Reduce Completion Time and Guarantee Throughput by Transport with Slight Congestion

Zongyi Zhao†, Qing Li*, Mingwei Xu†, Xingang Shi‡, Han Zhang†
†Department of Computer Science and Technology, Tsinghua University
*Graduate School at Shenzhen, Tsinghua University ‡Institute for Network Science and Cyberspace, Tsinghua University
zhaozy13@mails.tsinghua.edu.cn li.qing@sz.tsinghua.edu.cn xumw@tsinghua.edu.cn
shixg@cernet.edu.cn zhanghan@csnet1.cs.tsinghua.edu.cn

***Abstract*—In typical data center networks, an overwhelming majority of the flows are smaller than 200 KB in size, while most transmitted bytes are from a small fraction of large flows. The small flows are usually from the applications interacting with end users, thus they require small completion times. Meanwhile, the data center owners hope to keep the high throughput of the network to make full use of their investments on the network devices. To reduce the completion times of small flows while maintaining the high throughput of the network, we propose a novel transport algorithm, SCT (Transport with Slight Congestion), in this paper. SCT gives small flows higher priority by increasing their congestion windows at a higher rate. Moreover, SCT keeps the network to be in high utilization, thus the throughput of network is guaranteed. Extensive simulations show that SCT can reduce the average completion time of small flows by up to 48% at the expense of degrading the throughput of network by 5% only, compared with DCTCP.**

I. Introduction

In typical data center networks, almost all the traffic are from TCP flows [5][14], and the flows usually consist of a mixture of small flows and large flows[1] [3]. For many data center networks, particularly these supporting interactive applications, an overwhelming majority of the flows are small flows, while a small fraction of the large flows contribute most of the transmitted bytes and the top 1% of the largest flows can account for more than half of the data volume of the network [8][3]. The small flows are usually from interactive applications like web search, social network content composition and advertisement selection [5], for which latency is highly correlated with the revenue of the enterprises. For example, Amazon found that a latency of 100 ms will cost their sales by 1%, while Google reported that an extra 0.5 s of page generation time will drop their traffic by 20% [1]. Thus the small flows require small completion times. On the other hand, the large flows are usually from background traffic such as web index update traffic [18] or batch processing jobs such as MapReduce [10]. So the large flows usually don't have explicit deadlines but require a high throughput.

Notice that the latency for a packet when it is transmitted from its sender to the receiver is mainly caused by the waiting delay in the buffers of switches. DCTCP [5] proposes to reduce the completion times of flows by keeping the queue length of switches to be around a pre-configured threshold K. It requires that active queue management schemes such as RED [11] are implemented in switches and reuses the ECN machinery [17] that is widely available in modern switches. When the queue length of a switch exceeds the threshold K, the new arriving packets will be marked with the CE (Congestion Experienced) codepoint. The sender of a flow maintains an estimate for the fraction of bytes marked with the CE codepoint in the total bytes sent in last window, and takes this estimate as the extent of network congestion. When congestion is detected, the sender will reduce its congestion window according to the congestion extent, and the congestion window is reduced by half only in the worst case, while it is simply reduced by half once congestion is detected in conventional TCP. Through reacting to congestion timely and in finer granularity, DCTCP avoids the queue buildup due to the large flows injecting bytes into the network over aggressively and makes the queue length linger around the threshold K. HULL [6] is similar to DCTCP but it tries to keep the queue length to be zero by leaving bandwidth headroom, avoiding the waiting delay in switches. DCTCP and HULL perform well in reducing flow completion times, but they fail to distinguish between small flows and large flows. As a result, the small flows will be blocked by large flows and their completion times will be damaged seriously when large flows exist.

L2DCT [15] is based on DCTCP and designed to reduce the completion times of small flows by giving small flows higher priority than large flows. Specifically, when in congestion avoidance phase, the congestion window of a small flow is increased by 1 every RTT, while that of a large flow is increased by 0.05 only. Moreover, when congestion is detected, even if the network is slightly congested, the congestion window of a large flow is decreased aggressively, while the small flows respond much more slowly to the congestion of network. L2DCT works well in reducing the completion times of small flows, however at the expense of degrading the throughput of large flows seriously.

We argue that an ideal transport algorithm should ensure that 1) the small flows are transmitted with higher priority than the large flows thus their completion times are guaranteed even when large flows exist, and 2) the large flows are transmitted efficiently thus the network can achieve a high throughput.

[1]Considering the characteristics of flows in data center networks, we adopt the method introduced in [15] to distinguish between small flows and large flows, i.e., when a flow is started, it is regarded as a small flow; once its transmitted bytes grow beyond 200 KB, it is considered a large flow.

Authorized licensed use limited to: Tsinghua University. Downloaded on October 16,2026 at 06:05:45 UTC from IEEE Xplore. Restrictions apply.

978-1-4799-6664-6/16/$31.00 ©2016 IEEE

Unfortunately, the two requirements appear to be conflicted, since the large flows will contending for bandwidth with the small flows once they are transmitted efficiently, which will damage the completion times of small flows.

In this paper we try to reconcile the contradiction by proposing a novel transport algorithm, SCT (Transport with Slight Congestion). Our basic idea is to increase the congestion windows of small flows at a higher rate than that of large flows when in congestion avoidance phase. Thus the small flows will be allocated with more bandwidth and their completion times will be reduced. Meanwhile, the network is kept to be in slight congestion (as in DCTCP, "slight congestion" here means that a small fraction of the packets transmitted are marked with the CE codepoint, but not necessarily packet drops), which prevents the buffers of switches from being drained and the bandwidth is in high utilization, thus the high throughput of the network is maintained.

Extensive simulations show that SCT can reduce the average completion time of small flows by at least 34% and by up to 50% in the best case, at the expense that the throughput of the network is degraded by 16.5% in the worst case and by as small as 5% in the best case. Moreover, the simulations show that SCT can improve the completion rate of flows and reduce the loss rate of packets significantly.

We make three contributions in this paper. Firstly, we propose the idea of differentiating the priority of small flows from that of large flows by adaptively adjusting the increase rate of congestion window of large flows, as well as the idea of maintaining the high throughput of the network through keeping it slightly congested. Then, based on the above ideas, we design a novel transport algorithm, SCT, to reduce the completion times of small flows while maintaining the high throughput of the network. Finally, we implement SCT in ns-2 simulator [2] and conduct extensive simulations to demonstrate its performance.

The rest of the paper is organized as follows. We discuss the related work in Section II. Then we present the basic ideas and algorithm details of SCT in Section III. In Section IV, we demonstrate the performance of SCT through extensive simulations. We give our conclusion about SCT in Section V at last.

II. Related Work

Besides DCTCP and L2DCT, many previous solutions have been proposed to reduce the completion times of small flows in data center networks.

pFabric [7] requires every packet to take some information (e.g. the deadline or the remaining size of the flow it belongs to) to indicate its priority. A switch will identify the flow to which the packet with the highest priority belongs, then transmit the earliest packet from this flow. pFabric can achieve the near-optimal performance, but it requires to modify both the switches and the end hosts, which makes it difficult to be deployed on a large scale.

Notice that fat-tree [4][12] is a widely adopted topology for the data center networks and it contains rich equal-cost paths between two hosts. RepFlow [21] and SmartRep [19] propose to transmit the replicas of a small flow through multiple equal-cost paths. The receiver will use the result from the replica which is completed first. By transmitting the replicas of a flow through multiple paths, the probability that the flow is blocked by large flows is reduced and thus its completion time is expected to be improved. But these methods are orthogonal to other solutions and they may exacerbate the congestion by injecting more traffic into the network.

CP [9] and E-DCTCP [13] propose to reduce the flow completion times by explicit packet loss notification and reacting to packet losses quickly. In CP, if a packet arrives at a switch whose queue length exceeds a pre-configured threshold, its payload will be cut off and then the payload-cut packet will be transmitted as a normal packet. On receiving the packet, the receiver records the packet loss information in a PACK (Precise ACK) packet, which will be sent back to the sender. In E-DCTCP, if a packet arrives at a switch whose buffer is overflowed, this switch will mark the packet as PL (Packet Loss), swap its source address and destination address, and then send it back to the sender directly. As the two methods work well in reducing the packet loss detection time, they are irrelevant to the priority between small flows and large flows and the completion times of small flows will be impacted greatly by the existence of large flows.

D^2TCP [18] and LPD [22] give priority to flows based on their deadlines. A flow with closer deadline will be given higher priority and the decrease of its congestion window will be smaller when congestion is detected. But they require that the explicit deadlines of flows are given, which are usually unavailable in real data center networks.

III. Algorithm Design of SCT

A. Basic Ideas

As stated before, when a flow is started, it is classified as a small flow, and it is re-classified as a large flow once its transmitted bytes exceed 200 KB. Since the number of large flows is far smaller than that of small flows in typical data center networks, only a small fraction of flows are mistakenly classified as small flows when they are started, which is acceptable. Note that the target of SCT is to reduce the completion times of small flows while maintaining the high throughput of the network. The rationale behind SCT to achieve this is to keep the network "slightly congested" through adjusting the transmitting rate of large flows. On the one hand, since the network is congested, which means that a fraction of the transmitted packets encounter a queue length greater than the marking threshold and are marked with the CE (Congestion Experienced) codepoint for DCTCP-like transport algorithms, the buffer of the bottleneck link is not drained and the bandwidth is in relatively high utilization. Thus the throughput of the network is guaranteed. On the other hand, since the network is only slightly congested, the small flows will encounter a "not so congested" network when they are started even if a large number of large flows exist, which helps to reduce the completion times of small flows.

Authorized licensed use limited to: Tsinghua University. Downloaded on October 16,2026 at 06:05:45 UTC from IEEE Xplore. Restrictions apply.

As in DCTCP, the sender of a flow in SCT maintains an estimate, α, for the congestion extent of the network. α is updated every RTT as follows:

$$\alpha \leftarrow (1-g) \times \alpha + g \times F$$

where F is the fraction of bytes marked with the CE codepoint in the total bytes sent in last congestion window, and $0 < g < 1$ is the weight given to this new sample.

To reduce the completion times of small flows, we highlight the priority of small flows as follows.

From the viewpoint of the sender of a large flow, if the congestion extent it maintains is greater than expected, it will try to release the congestion by slowing down its sending rate. If the network is constantly more congested than expected, it will decrease its sending rate to the lowest possible value. Otherwise, if the congestion extent is smaller than expected, it will attempt to increase its sending rate to fully utilize the bandwidth. Since the additive-increase-multiplicative-decrease pattern of congestion control for TCP is a relatively mature solution, we will adhere to this pattern when adjusting the transmitting rate of a large flow. So instead of decreasing or increasing the congestion window directly, the sender will adjust the increase rate of the congestion window (when in congestion avoidance phase) according to the maintained congestion extent α.

Unlike the large flows, for a small flow in congestion avoidance phase, if congestion is not detected, the sender will increase its congestion window at a constant rate that is greater than that of large flows, ignoring the congestion status of the network. Thus small flows will be given higher priority.

B. Algorithm Details

Algorithm 1 Update the increment of congestion window.

1: **when** the update timer expires **do**
2: **if** $\alpha > t$ **then**
3: $c \leftarrow c \times (1 - \frac{\alpha - t}{1-t}/2)$
4: **else if** $\alpha < t$ **then**
5: $c \leftarrow c + c_{incr} \times \frac{t-\alpha}{t}$
6: **end if**
7: $c \leftarrow \max(c, c_{min})$
8: $c \leftarrow \min(c, c_{max})$
9: Reschedule the update timer in RTT sec.
10: **end when**

As stated above, for a large flow in congestion avoidance phase, if congestion is not detected, its congestion window will be increased by

$$cwnd \leftarrow cwnd + c$$

where c is updated every RTT sec to respond to the congestion status of the network. The method to update c is presented in Algorithm 1.

At first, we configure a threshold t as the target congestion extent of the network. The sender of a large flow aims to keep the congestion extent α it maintains to be around t through adjusting the value of c periodically. If the congestion extent α is greater than t, to release the congestion, the sender will decrease c by

$$c \leftarrow c \times (1 - \frac{\alpha - t}{1-t}/2)$$

Thus the larger α is, the more sharply c will be decreased. When the network is extremely congested and $\alpha = 1$, c will be decreased by half. On the contrary, if the congestion extent is close to the expected value, i.e., α approaches t, c will be almost unchanged.

Otherwise, if the congestion extent α is smaller than t, to fully utilize the bandwidth of the network, the sender will increase the value of c as follows:

$$c \leftarrow c + c_{incr} \times \frac{t - \alpha}{t}$$

where c_{incr} is a pre-configured parameter and it is the largest possible increase of c. Thus the smaller α is, the larger the increase of c will be. Especially, if $\alpha = 0$, the increase of c will be c_{incr}. If the congestion extent is close to the expected value, i.e., α is close to t, the increase of c will be close to 0.

To prevent c from being too large or too small, we limit the value of c into the range of $[c_{min}, c_{max}]$.

When congestion occurs, the congestion window of a large flow will be decreased by

$$cwnd \leftarrow cwnd \times (1 - \alpha/2)$$

which is the same as that in DCTCP.

For a small flow, if it is in congestion avoidance phase and congestion is not detected, its congestion window will be increased by

$$cwnd \leftarrow cwnd + c_{max}$$

While the increment of congestion window is upper bounded by c_{max} and can be as small as c_{min} for a large flow, it is set to c_{max} constantly for a small flow. Thus the transmitting rate of a small flow will be increased more quickly than that of a large flow if they coexist in the network, giving the small flow higher priority.

When congestion occurs, the congestion window of a small flow will be decreased using gamma function [16] as follows:

$$cwnd \leftarrow cwnd \times (1 - \alpha^{2.5}/2)$$

This gamma function allows that the decrease of congestion window of a small flow is always smaller than that of a large flow, and when the network is extremely congested and α approaches 1, the congestion window of a small flow will be decreased by half, relieving the congestion effectively.

C. Guidelines for Parameter Selection

As the increment of congestion window for a large flow could be as large as c_{max} and that for a small flow is constantly c_{max}, to keep SCT stable, it is not suggested to set c_{max} to be greater than 1. While a large c_{min} can guarantee that the congestion windows of large flows increase quickly in the worst case, a smaller c_{min} leaves more room for the

Authorized licensed use limited to: Tsinghua University. Downloaded on October 16,2026 at 06:05:45 UTC from IEEE Xplore. Restrictions apply.

small flows to increase their share of bandwidth. Since t is the target congestion extent of the network, as a smaller t helps the small flows to obtain more bandwidth, it will limit the transmitting rate of large flows more strictly and hence negatively impact the throughput of large flows. A smaller c_{incr} makes the large flows to recover from congestion window reduction more slowly, which allows small flows to obtain more bandwidth, however at the cost of possibly leaving the network idle when there are not enough small flows to take up the available bandwidth immediately. When choosing values for the parameters, careful consideration should be paid to make a reasonable trade-off.

D. More about SCT

In DCTCP, the queue length of a link will oscillate more aggressively when more flows pass through it. This may result in packet losses as well as leave the link idle, degrading the throughput of the network. In SCT, suppose the flows are large flows and $c_{max} = 1$ (this is the default value of c_{max} in our simulations), the increment c will be often smaller than 1, thus the aggregate increment of congestion window of the flows will be smaller than that in DCTCP, allowing the queue length of the link to be increased more slowly. Moreover, since SCT aims to keep the congestion extent α to be around t, if t is set to a reasonably small value, the decrease of congestion window of the flows will be smaller than that in DCTCP when congestion is detected, resulting in the queue length of the link to be reduced more mildly. Thus the queue length will oscillate less aggressively in SCT, which helps to reduce the packet losses and keep the high throughput of the network.

Notice that the main feature of SCT is to adjust the transmitting rate of large flows according to the congestion extent of the network and both DCTCP and HULL [6] support to extract the congestion extent from the single bit stream of ECN marks. As SCT described above is based on DCTCP, it can be implemented as being based on HULL as well.

IV. Evaluations

In this section, we will demonstrate the advantages of SCT over DCTCP through comprehensive simulations in the ns-2 simulator [2]. As SCT helps to keep the queue length steady and a short queue will reduce the waiting delay experienced by packets, we set the marking threshold K to 15. Moreover, we set the sample weight g to $1/16$ when updating the congestion extent α, which is the suggested value of DCTCP. Following the guidelines stated in Subsection III-C, we set the parameters of SCT as in Table I. The buffer size of a link is 100 KB, which is able to accommodate about 70 packets given that the mean packet size is 1500 B. To be more sensitive to packet losses, we set RTO_{min} to 10 ms.

Notice that DCTCP increases the congestion windows by 1 constantly, while SCT increases the congestion windows of large flows by c, which is upper bounded by $c_{max} = 1$. When some flow is completed and leaves its share of bandwidth available to other flows, other flows will take up this share more quickly in DCTCP, reducing the waste of bandwidth.

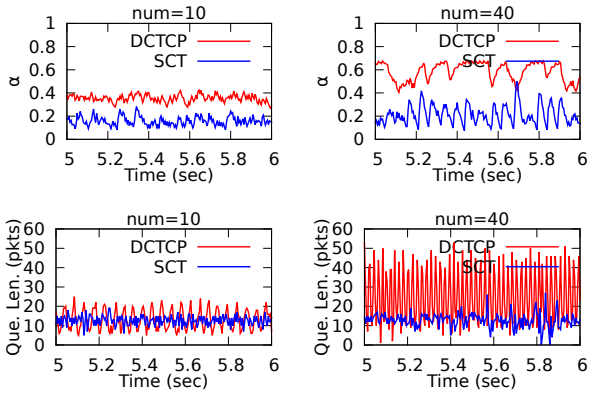


Fig. 1. Congestion extent α and the queue length of the bottleneck link when the number of long-lived flows increases from 10 to 40.

Thus the throughput of the network will be higher when DCTCP is adopted as the transport algorithm, especially when flows enter and leave the network frequently. Our target in SCT is to make the throughput of the network be close to that of DCTCP while reducing the average completion time of small flows. Moreover, as stated in Subsection III-D, in SCT the queue length of a link is expected to oscillate less aggressively, and the packet loss rate is expected to be low.

A. Release Congestion & Reduce Latency

In this simulation, the topology of the network consists of a single bottleneck link and a number of (source node, destination node) pairs connected to this link. To support a large number of concurrent flows, we set the RTT in absence of queuing to 1.0 ms, which is slightly greater than that in real data center networks. There are a number of long-lived flows transmitting from the source nodes to the destination nodes simultaneously. We increase the number of flows from 10 to 40 and measure the congestion extent α maintained by one of the flows as well as the queue length of the bottleneck link.

While the congestion extent α grows dramatically as the number of flows increases in DCTCP, as shown in Figure 1, the congestion extent α grows much more gently in SCT. Hence SCT works well in limiting the congestion extent of the network through adjusting the transmitting rate of long flows, which is important for reducing the completion times of small flows. Figure 1 further shows that the queue length in SCT is much more steady than that in DCTCP. As proved by later simulations, a steady queue length will help to reduce the packet loss rate, especially when the bursty traffic is present.

Since the latency experienced by packets is mainly from the waiting delay in the buffers of switches, SCT helps to reduce the latency experienced by packets as well through making the queue length closely linger around the marking threshold. Consider a scenario of incast traffic which is widespread in data center networks. The degree of the incast traffic is 20, i.e., there are 20 senders sending packets to a single receiver simultaneously. Each sender will send 20 MB of data. We sample the queue length of the bottleneck link every 100

Authorized licensed use limited to: Tsinghua University. Downloaded on October 16,2026 at 06:05:45 UTC from IEEE Xplore. Restrictions apply.

TABLE I
THE PARAMETER SETTINGS OF SCT

Param	Value
t	0.1
c_{max}	1.0
c_{min}	0.0
c_{incr}	1.0
c	1.0 (initially)

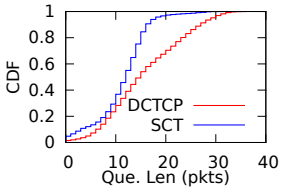


Fig. 2. The CDF (Cumulative Distribution Function) of the queue length at the bottleneck link.

μs and take it as an indicator of the latency experienced by packets. Figure 2 shows that the packets in SCT will experience shorter latency as the 95-percentile queue length of the bottleneck link is 29 packets with DCTCP and 18 packets only with SCT.

B. Reduce AFCT & Maintain Throughput

The network topology in this simulation is the same as that in the previous simulation. There are 10 long-lived flows in the network as the background traffic. We start a number of small flows that have sizes distributed in the range of $[2\text{ KB}, 200\text{ KB}]$ uniformly and arrive according to a Poisson process. The simulation lasts for 10 s. We repeat the simulation 8 times and increase the network load caused by small flows from 0.1 to 0.8. We will measure the average completion time of small flows and the throughput of network. Besides, to illustrate the benefits of SCT, we will calculate the loss rate of packets as well as the completion rate of flows.

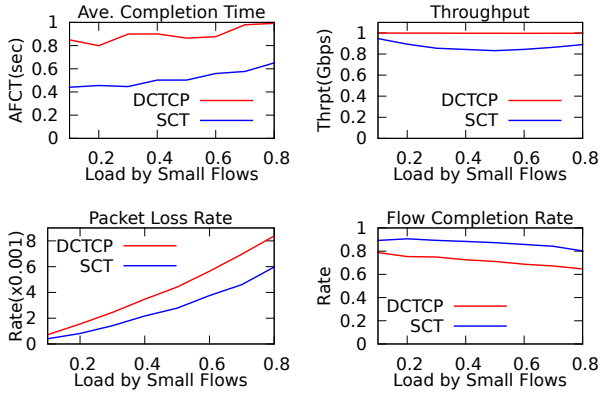


Fig. 3. The AFCT (Average Flow Completion Time) of small flows, the throughput of network, the loss rate of packets and the completion rate of flows when the load caused by small flows increases from 0.1 to 0.8.

Figure 3 shows that as SCT degrades the throughput of the network slightly, it reduces the AFCT of small flows significantly. Especially, when the network load caused by the small flows is 0.1, SCT reduces the AFCT of small flows by 48% while sacrificing the throughput of network by 5% only. In fact, in this simulation, SCT degrades the network throughput by 16.5% in the worst case, while it improves the AFCT of small flows by at least 34% and by up to 50% in the best case. Moreover, the completion rate of flows and loss rate of packets in SCT are always better than that in DCTCP. For example, when the network load caused by small flows is 0.2 the packet loss rate in SCT is 47.4% smaller than that in DCTCP, and when the network load caused by small flows is 0.8 the flow completion rate in SCT is 25.3% greater than that in DCTCP. We argue that the sacrifice in throughput is acceptable considering the gains in the AFCT of small flows as well as the packet loss rate and the flow completion rate.

C. Simulation with Real Data Center Traffic Pattern

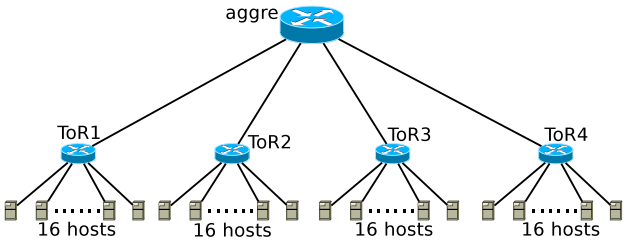


Fig. 4. A multi-layer topology of the data center network.

We will demonstrate the performance of SCT in a more realistic scenario in this simulation. As in [20], we adopt a single-root tree as the topology of the network (shown in Figure 4). The network consists of a single aggregate switch, to which 4 ToR switches are connected, and then each ToR switch is connected to 16 hosts. The bandwidth of a host-ToR link is 1 Gbps and that of a ToR-aggre link is 10 Gbps. The transmission delay of each link is 25 μs. We adopt the traffic pattern used in [7], which is from a data center supporting web search applications. As in [7], the flows arrive according to a Poisson process and its source node and destination node are selected randomly among the hosts. To mimic the background traffic, we run a long-lived flow on every host-ToR link. The simulation lasts for 10 s.

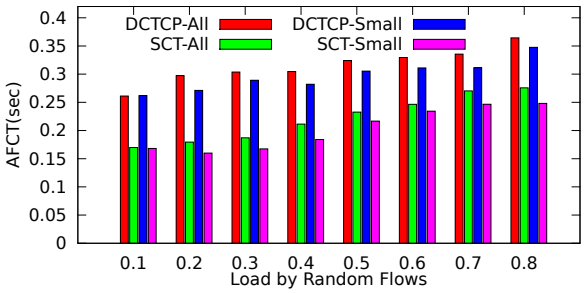


Fig. 5. The average completion time of small flows as well as that of all flows in DCTCP and SCT.

Figure 5 shows the AFCT of all the flows in DCTCP and SCT (denoted by DCTCP-All and SCT-All respectively), that of small flows in DCTCP and SCT (denoted by DCTCP-Small and SCT-Small respectively). Obviously the performance of SCT is always better than that of DCTCP. The best improvement is achieved in the scenario where the network load caused by random traffic is 0.3. In this scenario, SCT reduces the

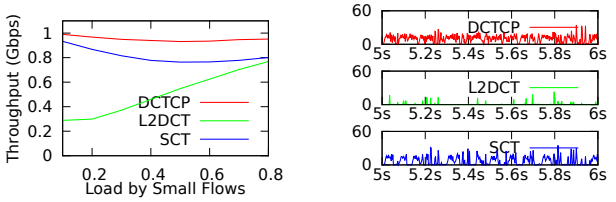
Authorized licensed use limited to: Tsinghua University. Downloaded on October 16,2026 at 06:05:45 UTC from IEEE Xplore. Restrictions apply.

AFCT of all the flows by 38.39% and that of small flows by 42.06%. We further find that the AFCT of large flows are also reduced, and flow completion rate in SCT is slightly greater than that in DCTCP. This is caused by the fact that through adjusting the transmitting rate of large flows, packet losses are reduced and the bandwidth is utilized more efficiently.

D. Comparison with L2DCT

As SCT follows the line of DCTCP [5] and L2DCT [15], in this simulation we will demonstrate the advantage of SCT over L2DCT in maintaining the high throughput of the network. As in Subsection IV-A, the topology of the network contains a single bottleneck link which has the bandwidth of 1 Gbps. We run 2 long-lived flows and a number of small flows whose sizes are distributed uniformly in the range of $[2\ \text{KB}, 200\ \text{KB}]$. The small flows are started according to a Poisson process. We increase the network load caused by small flows from 0.1 to 0.8 and calculate the throughput of the network as well as the queue length at the bottleneck link.

Figure 6(a) shows that the throughput of the bottleneck link in L2DCT is constantly smaller than that in SCT. Especially, when the load caused by small flows is 0.1, L2DCT degrades the throughput of the network by 71%, while SCT degrades the throughput by 6% only, compared with DCTCP. This is because in L2DCT, large flows reduce their congestion windows over aggressively when congestion is detected, and recover from this reduction very slowly when congestion is released. This will leave the link idle while the large flows have more data to transmit, degrading the throughput of the network. Figure 6(b) shows that when the load caused by small flows is 0.1, the buffer of the bottleneck link is frequently drained if L2DCT is adopted. In fact, the samples where the queue length is zero account for 92% of the 4000 samples we collect in L2DCT, while that account for 28% of the total samples only in DCTCP.



(a) (b)

Fig. 6. (a) the throughput of network when the load caused by the small flows increases from 0.1 to 0.8; (b) the queue length (in packets) of bottleneck link when the load caused by small flows is 0.1.

V. Conclusion

Since the latency of interactive applications is highly correlated with the revenue of enterprises, and the data center owners hope to improve the utilization of data center networks to make full use of their investments, reducing the completion times of small flows while maintaining the high throughput of the network is always an important target for data center networks. In this paper, we propose SCT to try to achieve this target. The rationale behind SCT follows two principles: 1) keep the network slightly congested to keep the high utilization of the network; 2) give higher priority to small flows over large flows when they contend for bandwidth. The performance of SCT in reducing the completion times of small flows is particularly outstanding when more large flows exist, while it is lower bounded by that of DCTCP if fewer large flows exist.

Acknowledgments

This work is supported by the National High-Tech Research and Development Program of China (863 Program) (Grant No. 2015AA015701, 2015AA015603), the National Natural Science Foundation of China (Grant No. 61402255, 61402253), the R&D Program of Shenzhen under Grant No. JCYJ20150630170146830.

References

[1] Latency is everywhere and it costs you sales - how to crush it. http://highscalability.com/latency-everywhere-and-it-costs-you-sales-how-crush-it.
[2] The Network Simulator - ns-2. http://www.isi.edu/nsnam/ns/.
[3] D. Abts and B. Felderman. A guided tour of data-center networking. *Commun. ACM*, 2012.
[4] M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. In *SIGCOMM*, 2008.
[5] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan. Data center tcp (dctcp). In *SIGCOMM*, 2010.
[6] M. Alizadeh, A. Kabbani, T. Edsall, B. Prabhakar, A. Vahdat, and M. Yasuda. Less is more: Trading a little bandwidth for ultra-low latency in the data center. In *NSDI*, 2012.
[7] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker. pfabric: Minimal near-optimal datacenter transport. In *SIGCOMM*, 2013.
[8] T. Benson, A. Akella, and D. A. Maltz. Network traffic characteristics of data centers in the wild. In *IMC*, 2010.
[9] P. Cheng, F. Ren, R. Shu, and C. Lin. Catch the whole lot in an action: Rapid precise packet loss notification in data centers. In *NSDI*, 2014.
[10] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. In *OSDI*, 2004.
[11] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Trans. Netw.*, 1993.
[12] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta. Vl2: A scalable and flexible data center network. In *SIGCOMM*, 2009.
[13] Y. Huang and B. Hu. Enhanced dctcp to explicitly inform of packet loss. In *ICC*, 2015.
[14] V. Jacobson. Congestion avoidance and control. *SIGCOMM Comput. Commun. Rev.*, 1988.
[15] A. Munir, I. A. Qazi, Z. A. Uzmi, A. Mushtaq, S. N. Ismail, M. S. Iqbal, and B. Khan. Minimizing flow completion times in data centers. In *INFOCOM*, 2013.
[16] C. A. Poynton. *Digital video and HDTV : algorithms and interfaces*. Morgan Kaufmann, 2003.
[17] K. Ramakrishnan, S. Floyd, and D. L. Black. The addition of explicit congestion notification (ecn) to ip, 2001. RFC 3168.
[18] B. Vamanan, J. Hasan, and T. Vijaykumar. Deadline-aware datacenter tcp (d2tcp). In *SIGCOMM*, 2012.
[19] F. Wang, Z. Qian, S. Zhang, M. Dong, and S. Lu. Smartrep: Reducing flow completion times with minimal replication in data centers. In *ICC*, 2015.
[20] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowtron. Better never than late: Meeting deadlines in datacenter networks. In *SIGCOMM*, 2011.
[21] H. Xu and B. Li. Repflow: Minimizing flow completion times with replicated flows in data centers. In *INFOCOM*, 2014.
[22] H. Zhang, X. Shi, X. Yin, F. Ren, and Z. Wang. More load, more differentiation - a design principle for deadline-aware congestion control. In *INFOCOM*, 2015.

Authorized licensed use limited to: Tsinghua University. Downloaded on October 16,2026 at 06:05:45 UTC from IEEE Xplore. Restrictions apply.