Aeolus: A Building Block for Proactive Transport in Datacenters

Shuihai Hu[1,2*] Wei Bai[1,3*] Gaoxiong Zeng[1] Zilong Wang[1] Baochen Qiao[1] Kai Chen[1]
Kun Tan[4] Yi Wang[5]
[1]SING Lab, Hong Kong University of Science and Technology
[2]Clustar [3]Microsoft Research [4]Huawei [5]Peng Cheng Lab

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. 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. We have integrated Aeolus into ExpressPass[14], NDP[18] and Homa[29], and shown, through 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.

CCS CONCEPTS

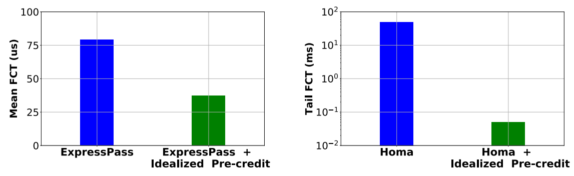
• **Networks** → **Transport protocols**; **Data center networks**.

KEYWORDS

Data Center Networks, Proactive Transport, First RTT, Selective Dropping

ACM Reference Format:
Shuihai Hu, Wei Bai, Gaoxiong Zeng, Zilong Wang, Baochen Qiao, Kai Chen, Kun Tan and Yi Wang. 2020. Aeolus: A Building Block for Proactive Transport in Datacenters. In *Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication (SIGCOMM '20), August 10–14, 2020, Virtual Event, NY, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3387514.3405878

*Work done while Shuihai and Wei were both at SING Lab @ HKUST.



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

Figure 1: A substantial gap between existing proactive transport baselines and the ideal performance. The setup is in §2. Aeolus provides a common building block of proactive transport to systematically bridge the performance gap caused by the pre-credit phase.

1 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 (§2.2). Therefore, it is crucial for transport to maintain low latency and high throughput at *every single RTT*.

Traditional "try and backoff" transports (e.g., DCTCP [9], DCQCN [36], Timely [27]) 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 [14], NDP [18], Homa [29], FastPass [30], pHost [16]) 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 [14]), the new flows will be *paused* by one RTT even though the network is underutilized (Figure 1(a)). If it bursts packets (e.g. Homa [29]), 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 [18]), 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 §2.3 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 (§2.4). 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× 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. [16, 18, 29]) 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.

Aeolus realizes its design goal by proposing a novel selective dropping mechanism (§3.2) 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 (§4.1).

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 (§3.3).

[1]We presented a preliminary idea of Aeolus in an earlier workshop paper [21]

Aeolus is architecturally compatible with all existing proactive solutions. We implemented an Aeolus prototype using DPDK [2] and commodity switch hardware (§4.2), and integrated it with the latest proactive solutions such as ExpressPass [14], Homa [29], and NDP [18]. We further built a small testbed with one Mellanox SN2000 switch and eight servers at 10Gbps (§5.1), 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×, from 100s of ms to a few ms in 10G testbed experiments, while achieving 1400× 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 [13] 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.

2 BACKGROUND AND MOTIVATION

2.1 Proactive datacenter transport

Datacenter congestion control traditionally (e.g. [9, 22, 27, 36]) 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 [30] employs a centralized arbiter to enforce a tight control over packet transmission time and path. PDQ [20] and TFC [35] leverage switches to explicitly allocate link bandwidth among the passing flows. ExpressPass [14], pHost [16], NDP [18] and Homa [29] use receiver-driven credit-based approaches to explicitly schedule the arrival of data packets destined for different receivers.

2.2 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

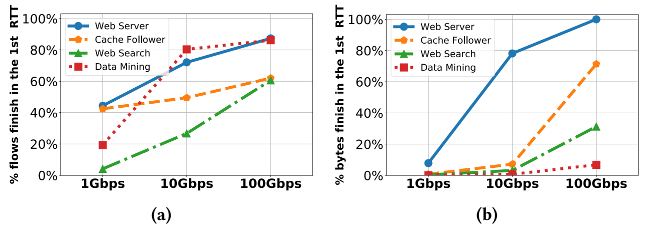


Figure 2: A substantial fraction of flows (and bytes) could have been finished within the first RTT (pre-credit phase), and this fraction grows rapidly as link speed increases.

within the first RTT (pre-credit phase) under different link speeds. Flows are generated according to four realistic workloads including Web Server [31], Cache Follower [31] Web Search [9] and Data Mining [17].

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 [14, 30]). This coincides with the ethos of recent proactive transport designs [16, 18, 29].
- *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.

2.3 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 [14], 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[2]. Flows are generated according to two realistic workloads, including Cache Follower [31] and Web Server [31].

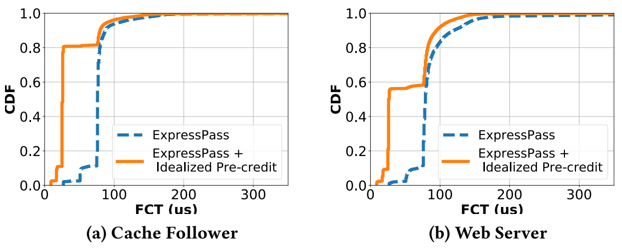


Figure 3: FCT of 0-100KB flows under the original ExpressPass and the hypothetical ExpressPass with idealized pre-credit solution (fully utilizes the spare bandwidth in the first RTT).

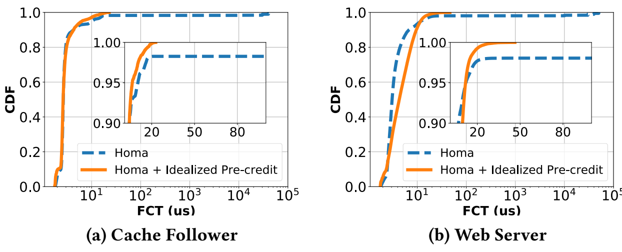


Figure 4: FCT of 0-100KB flows under the original Homa and the hypothetical Homa with idealized pre-credit solution (no interference between scheduled and unscheduled packets).

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× 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 [29], 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 [6] with a two-tier

[2]The same topology as used in the ExpressPass [14] paper.

	Tail FCT/μs (0-100KB)	Transfer Efficiency	Average FCT/μs (all flows)
Hypothetical Homa	25.04	0.90	34.84
Eager Homa	99.59	0.31	141.82
Original Homa	50030	0.90	74.39 (tail excluded)

Table 1: Tail FCT, transfer efficiency and average FCT under hypothetical Homa, eager Homa and original Homa (with Cache Follower workload).

tree topology of 8 spine switches, 8 leaf switches and 64 servers (8 servers per leaf switch) connected by 100Gbps links[3]. 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 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× reduction.

Readers may wonder, can Homa significantly reduce its tail FCT by adopting a much more aggressive loss recovery? To study this, we consider an eager version of Homa that uses 20μs retransmission timeout (the base RTT is 4.5μs). The simulation results[4] with Cache Follower workload are summarized in Table 1.

As we can see, while eager Homa does achieve a much better tail FCT for small flows (only 3× worse than Hypothetical Homa), the cost is very expensive — we observe a much lower transfer efficiency[5] (~65% downgrade) and a much higher average FCT of all flows (~300% increase). 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 drops to a small fraction of normal, which significantly prolongs the completion of the majority of flows. Note that for better demonstrating the overall performance of the majority of flows achieved by original Homa, we excluded its tail flows when calculating the average FCT (If tail flows included, the average FCT will be 641.74μs, which is the largest among the three schemes.).

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.

[3]The same topology as used in the Homa [29] paper.
[4]We ran simulations with the same setup, and omit similar results under other workloads for simplicity.
[5] We calculate transfer efficiency as total received data bytes over total sent bytes.

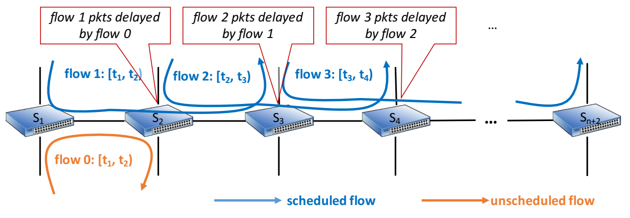


Figure 5: An illustrative example of a flow of unscheduled packets causing delays on many scheduled flows at downstream switches in a cascading manner.

2.4 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 much higher average FCT (Table 1). 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 collapse of transfer efficiency shown in Table 1.

To see a concrete example, let us consider Figure 5. Each link is fully scheduled to transmit scheduled packets when a 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.

3 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 6 overviews Aeolus, which mainly contains 3 components: rate control, selective dropping and loss recovery.

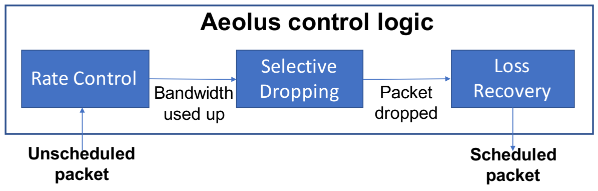


Figure 6: Aeolus overview.

- **Rate control (§3.1):** 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 (§3.2):** 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 (§4).
- **Loss recovery (§3.3):** 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*.

3.1 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 §3.2) 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 (§3.2) while any packet loss in the pre-credit stage can also be recovered shortly in the upcoming credit-based stage through scheduled packets (§3.3).

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 [16, 18, 30].

3.2 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 [10, 11, 28]. 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 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 §2.3). 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 §5.5. Second, unscheduled packets in the low priority may still occupy considerable buffer that risks affecting scheduled packets (showcase in §5.5), 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 [14]. Third, commodity switches have a smaller number of queues (typically 8), which may be usef for other purposes such as isolating traffic of different services [12]. 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 §4.1, 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.

3.3 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.

3.4 Why this Works

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 which has many complex loss recovery mechanisms [15] for different scenarios, Aeolus's loss recovery is extremely simple but more efficient.

4 IMPLEMENTATION

4.1 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[6]: red, yellow and green, and each color has its own dropping thresholds in a switch queue. WRED is widely supported by commodity switching chips [4, 5].

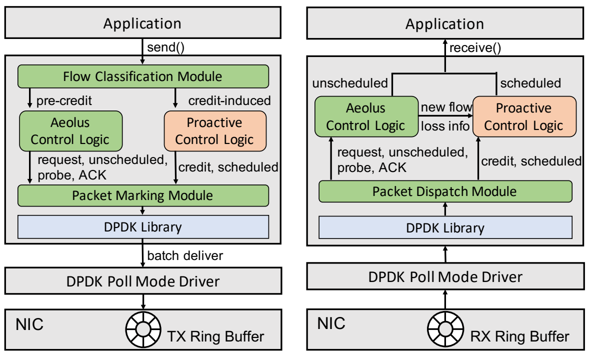
To implement selective dropping using WRED, we mark scheduled and unscheduled packets with different DSCP values at the end host. At the switch, we configure access control list (ACL) table to set the arriving packet's color based on its DSCP field. Therefore, scheduled and unscheduled packets can be 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 [1]) 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 [19, 24, 33].

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.

[6]Color is a metadata attached to the packet in switch processing pipeline.



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

Figure 7: Aeolus software implementation architecture on top of proactive solutions.

4.2 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 [14] and Homa [29]. Our implementation is based on DPDK 18.05 [2], which allows the network stack to bypass the kernel and communicate directly with the NIC.

As shown in Figure 7, 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 7(a), application starts data transmission by calling a send() function.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.

	Web Server [31]	Cache Follower [31]	Web Search [9]	Data Mining [17]
0 - 100KB	81%	53%	52%	83%
100KB - 1MB	19%	18%	18%	8%
> 1MB	0%	29%	20%	9%
Ave. flow size	64KB	701KB	1.6MB	7.41MB

Table 2: Flow size distributions of realistic workloads.

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.

5 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 [14]**, with the mean FCT reduced by up to 56%.
- **Aeolus improves the tail FCT of Homa [29]**, with the 99th percentile FCT reduced by up to 1400×.
- **Aeolus preserves the superior performance of NDP [18] without requiring switch modifications.**

5.1 Evaluation setup

Choices of baseline proactive transport: We choose three recent proactive transport solutions: ExpressPass [14], Homa [29] and NDP [18], 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 [14], the configuration is simple as we only have a single queue to transmit data packets, including scheduled packets and unscheduled packets. In contrast, Homa [29] 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 [12].

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 [3] with ns-2 simulator. For Homa, we implemented Aeolus on top of Homa's open source code [6] 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

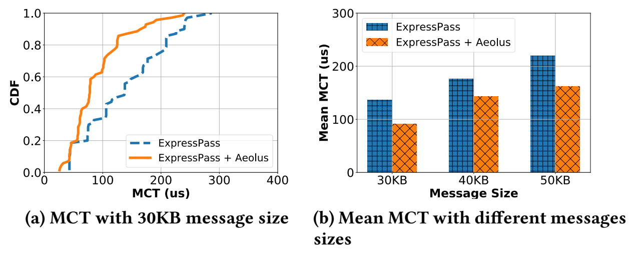
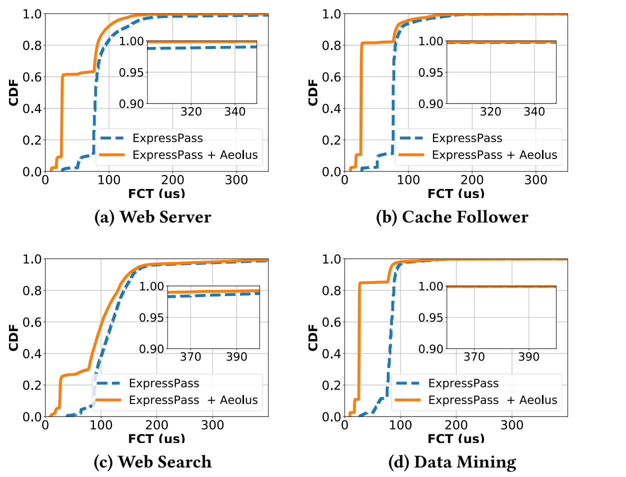

(a) MCT with 30KB message size (b) Mean MCT with different messages sizes

Figure 8: 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

Figure 9: FCT of 0-100KB flows in an oversubscribed fat-tree topology. The average load of the network core is 40%.

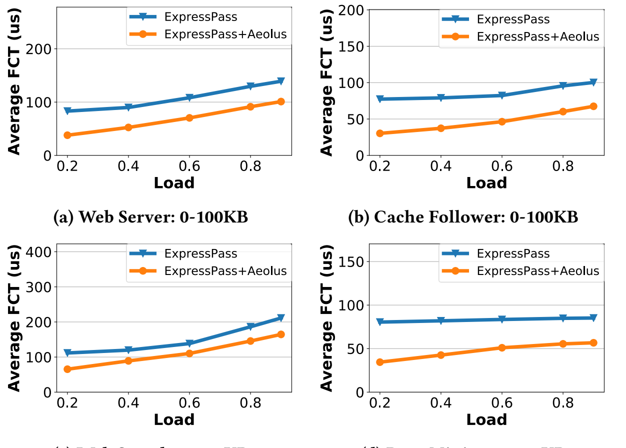
recovery mechanism according to the description in Homa paper. For NDP, we implemented Aeolus on top of NDP's packet-level htsim simulator [7].

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[7]. 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 10ms/20μs/40μs 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 [31], Cache Follower [31], Web Search [9] and Data Mining [17]. Their flow size distributions are shown in Table 2. 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

[7]NDP paper by default uses 9KB jumbo packet. We set MTU to be 9KB for NDP.


(a) Web Server: 0-100KB (b) Cache Follower: 0-100KB

(c) Web Search: 0-100KB (d) Data Mining: 0-100KB

Figure 10: Average FCT of 0-100KB small flows with the varying load. The average FCTs of larger flows (>100KB) with both schemes are similar. Hence, we omit them for abbreviation.

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.

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.

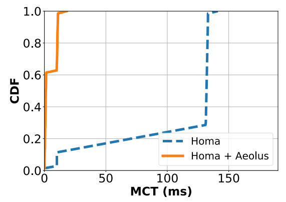
5.2 ExpressPass + Aeolus

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.

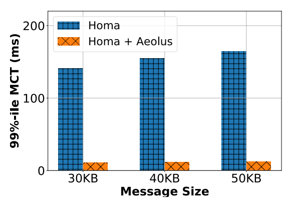
Testbed experiments: Figure 8 shows the message completion times (MCT) of 7-to-1 incast scenario when message size varies from 30KB to 50KB. The results indicate that Aeolus can assist ExpressPass to speed up small flows even under stressed incast traffic pattern: median MCT is improved by 43% with 30KB message size (Figure 8(a)), and average MCT is improved by 19%-33% across different message sizes (Figure 8(b)).

Real workload-driven simulations: We run ns-2 simulations to evaluate Aeolus with the four realistic workloads. Figure 9 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 10 shows the improvement of Aeolus as the system load varies (from 20% to 90% of the network capacity). We can see that

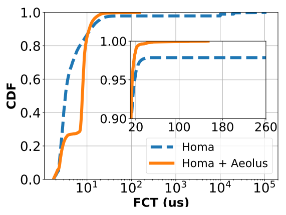


(a) MCT with 30KB message size

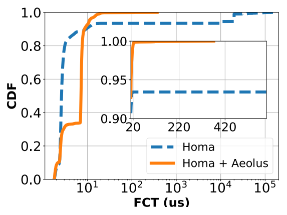


(b) Mean MCT with different message sizes

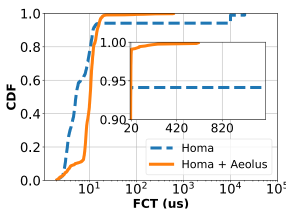
Figure 11: Message completion times (MCT) of a 7-to-1 incast. The message size varies from 30KB to 50KB.



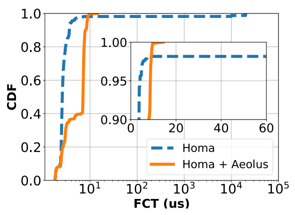
(a) Web Server



(b) Cache Follower



(c) Web Search



(d) Data Mining

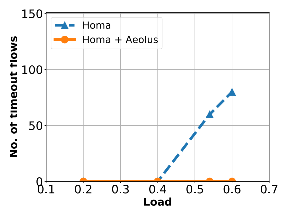
Figure 12: FCT of 0-100KB flows in a two-tier spine-leaf topology. The average load of network core is 54%.

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.

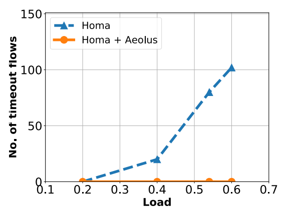
5.3 Homa + Aeolus

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.

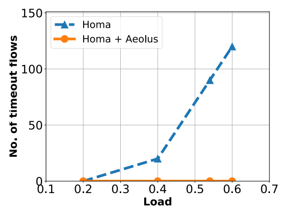
Testbed experiments: Figure 11 shows the distribution of message completion times (MCT) over messages of size between 30KB and 50KB. We can see that Aeolus effectively cuts the tail MCT from 141ms to 18ms, and reduces the average MCT from 100s of ms to a few ms! This is because although both Homa and 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.



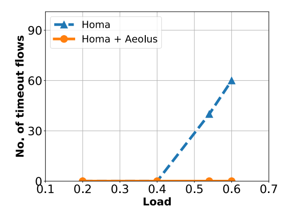
(a) Web Server: 0-100KB



(b) Cache Follower: 0-100KB



(c) Web Search: 0-100KB



(d) Data Mining: 0-100KB

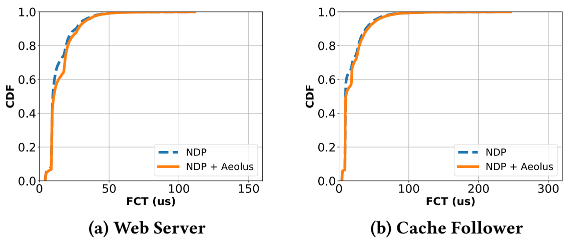
Figure 13: Number of flows suffering from timeout. The load varies from 0.2 to 0.9.

	Web Server/μs	Cache Follower/μs	Web Search/μs	Data Mining/μs
Eager Homa	13.59	141.82	281.62	25.86
Homa + Aeolus	6.93	35.34	107.47	24.22

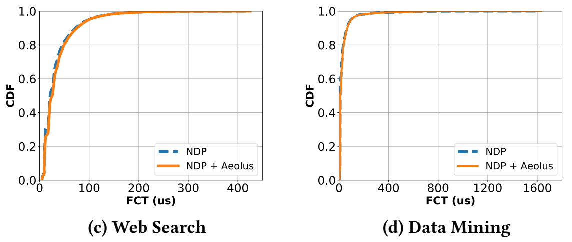
Table 3: Average FCT of all flows under eager Homa and Homa+Aeolus across the four workloads.

Real workload-driven simulations: We run OMNET++ simulations to evaluate Aeolus with the four realistic workloads. We use a network load of 54% because we observe that this is the maximum sustainable network load that Homa can support (check Figure 18 for more details). Figure 12 shows the FCT distributions of flows smaller than 100KB. We can see that across all workloads, Homa+Aeolus completes all flows within 610μs whereas the 99th-percentile tail FCT of Homa is ~150ms. This is because Aeolus avoids the losses of scheduled packets and can do fast recovery for the dropped unscheduled packets. Although the median FCT of Homa+Aeolus is slightly higher (e.g., 7.86μs vs 3.26μs for Web Server), the reduction on tail FCT dramatically improves the mean FCTs (e.g., from 403.4μs to 7.6μs for Web Server). With the simulation setting introduced in §5.5, we also evaluate Homa+Aeolus using a mix of realistic traffic and incast traffic and find that Aeolus achieves up to 850× reduction on the tail FCT.

To confirm the intuition that the drops of scheduled packets cause the performance gap between Homa and Aeolus, Figure 13 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 (b) Cache Follower



(c) Web Search (d) Data Mining

Figure 14: FCT of 0-100KB flows in a two-tier spine-leaf topology. The average load of network core is 40%.

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 3 shows the average FCT of all flows under eager Homa (20μs RTO) and Aeolus, respectively. We can see that, compared to eager Homa, Aeolus reduces the average FCT of all flows by 49%, 75%, 62% , 6% 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 small fraction.). We also measure the transfer efficiency achieved by Aeolus. As expected, Aeolus achieves near-optimal transfer efficiency across the four workloads, identical to the transfer efficiency achieved by hypothetical Homa shown in Table 1.

5.4 NDP + Aeolus

We show that Aeolus can enable NDP to maintain its high performance without cutting payload (CP) [13] 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 14 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 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.

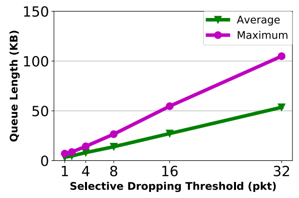


Figure 15: Avg. and max. queue length with different thresholds.

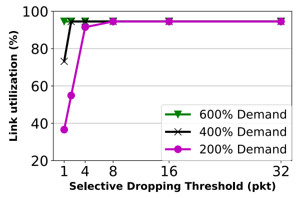


Figure 16: Avg. link utilization with different thresholds.

With the simulation setting introduced in §5.5, 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.

5.5 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 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 15, 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 16, 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.

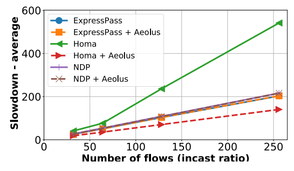
Why not priority queueing? We compare Aeolus with an alternative design: isolate unscheduled packets and scheduled packets with two priority queues. As stated in §3.2, the most serious problem of priority queueing is ambiguity: when the receiver has been waiting for an unscheduled packet for a long time, it is hard to decide whether his packets has been dropped or it still being trapped in the network.

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

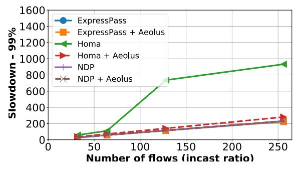
Table 4: Aeolus vs Priority Queueing: problem of ambiguity.

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

Table 5: Aeolus vs Priority Queueing: unscheduled packets results in the dropping of scheduled packets



(a) Slowdown-average



(b) Slowdown-99%

Figure 17: FCT slowdown with varying incast ratio in a two-tier spine-leaf topology.

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 4, 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 5. It is easy to see that, compared with Aeolus, priority queueing results in much longer FCTs (~10× worse than Aeolus). The reason is that, switch buffer is fully occupied by unscheduled packets queued 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 [14], 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 behavior 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.

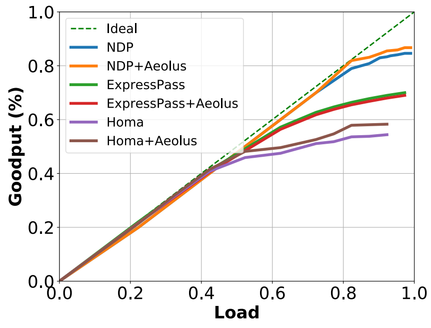


Figure 18: Goodput across varying network loads.

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 17 shows the FCT slowdown[8] 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.

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 18 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

[8]"FCT slowdown" means a flow's actual FCT normalized by its ideal FCT when the network only serves this flow.

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 [29]. 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.

6 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 (§4.1). 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 [18], Homa [29] and pHost [16]) 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.

7 RELATED WORK

There are tons of DCN transport designs aiming at low latency and high throughput. We have discussed the closely related proactive solutions [14, 18, 29] extensively in the paper. Here, we only overview some other ideas which have not been discussed elsewhere.

In contrast to proactive solutions, reactive DCN congestion control algorithms leverage advanced signals, e.g., ECN, RTT and in-network telemetry (INT), to detect congestion. For example, DCTCP [9], D^2TCP [32] and DCQCN [36] leverage ECN as the congestion signal. TIMELY [27] and DX [25] use delay as the signal. More recently, HPCC [26] 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 [10] and PIAS [11]) and multi-path load balancing (e.g., CONGA [8] and Hermes [34]). These designs either help to reduce flow 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 [28] 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. In the context of system area networks, SRP [23] 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 §3.2, Aeolus proposed a novel selective dropping scheme that avoids these downsides by using only one queue.

8 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. *This work does not raise any ethical issues.*

Acknowledgements: This work is supported in part by the Hong Kong RGC GRF-16215119 and PCL FANet No.LZC0019. We thank our shepherd Rachit Agarwal and the anonymous SIGCOMM reviewers for their constructive feedback and suggestions. We also thank Junchen Jiang for insightful discussions on this paper.

REFERENCES

[1] Arista eos. https://www.arista.com/en/products/eos.
[2] Dpdk. https://www.dpdk.org/.
[3] Expresspass simulator. https://github.com/kaist-ina/ns2-xpass.
[4] High-capacity strataxgs trident ii ethernet switch series. https://www.broadcom.com/products/ethernet-connectivity/switch-fabric/bcm56850.
[5] High-density 25/100 gigabit ethernet strataxgs tomahawk ethernet switch series. https://www.broadcom.com/products/ethernet-connectivity/switch-fabric/bcm56960.
[6] Homa simulator. https://github.com/PlatformLab/HomaSimulation.
[7] NDP simulator. https://github.com/nets-cs-pub-ro/NDP/wiki/NDP-Simulator.
[8] Mohammad Alizadeh, Tom Edsall, Sarang Dharmapurikar, Ramanan Vaidyanathan, Kevin Chu, Andy Fingerhut, Vinh The Lam, Francis Matus, Rong Pan, Navindra Yadav, et al. Conga: Distributed congestion-aware load balancing for datacenters. In *SIGCOMM 2014*.
[9] Mohammad Alizadeh, Albert Greenberg, David A. Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. Data center tcp (dctcp). In *SIGCOMM 2010*.
[10] Mohammad Alizadeh, Shuang Yang, Milad Sharif, Sachin Katti, Nick McKeown, Balaji Prabhakar, and Scott Shenker. pfabric: Minimal near-optimal datacenter transport. In *SIGCOMM 2013*.
[11] Wei Bai, Li Chen, Kai Chen, Dongsu Han, Chen Tian, and Hao Wang. Information-agnostic flow scheduling for commodity data centers. In *NSDI 2015*.
[12] Wei Bai, Li Chen, Kai Chen, and Haitao Wu. Enabling ecn in multi-service multi-queue data centers. In *NSDI 2016*.
[13] Peng Cheng, Fengyuan Ren, Ran Shu, and Chuang Lin. Catch the whole lot in an action: Rapid precise packet loss notification in data centers. NSDI'14.
[14] Inho Cho, Keon Jang, and Dongsu Han. Credit-scheduled delay-bounded congestion control for datacenters. In *SIGCOMM 2017*.
[15] Tobias Flach, Nandita Dukkipati, Andreas Terzis, Barath Raghavan, Neal Cardwell, Yuchung Cheng, Ankur Jain, Shuai Hao, Ethan Katz-Bassett, and Ramesh Govindan. Reducing web latency: The virtue of gentle aggression.
[16] Peter X. Gao, Akshay Narayan, Gautam Kumar, Rachit Agarwal, Sylvia Ratnasamy, and Scott Shenker. phost: Distributed near-optimal datacenter transport over commodity network fabric. In *CoNEXT 2015*.
[17] Albert Greenberg, James R. Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A. Maltz, Parveen Patel, and Sudipta Sengupta. Vl2: a scalable and flexible data center network. In *SIGCOMM 2009*.
[18] Mark Handley, Costin Raiciu, Alexandru Agache, Andrei Voinescu, Andrew W Moore, Gianni Antichi, and Marcin Wójcik. Re-architecting datacenter networks and stacks for low latency and high performance. In *SIGCOMM 2017*.
[19] Keqiang He, Eric Rozner, Kanak Agarwal, Yu (Jason) Gu, Wes Felter, John Carter, and Aditya Akella. Ac/dc tcp: Virtual congestion control enforcement for datacenter networks. In *SIGCOMM 2016*.
[20] Chi-Yao Hong, Matthew Caesar, and P Godfrey. Finishing flows quickly with preemptive scheduling. In *SIGCOMM 2012*.
[21] Shuihai Hu, Wei Bai, Baochen Qiao, Kai Chen, and Kun Tan. Augmenting proactive congestion control with aeolus. In *Proceedings of the 2nd Asia-Pacific Workshop on Networking*.
[22] V. Jacobson. Congestion avoidance and control. In *SIGCOMM 1988*.
[23] Nan Jiang, Daniel U. Becker, George Michelogiannakis, and William J. Dally. Network congestion avoidance through speculative reservation. HPCA '12.
[24] Glenn Judd. Attaining the promise and avoiding the pitfalls of tcp in the datacenter. In *NSDI 2015*.
[25] Changhyun Lee, Chunjong Park, Keon Jang, Sue Moon, and Dongsu Han. Accurate latency-based congestion feedback for datacenters. In *ATC 2015*.
[26] Yuliang Li, Rui Miao, Hongqiang Harry Liu, Yan Zhuang, Fei Feng, Lingbo Tang, Zheng Cao, Ming Zhang, Frank Kelly, Mohammad Alizadeh, and Minlan Yu. Hpcc: High precision congestion control. SIGCOMM '19.
[27] Radhika Mittal, Vinh The Lam, Nandita Dukkipati, Emily Blem, Hassan Wassel, Monia Ghobadi, Amin Vahdat, Yaogong Wang, David Wetherall, and David Zats. Timely: Rtt-based congestion control for the datacenter. In *SIGCOMM 2015*.
[28] Radhika Mittal, Justine Sherry, Sylvia Ratnasamy, and Scott Shenker. Recursively cautious congestion control. NSDI'14.
[29] Behnam Montazeri, Yilong Li, Mohammad Alizadeh, and John K. Ousterhout. Homa: A receiver-driven low-latency transport protocol using network priorities.
[30] Jonathan Perry, Amy Ousterhout, Hari Balakrishnan, Deverat Shah, and Hans Fugal. Fastpass: A centralized "zero-queue" datacenter network. In *SIGCOMM 2014*.
[31] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, George Porter, and Alex C. Snoeren. Inside the social network's (datacenter) network. In *SIGCOMM 2015*.
[32] Balajee Vamanan, Jahangir Hasan, and TN Vijaykumar. Deadline-aware datacenter tcp (d2tcp). In *SIGCOMM 2012*.
[33] Haitao Wu, Jiabo Ju, Guohan Lu, Chuanxiong Guo, Yongqiang Xiong, and Yongguang Zhang. Tuning ecn for data center networks. In *CoNEXT 2012*.
[34] Hong Zhang, Junxue Zhang, Wei Bai, Kai Chen, and Mosharaf Chowdhury. Resilient datacenter load balancing in the wild. In *SIGCOMM 2017*.
[35] Jiao Zhang, Fengyuan Ren, Ran Shu, and Peng Cheng. Tfc: token flow control in data center networks. In *EuroSys 2016*.
[36] Yibo Zhu, Haggai Eran, Daniel Firestone, Chuanxiong Guo, Marina Lipshteyn, Yehonatan Liron, Jitendra Padhye, Shachar Raindel, Mohamad Haj Yahia, and Ming Zhang. Congestion control for large-scale rdma deployments. In *SIGCOMM 2015*.