Aeolus: A Building Block for Proactive Transport in Datacenter Networks

Shuihai Hu, Gaoxiong Zeng, Wei Bai, Zilong Wang, Baochen Qiao, Kai Chen, *Senior Member, IEEE*, Kun Tan, and Yi Wang, *Member, IEEE*

***Abstract*— As datacenter network bandwidth keeps growing, proactive transport becomes attractive, where bandwidth is *proactively* allocated as "credits" to senders who then can send "scheduled packets" at a right rate to ensure high link utilization, low latency, and zero packet loss. Consequently, proactive solutions such as ExpressPass, NDP, Homa, etc., have been proposed recently. While promising, a fundamental challenge is that proactive transport requires at least one-RTT for credits to be computed and delivered. In this paper, we show such one-RTT "pre-credit" phase could carry a substantial amount of flows at high link-speeds, but none of existing proactive solutions treats it appropriately. We present Aeolus, a solution focusing on "pre-credit" packet transmission as a building block for proactive transports. Aeolus contains unconventional design principles such as scheduled-packet-first (SPF) that de-prioritizes the first-RTT packets, instead of prioritizing them as prior work. It further exploits the preserved, deterministic nature of proactive transport as a means to recover lost first-RTT packets efficiently. Aeolus is compatible with all existing proactive solutions and readily implementable with commodity switches. We have integrated Aeolus into ExpressPass, NDP and Homa, and shown, via both implementation and simulations, that the Aeolus-enhanced solutions deliver significant performance or deployability advantages. For example, it improves the average FCT of ExpressPass by 56%, cuts the tail FCT of Homa by 20×, while achieving similar performance as NDP without switch modifications.**

***Index Terms*— Data center networks, proactive transport, first RTT, selective dropping.**

I. Introduction

WITH datacenter network link speed growing rapidly from 1/10G to 100G, more flows become "smaller" and can be finished (in theory) within a few RTTs (round trip time). Measurement of production workloads reveals that, ideally, 60-90% of the flows can be finished just in one RTT (§II-B). Therefore, it is crucial for transport to maintain low latency and high throughput at *every single RTT*.

Manuscript received August 25, 2020; revised February 5, 2021 and July 16, 2021; accepted September 20, 2021; approved by IEEE/ACM Transactions on Networking Editor H. Shen. This work was supported in part by the Hong Kong RGC under Grant GRF 16215119 and Grant TRS T41-603/20-R and in part by Huawei Research Grant. *(Corresponding author: Kai Chen.)*

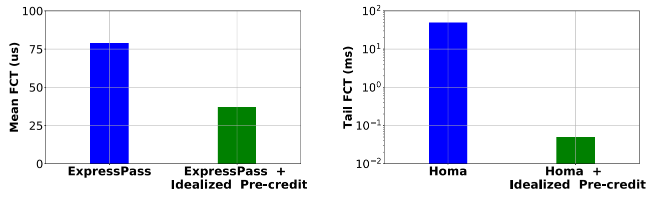
Shuihai Hu was with the iSING Lab, The Hong Kong University of Science and Technology, Hong Kong. He is now with Huawei Technology, Shenzhen 518129, China (e-mail: shuaa@connect.ust.hk).

Gaoxiong Zeng, Zilong Wang, Baochen Qiao, Kai Chen, and Yi Wang are with the iSING Lab, The Hong Kong University of Science and Technology, Hong Kong (e-mail: gzengaa@cse.ust.hk; zwangfb@cse.ust.hk; bqiao@cse.ust.hk; kaichen@cse.ust.hk; wangyihk@cse.ust.hk).

Wei Bai was with the iSING Lab, The Hong Kong University of Science and Technology, Hong Kong. He is now with the Microsoft Research Lab, Redmond, WA 98052 USA (e-mail: baiwei0427@gmail.com).

Kun Tan is with Huawei Technology, Shenzhen 518129, China (e-mail: kun.tan@huawei.com).

Digital Object Identifier 10.1109/TNET.2021.3119986



(a) Waiting credits in pre-credit phase (b) Blind burst in pre-credit phase

Fig. 1. A substantial gap between existing proactive transport baselines and the ideal performance. The setup is in §II. Aeolus provides a common building block of proactive transport to systematically bridge the performance gap caused by the pre-credit phase.

Traditional "try and backoff" transports (e.g., DCTCP [4], DCQCN [5], Timely [6]) are thus ill-suited to these requirements, as they only react to congestion signals (e.g., ECN or delay) "after the fact" and take multiple rounds to converge to the right rate. While they can maintain good average performance for long flows, it is hard to reach the right rate in each round, which is crucial for small flows and tail performance. Hence, a recent line of work (e.g. ExpressPass [1], NDP [2], Homa [3], FastPass [7], pHost [8]) explores a promising alternative, called *proactive transport*, in which link capacities are proactively allocated, by the receivers or a centralized controller, as credits to each active sender who then can send scheduled packets at an optimal rate to ensure high bandwidth utilization, low queueing delay, and zero packet loss.

Despite being promising, on a closer analysis, we found all existing proactive solutions fall short of achieving the best possible performance stated above. The key culprit is that, as they require at least one RTT to allocate credits to a new flow, the first RTT (the "pre-credit phase") poses a basic dilemma that compromises the performance of these solutions (Figure 1). If the sender sends no packet when waiting for credits (e.g. ExpressPass [1]), the new flows will be *paused* by one RTT even though the network is under-utilized (Figure 1(a)). If it bursts packets (e.g. Homa [3]), called unscheduled packets, at a high rate, it can cause sporadic traffic spikes, non-trivial queueing delay, and eventually packet losses of scheduled packets (Figure 1(b)). While there exists a potential solution that relies on special hardware support from switches to mitigate the consequence of packet losses (e.g. NDP [2]), it remains an open question whether the proactive transport's potential can be realized in a readily deployable way.

To address the problem, we observe that existing proactive transports can benefit from an *idealized* pre-credit solution that meets two seemingly contradicting principles:

- **Fully utilizing spare bandwidth:** new flows (with pre-credit unscheduled packets) should burst in the first RTT and strive to complete if they can.
- **Scheduled packet first (SPF):** scheduled packets should proceed as if no unscheduled packets are present.

As shown in Figure 1, this idealized pre-credit solution greatly improves the average FCT for ExpressPass and tail FCT for Homa, albeit for different reasons (see §II-C for details).

The insight behind the idealized pre-credit solution is that proactive transport is very susceptible to any delay or loss of scheduled packets. A slight delay of scheduled packets can cause temporary traffic spikes at downstream switches, which can break the delicate bandwidth allocation and affect more flows in a cascading style, eventually creating a perfect storm (§II-D). Moreover, these uncertainties cripple the proactive transport's unique performance predictability. In our experiment, we found that dropping one scheduled packet can increase flow completion time by up to $100\times$ due to the retransmission timeout. These problems can be further exacerbated by the bursts of many short flows comprising mostly of unscheduled packets.

To summarize, the deterministic nature of proactive transport means any drop or delay of scheduled packets could inflict a disproportional damage. As a solution, the idealized pre-credit scheme can effectively avoid the pitfalls in recent proactive solutions (e.g. [2], [3], [8]) by safeguarding the scheduled packets and de-prioritizing the unscheduled packets, as opposed to the other way around.

The key contribution of this work is to make the above idealized pre-credit solution practical. We present Aeolus,[1] a readily deployable building block for proactive transport that meets the above two principles of scheduled packet first and fully utilizing spare bandwidth with unscheduled packets simultaneously. With Aeolus, the performance of small flows can be significantly improved at low network load due to the full utilization of spare bandwidth in the first RTT. In the meanwhile, by protecting the deterministic nature of proactive schemes, Aeolus avoids the dropping of scheduled packets even at high network load and thus guarantees good tail latency.

Aeolus realizes its design goal by proposing a novel selective dropping mechanism (§III-B) which allows pre-credit new flows to burst at line-rate when there exists spare bandwidth left over by scheduled packets, but immediately drops them selectively once the bandwidth is used up. In this way, Aeolus effectively utilizes available bandwidth with unscheduled packets while safeguarding the scheduled packets, thus achieving the above two principles simultaneously. In particular, we show that our selective dropping is readily implemented with *only one queue* at commodity switches by using the Active Queue Management (AQM) feature (§IV-A).

Furthermore, it is worthwhile to note that since we have protected the scheduled packets, as a reward, our loss recovery of unscheduled packets can be designed much simpler yet efficient. The idea is to reuse the preserved proactive transport as a reliable means to recover dropped pre-credit packets—any dropped unscheduled packet will become a scheduled packet in the next round, whose delivery is guaranteed. Therefore, we just need to locate packet losses in the first RTT and then retransmit them once using scheduled packets (§III-C).

[1]The earlier idea of Aeolus was presented in [9], [10].

Aeolus is architecturally compatible with all existing proactive solutions. We implemented an Aeolus prototype using DPDK [11] and commodity switch hardware (§IV-B), and integrated it with the latest proactive solutions such as ExpressPass [1], Homa [3], and NDP [2]. We further built a small testbed with one Mellanox SN2000 switch and eight servers at 10Gbps (§V-A), together with larger-scale trace-driven simulations at 100Gbps, to evaluate the performance of Aeolus. We find that:

- **Aeolus + ExpressPass** reduces the FCT by up to 33% on average at 10G testbed experiments, while achieving 56% improvement in large-scale 100G simulations. This is because Aeolus fully utilizes the spare bandwidth with pre-credit unscheduled packets in the first RTT which has not been used in ExpressPass.
- **Aeolus + Homa** reduces the tail FCT of small flows by $20\times$, from 100s of ms to a few ms in 10G testbed experiments, while achieving $190\times$ improvement in simulations. This is because Aeolus effectively eliminates losses of scheduled packets caused by the burst of unscheduled packets, by enforcing the scheduled packet first principle.
- **Aeolus + NDP** achieves similar performance as NDP, but without requiring switch modifications. This is because similar to the cutting payload technique [12] adopted by NDP, Aeolus can eliminate large queue buildup by selectively dropping excessive unscheduled packets at the switch, while ensuring fast loss recovery by reusing the preserved deterministic nature of proactive transport.

II. Background and Motivation

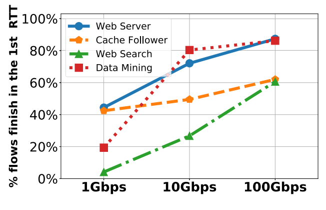
A. Proactive Datacenter Transport

Datacenter congestion control traditionally (e.g. [4]–[6]) uses a "try and backoff" approach and is thus largely reactive to congestion. To meet increasing performance requirements, many recent works are based on proactive transport, which operates in a "request and allocation" style. The key conceptual idea behind proactive transport is to explicitly allocate the bandwidth of bottleneck link(s) among active flows and proactively prevent congestion. As a result, the switch will have ultra-low buffer occupancy and (near) zero packet loss. Central to proactive transport's superior performance is the perfect credit allocation to active flows, so any new sender needs one RTT, which we call *pre-credit phase*, to inform the receiver/controller to assign the credits.

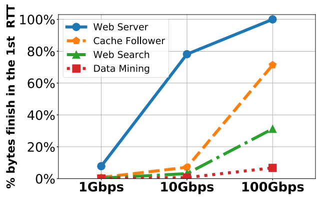
There have been several implementations of the concept of proactive transport. Fastpass [7] employs a centralized arbiter to enforce a tight control over packet transmission time and path. PDQ [13] and TFC [14] leverage switches to explicitly allocate link bandwidth among the passing flows. ExpressPass [1], pHost [8], NDP [2] and Homa [3] use receiver-driven credit-based approaches to explicitly schedule the arrival of data packets destined for different receivers.

B. The Pre-Credit Phase (1st RTT) Matters

The rapid growth of DCN link speeds (from 1/10G to 100G) has fundamentally changed the flow characteristics, in particular, an explosion number of the flows can complete in the first RTT. Figure 2 shows the fraction of flows (and bytes) could have been finished within the first RTT (pre-credit phase) under different link speeds. Flows are generated according

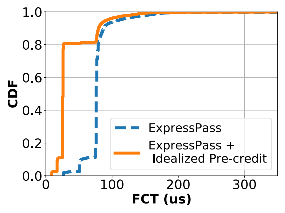


(a)

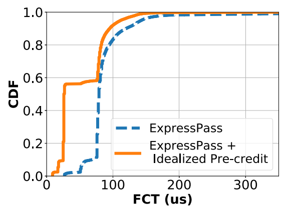


(b)

Fig. 2. A substantial fraction of flows (and bytes) could have been finished within the first RTT (pre-credit phase), and this fraction grows rapidly as link speed increases. (For calculation simplicity, we assume the level of flow multiplexing is 1.)

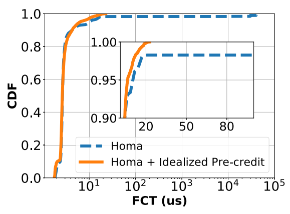


(a) Cache Follower

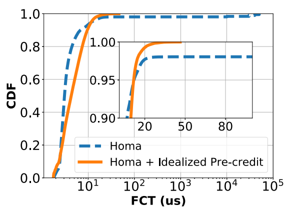


(b) Web Server

Fig. 3. FCT of 0-100KB flows under the original ExpressPass and the hypothetical ExpressPass with idealized pre-credit solution (fully utilizes the spare bandwidth in the first RTT).



(a) Cache Follower



(b) Web Server

Fig. 4. FCT of 0-100KB flows under the original Homa and the hypothetical Homa with idealized pre-credit solution (no interference between scheduled and unscheduled packets).

to four realistic workloads including Web Server [15], Cache Follower [15] Web Search [4] and Data Mining [16].

For Figure 2(a), we calculate FCTs of flows by simply dividing flow size by the given link speed. For Figure 2(b), we calculate the expected average flow size of a given workload (denoted as A), and the number of bytes a given link speed can transmit in one RTT (denoted as B). We simply use B/A as the fraction of bytes could have been finished within the first RTT. Although admittedly, this methodology is greatly idealized, it suggests a clear trend that the rise of high-speed DCNs have dramatically shifted the distributions of flow completion time, with many flows, in theory, being able to complete in the first RTT.

In the light of existing proactive transport designs, the fact that more flows can complete in the first RTT has several important implications:

- *Many flows will benefit from sending data immediately after they arrive*, as opposed to waiting for credits (as in [1], [7]). This coincides with the ethos of recent proactive transport designs [2], [3], [8].
- *There will be more spare bandwidth.* This creates more potential benefits and motivation to send (unscheduled) packets in a speculative fashion to take advantage of the spare capacity.
- *More packets will be first-RTT packets.* This means more frequent contention between unscheduled packets (sent in the pre-credit phase) and scheduled packets (sent with credits in all subsequent RTTs), which potentially undermines the gains of unscheduled packets.

In short, this short analysis indicates existing proactive transport designs demand an effective pre-credit solution to fully utilize spare bandwidth in the first RTT as the link speed significantly increases.

C. Performance Issues of Exiting Solutions

Through an empirical analysis on the representative proactive transport solutions, we demonstrate a key tradeoff in how they handle the first RTT (i.e., the pre-credit phase).

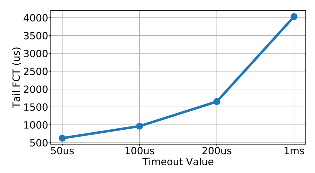
Why not wasting the pre-credit phase? On the one hand, if the sender holds during the pre-credit phase, it can deal a heavy blow on short messages, which could have been completed in the first RTT. To concretely show its impact on performance, we chose ExpressPass [1], the most recent proactive transport proposal that sends only scheduled packets after the pre-credit phase (although it uses probe packets, but they do not carry actual data). We ran an ns-2 simulation with a fat-tree topology of 8 spine switches, 16 leaf switches, 32 top-of-rack (ToR) switches and 192 servers connected via 100Gbps links (the same topology as used in ExpressPass [1]). Flows are generated according to two realistic workloads, including Cache Follower [15] and Web Server [15].

In addition, to show the potential performance benefit of *not* hold in the first RTT, we considered a hypothetical ExpressPass, which leverages an idealized pre-credit solution to send just enough data to fully utilize the spare bandwidth in the first RTT (i.e. with hindsight knowledge), and follows ExpressPass after the first RTT.

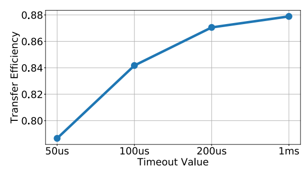
Figure 3 shows the FCT of small flows (0-100KB) under the original ExpressPass and the hypothetical ExpressPass. Across the two workloads, we can see that $57-80\%$ of small flows take one extra RTT to complete in ExpressPass than necessary, i.e. an almost $3\times$ inflation (from 0.5 to 1.5RTT)!

Why not bursting in the pre-credit phase? On the other hand, if each new sender sends data speculatively before credits are allocated, it could increase the network load unpredictably and break the delicate credit allocation, crippling the desirable properties of proactive transport. To demonstrate this problem, we chose Homa [3], a recent proactive transport variant that lets new flows blindly transmit unscheduled packets in the first RTT. We ran a simulation with Homa's OMNet++ simulator [17] with a two-tier tree topology of 8 spine switches, 8 leaf switches and 64 servers (8 servers per leaf switch) connected by 100Gbps links (the same topology as used in Homa [3]). The retransmission timeout of Homa is set to be 10ms. Each switch has a per-port buffer of 200KB. Flows are generated according to Figure 3. To understand the impact of the interference between scheduled and unscheduled packets, we consider a hypothetical Homa, which knows the exact amount of spare bandwidth on each link in the first RTT, with hindsight knowledge. This way, the hypothetical Homa ensures the scheduled packets will always have enough bandwidth to transmit and will not be queued or dropped.

Figure 4 compares the FCT distribution of the original Homa and the hypothetical Homa with the idealized first RTT. We can see that, although most flows complete very quickly ($<30\mu$s), the tail FCT can be excessively bad, with 99.9th percentile exceeding 50 milliseconds in both workloads. We found that the tails are due to buffer buildups and subsequent packet drops caused by senders bursting too many



(a) Tail FCT of 0-100KB flows



(b) Transfer efficiency

Fig. 5. Tail FCT and transfer efficiency of Homa over varying retransmission timeouts.

unscheduled packets in the first RTT. Worse still, as scheduled packets are no longer lossless, the retransmitted packets may also get lost. In contrast, the tail FCTs of the hypothetical Homa are dramatically improved—99.9th percentiles are less than 50 μs, a nearly 1000$\times$ reduction.

Readers may wonder, can Homa significantly reduce its tail FCT by adopting a much more aggressive loss recovery? To study this, we vary the retransmission timeout from 50μs to 1ms and measure the corresponding results. The simulation results with Cache Follower workload are summarized in Figure 5 (we omit similar results under other workloads for space).

As we can see, while Homa does achieve a much better tail FCT for small flows with a smaller retransmission timeout (e.g., 50μs), the cost is also expensive—a smaller timeout results in lower transfer efficiency[2] ($\sim$10% downgrade). This is mainly because an aggressive loss recovery will easily trigger pre-mature packet retransmission. Then many packets are duplicately transmitted several times, wasting scarce bandwidth that could have been used productively. As a result, the transfer efficiency becomes lower, which will prolong the completion of the majority of flows.

In conclusion, due to the breaking of the delicate credit allocation for scheduled packets, Homa faces a dilemma in handling the delayed/dropped scheduled packets.

D. Summary: Protect Scheduled Packets?

The above microbenchmark shows that neither approaches to proactive transport is ideal—wasting the first RTT leads to longer-than-necessary FCTs in normal cases (Figure 3), while bursting with unscheduled packets leads to excessively long tail FCTs (Figure 4) or worse transfer efficiency (Figure 5). Meanwhile, it indicates that *both* solutions can greatly benefit from *an ideal solution to the first RTT (i.e. pre-credit phase)*. In particular, such an idealized first-RTT solution should achieve two seemingly conflicting objectives: (1) fully utilize the spare bandwidth with unscheduled packets, and (2) not interfere with scheduled packets.

Before diving into our design, we pause briefly and put these two goals into the perspective of existing solutions (Homa, pHost, NDP) that send unscheduled packets in the pre-credit phase. While they all aim to fully utilize the bandwidth in the first RTT with unscheduled packets, they fundamentally differ from us in how unscheduled should share bandwidth with scheduled packets. Homa and pHost prioritize unscheduled packets over scheduled ones, while NDP does not discriminate and let them share bandwidth fairly. Thus, all of them might delay (or cut the payload of) scheduled packets, potentially leading to the tail latency shown in Figure 4 or the downgrade of transfer efficiency shown in Figure 5.

To see a concrete example, let us consider Figure 6. Each link is fully scheduled to transmit scheduled packets when a

[2]Defined as total bytes received over total bytes sent.

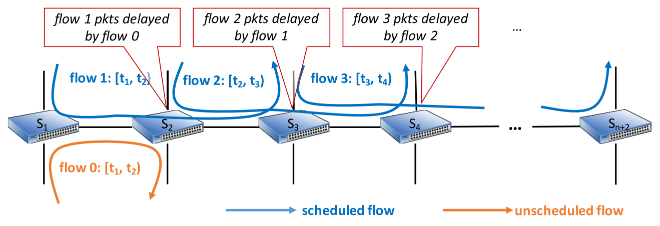


Fig. 6. An illustrative example of a flow of unscheduled packets causing delays on many scheduled flows at downstream switches in a cascading manner.

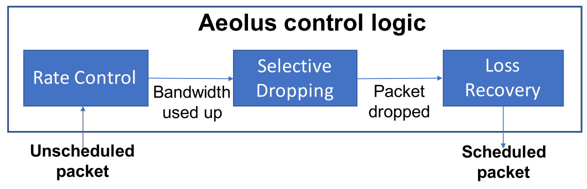


Fig. 7. Overview of Aeolus.

flow of unscheduled packets arrives. Because the scheduled flows have equal or lower priority, flow 1 will be delayed, which then delays flow 2 on the next link, and then flow 3, and so on. Note such cascading delaying of scheduled flows can even propagate to switches where unscheduled packets are not present. Even worse, such delaying can increase the chance of packet losses in proactive transport as the queues can no longer absorb occasional bursts.

In short, the cost of delaying/dropping scheduled packets suggests one should revisit the tenet of prioritizing unscheduled packets, even though they are from short flows.

III. Aeolus

Aeolus aims to achieve three design goals simultaneously: (1) new flows fully utilize spare bandwidth and strive to complete if they can, avoiding longer-than-necessary FCTs; (2) safeguarding the scheduled packets to preserve the deterministic nature of proactive transport; and (3) to make it easy to deploy in production datacenters, i.e., Aeolus must be implementable with commodity switches.

Figure 7 overviews Aeolus, which mainly contains 3 components: rate control, selective dropping and loss recovery.

- **Rate control (§III-A):** Aeolus adopts a minimal rate control at the end hosts: all flows start at line-rate at the pre-credit stage and then adjust their sending rates according to received credits later on.
- **Selective dropping (§III-B):** The key for Aeolus to safeguard scheduled packets is to enforce the scheduled packet first (SPF) principle in the network. To do that, Aeolus introduces a novel selective dropping mechanism at the switch, which selectively drops unscheduled packets when the bandwidth is just used up, without affecting scheduled packets. With such a scheme, Aeolus can effectively utilize leftover bandwidth for unscheduled packets without crippling the desirable properties of proactive transport. In addition, our selective dropping mechanism is readily deployable using commodity switches (§IV).
- **Loss recovery (§III-C):** Given Aeolus has safeguarded the scheduled packets, loss recovery is needed only for lost unscheduled packets. For recovery, we exploit the well-protected proactive transmission as a safe and efficient

means for loss re-transmission — we come up with a loss detection mechanism that can accurately locate unscheduled packet losses in the pre-credit phase, and retransmit them as credit-induced scheduled packets *only once*.

A. Rate Control

Ideally, the flow's sending rate in the pre-credit phase should be determined by the spare bandwidth left by scheduled packets, which keeps varying across time and space. Since it is almost impossible to calculate such dynamic spare bandwidth, we leave it to the switch to implement the desired bandwidth allocation (i.e., SPF in §III-B) between scheduled packets and unscheduled packets. As a result, the need for rate control at the end hosts becomes minimal. In particular, we do not require sophisticated rate control to prevent either queue buildups or spurious packet drops due to traffic bursts. This is because queue buildup can be eliminated by selective dropping (§III-B) while any packet loss in the pre-credit stage can also be recovered shortly in the upcoming credit-based stage through scheduled packets (§III-C).

With the above thought, our minimal rate control mechanism is designed to work simply as follows:

- **Pre-credit line-rate burst:** A flow sender enters the pre-credit state on its initiation and sends a bandwidth-delay product BDP) worth of unscheduled packets at line-rate. We use such an aggressive rate to fully utilize any spare bandwidth when it presents in the network.
- **Credit-based rate control:** Along with the unscheduled packet bursting, the sender also sends a request to the receiver or central arbitrator to seek credits. Once the credit returns, it will exit the pre-credit state *immediately* even it has not yet sent out all unscheduled packets. After that, the sender enters the credit-induced state and transmits scheduled packets according to the assigned credits. We design Aeolus to be compatible with all existing credit-based rate control algorithms [2], [7], [8].

B. Selective Dropping

As Aeolus imposes nearly no rate control on unscheduled packets at the end host, it should safeguard scheduled packets in the network. To ensure unscheduled packets only utilize the spare bandwidth leftover by scheduled packets, Aeolus enforces SPF by prioritizing scheduled packets over unscheduled packets at the switch. A conventional way to realize this is through priority queueing [18]–[20]. However, we identify a few problems of directly using priority queues in our design of Aeolus. Instead, we implement a novel selective dropping scheme by re-interpreting RED/ECN feature of commodity switches in an unconventional way.

Why not priority queueing? We choose not to use priority queueing for three reasons. First, it creates ambiguity: when the receiver has been waiting for an unscheduled packet for a long time, it is hard to decide whether this packet has been dropped or is still being trapped in the network. This is because, with priority queueing, subsequent scheduled packets in the high priority may arrive earlier than unscheduled packets in the low priority. Such ambiguity introduces a similar dilemma faced by Homa (as discussed in §II-C). If we use a conservative loss recovery approach (e.g., a large RTO), we may prolong tail latency for lost packets. If we use an aggressive approach (e.g., a small RTO), we may incur unnecessary retransmissions for trapped packets, downgrading the transfer efficiency. We showcase this problem numerically in §V-E. Second, unscheduled packets in the low priority may still occupy considerable buffer that risks affecting scheduled packets (showcase in §V-E), due to the reason that proactive solutions require certain buffer space to accommodate imperfect network conditions such as transient queue buildups caused by RTT variations [1]. Third, commodity switches have a smaller number of queues (typically 8), which may be used for other purposes such as isolating traffic of different services [21]. We do not want to consume additional queue resources by presenting Aeolus.

Selective dropping: We seek to implement SPF while avoiding the downsides of priority queueing. To avoid ambiguity and save queue resources, we prefer a mechanism that uses only one queue and keeps in-order packet transmissions. Furthermore, to reserve sufficient buffer headroom to hold scheduled packets, we should limit the buffer space used by unscheduled packets.

According to this insight, we transmit all the data packets in a FIFO queue (unless special requirement of the transport) and enforce a selective dropping mechanism at the switch: when an unscheduled packet arrives, the switch drops it if the buffer occupancy exceeds a very small threshold (e.g., 2-8KB), but such dropping does not apply to scheduled packets. In this way, Aeolus achieves multiple benefits simultaneously—it avoids ambiguity with just one queue in-order transmission, prioritizes scheduled packets through proactively dropping unscheduled packets once queue builds up, while still allowing unscheduled packets to fully utilize any leftover bandwidth with minimal buffer occupancy. One contribution of this paper is that we show such selective dropping is effective yet very easy to implement using the Active Queue Management (AQM) feature at commodity switches. In §IV-A, we introduce two implementation options using Weighted Random Early Detection (WRED) and RED/ECN, respectively. In our testbed experiments, we adopt RED/ECN to realize the proposed selective dropping.

C. Loss Recovery

In Aeolus, scheduled packets are less likely to be dropped as we well protect them. However, unscheduled packets can be dropped under selective dropping. Hence, a fast loss recovery of unscheduled packets is needed. Given we have safeguarded scheduled packets, our idea is to retransmit lost unscheduled packets using subsequent scheduled packets, whose deliveries are guaranteed by the property of proactive transport. Therefore, loss recovery simply reduces to loss detection in Aeolus.

Loss detection: Aeolus enables per packet ACK at the receiver to quickly notify the sender of arrival unscheduled packets. We use selective ACK rather than cumulative ACK for loss detection in the middle, and leverage a simple probing to detect tail losses of unscheduled packets. Specifically, the Aeolus sender transmits a probe packet *right after* the last unscheduled packet. This probe packet carries the sequence number of the last unscheduled packet, and is of minimum Ethernet packet size, e.g., 64 bytes. When the receiver receives the probe packet, it returns an ACK carrying the sequence number of the probe packet. Once the sender receives such a probe ACK, it can immediately infer all the losses of unscheduled packets, including the last one. Finally, it is worthwhile to note that to guarantee the delivery of the probe packet and all ACKs, we treat them as scheduled in the network.

Retransmission: As introduced above, Aeolus retransmits lost unscheduled packets using subsequent scheduled packets. Upon receiving credits, the sender can retransmit old packets or transmit new packets. Specifically, the sender has three types of packets to transmit: sent but unacknowledged unscheduled packets, loss-detected unscheduled packets, and unsent scheduled packets. We prioritize them in the order of loss-detected unscheduled packets, unsent scheduled packets, and sent but unacknowledged unscheduled packets, respectively. We give the highest priority to loss-detected unscheduled packets because we want to fill the gap as soon as possible to minimize the memory footprint of resequence buffer. We prioritize unsent scheduled packets over sent but unacknowledged unscheduled packets to avoid redundant retransmissions.

D. Why Does This Work?

The key of Aeolus is its simple yet effective selective dropping mechanism, which not only delivers good performance but also significantly simplifies both rate control and loss recovery designs. With selective dropping, new flows can start at line-rate to fully utilize spare bandwidth without affecting scheduled packets. For pre-credit unscheduled packets, the cooperation of line-rate start and selective dropping maximizes their potential benefits (e.g., utilize the spare bandwidth) and minimizes their side effects (e.g., affect scheduled packets) simultaneously. Furthermore, by selective dropping, Aeolus only drops unscheduled packets. Therefore, loss recovery becomes relatively simple because packet losses only happen in the pre-credit stage (or first batch) and the deliveries of subsequent scheduled packets are guaranteed. We just need to locate the losses in the first batch and then efficiently retransmit them only *once* using scheduled packets. We do not need sophisticated schemes to handle many challenging corner cases, e.g., packet loss of retransmission packets. Compared to TCP variants that have complex loss recovery mechanisms for different scenarios, Aeolus's loss recovery is extremely simple but more efficient.

E. How Does This Work?

Integrating Aeolus with existing proactive mechanisms is logically simple—we only need to add a separate control loop for transmitting unscheduled packets in the first RTT, which runs in parallel with the original control loop that schedules the transmission of scheduled packets.

It is true that existing proactive mechanisms do not share much similarity among each other. However, their behaviours in the first RTT are identical, i.e., either forbid data transmission or transmit a fixed bytes of unscheduled packet. The main modification needed by Aeolus to enable data transmission in the first RTT, and assign lower network priority to unscheduled packets. We believe such a modification is not significant for existing proactive mechanisms.

IV. Implementation

A. Switch Implementation

The Aeolus switch *selectively* drops unscheduled packets while preserving scheduled packets in one switch queue. Here we propose two implementation options to realize this.

WRED: Weighted random early detection (WRED) is an extension to random early detection (RED) where a single queue has several *different* sets of queue dropping/marking thresholds. WRED typically supports three packet colors (which is a metadata attached to the packet in switch processing pipeline): red, yellow and green, and each color has its own dropping thresholds in a switch queue. WRED is widely supported by commodity switching chips [22], [23].

To implement selective dropping using WRED, we mark scheduled and unscheduled packets with different DSCP values at the end host. At the switch, we configure access control list (ACL) table to set the arriving packet's color based on its DSCP field. Therefore, scheduled and unscheduled packets can be marked with different colors in the switch pipeline. For unscheduled packets, we can set both the high and low dropping thresholds to the desired selective dropping threshold. For scheduled packets, we can set its high/low dropping threshold to a very large value (e.g. total buffer size) so that scheduled packets will not be dropped by WRED.

RED/ECN: Though WRED is widely supported by switching chips, it may not be *exposed* by all switch OSes to users. Some switch OSes (e.g., Arista EOS [24]) just provide a simple RED/ECN configuration interface where a switch queue only has a single set of dropping/marking thresholds.

Now, we show how to realize selective dropping only using RED/ECN feature exposed to users. ECN mechanism uses the 2-bit ECN field in the IP header to encode whether a packet is ECN capable or has experienced congestion. When both endpoints support ECN, they will mark their data packets with 10 (ECN capable transport, ECT(0)) or 01 (ECT(1)). Otherwise, packets will be marked with 00 (Non-ECT). At the switch, when the buffer occupancy is larger than the ECN marking threshold, the arriving packet will be marked (changed the code point to 11) if it is ECN capable, otherwise get dropped. This mechanism has been well studied in previous work [25], [26].

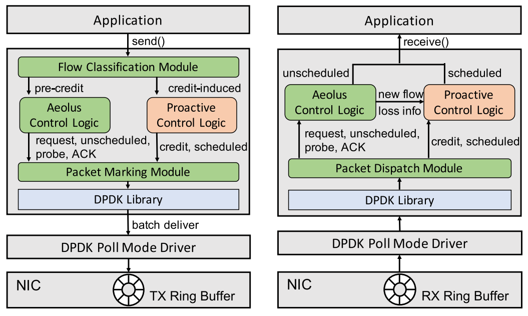
Therefore, we can implement the selective dropping by reinterpreting the RED/ECN as follows. At the sender side, we set the ECN fields of unscheduled packets and scheduled packets to Non-ECT and ECT(0), respectively. At the switch, we enable ECN marking and configure both the high and low RED thresholds to the selective dropping threshold. In this way, any unscheduled packets exceeding this threshold will be selectively dropped by switch. At the receiver side, we simply ignore the ECN marks of the arriving packets.

B. Host Implementation

To evaluate the benefits of Aeolus to augment proactive solutions, we have implemented a prototype of Aeolus with two recent proactive transports, ExpressPass [1] and Homa [3]. Our implementation is based on DPDK 18.05 [11], which allows the network stack to bypass the kernel and communicate directly with the NIC.

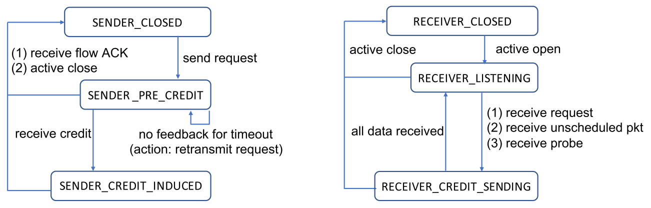
As shown in Figure 8, the main modification of Aeolus on top of existing proactive transports is to add an Aeolus control logic, a flow classification module, a packet marking module and a packet dispatch module. As our implementation does not touch the core code of proactive transports, different proactive protocols can be "swapped out" easily while still remaining compatibility with Aeolus.

Packet sending pipeline: As shown in Figure 8(a), application starts data transmission by calling a send() function.



(a) Packet sending pipeline (b) Packet receiving pipeline

Fig. 8. Aeolus software implementation architecture on top of proactive solutions.



(a) Aeolus sender (b) Aeolus receiver

Fig. 9. State transition of sender and receiver.

The flow classification module tracks the per-flow state using a table. Each flow is identified using the 5-tuple (i.e., source/destination IPs, source/destination ports and protocol ID), and initially classified as pre-credit flow. The flow enters credit-induced state once it finishes its first-RTT packet transmission. Pre-credit flows and credit-induced flows are processed by the Aeolus control logic and the proactive control logic separately.

The Aeolus control logic checks sender buffers of its belonging flows iteratively in a round robin fashion, reading in the data, segmenting the data into unscheduled packets, constructing request, probe and ACK, and forwarding these packets to the next-stage processing module. As for the proactive control logic, it follows its original processing logic without any modification.

We note that in the design of some proactive solutions like Homa, the receiver needs to learn about the flow size information from the header of successfully received unscheduled packet(s). We encode the flow size information in the header of probe such that the receiver can still learn about the flow demand even when all the unscheduled packets get dropped.

The packet marking module marks outgoing packets. It sets ECN fields of unscheduled packets and scheduled packets to 00 (Non-ECT), and 10 (ECT(0)), respectively. To increase throughput, the marked packets are sent to the TX ring buffer of NIC in batch via DPDK. We choose a batch size of 15 packets in our current implementation.

Packet receiving pipeline: We leverage DPDK poll model driver to periodically poll the RX Ring buffer of NIC. Once a batch of packets are received, the packet dispatch module will distribute them to the corresponding control logic.

The Aeolus control logic mainly performs three operations on receipt of a packet: (1) notify the flow classification module to change the state of a flow in case an ACK of a flow is received for the first time; (2) notify proactive control logic the arrival of a new flow when a request is received; (3) do loss detection based on received ACKs and notify the proactive control logic to perform loss retransmission with scheduled packets.

TABLE I

FLOW SIZE DISTRIBUTIONS OF REALISTIC WORKLOADS

	Web Server [15]	Cache Follower [15]	Web Search [4]	Data Mining [16]
0-100KB	81%	53%	52%	83%
100KB-1MB	19%	18%	18%	8%
$>$ 1MB	0%	29%	20%	9%
Average size	64KB	701KB	1.6MB	7.41MB

State transition: Figure 9 depicts the state transition of sender and receiver. Aeolus sender will progress to state SENDER_PRE_CREDIT after it sends a request to the receiver. The sender will remain in this state, transmitting unscheduled packets and probe until it receives credit from the receiver. After transiting into state SENDER_CREDIT_INDUCED, the sender will focus on the task to transmit scheduled packet(s) on the receipt of credit until all the data are received by the receiver.

Aeolus receiver will enter state RECEIVER_LISTENING when the application layer issues an "active open" operation. The receiver further progresses into state RECEIVER_CREDIT_SENDING after it receives a request or an unscheduled packet or a probe. The receiver will allocate credit to the sender to pull the remaining data to be received. A receiver will go back to state RECEIVER_LISTENING when all the data of this flow/message are received.

V. EVALUATION

We evaluate Aeolus using a combination of large-scale simulations and testbed experiments. The key findings are:

- **Aeolus improves the normal-case FCT of ExpressPass [1]**, with the mean FCT reduced by up to 56%.
- **Aeolus improves the tail FCT of Homa [3]**, with the 99th percentile FCT reduced by up to 190×.
- **Aeolus preserves the superior performance of NDP [2] without requiring switch modifications.**

A. Evaluation Setup

Choices of baseline proactive transport: We choose three recent proactive transport solutions: ExpressPass [1], Homa [3] and NDP [2], to represent different design choices of proactive transport. ExpressPass forbids data transmissions in the first RTT, and Homa and NDP blindly send unscheduled packets in the first RTT.

Testbed: We built a small testbed that consists of 8 servers connected to a Mellanox SN2000 switch using 10Gbps links. Our switch supports ECN and strict priority queueing with 8 queues. Each server is equipped with an Intel 82599EB 10GbE NIC that supports DPDK. We enable RED/ECN at the switch to implement selective dropping. The base RTT is around 14us. For ExpressPass [1], the configuration is simple as we only have a single queue to transmit data packets, including scheduled packets and unscheduled packets. In contrast, Homa [3] uses multiple priority queues to serve unscheduled and scheduled packets separately. As a result, configuring per queue selective dropping at switches can no longer protect scheduled packets from the impact of unscheduled packets. For Homa, we configure per-port ECN/RED [21] (see more detail in §VI).

Simulator: For all the three schemes, we use the simulators provided by the authors with their recommended configuration options. For ExpressPass, we implemented Aeolus on top of ExpressPass's open source code [27] with ns-2 simulator. For Homa, we implemented Aeolus on top of Homa's open source code [17] with OMNeT++ simulator. Homa assumes infinite switch buffer in its simulations, and its simulator lacks loss recovery mechanism. Hence, we extended Homa's simulator to implement a timeout-based loss recovery mechanism according to the description in Homa paper. For NDP, we implemented Aeolus on top of NDP's packet-level htsim simulator [28].

Default configuration: Unless stated otherwise, our evaluation uses a default configuration that is based on a network load of 0.4 and a per-port buffer of 200KB at switches. By default, we set the selective dropping threshold to be 6KB (4 packets). The MTU is set to be 1.5KB (as NDP [2] by default uses 9KB jumbo packet, we set MTU to be 9KB for NDP). For ExpressPass, we set the initial credit sending rate to be 1/16 of link capacity and the aggressiveness factor ω to be 1/16. For Homa, we use 8 priority queues at switches and set the overcommitment degree to 6. The retransmission timeout is set to 1ms/100μs in different experiments. For NDP, the threshold of packet trimming (payload cut) is set to 8 packets (72KB).

Workload: We generate realistic workloads according to 4 production traces including Web Server [15], Cache Follower [15], Web Search [4] and Data Mining [16]. Their flow size distributions are shown in Table I. All the distributions are highly-skewed: the most of bytes are from few large flows. We generate flows using a Poisson arrival process to achieve a specified network load. For every flow, the sender and the receiver are randomly chosen.

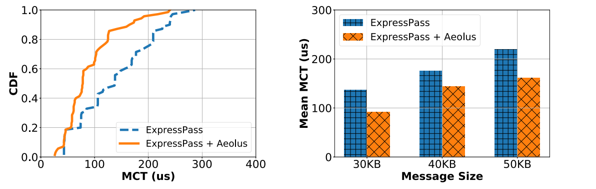
Experiment/simulation setup: We conduct 7-to-1 incast experiments in our testbed as follows: one client node sends requests to other 7 servers simultaneously and each server responds with messages of fixed size. We vary the size of the response message from 30KB to 50KB, and measure the message completion times (MCT).

In large-scale simulations, we use the same network topologies as the ones adopted by the papers of compared schemes. For ExpressPass, we simulate an oversubscribed fat-tree topology with 8 spine switches, 16 leaf switches, 32 top-of-rack (ToR) switches and 192 servers. We set network link delay to 4μs, and host delay to 1μs, which gives a maximum base RTT of 52μs. For Homa and NDP, we simulate a two-tier tree with 8 spine switches, 8 leaf switches and 64 servers. The base RTT is set to 4.5μs. For all the simulated topologies, all the links have 100Gbps capacity.

B. ExpressPass + Aeolus

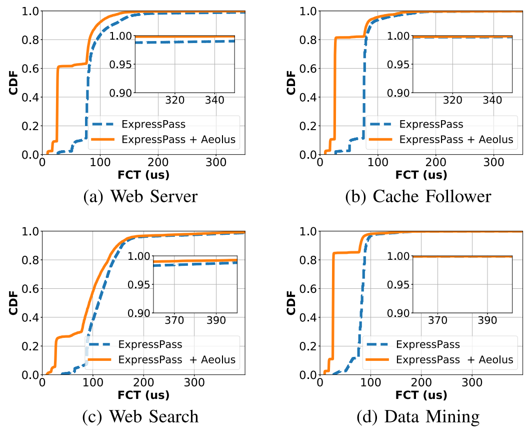
With testbed experiments and simulations, we show that Aeolus can help ExpressPass significantly speed up small flows by fully utilizing spare bandwidth in the first RTT, while keeping the queue occupancy small.

Testbed experiments: Figure 10 shows the message completion times (MCT) of 7-to-1 incast scenario when message size varies from 30KB to 50KB. The results indicate that Aeolus can assist ExpressPass to speed up small flows even under stressed incast traffic pattern: median MCT is improved by 43% with 30KB message size (Figure 10(a)), and average



(a) MCT with 30KB message size (b) Mean MCT with different message sizes

Fig. 10. Message completion times (MCT) of 7-to-1 incast. The message size varies from 30KB to 50KB.



(a) Web Server (b) Cache Follower

(c) Web Search (d) Data Mining

Fig. 11. FCT of 0-100KB flows in an oversubscribed fat-tree topology. The average load of the network core is 40%.

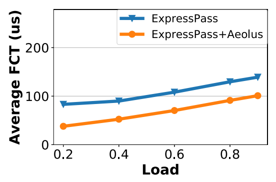
MCT is improved by 19%-33% across different message sizes (Figure 10(b)).

Real workload-driven simulations: We run ns-2 simulations to evaluate Aeolus with the four realistic workloads. Figure 11 shows the FCT distributions of flows of sizes between 0 and 100KB. We can see that Aeolus significantly improves FCTs of ExpressPass: with Aeolus, nearly 60%, 80%, 28% and 70% of 0-100KB small flows complete within the first RTT across the four workloads, respectively.

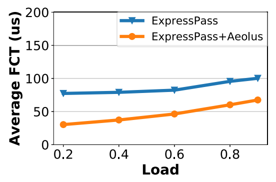
Figure 12 shows the improvement of Aeolus as the system load varies (from 20% to 90% of the network capacity). We can see that ExpressPass benefits from Aeolus with a sizable amount across a wide range of loads. As a second observation, we find that as the load increases, the room for improvement by Aeolus diminish slightly, which is a result of less spare bandwidth under high load. Nonetheless, we still observe a considerable improvement even at 90% load. This is partly because in practice, the bandwidth allocation of ExpressPass is not always perfectly work-conserving; some flows may get more credits than they demand, resulting in link underutilization. In contrast, Aeolus can use such spare bandwidth by injecting unscheduled packets in the first RTT.

Performance metric: We use flow completion time (FCT) as the primary performance metric. We also measure the queue length, link utilization and goodput for analysis.

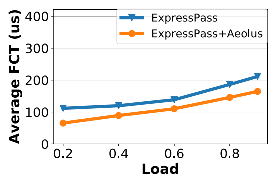
Figure 13 shows the FCT slowdown across all flows on average and at the 99th-percentile, respectively. Here, FCT slowdown means a flow's actual FCT normalized by its ideal FCT when the network only serves this flow. Flows are generated according to Web Search workload. As we can see, while Aeolus improves the FCT of all flows, the improvement decreases as flow size increases. The results are expected since a better utilization of bandwidth in the first RTT has little impact on the FCT of large flows.



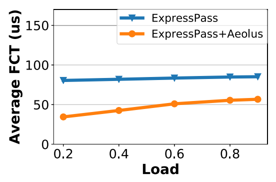
(a) Web Server: 0-100KB



(b) Cache Follower: 0-100KB

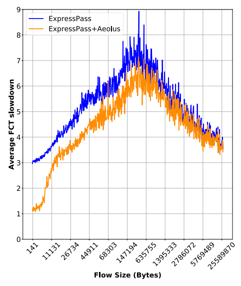


(c) Web Search: 0-100KB

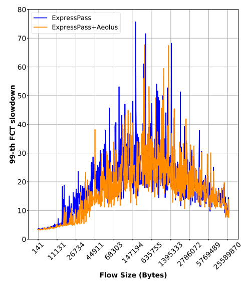


(d) Data Mining: 0-100KB

Fig. 12. Average FCT of 0-100KB small flows with the varying load.

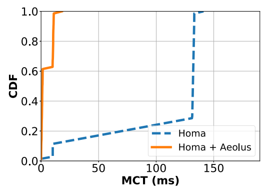


(a) Slowdown-average

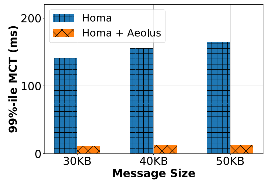


(b) Slowdown-99%

Fig. 13. FCT slowdown across all flows with Web Search workload. The average load of the network core is 40%.



(a) MCT with 30KB message size



(b) Mean MCT with different message sizes

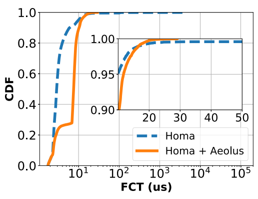
Fig. 14. Message completion times (MCT) of a 7-to-1 incast. The message size varies from 30KB to 50KB.

C. Homa + Aeolus

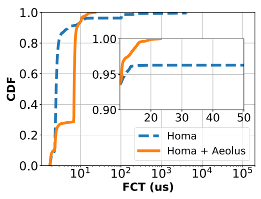
With testbed experiments and simulations, we demonstrate that Aeolus can help Homa eliminate large queue buildup and avoid losses of scheduled packets, thus significantly improving the tail FCTs of small flows.

Testbed experiments: Figure 14 shows the distribution of message completion times (MCT) over messages of size between 30KB and 50KB. We can see that Aeolus effectively cuts the tail MCT from 141ms to 18ms, and reduces the average MCT from 100s of ms to a few ms! This is because although both Homa and Homa+Aeolus send unscheduled packets in the first RTT, Aeolus only drops unscheduled packets and ensures that the dropped unscheduled packets can be quickly recovered from the second RTT without waiting for timeouts, thus achieving predictable tail latency. In contrast, Homa may suffer from timeouts if tail packets are dropped.

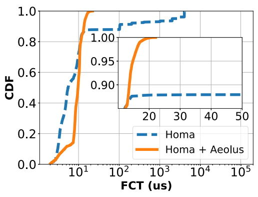
Real workload-driven simulations: We run OMNET++ simulations to evaluate Aeolus with the four realistic workloads. The timeout value is set to be 1ms. The network load is 40%. Figure 15 shows the FCT distributions of flows smaller than 100KB. We can see that across all workloads,



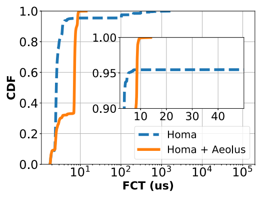
(a) Web Server



(b) Cache Follower



(c) Web Search



(d) Data Mining

Fig. 15. FCT of 0-100KB flows in a two-tier spine-leaf topology. The average load of network core is 40%.

TABLE II
TAIL FCT OF 0-100KB FLOWS UNDER HOMA AND HOMA+AEOLUS ACROSS THE FOUR WORKLOADS

	Web Server/μs	Cache Follower/μs	Web Search/μs	Data Mining/μs
Homa	4043.50	4051.69	4228.9	1802.65
Homa + Aeolus	30.30	23.05	22.09	13.93

TABLE III
MEDIAN FCT OF 0-100KB FLOWS UNDER HOMA AND HOMA+AEOLUS ACROSS THE FOUR WORKLOADS

	Web Server/μs	Cache Follower/μs	Web Search/μs	Data Mining/μs
Homa	2.88	2.54	5.53	2.56
Homa + Aeolus	7.52	7.05	9.69	7.13

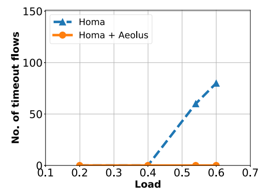
TABLE IV
AVERAGE FCT OF 0-100KB FLOWS UNDER HOMA AND HOMA+AEOLUS ACROSS THE FOUR WORKLOADS

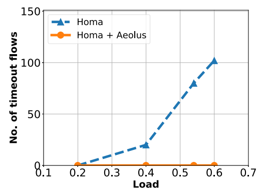
	Web Server/μs	Cache Follower/μs	Web Search/μs	Data Mining/μs
Homa	7.86	29.64	239.67	14.75
Homa + Aeolus	6.97	6.24	9.49	5.84

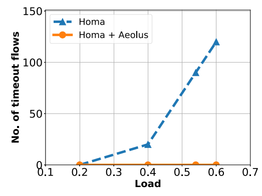
Homa+Aeolus completes all flows within 31μs whereas the 99th-percentile tail FCT of Homa is ~4ms. This is because Aeolus avoids the losses of scheduled packets and can do fast recovery for the dropped unscheduled packets.

Table II, Table III and Table IV summarize the tail, median and average FCTs of flows smaller than 100KB, respectively. Although the median FCT of Homa+Aeolus is slightly higher, Aeolus significantly improves the tail FCT and thus improves the average FCTs.

To confirm the intuition that the drops of scheduled packets cause the performance gap between Homa and Aeolus, Figure 16 shows the number of flows that experience at least one timeout under different levels of load. We can see that as the load increases, the spare bandwidth drops, which increases the chance of contention between scheduled and unscheduled packets. When contention occurs, Homa prioritizes unscheduled packets, causing some scheduled packets to be queued or dropped. In contrast, by design, Aeolus will protect the scheduled packets, so no flow experiences timeout even at 60% load.


(a) Web Server: 0-100KB


(b) Cache Follower: 0-100KB


(c) Web Search: 0-100KB

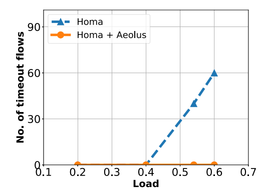

(d) Data Mining: 0-100KB

Fig. 16. Number of flows suffering from timeout. The load varies from 0.2 to 0.6.

TABLE V
AVERAGE FCT OF ALL FLOWS UNDER EAGER HOMA AND HOMA+AEOLUS ACROSS THE FOUR WORKLOADS

	Web Server/μs	Cache Follower/μs	Web Search/μs	Data Mining/μs
Eager Homa	27.66	369.82	421.10	1885.15
Homa + Aeolus	17.55	72.69	278.52	1637.11

With Aeolus, any dropped unscheduled packet can be detected by probe packets, and its retransmission is guaranteed with the scheduled packet. Therefore, Aeolus can utilize the network bandwidth in an accurate and efficient way. To confirm this, Table V shows the average FCT of all flows under eager Homa (20μs timeout) and Aeolus, respectively. We can see that, compared to eager Homa, Aeolus reduces the average FCT of all flows by 36.55% 80.34% 33.86% 13.16% across the four workloads, respectively (Note that in Data Mining workload, the 99% of flows are smaller than 100MB, but more than 90% of bytes are in flows larger than 100MB. For these >100MB large flows, Aeolus cannot greatly reduce its FCT. That's why Aeolus improves the average FCT only by a smaller fraction.).

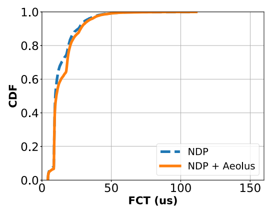
D. NDP + Aeolus

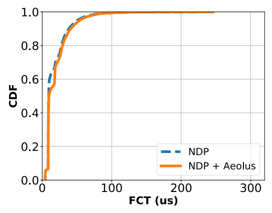
We show that Aeolus can enable NDP to maintain its high performance without cutting payload (CP) [12] support. The CP technique adopted by NDP is not supported by existing commodity switching ASICs yet, e.g., Broadcom Trident 2, Tomahawk and Tomahawk2. It remains an open question whether CP can be realized in a readily deployable and cost-effective way.

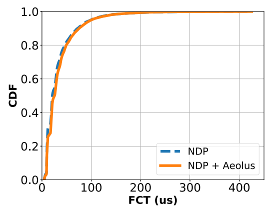
As we do not have NetFPGA card to implement CP, we only conduct simulations for the evaluation of NDP and Aeolus. Given we have already shown Aeolus can be implemented on commodity hardware, here we focus only on showing that NDP+Aeolus achieves similar performance as original NDP.

Figure 17 shows the FCT distributions of flows smaller than 100KB. We can see that NDP+Aeolus achieves similar FCT as original NDP in all percentiles. We also measure the average FCT under varying network loads from 20% to 90% across the four workloads (results are not presented due to space limitation), and have the similar finding.

CP plays an important role in the design of NDP. For NDP, Aeolus works as an alternative to CP. With Aeolus, NDP


(a) Web Server


(b) Cache Follower


(c) Web Search

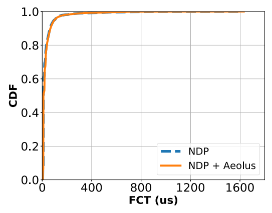

(d) Data Mining

Fig. 17. FCT of 0-100KB flows in a two-tier spine-leaf topology. The average load of network core is 40%.

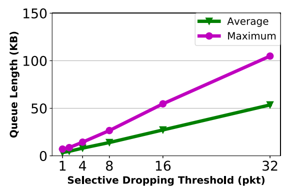


Fig. 18. Avg. and max. queue length with different thresholds.

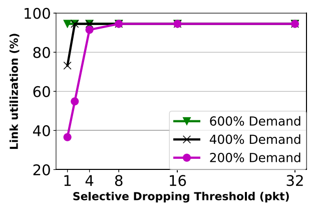


Fig. 19. Avg. link utilization with different thresholds.

avoids large queue buildups by selectively dropping excessive unscheduled packets at switch queues. Aeolus ensures effective retransmissions by leveraging the lossless property of proactive congestion control. The main advantage of Aeolus over CP is that, Aeolus is compatible with existing commodity switches, and thus can significantly reduce the complexity to deploy NDP in production DCNs.

With the simulation setting introduced in $\S V-E$, we also evaluate NDP+Aeolus using a mix of realistic traffic and bursty incast traffic. In such a setting with more serious congestion and more packet drops, we find that NDP+Aeolus degrades the performance for small flows (FCT prolonged by 3x in the worst case). The reasons are twofold. First, aggressively dropping unscheduled packets has a larger impact on small flows. Second, Aeolus's probe-based loss detection is not always as efficient as CP for recovering packets lost in the first RTT.

E. Aeolus Deep Dive

Parameter sensitivity: Readers may wonder how to set a proper selective dropping threshold for Aeolus— a very small threshold may too aggressively drop the unscheduled packets (thus fail to fully utilize spare bandwidth), while a very large one may build long switch queues (thus significantly delay scheduled packets). We conduct a many-to-one simulation to

TABLE VI
AEOLUS VS PRIORITY QUEUEING: PROBLEM OF AMBIGUITY

	Max. FCT / μs	Transfer Efficiency
ExpressPass + Aeolus	135.04	0.90
ExpressPass + Prio. Queueing (RTO = 10ms)	10230.13	0.90
ExpressPass + Prio. Queueing (RTO = 20μs)	158.13	0.41

TABLE VII
AEOLUS VS PRIORITY QUEUEING: UNSCHEDULED PACKETS RESULTS IN THE DROPPING OF SCHEDULED PACKETS

	Avg. FCT (μs)	Max. FCT (μs)
ExpressPass + Aeolus	656	986
ExpressPass + Priority Queueing	8694	10866

evaluate the parameter sensitivity of Aeolus. There are N senders and one receiver. All the hosts are connected to a switch using 100Gbps links. In each RTT, all the senders transfer 200KB data to the receiver.

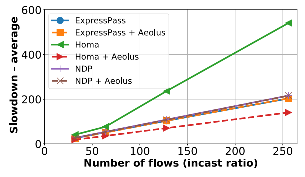
In Figure 18, we plot the average and maximum queue length on the congested link with different selective dropping thresholds. We find that, the queue length is nearly linear to the selective dropping threshold. Hence to avoid large switch queues, we should use a small selective dropping threshold.

So how to choose a small threshold without sacrificing much throughput? To explore this, we measure the average link utilization of the bottleneck link in the first RTT. We create different traffic demands by adjusting the fan-in degree N. In Figure 19, we plot the average utilization of the bottleneck link under different traffic demands. As we can see, a small threshold of 4 packets (6KB) is large enough to achieve high throughput under all traffic demands.

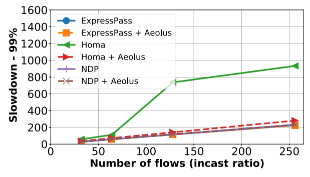
Why not priority queueing? We compare Aeolus with an alternative design: isolate unscheduled packets and scheduled packets with two priority queues. As stated in $\S III-B$, the most serious problem of priority queueing is ambiguity: when the receiver has been waiting for an unscheduled packet for a long time, it is hard to decide whether his packets has been dropped or it still being trapped in the network.

To showcase this ambiguity, we implement the priority queueing based solution in ns-2 simulator. We consider two retransmission timeouts (RTOs): 10ms and 20μs. The large RTO is resilient to packet trapping, but cannot efficiently recover unscheduled packet losses. In contrast, the small RTO incurs severe redundant transmissions. We run the cache follower workload in the 100G fat-tree topology. The proactive algorithm is ExpressPass. We measure the maximum FCT and transfer efficiency. As shown in Table VI, the large RTO suffers from high tail latency due to slow loss recovery, while the small RTO causes many redundant transmissions, thus degrading transfer efficiency.

We also show that isolating unscheduled packets in low priority queues does have the risk of affecting scheduled packets, at extreme case. We consider a contrived 20-to-1 incast scenario where each sender sends 400KB data to a common receiver. All servers are directly connected to a 100G switch, where shared buffer scheme is adopted across different priority queues. The average and maximum FCT under Aeolus and priority queueing are shown in Table VII. It is easy to see that, compared with Aeolus, priority queueing results in much longer FCTs (~10$\times$ worse than Aeolus). The reason is that, switch buffer is fully occupied by unscheduled packets queued



(a) Slowdown-average



(b) Slowdown-99%

Fig. 20. FCT slowdown with varying incast ratio in a 2-tier spine-leaf topology.

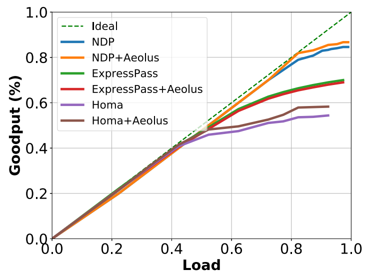
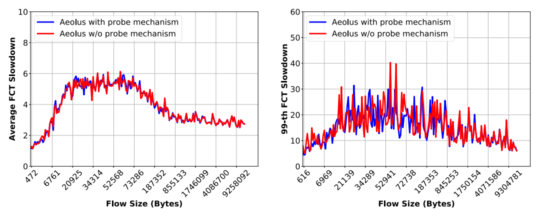


Fig. 21. Goodput across varying network loads.

at low priority queue. As a result, scheduled packets are rejected to enter high priority queue due to the lack of available buffer (drop-tail). As the dropping of scheduled packets is rare for proactive solutions like ExpressPass [1], a large RTO = 10ms is used for the recovery of dropped scheduled packets, which results in worse FCTs.

Heavy incast with larger network: As a stress test, we study the behaviour of Aeolus under heavy incast by conducting N-to-1 incast simulations in a two-tier spine-leaf network (N = 32, 64, 128 and 256). The network has 4 spine switches, 9 leaf switches and 144 servers (16 under each leaf switch). Server links operate at 100 Gbps and spine-leaf links operate at 400 Gbps. All links have 0.2μs propagation delay. All switches have 0.25μs switching delay. Each switch port has 500KB buffer. All the flows have 64KB data. We choose senders randomly across all servers. For Homa, we use 40μs as the retransmission timeout, which is equal to the largest queueing delay a packet could experience in the network.

Figure 20 shows the FCT slowdown on average and at the 99th-percentile, respectively. We mainly make three observations. First, compared with ExpressPass, ExpressPass+Aeolus achieves similar performance. This is expected because the main benefit brought by Aeolus is the ability to utilize spare bandwidth with unscheduled packets in the first RTT. However, in the heavy incast scenarios, the proportion of data bytes that can be transmitted in the first RTT is minimal compared to the total bytes of all flows. As a result, Aeolus can hardly make further improvement. Second, Aeolus enables Homa to achieve good performance even under heavy incast. This is because Aeolus can avoid large queue buildup by selectively dropping the overwhelming unscheduled packets in the first RTT. As a result, scheduled packets are protected from large queueing delay and packet loss since the second RTT. Lost unscheduled packets are also recovered quickly using scheduled packets. Under Homa, however, both unscheduled and scheduled packets will suffer from severe losses due to large queue buildups. With the inefficient timeout-based loss recovery mechanism, Homa will spend much longer time on completing the transmission of all flows. Third, compared with NDP, NDP+Aeolus achieves similar performance. This is consistent with our previous evaluation result.



(a) Slowdown-average (b) Slowdown-99%

Fig. 22. FCT slowdown of Aeolus with and w/o SACK.

Impact on goodput: Readers may wonder whether aggressively dropping unscheduled packets in the first RTT would negatively affect the effective bandwidth utilization of each scheme. To study this, we evaluate each scheme with increasing network loads to identify the maximum goodput it can achieve. For this simulation, we use the same spine-leaf topology as above. We generate network loads using a mix of Web Search traffic and incast traffic. We generate the incast traffic by randomly selecting 64 senders and one receiver, each sending 64KB data.

Figure 21 shows the goodput (normalized by the link capacity) each scheme can achieve over varying network loads. Compared with ExpressPass, Aeolus has no negative impact on goodput. For Homa, Aeolus improves the goodput by 4%. This is mainly because Aeolus eliminates the losses of scheduled packets and does fast recovery for lost unscheduled packets. For NDP, Aeolus improves the goodput by 2%. This is because NDP needs to reserve some bandwidth headroom for transmitting trimmed packet headers. In contrast, Aeolus only requires 64 bytes per flow for transmitting the probe.

Furthermore, across these schemes, we observe that NDP achieves the highest goodput (84%) mainly for two reasons. First, it performs per-packet load balancing to fully utilize the bandwidth over multiple paths. Second, it leverages CP to enable fast loss recovery. In contrast, ExpressPass uses per-flow ECMP for load balancing, thus only achieving 70% goodput. In addition, Homa achieves the worst goodput (54%) due to the losses of many scheduled packets and the use of inefficient timeout-based loss recovery mechanism. We note that our result with Homa is inconsistent with the result presented in the Homa paper [3]. We suspect one possible reason is that Homa assumes infinite switch buffer in their simulations. In contrast, in our simulations we allocate 500KB buffer for each switch port.

Benefit of the probe mechanism: To study the performance gain provided by the probe mechanism, we compare the flow performance under two schemes"Aeolus with probe mechanism" and "Aeolus w/o probe mechanism". In this simulation, we use the same spine-leaf topology as above. Flows are generated according to Web Search workload. The network load is 0.4.

Figure 22 shows the average and the 99-th percentile FCT slowdown under both schemes. As we can see, both schemes achieve very similar performance. The reason is as follows. In the first RTT, both schemes will successfully transmit the same bytes of unscheduled packets as the dropping threshold is the same. Since the second RTT, both schemes will utilize the granted credits to transmit scheduled packets or retransmit lost unscheduled packets. The main benefit of probe mechanism is the fast detection of tail loss in the first RTT. For scheme "Aeolus with probe mechanism", lost unscheduled packets at the tail can be quickly detected, and credits will be used to retransmitted these packets with highest priority. For "Aeolus w/o probe mechanism", while lost unscheduled packets at the tail cannot be detected before timeout, the credits will be used either to transmit unsent new data or to retransmit unacknowledged unscheduled packets (in the case there is no new data to send). So there is no wastage of credit even if lost unscheduled packets are not quickly detected. Since both schemes can fully utilize the granted credits, their performance should be similar.

We want to point out that, while FCT is similar under both schemes, application layer will see a very different performance. In the next, we use a toy experiment to demonstrate how the out-of-order data delivery under "Aeolus w/o probe mechanism" hurts the application performance. Transaction is a function widely supported by modern databases. A transaction usually contains a set of database operations that must be executed in batch in an atomic way. In our experiment, we have two servers connected to a 10G switch with one server working as client node and the other working as server node. The client node sends a transaction, which contains 20 operations, to the server node to execute on its database. For simplicity, we directly drop the last unscheduled packet at the client node to create tail loss. Compared with "Aeolus with probe mechanism", "Aeolus w/o probe mechanism" prolongs the transaction execution time by 16.9% (from 20.1ms to 23.5ms).

The reason is as follows. For "Aeolus with probe mechanism", the lost unscheduled packets will be quickly detected at the sender side and get retransmitted soon. As a result, the receiver will observe a good in-order delivery of packets, and thus the first few database operations will be received and executed soon. For "Aeolus w/o probe mechanism", however, as the lost unscheduled packets cannot be quickly detected, the sender will transmit the scheduled packets with larger sequence number at first on the receipt of credits. The lost unscheduled packets (whose payload carries the first few database operations) will be retransmitted at last when no unsent new data can be transmitted using credit. As a result, the transaction cannot be executed until all the data are received. This will significantly prolong the execution time of the transaction.

VI. Discussion

Design tradeoff: Aeolus guarantees good tail latency by protecting the deterministic nature of proactive schemes, but only improves the average latency in a best effort manner. In certain cases, it may also make some compromise. For example, it is possible that Aeolus may drop unscheduled packets even when the switch has enough buffer space to hold all the in-flight traffic, due to small selective dropping threshold (§IV-A). As a result, Aeolus may delay the completion of some small flows due to bandwidth wastage in the first RTT. However, Aeolus can consistently eliminate congestion timeouts, even under serious congestion.

Resilience under heavy incast: Aeolus uses minimum-sized probe packets to improve the resilience under heavy incast workloads. For example, with a typical setting of 200KB switch buffer, 6KB dropping threshold, and 64B probe packet size, Aeolus can effectively detect unscheduled packet losses even when there are 3100 (194KB/64B) new arrival flows. To handle the extreme cases where even the probe packet can get dropped, we may let the sender set a timer to retransmit unscheduled packets and the probe packet if no credit is received in a given duration.

Overhead of per-packet ACK for hardware offloading: Per-packet ACK can be a burden for hardware transport at high speed. However, Aeolus minimizes such overhead by only generating per-packet ACK for unscheduled packets, which are more likely to be dropped. For example, with 100Gbps link, 10μs base RTT and 1.5KB MTU, each flow only needs 84 ACKs for unscheduled packets.

Oversubscribed topology: Some proactive schemes (e.g., NDP [2], Homa [3] and pHost [8]) assume the network core is free of congestion. However, enabling proactive solutions to work with oversubscribed topologies is not a goal of Aeolus. For example, when integrated with above proactive schemes, Aeolus cannot avoid congestion losses of unscheduled packets in oversubscribed topologies.

Multi-queue scenario: Some proactive mechanisms, e.g., Homa [3], use multiple priority queues to serve unscheduled and scheduled packets separately. As a result, configuring per queue selective dropping at switches can no longer protect scheduled packets from the impact of unscheduled packets.[3]

To handle the above problem, we adopt per-port selective dropping mechanism, which observes the sum of queue buffer occupancy belonging to the same port. For each switch port, once the buffer occupancy reaches the dropping threshold K, the incoming unscheduled packets will get dropped. As a result, as long as the queued bytes of scheduled packets reach K, unscheduled packets can no longer consume the bandwidth allocated to scheduled packets, even if unscheduled packets are served with queues of higher priority. Note that per port selective dropping mechanism can also be implemented with commodity switches by configuring per-port ECN.

Extending Aeolus for other transport protocols: It is possible to extend Aeolus to augment other transports. For example, with Aeolus, TCP can start with a large initial window to fully utilize the available bandwidth instead of starting with a small initial window and then iteratively increasing. Aeolus's selective dropping mechanism can effectively minimize the impact of initial burst in the first RTT. But unlike proactive transports, TCP does not send scheduled packets whose deliveries are guaranteed. As a result, loss could happen to all packets and a more general loss recovery mechanism is needed.

The idea of Aeolus can also be applied to RDMA transports (e.g, DCQCN [5]) to enable them operate without PFC. In the design of DCQCN, a flow starts at full line rate on its arrival and PFC is leveraged to prevent packet loss caused by initial traffic burst. However, PFC is known to have many problems [5], [29]–[31], such as head-of-the-line blocking, congestion spreading and deadlock. Aeolus's selective dropping mechanism can be used to replace PFC for handling the overwhelming traffic burst at the beginning. Without PFC, the network is no longer lossless, so a more efficient loss recovery mechanism (e.g., the one proposed by IRN [32]) will be needed for handling packet losses.

Impact on bandwidth sharing among flows: By design, Aeolus will not significantly affect the bandwidth sharing among flows, for three reasons. First, Aeolus does not change the bandwidth allocation mechanism of original proactive transport. Second, at endhost, the sending of unscheduled packets in the first RTT will not delay the transmission of scheduled packets, as scheduled packets are sent since second RTT. Third, in the network, Aeolus's selective dropping mechanism guarantees that the bandwidth allocated to scheduled packets will not be occupied by unscheduled packets.

[3]In Homa, unscheduled packets are served in the queues of higher priority.

VII. Related Work

There are tons of DCN transport designs aiming at low latency and high throughput. We have discussed the closely related proactive solutions [1]–[3] extensively in the paper. Here, we review some other ideas which have not been discussed elsewhere.

In contrast to the proactive solutions, reactive DCN congestion control algorithms leverage signals such as ECN, RTT, and in-network telemetry (INT) to detect congestions. For example, DCTCP [4], MCP [33], BCC [34], [35] and DCQCN [5] use ECN as the congestion signal, TIMELY [6] uses RTT as the signal, whereas Gemini [36], [37] use a mix of both ECN and RTT as the signal. More recently, HPCC [38] leverages INT to obtain precise link load information. However, most of these solutions require at least one RTT to *react* to congestion and usually take multiple rounds to converge to the ideal rates. As a result, they are inefficient to provide persistent low latency in high speed DCNs.

There are other DCN research efforts such as flow scheduling (e.g., pFabric [19], PIAS [39], [40] and TCN [41]), mixflow scheduling [42], coflow scheduling (e.g., Aalo [43], CODA [44] and Stream [45]), multi-path load balancing (e.g., CONGA [46] and Hermes [47]) and works that combine a set of ideas (e.g., PASE [48] employs distributed arbitration, in-network prioritization and TCP-like end-host transport together). These designs either help to reduce flow/coflow completion times, or strike for higher network utilization with multiple path. However, none of them targets at the first RTT problem focused by this paper.

We note that in broader contexts other than DCN, efforts have also been made to enable fast flow start with large initial rates of transport protocols. For example, in the context of Internet, RC3 [20] proposed to use k levels of lower network priorities to transmit a larger number of additional packets during TCP's slow start phase in order to compensate its over-caution in window increase. However, it requires a large re-sequence buffer, which may not be affordable for transport in NIC hardware. In addition, the tail content of flow may keep changing if application continuously copies data into transport send buffer, making the design over-complicated. In the context of system area networks, SRP [49] allows senders to transmit speculative packets in the first RTT at lower network priority before bandwidth reservation is granted. Relative to Aeolus, both RC3 and SRP share the similar motivation of better utilizing spare bandwidth in the first RTT with prioritization. However, Aeolus differs from them in the way of implementing the prioritization. By identifying the problems of multiple priority queues as we discussed in §III-B, Aeolus proposed a novel selective dropping scheme that avoids these downsides by using only one queue.

VIII. Conclusion

This paper presented Aeolus, a readily deployable solution focusing on "pre-credit" packet transmission as a building block for all proactive transports. At the core of Aeolus, it prioritizes scheduled packets over unscheduled packets so that proactive transports can fully utilize spare bandwidth while preserving their deterministic nature. Furthermore, Aeolus introduces a novel selective dropping scheme which allows pre-credit new flows to burst at line-rate when there exists spare bandwidth, but immediately drops them selectively once the bandwidth is used up. In addition, Aeolus reuses the preserved proactive transport as a means to recover dropped first-RTT packets safely and efficiently. Aeolus is compatible with all existing proactive solutions. We have implemented an Aeolus prototype using DPDK and commodity switch hardware, and evaluated it through small testbed experiments and larger simulations. Both our implementation and evaluation results indicate that Aeolus is a promising substrate strengthening all existing proactive transport solutions.

References

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

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

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

[4] M. Alizadeh *et al.*, "Data center TCP (DCTCP)," in *Proc. ACM SIGCOMM Conf. SIGCOMM (SIGCOMM)*, 2010, pp. 63–74.

[5] Y. Zhu *et al.*, "Congestion control for large-scale RDMA deployments," in *Proc. ACM Conf. Special Interest Group Data Commun.*, Aug. 2015, pp. .

[6] R. Mittal *et al.*, "TIMELY: RTT-based congestion control for the datacenter," in *Proc. ACM Conf. Special Interest Group Data Commun.*, Aug. 2015, pp. 1–4.

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

[8] P. X. Gao, A. Narayan, G. Kumar, R. Agarwal, S. Ratnasamy, and S. Shenker, "PHost: Distributed near-optimal datacenter transport over commodity network fabric," in *Proc. 11th ACM Conf. Emerg. Netw. Exp. Technol.*, Dec. 2015, pp. 1–12.

[9] S. Hu, W. Bai, B. Qiao, K. Chen, and K. Tan, "Augmenting proactive congestion control with Aeolus," in *Proc. 2nd Asia–Pacific Workshop Netw. (APNet)*, 2018, pp. 22–28.

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

[11] *Dpdk*. [Online]. Available: https://www.dpdk.org/

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

[13] C.-Y. Hong, M. Caesar, and P. B. Godfrey, "Finishing flows quickly with preemptive scheduling," in *Proc. ACM SIGCOMM Conf. Appl., Technol., Archit., Protocols Comput. Commun. (SIGCOMM)*, 2012, pp. 127–138.

[14] J. Zhang, F. Ren, R. Shu, and P. Cheng, "TFC: Token flow control in data center networks," in *Proc. EuroSys*, 2016, pp. 1–14.

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

[16] A. Greenberg *et al.*, "VL2: A scalable and flexible data center network," in *Proc. ACM SIGCOMM Conf. Data Commun. (SIGCOMM)*, 2009, pp. 51–62.

[17] *Homa Simulator*. [Online]. Available: https://github.com/PlatformLab/HomaSimulation

[18] W. Bai, L. Chen, K. Chen, D. Han, C. Tian, and H. Wang, "Information-agnostic flow scheduling for commodity data centers," in *Proc. NSDI*, 2015, pp. 455–468.

[19] M. Alizadeh *et al.*, "PFabric: Minimal near-optimal datacenter transport," in *Proc. ACM SIGCOMM Conf. (SIGCOMM)*, Aug. 2013, pp. 435–446.

[20] R. Mittal, J. Sherry, S. Ratnasamy, and S. Shenker, "Recursively cautious congestion control," in *Proc. NSDI*, 2014, pp. 373–385.

[21] W. Bai, L. Chen, K. Chen, and H. Wu, "Enabling ECN in multi-service multi-queue data centers," in *Proc. NSDI*, 2016, pp. 537–549.

[22] *High-Density 25/100 Gigabit Ethernet Strataxgs Tomahawk Ethernet Switch Series*. [Online]. Available: https://www.broadcom.com/products/ethernet-connectivity/switch-fabric/b%cm56960

[23] *High-Capacity Strataxgs Trident II Ethernet Switch Series*. [Online]. Available: https://www.broadcom.com/products/ethernet-connectivity/switch-fabric/b%cm56850

[24] *Arista EOS*. [Online]. Available: https://www.arista.com/en/products/eos

[25] K. He *et al.*, "AC/DC TCP: Virtual congestion control enforcement for datacenter networks," in *Proc. ACM SIGCOMM Conf.*, Aug. 2016, pp. .

[26] H. Wu, J. Ju, G. Lu, C. Guo, Y. Xiong, and Y. Zhang, "Tuning ECN for data center networks," in *Proc. 8th Int. Conf. Emerg. Netw. Exp. Technol. (CoNEXT)*, 2012, pp. 244–257.

[27] *Expresspass Simulator*. [Online]. Available: https://github.com/kaist-ina/ns2-xpass

[28] *NDP Simulator*. [Online]. Available: https://github.com/nets-cs-pub-ro/NDP/wiki/NDP-Simulator

[29] S. Hu *et al.*, "Deadlocks in datacenter networks: Why do they form, and how to avoid them," in *Proc. 15th ACM Workshop Hot Topics Netw.*, Nov. 2016, pp. 92–98.

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

[31] S. Hu *et al.*, "Tagger: Practical PFC deadlock prevention in data center networks," in *Proc. 13th Int. Conf. Emerg. Netw. Exp. Technol.*, Nov. 2017, pp. 451–463.

[32] R. Mittal *et al.*, "Revisiting network support for RDMA," in *Proc. Conf. ACM Special Interest Group Data Commun.*, Aug. 2018, pp. 313–326.

[33] L. Chen, S. Hu, K. Chen, H. Wu, and D. H. K. Tsang, "Towards minimal-delay deadline-driven data center TCP," in *Proc. 12th ACM Workshop Hot Topics Netw.*, Nov. 2013, pp. 1–7.

[34] W. Bai, K. Chen, S. Hu, K. Tan, and Y. Xiong, "Congestion control for high-speed extremely shallow-buffered datacenter networks," in *Proc. 1st Asia–Pacific Workshop Netw.*, Aug. 2017, pp. 29–35.

[35] W. Bai, S. Hu, K. Chen, K. Tan, and Y. Xiong, "One more config is enough: Saving (DC) TCP for high-speed extremely shallow-buffered datacenters," *IEEE/ACM Trans. Netw.*, vol. 29, no. 2, pp. 489–502, Dec. 2020.

[36] G. Zeng, W. Bai, G. Chen, K. Chen, D. Han, and Y. Zhu, "Combining ECN and RTT for datacenter transport," in *Proc. 1st Asia–Pacific Workshop Netw.*, Aug. 2017, pp. 36–42.

[37] G. Zeng *et al.*, "Congestion control for cross-datacenter networks," in *Proc. IEEE 27th Int. Conf. Netw. Protocols (ICNP)*, Oct. 2019, pp. 1–12.

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

[39] W. Bai, L. Chen, K. Chen, D. Han, C. Tian, and W. Sun, "PIAS: Practical information-agnostic flow scheduling for data center networks," in *Proc. 13th ACM Workshop Hot Topics Netw.*, Oct. 2014, pp. 1–7.

[40] W. Bai, L. Chen, K. Chen, D. Han, C. Tian, and H. Wang, "PIAS: Practical information-agnostic flow scheduling for commodity data centers," *IEEE/ACM Trans. Netw.*, vol. 25, no. 4, pp. 1954–1967, Aug. 2017.

[41] W. Bai, K. Chen, L. Chen, C. Kim, and H. Wu, "Enabling ECN over generic packet scheduling," in *Proc. 12th Int. Conf. Emerg. Netw. Exp. Technol.*, Dec. 2016, pp. 191–204.

[42] L. Chen, K. Chen, W. Bai, and M. Alizadeh, "Scheduling mix-flows in commodity datacenters with karuna," in *Proc. ACM SIGCOMM Conf.*, Aug. 2016, pp. 174–187.

[43] M. Chowdhury and I. Stoica, "Efficient coflow scheduling without prior knowledge," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 393–406, 2015.

[44] H. Zhang, L. Chen, B. Yi, K. Chen, M. Chowdhury, and Y. Geng, "CODA: Toward automatically identifying and scheduling coflows in the dark," in *Proc. ACM SIGCOMM Conf.*, Aug. 2016, pp. 160–173.

[45] H. Susanto, H. Jin, and K. Chen, "Stream: Decentralized opportunistic inter-coflow scheduling for datacenter networks," in *Proc. IEEE 24th Int. Conf. Netw. Protocols (ICNP)*, Nov. 2016, pp. 1–10.

[46] M. Alizadeh *et al.*, "CONGA: Distributed congestion-aware load balancing for datacenters," in *Proc. ACM Conf. SIGCOMM*, Aug. 2014, pp. 503–514.

[47] H. Zhang, J. Zhang, W. Bai, K. Chen, and M. Chowdhury, "Resilient datacenter load balancing in the wild," in *Proc. Conf. ACM Special Interest Group Data Commun.*, Aug. 2017, pp. 253–266.

[48] A. Munir, G. Baig, S. M. Irteza, I. A. Qazi, A. X. Liu, and F. R. Dogar, "Friends, not foes: Synthesizing existing transport strategies for data center networks," in *Proc. ACM Conf. SIGCOMM*, Aug. 2014, pp. 491–502.

[49] N. Jiang, D. U. Becker, G. Michelogiannakis, and W. J. Dally, "Network congestion avoidance through speculative reservation," in *Proc. IEEE Int. Symp. High-Perform. Comp Archit.*, Feb. 2012, pp. 1–12.

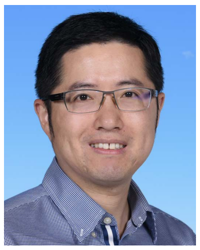
Shuihai Hu received the B.S. degree in computer science from USTC in 2013 and the Ph.D. degree from the Department of Computer Science and Engineering, The Hong Kong University of Science and Technology, in 2019. He is currently a Researcher at Huawei. His current interests are mainly about datacenter networks and machine learning systems.

Gaoxiong Zeng received the B.E. degree in electronic engineering from the University of Science and Technology of China in 2015. He is currently pursuing the Ph.D. degree in computer science with The Hong Kong University of Science and Technology. His research interests include computer networks and systems, with special focuses on data center networking and transport protocols.

Wei Bai received the B.E. degree in information security from Shanghai Jiao Tong University in 2013 and the Ph.D. degree from the Department of Computer Science and Engineering, The Hong Kong University of Science and Technology, in 2017. He is currently a Senior Researcher at the Microsoft Research Lab, Redmond. He is broadly interested in computer networking with a special focus on data center networking. His research work has been published in many top conferences and journals, such as SIGCOMM, NSDI, CoNEXT, and IEEE/ACM TRANSACTIONS ON NETWORKING. He is currently focusing on network infrastructure to support large-scale RDMA deployments.

Zilong Wang received the B.S. degree in physics and computer science from Peking University in 2019. He is currently pursuing the Ph.D. degree in computer science with The Hong Kong University of Science and Technology, Hong Kong. His research interest is in the area of data center networks.

Baochen Qiao is currently pursuing the M.Phil. degree with the Computer Science and Engineering Department, The Hong Kong University of Science and Technology. The objectives of his research are (i) to understand the problems in extremely high-speed data center networks, (ii) and to explore the deployable solutions to deliver a high-speed and low-latency network environment.



Kai Chen (Senior Member, IEEE) received the Ph.D. degree in computer science from Northwestern University, Evanston, IL, USA, in 2012. He is currently an Associate Professor with the Department of Computer Science and Engineering, The Hong Kong University of Science and Technology, Hong Kong. His research interests include data center networking, machine learning systems, and privacy-preserving computing.

Kun Tan is currently the Director with the Distributed and Parallel Software Laboratory, 2012 Labs, Huawei. Before he joined Huawei in 2016, he was a Senior Researcher and a Research Manager at Microsoft Research Asia, Beijing. His research interests include networked systems, cloud networking, and mobile computing. He received the Best Paper Award at NSDI in 2009 and USENIX Test-of-Time Award in 2019.

Yi Wang (Member, IEEE) received the Ph.D. degree in computer science and technology from Tsinghua University in July 2013. He was with the Sustech Institute of Future Networks, Southern University of Science and Technology. He is currently a Researcher with The Hong Kong University of Science and Technology. His research interests include future network architectures, information centric networking, software-defined networks, and the design and implementation of high-performance network devices.