Augmenting Proactive Congestion Control with Aeolus

Shuihai Hu[1], Wei Bai[2], Baochen Qiao[1], Kai Chen[1], Kun Tan[3]
[1]Hong Kong University of Science and Technology [2]Microsoft [3]Huawei

ABSTRACT

Recently, proactive congestion control solutions have drawn great attention in the community. By explicitly scheduling data transmissions based on the availability of network bandwidth, proactive solutions offer a lossless, near-zero queueing network for serving network transfers. Despite the advantages, proactive solutions require an extra RTT to allocate the ideal sending rate for new arrivalfl ows. To resolve this, current solutions let newfl ows blindly transmit *unscheduled packets* in the first RTT, and assign these packets with high priority in the network. The unscheduled packets, however, can cause serious network congestion, resulting in large queue buildups and excessive packet losses.

This paper describes Aeolus, a solution aimed at achieving both: eliminating the one RTT additional delay and preserving all the good properties of proactive solutions. Similar to current solutions, Aeolus lets newfl ows start at the line rate on their arrivals. However, Aeolus introduces a new switch mechanism which allows the switch to selectively drop excessive unscheduled packets once congestion occurs, thus protecting the scheduled packets from the potential queueing delay and packet loss caused by the unscheduled packets. Our simulations with realistic workloads show that Aeolus can significantly speed up smallfl ows, e.g., deliver 55.9% lower 99^{th} percentile completion time, while preserving all the good properties of proactive solutions.

CCS CONCEPTS

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

APNet '18, August 2–3, 2018, Beijing, China

ACM ISBN 978-1-4503-6395-2/18/08…$15.00
https://doi.org/10.1145/3232565.3232567

KEYWORDS

Data Center Networks, Proactive Congestion Control, Selective Dropping

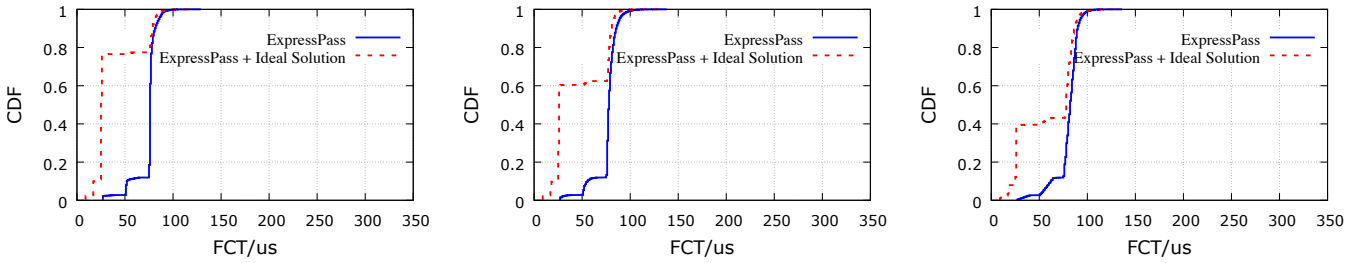
ACM Reference Format:
Shuihai Hu[1], Wei Bai[2], Baochen Qiao[1], Kai Chen[1], Kun Tan[3] . 2018. Augmenting Proactive Congestion Control with Aeolus. In *APNet '18: 2nd Asia-Pacific Workshop on Networking, August 2–3, 2018, Beijing, China.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3232565.3232567

1 INTRODUCTION

Many cloud applications, such as web search, social networking, and retail recommendation impose the stringent latency requirement to the underlying data center networks (DCNs). In recent years, the link speed of DCNs significantly increases, from 1Gbps to 10Gbps, to 40/100 Gbps with 200 Gbps on the horizons [13]. Given this trend, it is more and more challenging for traditional *reactive* congestion control algorithms, e.g., TCP, to meet the demanding latency requirement. Reactive congestion control algorithms only react after congestion already happens. However, at high speed DCNs, network transfersfi nish in much fewer RTTs, thus having little time to react to congestion signals.

Realizing this, *proactive* congestion control mechanisms [7, 9, 11, 17] have drawn great attention in recent years. Unlike reactive solutions, proactive algorithms proactively avoid congestion by explicitly scheduling packet transmissions based on the availability of network resources. It has been widely demonstrated that, proactive algorithms can achieve zero packet loss, ultra-low buffer occupancy and fast convergence while supporting various bandwidth allocation policies.

Despite above advantages, proactive solutions require an extra RTT to allocate the ideal sending rate for new arrival flows. As a result, allfl ows including single-packet small ones are delayed by one RTT, even when the network is very idle. To resolve this, some of the existing solutions [9, 11, 16] let newfl ows blindly transmit *unscheduled packets* in thefirst RTT, and assign these packets with high priority in the network. The unscheduled packets, however, can cause serious network congestion, resulting in large queue buildups and excessive packet losses. The congestion not only degrades



(a) Cache Follower (b) Web Server (c) Data Mining

Figure 1: FCT of 0-100KBfl ows under ExpressPass and ideal solution.

the tail performance of smallfl ows[1], but also violates the desired bandwidth allocation policy, e.g., per-flow fair sharing, weighted fair sharing among applications.

Motivated by above problems, we present Aeolus, a solution aimed at achieving both: eliminating the one RTT additional delay and preserving all the good properties of proactive solutions. The key idea of Aeolus is to let newfl ows only utilize the *spare* bandwidth for thefi rst RTT transfers. More specifically, Aeolus lets newfl ows start at the line rate on their arrivals. When the queue is built at the switch, it indicates that the *unscheduled packets* sent in thefi rst RTT utilizes *excessive* bandwidth and causes congestion. To preserve proactive solutions' original properties, Aeolus leverages the switch to selectively drop excessive unscheduled packets packets. Thus, the aggressive unscheduled traffic does not adversely affect the scheduled traffic. Furthermore, the selective dropping feature can be easily implemented using ECN/RED, a built-in function of most commodity switches.

While the selective dropping feature shields existingfl ows from queueing delay and packet loss, fast start could turn out being too aggressive and cause newfl ows to suffer severe packet loss. Hence we also design a quick recovery scheme for recovering the lost unscheduled packets.

We have implemented Aeolus on top of ExpressPass's open source code [3] with NS-2 simulator. Our preliminary simulation results with realistic workloads [5, 10, 18] indicate that, Aeolus can significantly speed up smallfl ows while well preserve proactive approaches' original properties. For example, under Cache Follower workload, Aeolus facilitates nearly 60% of 0-100KB smallfl ows to complete one RTT faster and reduces the tailfl ow completion time by 55.9% at the 99^{th} percentile, while keeps the low buffer occupancy like ExpressPass.

The rest of this paper is organized as follows. Section 2 introduces some observations that motivates this paper. Section 3 describes the design of our solution in detail. Section 4 presents some preliminary simulation results. Section 5 discusses some related works. Section 6 concludes this paper as well as mentions some future works.

[1]To minimize the impact of packet losses, NDP [11] trim packet headers at the switch to achieve fast loss recovery.

	Cache Follower	Web Server	Data Mining
1Gbps	8.0 RTT	11.1 RTT	18.9 RTT
10Gbps	2.7 RTT	3.0 RTT	4.2 RTT
100Gbps	2.1 RTT	2.1 RTT	2.2 RTT

Table 1: Average FCTs of 0-100KBfl ows under ExpressPass.

2 MOTIVATION

2.1 High Speed Network Needs Fast Start

As 40/100 Gbps networks become prevalent in production data centers, network transfers complete in much fewer RTTs. In Table 1, we measured the average FCTs of 0-100KB smallfl ows with different link speeds. We ran the experiment using ns-2 simulator, and choose ExpressPass for congestion control — a credit based proactive algorithm that forbids data transfer in thefi rst RTT. Flows are generated according to three realistic workloads including Cache Follower, Web Server and Data Mining. The topology we use is a fat-tree with 8 spine switches, 16 leaf switches, 32 top-of-rack (ToR) switches and 192 servers. The average load at bottleneck links is 0.4.

As we can see from Table 1, under 1Gbps networks, on average it takes around 8-19 RTTs for smallfl ows tofinish, hence it may not be a big concern to forbid packet sending in thefi rst RTT. However, under 100Gbps networks, on average only around 2 RTTs are needed for smallfl ows to complete. This indicates that wasting thefi rst RTT can prolong the FCTs of smallfl ows by up to 50% in high speed networks.

To illustrate the potential performance benefit fast start can bring, we ran a second experiment which assumes a hypothetical but idealfi rst-RTT solution. This ideal solution can instantly know the accurate spare bandwidth on every

network path, and allocate them to newfl ows without any delay. The FCTs of 0-100KBfl ows under ExpressPass and "ExpressPass + Ideal Solution" are plotted in Figure 1.

As we can see from Figure 1(a), without allowing new flows to utilize the spare bandwidth in thefi rst RTT, nearly 80% of 0-100KB smallfl ows take one extra RTT to complete - thus their FCTs are prolonged by nearly 75%. Similar improvements can be observed for all the three workloads. The results indicate that, the waste of thefi rst RTT will significantly downgrade the smallfl ow performance.

2.2 The Challenges of Fast Start

While fast start can bring significant performance improvement for proactive algorithms, there are several challenges to be addressed.

Challenge #1: queue buildup and packet loss. *Losslessness* and *low buffer occupancy* are two desired features provided by proactive algorithms. To guarantee these two features, it is a must for proactive algorithms to enforce a (very) tight control over network transfers, under which no sender injects any unscheduled data packets into the network before credit is granted.

Assigning each newfl ow a limited budget of "free credit" is a way to make a tradeoff between fast start and ensuring zero data loss and low buffer occupancy. However, the bursty nature of data center traffic makes it difficult to decide the amount of data a newfl ow can send in thefi rst RTT. If new flows are too conservative, it may under-utilize the available bandwidth and delay the completion of smallfl ows; While being too aggressive may affect the scheduled packets, and lead to large queue buildups and severe packet loss.

Challenge #2: violation of policy goals. Today's DCNs are typically shared by a rich mix of services and applications, each of which may desire a different performance metric. Motivated by this, many proactive algorithms are designed to offer algorithmicfl exibility in achieving various policy goals, such as Shortest Job First (SJF), Weighted Fair Sharing (WFS), etc..

Fast start may violate these policy goals. The reason is as follows. To meet a certain policy goal, link bandwidth needs to be allocated tofl ows in a way that follows the policy-induced allocation decisions. For example, with WFS policy, link bandwidth should be divided amongfl ows with respect to their weights. However, fast start allows newfl ows to generate a lot of packets that is unexpected. These packets may consume the bandwidth already allocated to some existing flows — thus resulting in the violation of policy goals.

2.3 Compatibility with Existing Commodity Switches

To deliver high performance at low cost, most of today's data center switches arefi xed-function switches built with specialized hardware, which offers only very rigid functionality. While it is possible to add new features by redoing the chip. it ofen takes a few years to launch new switch chips and the cost can be very expensive. Hence to be readily-deployable, the solution should be implementable with commodity switches.

3 SOLUTION

In this section, we describe the design of Aeolus, which aims to achieve two goals: (1) allowing newfl ows to sent packets freely in thefi rst RTT; (2) preserving all the good features of proactive algorithms, including zero packet loss, low buffer occupancy, fast convergence, andfl exibility in supporting various policy goals.

3.1 Design Rationale and Basic Idea

Wefi rst introduce the key idea of Aeolus. As discussed above, it is very difficult to calculate a *right* sending rate for the unscheduled packets. Hence, in Aeolus, newfl ows start at line rate. Such aggressivefl ow burst unavoidably leads to large buffer occupancy and severe packet loss under heavy load. To handle this problem, we seek network support instead of pure endhost solution.

To ensure low buffer occupancy, we can drop incoming packets once a switch queue starts to build up. While this avoids queue buildup, the drop of scheduled packets violates the losslessness property of proactive algorithms. To avoid this, we assign a higher drop priority for unscheduled packets than the scheduled packets, so the scheduled packets will not get dropped due to congestion caused by fast start.

The drop of unscheduled packets, by itself, is not a problem. In our design, the mission of these packets is to utilize the *spare* bandwidth. They can get dropped when there is no spare bandwidth. The drop of unscheduled packets becomes a problem only when it takes a long time for them to recover. Hence we complement our design with a fast loss recovery scheme for recovering lost unscheduled packets.

3.2 Switch Mechanism

Aeolus employs a selective dropping mechanism at switches. It distinguishes between packets based on the tag value (e.g., DSCP) carrying in the packet header. For each switch queue, when its queue length reaches the dropping threshold K, the incoming packets carrying low priority[2] tag will get

[2]Low priority is equivalent to high drop priority. We use them interchangeably in this paper.

dropped. In contrast, packets carrying high priority tag only get dropped when the buffer overflows. By tagging *unscheduled packets* with low priority and *scheduled packets* with high priority, this mechanism ensures that an aggressive fast start does not cause any drop of scheduled packets even under the heavy load.

This mechanism requires no change to the packet scheduling of the switch. Unless required by the proactive algorithm, we assume all data packets are transmitted using a single FIFO queue. This means that low-priority packets could still result in queuing delay for high-priority packets. To resolve this, we choose a very small dropping threshold, e.g., 2-8 KB, to avoid large queue buildups due to the buffering of excessive low-priority packets.

While the above mechanism is appealing, the key question is how to implement it with existing commodity switches? In the following, we introduce our approach that exploits existing functionality supported by commodity switches to implement the above switch mechanism.

ECN-based Implementation. We leverage the ECN marking function, which is well supported by data center switches. ECN mechanism uses the ECTfi eld in the IPv4 or IPv6 header to encode whether a packet is ECN capable. At the sender side, a packet will be marked with ECT(1) if it is ECN capable, otherwise with ECT(0). A single marking threshold K is set at each switch queue. In our past testbed experiments with ECN, we made an interesting observation. Let's assume the queue occupancy is greater than K on the arrival of a packet. Ourfi nding is that, this packet will get ECN marked if it is ECN capable; whereas this packet will get dropped if it is not ECN capable.

Therefore, we can implement the selective dropping mechanism by reinterpreting the ECN marking function as follows. At the sender side, we set the ECTfi eld of all unscheduled packets to be ECT(0), while scheduled packets to be ECT(1). At the switches, we set the ECN marking threshold K to be the dropping threshold we want to configure. Then when the marking threshold is reached at a switch queue, unscheduled packets will be directly dropped while scheduled packets will enter the queue and get ECN marked. At the receiver side, we simply ignore the ECN marking of the received packets.

Why not priority queueing? One alternative idea is to leverage priority queueing at switches to prioritize scheduled packets over unscheduled packets. However, existing commodity switches only support a limited number of queues, e.g, 4-8. In production data centers, these queues are typically used to isolate different service classes, such that traffic belonging to different classes can be kept from impacting each other. The priority queueing based solution would require one additional queue for each service class, such that the number of supported service classes will be reduced by half.

In addition, due to the queueing delay in the low-priority queue, for the samefl ow, packets sent in the second RTT may be received earlier than those unscheduled packets sent in thefi rst RTT. This will lead to packet out-of-order problem.

3.3 Endhost Design

3.3.1 Duration of Fast Start. Afl ow enters fast start phase on its initiation. Fast start ceases at the end offi rst RTT when any response is received, either from the receiver or the central arbiter (in the case a centralized proactive algorithm is adopted). The response can be data ACKs, bandwidth allocations, loss notifications, etc.. Packets sent during the fast start phase are marked as low priority, while other packets are marked as high priority.

In production data centers, applications may maintain persistent connections for transmitting multiple short messages for a long time. After the termination of fast start phase, these connections will be unable to send packets during the first RTT since the second message. To address this, Aeolus periodically renew afl ow once afl ow idles for a period.

3.3.2 Loss Recovery for Unscheduled Packets. Fast start could turn out being too aggressive in the case when a large number offl ows arrive in a burst. The selective dropping mechanism protects scheduled packets from large queueing delay and loss in such a situation. However, a newfl ow that performs fast start could itself suffer from excessive packet loss. Without a quick loss recovery scheme, a newfl ow may end up with worsefl ow completion time than if fast start had not been used at all. In this part, we present our design for quickly recovering the lost unscheduled packets.

Loss detection. Aeolus enables per packet ACK at the receiver to quickly notify the sender which packets have been received, but only for packets sent during fast start phase. For each returned ACK, it carries the sequence number of the last received packet, instead of repeating the smallest sequence number of expected data.

In addition to the normal data ACK, Aeolus uses a probe mechanism for detecting tail loss quickly. The idea is to send a probe packet right after the transmission of last unscheduled packet. This probe packet carries the sequence number of last sent packet, and is of minimum ethernet size, i.e., 64 bytes. We tag this probe packet with high priority, such that it can be received by the receiver even if all the unscheduled packets are dropped in the network due to severe congestion.

Both data ACK and probe ACK are tagged with high priority to ensure reliable acknowledgement. Based on the received ACKs, senders can explicitly infer the lost packets including tail loss quickly.

Loss retransmission. Retransmitting lost unscheduled packets using unscheduled packets could result in a very slow loss recovery procedure, as retransmitted packets may

	Web Server	Cache Follower	Web Search	Data Mining
0 - 100KB	81%	53%	52%	83%
100KB - 1MB	19%	18%	18%	8%
> 1MB	0%	29%	20%	9%
Averagefl ow size	64KB	701KB	1.6MB	7.41MB

Table 2: Flow size distribution of realistic workload.

get lost again in the network. To avoid this, Aeolus performs retransmission only with scheduled packets.

Note that in existing proactive solutions, the bandwidth allocated to afl ow has an effective period. It is possible that in the second RTT the sender fails to receive the loss signal prior to the expiration of received bandwidth allocations. As a result, the allocated bandwidth will be wasted (assuming there is no more new data to send in the second RTT).

To handle the above problem, Aeolus employs *proactive retransmission* at the sender, i.e., let a sender, upon receiving bandwidth allocations, retransmit corresponding amount of unacked unscheduled packets immediately even before detecting the loss. Note that a sender performs proactive retransmission only when there is no new and loss-confirmed data buffered. In addition, to reserve more time for ACKs to return, the proactive retransmission can be done in a reverse order (from the last byte towards thefi rst byte).

4 SIMULATION RESULTS

We have implemented Aeolus on top of ExpressPass's open source code [3] with NS-2 simulator. In the next, we will present our preliminary results for evaluating the performance improvement when augmenting ExpressPass with Aeolus.

Topology: Same as [7], we simulate a oversubscribed fat-tree topology with 8 spine switches, 16 leaf switches, 32 top-of-rack (ToR) switches and 192 servers. Hence, the oversubscription ratio is around 3 : 1. All links have 100Gbps capacity. The per-link propagation delay and end-host processing delay are 4μs and 1μs, respectively. Hence, the based fabric latency across the spine switch is ~52μs. We employ Equal Cost Multi Path (ECMP) with path symmetry for fabric load balancing. Each switch port has 1.5MB buffer to store data packets.

Workloads: Same as [7], We generate realistic workloads according to four distributions: Web Server [18], Cache Follower [18], Web Search [5] and Data Mining [10]. We plot flow size distributions in Table 2. All the distributions are highly-skewed: the most of bytes are from few largefl ows. Thefl ows arrive according to the Poisson process. The source

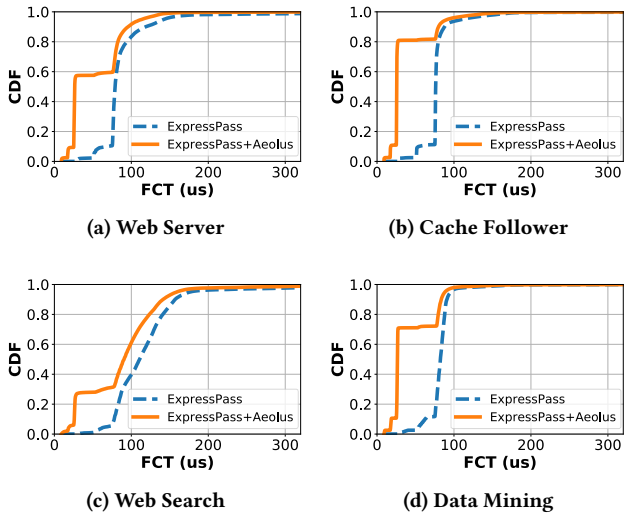


Figure 2: FCT of 0-100KBfl ows in a oversubscribed fat-tree topology. The average load of the network core is 40%.

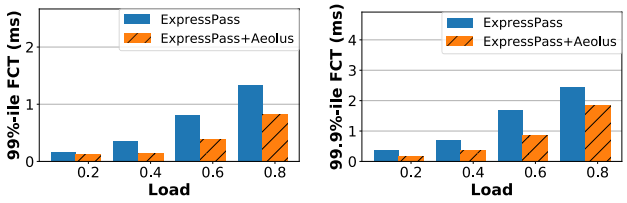
and the destination of afl ow is chosen randomly. We control the averagefl ow arrival interval to achieve the desired load (utilization) in the fabric core (spine-leaf links). We run 100,000fl ows for each simulation setting.

Performance metrics: We usefl ow completion time (FCT) as the main performance metric. We also measure the queue length for further analysis.

Result analysis: In Figure 2, we plot the FCT distributions of 0-100KB smallfl ows. The network load is 40%. We have the following two observations. First, with Aeolus, nearly 60%, 80%, 28% and 70% of 0-100KB smallfl ows complete within the first RTT for the four workloads, respectively. This indicates that Aeolus can fully utilize the spare bandwidth in thefirst RTT to speed up smallfl ows. Second, the tail FCTs with both schemes are similar. This indicates that under Aeolus, the unscheduled packets has little negative impact on scheduled packets.

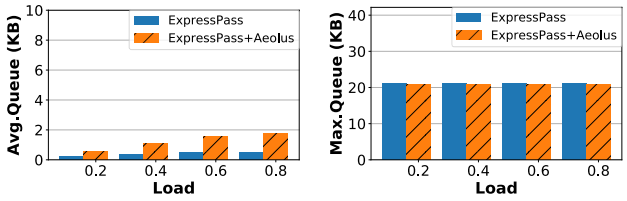
In Figure 3, we plot the 99%-ile FCT and 99.9%-ile FCT of 0-100KBfl ows with the varying load for the web server workload. Across all loads, Aeolus improves the FCT of 0-100KBfl ows by 19.6%-55.9% at the 99th percentile, and by 1.6%-18.8% at the 99.9th percentile, respectively. The results further confirm that Aeolus'sfi rst RTT bursts do not degrade the tail FCT of smallfl ows. Instead, better tail FCT can be achieved for smallfl ows due to the fastfl ow start. We also achieve the similar results under the other three workloads. But we omit them in the interest of the space.

In Figure 4, we show the average and maximum queue occupancy measured during the simulation. On average, both



(a) 99th percentile (b) 99.9th percentile

Figure 3: Tail FCT of 0-100KBfl ows for the web server workload.



(a) Average queue length. (b) Maximum queue length.

Figure 4: Average and maximum queue length for the web server workload.

schemes achieve very small buffer occupancies (< 550 bytes under ExpressPass, and < 2KB under Aeolus). Furthermore, both schemes' maximum queues are small and stable with the increased load. The results indicate that Aeolus keeps the low buffer occupancy like ExpressPass.

Summary: Through large-scale simulations, wefi nd that Aeolus generally outperforms ExpressPass. With fast start, Aeolus significantly speeds up smallfl ows while not leads to large queue buildup.

5 RELATED WORK

DCTCP [5] is an ECN-based transport which has been widely adopted by the industry [1, 2, 14]. It uses instantaneous ECN marking at the switch to detect congestion and adjusts the window according to the extent of congestion at the end host. However, to balance throughput and latency, DCTCP still keep moderate buffer occupancies at the switch. When there is a extremely large number (e.g., hundreds of) of concurrent connections, DCTCP may fail to keep the low buffer occupancies, resulting in high queueing delay and excessive packet losses [5, 14]. In addition, DCTCP suffers from slow convergence speed in high speed networks, which may take hundreds of RTTs to reach ideal sending rate.

There is a large body of work built on the top of DCTCP. D^{2}TCP [19] adds deadline information into DCTCP's window adjustment function. AC/DC [12] and vCC [8] enforce DCTCP algorithm in the hypervisor to benefit tenant virtual machines in the public cloud. HULL [6] uses phantom queue at the switch to deliver congestion notification before queue build-ups. It sacrifices some throughput to achieve near zero queueing delay.

DCQCN [21] and TIMELY [15] are congestion control solutions for RDMA over Converged Ethernet (RoCE). Unlike DCTCP, even with a large number of concurrent connections, both schemes can avoid congestion packet losses with PFC [4]. However, PFC may cause large queue buildups under heavy load. In addition, PFC is known to have several important drawbacks, including deadlock, head-of-line blocking, unfairness and congestion spreading.

In recent years, several proactive congestion control algorithms [7, 9, 11, 13, 17, 20] have been proposed, such as FastPass [17], ExpressPass [7] and TFC [20]. Aeolus can be integrated with all these proactive algorithms to assist them to make a sufficient use of spare bandwidth in thefi rst RTT without violating their desired features.

6 CONCLUSION

This paper presented Aeolus, a simple yet effective solution that augments all existing proactive algorithms to make a sufficient use of spare bandwidth for thefi rst RTT transfers without violating the good features of proactive approaches. By differentiating traffic at the end and enforcing selective dropping in the network, Aeolus allows aggressive fast start without affecting the traffic scheduled by proactive congestion control algorithms. Our preliminary simulation results indicate that, Aeolus can significantly speed up smallfl ows while keeps low buffer occupancy and zero loss for scheduled packets.

In the next, we will focus on implementing our solution with commodity servers and switches in a real testbed environment. In addition to ExpressPass, we will integrate Aeolus with more proactive approaches, and demonstrate that Aeolus can generally improve their performance while preserving all the good features.

REFERENCES

[1] Dctcp in linux kernel 3.18. http://kernelnewbies.org/Linux_3.18.
[2] Dctcp in windows server 2012. http://technet.microsoft.com/en-us/library/hh997028.aspx.
[3] Expresspass simulator. https://github.com/kaist-ina/ns2-xpass.
[4] Ieee dcb. 802.1qbb - priority-basedfl ow control. http://www.ieee802.org/1/pages/802.1bb.html.
[5] 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*.
[6] Mohammad Alizadeh, Abdul Kabbani, Tom Edsall, Balaji Prabhakar, Amin Vahdat, and Masato Yasuda. Less is more: trading a little bandwidth for ultra-low latency in the data center. In *NSDI 2012*.
[7] Inho Cho, Keon Jang, and Dongsu Han. Credit-scheduled delay-bounded congestion control for datacenters. In *SIGCOMM 2017*.

[8] Bryce Cronkite-Ratcliff, Aran Bergman, Shay Vargaftik, Madhusudhan Ravi, Nick McKeown, Ittai Abraham, and Isaac Keslassy. Virtualized congestion control. In *SIGCOMM 2016*.

[9] 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*.

[10] Albert Greenberg, James R. Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A. Maltz, Parveen Patel, and Sudipta Sengupta. Vl2: a scalable andfl exible data center network. In *SIGCOMM 2009*.

[11] 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*.

[12] 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*.

[13] Lavanya Jose, Lisa Yan, Mohammad Alizadeh, George Varghese, Nick McKeown, and Sachin Katti. High speed networks need proactive congestion control. In *HotNets 2015*.

[14] Glenn Judd. Attaining the promise and avoiding the pitfalls of tcp in the datacenter. In *NSDI 2015*.

[15] 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*.

[16] Behnam Montazeri, Yilong Li, Mohammad Alizadeh, and John K. Ousterhout. Homa: A receiver-driven low-latency transport protocol using network priorities. In *SIGCOMM 2018*.

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

[18] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, George Porter, and Alex C. Snoeren. Inside the social network's (datacenter) network. In *SIGCOMM 2015*.

[19] Balajee Vamanan, Jahangir Hasan, and TN Vijaykumar. Deadline-aware datacenter tcp (d2tcp). In *SIGCOMM 2012*.

[20] Jiao Zhang, Fengyuan Ren, Ran Shu, and Peng Cheng. Tfc: tokenfl ow control in data center networks. In *EuroSys 2016*.

[21] 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*.