal, Justine Sherry, Sylvia Ratnasamy, and Scott Shenker. 2014. Recursively Cautious Congestion Control. In *USENIX Conference on Networked Systems Design and Implementation*.

[43] Ali Munir, Ghufran Baig, Syed M Irteza, Ihsan A Qazi, Alex X Liu, and Fahad R Dogar. 2014. Friends, not foes: synthesizing existing transport strategies for data center networks. In *ACM SIGCOMM*.

[44] Kanthi Nagaraj, Dinesh Bharadia, Hongzi Mao, Sandeep Chinchali, Mohammad Alizadeh, and Sachin Katti. 2016. NUMFabric: Fast and Flexible Bandwidth Allocation in Datacenters. In *ACM SIGCOMM*. 14.

[45] Juniper Networks. 2016. Configuring PIC-Level Symmetrical Hashing for Load Balancing on 802.3ad LAGs for MX Series Routers. https://www.juniper.net/techpubs/en_US/junos15.1/topics/task/configuration/802-3ad-lags-load-balancing-symmetric-hashing-mx-series-pic-level-configuring.html. (2016). [Online; accessed Jan-2017].

[46] Jitendra Padhye, Victor Firoiu, Don Towsley, and Jim Kurose. 1998. Modeling TCP throughput: A simple model and its empirical validation. *ACM SIGCOMM Computer Communication Review* 28, 4 (1998).

[47] Jonathan Perry, Amy Ousterhout, Hari Balakrishnan, Devavrat Shah, and Hans Fugal. 2014. Fastpass: A centralized zero-queue datacenter network. In *ACM SIGCOMM*.

[48] Sivasankar Radhakrishnan, Yilong Geng, Vimalkumar Jeyakumar, Abdul Kabbani, George Porter, and Amin Vahdat. 2014. SENIC: Scalable NIC for End-Host Rate Limiting.. In *NSDI*, Vol. 14.

[49] Sivasankar Radhakrishnan, Vimalkumar Jeyakumar, Abdul Kabbani, George Porter, and Amin Vahdat. 2013. NicPic: Scalable and Accurate End-Host Rate Limiting. In *USENIX HotCloud*.

[50] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, George Porter, and Alex C Snoeren. 2015. Inside the social network's (datacenter) network. In *ACM SIGCOMM Computer Communication Review*, Vol. 45. ACM.

[51] M. Schlansker, J. Tourrilhes, and Y. Turner. 2015. Method for routing data packets in a fat tree network. (April 14 2015). https://www.google.com/patents/US9007895 US Patent 9,007,895.

[52] Arjun Singh, Joon Ong, Amit Agarwal, Glen Anderson, Ashby Armistead, Roy Bannon, Seb Boving, Gaurav Desai, Bob Felderman, Paulie Germano, and others. 2015. Jupiter rising: A decade of clos topologies and centralized control in google's datacenter network. In *ACM SIGCOMM*.

[53] David Slogsnat, Alexander Giese, and Ulrich Brüning. 2007. A Versatile, Low Latency HyperTransport Core. In *ACM/SIGDA International Symposium on Field Programmable Gate Arrays*. 8. DOI: https://doi.org/10.1145/1216919.1216926

[54] Jim Warner. 2014. Packet Buffer. https://people.ucsc.edu/~warner/buffer.html. (2014). [Online; accessed Jan-2017].

[55] H. Wu, Z. Feng, C. Guo, and Y. Zhang. 2013. ICTCP: Incast Congestion Control for TCP in Data-Center Networks. *IEEE/ACM Transactions on Networking* 21, 2 (2013).

[56] Lisong Xu, Khaled Harfoush, and Injong Rhee. 2004. Binary increase congestion control (BIC) for fast long-distance networks. In *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, Vol. 4. IEEE.

[57] Xiaowei Yang, David Wetherall, and Thomas Anderson. 2005. A DoS-limiting Network Architecture. In *ACM SIGCOMM*.

[58] Yibo Zhu, Haggai Eran, Daniel Firestone, Chuanxiong Guo, Marina Lipshteyn, Yehonatan Liron, Jitendra Padhye, Shachar Raindel, Mohamad Haj Yahia, and Ming Zhang. 2015. Congestion control for large-scale RDMA deployments. In *ACM SIGCOMM*.

[59] Yibo Zhu, Monia Ghobadi, Vishal Misra, and Jitendra Padhye. 2016. ECN or Delay: Lessons Learnt from Analysis of DCQCN and TIMELY. In *ACM CoNEXT*.