STCC: A SDN-oriented TCP congestion control mechanism for datacenter network

Yu Tianfang | Qiu Xuesong

State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing, China

Abstract
Due to characteristics of high bandwidth and low latency, datacenter networks ensure tremendous data could be transmitted in an efficient way. However, in many-to-one transmission scenarios, high concurrence of TCP flows aggravates network congestion and causes overflows in switches, seriously impairing network performance. To solve the problem, a TCP congestion control mechanism based on software-defined networking (STCC) is proposed. Without any modification in TCP stack, STCC monitors network performance through centralized control and global network view of SDN, employs a routing algorithm based on the minimum path bandwidth utilization rate to forward packets and uses different methods to adjust congestion windows of senders so that network congestion can be greatly mitigated. An experiment platform is built to carry out simulation tests for evaluating STCC, and the results show that under the same network conditions, STCC effectively reduces the number of retransmission timeout of senders and noticeably raises network throughput, compared with other congestion control algorithms.

1 | INTRODUCTION

With huge amounts of computing resources and storage space, modern datacenters [1] can provide users with diverse kinds of network services. To give fast responses to application requests, datacenters adopt high concurrent network transmission [2] and the task disintegration process based on partition/aggregation pattern. However, under this particular communication mode, employing traditional TCP protocol to control network congestion may lead a serious decline of network performance. There is a typical problem called TCP Incast. In a many-to-one communication scenario, multiple senders send server request units (SRUs) to a single receiver simultaneously; when the number of senders continuously increases, these synchronized flows easily make buffers of the bottleneck switch overflow, and as a result, lots of packets are dropped and senders have to wait for timeout of timers and retransmit these packets. During this session, TCP connections experience many times of retransmission timeout (RTO) and turn into a global synchronization status. Consequently, the goodput of these connections will go down strikingly.

The fundamental cause of TCP Incast is that TCP protocol is initially designed for Internet and lacks the well-adapted capacity for datacenter environments that own the characteristics of high bandwidth and low delays. The main differences between the Internet and datacenters are that, firstly, nowadays 1 Gbps and 10 Gbps links in datacenters are pervasive. On the contrary, there are still lots of 1 Mbps and 10 Mbps links in Internet. Secondly, the propagation delay in datacenters is usually within tens of microseconds while the propagation delay in Internet is often tens of milliseconds. Most importantly, commercial shallow-buffered switches are adopted in datacenters, so the switches easily give rise to overflows and congestion, as opposed to Internet router buffers which are hundreds of Mbps deep.

Software-defined networking (SDN) [3] paves a new way for TCP congestion control in datacenters. First, SDN obtains global network topology information more easily because of its centralized control and high programmability. Second, SDN acquires detailed statistics of ports and queues of each switch by means of OpenFlow messages [4]. We present and evaluate STCC, a new SDN-based TCP congestion control mechanism that aims to improve the overall network performance and utilization of modern datacenters using the legacy TCP protocol. STCC can realize this by centrally collecting the network properties and subsequently compute and adjust congestion

This is an open access article under the terms of the Creative Commons Attribution License, which permits use, distribution and reproduction in any medium, provided the original work is properly cited.

© 2020 The Authors. IET Networks published by John Wiley & Sons Ltd on behalf of The Institution of Engineering and Technology.
control parameters suitable for the operating environment. By means of OpenFlow, STCC can be deployed into a controller to collect network metrics, such as bandwidth, delay and switches' buffer sizes. Once these metrics are collected, STCC can calculate an optimal path for each flow and quickly detect congestion occurrence according to network performance variation. What is more, STCC is able to take different strategies to eliminate the effect of network congestion in terms of the congestion degree of paths. The main contributions of this paper are summarized as follows:

i. We build a mathematical model to explain TCP Incast issue from angles of switch queue, packet losses and RTO
ii. We design an optimal routing algorithm to minimize the emergence of traffic congestion occurred on the forwarding paths
iii. We present a SDN-based congestion control algorithm that can self-adaptively regulate senders' congestion windows according to the congestion status of the forwarding paths
iv. We also expound the TCP traffic control in the kernel of mainstream operation system in order to explain the impact of the advertised window (AWND) of the receiver side on congestion control

2 | BACKGROUND

With the rapid development of network technology, as the mainstream architecture of datacenter networks, traditional tree topology incurs a chain of problems. First of all, the increasing number of servers makes it very easy to let core switches become bandwidth bottlenecks. Next, shallow-buffered commercial switches are used to forward traffic. When a large number of packets arrive in switches, there is a tendency that buffers overflow and network performance seriously descends. Third, inter-server traffic has dramatically increased due to the demand of various applications in datacenters. Authors in Ref. [5] argued that elephant flows (long-lived and throughput-sensitive) account for less than 10% of all flows but carry more than 80% of the total traffic volume. By comparison, mice flows (short-lived and delay-sensitive) have a proportion of more than 90% of all flows and the size of them is usually less than 10 KB. These flow characteristics lead an exceeding unbalance on network traffic distribution. In a word, network congestion is a global issue and each switch in the network may be a bottleneck device because of dynamic variation of traffic distribution. Figure 1 illustrates a many-to-one communication scenario, that is, the simplified topology of a distributed file system. Suppose that the topology conducts traffic scheduling using Equal-Cost Multipath Routing (ECMP) protocol, and each datanode $D_i$ ($i = 1, 2, \ldots, 5$) sends SRUs to the namenode through alternative paths. For edge switches $E_2\sim E_5$, ECMP can allocate a same egress port for every two adjacent datanodes. So different kinds of flows, such as elephant flows and mice flows, have to compete for limited bandwidth resources of these ports. In addition, when traversing the bottleneck switch $C_1$, plenty of SRUs also saturate the buffer of $C_1$, leading the emergence of TCP Incast and a huge decline of the namenode's goodput. It is clear that in many-to-one communication patterns, traffic congestion places include intermediate switches or/and the bottleneck switches of the forwarding paths. So it needs to employ different methods to control network congestion according to current status of these paths.

3 | RELATED WORK

3.1 | Existing proposals and analysis

Over the past few years, there have been many solutions for TCP Incast and congestion control in datacenters. TCP Vegas [6] is a delay-based congestion control strategy that uses end-
to-end queuing delay as a congestion feedback to proactively reduce sending rate before packet losses appear. Data Center TCP (DCTCP) [7] leverages the explicit congestion notification (ECN) technology to adjust senders’ congestion windows (CWND) and slow down sending rate before the queue becomes full. By counting up the number of ECN-marked ACK packets, DCTCP achieves fine-grained congestion control. However, ECN technology is limited by datacenter hardwares and how to set the ECN threshold properly is also a challenge. Incast Congestion Control for TCP (ICTCP) [8] is designed for the receiver side in datacenters. It measures bandwidth available and per-flow throughput in each cycle, and only increases the AWND when the measured throughput is very close to the expected throughput. Although the experiment results show that ICTCP can achieve almost zero timeout and high throughput, it is only suitable for the scenario in which the bottleneck link is the last-hop to the receiver. Guarantee Important Packets (GIPs) [9] use boundary information of stripe units to adjust congestion window and avoid timeout caused by ACK missing or window message loss. However, GIPs need to modify TCP/IP stack. Multi-path TCP (MPTCP) [10] employs concurrent transmission to redirect packets from overloaded paths onto light-loaded paths, yet MPTCP ignores the otherness of bandwidth and delay of each path, and these features bring a strong influence on congestion control.

3.2 | Congestion control techniques using SDN

The introduction of SDN in datacenters has provided new opportunities enabling fine-grained traffic engineering inside the switching fabric for congestion control in an efficient manner. The centralized SDN controller is in a predominant position to handle traffic scheduling for poising the load on the network because it has global visibility of the complete topology, the current traffic distribution and the load on individual network elements. Leveraging the superiorities of SDN, some researchers propose SDN-based solutions for addressing incast congestion in datacenters. Authors in Ref. [11] emphasize that increase in buffer size at OpenFlow switches suspends the onset of TCP Incast and supports high concurrent transmission from more servers. But modifying the switch hardware is a costly alternative and can only be applied to small-scale datacenters. Moreover, excessive buffering may increase round-trip time (RTT) and even lead to TCP timeouts, thus affecting the TCP standard congestion control measures. Hedera [12], a dynamic flow scheduling strategy for SDN-enabled datacenter network, which collects flow information from edge switches in order to identify the elephant flows, and uses two kinds of placement algorithms to realize flow re-routing even with large number of hosts. In Hedera, a forwarding path will be first selected when the path can accommodate the bandwidth request of an incoming flow, yet if there are multiple reachable paths between a pair of hosts, the selected path may not be an optimal path due to its bandwidth utilization, and this still leads an unbalance of traffic distribution. eSDN framework [13] employs lightweight end-host controllers for collecting network status information and sends these statistics to the centralized SDN controller for identifying congestion. But this master-slave work mode not only overloads the central controller, but also brings network a very high cost. Fair Data Center TCP (F-DCTCP) [14] monitors queue occupancy and number of concurrent connections of switch outgoing ports. For an outgoing port, if its current queue length or the number of parallel connections exceeds the defined threshold, F-DCTCP will compute the optimal transfer rate for all the flows over this congested port using Max–Min fair algorithm and adjust senders’ sending window through modifying received window of TCP ACK packets. However F-DCTCP is limited by the OpenFlow version and raises the competition time of flows. Based on SYN/FIN arrival rate, SDN-based Incast Congestion Control (SICC) [15] forecasts incast events and directly sends customized incast ON/OFF messages to the involved senders’ virtual machine addresses. Meanwhile, SICC rewrites advertised window of incoming ACK packets so that all flows are throttled to a sending rate of 1 MSS per RTT. Although this enables the congested switches to keep the queue below congestion threshold, there is a lack of a long-term fairness among flows, because for delay-sensitive flows, a big decline of sending rate may violate their deadlines and normal services have to be suspended. Omniscient TCP (OTCP) [16] utilizes SDN’s centralized network management capabilities to overcome TCP Incast. By means of a SDN controller, OTCP collects network metrics and computes retransmission timers, RTO and maximum congestion window that matches the bandwidth delay product between hosts, and then OTCP distributes these congestion control parameters into hosts by a JSON/REST northbound API. However OTCP needs a kernel level modification and also brings additional overhead of delay measurements. SDTCP [17] aims to allocate more bandwidth resource for mice flows by minimizing the share of elephant flows during the onset of incast congestion. It measures the queue length of each switch port and selects elephant flows according to traffic volume and lifetime of a flow. SDTCP calculates the fair share of all flows traversing the bottleneck link and adjusts sending rate of selected background flows by rewriting the AWND field in TCP header at switch.

4 | PROPOSED METHODOLOGY

According to Section 3, we know that there are two main causes for TCP Incast. One is triggering RTO. Overflow occurred on a switch buffer usually brings serious packet losses. If multiple senders have to wait for timeout of TCP retransmission timers, the network goodput will dramatically fall. Another cause is the impact of synchronized blocking. Among all senders, if a sender is ready for RTO, it must wait for a duration of the minimum RTO (RTO_{min}), and then retransmit lost packets. This obviously reduces link utilization rate. At present, the existing TCP Incast models can be
categorized into two groups: analysis models and data fitting models. Analysis models predict network goodput by simulating TCP dynamic transmission process, but these models are only suitable for specific TCP protocol version. Data fitting models get throughput functions via fitting experimental data; however some key parameter settings need plenty of experimental data and practical experience. In summary, there is a lack of critical system variables, such as end-to-end congestion control parameters, in either analysis models or in data fitting models. Therefore, it is difficult to dissect the TCP Incast problem in a comprehensive way. On the basis of description above, we build a mathematical model to explain the TCP Incast issue from angles of switch queue, packet losses and RTO. Furthermore, we design STCC and dwell on its realization mechanisms and algorithms. Table 1 lists main notations used in the section.

### 4.1 Incidence model of TCP Incast

To depict conditions that cause TCP Incast, we make several assumptions that switches manage their buffers with First Input First Output and Droptail mechanism, that there are no packet losses on switch ports and that hosts support timestamp function.

For a TCP flow, its forwarding path can be denoted as $<S_1, S_2, \ldots, S_k>$ ($S_k \in \mathcal{S}$), when $i = 1, 2, \ldots, k-1$ and ($S_i, S_{i+1}) \in \mathcal{E}$. For $S_0$, we assume that TCP flows traversing the egress port $m$, come from $n$ different ingress ports, and the current queue length of $m$ is $Q_m$. Moreover, we let $I_{rx}$ denote the number of received packets of an ingress port. We also let $I_{tx}$ represent the number of transmitted packets of an egress port. If $t$ is the new time the network runs, then $Q'_m$

and $PS_m$ can be expressed as Equations (1) and (4), respectively.

$$Q'_m = Q_m + Q'_m - Q_{m\text{lost}}$$

$$Q'_m = \sum_{j=1}^{n} t \times I_{rx}$$

$$Q_{m\text{lost}} = t \times I_{rx}$$

$$PS_m = \begin{cases} 
B_m - Q'_m & \text{if } Q'_m < B_m \\
\frac{Q'_m}{B_m} & \text{if } Q'_m \geq B_m 
\end{cases}$$

When $PS_m \geq 1$, it means that the number of packets arriving in port $m$ surpasses $B_m$, so those new incoming packets will be directly abandoned. In term of Equation (4), we give a definition of congestion status of a switch, that is, if any one of egress ports of the switch congests, then the switch gets into congestion status. Furthermore, we consider it as an independent event that each switch on the forwarding path congests at a moment. So for a path $r$ ($r \in \mathcal{E}$), $PS_r$ can be represented as Equation (5), where $k = L-1$. When $PS_r < 1$, it suggests that the path is in congestion-free status. Otherwise some links on the path become crowded.

$$PS_r = \text{Max} \left\{ \frac{1}{k} \cup PS_i \right\}$$

In the process of TCP congestion control, there are two important repetitive procedures where congestion control parameters are modestly set to adjust sending rate in terms of network status, including slow start and congestion avoidance. A slow start suggests that a sender initially sets CWND to 1 and increases CWND by 1 when receiving a new ACK packet. For instance, in the first transmission round, we send 1 packet; in the second, we send 2 and in the third, we send 4. Congestion avoidance means that the sender saves the half of current CWND as a threshold value and resets CWND to 1. As a consequence, slow start begins again until CWND reaches the threshold value. After that CWND slowly increases in a linear manner until it encounters congestion.

In a TCP connection, we assume that a sender’s sending window, congestion window and congestion threshold are $swnd$, $cwnd$ and $cthold$, respectively. If the advertised window of the receiver is $awnd$, then the sender will adjust $swnd$ according to Equation (6).

$$swnd = \text{Min} \{cwnd, awnd\}$$

When the sender goes into slow start, its $cwnd$ increases in an exponent way and reaches congestion avoidance via $\log_2$
(c) + 1 rounds. Suppose that the sender takes each different ACK packet as calculation evidence of RTT, then the first ST and MT can be expressed as:

\[ ST_1 = RTT_1 \]  
\[ MT_1 = \frac{RTT_1}{2} \]

Subsequently, for each new RTT, variables including ST, MT and RTO are recalculated as follows [18].

\[ ST_i = (1 - \alpha) \times ST_{i-1} + \alpha \times RTT_i \]  
where \( i \in [2, swnd] \) and \( \alpha = 1/8 \).

\[ MT_i = (1 - \beta) \times MT_{i-1} + \beta \times |ST_i - RTT_i| \]  
where \( i \in [2, swnd] \) and \( \beta = 1/4 \).

\[ RTO_i = ST_i + \text{Max}\{RTO_{\text{min}}, k \times MT_i\} \]  
where \( i \in [1, swnd] \) and \( k = 4 \).

In the \( n \)th transmission round, if the forwarding path is not in congestion status, then \( T_n \) can be formulated as Equation (12), where \( \phi = \text{swnd}_n \). But if the number of packet losses of sending window is \( \delta \) and those lost packets only need to be retransmitted once, \( T_n \) will change into Equation (13), where \( \phi = \text{swnd}_n - \delta \).

\[ T_n = \frac{\phi \times \text{MSS}}{\sum_{i=1}^{\phi} RTT_i} \]  
\[ T_n = \frac{\text{swnd}_n \times \text{MSS}}{\sum_{i=1}^{\phi} RTT_i + \sum_{j=1}^{RTO_i} RTT_j} \]

Comparing Equation (12) with Equation (13), we can see that \( T \) is inverse to the frequency of RTO, and the more serious packet losses take place, the sharper the throughput descends. So if there are multiple senders waiting for RTO simultaneously, TCP Incast inevitably occurs. According to analysis above, we present STCC, which includes two core components, that is, route selection and congestion control. Next, we will discuss design principles of the components.

### 4.2 Route selection

The main goal of route selection is picking a forwarding path that satisfies some restriction conditions, from the path collection of each TCP socket connection. For a path, the bigger its bandwidth available is, the smaller its congestion probability is and vice-versa. Therefore, we consider the bandwidth utilization rate of a path as the metric for choosing the optimal path. In SDN, OpenFlow protocol provides some controller-to-switch messages to ensure controller directly manages switches with great simplicity. As seen from Figure 2, STCC periodically sends PORT_STATUS and QUEUE_STATUS to switches to obtain overall statistics for each port and each queue. By parsing these statistics, STCC can calculate link bandwidth, port utilization rate and queue length. For a link \( l (l \in E) \), we let transmission data volume of the ingress port and the egress port of \( l \) be \( I_l \) and \( I_r \), respectively, during \( t \) seconds, and then the bandwidth available of \( l \) can be expressed as Equation (14). Therefore, for the path \( r \), its bandwidth available should be the minimum among all links on the \( r \) and can be formulated as Equation (15). Furthermore, if bandwidth demand of an incoming flow is \( \lambda \) and \( l \) can accommodate the flow, bandwidth utilization rate of \( l \) can be represented as Equation (16), and correspondingly, bandwidth utilization rate of \( r \) should be the maximum among all links on the \( r \), namely Equation (17).

\[ A_i = C_l - \text{Min}\{I_l, I_r\} \]  
\[ A_r = \text{Min}_{l \in r} (A_i) \]  
\[ U_l = \left( 1 - \frac{A_i - \lambda}{C_l} \right) \times 100\% \]  
\[ U_r = \text{Max}_{l \in r} (U_l) \]

Because traffic distribution is unbalanced in the network, we should limit the forwarding path length and least use heavy overloaded links to guarantee network QoS. To find an optimal path, we adopt Dijkstra’s algorithm [19] to realize the goal. Dijkstra’s algorithm is a graph search algorithm that solves the single-source shortest path problem for a graph with nonnegative edge path costs, producing a shortest path tree. In a weighted digraph \( G \), the shortest path between a given pair of source and destination vertices is defined as the set of links connecting the two vertices and whose sum of weights is the minimum among all other paths. Here, we let bandwidth utilization rate be the weight of a link, so for a sender and a receiver, bandwidth utilization rate of the optimal path should be the minimum among all reachable paths. Figure 3 shows the route selection process. For \( G \), if the number of nodes and edges is \( N \) and \( M \) respectively, then the time complexity of the route selection algorithm is \( O(N(M + N \log N)) \).

### 4.3 Congestion control

According to the TCP Incast mathematical model in Section 4.1, we define a queue length ratio threshold \( K (K < 1) \). For any egress port of a switch, if the ratio of current queue length and buffer size exceeds \( K \), then the switch becomes
congested. Given that in a network with $X$ hosts, there are $Y$ switches in congestion status at a transmission round, STCC needs to implement two tasks to mitigate network congestion. One is locating those congested switches, and another is finding paths in congestion status and taking measures to self-adaptively adjust CWND of affected hosts according to the congestion level of different paths. Therefore we propose a congestion control algorithm and describe it as follows.

i. STCC picks those paths that contain the $Y$ switches. For each path in the collection, if the number of congested switches is $\eta$ ($\eta \leq Y$), then the congestion level of the path $P_c$ can be expressed as Equation (18).

$$P_c = \frac{\eta}{L+1}$$

When $P_c \geq 1/2$, it suggests that the path is under serious congestion status, whereas if $P_c \in [1/4, 1/2)$, it means that the path suffers general congestion. Otherwise, if $P_c < 1/4$, the path is in light congestion.

ii. STCC picks those hosts correlated with paths in the collection, and by means of TCP socket technology, STCC sends congestion messages to them so that the hosts can timely adjust their sending rate. The congestion message format comprises a host IP and corresponding path congestion status.

iii. Once receiving a congestion message, a host immediately resets its congestion window $cwnd$ and congestion threshold $cthresh$, according to Figure 4.
5 | TCP TRAFFIC CONTROL

Traffic control basically suggests that TCP ensures that a sender is not overwhelming a receiver by sending packets faster than it can consume. Although there are some overlap between traffic control and congestion control, they are distinct features. Congestion control is about preventing a host from congesting the network, while traffic control concerns end-to-end data volume transmission.

In a TCP connection, every time the receiver receives a packet, it needs to send an ACK packet to the sender, acknowledging it received that packet correctly. This ACK packet maintains an AWND field, which is the current window size of the receiver. By continually advertising buffer space available to the sender, the receiver can control the sender’s sending rate and ensure the buffer space won’t overflow.

Figure 5 illustrates the structure of the receiver buffer. In terms of TCP three-way handshake, TCP initializes main parameters including rcv_wnd, thresv and mwin as: (1) rcv_wnd = 10 × MSS, (2) thresv = rcv_wnd and (3) mwin = \(2^{(16 + \omega)} - 1\), where \(\omega\) is the window enlargement factor. Before rewriting a new advertised window, TCP firstly calculates current advertised window value cwin, which is expressed as: \(cwin = rcv_wup + rcv_wnd-rcv_nxt\). Next, TCP figures out a temporary window value \(twin\), according to Figure 6. Finally, TCP considers the bigger of \(cwin\) and \(twin\) as the new advertised window.

6 | PERFORMANCE EVALUATION

In this section, we select multiple metrics to evaluate STCC through simulation experiments, and compare it with the solutions in Refs. [6,7,17]. To simulate a many-to-one scenario, we choose the FatTree topology as the experimental topology. Unlike single-rooted tree topologies, the FatTree, which is a non-blocking and TCP/IP compatible topology for modern datacenters, can realize the full bisection bandwidth of clusters comprising thousands of nodes at lower cost and support multi-path routing technique for traffic engineering, requiring no changes to hosts and imposing only moderate modifications to the forwarding functions of the switches themselves. Due to high scalability and maintainability, we adopt component-based Mininet and Ryu as the network simulation platform and the SDN controller, respectively. As seen from Figure 7, we build a four-pod FatTree topology with four core, eight aggregation and eight Top-of-Rack switches using a testbed that owns a dual-core processor with 3.2 GHz, 3 GB memory and 120 GB disk space. Table 2 lists all of parameter settings in our experiments. To conveniently configure these parameters and build interconnectivity between virtual hosts and practical network, we choose Ubuntu 14.04 LTS with
kernel version 3.3.2 as the system environment because this Linux-based operation system not only includes common TCP variant modules and well supports the congestion control protocols using ECN feedback mechanism but also provides lots of third-party plugins for developing SDN applications and complex data processing. In our experiments, we set TCP New Reno as the default TCP protocol and select OpenFlow protocol 1.3 to keep consistency with the OpenFlow virtual switch 2.3. In addition, we utilize Iperf to produce many-to-one traffic distribution. For each host, we let it launch TCP requests towards the receiver in a parallel way. Considering limitation of system resources, we set bandwidth of each link in the topology as a same value, and this setting is also applied to delay of each link. Similarly, we employ the default RTO and CWND in Ubuntu, as the minimum RTO and senders’ initial congestion window, respectively.

6.1 | Experiment results

Figure 8 demonstrates practical transmission time of each algorithm. It is clear that the elapsed time of STCC is much less than that of other three algorithms. Undoubtedly, queuing delays and RTOs raise data transfer time. Compared with other algorithms, STCC forwards packets using the optimal path based on minimum bandwidth utilization rate, and what is more, STCC monitors queue length of switch ports in real time and self-adaptively adjusts senders’ sending rate so that network congestion can be greatly mitigated.

As shown in Figure 9, senders’ average throughput in each algorithm sharply falls at \( t = 2 \) s. After that, STCC keeps its throughput above 500 Kbps all the time, and for DCTCP and VEGAS, the throughput curves range from 400 Kbps to 500 Kbps, whereas SDTCP still holds its throughput merely below 400 Kbps. In Figure 10, bandwidth of bottleneck link (\( E_9, H_9 \)), gradually grows with the increase of senders and reaches the peak at \( t = 8 \) s. When more and more packets are buffered by the receiver, there are only a few data volumes left to be transmitted in the network, so bandwidth utilization of the bottleneck link starts to drop from \( t = 9 \) s. The main cause of these phenomena is that in a many-to-one communication scenario, concurrent flows make it very easy to lead the bottleneck switch to overflow, resulting in a sharp decline of senders’ goodput. Among the four algorithms, DCTCP adopts a single threshold to control network congestion, but there exists a round-trip delay, which incurs a fierce variation of queue length of switch ports and brings about bandwidth waste. VEGAS employs the minimum RTT to calculate expected throughput and adjusts congestion window by the difference of the expected

| Parameter                                      | Value          |
|------------------------------------------------|----------------|
| Senders                                        | \( H_1 \rightarrow H_7 \) |
| Receiver                                       | H_6           |
| Concurrent flows each sender sends             | 4             |
| Bandwidth of a link                            | 10 M           |
| Delay of a link                                | 1 ms           |
| Buffer size of a switch port                   | 50 packets     |
| Initial CWND of each sender                    | 10MSS          |
| Payload of a packet                            | 1200 bytes     |
| Transmission time of each sender               | 10 s           |
| Minimum RTO                                    | 200 ms         |
| Cycle of monitoring link bandwidth             | 1 s            |
| Cycle of monitoring switches                   | 0.01 s         |
throughput and the goodput. Although this method can keep senders’ sending rate steady, it also causes unfairness of bandwidth allocation and limits bandwidth utilization. Similarly, SDTCP selects a flow with the longest lifecycle and calculates its CWND using average congestion window of flows traversing the same port. However, SDTCP does not alter CWND directly, but update AWND field of ACK packets using a fixed parameter, which cannot adjust advertised window in a self-adaptive manner. On the contrary, STCC can adjust senders’ sending rate in the light of congestion status of different forwarding paths, and this not only reduces congestion occurred on nodes, but efficiently elevates output bandwidth of the bottleneck switch.

In Figure 11, we can see that STCC keeps buffer space of switches much ampler than that in other algorithms. This is because with the help of the optimal path selection algorithm and centralized control of SDN, STCC quickly diagnoses congestion nodes through monitoring variation of queue length of switch ports and directly notifies senders of adjusting their CWND. In contrast, DCTCP cannot ensure those ACK packets with ECN flag are received on time when network congests, so senders may not adjust their sending rate in a RTT, and this brings longer queuing delays. For VEGAS, it will recollect RTT samples if the route changes. When a new RTT is even longer, VEGAS cannot deduce whether this RTT derives from route change or comes from network congestion, so RTT estimation failure obviously increases queue length of switch ports. To mitigate network congestion, SDTCP merely modifies receiver’s advertised window, but if there are multiple congested switches, ACK packets may need a long time for transmission, and during this session, senders still keep their sending window unchangeable so that more and more packets have to queue for propagation.

As we can see from Figure 12, STCC sustains its average RTT always below 40 ms, yet DCTCP and VEGAS let their RTT values range from 36 to 42 ms, and SDTCP even makes the RTT curve vary around 45 ms. In Figure 13, it is clear that
the number of retransmitted packets is directly proportional to run time, and moreover, the more run time consumes, the more retransmitted packets there are. Generally, RTT closely correlates with transmission delay and queuing delay. Similarly, RTO is in connection with switches' congestion degree. Compared with STCC, DCTCP has an obvious shortcoming that congestion notifications may not be timely received by senders sometime, and this impacts RTT measurement and the queue steadiness of switch ports. Besides, DCTCP is unable to further adjust sending rate when the congestion window of some senders reaches to the minimum size despite receiving the ECN feedback yet, and this also easily results in the full window loss timeout and the lack of ACKs timeout. For VEGAS, although it uses an accurate system clock for the RTT calculation, and considers the receipt of certain ACK packets as a trigger to check whether a timeout happens, if there is no guarantee that ACK packets can be sent to senders in the same inverted path, RTTs and the number of retransmitted packets may notably increase. SDTCP ignores the complexity of TCP receiver buffer, and only chooses average sending window as the window size of longest-lived flow. At the same time, SDTCP simply leverages a fixed parameter to adjust AWND field of ACK packets so that congestion control is not as effective as expected.

7 | CONCLUSION

The distinct architecture and communication mode of datacenters offer a new challenge to traditional TCP protocol. Considering the shortcomings of existing TCP congestion control solutions, we present STCC, a SDN-based mechanism to calculate and manage congestion control parameters for the low-latency and high-throughput environment of datacenter networks. Using a centralized controller to periodically obtain topological information, STCC calculates an optimal forwarding path for each flow and quickly locates congested switches. Furthermore, STCC self-adaptively sets route-specific congestion control parameters according to the congestion status of each path so that network congestion can be greatly mitigated. We evaluate STCC via quantitative simulation experiments, and our results show that STCC significantly shortens flow completion time and packet round-trip time, improving the performance of soft-real-time applications. Moreover, STCC outperforms other congestion control algorithms for incast traffic, with a huge decrease of retransmitted packets as well as higher and steadier goodput.

ACKNOWLEDGEMENTS

This research was supported in part by the National Natural Science Foundation of China under Grant No. 61702048.

REFERENCES

1. Hafeez, T., et al.: Detection and mitigation of congestion in SDN enabled data center networks: a survey. IEEE Access. 6, 1730–1740 (2018)
2. Judd, G.: Attaining the promise and avoiding the pitfalls of TCP in the datacenter. In: Proc. 12th Conf. Netw. Syst. Des. Implementation, Berkeley, USA, pp. 145–157 (May 2015)
3. Khaleel, A., Al-Raweshidy, H.: Effective routing algorithm based on software defined networking for big data applications in data center network. In: 16th Int. Conf. Dependable, Autonomic and Secure Computing, Athens, Greece, pp. 842–848 (August 2018)
4. Singh, S., Jha, R.: A survey on software defined networking: architecture for next generation network. J. Netw. Syst. Manag. 25(4), 321–374 (2017)
5. Kandula, S., et al.: The nature of data center traffic: measurements and analysis. In: Proc. 9th Conf. Internet meas., New York, USA, pp. 202–208 (November 2009)
6. Abolfazli, E., Mansouri, V.: Dynamic adjustment of queue levels in TCP Vegas-based networks. Electron. Lett. 52(5), 361–363 (2016)
7. Zafar, S., Bashir, A., Chaudhry, S.: On implementation of DCTCP on three-tier and fat-tree data center network topologies, SpringerPlus 5(1), 766–784 (2016)
8. Wu, H., et al.: ICTCP: incast congestion control for TCP in data-center networks. IEEE ACM Trans. Netw. 21(2), 345–358 (2013)
9. Zhang, N., et al.: Taming TCP Incast throughput collapse in data center networks. In: 21st Int. Conf. Netw. Protocols, Göttingen, Germany, pp. 1–10 (October 2013)
10. Chaturvedi, R., Chand, S.: MPTCP over datacenter networks. In: Int. Conf. Inventive Commun. Comput. Technol., Coimbatore, India, pp. 894–898 (April 2018)
11. Phanishayee, A., et al.: Measurement and analysis of TCP throughput collapse in cluster-based storage systems. In: Proc. 6th Conf. File and Storage Technol., Berkeley, USA, pp. 1–14 (February 2008)

12. Chao, S., Lin, K., Chen, M.: Flow classification for software-defined data centers using stream mining. IEEE Trans. Serv. Comput. 12(1), 105–116 (2019)

13. Pirzada, H., Mahboob, M., Qazi, I.: 'eSDN: rethinking datacenter transports using end-host SDN controllers. ACM Comput Comm Rev. 45(4), 605–606 (2015)

14. Aina, J., et al.: F-DCTCP: fair congestion control for SDN-based data center networks. In: Int. Symp. Netw. Comput. Commun., Istanbul, Turkey, pp. 1–6 (June 2019)

15. Abdelmoniem, A., Bensaou, B., Abu, A.: SICC: SDN-based incast congestion control for data centers. In: Int. Conf. Commun., Paris, France, pp. 1–6 (May 2017)

16. Jouet, S., Perkins, C., Pezaros, D.: OTCP: SDN-managed congestion control for data center networks. In: Int. Network Operations and Management Symp., Istanbul, Turkey, pp. 171–179 (April 2016)

17. Lu, Y., Zhu, S.: SDN-based TCP congestion control in data center networks. In: 34th Int. Conf. Perform. Comput. Commun., Nanjing, China, pp. 1–7 (December 2015)

18. Paxson, V., et al.: Computing TCP’s retransmission timer (2011)

19. Wei, K., et al.: A modified Dijkstra’s algorithm for solving the problem of finding the maximum load path. In: 2nd Int. Conf. Inf. Comput. Technol., Kahului, USA, pp. 10–13 (March 2019)

20. Stevens, W., Fall, R.: TCP Data Flow and Window Management. TCP/IP illustrated, Volume 1 Second Edition: the protocols, Addison-Wesley Professional Computing Series, pp. 691–723. Addison-Wesley Professional Press (2011)

21. Seth, S., Venkatesulu, M.: TCP Receive. TCP/IP architecture, design and implementation in Linux, Practitioners, vol. 1, pp. 225–290. Wiley-IEEE Computer Society Press (2008)

**How to cite this article:** Tianfang Y, Xuesong Q. STCC: A SDN-oriented TCP congestion control mechanism for datacenter network. *IET Netw.* 2021;10:13–23. [https://doi.org/10.1049/ntw2.12005](https://doi.org/10.1049/ntw2.12005)