Low Latency Software Rate Limiters for Cloud Networks

Keqiang He† Weite Qin‡ Qiwei Zhang§ Wenfei Wu$^{\circ}$ Junjie Yang‡ Tian Pan‡
Chengchen Hu§ Jiao Zhang‡ Brent Stephens† Aditya Akella† Ying Zhang$^{\diamond}$

†UW-Madison ‡BUPT §XJTU $^{\circ}$Tsinghua $^{\diamond}$Facebook

ABSTRACT

A lot of recent work has focused on reducing in network queueing latency in datacenter networks. In this paper, we focus on a less explored topic — latency increases caused by queueing in rate limiters on the end-host. First, we show that latency can be increased by an order of magnitude by rate limiters in cloud networks. To solve this problem, we extend ECN marking into rate limiters and use a datacenter congestion control algorithm — DCTCP. Unfortunately, while this reduces latency, it also leads to throughput oscillation. Thus, this solution is not sufficient. In this paper, we also analyze the specific reasons that ECN marking in software rate limiters leads to the throughput oscillation problem. Finally, we propose two potential solutions to design software rate limiters that can achieve stable high throughput and low latency.

CCS CONCEPTS

• **Networks** → **Transport protocols**;

KEYWORDS

Rate Limiter, Latency, Performance

ACM Reference format:
Keqiang He† Weite Qin‡ Qiwei Zhang§ Wenfei Wu$^{\circ}$ Junjie Yang‡ Tian Pan‡ Chengchen Hu§ Jiao Zhang‡ Brent Stephens† Aditya Akella† Ying Zhang$^{\diamond}$. 2017. Low Latency Software Rate Limiters for Cloud Networks. In *Proceedings of APNet'17, Hong Kong, China, August 03-04, 2017,* 7 pages.
DOI: http://dx.doi.org/10.1145/3106989.3107005

1 INTRODUCTION

The ability to create bandwidth allocations is an indispensable feature of multi-tenant clouds. Bandwidth allocations can be used to provide bandwidth reservations to a tenant or to guarantee that network bandwidth is fairly shared between multiple competing tenants [18, 21, 22]. Bandwidth allocations are often implemented with *software rate limiters*[1] running in the hypervisor or operating system of the end-hosts attached to the network (e.g., Linux Hierarchical Token Bucket, aka HTB). This is because software rate limiters are flexible and scalable.

Unfortunately, typical software rate limiters (e.g., HTB) also increase network latency by adding an additional layer of queuing for packets. To be able to absorb bursts of incoming packets while also ensuring that network traffic does not exceed the configured rate, rate limiters maintain a queue of packets to send and control the speed at which packets are dequeued into the network. This queuing introduces additional network latency. For example, in our experiments, we find that software rate limiting (HTB) increases latency by 1-3 milliseconds across a range of different environment settings. This increase in latency is about an order of magnitude higher than the base latency of the network (200 us) [14]. In multi-tenant clouds, this additional queuing latency can increase flow completion times, leading to unacceptable service-level agreement (SLA) violations [16, 24, 25].

Inspired by recent work that reduces queuing delay for in-network devices like switches [6, 6, 15, 19, 27], in this paper, we explore how to use a congestion-control-based approach to address the latency issues associated with using software rate limiters. As a promising first step, we find that the existing datacenter congestion control protocol DCTCP [6] can be used to reduce the latency incurred by rate limiters. Unfortunately, we find that a straightforward application of DCTCP to software rate limiters also hurts throughput. In this paper, we also identify two problems unique to end-hosts that hurt the throughput of DCTCP and propose potential solutions to these problems.

[1]In this paper, we use software rate limiter and rate limiter interchangeably.

In order to use DCTCP to try to reduce end-host queuing latency, we implement software rate limiters that perform ECN marking and configure the end-host in DCTCP mode. This causes the end-host to use the fraction of ECN-marked packets to adjust congestion window (CWND). Unfortunately, we find this naive application of DCTCP introduces a significant performance problem — TCP throughput tends to oscillate (between 50% to 95% in some cases, see Section 3). This reduction in throughput is unacceptable in cloud networks because it could significantly degrade the performance of some applications.

We have identified two issues that arise with simply applying DCTCP+ECN on end-host rate limiters: Firstly, TCP segmentation offload (TSO) causes ECN to be applied at a coarse granularity. Secondly, the latency of the congestion control loop latency is too high.

Different from hardware switches in the network, end-hosts process TCP segments instead of MTU-sized packets. TCP Segmentation Offload (TSO) [3] is an optimization technique that is widely used in modern operating systems to reduce CPU overhead for fast speed networks. Because Linux has difficult driving 10Gbps (and beyond) line-rates when TSO is not enabled, Linux uses a TSO size of 64KB by default. That means that marking the ECN bit in one 64KB segment causes 44 consecutive MTU-sized packets to have the ECN bit marked. This is because the ECN bits in the segment header are copied into each packet by the NIC. Oppositely, if a TCP segment is not marked, none of the packets in this segment is marked. This kind of coarse-grained segment-level marking leads to an inaccurate estimation of congestion level which consequently leads to throughput oscillation.

The second problem with DCTCP+ECN is that the ECN mark takes one round-trip time (RTT) to get back to the source. Because of this, the congestion window computation at the source uses a stale value from one RTT ago. As a result, congestion cannot be detected at early stage, and the congestion level would be exacerbated during this one-RTT delay.

Due to the insufficiencies of existing solutions (Linux HTB) and the strawman solution (DCTCP+ECN), we aim to design a software rate limiter for datacenter networks. It should satisfy four requirements: high bandwidth saturation, low latency, low throughput oscillation, and generic (i.e., the ability to handle both ECN flows and non-ECN flows). We propose two potential methods for these four requirements. The first method refines the ECN marking mechanism, instead of marking outgoing TCP segments, it directly marks ECE bit of ACK packets on the reverse path. This method avoids marking large TCP segments and also reduces congestion control loop latency. Thus, it can achieve stable high throughput with low latency. The second method runs a real time queue length based congestion control algorithm out of the VMs and enforces congestion control decisions using (RWND) field in ACK packets. Compared with the first method, the second method is generic and can handle both ECN flows and non-ECN flows. The contributions of this paper are as follows:

- We conduct systematic measurements of an existing software rate limiter (i.e., Linux HTB) and show that it introduces unacceptable latency to cloud networks.
- We implement DCTCP+ECN as a strawman solution to solve the latency problem, and reveal that it leads to the throughput oscillation problem that is specific in end hosts software rate limiters.
- We analyze the design requirements for software rate limiters in cloud networks and propose two potential solutions to achieve the design requirements.

2 BACKGROUND AND RELATED WORK

Hardware/software rate limiters. In order to perform bandwidth allocation, prior work has relied on end-host rate limiting [9, 17, 18]. In this paper, we focus on software rate limiters because they are commonly used method for providing bandwidth allocation in clouds [4]. Compared with hardware rate limiters [20], software rate limiters are more flexible (i.e., can boot up at any server), scalable (i.e., multiple instances can be created for different tenants), and have more functionalities (e.g., hierarchical rate limiting). However, even though we only focus on software rate limiters, hardware rate limiters must also queue data. Because of this, we expect that hardware rate limiters also suffer from throughput versus latency trade-offs that are similar to software rate limiters. We leave an investigation into the performance of hardware rate limiters to future work.

Fundamentally, there are two ways to configure rate limiters: traffic shaping, and traffic policing. In traffic shaping, packets are first pushed into a queue, and then packets are sent to the network based on periodically refreshed tokens assigned to the queue. The queue in traffic shaping works as a buffer which can effectively absorb bursty traffic, avoiding the need to drop packets. Tokens are refreshed at the speed of the configured sending rate, which ensures that outgoing packets are well paced. This avoids sending bursts of packets into the network, which is especially important for modern datacenter networks because the network consists of shallow-buffered switches [23]. In contrast, traffic policing monitors the packet arrival rate and only pushes a packet into the network when traffic rate is not above the desired rate, otherwise the packet is dropped.

In this paper, we only focus on rate limiting through traffic shaping because it is the preferred method of rate limiting for cloud networks [4]. This is because traffic policing drops packets, and prior work has found that packet drops are very harmful to small flows in data center applications [26].

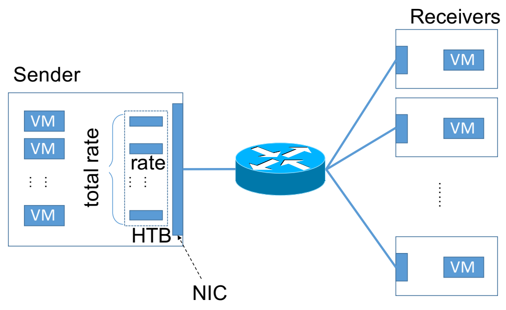

Figure 1: Experiment setup

Low-latency datacenter networks. Some datacenter applications are latency sensitive [5]. To improve performance, recent work has focused on reducing network latency while simultaneously providing high throughput and a low loss rate [6, 8, 13, 15, 19, 27]. This work can be roughly classified into two categories. The first category is congestion control based, where network congestion signal (e.g., ECN) is sent back to the source, and the source run congestion control algorithms to reduce the sending rate, so that queues in networks are drained to reduce queuing delay [6, 27]. The second category is priority based, where flows in high priority queues would be sent first, getting low latency [7, 8, 13].

To solve the latency problems of rate limiters, we use congestion control instead of traffic prioritization. This is for a few reasons. First, priority-based solutions may cause starvation for low-priority flows, and flows with the same low priority still suffer from queuing delay. Second, hypervisors in public clouds are typically agnostic to the priority of tenant traffic. In this scenario, a priority-based solution is not possible.

Summary. Software rate limiters with traffic shaping in clouds cause queuing latency to traversing flows. Existing congestion-control (CC) based low-latency solutions are preferred in clouds and focus on in-network latency. Thus, it is intuitive for us to consider transplanting a CC based low-latency solution to software rate limiters.

3 MEASUREMENT AND ANALYSIS

In this section, we first show that the Linux HTB rate limiter introduces a serious latency problem. Next, we design a strawman solution that uses DCTCP and ECN marking to reduce queuing latency. Finally, we evaluate this strawman solution. Our experiments show that naively applying DCTCP to software rate limiters has a side effect of causing throughput oscillation.

3.1 Performance of Linux HTB

In HTB, packets are classified into traffic classes, and HTB is designed to ensure that a minimum amount of bandwidth is guaranteed to each traffic class. If the required minimum amount of bandwidth is not fully used, the remaining bandwidth is distributed to other classes. The distribution of spare bandwidth is in proportion to the minimum bandwidth specified to a class [2].

To evaluate HTB, we setup servers with 10 Gbps NICs in CloudLab [1]. The experiment setup is shown in Figure 1. In these experiments, we setup multiple VMs and configure a rate limiter for each sender VM. To generate background traffic, we use iperf to send a variable number of flows between each sender-receiver VM pair. We configure HTB to control both the bandwidth for each VM pair and the total sending rate of all VM pairs (i.e., using the hierarchical feature of HTB). With these settings, we run sockperf between sender and receiver pairs to measure TCP RTT.

We conduct two sets of experiments. In the first set of experiments, we have one sender VM and one receiver VM. In these experiments, we set the sender side rate limiter to 1Gbps, 2Gbps, 4Gbps and 8Gbps, and we vary the number of iperf flows from the sender to receiver (1, 2, 8 and 16). In the second set of experiments, we configure two rate limiters on the sender server and setup two VMs (one rate limiter for each VM). We configure the minimum rate of each rate limiter to 2Gbps, 3Gbps, 4Gbps and 5Gbps, and we configure the total rate of the two rate limiters to always be 10Gbps.

The experimental results are shown in Table 1 and 2. In all the experiments, network bandwidth is saturated. The total throughput is between 91–96% of the configured rate limit. Note that in Table 2, if the sum of individual rate limiter's rate is smaller than the configured total rate, HTB would allow all flows to utilize and compete for the spare bandwidth. Surprisingly, using more flows in these experiments leads to lower bandwidth saturation. This is because there is more competition between flows, which leads to throughput oscillation.

To better understand the scenario where there is only one receiver VM, we visualize the TCP RTT results in Figure 2. In this figure, each subfigure shows the CDF of the sockperf RTT measurements given different rate limits. Different subfigures show the results for a different number of background iperf flows. We can draw three conclusions from these results. First, TCP RTT increases dramatically when packets go through a congested rate limiter. In the baseline case where no HTB is configured and no iperf background flow running, the median TCP RTT is 62us. In contrast, with one background iperf flow and rate limits from 1Gbps to 8Gbps, the median RTT increases to 957us-1583us, which is 15-25X larger compared with the baseline case. Second, TCP RTT increases as the number of background flows running increases. For example, with a 1Gbps rate limit, the median RTT is 957us for one background flow and 3192us for 16 background flows. Third,

Table 1: HTB experiments for one receiver VM

numReceiver	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
numFlows/receiver	-	1	1	1	1	2	2	2	2	8	8	8	8	16	16	16	16
rate/receiver (Gbps)	-	1	2	4	8	1	2	4	8	1	2	4	8	1	2	4	8
total rate (Gbps)	-	1	2	4	8	1	2	4	8	1	2	4	8	1	2	4	8
total tput (mbps)	-	951	1910	3820	7380	945	1910	3820	7500	958	1915	3827	7627	960	1920	3829	7655
b/w saturation (%)	-	95.1	95.5	95.5	95.3	94.5	95.5	95.5	93.8	95.8	95.8	95.7	95.3	96	96	95.7	95.7
50% RTT (us)	116	957	883	643	1583	1513	1078	1047	853	2316	1766	1529	1110	3192	2373	1880	1262
99.9% RTT (us)	237	1115	1000	706	1673	1701	1203	1132	933	2527	1939	1637	1208	3320	2511	2016	1486

Table 2: HTB experiments for two receiver VMs

numReceiver	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
numFlows/receiver	-	1	1	1	1	2	2	2	2	8	8	8	8	16	16	16	16
rate/receiver (Gbps)	-	2	3	4	5	2	3	4	5	2	3	4	5	2	3	4	5
total rate (Gbps)	-	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
total tput (mbps)	-	9420	9420	9410	9410	9410	9420	9420	9420	9224	9392	9321	9401	9182	9161	9225	9296
b/w saturation (%)	-	94.2	94.2	94.1	94.1	94.1	94.2	94.2	94.2	92.2	93.9	93.2	94.0	91.8	91.6	92.3	93.0
50% RTT (us)	118	475	797	1515	1658	699	916	1006	1036	1512	1407	1410	1587	2023	2040	2064	1986
99.9% RTT (us)	135	551	849	1626	1751	983	989	1102	1115	1697	1673	1532	1768	2147	2182	2185	2100

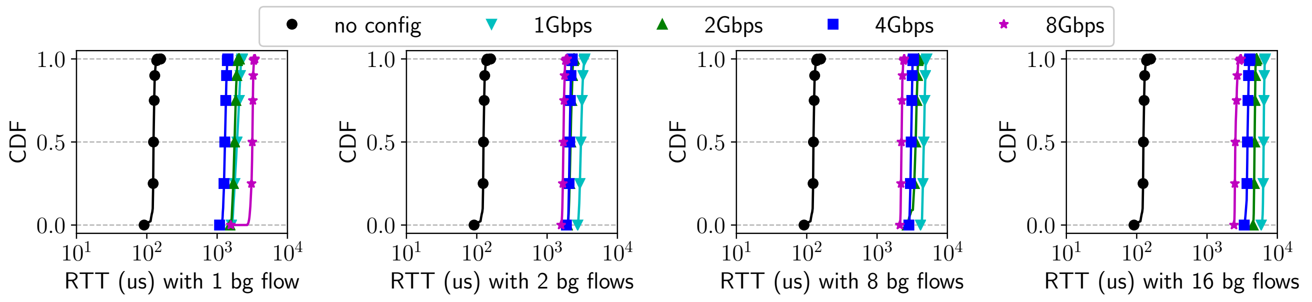

Figure 2: HTB experiment: one receiver VM, varying rate limiting and number of background flows

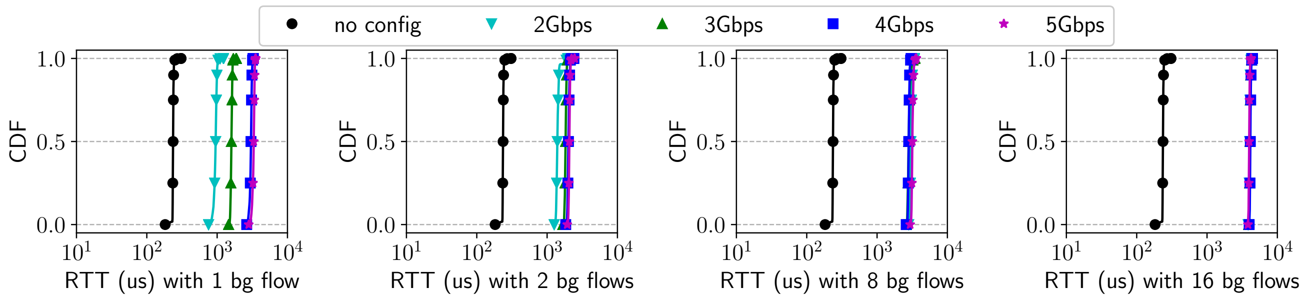

Figure 3: HTB experiment: two receiver VMs, varying rate limiting and number of background flows

TCP RTT decreases as the configured rate limiter speed increases[2]. This is because rate limiter speed determines the dequeue speed of the HTB queue. Thus, with a higher rate limit, the queue tends to be drained faster.

In the experiments with two receiver VMs (Figure 3), we can draw the same conclusions regarding RTT increase and the impact of the number of background flows. The main difference in these experiments is that TCP RTT increases as the configure rate limit increases. This is because we did not constrain the total transmission rate in these experiments,

[2]The only exception is the case with one background flow and 8Gbps rate limiting. We suspect that this is caused by experiment noise.

which allows HTB to utilize spare bandwidth. Thus, the dequeue speed is constant in each figure (10Gbps/numFlow). One possible reason for the RTT trend is that enqueue speed is higher when the rate limiter speed is higher. For a fixed dequeue speed, larger enqueue speed implies higher latency.

3.2 Strawman Solution: DCTCP + ECN

Inspired by solutions that reduce queuing latency in switches [6], we test using DCTCP+ECN as a strawman solution to reduce rate limiter latency. Specifically, in our strawman solution, we implement ECN marking in Linux HTB queues and enable DCTCP at the end-points. For ECN marking, there is a tunable parameter — *marking threshold*. When the queue length exceeds the marking threshold, all enqueued packets would have their ECN bits set; otherwise, packets are not modified. DCTCP then reacts to ECN marking and adjusts sender's congestion window based on the ratio of marked packets [6].

To evaluate DCTCP+ECN, we setup experiments with one sender and one receiver. In these experiments, we configure the HTB rate limit to be 1Gbps and 2Gbps, and we vary the ECN marking threshold. The results from this experiment are shown in Figure 4. We observe that TCP RTT can be reduced significantly (<1ms) by extending ECN into rate limiter queues. For example, with the marking threshold set to 60KB, median TCP RTT is 224us (1Gbps case in Figure 4), which is less than 1/4 of the native HTB RTT (957us). Further, a smaller ECN marking threshold can achieve even lower latency — with the threshold from 100KB to 20KB, median TCP RTT is reduced from 375 us to 93us.

While latency can be improved, we observe that DCTCP+ECN can have negative effect on throughput, which is shown in Figure 4. TCP throughput appears to have large oscillation, which implies that applications cannot get constantly high throughput and that the available bandwidth is not fully utilized. For example, with a 2Gbps rate limit (the 2Gbps case in Figure 4), even when we set the marking threshold to be 100KB (much larger than the best theoretical value recommended by Alizadeh *et al.* [6]), there is still occasional low throughput (e.g., 1000Mbps) within a 20-second experiment duration.

4 ANALYSIS AND POTENTIAL DESIGN

Analysis of throughput oscillation. There are two reasons that directly applying the standard ECN marking approach to rate limiter queues causes TCP throughput oscillation. First, end-host networking stacks enable optimization techniques such TSO (TCP Segmentation Offload [3]) to improve throughput and reduce CPU overhead. By default, the maximum TCP segment size is 64KB. Because of TSO, the end-host networking stack (including the software rate limiters) processes TCP segments instead of MTU-sized packets. In order to implement TSO, a NIC copies a segment's IP header into each MTU-sized packet generated by the NIC. Because of this, marking one TCP segment in the rate limiter queue results in many consecutive MTU-sized packets being marked. For example, marking a 64KB segment means 44 consecutive Ethernet frames are marked. Such coarse-grained segment-level marking causes the accuracy of congestion estimation in DCTCP to be greatly decreased.

Second, ECN marking happens on the transmitting path, and it takes one RTT for congestion feedback to travel back to the sender before congestion control actions are taken. Moreover, TCP RTT can be affected by the "in network" latency, and "in network" latency can be on the order of milliseconds or even tens of milliseconds [14]. In DCTCP+ECN, this one RTT control loop latency causes the ECN marks to be "outdated", not precisely reflecting the instantaneous queue length when the marks are used for congestion window update in DCTCP. Without a congestion control algorithm that can react to instantaneous queue length, the one-RTT control loop latency exacerbates the incorrect segment-level ECN marking. Thus, congestion window computation in DCTCP tends to change more dramatically, leading to throughput oscillation.

Requirements of high performance software rate limiters. Niether raw HTB nor DCTCP+ECN achieve satisfactory performance. Raw HTB can achieve high and stable throughput but with very high latency. Further, raw HTB has the advantage of being generic in that it can handle both ECN-capable and non-ECN-capable flows. In contrast, HTB with DCTCP+ECN satisfies our low latency requirement, but it cannot achieve stable high throughput and can only handle ECN-capable flows. Therefore, there is a need for better software rate limiters.

Summarizing the problems with raw HTB and DCTCP+ECN as well as considering the practicability concerns that arise in multi-tenant clouds, we list the design requirements of high-performance software rate limiters. First, rate limiters should be able to control configured sending rate and provide *high bandwidth saturation without introducing significant latency*. Second, rate limiters should also overcome the throughput oscillation problem in the strawman solution, achieving *constantly stable throughput*. Violating any of these three requirements would cause SLA violations, degrading the performance of network applications. Third, considering the practicability of the solution, a rate limiter must be *compatible with various TCP flows* (ECN-capable and non-ECN-capable).

Potential solutions. Fortunately, software rate limiters are implemented on the end-host, which gives us opportunities to design and implement better software rate limiters for cloud networks. First, end-host has enough memory to store per-flow information. Second, we have sufficient programmability (e.g., loadable OVS module). For example, we can correlate an incoming ACK with its outgoing queue length, and we

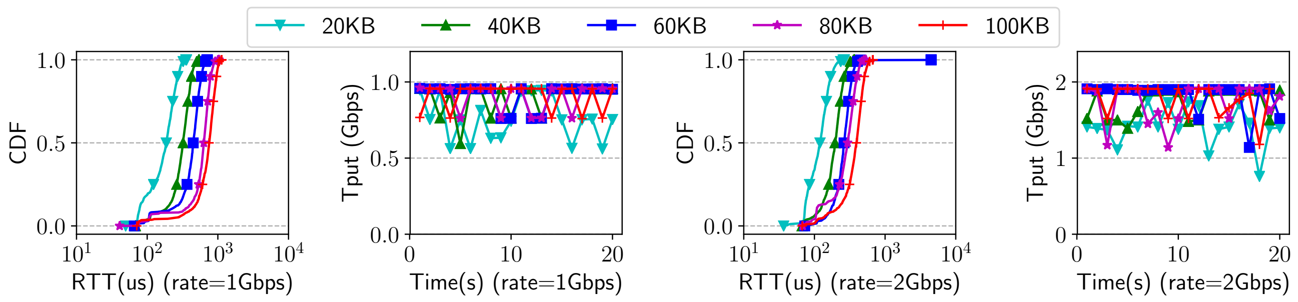

Figure 4: DCTCP experiments, 1 flow, varying threshold, 1Gbps(left) and 2Gbps(right)

can compute per-flow window size and encode it in packets before they arrive at VMs.

We propose the following solutions to satisfy these four requirements: First, to avoid the shortcomings of DCTCP+ECN marking, our key idea is to directly set TCP ECN-Echo (ECE) bit on the reverse path (ACK) based on real-time queue length. This approach avoids the problems caused by performing coarse-grained segment-level marking and allows the congestion control loop latency to be reduced to almost 0.

Second, to make our solution not dependent on tenant VMs or TCP flows being ECN capable, we propose implementing a rate limiter queue length-based congestion control algorithm outside of the VM (e.g., at the virtual switches) and enforcing congestion control decisions using TCP ACK's `RWND` field [12, 15].

Preliminary Results. We implemented the two proposed solutions in Open vSwitch (OVS) and Linux HTB. Our preliminary experimental results show that they are able to achieve both stable high throughput and low latency. Applying the first scheme, we are able to achieve: 950Mbps throughput with median latency of 166us for a 1Gbps rate limiter; 1.88Gbps throughput with median latency of 102us for a 2Gbps rate limiter. Applying the second scheme, we are able to achieve: 956Mbps throughput with median latency of 161us for a 1Gbps rate limiter; 1.90Gbps throughput with median latency of 115us for a 2Gbps rate limiter.

5 DISCUSSION

"Back pressure" is another potential solution to reduce rate limiter queueing latency. It blocks TCP write/send calls when the queue size reaches a small threshold. However, this approach has a few disadvantages. First, back pressure signals should be populated from the rate limiter queue back up to the source TCP stack. Along this path, there are multiple buffers [11]. Adding back pressure signals for all the buffers in the virtualized network stack seems to involve non-trivial engineering efforts. Second, backp ressure schemes need to enqueue at least a single segment from each VM and/or TCP flow to avoid starvation. Given that segments may be large, even allowing a single segment per-VM/flow can lead unacceptable latencies.

The two potential solutions proposed in this paper also have a few limitations. First, they can only work with TCP traffic. It has been reported that 99.91% of the traffic in datacenters is TCP traffic [6]. Second, they support nonencrypted TCP traffic and SSL/TLS traffic but not IPSec traffic because they need to inspect and modify TCP headers. Recently, cloud providers like Microsoft proposed to use FPGA to perform computation intensive operations such as encryption and compression [10], which implies that the virtual switches are most likely exposed to non-encrypted traffic.

6 CONCLUSION

Rate limiters are important for network bandwidth sharing and reservation in multi-tenant clouds, but they also introduce large latency to the traversing flows. Thus, in this paper, we systematically quantify the impact of software rate limiter (i.e., HTB) on network latency and demonstrate that network latency can be increased by an order of magnitude or more. To the best of our knowledge, we are the first to perform this quantification. We also measure the performance of a strawman solution — DCTCP with ECN marking in the rate limiter queue, and find that this approach leads to TCP throughput oscillation. We analyze the reasons of the throughput oscillation problem (coarse-grained segment-level ECN marking and long control loop latency) and also point out potential solutions (i.e., direct ECE marking and queue length-based congestion control enforcement via `RWND`).

ACKNOWLEDGEMENT

We thank the anonymous reviewers for their insightful comments and our shepherd Kun Tan. This work is supported by US National Science Foundation grants CNS-1302041, CNS-1330308, and CNS-1345249, National Natural Science Foundation of China (NSFC) grants No. 61502049, 61672425, and 61373002, Microsoft Research Asia Collaborative Research Program 2016JM6066, and China Postdoctoral Science Foundation grant No. 2016M590068.

REFERENCES

[1] CloudLab. https://www.cloudlab.us/.

[2] HTB Linux queuing discipline manual - user guide. http://luxik.cdi.cz/~devik/qos/htb/manual/userg.htm.

[3] Offloading the Segmentation of Large TCP Packets. https://msdn.microsoft.com/en-us/windows/hardware/drivers/network/offloading-the-segmentation-of-large-tcp-packets.

[4] OpenvSwitch QoS. http://docs.openvswitch.org/en/latest/faq/qos/.

[5] Mohammad Al-Fares, Sivasankar Radhakrishnan, Barath Raghavan, Nelson Huang, and Amin Vahdat. 2010. Hedera: Dynamic Flow Scheduling for Data Center Networks.. In *NSDI*, Vol. 10. 19–19.

[6] Mohammad Alizadeh, Albert Greenberg, David A Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. 2010. Data center tcp (dctcp). In *ACM SIGCOMM computer communication review*, Vol. 40. ACM, 63–74.

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

[8] Wei Bai, Li Chen, Kai Chen, Dongsu Han, Chen Tian, and Hao Wang. 2015. Information-agnostic flow scheduling for commodity data centers. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. 455–468.

[9] Hitesh Ballani, Paolo Costa, Thomas Karagiannis, and Ant Rowstron. 2011. Towards predictable datacenter networks. In *ACM SIGCOMM Computer Communication Review*, Vol. 41. ACM, 242–253.

[10] Adrian M. Caulfield, Eric S. Chung, Andrew Putnam, Hari Angepat, Jeremy Fowers, Michael Haselman, Stephen Heil, Matt Humphrey, Puneet Kaur, Joo-Young Kim, Daniel Lo, Todd Massengill, Kalin Ovtcharov, Michael Papamichael, Lisa Woods, Sitaram Lanka, Derek Chiou, and Doug Burger. 2016. A cloud-scale acceleration architecture. In *Microarchitecture (MICRO), 2016 49th Annual IEEE/ACM International Symposium on*. IEEE.

[11] Daniel Crisan, Robert Birke, Gilles Cressier, Cyriel Minkenberg, and Mitch Gusat. 2013. Got loss? get zovn! *ACM SIGCOMM Computer Communication Review* 43, 4 (2013), 423–434.

[12] Bryce Cronkite-Ratcliff, Aran Bergman, Shay Vargaftik, Madhusudhan Ravi, Nick McKeown, Ittai Abraham, and Isaac Keslassy. 2016. Virtualized Congestion Control. In *Proceedings of the 2016 conference on ACM SIGCOMM 2016 Conference*. ACM, 230–243.

[13] Matthew P Grosvenor, Malte Schwarzkopf, Ionel Gog, Robert NM Watson, Andrew W Moore, Steven Hand, and Jon Crowcroft. 2015. Queues don't matter when you can jump them!. In *NSDI*. 1–14.

[14] Chuanxiong Guo, Lihua Yuan, Dong Xiang, Yingnong Dang, Ray Huang, Dave Maltz, Zhaoyi Liu, Vin Wang, Bin Pang, Hua Chen, and others. 2015. Pingmesh: A large-scale system for data center network latency measurement and analysis. In *SIGCOMM*. ACM.

[15] Keqiang He, Eric Rozner, Kanak Agarwal, Yu Jason Gu, Wes Felter, John Carter, and Aditya Akella. 2016. AC/DC TCP: Virtual Congestion Control Enforcement for Datacenter Networks. In *Proceedings of the 2016 conference on ACM SIGCOMM 2016 Conference*. ACM, 244–257.

[16] Chi-Yao Hong, Matthew Caesar, and P Godfrey. 2012. Finishing flows quickly with preemptive scheduling. *ACM SIGCOMM Computer Communication Review* 42, 4 (2012), 127–138.

[17] Keon Jang, Justine Sherry, Hitesh Ballani, and Toby Moncaster. 2015. Silo: Predictable message latency in the cloud. *ACM SIGCOMM Computer Communication Review* 45, 4 (2015), 435–448.

[18] Vimalkumar Jeyakumar, Mohammad Alizadeh, David Mazières, Balaji Prabhakar, Albert Greenberg, and Changhoon Kim. 2013. EyeQ: practical network performance isolation at the edge. In *Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*. 297–311.

[19] Radhika Mittal, Nandita Dukkipati, Emily Blem, Hassan Wassel, Monia Ghobadi, Amin Vahdat, Yaogong Wang, David Wetherall, David Zats, and others. 2015. TIMELY: RTT-based Congestion Control for the Datacenter. In *ACM SIGCOMM Computer Communication Review*, Vol. 45. ACM, 537–550.

[20] Sivasankar Radhakrishnan, Yilong Geng, Vimalkumar Jeyakumar, Abdul Kabbani, George Porter, and Amin Vahdat. 2014. SENIC: Scalable NIC for End-Host Rate Limiting.. In *NSDI*, Vol. 14. 475–488.

[21] Henrique Rodrigues, Jose Renato Santos, Yoshio Turner, Paolo Soares, and Dorgival O Guedes. 2011. Gatekeeper: Supporting Bandwidth Guarantees for Multi-tenant Datacenter Networks.. In *WIOV*.

[22] Alan Shieh, Srikanth Kandula, Albert G Greenberg, Changhoon Kim, and Bikas Saha. 2011. Sharing the Data Center Network.. In *NSDI*, Vol. 11. 23–23.

[23] Arjun Singh, Joon Ong, Amit Agarwal, Glen Anderson, Ashby Armistead, Roy Bannon, Seb Boving, Gaurav Desai, Bob Felderman, Paulie Germano, and others. 2015. Jupiter rising: A decade of clos topologies and centralized control in google's datacenter network. *ACM SIGCOMM Computer Communication Review* 45, 4 (2015), 183–197.

[24] Balajee Vamanan, Jahangir Hasan, and TN Vijaykumar. 2012. Deadline-aware datacenter tcp (d2tcp). *ACM SIGCOMM Computer Communication Review* 42, 4 (2012), 115–126.

[25] Christo Wilson, Hitesh Ballani, Thomas Karagiannis, and Ant Rowtron. 2011. Better never than late: Meeting deadlines in datacenter networks. In *ACM SIGCOMM Computer Communication Review*, Vol. 41. ACM, 50–61.

[26] David Zats, Tathagata Das, Prashanth Mohan, Dhruba Borthakur, and Randy Katz. 2012. DeTail: reducing the flow completion time tail in datacenter networks. *ACM SIGCOMM Computer Communication Review* 42, 4 (2012), 139–150.

[27] Yibo Zhu, Haggai Eran, Daniel Firestone, Chuanxiong Guo, Marina Lipshteyn, Yehonatan Liron, Jitendra Padhye, Shachar Raindel, Mohamad Haj Yahia, and Ming Zhang. 2015. Congestion control for large-scale RDMA deployments. In *ACM SIGCOMM Computer Communication Review*, Vol. 45. ACM, 523–536.