Scalable Congestion Control Protocol based on SDN in Data Center Networks

Jaehyun Hwang*, Joon Yoo†, Sang-Hun Lee‡, and Hyun-Wook Jin‡
*Bell Labs, Alcatel-Lucent, Seoul, Korea
†Gachon University, Seongnam, Korea
‡Konkuk University, Seoul, Korea
Email: *jh.hwang@alcatel-lucent.com, †joon.yoo@gachon.ac.kr, ‡{leeh,jinh}@konkuk.ac.kr

Abstract—**On-line data center applications render challenging network latency demands to meet their service level requirements. These applications, however, frequently suffer from increased latency due to the packet loss and queueing delay at the network switches. These are mainly as a result of the momentary massive bursts by the Partition/Aggregation application traffic patterns, which causes incast network congestion at the network switches. In this paper, we propose a scalable congestion control protocol, called SCCP. Our scheme effectively limits the data rate of the TCP senders by leveraging the Software Defined Networking (SDN)switches, so that the total utilization does not exceed the bottleneck link capacity. Furthermore, SCCP can be easily deployed to the existing SDN data center switches by extending the OpenFlow specifications. Our Open vSwitch-based prototype experiments and ns-3 simulations show that SCCP is scalable for up to hundreds of concurrent flows traversing through the data center network switch port.**

I. Introduction

Data centers have emerged as a key resource in the past decade for providing a plethora of online services such as web search and cloud computing. The network latency demands, which originate from the service level agreements (SLAs), are critical for the service providers since they directly affect the operator revenue [1]–[3]. For example, according to Amazon's report, the effect of 100ms latency approximately results in 1% drop in sales [1].

Generally, the round-trip time (RTT) is normally less than $250\mu s$ for non-congested flows in data center networks [2]. The network latency is critical for many soft real-time data center applications because the worker nodes generally have response deadlines between 20ms and 50ms based on the SLA [2], [3]. Due to several reasons, however, the network latencies can be significantly increased. First, the data center applications typically employ the Partition/Aggregate traffic pattern [2]; an application request is broken down into smaller jobs and allocated to multiple workers, and the responses of the workers are gathered by the root server. This may cause massive bursts at the switches, especially at the shallow-buffered Top-of-Rack (ToR) switches, thus result in *incast congestion*. Second, non-realtime background flows may take a large portion of the link bandwidth, thus consistently occupying the switches' buffer. It has been reported that under heavily congested situations, the queuing delay may increase up to 14ms even with 4MB of shallow-buffered data center switches [2]. Furthermore, a large switch buffer occupancy renders very small buffering opportunities for the massive bursts mentioned above, causing more packets losses and TCP retransmission timeouts (RTOs). In result, the application deadlines would be easily missed if the flow completion times of the worker nodes are delayed by the network congestion.

In this paper, we propose a new scalable congestion control protocol, called SCCP, that is implemented to the Software Defined Networking (SDN) data center switches. Our main objective is to avoid the switch buffer overflow and reduce the queuing delay even under the circumstances where a massive number of bursty flows, e.g., up to hundreds, exist. To achieve this goal, we first extend the OpenFlow specifications [4] at the SDN switch, to accurately assess the number of TCP flows that traverse each switch port. This information is used to compute the *fair-share* of each flow, so that the total link utilization does not exceed the bandwidth delay product (BDP). This information is eventually delivered to each TCP sender by updating the advertisement window field in the TCP header.

SCCP is easily employable to the legacy SDN switch based data center networks due to the following reasons. First, SCCP does not require any hardware modifications to the conventional SDN data center network switches, since SCCP abides by the principles of the open flow specifications. Second, SCCP does not require any change to the end-hosts and their applications. To validate the deployability, we have built the prototype of the SCCP algorithm on the Open vSwitch (OVS) [5] and evaluated it on our real data center testbed. The testbed experiment and ns-3 simulation results show that our scheme copes with a large number of concurrent flows without suffering any packet loss and large queuing delay.

The rest of the paper is organized as follows. Section II briefly reviews the related work and describes motivation. And then, our SCCP algorithm is explained in detail in Section III. Section IV evaluates the proposed scheme through extensive ns-3 simulations, and Section V presents our prototype experiments. Finally, Section VI concludes the paper.

II. Background and Motivation

A. Data center protocols and limitations

Traditionally, one way for avoiding the network congestion is to utilize the delay information like TCP Vegas [6] as queuing delay is a good indicator for the congestion. However, RTT measurements are not reliable in low latency data center networks because even small noisy fluctuations can affect the result of the measurements. To be *explicitly* aware of the congestion level, DCTCP [2] utilizes the Explicit Congestion Notification (ECN) [7] functionality, already deployed in many data center switches, and controls server's sending rate by the help of the ECN. The problem is that DCTCP shows poor scalability in terms of the number of bursty flows; it copes with up to 35 servers without suffering any packet losses in their experimental environments. IA-TCP [8] and DIATCP [9] show better scalability by controlling the workers' sending rate in order not to exceed the link capacity at the receiver side. However, they do not solve the network congestion that occurs at the aggregation/core switches since they assume that the bottleneck point is usually at the edge (i.e., Top-of-Rack) switches. The aforementioned schemes are called *host-based approaches*, since they are implemented only at the host-side.

The scalability is an important issue since the number of concurrent flows can generally be larger than 35 [2]. Moreover, it is reported that the number of active flows at any given interval is less than 10,000 per rack in cloud data centers [10]. Our insight is that, however, the existing host-based approaches are limited in terms of the scalability; the servers often react to the congestion *slowly* so that they inevitably suffer from the momentary burst of traffic. The main reason of such slow reaction is that the network estimation that they perform fundamentally takes at least more than one RTT. In addition, they usually exploit smooth estimation schemes (e.g., exponentially weighted moving average) to tolerate sudden spikes in measurements, so a few more RTTs may be required for the accurate estimation.

On the other hand, *switch-based approaches* such as D^3 [3] and DeTail [11] have a key advantage over the host-based approach; they can monitor the current network status and provide accurate feedback to the TCP sender immediately. Therefore, they can quickly react to the large number of bursty flows. However, the main drawback is that they generally require high-cost and/or customized hardware chips as well as modifications at the end-hosts and applications, which could be a big hurdle for deployment.

B. SDN-based approach

The SDN is emerging as a promising technology for the data center management; recently many switch vendors have released OpenFlow-based SDN switches. Fig. 1 shows an example of the OpenFlow switch operations [4] on SDN switches. Whenever a packet arrives at the SDN switch, it tries to find an entry from the forwarding table that matches with the packet-header information. If there are no matching entries, it asks the SDN controller to create a new rule and insert it into the forwarding table so as to handle the new flow. Then, the packet, and the subsequent packets, will be forwarded according to the new rule.

This programmability on the data plane can be advantageous, since it does not require any hardware modification to the data center switches. Therefore, a new function can be easily adopted via SDN [12], significantly reducing the *hurdle* of deployment. In the following section, we will show how our scheme is embedded into the current SDN framework, without any modifications to the end-hosts and applications.

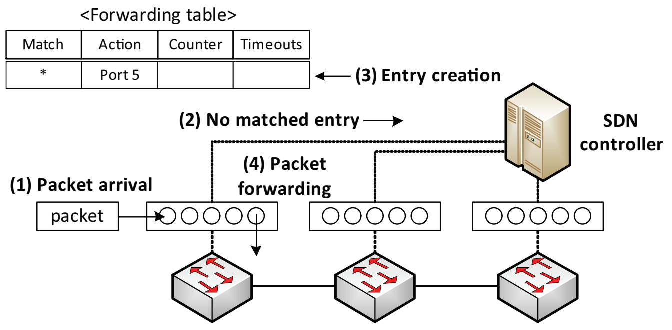


Fig. 1. OpenFlow operations on SDN switches.

III. SCCP: Scalable Congestion Control Protocol

A. Basic operations

The data center switch generally consists of multiple input and output ports, where each port is shared by multiple flows. The output-port is where the network congestion in data centers mainly occurs. So our congestion control scheme is conducted at each switch output-port, which we simply term *port* hereafter. More specifically, at the ith port ($1 \leq i \leq n$) of the switch,

1) For an outgoing packet:
   (a) Keep track of the number of TCP flows (N_i).
2) For an incoming packet:
   (a) Compute the fair-share ($fair\_share_i$).
   (b) Change the value of the advertisement window field in the TCP header ($awnd$), to the value of $fair\_share_i$, if $awnd > fair\_share_i$.

In (1)-(a), N_i is the number of flows traversing the ith switch port. To keep track of this number, the switch uses flow initiation and termination packets such as TCP SYN/FIN. In (2)-(a), we compute the fair-share by simply dividing the BDP of the ith port by N_i. In the BDP, the bandwidth is the link capacity of the port, and the delay is the common RTT of all flows. We discuss how to define this common RTT in the following subsection. So the fair-share is computed as follows:

$$fair\_share_i = \frac{link\_capacity_i \times common\_RTT}{N_i} \quad (1)$$

To deliver the fair-share to the end-host server, the advertisement window field in the TCP header is replaced by the fair-share only when the existing value is larger than the new one. By doing so, each server knows the fair-share of the bottleneck port (in the bottleneck switch) on its end-to-end path.

The major advantage of SCCP are not only its effectiveness to control congestion, but also its low overhead and ease of deployment. The computation overhead is very low due to its simplicity, and more importantly, SCCP does not need per-flow information, which could incur significant overhead, but instead only an integer variable per port to keep track of N_i. In addition, by utilizing the already existing TCP advertisement window field, it feedbacks to the TCP senders without any changes in the current TCP packet header.

B. Common RTT

To measure the fair-share, a naive approach is to measure each flow's RTT and keep the per-flow information in the switch, but this incurs significant overhead. To avoid this, we define a *common RTT*, the common representative RTT for every flow. Assuming that the RTT is generally less than 250μs in data center networks [2], we take a sufficiently larger value for the common RTT, such as 300μs or 400μs, so that the RTT is always lower than the common RTT. We note that, therefore, the computed BDP becomes slightly larger than the actual one, causing link overflow. To absorb the overshot data packets, we exploit a small portion of the packet buffer at each switch port. For example, when the RTT of the flows is 200μs in average and the link capacity is 1Gbps, the actual BDP would be 25KB. If we set the common RTT to 300μs, the calculated BDP is 37.5KB, thus about 12.5KB of data packets will be buffered at the switches in each RTT. This is sustainable even in shallow-buffered switches and even helpful to fully utilize the link capacity.

C. OpenFlow extensions

To support SCCP operations in the SDN framework, we need to extend the *flow match fields* defined in the OpenFlow specification first, in order to access (i) SYN/FIN flags and (ii) advertisement window fields in the TCP header at the SDN switch. The latest version of OpenFlow specification (version 1.5.0) [4] defines three flow match fields for TCP as follows:

```
/* OXM Flow match field types for OpenFlow basic class. */
enum oxm_ofb_match_fields \{
    ...
    OFPXMT_OFB_TCP_SRC = 13, /* TCP source port. */
    OFPXMT_OFB_TCP_DST = 14, /* TCP destination port. */
    ...
    OFPXMT_OFB_TCP_FLAGS = 42, /* TCP flags. */
    ...
\};
```

As shown above, version 1.5.0 supports a flow match field for TCP flags, so we can detect the packets that include TCP SYN/FIN flags. For the advertisement window field, we define a new field-type, `OFPXMT_OFB_TCP_WINDOW`, so that the fair-share information is set via the *set-window* action that is defined as follows.

Currently OpenFlow defines 19 action-types including `OFPAT_SET_FIELD`, which is used for setting a header field. In this work, we define three new action-types:

- OFPAT_INCREMENT_NI: increase N_i by one when the port number i is given.
- OFPAT_DECREMENT_NI: decrease N_i by one when the port number i is given.
- OFPAT_SET_WINDOW: set a header field if the new value is smaller than the existing value.

The first two actions are used to count the number of TCP flows for the ith switch port when SYN/FIN flags are matched. The purpose of the action-type `OFPAT_SET_WINDOW` is similar with that of `OFPAT_SET_FIELD`, but used only for the SCCP-specific operation; the advertisement window field should be updated only when the new fair-share value is smaller than the existing value to deliver the bottleneck fair-share to the server. Table I shows an example of the OpenFlow table that implements SCCP operations, described in Section III-A. We note that our scheme mainly focuses on TCP in the protocol match (i.e., `OFPXMT_OFB_IP_PROTO`) as it shows that 99.91% of traffic in data center networks is TCP [2].

TABLE I
EXAMPLE OF THE OPENFLOW TABLE FOR SCCP OPERATIONS.

No.	Flow Entry
1	**priority**=100 **packets**=0 **bytes**=0 **hard_timeout**=0 ... **match**: OFPXMT_OFB_IP_PROTO **value**=*TCP* **match**: OFPXMT_OFB_TCP_FLAGS **value**=*SYN* **instruction**: OFPIT_WRITE_ACTIONS **action**: OFPAT_INCREMENT_NI **value**=*output_port*
2	**priority**=100 **packets**=0 **bytes**=0 **hard_timeout**=0 ... **match**: OFPXMT_OFB_IP_PROTO **value**=*TCP* **match**: OFPXMT_OFB_TCP_FLAGS **value**=*FIN* **instruction**: OFPIT_WRITE_ACTIONS **action**: OFPAT_DECREMENT_NI **value**=*output_port*
3	**priority**=100 **packets**=0 **bytes**=0 **hard_timeout**=0 ... **match**: OFPXMT_OFB_IP_PROTO **value**=*TCP* **instruction**: OFPIT_WRITE_ACTIONS **action**: OFPAT_SET_WINDOW **match**: OFPXMT_OFB_TCP_WINDOW **value**=*fair_share_of_output_port*

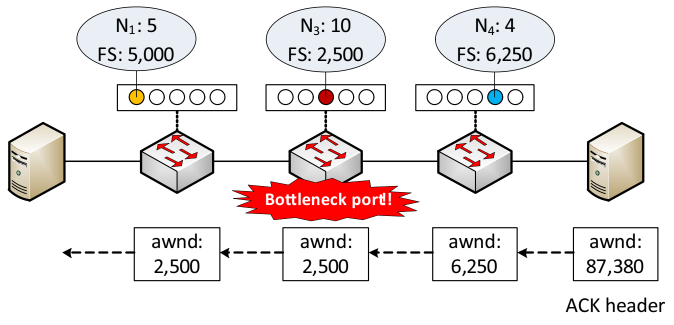


Fig. 2. SCCP operation example: BDP at every port is 25,000 bytes assuming that the link capacity is 1Gbps and the common RTT of the flows is 200μs.

D. SCCP operation example

We give an example of the SCCP operation in this subsection. Let us assume that there are three switches between the TCP sender and the receiver as shown in Fig. 2. When the sender sends a SYN packet to the receiver, this increases N_i at each switch. In this example, the packet traverses port 1, port 3, and port 4 in each switch, increasing their N_i to 5, 10, and 4, respectively. When the receiver sends an acknowledgement (ACK) packet to the sender along the reverse path, the switches compute the fair-share of their port and update the advertisement window field (*awnd*) accordingly. We assume that the BDP is same (25,000 bytes) at each port in this example, so the fair-share of the bottleneck port is 2,500 bytes at the middle switch. Finally, this number is delivered to the sender via the ACK. Note that the bottleneck point may frequently change due to the dynamic TCP traffic. Nonetheless, the bottleneck information is continuously updated to the senders via the

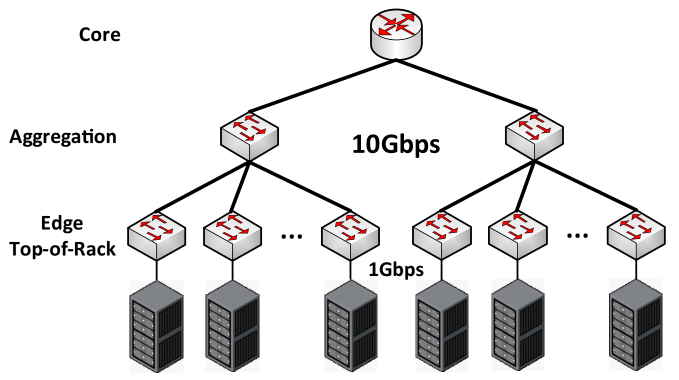


Fig. 3. 3-tier data center network topology for our simulations.

ACK packets, so each sender can quickly adjust its data size according to the fair-share.

E. Idle and ghost flow detection

In data center networks, *idle* flows, such as *heartbeat* messages, may exist. These idle flows generally do not tear down their connections, even after all the data is transmitted, to wait for future transmission in the *idle* state to avoid the cost of establishing a new connection via three-way handshake. Moreover, there is a possibility that a connection is closed without any explicit FIN packet, which is called a *ghost* flow. In these scenarios, SCCP would waste the bandwidth allocated to those flows. Our solution is to utilize the timeout field of the OpenFlow table at the end-servers (e.g., OVS) so that the idle/ghost flows are detected when the timer expires. Then the end-servers can send a control packet along the end-to-end path the flows traverse to reduce N_i explicitly at each switch port. We plan to devise an optimal solution to employ this function as future work.

IV. Simulations

We implement the SCCP algorithm in the ns-3 simulator [13]. Fig. 3 depicts our simulation topology, a form of classical 3-tier data center networks, which is widely used in commercial cloud data centers [10]. It consists of 1 core, 2 aggregation, 10 Top-of-Rack (ToR) switches, and 48 servers per rack (i.e., total 48 $\times$ 10 = 480 servers assuming general 48-port ToR switches). The link rate is 1Gbps for ToR switches and 10Gbps for core/aggregation switches, and the packet buffer size per port is set to 128, 300, and 400 Kbytes for ToR, aggregation, and core switches, respectively. The link delay is 20μs so that the longest round-trip propagation delay is about 240μs, a commonly acceptable value in today's data center networks [2].

We make comparisons with NewReno + Droptail, DCTCP + ECN, and NewReno + SCCP. For the key parameters of DCTCP, we set g, the weighted averaging factor, to 1/16 and K, the buffer occupancy threshold for marking CE-bits, to 20 packets for 1Gbps links and 65 packets for 10Gbps according to [2]. For SCCP, the common RTT is set to 300μs and the window scaling factor [14] is 0 in all switches and servers. We also turn off the Silly Window Syndrome (SWS) function [15]

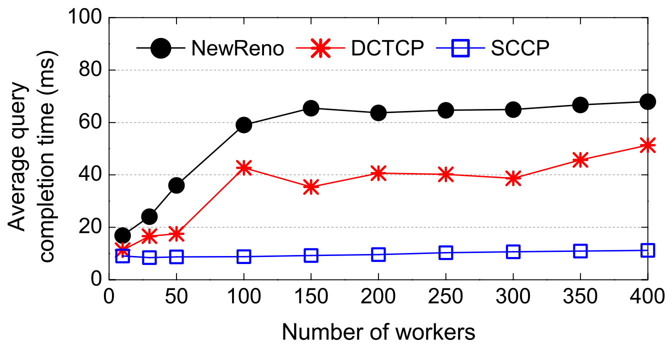


Fig. 4. Average query completion time with background traffic.

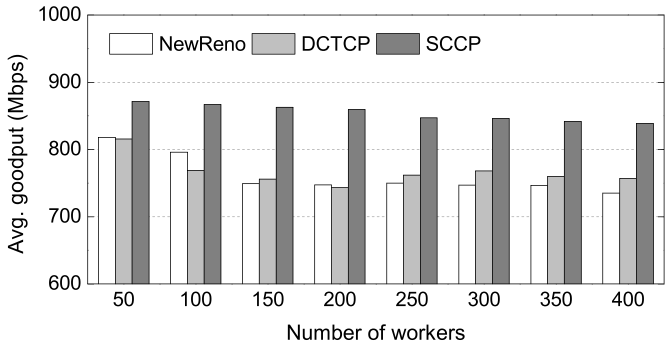


Fig. 5. Average goodput of the background flow.

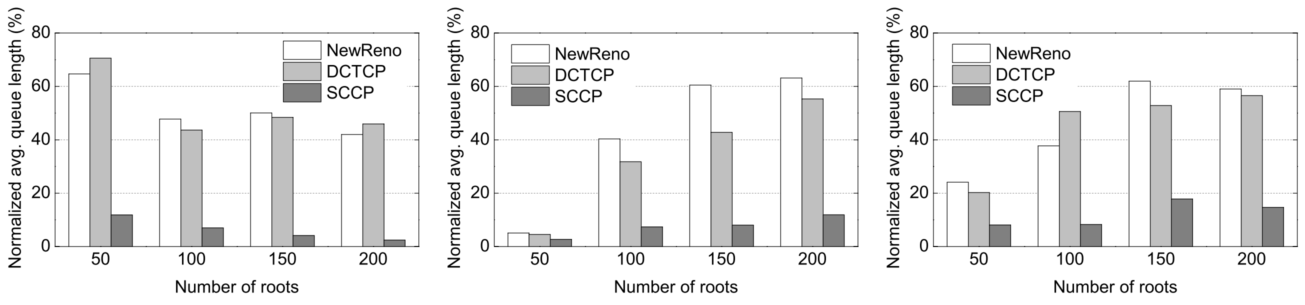
so that the servers can transmit a small amount of data, e.g., less than 1 Maximum Segment Size (MSS) immediately. The RTO_{min} for all TCP senders is set to 10ms.

To emulate the momentary bursts of traffic, we develop a Partition/Aggregate application that consists of one root and n workers; the root sends a query to its workers, and each worker responds with the requested amount of data. After all response data are received from the workers, the query completion time is measured. The measurements are repeated 100 times in all simulations.

A. Single root scenario

To investigate the scalability in terms of the number of flows, we conduct the basic *incast* experiment [2], [16], where the Partition/Aggregate application is deployed on the network with a single bottleneck port (in a ToR switch). We increase the number of workers, n, from 10 to 400. The root is located on the first server and the workers are distributed on the other servers so that each server has only one worker. The response data size of each worker is 1MB/n bytes. We also add one background flow that transmits 10MB of data to the root as the median number of concurrent large flows is 1 in data center networks [2]. The background flow fully utilizes the bottleneck link before the Partition/Aggregate application begins.

Fig. 4 shows the average query completion time. In this experiment, the minimum query completion time is about 8.4ms. As shown in the figure, SCCP achieves very low query completion time, close to the minimum number in all cases. It increases only by a few milliseconds even when the



(a) ToR switch (b) Aggregation switch (c) Core switch

Fig. 6. Normalized average queue length at the first port of each switch.

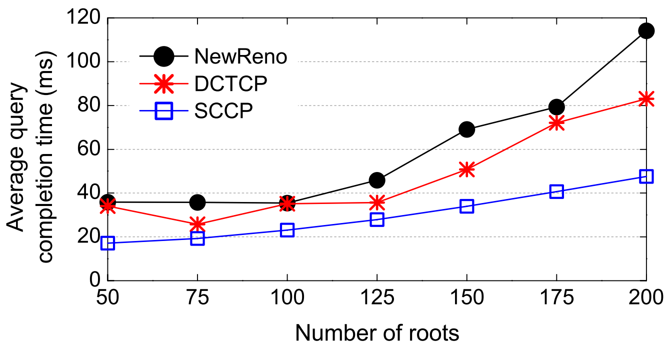


Fig. 7. Average query completion time of the roots.

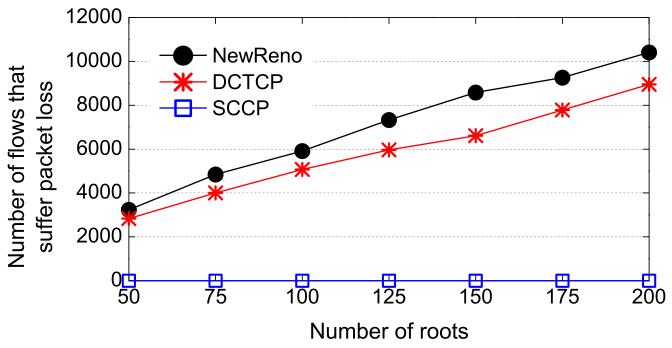


Fig. 8. Number of flows that suffer at least one packet loss.

number of workers is 400, showing the effectiveness of *fair-share* in SCCP. On the other hand, NewReno with the droptail queue experiences severe network congestion as the number of workers increases. Many flows suffer from multiple packet losses, inducing several TCP timeouts. As a result, the average query completion time of NewReno is larger than 60ms when the number of workers is more than 100. DCTCP performs better than NewReno by estimating the network congestion level with ECN, but its query completion time is still around 40ms when the number of workers is 100 to 300.

We also measure the average goodput of the background flow to see the link utilization of long-term traffic, normally generated by data update, backup, virtual machine migration, etc., as shown in Fig. 5. The overall goodput tend to decrease as the number of workers increases, but the goodput of SCCP is consistently higher than that of NewReno and DCTCP as much as 80–100Mbps in most cases. This signifies that even for the background traffic, avoiding network congestion is very important to improve the performance.

B. Multiple roots scenario

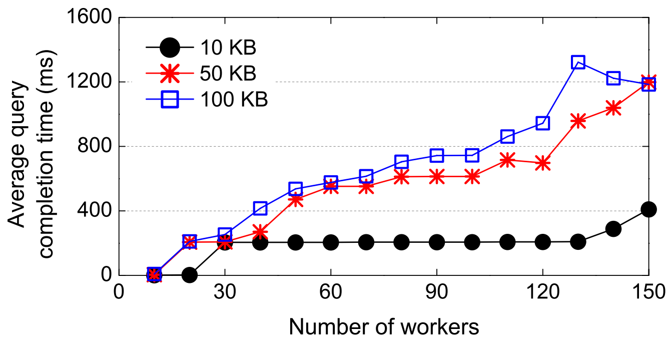
Now we deploy multiple Partition/Aggregate applications to see how the protocols work when the network congestion occurs at the aggregation/core switches as well. In this scenario, the roots are evenly distributed over the first 5 racks, and each root has 90 workers; 47 workers are located in the same rack and 35 workers are in different racks, but under the same aggregation switch. Lastly, 8 flows come from other racks via the core switch. The transmitting data size of each worker is 20KB.

Fig. 6 shows the average queue length, normalized to the buffer size at the ToR, aggregation, and core switches. When the number of roots is 50, the main bottleneck port is in the ToR switch. As the number of roots increases, the queue lengths of the aggregation and core switches gradually increase, indicating that the traffic bottleneck spreads to the aggregation and core switches. Through this scenario, we confirm that SCCP keeps low average queue length (less than 20%) in all layers whereas NewReno and DCTCP induce high queue length at the congested ports.

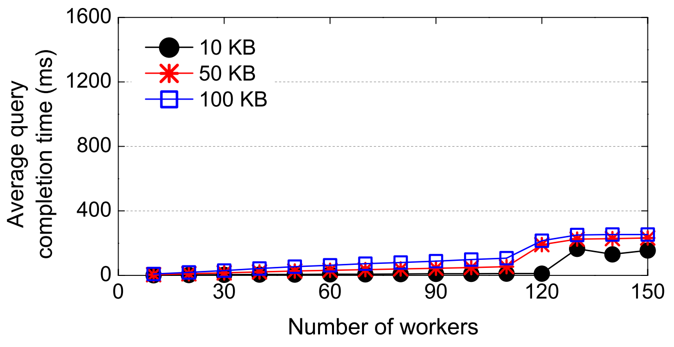
In Fig. 7, we measure the average query completion time as the number of roots increases, and it is observed that NewReno and DCTCP increase up to about 80ms and 115ms respectively, while SCCP shows under 45ms even when the number of roots is 200. We also observe that SCCP achieves zero packet loss in all cases as shown in Fig. 8.

V. Prototype Experiment

To show that SCCP works well in real data center environments, we implemented the prototype of SCCP in the Open vSwitch (version 2.3.0) [5], which currently supports OpenFlow 1.1, 1.2, and 1.3, based on Linux kernel 3.11.0. We deployed the prototype on our data center testbed; the testbed consists of 32 servers, two ToR switches, and an Aggregation switch. All servers are Dell OptiPlex 790DT with a quad-core Intel i5 3.3 GHz processor, 4GB RAM and 1Gbps Ethernet interfaces, and connected to ToR switches with 1Gbps links. The ToR switches are connected to the Aggregation switch



(a) Default OVS



(b) SCCP-enabled OVS

Fig. 9. Prototype experiment: OVS vs. SCCP-enabled OVS. Average query completion time is measured increasing the number of responding workers as well as the response data size of each worker.

with 10G links. The packet buffer size per switch port is set to 178KB for the ToR switches. The RTO_{min} for Linux TCP is 200ms. The measurements are repeated 10 times and averaged.

Since OVS runs on the servers, not the physical switches, our prototype works only for a ToR switch port where the OVS-server is directly connected. In our prototype, we set the minimum value of the fair-share to 1 MSS because we did not turn off the SWS function in the experiments. The common RTT is set to 600μs.

In Fig. 9, we compare the default OVS with SCCP-enabled OVS; we measure the average query completion time as the number of workers increases, varying the response size of each worker to 10KB, 50KB, and 100KB. With the default OVS, we observe that it suffers from TCP timeouts when the number of workers is more than 20–30, resulting in high query completion time, up to about 1300ms. On the other hands, SCCP achieves very low query completion time when the number of workers is under 110, which is consistent with our simulation results. With more than 120 workers, packet-buffer overflow (i.e., TCP timeouts) is inevitable as the fair-share cannot be smaller than 1 MSS. We expect that SCCP will achieve scalability even after 120 workers if we control the SWS function, but we leave this as future work.

VI. Conclusion

In this paper, we propose a new scalable congestion control protocol, called SCCP, that is implemented in the data center SDN switches. To minimize service latencies, i.e., provide better quality of service, in the data centers, it is essential to fundamentally address the network congestion, caused by the momentary bursts traffic. To deal with a large number of bursty flows, we design SCCP to provide more exact information, i.e., the fair-share of each flow, to the senders quickly via the advertisement window field in the TCP header.

The SCCP function can easily deployed to the existing data center network SDN switches, by slightly extending the existing OpenFlow specifications. Furthermore, SCCP does not require any change to the end-hosts and their applications. Through prototype experiments and ns-3 simulations, we confirm that SCCP efficiently avoids the network congestion at all data center switches, which results in low query completion time for short-bursty flows and high goodput for long-background flows.

Acknowledgment

This work was supported by the National Research Foundation of Korea Grant funded by the Korean Government (NRF-2013R1A1A1006823). This work was also supported by MSIP, Korea, under the Research Project on High Performance and Scalable Manycore Operating System (#14-824-09-011).

References

[1] T. Hoff, “Latency Is Everywhere And It Costs You Sales - How To Crush It,” Jul. 2009, http://highscalability.com/blog/2009/7/25/latency-is-everywhere-and-it-costs-you-sales-how-to-crush-it.html.
[2] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, “Data Center TCP (DCTCP),” in *Proc. ACM SIGCOMM*, 2010.
[3] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowstron, “Better Never than Late: Meeting Deadlines in Datacenter Networks,” in *Proc. ACM SIGCOMM*, Aug. 2011.
[4] “OpenFlow Switch Specification version 1.5.0.” [Online]. Available: https://www.opennetworking.org
[5] “Open vSwitch.” [Online]. Available: http://www.openvswitch.org/
[6] L. S. Brakmo and L. Peterson, “TCP Vegas: End to End Congestion Avoidance on a Global Internet,” *IEEE J. Sel. Areas Commun.*, vol. 13, no. 8, pp. 1465–1480, Oct. 1995.
[7] K. Ramakrishnan, S. Floyd, and D. Black, “The addition of explicit congestion notification (ECN) to IP,” RFC 3168, IETF, Sep. 2001.
[8] J. Hwang, J. Yoo, and N. Choi, “IA-TCP: A Rate Based Incast-Avoidance Algorithm for TCP in Data Center Networks,” in *Proc. IEEE ICC*, 2012.
[9] ——, “Deadline and Incast Aware TCP for cloud data center networks,” *Computer Networks*, vol. 68, pp. 20–34, Aug. 2014.
[10] T. Benson, A. Akella, and D. A. Maltz, “Network Traffic Characteristics of Data Centers in the Wild,” in *Proc. of ACM IMC*, Nov. 2010.
[11] D. Zats, T. Das, P. Mohan, and R. Katz, “DeTail: Reducing the Flow Completion Time Tail in Datacenter Networks,” in *Proc. ACM SIGCOMM*, Aug. 2012.
[12] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, “Hedera: Dynamic Flow Scheduling for Data Center Networks,” in *Proc. USENIX NSDI*, 2010.
[13] “The ns-3 discrete-event network simulator.” [Online]. Available: http://www.nsnam.org/
[14] V. Jacobson, R. Braden, and D. Borman, “TCP Extensions for High Performance,” RFC 1323, IETF, May 1992.
[15] D. D. Clark, “Window and Acknowledgement Strategy in TCP,” RFC 813, IETF, Jul. 1982.
[16] V. Vasudevan, A. Phanishayee, H. Shah, E. Krevat, D. G. Anderson, G. R. Ganger, G. A. Gibson, and B. Mueller, “Safe and Effective Fine-grained TCP Retransmissions for Datacenter Communication,” in *Proc. ACM SIGCOMM*, 2009.