FlexPass: A Case for Flexible Credit-based Transport for Datacenter Networks

Hwijoon Lim[1] Jaehong Kim[1] Inho Cho[2] Keon Jang[3] Wei Bai[4] Dongsu Han[1]
[1]KAIST [2]MIT [3]Rubrik [4]Microsoft Research

Abstract

Proactive transports explicitly allocate bandwidth to each sender with credits which schedule packet transmission. While promising, existing proactive solutions share a stringent deployment requirement; they assume the perfect control of every link and packet in the network. However, the assumption breaks in practice because new transports are usually deployed gradually over time and legacy traffic always coexists. In this paper, we present FlexPass, a credit-based transport that takes deployment flexibility as a first-class citizen. FlexPass uses a novel combination of network and end-host designs to solve the problem of co-existence and gradual deployment. FlexPass leverages a proactive control loop to send credit-scheduled packets and a complementary reactive control loop to send unscheduled packets to utilize the spare bandwidth. Finally, FlexPass prevents queue build-ups of both scheduled and unscheduled packets, and recovers lost packets efficiently. Our evaluation on the testbed shows that FlexPass maintains co-existence with legacy transports (DCTCP), while preserving the high-performance properties of the proactive transport. In large-scale simulations, we show that FlexPass delivers the best incremental benefits during the gradual deployment. We find traffic upgraded to FlexPass benefits from the bounded queue and reduced flow completion time by up to 44% compared to the legacy traffic, while minimizing the side-effect on the legacy flows.

CCS Concepts: • **Networks** → **Transport protocols**.

Keywords: datacenter networking, congestion control

ACM Reference Format:
Hwijoon Lim, Jaehong Kim, Inho Cho, Keon Jang, Wei Bai, Dongsu Han. 2023. FlexPass: A Case for Flexible Credit-based Transport for Datacenter Networks. In *Eighteenth European Conference on Computer Systems (EuroSys '23), May 8–12, 2023, Rome, Italy*. ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3552326.3587453

1 Introduction

Low latency has become a virtue of modern datacenter networks (DCNs). To accommodate the need for low latency, many efforts have been made on improving datacenter transports over the past decade. As the link speed in datacenters scales up to 100 Gbps and beyond, flows become "shorter" as they can be finished in fewer round-trip times (RTTs).

Traditional congestion control algorithms, such as ECN-based [1, 48] or delay-based [26, 32], are insufficient in such a trend due to their reactive nature. Therefore, many recent proposals [9, 13, 18, 35, 36] shift to proactive congestion control. Instead of iteratively probing the bandwidth, proactive transports proactively allocate network bandwidth to each sender as credits so that senders can send scheduled packets at the optimal rate to achieve high throughput, low queuing delay, and near-zero packet loss. We refer to these transports as "credit-based" transports[1].

However, despite the outstanding performance, we find that all the existing proactive transport solutions assume a *perfect* single administrative domain where the transport has full control of *every* packet and network element. This is because proactive transports require the *complete* knowledge of the network topology, routing, and per-link bandwidth to allocate credits to schedule packet transmissions. However, such stringent assumption hardly holds in practice, thus significantly blocking the production adoption of proactive transports at scale. Rolling out a new transport at scale is often a gradual process [15] in which operators start from a pilot deployment, run various benchmark workloads, and increase the deployment scale. During the gradual deployment, legacy flows co-exist with proactive flows, resulting in a heterogeneous environment. Even after the full deployment, such heterogeneity still remains due to north-south traffic (e.g., 18% of total traffic [41]) crossing the boundary of datacenters, driven by Internet-facing applications and geo-distributed workloads [38]. Such uncontrolled traffic can easily break the delicate credit allocation and clip the desirable properties of proactive transports (§2.2).

When a legacy transport shares the network with a proactive transport, they interfere with each other, causing various problems such as link under-utilization or high latency. Reactive transports will grab some bandwidth, leaving the available bandwidth for the proactive transport as a variable that changes over time. This runs into a dilemma. On the one hand, if the proactive transport allocates credits using the full

[1]Proactive and credit-based transport are used interchangeably in this paper.

link capacity, it will cause persistent link over-subscription and even starve competing reactive flows. On the other hand, if we use weighted queue isolation and conservatively allocate credits using the minimum guaranteed rate, the proactive transport will suffer from link under-utilization.

Motivated by the problem, we present FlexPass, a new credit-based proactive transport that *takes deployment flexibility as the first-class citizen.* By deployment flexibility, we mean that the solution should be able to preserve the performance benefits of proactive transports when co-existing with legacy traffic, which is the common case in the production environment. The key challenge for FlexPass is achieving high throughput and low latency under *dynamic* network bandwidth due to the existence of *unpredictable* legacy flows. To this end, FlexPass uses weighted fair queueing at the switch to reserve static bandwidth for credit allocation, without starving legacy transports. To achieve high throughput with partial bandwidth knowledge, FlexPass leverages two control loops to send packets in parallel: a credit-based proactive control loop to send scheduled packets based on the minimum guaranteed bandwidth, and a complementary reactive control loop to send unscheduled packets to utilize the spare bandwidth left by legacy traffic. On the credit-based control loop, FlexPass adopts ExpressPass [9] to allocate credits as ExpressPass can mitigate congestion in the network core, which is common in production networks due to oversubscription. On the reactive control loop, FlexPass uses an ECN-based algorithm to probe the spare bandwidth. To preserve low latency, FlexPass uses selective dropping at the switch to prioritize scheduled packets over unscheduled ones in case of queue build-ups.

We evaluate FlexPass using small-scale experiments on a ten-node testbed and large-scale simulations. We implement FlexPass on the host-side virtual NIC built with BESS [17] for testbed experiments, and in ns-2 [31] for simulations. Our evaluation shows FlexPass achieves similar performance as an ideal but impractical weighted configuration while significantly outperforming the naïve deployment scheme. For example, FlexPass does not starve legacy flows, achieving 99.9% lower starvation time, defined as a duration of each transport's bandwidth is less than 20%, compared to the naïve deployment scheme. Also, during the gradual deployment, FlexPass achieves 2.4% to 16% lower average FCT and 18% to 33% lower 99th percentile FCT for small flows compared to the naïve deployment.

Contribution. We systematically study the co-existence problem between credit-based proactive and reactive transports, which is one of the major hurdles for deploying proactive transports in production datacenters. Motivated by the above problem, we propose FlexPass, a new credit-based proactive transport that takes deployment flexibility as the first-class citizen. We show that a novel combination of existing techniques solves the problem.

2 Background and Motivation

2.1 Background

Reactive congestion control. Traditional congestion control algorithms [1, 5, 26, 28, 32, 37, 48] react to congestion signals (e.g., packet loss, delay, ECN) "after the fact" and iteratively adjust the sending rate. However, with the rapid increase of DCN bandwidth, flows become "smaller", leaving little time to reach convergence. Furthermore, congestion events in production DCNs are often short-lived [47]. These make reactive congestion control algorithms ill-suited to meet the low latency requirements in high-speed DCNs.

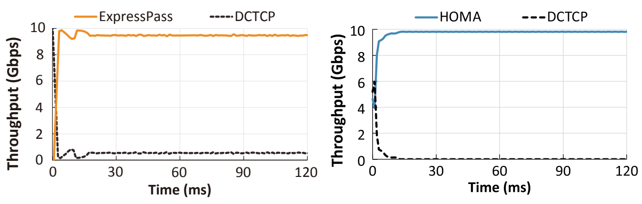
Nonetheless, reactive congestion control still dominates traffic in production DCNs [25, 26] due to their deployability. Reactive congestion control does not make strong assumptions to the underlying network, e.g., the prior knowledge of the topology (routing and per-link bandwidth). In production DCNs, it is common to see that multiple reactive solutions are enabled simultaneously for different workloads, e.g., DCTCP for intra-DC TCP traffic, DCQCN for RDMA traffic, and BBR/Cubic [7, 16] for wide-area traffic. To support co-existence, queue isolation is often used [25, 26].

Proactive congestion control. Many recent DCN transport proposals [6, 9, 13, 18, 35, 36] adopt proactive congestion control algorithms. Proactive congestion control requires the prior knowledge of the *entire* network and proactively allocates bandwidth to *all* the senders as credits so that each sender transmits "scheduled packets" at the right rate to ensure high throughput, low latency, and fast convergence. Different approaches allocate credits in different ways. For example, FastPass [36] uses a centralized arbiter while NDP [18], Homa [35], pHost [13], and ExpressPass [9] leverage receiver-driven credit allocation. NDP, Homa and pHost assume the network core is free of congestion and only mitigate the congestion at the edge[2]. ExpressPass does not rely on this assumption and does not require switch hardware modification. Hence, in this paper, we incorporate the main idea of ExpressPass into our design.

Deploying new transports in production. In production DCNs, gradual deployment is a common practice to enable new transports. Operators typically start by rolling out the new transport in a pilot deployment, run various benchmarks to evaluate performance benefits, and then increase the deployment scale. For example, Guo et al. took a step-by-step procedure to roll out RDMA from rack level, to podset level, and eventually to the entire datacenter [15].

However, gradual deployment results in a heterogeneous environment where the new transport and legacy reactive transports co-exist. Even after the full deployment in a datacenter, such heterogeneity still exists due to north-south

[2]Given the low average network utilization, production datacenters intentionally adopt oversubscribed topologies to reduce cost, thus leaving congestion in the network core. For example, Singh et al. [43] report that around 37.2% of packet drops happen in the network core.



(a) ExpressPass vs. DCTCP (b) HOMA vs. DCTCP

Figure 1. [Simulation] Aggregate throughput of DCTCP and ExpressPass / HOMA flows competing for a 10 Gbps link.

traffic crossing the boundary of datacenters. For instance, Roy et al. showed that around 1/6 of Facebook's datacenter traffic was north-south traffic (Table 3 of [41]).

2.2 Key Challenges in Co-existence

The superior performance of proactive transports comes from stringent deployment assumptions: the prior knowledge of the *entire* network (e.g., topology, link capacity, and routing) and the capability to control the transmission of *every* packet. However, these assumptions do not hold in production DCNs where new transport and legacy transports always co-exist. Proactive transports are generally *incompatible* with such a heterogeneous environment as legacy traffic will break the delicate credit allocation, thus clipping the desired properties of proactive transports. On the other hand, the performance of legacy traffic is also degraded. This is because proactive transports may not effectively detect or react to congestion when competing the link capacity with legacy traffic. Thus, legacy reactive flows will be *starved* by proactive flows and significantly back off, thus suffering from large completion times.

To evaluate the impact of proactive transport flows on legacy reactive flows, we conduct a simple simulation using a dumbbell topology with a 10 Gbps bottleneck. Figure 1 (a) shows that the starvation occurs when a DCTCP [1] flow compete with a ExpressPass [9] flow on the bottleneck link. Figure 1 (b) shows that the starvation also occurs with 16 HOMA [1] and 16 DCTCP flows [3]. Despite mechanism differences, both ExpressPass and HOMA allocate credits based on the full link capacity without awareness of the co-existing reactive flows. In contrast, DCTCP cuts the window upon detecting queue build-ups and ends up using only 5% of the link capacity. We note that the starvation of reactive flows is also common with other combinations of proactive and reactive transports.

We outline existing approaches to congestion control co-existence and discuss their shortcomings in supporting the incremental deployment of proactive congestion control.

[3]HOMA uses 8 strict priority queues at the switch. We map DCTCP flows to the highest priority queue. As HOMA maintains at most RTTbytes of in-flight data per flow, a single HOMA flow may not starve a DCTCP flow. However, multiple HOMA flows can easily starve DCTCP flows.

Weighted Fair Queueing is the most widely used approach to enable the co-existence of multiple transports [3, 25]. Operators map traffic to different switch queues by tagging packets with different Differentiated Services Code Point (DSCP) values. At the switch, operators use Deficit Weighted Round Robin (DWRR) [42] as the scheduling algorithm to avoid starvation. Operators also carefully tune parameters of dynamic buffer management [10] to ensure that a single transport will only use a proper fraction of the switch buffer.

Although this approach seems promising, it cannot effectively multiplex proactive and reactive transports. This is because proactive transports require prior knowledge of the available bandwidth of each link. When there is only a single proactive transport, the available bandwidth is exactly the link capacity. When a proactive transport and a reactive transport are scheduled by DWRR, we can only know the *minimum guaranteed bandwidth* for the proactive transport. The actual available bandwidth varies across time as it depends on both the DWRR setting and the demand of the reactive traffic. On the one hand, if the proactive transport uses the minimum guaranteed bandwidth to allocate credits, it will cause link under-utilization when there is not enough reactive traffic. On the other hand, if the proactive transport uses the full link capacity to allocate credits, it will cause congestion and a series of performance problems when reactive traffic exists.

The reader may wonder about the feasibility of adjusting queue weights based on the fraction of proactive traffic. As datacenter traffic is highly volatile, this approach requires network operators to monitor per-link traffic and update per-port settings in a real-time manner. In addition to queue weights, operators also need to adjust the per-link credit allocation rate accordingly. Such real-time monitoring and distributed update at datacenter scale are very expensive and hence unlikely to be adopted in production DCNs.

Strict Priority Queueing. We note that giving a strictly higher priority to the proactive transport is not an option because it deprioritizes all the legacy traffic regardless of their importance. Many legacy flows crossing deployment boundaries (rack/cluster/datacenter) are of significant importance for applications, e.g., a search request from the Internet. If we blindly deprioritize all the legacy traffic, we are likely to degrade the end-to-end application performance.

Network separation. Another more *aggressive* isolation approach is using physically separated networks to isolate different transports, e.g., one for new proactive transport and the other one for legacy traffic. Specifically speaking, the host can have multiple NIC ports with each port connected to a physically separated network. Each network has its own switches, links, and routing protocols [49]. While this approach can completely eliminate the interference of different transports, it significantly increases the economic (e.g., hardware purchase and cooling) and operation costs, which are top priorities for DC operators and cloud providers. Hence,

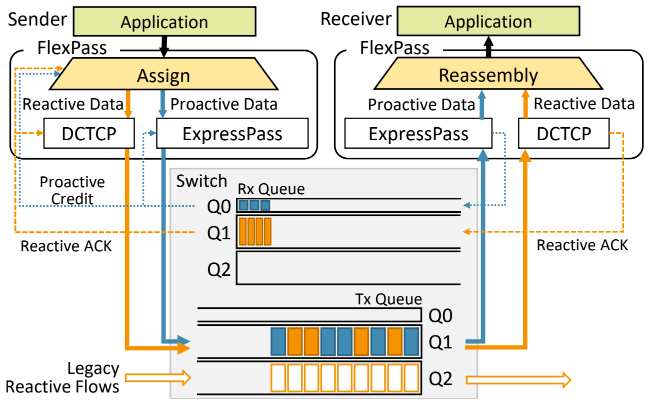


Figure 2. FlexPass Overview

operators are very unlikely to take this option to deploy proactive transports at a large scale.

3 Goal and Approach

3.1 Goals

Design goal. Our goal is to design a practical proactive transport, named FlexPass, with the following three key proprieties:

- FlexPass must preserve the high-performance properties of proactive congestion control, e.g., high throughput, bounded queue, and zero timeouts.
- FlexPass flows must not induce harm on legacy transports. For example, it should not cause starvation or interfere with the congestion signal of legacy traffic.
- FlexPass must provide gradual deployability with incremental benefit; partial deployment of a new transport should introduce a corresponding performance gain.

Non-goals. FlexPass is not a generic building block like Aeolus [20] and TLT [29] to augment different transports. In addition, our goal is to provide reasonable performance isolation instead of perfect per-flow fairness among transports.

3.2 Design Decisions

Queue isolation is unavoidable. Proactive congestion control requires accurate prior knowledge of per-hop bandwidth information. However, this requirement completely breaks when reactive flows and proactive flows co-exist in the same queue. Thus, to prevent interference we use weighted fair queuing (WFQ) to guarantee the minimum bandwidth for each transport. This ensures that 1) legacy flows will not be starved and 2) proactive transports at least know the minimum guaranteed bandwidth and can use it to allocate credits, which ensures that the scheduled data packets are delivered by the network without any loss. However, this approach also causes link under-utilization as proactive flows cannot use spare bandwidth left over by reactive flows.

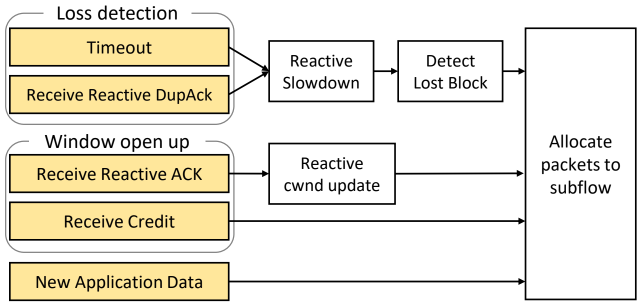


Figure 3. FlexPass assigns a segment to its sub-flow on *credit* reception, reactive loss detection, and new app data.

Split FlexPass flows into proactive and reactive sub-flows. To address link under-utilization, we introduce a reactive component in FlexPass that probes spare bandwidth in an opportunistic manner. A FlexPass flow is split into a proactive sub-flow and a reactive sub-flow. The reactive sub-flow shares the same queue with the proactive sub-flow, instead of directly competing with legacy traffic which is in a different queue.

The reader may think that the co-existence of reactive sub-flows and the proactive sub-flows is essentially equivalent to the scenario where proactive traffic and reactive traffic share the same switch queue. However, the key difference is that FlexPass controls both reactive and proactive sub-flows. This enables us to cooperatively schedule packet transmissions of two sub-flows to achieve bounded delay and zero timeouts.

Scheduling sub-flows for low latency. The two sub-flows exhibit distinct properties. The proactive sub-flow delivers predictable low latency because it schedules packet transmissions based on the minimum guaranteed bandwidth, while the reactive one can achieve high throughput. The question then is how we utilize the two control loops to deliver both bounded queuing delay and high throughput. The reactive sub-flow essentially carries "unscheduled" packets whose deliveries are not guaranteed. Hence, loss recovery must be carefully designed to minimize the end-to-end flow completion time. In Section 4, we show how FlexPass solves these problems.

4 FlexPass Design

Figure 2 presents an overview of FlexPass design with three main components:

Sender. FlexPass uses ExpressPass [9] and DCTCP [1] to control packet transmissions of the proactive sub-flow and reactive sub-flow, respectively. We choose ExpressPass as it can mitigate congestion in both network edge and core using commodity switch hardware. In contrast, other state-of-the-art schemes such as NDP and HOMA require modification of switch hardware or cannot mitigate the congestion at the network core. We choose DCTCP as it is well adopted [25, 43] in production DCNs. The two congestion control algorithms run independently without sharing any states, but FlexPass coherently manages packet transmissions and loss recovery across the two.

For each flow, a FlexPass sender maintains a single FIFO queue as its send buffer. When a credit packet of the proactive sub-flow arrives, the proactive sub-flow will pull a data packet from the send buffer to transmit. Reactive sub-flows are ACK-clocked and employ cumulative ACK with selective acknowledgment. If an arriving ACK opens up the window, the reactive sub-flow will transmit a data packet. We present more details on co-scheduling the two sub-flows in §4.2.

Switch. FlexPass requires three queues per egress port to isolate traffic: Q0 for credit packets, Q1 for FlexPass data packets (both proactive and reactive sub-flows), and Q2 for legacy traffic[4].

As required by ExpressPass, we give Q0 a strict high priority to ensure that credit packets will not be delayed due to data packets in Q1 and Q2. We also configure rate-limiting with a very small buffer (<1 KB) at Q0 to drop excessive credits. Q1 and Q2 are scheduled using Deficit Weighted Round Robin (DWRR) to avoid starvation. As we adopt DCTCP for reactive sub-flows, we enable RED/ECN marking on Q1. We introduce more details in §4.1.

Receiver. At the receiver side, FlexPass reassembles messages using data packets from proactive and reactive sub-flows, and delivers them to the application. The receiver also supports per-packet ACK for both proactive and reactive sub-flows.

4.1 Enabling Co-existence in the Network

As a deployment-friendly transport protocol, FlexPass should be able to preserve the performance benefits of proactive congestion control when co-existing with legacy traffic. There are two co-existence problems with distinct requirements:

- Co-existence between FlexPass and legacy traffic: Legacy traffic can be first-party workloads running different transports and third-party workloads, e.g., Internet traffic, and traffic between tenants' VMs. As we lack full control, we aim to enforce weighted fair sharing between legacy and FlexPass traffic without causing starvation.
- Co-existence between proactive and reactive sub-flows of FlexPass: Unlike the legacy traffic, FlexPass has control over its own reactive sub-flow, which should only use spare bandwidth left by legacy traffic. When there is a sufficient amount of legacy traffic, reactive sub-flows must not occupy any bandwidth. In addition, in any situation, we must bound the buffer occupancy of the reactive sub-flows to preserve low latency benefits.

Co-existence with legacy traffic. At the switch, FlexPass leverages three queues to isolate FlexPass and legacy traffic. We assign each traffic to the designated queues below:

- Q0: proactive credit packets.

[4]Operators may use multiple queues to further isolate different types of legacy traffic. Without loss of generality, we assume operators only use a single queue for legacy traffic in this section.

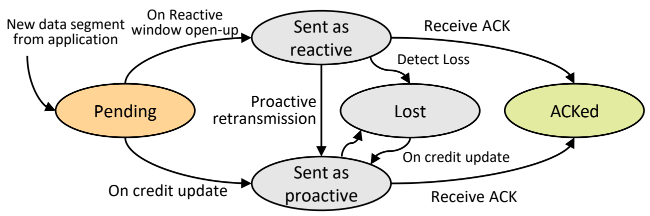


Figure 4. Per-packet state machine at the sender.

- Q1: FlexPass data packets (both proactive and reactive).
- Q2: legacy reactive traffic.

We set the queue weight for Q1 and Q2 to w_q ($0 < w_q < 1$) and $1-w_q$, respectively. w_q denotes the portion of bandwidth we would like to reserve for FlexPass in the network. We configure Q0 to high priority over the others and enforce rate limiting of credit packets. The original ExpressPass [9] sets the credit rate limit to the point where data packets fully utilize the link capacity. *However, in the case of the FlexPass, proactive sub-flows should only take the minimum guaranteed bandwidth to allocated credits.* Therefore, FlexPass sets the credit rate limit scaled down to w_q, so that the proactive sub-flow can take up to w_q of the line rate. Note that FlexPass does not require w_q to match with the actual proportion of FlexPass traffic in the network. Unlike other proactive transports, the performance of FlexPass is insensitive to the value of w_q (refer to Appendix A).

Co-existence between two sub-flows. When proactive and reactive sub-flows co-exist, the reactive sub-flow should only use spare bandwidth without causing queue buildups. A straightforward solution is using two queues to further isolate proactive and reactive sub-flows and giving the proactive queue a higher priority. This approach can achieve desired bandwidth sharing, but can cause serious out-of-order arrivals, thus bringing many challenges to loss recovery.

Realizing this limitation, we seek an approach that achieves desired bandwidth sharing and queue bound inside a switch queue. We notice that once the reactive sub-flow uses excessive bandwidth, there will be switch queue buildups. Therefore, as long as we bound the buffer usage of reactive packets, we can ensure that the reactive sub-flow only uses the spare bandwidth. To this end, FlexPass uses a combination of ECN marking and selective dropping.

As shown in many ECN-based congestion control schemes [1, 48], reactive sub-flows can react to the congestion before a loss happens, thus achieving lower average queue occupancy. However, ECN marking itself is not sufficient, as it does not guarantee a queue bound, and cannot handle bursty flows, i.e. incast. We need a more aggressive approach to throttle reactive traffic in case of many concurrent flows.

To this end, we further introduce selective dropping [20, 29] onto the FlexPass queue (Q1) to limit the queue buildup of reactive sub-flow. When the queue length of the *reactive sub-flow* exceeds a certain threshold, the switch drops incoming packets from reactive sub-flows. Packets from proactive sub-flows are still not dropped until the entire queue length exceeds the buffer limit. This mechanism effectively limits the queue buildup of reactive sub-flows. Selective dropping can be implemented using color-aware dropping [20, 29], a feature supported by a wide range of commodity switches.

Selective dropping brings two key benefits. First, it can quickly throttle the reactive sub-flow when no bandwidth is available for the reactive sub-flow. Unlike other mechanisms (e.g. ECN [1]) which require at least 1-RTT for congestion avoidance, selective dropping proactively drops the packet inside the switch fabric and thus results in quicker throttling of reactive flows. Second, selective dropping ensures bounded delay. As proactive sub-flows using ExpressPass already have a low queue bound, the entire queue length of the FlexPass queue (Q1) is eventually bounded.

To enable FlexPass at a switch, the operator only needs to allocate two additional queues (Q0 and Q1), configure queues (priority, rate limiting, buffer), and map traffic to them based on DSCP. These configurations can be applied incrementally through switch OS commands without rebooting the switch or reloading switch configurations. Hence, legacy traffic remains unimpacted during the switch configuration.

Example. When both FlexPass and legacy traffic with enough demands co-exist in the same link, proactive sub-flow of FlexPass gets $w_q \times \texttt{LinkRate}$, and legacy traffic gets the rest – $(1-w_q)\times\texttt{LinkRate}$, ensuring the co-existence between FlexPass and legacy traffic. Note reactive sub-flow of FlexPass gets no bandwidth. In contrast, when there is only FlexPass traffic in the link, the proactive sub-flow of FlexPass gets $w_q \times \texttt{LinkRate}$, and the reactive sub-flow gets the rest.

4.2 Sub-flow Scheduling at the Host

Given the bandwidth allocation between proactive and reactive sub-flows mentioned in §4.1, FlexPass must assign data segments to the two sub-flows correspondingly, recover lost packets, and ensure in-order delivery at the receiver side.

State machine. A FlexPass sender keeps a per-packet state machine to track whether a packet has been transmitted or delivered. Figure 4 shows the state machine with five states:

- **Pending:** The data packet is in the pending state if it has never been transmitted.
- **Sent as reactive/proactive:** When a packet is sent either as reactive or proactive, it is marked "sent as reactive (or proactive)".
- **Lost:** When the sender detects a packet loss, the lost packet is marked "Lost".
- **ACKed:** Upon ACK arrival, corresponding data packets are marked "ACKed" and removed from the send buffer.

Scheduling transmission on two sub-flows. When a sub-flow is available for further packet transmissions, the FlexPass sender assigns data segments to the sub-flow and starts transmissions. This happens when (1) the sender receives a credit packet of the proactive sub-flow, and (2) the window of the reactive sub-flow opens up. This is similar to the shared send buffer design of MPTCP [40], where the scheduling on each sub-flow is done on transmission time rather than in advance. Note that, reactive sub-flows can immediately transmit data during the first RTT, whereas proactive sub-flows need to wait for one RTT for credits [20].

When credit and ACK packets arrive at the sender, FlexPass makes state transitions and transmits packets as follows:

- **Receive a credit packet:** The sender can transmit one data packet using the proactive sub-flow. FlexPass can either transmit a new packet (`"Pending"`) or retransmit a packet (`"Lost"`, `"Sent as reactive"`). To accelerate loss recovery, FlexPass gives the highest transmission priority to the `"Lost"` packets. Next, to reduce spurious retransmissions, FlexPass prioritizes `"Pending"` packets over `"Sent as reactive"` packets. Note that FlexPass retransmits unacked reactive packets (`"Sent as reactive"`) for efficient loss recovery. We will explain the rationale at the end of this subsection when we describe the loss recovery.
- **Receive an ACK of the reactive sub-flow:** The sender marks corresponding data packets as either `"ACKed"` or `"Lost"`, depending on whether the ACK indicates a loss (`"Sent as reactive"` $\rightarrow$ `"ACKed"`, `"Sent as reactive"` $\rightarrow$ `"Lost"`), and slides the window of the reactive sub-flow. In addition, if the ACK opens up the window, it will trigger further transmissions of new packets via reactive sub-flow (`"pending"` $\rightarrow$ `"Sent as reactive"`). Unlike the proactive sub-flow, the reactive sub-flow is not used for retransmissions.
- **Receive an ACK of the proactive sub-flow:** The sender marks the corresponding data packet as ACKed. (`"Sent as proactive"` $\rightarrow$ `"ACKed"`).

Each FlexPass data packet carries two sequence numbers as in MPTCP [40]. One indicates the sequence within the entire FlexPass flow for reassembly, and the other is used within each sub-flow for congestion control and loss detection. Note, each flow and sub-flow has its own sequence number space.

Loss recovery. FlexPass employs ACK-based loss detection. Each sub-flow leverages the per-sub-flow sequence number to detect losses individually. When the receiver receives a packet, it checks the per-sub-flow sequence number and sends a cumulative ACK (with SACK-enabled) back to the sender. Unlike data packets, the ACK packet only carries the per-sub-flow sequence number as the sender maintains the mapping from the per-flow sequence number to the per-sub-flow sequence number. As we assume no reordering due to routing inside a network, the sender will detect losses (update packet state into `"Lost"`) upon receiving a duplicated ACK.

However, unlike TCP, FlexPass does not immediately retransmit lost packets upon detection. When an ACK notifying a loss arrives at the sender, we mark the packet as `"Lost"`, update the window size using the DCTCP algorithm, but still slide the left edge of the window of the reactive sub-flow. FlexPass only uses the proactive sub-flow to retransmit lost packets. This is because the delivery of proactive packets is guaranteed without experiencing congestion losses. In contrast, reactive packets are subject to selective dropping. Therefore, inspired by Aeolus [20], we choose to use the proactive sub-flow as the highly reliable channel for loss recovery.

This loss recovery mechanism effectively mitigates congestions when a FlexPass flow is competing with legacy reactive flows. When FlexPass flows are competing with legacy reactive flows, FlexPass does not allocate bandwidth for the reactive sub-flow. Thus, retransmitting packets via the proactive sub-flow instead of the reactive sub-flow would accelerate loss recovery. In contrast, naïve loss recovery mechanism (i.e. MPTCP with ExpressPass and DCTCP sub-flows) would have to wait for the reactive sub-flow to complete its loss recovery.

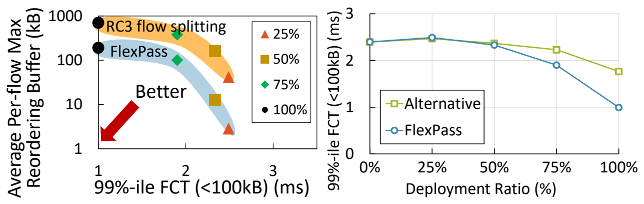
Optimizing for tail latency. Although the loss recovery design above seems promising, the ACK-based loss detection can lead to timeouts in case of tail losses [29]. For example, consider a flow with two packets, where the first is sent via the proactive sub-flow and the second packet is sent via the reactive sub-flow. Assume the second packet gets dropped. The sender can only rely on timeout to detect it, thus significantly increasing the flow completion time.

To optimize for tail latency, the proactive sub-flow proactively retransmits an unacknowledged segment assigned to the reactive sub-flow. The reason is that when reactive flows suffer from a tail loss, it results in a long timeout, dramatically increasing the FCT. To minimize redundant traffic in the network, we only trigger such "proactive retransmission" when the proactive sub-flow has no more `"Lost"` or `"Pending"` packets to send. Note that our simulation results show that proactive retransmission generates *only 0.7% of redundant retransmission in traffic volume,* when 50% of hosts switched to FlexPass.[5] At the receiver side, redundant packets are identified by per-flow sequence number and discarded during reassembly.

In summary, upon receiving a credit packet, the sender transmits a data packet via the proactive sub-flow in the following order:

1. `"Lost"`: When the sender detects the loss, the sender performs loss recovery instead of transmitting new data. This is to accelerate loss recovery and reduce the reordering of segments.
2. `"Pending"`: If there is no lost packet, the sender transmits new data.
3. `"Sent as Reactive"`: Otherwise, the sender retransmits an unacknowledged segment assigned to the reactive sub-flow. We call this *"proactive retransmission".*

[5]Web search workload. We use DCTCP for the legacy traffic. The deployment ratio is the fraction of ExpressPass/FlexPass-enabled racks. Refer to §6 for the detailed settings.



(a) vs. RC3 flow splitting (b) vs. alternative queueing

Figure 5. [Simulation] 99%-ile FCT comparison of FlexPass with different schemes under a realistic workload.

At the receiver side, FlexPass reassembles the data packets received via the proactive and reactive sub-flows. FlexPass uses the per-flow sequence number to reorder packets.

4.3 Discussions

Alternative flow splitting schemes. A few existing transport designs split a flow into multiple sub-flows. For example, RC3 uses two sub-flows (a primary control loop and a recursive low priority (RLP) control loop) running in different priorities to take advantage of available link capacity. To prevent redundant transmissions, the primary loop transmits data from the beginning, and the RLP loop transmits data from the end of the flow. When designing FlexPass, we also consider this design. However, we do not adopt this for two reasons. First, this scheme implicitly requires prior knowledge of the *entire* flow/message before the transmission. In other words, this scheme cannot be used for applications that continuously generate traffic [4], e.g. video streaming. Second, this scheme also requires a much larger reordering buffer (e.g., half of the flow size). In contrast, our design does not have these problems.

We use simulations to compare FlexPass and FlexPass that use RC3's flow splitting. Figure 5 (a) shows the 99%-ile flow completion times (FCTs) of small flows and average reordering buffer size in two different allocation schemes under a realistic workload[5]. Compared to using RC3, FlexPass uses less reordering buffer, while exhibiting comparable FCTs.

Alternative queuing scheme. Instead of putting both proactive and reactive sub-flows into a single queue (Q1), an obvious alternative is to assign proactive sub-flows into Q1, and put reactive sub-flows into Q2. At first glance, this scheme looks very promising; Q1 will have near-zero queuing and very low queue bounds even without selective dropping. Proactive sub-flows would have very low delay bounds.

Nonetheless, we do not adopt this design because reactive data packets can suffer from serious queuing delays and losses due to the bursty legacy traffic in Q2. The large queuing delay at Q2 also increases the size of the required reorder buffer as it will take longer for the sender to detect losses. While the reactive data packets are trapped inside the switch queue due to the long queuing delay, the sender may retransmit them via "proactive retransmission" when the sender has sent out all the remaining data packets. This will increase redundant traffic. Note that assigning each traffic a different

ECN marking or selective dropping threshold still cannot prevent reactive sub-flows from experiencing the same problem. We compare FlexPass with the alternative in Figure 5 (b) under a realistic workload.[5] The alternative scheme fails to achieve comparable performance with FlexPass.

Readers may wonder why FlexPass does not simply use three queues for proactive, reactive, and legacy flows, respectively. This obvious alternative, however, has two major drawbacks. First, this incurs severe reordering within a FlexPass flow, which requires a large reordering buffer at the host side. Second, FlexPass would no longer have a low delay bound, due to the queue buildup in the reactive sub-flow queue. The increase in the queuing delay of the reactive sub-flow would bring a longer FCT of the entire flow.

Credit waste mitigation. Perfect credit allocation is challenging at scale as it requires perfect coordination among all the links. For example, a sender may receive *excessive* credits from multiple receivers, and unavoidably wastes some credits due to the bottleneck of the sender's NIC. The credit waste problem has been widely identified and different proactive transports adopt different credit allocation enhancements, e.g., source downgrading in pHost [13], over-commitment in Homa [35], credit feedback control in ExpressPass [9], and token clocking in dcPIM [6].

FlexPass mitigates this problem from another perspective by using the reactive sub-flow to utilize *any* spare bandwidth in the network, not only that left by legacy traffic, but also that wasted by imperfect credit allocation. In large-scale simulations (Figure 10), we find that FlexPass can achieve better link utilization than ExpressPass at 100% deployment.

Handling proactive data packet losses. The proactive sub-flow does not experience any congestive loss by its design. Nonetheless, in practice, it can still experience non-congestion losses, e.g. due to the switch failures. To this end, we add an ACK-based loss recovery mechanism and a recovery timer to the proactive sub-flow. Once the sender detects a loss on the proactive sub-flow, FlexPass gives the highest transmission priority to the lost packet. Upon timeout, the sender will re-request credit and resume recovery.

Extensibility of FlexPass. In FlexPass, we adopt ExpressPass [9] for credit allocation as it can mitigate the congestion in the oversubscribed networks using commodity switch hardware. FlexPass can also apply other credit allocation algorithms, e.g., pHost [13] and dcPIM [6] in non-blocking networks with per-packet load balancing. In particular, the approach of FlexPass is useful for many transports that make assumptions on the network, since FlexPass alleviates stringent requirements on the prior knowledge of the network. Also, we adopt DCTCP [1] for reactive sub-flows, as it is well-adopted in production environments. We can also consider applying other reactive congestion control algorithms (e.g., loss-based, latency-based, or ECN-based) for the reactive sub-flows. We leave this as our future work.

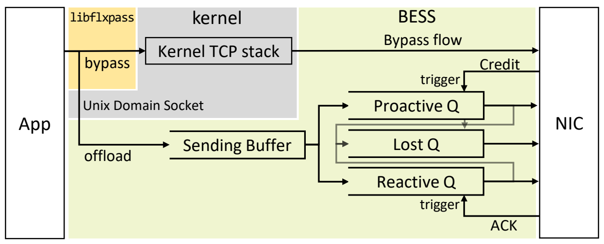


Figure 6. FlexPass network stack

Deployment scenario of FlexPass. To deploy FlexPass, operators need to upgrade both the switch configuration and the host networking stack. First, operators roll out the FlexPass switch configuration to every switch[6]. Note that the upgraded switch configuration does not affect legacy flows. Once the switch configuration is deployed, operators can gradually upgrade the host networking stack to FlexPass. We assume that operators deploy FlexPass in a per-rack manner, but operators may take different deployment strategies, e.g., per-host, per-VM.

Operators do not need to adjust the queue weight parameter w_q during the deployment of FlexPass, nor after the full deployment. Adjusting w_q during the deployment process is impractical since we need to update the parameters for each host. In fact, there are negligible performance differences between maintaining the initial w_q after the full deployment and adjusting w_q despite the impracticality (Figure 18).

5 Implementation

Host network stack. Our prototype provides high performance and backward compatibility to network applications without modifications. To achieve these goals, we implement FlexPass's transport protocol using Berkeley Extensible Software Switch (BESS) [17] and build a library `libflxpass` to enable applications to use FlexPass transparently. Figure 6 shows the architecture of FlexPass. `libflxpass` is a library in user space. It overrides the C runtime library to hook network system calls, e.g., `send()` and `recv()`, and exchanges application data with BESS via Unix domain socket. This enables legacy applications to use FlexPass without modifications.

BESS is an extensible software switch that exploits high-performance kernel bypass I/O framework (DPDK). We implement FlexPass's transport protocol according to §4.2 as a BESS module and insert it into the existing data path. A FlexPass packet has Ethernet, IP, UDP, FlexPass headers, and a payload. We use the IP and UDP headers to store the connection information (e.g., IP addresses and ports) and leverage the FlexPass header to store the information for reliable transmission. In the IP header, we use five DSCP values to distinguish packets in the network, including proactive data,

[6]NIC is essentially a special type of edge switch, in addition to physical switches in the network. Thus, the same configuration to support FlexPass should be applied to NICs. In VM-based cloud deployment, virtual switches connecting VMs also need to be upgraded as such.

reactive data, credit, FlexPass control packets (e.g., ACK), and legacy traffic. FlexPass header has 18 bytes in total, containing per-flow sequence number (4 B), per-sub-flow sequence number (4 B), and information required for ExpressPass.

For high performance, we also implement TCP segmentation offload (TSO), large receive offload (LRO), and jumbo frame support. We further implement logic to check and fall back to kernel TCP, if the remote host does not support FlexPass. We added 5.7k LoC to implement FlexPass on BESS.

Switch configuration. We allocate three queues per port and classify proactive credit packets into Q0, FlexPass data packets (both proactive and reactive) and control packets into Q1, and legacy reactive packets into Q2. Packets are classified into different queues based on their DSCP values.

FlexPass requires selective dropping and ECN marking at Q1 to throttle excessive reactive sub-flows. To minimize packet losses, we want to trigger ECN marking before selective dropping. Aeolus [20] implements selective dropping by using RED/ECN to drop non-ECN-capable packets. However, this approach will disable normal ECN marking.

Realizing this limitation, we use *color-aware dropping* [29] at switches to implement selective dropping while keeping RED/ECN to mark packets. A wide range of commodity switching chips (e.g., Broadcom Tomahawk and Trident series [21–24]) support this feature, which leverages per-packet metadata called *color* (red, yellow, and green). By setting up the access control list (ACL), operators can designate packet colors, for example, by mapping a certain DSCP value to a color. Operators can associate a buffer threshold for each color within the same queue. Switches keep track of the queue length of packets *in each color*, and drop incoming packets if the length exceeds the associated threshold. To selectively drop reactive data packets, we mark reactive data packets as red and set a buffer threshold to limit red packets. This buffer threshold should be larger than the ECN marking threshold. We assume all the switches are fully upgraded to support FlexPass before gradually deploying FlexPass at hosts. Note this upgrade to switch does not affect legacy flows at all.

6 Evaluation

We evaluate FlexPass using testbed experiments and ns-2 simulations [31]. Our main findings are as follows:

- Our testbed experiments show FlexPass co-exists well with legacy transport and it achieves high-performance properties of proactive congestion control, including high throughput and zero timeouts. (§6.1)
- In large-scale simulations, we compare FlexPass with other alternative deployment schemes; FlexPass provides gradual deployability with *the best* incremental benefits. During the deployment, FlexPass improves the 99%-ile FCT and keeps the average FCT low, while posing minimal harm to legacy traffic. FlexPass improves the 99%-ile FCT of small flows up to 44% after full deployment. (§6.2)
- Through a series of targeted simulations, we show that FlexPass can deliver incremental benefits under a higher load. We further explore the space of parameters and evaluate the trade-off relationship of such parameters. (§6.3)

6.1 Testbed Experiments

Setup. We build a testbed with 9 servers connected to a Netberg Aurora 720 switch which uses Broadcom Tomahawk ASIC and OpenSwitch 2.0.4 as the switch OS. Each server has an Intel 10 GbE NIC and runs Linux kernel 5.4.0. We set RTO_{min} of kernel TCP to 4 ms. We enable three queues at the switch. We set a strict high priority on the credit queue (queue 0), and enable DWRR on the remaining queues (queue 1 and 2) with equal queue weight. We set w_q to 0.5 and limit the bandwidth of the credit queue to ≈0.4% of the link capacity, which allows proactive sub-flows to take up to half of the link capacity. We use five DSCP values to differentiate proactive data packets, reactive data packets, credit packets, FlexPass control packets, and legacy packets. At queue 1, we mark proactive data packets and FlexPass control packets as green, and reactive data packets as red by configuring QoS DSCP mapping. We set the buffer threshold of red packets (selective dropping threshold) to 100 kB and set the ECN marking threshold to 60 kB.

Microbenchmark with long-running flows. Maintaining co-existence with the legacy congestion control is one of the key goals of this paper. We conduct an experiment to verify how FlexPass fairly shares the bandwidth with legacy reactive flows and how the existing approach fails to achieve co-existence. We build a two-to-one topology connected via a single switch. Each sender generates a single 50 MB flow.

Figure 9 (a) depicts the throughput when one DCTCP flow and one ExpressPass flow compete for a single 10 Gbps link in our testbed. When the ExpressPass flow exists, the DCTCP flow takes only 9.3% of the link capacity, while the ExpressPass flow takes the rest, as pointed out in §2.2. This may cause starvation during the phased rollout of new congestion control, resulting in high tail latency of legacy flows. In contrast, Figure 9 (b) shows the throughput when the DCTCP flow and the FlexPass flow compete under the same scenario. In this case, DCTCP and FlexPass flows take 51% and 48% of the link bandwidth respectively. To quantify the degree of legacy flows' starvation during the phased rollout of new congestion control, we measure the starvation time in Figure 9 (c), which shows the duration of each transport's bandwidth being less than 20%. While ExpressPass shows 97% of the starvation time, FlexPass does not starve legacy flows (<0.1%) and co-exists with legacy congestion control.

We further explore the bandwidth allocation between proactive and reactive sub-flows under multiple scenarios. Figure 7 (a) shows when only one FlexPass flow exists inside the link. As proactive sub-flows can take up to half of the link capacity, reactive sub-flow takes the rest and fully utilizes the link. In contrast, Figure 7 (b) shows when two

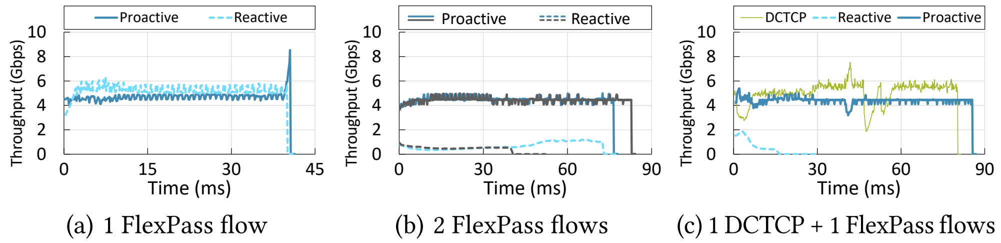


Figure 7. [Testbed] Throughput of each sub-flow under multiple scenarios

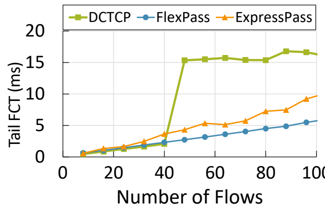


Figure 8. [Testbed] Tail FCT of 64 kB flows on incast.

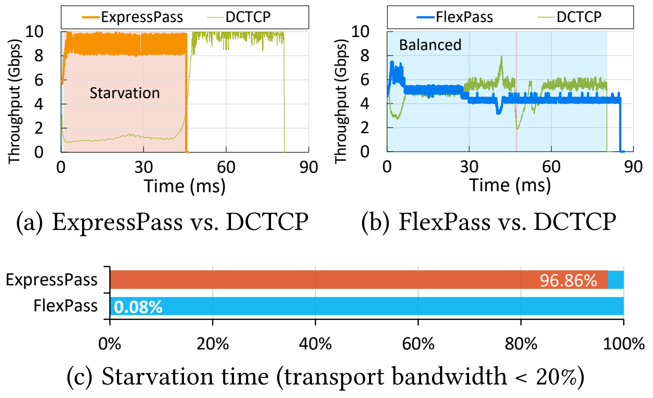


Figure 9. [Testbed] Throughput when ExpressPass and FlexPass are competing with DCTCP

FlexPass flows compete for the bottleneck link. Two proactive sub-flows from each flow try to take up to half of the link capacity, which brings the starvation of reactive sub-flows. Each flow fairly shares the bandwidth and mainly transmits the data using the proactive sub-flow. When a DCTCP and FlexPass flow compete, as in Figure 7 (c), both DCTCP and FlexPass use half of the link capacity, which is the minimum guaranteed bandwidth for each flow. Note that the proactive sub-flow tries to take up half of the link capacity, and the reactive sub-flow can hardly occupy the bandwidth.

Incast. We show that FlexPass does not experience any timeout even in a high incast degree. We create an 8-to-1 incast and measure the maximum FCT while increasing the number of flows. The receiver generates multiple synchronized requests toward 8 hosts, and they respond with a 64 kB-sized response. The requests are evenly distributed to each host. We report the average of the longest FCT of two runs.

Figure 8 shows the maximum FCT for kernel DCTCP stack, ExpressPass, and FlexPass. DCTCP experiences a timeout with more than 48 flows. Since DCTCP is a reactive congestion control that barely handles bursty flow arrivals, it cannot recover from tail loss without a timeout. In contrast, ExpressPass and FlexPass do not trigger any timeout, reducing the maximum FCT compared to DCTCP up to 83.5%. Note that FlexPass shows better FCT than Expresspass at a high incast degree. This is because the reactive sub-flow can utilize the first RTT before the credit arrives, as in Aeolus [20]. This especially brings benefits for small flows.

6.2 Large-scale Simulations

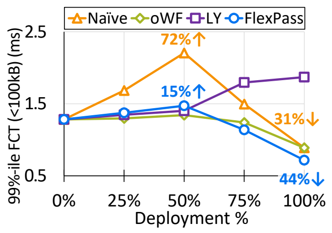
We use ns-2 simulations to evaluate FlexPass and other alternative deployment schemes to transit traffic from DCTCP to FlexPass/ExpressPass in large-scale networks.

Settings. We simulate a 3-tier Clos 40Gbps topology. The topology has 8 core switches, 16 aggregation switches, 32 ToR switches, and 192 hosts. The topology has a 3:1 oversubscription ratio at the ToR switch level. We set per-link propagation delay and host delay to 4 μs and 2 μs, respectively, which results in 28 μs base RTT across the core switch (6 hops). Each switch has 8 40Gbps ports and a 4.5MB shared buffer. We implement a dynamic buffer management mechanism according to [10], and set the egress dynamic buffer threshold to 1/4. As required by ExpressPass, we use ECMP routing with symmetric hash for fabric load balancing.

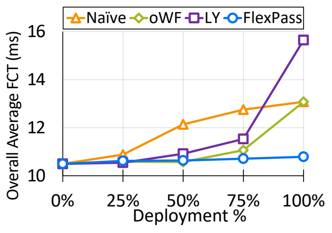
For ExpressPass, we set aggressiveness factor α to 2.0, the minimum change S_{min} to 1 credit per RTT, and the maximum change S_{max} to 50 Mbps which corresponds to 1 Gbps of returning data. For DCTCP, we set the minimum RTO to 4 ms, and the ECN marking threshold to 100 kB, which is high enough to use 40Gbps throughput.

Schemes compared. We evaluate the following schemes.

- **Naïve**: Naïve deployment of ExpressPass without any measure to maintain co-existence. ExpressPass data packets and legacy traffic (DCTCP) co-exist in the same queue. ExpressPass allocates credits according to the line rate.
- **Oracle Weighted Fair Queueing (oWF)**: ExpressPass data packets and legacy DCTCP traffic are separated using two queues scheduled by DWRR. The ideal queue weight and credit allocation rate depend on the fraction of ExpressPass traffic, which varies across time and space in production DCNs. In our controllable simulations, to explore the best performance that weighted fair queueing can achieve, we use the prior knowledge of the total fraction of ExpressPass traffic to compute the queue weight and credit allocation rate in advance and apply them to all the switches and hosts.
- **Layering (LY) [45]** scheme overlays congestion control of DCTCP and ExpressPass. This scheme adds a window limit on the top of ExpressPass, adjusted according to the DCTCP algorithm. ExpressPass data traffic and legacy traffic co-exist in the same queue. This scheme effectively mitigates the starvation of legacy traffic and provides fair

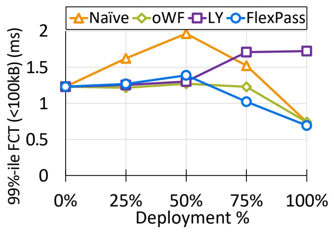


(a) 99%-ile FCT (<100 kB)

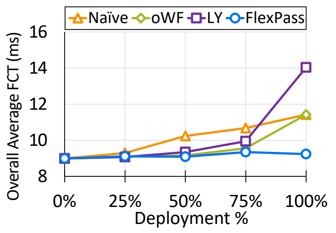


(b) Overall average FCT

Figure 10. [Simulation] FCTs during the transition from DCTCP to the new transport.



(a) 99%-ile FCT (<100 kB)



(b) Overall average FCT

Figure 11. [Simulation] FCTs during the transition from DCTCP to the new transport (mixed traffic).

co-existence, since a data packet can only be sent when a credit packet arrives and the window allows.

- **FlexPass**: We use equal weights for FlexPass (Q1) and legacy traffic (Q2) regardless of the deployment ratio. At Q1, we set the selective dropping threshold and ECN marking threshold to 150 kB and 65 kB. Note 65 kB is high enough for reactive sub-flows to use spare bandwidth.

Benchmark workloads. We evaluate the above schemes under two scenarios: one with only background traffic, and the other one with mixed traffic where background traffic and foreground incast traffic co-exist. We generate background traffic based on realistic flow size distributions of the web search workload [2]. We choose a pair of hosts randomly as the source and destination for each flow. Flows arrive according to a Poisson process. We vary the flow arrival rate to set the network load (utilization) of the core (up links of ToR) to 50%. In a mixed traffic scenario, we set 10% of the total traffic volume as foreground traffic. To generate foreground traffic, we randomly select a receiver, and each of the other hosts sends four 8 kB flows to the receiver. We assume all switches are configured to support FlexPass/ExpressPass before deployment at hosts, and vary the fraction of FlexPass/ExpressPass-enabled racks from 0% to 100%. A flow uses FlexPass/ExpressPass only if both endpoints are FlexPass/ExpressPass-enabled.

We use flow completion time (FCT) as the main performance metric. We consider the average FCT across all flows, which reflects bandwidth utilization, and the 99th percentile FCT for small flows (<100 KB), which reflects the tail latency.

Incremental benefit. Figure 10 shows the results with only background flows under four deployment schemes at different stages of deployment. In general, FlexPass and weighted fair sharing (WF) schemes greatly reduce the FCT of small flows with few side effects during deployment, but naïve deployment of ExpressPass and layering (LY) exhibit significant performance degradation during deployment.

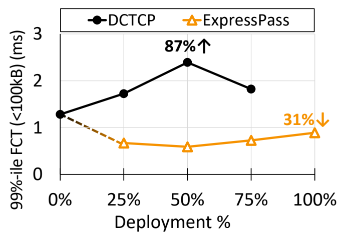
FlexPass cuts 99%-ile FCT of small flows by up to 44% when fully deployed, and generally provides the best 99%-ile FCT across different deployment scenarios. Moreover, FlexPass causes nearly no harm toward the overall average FCT during and after deployment. FlexPass maintains high utilization at any stage of deployment. Note that FlexPass is even better than the original ExpressPass (oWF and naïve) at 100% deployment. As discussed in §4.3, this is because the reactive sub-flow of FlexPass fully utilizes the spare bandwidth, including the bandwidth wasted by imperfect credit allocation.

In contrast, naïve deployment of ExpressPass has a severe impact on both tail latency and throughput. Although it reduces 99%-ile FCT when fully deployed (up to 31%), this harms the performance of legacy flows, inflating the tail latency by up to 72% during deployment. Also, the layering scheme (LY) does not improve performance during and after deployment. This is because, under the layering scheme, the window may unnecessarily limit packet transmissions, even if there is no other legacy traffic competing for the link. This will cause serious bandwidth waste, and greatly increase the FCT of short flows, which can finish in a few RTTs at the line rate. Note FlexPass outperforms the oracle weighted fair queueing (oWF) in our evaluation. This is because oWF's queue weight and credit allocation cannot be perfect due to variations in the fraction of ExpressPass and legacy traffic across links and time.

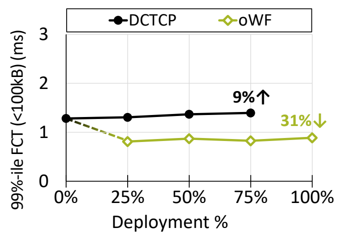
Figure 11 shows the simulation results with both foreground and background flows. The observation is similar to that with only background traffic; FlexPass brings the reduction of FCT up to 44% with only 13% of performance degradation during deployment, while other schemes exhibit up to 60% of performance degradation during deployment.

Takeaway 1. *During the deployment of FlexPass, it shows similar performance as the oracle weighted fair queuing, which is close to ideal but impractical.*

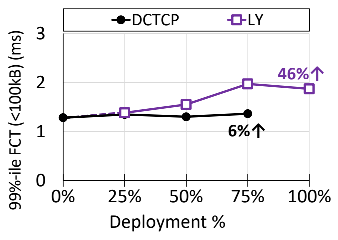
Coexistence. Figure 12 and Figure 13 show the simulation results with only background flows under the same deployment scenario. We plot legacy reactive flows (DCTCP) and proactive flows (FlexPass/ ExpressPass) into two different lines. The naïve deployment of ExpressPass causes severe side effects, increasing the tail latency of the legacy flows up to 87%. This comes with a 127% increase in the standard deviation of FCT, which drastically reduces the predictability. In contrast, FlexPass causes only a minimal amount of harm to legacy traffic during the deployment. Legacy traffic only experiences a 19% increase in the standard deviation of FCT. Note the performance of FlexPass is similar to that of the oracle weighted fair queuing scheme (oWF).



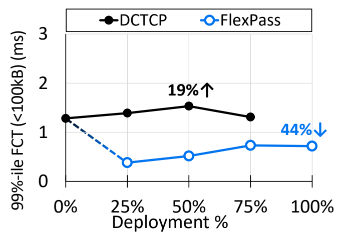
(a) Naïve ExpressPass



(b) Oracle WFQ

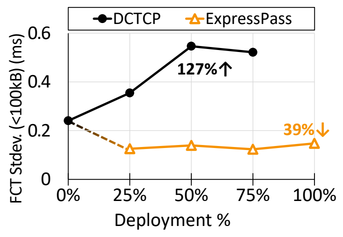


(c) Layering

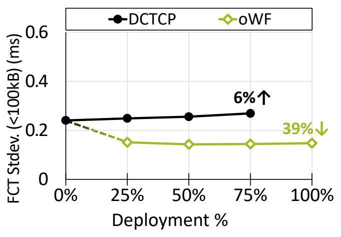


(d) FlexPass

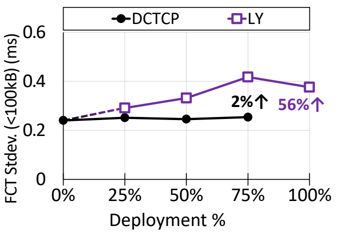
Figure 12. [Simulation] 99%-ile FCT (<100 kB) by the type of flows during the transition.



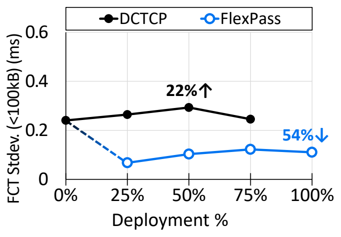
(a) Naïve ExpressPass



(b) Oracle WFQ



(c) Layering



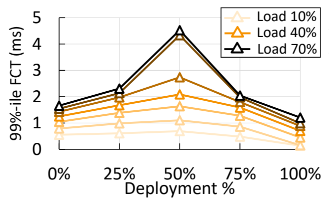
(d) FlexPass

Figure 13. [Simulation] Standard deviation of FCT of small flows (<100 kB) by the type of flows during the transition.

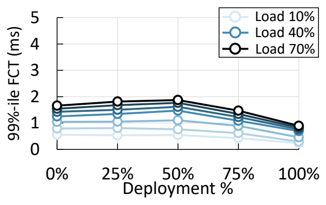
Latency benefits. We show that FlexPass also preserves the low latency benefit of proactive transports. Figure 12 (d) shows that traffic converted to FlexPass experiences better tail latency under co-existence. When FlexPass is fully deployed, FlexPass improves the 99%-ile FCT by 44% compared to legacy traffic (DCTCP). FlexPass provides even better FCT improvement than naïve ExpressPass deployment. As mentioned above, this is because FlexPass can better utilize the pre-credit phase than ExpressPass.

Bounded queue. Note that Q1 maintains a bounded queue, as the maximum queue buildup of reactive sub-flows is limited under the selective dropping threshold (150 kB). However, the queue occupancy is much smaller than the maximum even though, to utilize spare bandwidth, reactive traffic requires moderate buffer space. To show this, we measure the average and 90%-ile queue buildup of Q1 (FlexPass queue) during the deployment. We find that the maximum queue buildup reaches the peak at 50% deployment, and the queue buildup is usually much lower than the threshold. On average, it reaches 10.6 kB including 6.15 kB of the reactive packets. In 90%-ile, it reaches 29 kB including 21 kB of reactive packets. At full deployment, the average and 90%-ile queue length reduce to 22.0 kB and 73.9 kB respectively, including 13.3 kB and 44.7 kB of reactive packets. Note that FlexPass does not incur excessive retransmissions due to selective dropping. FlexPass only experiences <0.1% of packet drop due to the selective dropping at full deployment.

Multiple workloads. We also show the simulation results with multiple realistic workloads, including cache follower [41], data mining [14], and Hadoop [41]. FlexPass improves 99%-ile FCT up to 63% with minimal side-effect toward legacy traffic. Please refer to Appendix A for detailed results.



(a) ExpressPass



(b) FlexPass

Figure 14. [Simulation] 99%-ile FCT of small flows during the transition under different network loads.

Takeaway 2. *During the deployment, the legacy traffic remains unaffected by the introduction of FlexPass. Traffic converted to FlexPass benefits from the low latency and bounded queue length of proactive transports even under the co-existence.*

6.3 Deep Dive

Sensitivity to network load. We evaluate FlexPass and other schemes under different network loads. We measure 99%-ile FCT of small flows (<100 kB) during the deployment with only background traffic, while varying the network utilization. Figure 14 (a) illustrates the performance penalty during the deployment of ExpressPass. At a low load (less than 20%), ExpressPass rarely experiences performance degradation during deployment, as the chance for DCTCP and ExpressPass flows to co-exist in the same link is very low. However, as link utilization increases, the performance gets more penalized. Especially, DCTCP traffic experiences timeouts when the load is higher than 60%. In contrast, as shown in Figure 14 (b), FlexPass does not show performance degradation during deployment, even at a very high load (70%).

Additional findings. We also report simulation results regarding the impact of the selective dropping threshold and the queue weight configuration (w_q) in Appendix A. We summarize our findings below.

- Lower selective dropping threshold reduces FCT when FlexPass is fully deployed, but results in worse average throughput, due to increased packet drops.
- FlexPass is insensitive to the configuration of the queue weight(w_q), unlike ExpressPass where the mismatched weight configuration brings a high penalty in tail latency.

7 Related Work

The literature of datacenter transport is vast, from traditional reactive congestion control [1, 26–28, 32, 48] to proactive [9, 13, 18, 35, 36], from loss recovery [8, 12, 34, 44, 46] to flow control [11, 39]. In the interest of the space, we only discuss the ideas that are most relevant to FlexPass.

Hu et al. [20] proposed Aeolus to enable proactive transports to transmit data using the spare bandwidth in the "pre-credit" phase. FlexPass uses similar techniques but addresses a different problem. In FlexPass, the reactive sub-flow sends packets to use the spare bandwidth left by legacy traffic during the flow's whole lifetime, not limited to the pre-credit phase. FlexPass and Aeolus use selective dropping to limit the impact of unscheduled (reactive) packets on scheduled (proactive) packets, but leverage different switch features to implement this idea.

Some transport designs also split a single flow into multiple sub-flows, but address different problems. Multipath TCP (MPTCP) [40] leverages multiple paths for a single TCP connection to provide better redundancy and maximize resource utilization. MP-RDMA [30] applies this idea to RDMA in datacenters. In FlexPass, two sub-flows complement each other on the same path. RC3 [33] leverages multiple sub-flows in different priorities to quickly take advantage of the available bandwidth from the first RTT. RPO [19] takes a similar approach to RC3 to improve network utilization of receiver-driven transport. FlexPass does not use multiple priority queues to separate sub-flows in the network, and uses a different approach to split the flow into sub-flows.

8 Conclusion

In this paper, we introduce FlexPass, a practical credit-based transport designed to co-exist with legacy traffic by providing deployment flexibility. FlexPass leverages a combination of switch features and end-host designs to achieve fair co-existence with legacy traffic, while preserving the high-performance property of proactive transport. FlexPass ensures co-existence with legacy traffic by isolating FlexPass and legacy traffic using multiple queues, and allocating credits according to the minimum guaranteed bandwidth. To preserve the great properties of proactive transport, FlexPass splits a single flow into proactive and reactive sub-flows, and schedules two sub-flows at the host to maximize the link utilization and facilitate loss recovery. Our evaluation shows that FlexPass preserves the property of proactive transport including high-throughput, bounded queue, and zero timeouts. We verify that FlexPass is gradually deployable and delivers incremental benefits as the deployment progresses through extensive simulations.

Acknowledgments

We would like to thank our shepherd Edouard Bugnion and anonymous reviewers for their constructive feedback. This work is supported by Institute for Information & communications Technology Promotion (IITP) grant funded by the Korea government (Ministry of Science and ICT) [No.2018-0-00693].

References

[1] Mohammad Alizadeh, Albert Greenberg, David A Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. 2010. Data center tcp (DCTCP). In *Proceedings of the ACM SIGCOMM 2010 Conference*. Association for Computing Machinery, New York, NY, USA, 63–74.

[2] 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*. ACM, Association for Computing Machinery, New York, NY, USA, 73–84.

[3] Mohammad Alizadeh, Shuang Yang, Milad Sharif, Sachin Katti, Nick McKeown, Balaji Prabhakar, and Scott Shenker. 2013. pfabric: Minimal near-optimal datacenter transport. *ACM SIGCOMM Computer Communication Review* 43, 4 (2013), 435–446.

[4] 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)*. USENIX Association, Oakland, CA, 455–468.

[5] Lawrence S. Brakmo, Sean W. O'Malley, and Larry L. Peterson. 1994. TCP Vegas: New Techniques for Congestion Detection and Avoidance. *SIGCOMM Comput. Commun. Rev.* 24, 4 (Oct. 1994), 24–35. https://doi.org/10.1145/190809.190317

[6] Qizhe Cai, Mina Tahmasbi Arashloo, and Rachit Agarwal. 2022. DcPIM: Near-Optimal Proactive Datacenter Transport. In *Proceedings of the ACM SIGCOMM 2022 Conference* (Amsterdam, Netherlands) *(SIGCOMM '22)*. Association for Computing Machinery, New York, NY, USA, 53–65. https://doi.org/10.1145/3544216.3544235

[7] Neal Cardwell, Yuchung Cheng, C Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. 2016. BBR: Congestion-based congestion control. *Queue* 14, 5 (2016), 20–53.

[8] Peng Cheng, Fengyuan Ren, Ran Shu, and Chuang Lin. 2014. Catch the Whole Lot in an Action: Rapid Precise Packet Loss Notification in Data Center. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*. USENIX Association, Seattle, WA, 17–28. https://www.usenix.org/conference/nsdi14/technical-sessions/presentation/cheng

[9] Inho Cho, Keon Jang, and Dongsu Han. 2017. Credit-Scheduled Delay-Bounded Congestion Control for Datacenters. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication* (Los Angeles, CA, USA) *(SIGCOMM '17)*. Association for Computing Machinery, New York, NY, USA, 239–252. https://doi.org/10.1145/3098822.3098840

[10] Abhijit K Choudhury and Ellen L Hahne. 1998. Dynamic queue length thresholds for shared-memory packet switches. *IEEE/ACM Transactions On Networking* 6, 2 (1998), 130–140.

[11] Claudio DeSanti. 2011. IEEE DCB. 802.1Qbb - Priority-based Flow Control. https://1.ieee802.org/dcb/802-1qbb/.

[12] Tobias Flach, Nandita Dukkipati, Andreas Terzis, Barath Raghavan, Neal Cardwell, Yuchung Cheng, Ankur Jain, Shuai Hao, Ethan Katz-Bassett, and Ramesh Govindan. 2013. Reducing Web Latency: The Virtue of Gentle Aggression. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM* (Hong Kong, China) *(SIGCOMM '13)*. Association for Computing Machinery, New York, NY, USA, 159–170. https://doi.org/10.1145/2486001.2486014

[13] 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 *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies* (Heidelberg, Germany) *(CoNEXT '15)*. Association for Computing Machinery, New York, NY, USA, Article 1, 12 pages. https://doi.org/10.1145/2716281.2836086

[14] 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 *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication* (Barcelona, Spain) *(SIGCOMM '09)*. Association for Computing Machinery, New York, NY, USA, 51–62. https://doi.org/10.1145/1592568.1592576

[15] Chuanxiong Guo, Haitao Wu, Zhong Deng, Gaurav Soni, Jianxi Ye, Jitu Padhye, and Marina Lipshteyn. 2016. RDMA over Commodity Ethernet at Scale. In *Proceedings of the 2016 ACM SIGCOMM Conference* (Florianopolis, Brazil) *(SIGCOMM '16)*. Association for Computing Machinery, New York, NY, USA, 202–215. https://doi.org/10.1145/2934872.2934908

[16] 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), 64–74.

[17] 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, Berkeley, CA.

[18] Mark Handley, Costin Raiciu, Alexandru Agache, Andrei Voinescu, Andrew W. Moore, Gianni Antichi, and Marcin Wójcik. 2017. Re-Architecting Datacenter Networks and Stacks for Low Latency and High Performance. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication* (Los Angeles, CA, USA) *(SIGCOMM '17)*. Association for Computing Machinery, New York, NY, USA, 29–42. https://doi.org/10.1145/3098822.3098825

[19] Jinbin Hu, Jiawei Huang, Zhaoyi Li, Yijun Li, Wenchao Jiang, Kai Chen, Jianxin Wang, and Tian He. 2021. RPO: Receiver-driven Transport Protocol Using Opportunistic Transmission in Data Center. In *2021 IEEE 29th International Conference on Network Protocols (ICNP)*. IEEE, International Conference on Network Protocols, Virtual Event, 1–11.

[20] Shuihai Hu, Wei Bai, Gaoxiong Zeng, Zilong Wang, Baochen Qiao, Kai Chen, Kun Tan, and Yi Wang. 2020. Aeolus: A Building Block for Proactive Transport in Datacenters. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication* (Virtual Event, USA) *(SIGCOMM '20)*. Association for Computing Machinery, New York, NY, USA, 422–434. https://doi.org/10.1145/3387514.3405878

[21] Broadcom Inc. 2016. Broadcom First to Deliver 64 Ports of 100GE with Tomahawk II 6.4Tbps Ethernet Switch. https://www.broadcom.com/news/product-releases/broadcom-first-to-deliver-64-ports-of-100ge-with-tomahawk-ii-ethernet-switch.

[22] Broadcom Inc. 2016. High-Capacity StrataXGS® Trident II Ethernet Switch Series. https://www.broadcom.com/products/ethernet-connectivity/switching/strataxgs/bcm56850-series.

[23] Broadcom Inc. 2016. High-Density 25/100 Gigabit Ethernet StrataXGS Tomahawk Ethernet Switch Series. https://www.broadcom.com/products/ethernet-connectivity/switching/strataxgs/bcm56960-series.

[24] Broadcom Inc. 2017. 12.8 Tb/s StrataXGS Tomahawk 3 Ethernet Switch Series. https://www.broadcom.com/products/ethernet-connectivity/switching/strataxgs/bcm56980-series.

[25] Glenn Judd. 2015. Attaining the Promise and Avoiding the Pitfalls of TCP in the Datacenter. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. USENIX Association, Oakland, CA, 145–157. https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/judd

[26] Gautam Kumar, Nandita Dukkipati, Keon Jang, Hassan M. G. Wassel, Xian Wu, Behnam Montazeri, Yaogong Wang, Kevin Springborn, Christopher Alfeld, Michael Ryan, David Wetherall, and Amin Vahdat. 2020. Swift: Delay is Simple and Effective for Congestion Control in the Datacenter. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication* (Virtual Event, USA) *(SIGCOMM '20)*. Association for Computing Machinery, New York, NY, USA, 514–528. https://doi.org/10.1145/3387514.3406591

[27] C. Lee, C. Park, K. Jang, S. Moon, and D. Han. 2017. DX: Latency-Based Congestion Control for Datacenters. *IEEE/ACM Transactions on Networking* 25, 1 (2017), 335–348.

[28] Yuliang Li, Rui Miao, Hongqiang Harry Liu, Yan Zhuang, Fei Feng, Lingbo Tang, Zheng Cao, Ming Zhang, Frank Kelly, Mohammad Alizadeh, and Minlan Yu. 2019. HPCC: High Precision Congestion Control. In *Proceedings of the ACM Special Interest Group on Data Communication*. Association for Computing Machinery, New York, NY, USA, 44–58. https://doi.org/10.1145/3341302.3342085

[29] Hwijoon Lim, Wei Bai, Yibo Zhu, Youngmok Jung, and Dongsu Han. 2021. Towards Timeout-Less Transport in Commodity Datacenter Networks. In *Proceedings of the Sixteenth European Conference on Computer Systems* (Online Event, United Kingdom) *(EuroSys '21)*. Association for Computing Machinery, New York, NY, USA, 33–48. https://doi.org/10.1145/3447786.3456227

[30] Yuanwei Lu, Guo Chen, Bojie Li, Kun Tan, Yongqiang Xiong, Peng Cheng, Jiansong Zhang, Enhong Chen, and Thomas Moscibroda. 2018. Multi-Path Transport for RDMA in Datacenters. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. USENIX Association, Renton, WA, 357–371. https://www.usenix.org/conference/nsdi18/presentation/lu

[31] Steven McCanne, Sally Floyd, Kevin Fall, Kannan Varadhan, et al. 1997. Network simulator ns-2.

[32] Radhika Mittal, Vinh The Lam, Nandita Dukkipati, Emily Blem, Hassan Wassel, Monia Ghobadi, Amin Vahdat, Yaogong Wang, David Wetherall, and David Zats. 2015. TIMELY: RTT-Based Congestion Control for the Datacenter. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication* (London, United Kingdom) *(SIGCOMM '15)*. Association for Computing Machinery, New York, NY, USA, 537–550. https://doi.org/10.1145/2785956.2787510

[33] Radhika Mittal, Justine Sherry, Sylvia Ratnasamy, and Scott Shenker. 2014. Recursively Cautious Congestion Control. In *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation* (Seattle, WA) *(NSDI'14)*. USENIX Association, USA, 373–385.

[34] Radhika Mittal, Alexander Shpiner, Aurojit Panda, Eitan Zahavi, Arvind Krishnamurthy, Sylvia Ratnasamy, and Scott Shenker. 2018. Revisiting Network Support for RDMA. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication* (Budapest, Hungary) *(SIGCOMM '18)*. Association for Computing Machinery, New York, NY, USA, 313–326. https://doi.org/10.1145/

3230543.3230557

[35] Behnam Montazeri, Yilong Li, Mohammad Alizadeh, and John Ousterhout. 2018. Homa: A Receiver-Driven Low-Latency Transport Protocol Using Network Priorities. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication* (Budapest, Hungary) *(SIGCOMM '18)*. Association for Computing Machinery, New York, NY, USA, 221–235. https://doi.org/10.1145/3230543.3230564

[36] Jonathan Perry, Amy Ousterhout, Hari Balakrishnan, Devavrat Shah, and Hans Fugal. 2015. Fastpass: A centralized zero-queue datacenter network. *ACM SIGCOMM Computer Communication Review* 44, 4 (2015), 307–318.

[37] Jon Postel. 1981. *Transmission Control Protocol.* STD 7. RFC Editor. http://www.rfc-editor.org/rfc/rfc793.txt http://www.rfc-editor.org/rfc/rfc793.txt.

[38] Qifan Pu, Ganesh Ananthanarayanan, Peter Bodik, Srikanth Kandula, Aditya Akella, Paramvir Bahl, and Ion Stoica. 2015. Low latency geo-distributed data analytics. *ACM SIGCOMM Computer Communication Review* 45, 4 (2015), 421–434.

[39] Kun Qian, Wenxue Cheng, Tong Zhang, and Fengyuan Ren. 2019. Gentle Flow Control: Avoiding Deadlock in Lossless Networks. In *Proceedings of the ACM Special Interest Group on Data Communication* (Beijing, China) *(SIGCOMM '19)*. Association for Computing Machinery, New York, NY, USA, 75–89. https://doi.org/10.1145/3341302.3342065

[40] Costin Raiciu, Christoph Paasch, Sebastien Barre, Alan Ford, Michio Honda, Fabien Duchene, Olivier Bonaventure, and Mark Handley. 2012. How Hard Can It Be? Designing and Implementing a Deployable Multipath TCP. In *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*. USENIX Association, San Jose, CA, 399–412. https://www.usenix.org/conference/nsdi12/technical-sessions/presentation/raiciu

[41] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, George Porter, and Alex C. Snoeren. 2015. Inside the Social Network's (Datacenter) Network. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication* (London, United Kingdom) *(SIGCOMM '15)*. Association for Computing Machinery, New York, NY, USA, 123–137. https://doi.org/10.1145/2785956.2787472

[42] M. Shreedhar and George Varghese. 1995. Efficient Fair Queueing Using Deficit Round Robin. In *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication* (Cambridge, Massachusetts, USA) *(SIGCOMM '95)*. Association for Computing Machinery, New York, NY, USA, 231–242. https://doi.org/10.1145/217382.217453

[43] Arjun Singh, Joon Ong, Amit Agarwal, Glen Anderson, Ashby Armistead, Roy Bannon, Seb Boving, Gaurav Desai, Bob Felderman, Paulie Germano, Anand Kanagala, Jeff Provost, Jason Simmons, Eiichi Tanda, Jim Wanderer, Urs Hölzle, Stephen Stuart, and Amin Vahdat. 2015. Jupiter Rising: A Decade of Clos Topologies and Centralized Control in Google's Datacenter Network. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication* (London, United Kingdom) *(SIGCOMM '15)*. Association for Computing Machinery, New York, NY, USA, 183–197. https://doi.org/10.1145/2785956.2787508

[44] Vijay Vasudevan, Amar Phanishayee, Hiral Shah, Elie Krevat, David G Andersen, Gregory R Ganger, Garth A Gibson, and Brian Mueller. 2009. Safe and effective fine-grained TCP retransmissions for datacenter communication. *ACM SIGCOMM computer communication review* 39, 4 (2009), 303–314.

[45] Zihao Wei, Dezun Dong, Shan Huang, and Liquan Xiao. 2019. ExpressPass+: ECN-Friendly Credit Reservation Congestion Control for Datacenters. In *Proceedings of the ACM SIGCOMM 2019 Conference Posters and Demos* (Beijing, China) *(SIGCOMM Posters and Demos '19)*. Association for Computing Machinery, New York, NY, USA, 169–171. https://doi.org/10.1145/3342280.3342348

[46] David Zats, Anand Padmanabha Iyer, Ganesh Ananthanarayanan, Rachit Agarwal, Randy Katz, Ion Stoica, and Amin Vahdat. 2015. FastLane: Making Short Flows Shorter with Agile Drop Notification. In *Proceedings of the Sixth ACM Symposium on Cloud Computing* (Kohala Coast, Hawaii) *(SoCC '15)*. Association for Computing Machinery, New York, NY, USA, 84–96. https://doi.org/10.1145/2806777.2806852

[47] Qiao Zhang, Vincent Liu, Hongyi Zeng, and Arvind Krishnamurthy. 2017. High-Resolution Measurement of Data Center Microbursts. In *Proceedings of the 2017 Internet Measurement Conference* (London, United Kingdom) *(IMC '17)*. Association for Computing Machinery, New York, NY, USA, 78–85. https://doi.org/10.1145/3131365.3131375

[48] 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 *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication* (London, United Kingdom) *(SIGCOMM '15)*. Association for Computing Machinery, New York, NY, USA, 523–536. https://doi.org/10.1145/2785956.2787484

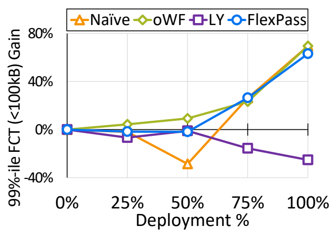
[49] Danyang Zhuo, Qiao Zhang, Xin Yang, and Vincent Liu. 2016. Canaries in the Network. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks* (Atlanta, GA, USA) *(HotNets '16)*. Association for Computing Machinery, New York, NY, USA, 36–42. https://doi.org/10.1145/3005745.3005767

Appendix A Additional simulation results

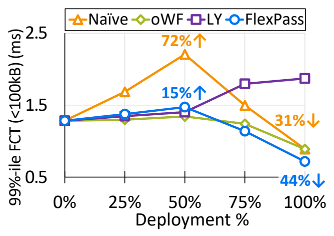
Result with multiple workloads. Figure 15 and Figure 16 show the simulation results with only background flows under four different realistic workloads, while varying the fraction of FlexPass/ExpressPass-enabled racks. As mentioned in §6.2, FlexPass and oracle weighted fair sharing (oWF) schemes greatly reduce the FCT of small flows with few side effects during deployment across all workloads. Naïve deployment of ExpressPass and layering (LY) exhibit significant performance degradation during deployment.

FlexPass cuts 99%-ile FCT of small flows by up to 63% when fully deployed and generally provides the best 99%-ile FCT across different deployment scenarios and different workloads. Moreover, FlexPass does nearly no harm toward the overall average FCT during deployment and after deployment. This shows FlexPass maintains high utilization at any degree of the deployment. In contrast, naïve deployment of ExpressPass has a severe impact on both tail latency and throughput during deployment. Although naïve deployment brings a reduction in 99%-ile FCT when fully deployed (up to 77%), this harms the performance of legacy DCTCP flows and thus inflates the tail latency up to 72% during the deployment.

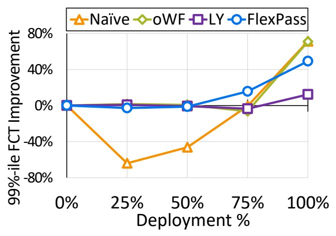
Impact of the selective dropping threshold. For different values of the selective dropping threshold, we show that there is a trade-off relationship between the 99%-ile FCT of small flows and the overall average FCT when FlexPass is fully deployed in Figure 17. A small selective dropping threshold enforces lower queue build-ups of reactive packets inside switches. What is more, this also reduces queue build-ups of proactive packets. This is because a lower threshold value



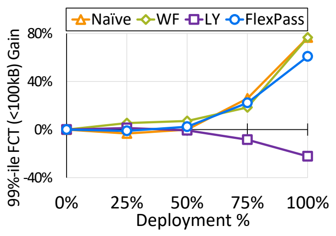
(a) Cache follower[41]



(b) Web search [2]

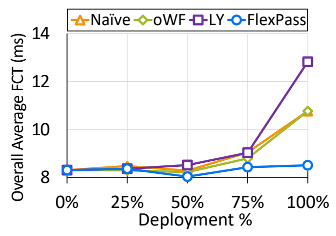


(c) Data mining [14]

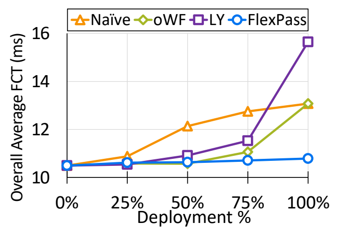


(d) Hadoop [41]

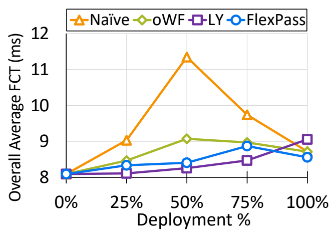
Figure 15. [Simulation] 99%-ile FCTs of small flows during the transition from DCTCP to ExpressPass.



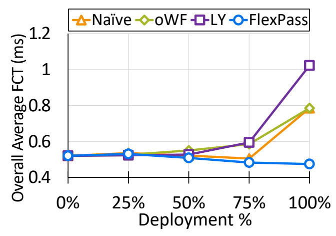
(a) Cache follower[41]



(b) Web search [2]



(c) Data mining [14]



(d) Hadoop [41]

Figure 16. [Simulation] Overall average FCTs during the transition from DCTCP to ExpressPass.

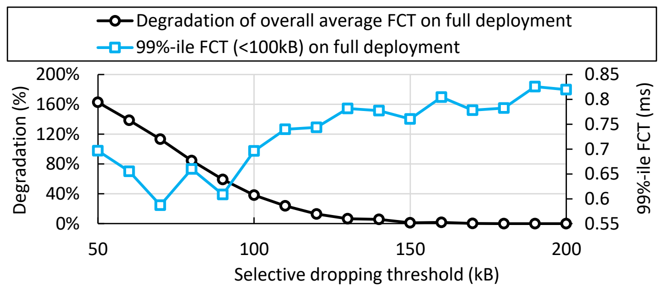


Figure 17. [Simulation] Trade-off between overall average FCT and 99%-ile FCTs of small flows as selective dropping threshold changes.

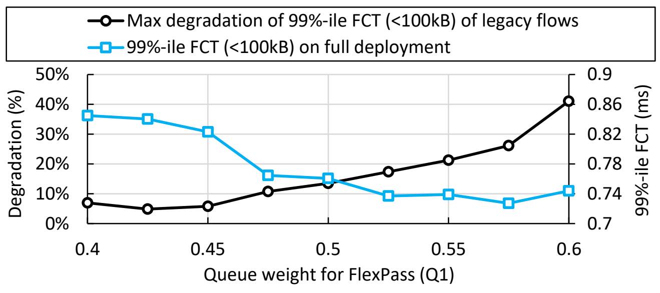


Figure 18. [Simulation] Trade-off between the degradation of legacy flows during deployment and 99%-ile FCTs when fully deployed as queue weight setting (w_q) changes.

reduces the RTT and the variance of RTT, and the lower variance of RTT leads to lower queue build-ups of proactive packets [9]. This results in better tail latency at full deployment. However, as a lower threshold means more packet drops, overall throughput gets worse, and average FCT increases, especially with the threshold smaller than the ECN marking threshold. In contrast, a high selective dropping threshold reduces the amount of proactively dropped reactive packets. This brings an increase in the overall throughput.

Impact of the queue weight setting (w_q). In previous evaluations, we equally set the queue weight for FlexPass queue (Q1) and legacy traffic queue (Q2) ($w_q = 0.5$). However, this queue weight setting may not be always optimal. Figure 18 shows the trade-off between the maximum degradation of the tail FCT of small legacy flows and the tail FCT of small flows on full deployment of FlexPass.

When w_q, the queue weight for FlexPass (Q1), is smaller than 0.5, legacy traffic gets more minimum guaranteed bandwidth and thus the maximum degradation of 99%-ile FCT of legacy flows is alleviated. However, when fully deployed, decreasing w_q reduces the amount of data transmitted using proactive sub-flow, This dilutes the high performance of proactive transport and results in higher tail FCTs. Note that ExpressPass is even more sensitive to this tradeoff – weight fair sharing of ExpressPass with mismatched queue weight setting results in a high penalty in tail latency of small flows.

Appendix B Artifact Appendix

B.1 Abstract

The artifact includes our implementation of FlexPass on ns-2 simulator. Our simulator demonstrates how FlexPass provides gradual deployability with the best incremental benefits. We provide an automated script to run all necessary simulations to reproduce the figures in the paper.

B.2 Description & Requirements

How to access. The source code of FlexPass simulator is available at https://github.com/kaist-ina/ns2-flexpass. We also provide a pre-built docker image at ghcr.io/kaist-ina/ns2-flexpass (recommended). The detailed instruction for the artifact evaluation is described at `README.md` in the repository.
Hardware dependencies. The evaluation does not require any special hardware. However, it is recommended to run evaluations on machines with many CPU cores. We tested our artifact on Intel Core i9-9900K (16 logical cores) with 32 GB of RAM.
Software dependencies. Generic Linux system (x86-64) with Docker installed. We tested our artifact on Ubuntu 20.04 LTS with Docker 20.10.7.
Benchmarks. All workloads and automation scripts are already included in the repository. A Tcl script located at /scripts/large-scale.tcl simulates 3-tier Clos 40Gbps topology with 192 hosts, and performs the simulation as described in §6.2. Also, we included a set of realistic workloads (search, cachefollower, etc.) in /workloads directory in the repository.

B.3 Set-up

We provide a pre-built Docker image for evaluation. To set up the Docker image, use the following command:

```
docker pull ghcr.io/kaist-ina/ns2-flexpass
```

B.4 Evaluation workflow

Major Claims.

- **(C1)**: When compared to other alternative deployment schemes, FlexPass achieves the best incremental benefits where FlexPass improves 99%-ile FCT and keeps the average FCT low, while posing minimal harm to legacy traffic. This is proven by Experiment (E1) and (E2) described in §6.2 whose results are illustrated in Figure 10 to Figure 13.
- **(C2)**: FlexPass improves the 99%-ile FCT of small flows after full deployment. This is proven by Experiment (E1) described in §6.2 whose results are illustrated in Figure 12.
- **(C3)**: Compared to an alternative deployment scheme, FlexPass does not show significant performance degradation during deployment, even at a very high load. This is proven by Experiment (E3) and (E4) described in §6.3 whose results are illustrated in Figure 14.

Experiments.

- **Experiment (E1)**: [5 human-minutes + 20-compute-cpu-hour] Evaluate FCTs (Flow Completion Times) of the background traffic during the transition from DCTCP to the new transport. Conduct in total of 20 Simulations by varying the deployment % (0%, 25%, 50%, 75%, and 100%) and the deployment scheme (naïve, oracle weighted fair queueing, layering, and FlexPass).
- **Experiment (E2)**: [5 human-minutes + 20-compute-cpu-hour] Conduct the same experiment with E1 but with mixed traffic (background + foreground traffic). 10% of the total traffic volume is set as foreground traffic.
- **Experiment (E3)**: [5 human-minutes + 60-compute-cpu-hour] Evaluate FlexPass and the naïve deployment scheme under different network loads. Conduct in total of 60 Simulations by varying the deployment % (0%, 25%, 50%, 75%, and 100%), link load (from 10% to 70%), and the deployment scheme (naïve, and FlexPass).

[Execution] We provide a script for automation (/run_simulations.py). This script will automatically run all the simulations above. As each simulation runs in a single thread, the given script automatically leverages multiple CPUs to parallelize simulations. Run the following command to run the script:

```
docker run --rm -it \
 -v $(pwd)/result:/ns-allinone-2.35/ns-2.35/outputs \
 ghcr.io/kaist-ina/ns2-flexpass \
 python ./run_simulations.py
```

The result will be stored in result directory. After the simulations, the result directory should contain fct_###.out files, where ### is an experiment ID designated at /run_simulations.py.
[Results] We also provide a script for analyzing obtained results and drawing figures from the results (/generate_figure.py). This script will parse the simulation result, analyze the result (compute 99%-ile FCT, average, etc.), and finally draw the figures (Figures 10 - 14). Run the following command to run the script:

```
docker run --rm -it \
 -v $(pwd)/result:/ns-allinone-2.35/ns-2.35/outputs \
 ghcr.io/kaist-ina/ns2-flexpass \
 python ./generate_figure.py
```

The CSV files and figures (in PNG) will be also stored in the result directory. After running the script, the result directory should contain fig#.png and fig#.csv, where # is the figure number.