Resilient TCP Variant Enabling Smooth Network Updates for Software Defined Data Center Networks

Abdul Basit Dogar, Sami Ullah, Yiran Zhang, Hisham Alasmary *Member, IEEE,*
Muhammad Waqas, *Senior Member, IEEE,* Sheng Chen, *Fellow, IEEE*,

***Abstract*—Network updates have become increasingly prevalent since the broad adoption of software-defined networks (SDNs) in data centers. Modern TCP designs, including cutting-edge TCP variants DCTCP, CUBIC, and BBR, however, are not resilient to network updates that provoke flow rerouting. In this paper, we first demonstrate that popular TCP implementations perform inadequately in the presence of frequent and inconsistent network updates because inconsistent and frequent network updates result in out-of-order packets and packet drops induced via transitory congestion and lead to serious performance deterioration. We look into the causes and propose a network update-friendly TCP (NUFTCP), which is an extension of the DCTCP variant, as a solution. Simulations are used to assess the proposed NUFTCP. Our findings reveal that NUFTCP can more effectively manage the problems of out-of-order packets and packet drops triggered in network updates and it outperforms DCTCP considerably.**

***Index Terms*—Software defined data center networks, network updates, DCTCP, out-of-order packets, packet drop, SDN**

I. Introduction

Computer networks are dynamic, complex and have diverse critical infrastructures. In order to maintain the availability, correctness and performance of networks in an efficient manner, network operators frequently involve in various tasks of updating the routing policies, changing the security policies, recovering from link failures, migrating flows, etc. These tasks are termed as network updates [1]. In a modern data center network (DCN), network updates are becoming even more frequent as some new scenarios of network updates include: (i) virtual machines migration among physical servers, (ii) reconfiguration between a load balancer and its backend servers, (iii) examining the functionality and compatibility of a new switch on-boarding through moving traffic, (iv) installation of a new firmware version at the switch, and so on [2]–[4]. Network updates in software defined DCN (SD-DCN) may have various additional reasons and circumstances, which inevitably leads to reroute the traffic [5], [6].

Rapidly expanding companies have employed the software defined network (SDN), which provides considerable advantages over traditional DCN in managing data transfer. For example, Microsoft [7] and Google [8] interconnect their data centers with SDN to achieve high network performance. Google reports that the link utilization can attain near about 100%, whereas traditional networks can only achieve an average of 30% to 40% [8]. An SDN-enabled network often requires frequent and fast network updates to manage rapid flow rescheduling decisions and utilize a virtually centralized controller to fulfill the different requirements at the data plane. Specifically, the procedures for updating the rules are performed over non-synchronized machines, and implementing the properties of consistency with rules dependencies demands frequent and fast updates to avoid forwarding anomalies and exacerbated network performance [9]–[13].

During the execution of successive network updates, various processes are carried out in a non-blocking manner, thereby ensuring the consistency properties is crucial [14]. In SDN setups, frequent flow rescheduling, if not carried out carefully, may cause major network update issues referred to as network confusions [1], such as link congestion, network policy violation, forwarding blackhole, and forwarding looping. These issues lead to inconsistencies in network updates. As a result, problems such as out-of-order packets and packet drops occur. In order to preserve network consistency properties during updates, the majority of previous research has focused on link congestion [2], [7], [9], [11], [12], [15]–[18], forwarding blackhole and forwarding looping [2], [7], [9], [11], [20]–[22], and policy violations [10], [22].

In view of the aforementioned discussion on the network update scenarios and consistency properties maintenance, we notice the key observations as follows.

1) Almost every scenario involving network updates requires rerouting of the flows or traffic, which may lead to the issues of out-of-order packets and packet drops.
2) If the rescheduling of flows is not handled carefully, it may cause inconsistent network updates, resulting in transient congestion and rerouting. Therefore, out-of-order packets and packet drops can occur.

According to [23], observations show that transmission control protocol (TCP) traffic accounts for 99.91% of the traffic in data centers. However, the rerouting violates TCP's assumption and has a negative repercussion on TCP traffic, resulting in severe network performance degradation. When

The authors extend their appreciation to the Deanship of Scientific Research at King Khalid University for funding this work through large group Research Project under grant number RGP.2/312/44.

A. B. Dogar is with the Department of Computer Science and Technology, Tsinghua University, China and also with the Department of Informatics and Systems, School of Systems and Technology, Lahore, Pakistan. (e-mail: bas15@mails.tsinghua.edu.cn).

S. Ullah is with the Department of Computer Science, Shaheed Benazir Bhutto University, Sheringal, Upper Dir 18050, Pakistan, (e-mail: sami@sbbu.edu.pk).

Y. Zhang is with School of Computer Science, Beijing University of Posts and Telecommunications (e-mail: yiranzhang@bupt.edu.cn).

H. Alasmary is with the Department of Computer Science, College of Computer Science, King Khalid University, Abha 61421, Kingdom of Saudi Arabia, (e-mail: alasmary@kku.edu.sa).

M. Waqas is with the Department of Computer Engineering, Faculty of Information Technology, University of Bahrain, Bahrain, and also with School of Engineering, Edith Cowan University, Perth WA 6027, Australia (e-mail: engr.waqas2079@gmail.com).

S. Chen is with the School of Electronics and Computer Science, University of Southampton, Southampton SO17 1BJ, U.K. (e-mail: sqc@ecs.soton.ac.uk).

rerouting, out-of-order packets and packet drops are serious issues. When the network is updated, the predicted difficulties of TCP can be classified as below.

1) Certain disorders may result in the suppression of window size and unusual retransmission. Many researchers attempt to address the difficulties using timer or DUPACK threshold estimate to solve the problems.
2) TCP invariably begins with a "slow start". When rerouting a flow, a network however gives the new route with a "quick start" that can result in out-of-order packets and severe congestion, particularly if the updates are frequent and inconsistent or if the flow scheduling is imperfect.

Given these issues related to TCP, the performance of TCP is inevitably poor whenever the network is being updated. As pointed out previously, network update occurs frequently. For example, Hedera [3] updates the network every 5 s, and the authors of [3] believe even sub-second and possibly sub-100 ms networking updates are achievable, as evidenced by the work of [24], which updates the network using 1 ms, 5 ms and 1000 ms. However, TCP performs significantly worse when updates occur frequently.

There exist many TCP variants. However, to the authors' best knowledge, so far no one has studied how TCP variants will react to the aforementioned problems during the network updates or reroutes. To further investigate the issues related to TCP variants, in this paper, we first perform extensive experiments using cutting-edge TCP variants including DCTCP [23], CUBIC [25], and BBR [27] to observe their behaviors during network updates. Our results show that CUBIC, BBR and DCTCP face the serious problems during inconsistent and frequent network updates, thereby confirming that the existing TCP variants are incapable of dealing with frequent and inconsistent network updates effectively.

A. Motivation

Motivated by this experimental investigation of TCP variants, we propose a TCP modification based on DCTCP, named network update friendly TCP (NUFTCP), to overcome the aforementioned issues when networks are updated in SD-DCN [26]. The key is to understand how existing TCP designs, particularly DCTCP, can be extended to handle network updates more gracefully. The fundamental challenges of smoothing network updates are handling packet reordering and avoiding packet drops, which may reduce the window size even though there is no actual congestion. By limiting window size and delaying duplicated acknowledgments (ACKs), NUFTCP can perform better than DCTCP, especially when the updates of networks are inconsistent and frequent. We investigate how the proposed NUFTCP can cope with frequent and inconsistent network updates in detail and demonstrate through simulations that our model accurately captures the essences of network updates. Our NUFTCP remains resilient at frequent and inconsistent network updates, and it outperforms DCTCP. The main contributions of this paper are summarized as follows.

1) Extensive experiments are conducted and the results are analyzed to better understand and pinpoint "flaws" in the current TCP designs utilized in data centers when networks are updated frequently and inconsistently.
2) We focus on the issues with out-of-order packets and dropped packets caused by network updates, which substantially impacts the real-time transmission of data.
3) We propose a novel NUFTCP, a TCP extension of DCTCP that gracefully handles network updates.
4) Simulations are used to evaluate NUFTCP, and the results obtained reveal that it achieves more satisfactory performance in SD-DCN during inconsistent and frequent network updates.

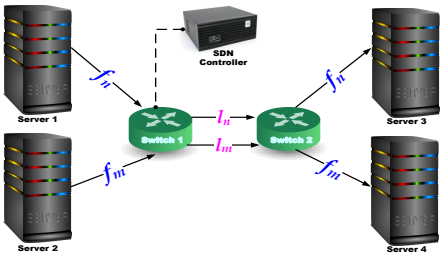

Fig. 1. Network topology for network updates with TCP variants.

The remaining sections of the paper are organized as follows. In Section II, we present extensive experiments and detailed analysis of existing TCP solutions with an example to demonstrate why gracefully handling network updates are crucial in SD-DCNs. Section III is devoted to our proposed NUFTCP design. We use simulation based evaluations to demonstrate that NUFTCP works well in different scenarios in Section IV. In Section V, the relevant literatures are reviewed, and in Section VI, we conclude this paper.

II. Analysis of TCP Variants in Network Updates

The performance of TCP is investigated in the context of inconsistent and frequent network updates. Specifically, we perform a series of tests to see how different TCP variants perform. The experimental results are analyzed and explained.

A. Experimental Setup

The topology we employed in the experiments is depicted in Fig 1. The network comprises four Intel Xeon X5650 servers with six cores running at 2.67 GHz that function as both senders (*Server 1* and *Server 2*) and receivers (*Server 3* and *Server 4*). We utilize the Linux kernel version 4.9. The senders and receivers are connected to the two switches, H3C S6800 (*Switch 1*) and H3C S6300 (*Switch 2*). Both the switches implement OpenFlow 1.3 and have a 10 Gbps link speed. They are connected by two ports. As a result, there are two ways for a sender to reach a receiver. A third port connects the SDN H3C VCF Controller to the switch.

The actual path of each flow is controlled by the controller. In the experiments, two groups of flows are used, and they are $\{f_n\}$: *Server 1* $\rightarrow$ *Server 3* and $\{f_m\}$: *Server 2* $\rightarrow$ *Server 4*. *Switch 1* receives these flows and then forwards to *Switch 2*. The network is shown in its starting condition in Fig. 1. The two groups of flows are forwarded as follows in their normal state: $\{f_n\}$: *Server 1* $\rightarrow$ *Server 3* through link l_n (*Switch 1* to

TABLE I
PARAMETERS OF EXPERIMENTAL SYSTEM

Parameters	Values
Packet MTU Size	1500 B
Queue Type	RED
Traffic Flow Pattern	Pre-Defined
Small Buffer Size	200 KB
Large Buffer Size	1.0 MB
Link Capacity	10 Gbps
Measurement Time	120 s - 300 s
ACK Delay Threshold	2 Packets
ACK Delay Timeout	200 ms
Reroute Time	1 s

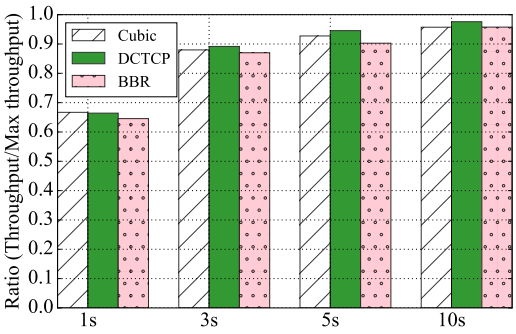

Fig. 2. Effect of network update frequency on performance of TCP variants. Network updates are consistent.

Switch 2); $\{f_m\}$: *Server 2* to *Server 4* through link l_m (*Switch 1* to *Switch 2*).

During the network update, routes change frequently, and the traffic on link l_n shifts to link l_m and vice versa. Then, the network state is as follows: $\{f_n\}$: *Server 1* to *Server 3* via link l_m (*Switch 1* to *Switch 2*); $\{f_m\}$: *Server 2* to *Server 4* via link l_n (*Switch 1* to *Switch 2*). Another scenario is when link l_n failure happens, and the network state is as follows: $\{f_n + f_m\}$: *Server 1* and *Server 2* to *Server 3* and *Server 4* via link l_m (*Switch 1* to *Switch 2*). The scenario of link l_m failure is similar.

To investigate the behavior of TCP variants during frequent network updates, we reroute the flow traffic initiated by *Server 1* towards link l_m and the flow traffic initiated by *Server 2* towards link l_n. These rerouting occur each time the controller initiates a network update. During the five-minute experiment, the duration for each route change is at most one second. The other system parameters for the experiment are specified in Table I. The experiment settings emulate unpredictable and frequent network update situations, which may result in link congestion, dropped packets, and out-of-order packets. Due to excessive retransmissions and reduced flow throughput, there will be a significant reduction in throughput.

B. Analysis for Consistent Network Updates

We first investigate the performance of the TCP variants in consistent network updates. The requirement for consistency demands that the transmission of flows is seamlessly toward new paths of routing or rerouting, in order to enforce the implementation rules with their interdependence in the flow tables across multiple switches on a routing path [28]. Therefore, the most important aspect of maintaining consistency is

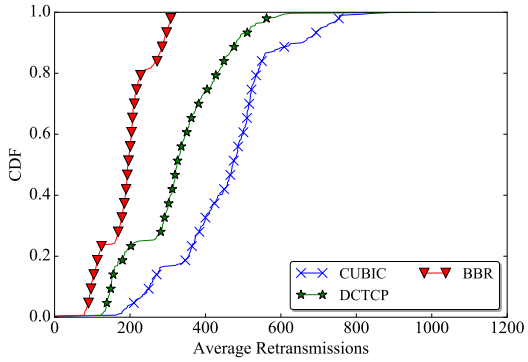

Fig. 3. Numbers of average retransmissions during consistent network updates for TCP variants with large buffer size.

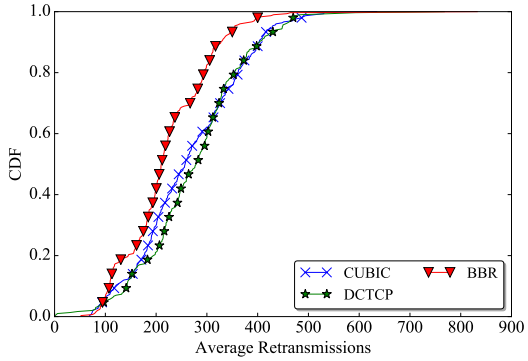

Fig. 4. Numbers of average retransmissions during consistent network updates for TCP variants with small buffer size.

the correct order of updates [29], and the updates follow the right sequencing are called consistent updates.

The impact of network update frequency on the achievable network performance of the TCP variants is investigated in Fig. 2. It can be seen that given the network update frequencies of ten seconds, five seconds, three seconds and one second, the corresponding network performance degradations are more than two percents, five percents, ten percents and thirty percents, respectively. In practice, frequent DCN updates often occur, initiated by operators, software or in unusual cases of failure [2]. The controller and data plane must quickly update in real-time because of the recurring flow dynamics [30]. Since tens of thousands of flows may occur in just a few milliseconds, high efficiency in network updates is crucial [7].

Below we examine how the TCP variants behave during consistent network updates, in terms of the total number of packet drops, average throughput, variation in congestion window size *cwnd* and average number of retransmissions. In the experiment, by defining a threshold, the SDN controller initiates a network update with reference to rerouting every second. Using the same topology with two scenarios of large buffer size and small buffer size, we run the experiment.

1) Number of Retransmissions: Figs. 3 and 4 plot the cumulative distribution functions (CDFs) of the numbers of average retransmissions for the three TCP variants during consistent network updates with large and small buffer sizes, respectively. There are different reasons for retransmissions, e.g., timeouts, damaged packet data, out-of-order packets, etc. For the case of large buffer and at the CDFs of 20%, 80%

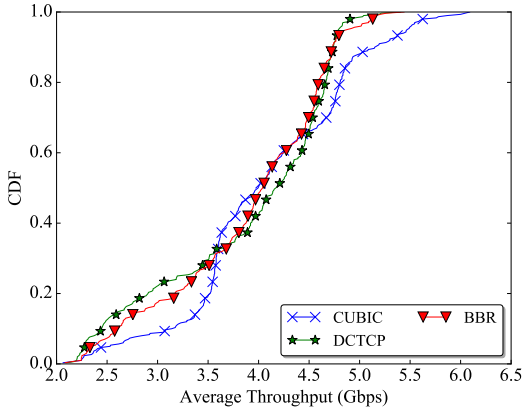

Fig. 5. Average throughput during consistent network updates for TCP variants with large buffer size.

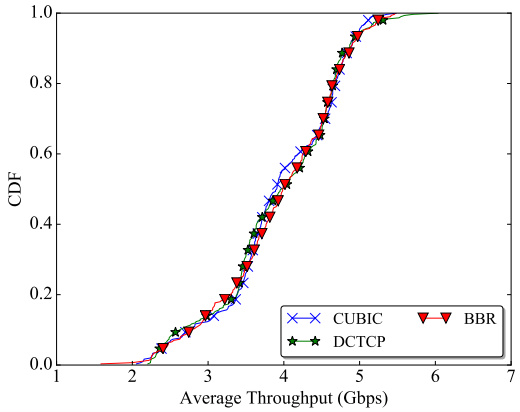

Fig. 6. Average throughput during consistent network updates for TCP variants with small buffer size.

and 100%, BBR requires 100, 200 and 280 retransmissions, and DCTCP requires 180, 400 and 600 retransmissions, while CUBIC imposes 380, 580 and 800 retransmissions. For the case of small buffer and at the CDFs of 20%, 80% and 100%, BBR requires 130, 300 and 500 retransmissions, and DCTCP requires 200, 350 and 550 retransmissions, while CUBIC imposes 180, 350 and 550 retransmissions. It is evident that BBR in the both cases outperforms the other two TCP variants. For BBR, larger buffer improves the performance and reduces the number of retransmissions, also see [31]. BBR attempts to find the optimal operating point during network updates by estimating the bandwidth and round-trip propagation delay to take care of bandwidth and round-trip time. BBR also ignores packet loss as a congestion signal [27], [32]. It can also be seen that with the small buffer size, the retransmission performance of DCTCP and CUBIC improve.

2) Throughput: Figs. 5 and 6 depict the CDFs of the average throughput for the three TCP variants during consistent network updates with the large and small buffer sizes, respectively. For the large buffer and at the CDFs of 20%, 80% and 100%, CUBIC can reach around 3.5 Gbps, 4.8 Gbps and 6 Gbps, and DCTCP achieves around 2.8 Gbps, 4.7 Gbps and 5 Gbps, while BBR reaches around 3.3 Gbps, 4.5 Gbps and 5 Gbps. It appears that CUBIC has the edge in this case. For the small buffer scenario, the three TCP variants have similar performance. Specifically, at the CDFs of 20% and 80%, the

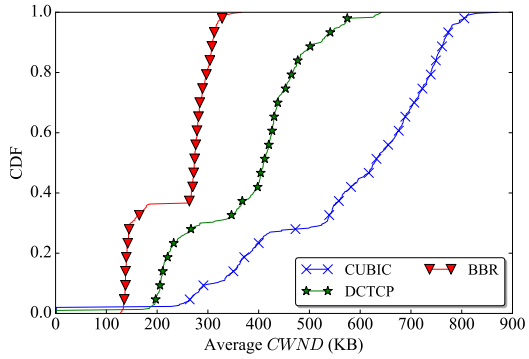

Fig. 7. Variations in average congestion window size during consistent network updates for TCP variants with large buffer size.

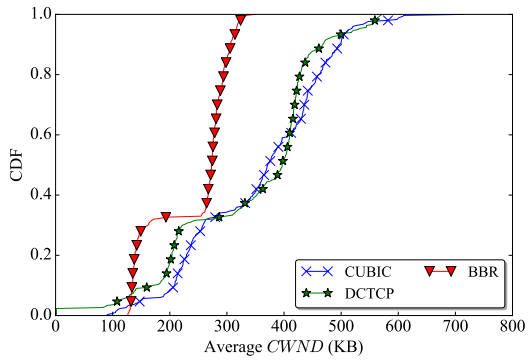

Fig. 8. Variations in average congestion window size during consistent network updates for TCP variants with small buffer size.

three schemes achieve around 3.3 Gbps and 4.8 Gbps, while at the CDF of 100%, CUBIC and BBR achieve about 5.2 Gbps, but DCTCP has a slight edge reaching near 6 Gbps.

3) Variation in Congestion Window: Figs. 7 and 8 characterize the variations in the average congestion window size *cwnd* by the three TCP variants with large and small buffer sizes, respectively. For the large buffer and at the CDFs of 20%, 80% and 100%, the *cwnd* sizes of BBR are 150 KB, 280 KB and 350 KB, and DCTCP has the *cwnd* sizes of 220 KB, 450 KB and 650 KB, while CUBIC has the *cwnd* sizes of 400 KB, 750 KB and 820 KB. For the small buffer and at the CDFs of 20%, 80% and 100%, BBR has the *cwnd* sizes of 140 KB, 280 KB and 330 KB, and DCTCP has the *cwnd* sizes of 200 KB, 430 KB and 550 KB, while CUBIC has the *cwnd* sizes of 230 KB, 450 KB and 700 KB. It is seen that BBR has the smallest *cwnd* and the buffer size has little impact on its congestion window size. This is because BBR controls the congestion window by setting *cwnd* to two times the estimated bandwidth-delay product (BDP) [31]. DCTCP uses an effective multiplicative reduction technique to adjust *cwnd* based on the level of network congestion. Specifically, DCTCP uses the explicit congestion notification (ECN) to estimate the predicted proportion of the marked packets and appropriately modifies its *cwnd* size [23]. CUBIC has the largest cwnd because it has a tendency to fully fill a buffer.

4) Number of Packet Drops: DCTCP and CUBIC have no dropped packets in consistent network updates with a large buffer. Therefore, we perform an experiment with the small buffer size, and the numbers of packet drops experienced by

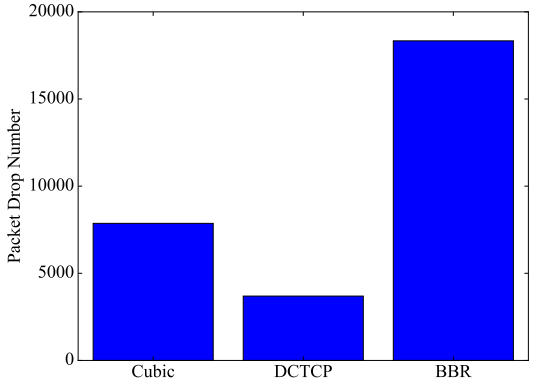

Fig. 9. Average numbers of packet drops during consistent network updates for TCP variants with small buffer size.

the three TCP variants are shown in Fig. 9. The results indicate that BBR has the worst performance and DCTCP has the best performance. More specifically, BBR drops 18336 packets, CUBIC drops 7865 packets, and DCTCP drops merely 3693 packets, respectively.

When packet loss is observed, BBR ignores it as a congestion indication and does not perform back off. Hence, it has no mechanism for congestion detection and reaction to congestion [31]. Due to the fact that CUBIC has an ACK-based congestion control mechanism, it cannot deal with packet drop. DCTCP is also unable to control packet loss if there are severe and short-lived traffic bursts [23]. Therefore, traditional TCP variants are unable to deal efficiently with the packet drop problem in frequent consistent network updates.

C. Analysis for Inconsistent Network Updates

Inconsistencies arise when data plane state changes violate policies or update rules due to varying delays in coordination or no coordination across multiple switches or between controller and switch [28]. Delayed updates in the network may cause inconsistent network updates. To evaluate TCP variants, we use switches with an SDN controller to produce an update delay inconsistency of 100 ms, with a large buffer size.

1) Number of Retransmissions: As can be seen from Fig. 10, BBR exhibits the best retransmission performance, while CUBIC has the worst retransmission performance, which is similar to Fig. 3. Specifically, at the CDFs of 20%, 80% and 100%, BBR requires 100, 200 and 320 retransmissions, and DCTCP requires 200, 450 and 650 retransmissions, while CUBIC imposes 250, 520 and 800 retransmissions. The retransmission performance of BBR and DCTCP are slightly worst than the corresponding performance under consistent network updates given in Fig. 3, but CUBIC has slightly better performance.

2) Throughput: Fig. 11 shows the throughput of the three TCP variants during inconsistent network updates. All the three TCP variants exhibit similar performance, which is different from Fig. 5. In particular, for the CDF above 40%, the throughput of all the three TCP variants are very close.

3) Variation in Congestion Window: Fig. 12 represents the average congestion window size variations in inconsistent network updates with a large buffer size, which are similar to

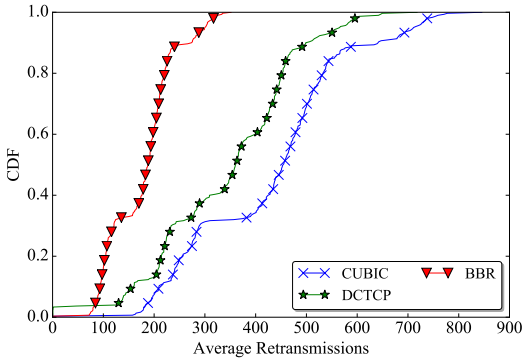

Fig. 10. Numbers of average retransmissions during inconsistent network updates for TCP variants with large buffer size.

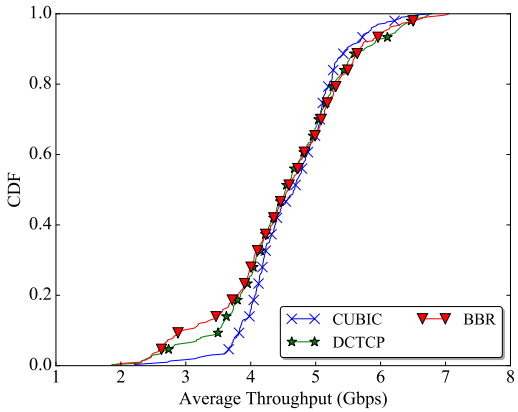

Fig. 11. Average throughput during inconsistent network updates for TCP variants with large buffer size.

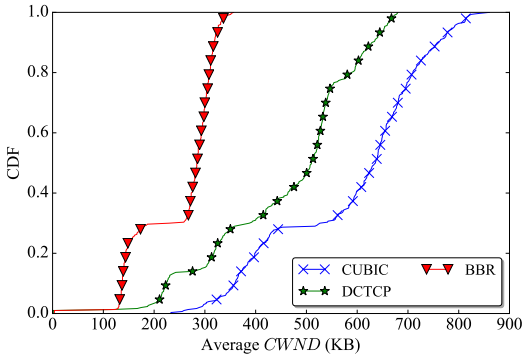

Fig. 12. Variations in average congestion window size during inconsistent network updates for TCP variants with large buffer size.

Fig. 7. As mentioned in Subsection II-B3, BBR controls the congestion window by setting *cwnd* to two times the BDP and it shows the least window size growth, while CUBIC uses an ACK-based congestion control mechanism to maximally grow the window size. DCTCP is ECN-based and it squeezes the window size based on ECN-marked packets.

4) Number of Packet Drops: According to Fig. 13, CUBIC shows the worst performance with 623 packets dropped, and the BBR shows the best performance with 11 packets dropped, while DCTCP drops 187 packets. BBR with a large buffer size has less packet loss and retransmissions because the inflight cap limits the usage of the buffer at around one BDP, which prevents packet loss most of the time [31]. It can easily

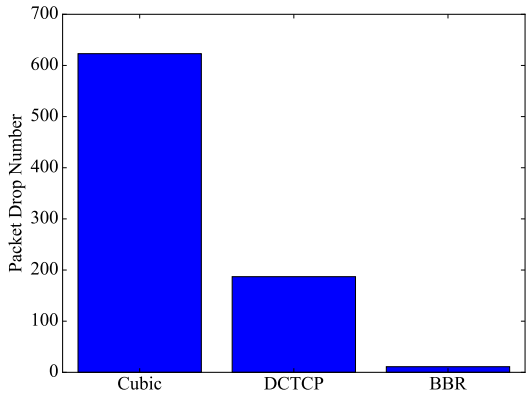

Fig. 13. Average numbers of packet drops during inconsistent network updates for TCP variants with large buffer size.

be inferred that the situation would worsen with a small buffer size during inconsistent network updates. Recalling that DCTCP and CUBIC have no dropped packets in consistent network updates with a large buffer, it can be seen that the problem of packet drops becomes much serious in inconsistent network updates.

D. Conclusions of TCP Variants Analysis

In the scenario of consistent and frequent network updates with large buffer size, DCTCP and CUBIC have no dropped packets but they impose higher average number of retransmissions and larger growth in *cwnd* size than BBR. In the same scenario with small buffer size, BBR has the largest number of dropped packets and DCTCP has the smallest number of dropped packets. This is because the sender in DCTCP estimates the buffer size by maintaining an estimate of fraction of packets that are experiencing congestion. The estimated fraction of packets α, can be updated after one RTT for every window of data as follows.

$$\alpha = (1 - g) \times \alpha + g \times F, \tag{1}$$

where the term $(1 - g) \times \alpha$ represents the decay of previous value of α, while the term $g \times F$ represents the new information from the feedback or increase in α due to the current packet being marked as congested. The gain factor g controls the relative weight of these two terms. Consequently, Eq. (1) is used to update the value of α based on the current value of α and the feedback received from ACKs and negative acknowledgments (NAKs). Thus, upon reception of every packet, the sender receives ECN-marks when the queue length is higher than threshold value K. Moreover, the value of estimation close to 1 indicates high congestion and close to 0 indicates low congestion. While the performance of the three TCP variants are closer in terms of retransmission. It can be observed from the experimental results that BBR has the smallest *cwnd* size. The reason is that BBR controls the *cwnd* based on the bandwidth delay product, BDP, as follows.

$$cwnd = 2 \times BDP, \tag{2}$$

where BDP can be calculated as

$$BDP = B_r \times RTT_{min}. \tag{3}$$

TABLE II
MAIN NOTATIONS

Symbol	Meaning
$\{f_i\}$	The set of flows
$\{s_i\}$	The set of data size
$\{t_i\}$	The set of round-trip times (RTTs)
$\{r_i\}$	The set of flows' throughput
$\widehat{t}$	The RTT without queuing latency
c	The link capacity
l	The queue length
$l_{\max}$	The maximum queue length
τ	The queue latency
w	The window size
$\overline{s}$	The data size acknowledged
α	Estimated marked packets
F	Fraction of marked packets
BDP	Bandwidth delay product
B_r	Smallest data rate
RTT_{min}	Minimum RTT
$cwnd$	Congestion window

Here, B_r is the smallest data rate (bottleneck data rate) and RTT_{min} is the minimal RTT. Conversely, the DCTCP calculates the $cwnd$ as

$$cwnd = cwnd \times (1 - \frac{\alpha}{2}) \tag{4}$$

Hence, when α is close to 0, indicating low congestion, reducing the window slightly. As the DCTCP senders reducing $cwnd$ gently, therefore, its average $cwnd$ is larger than BBR.

The situation is similar in terms of retransmission and *cwnd* size, when updating the network inconsistently with large buffer size, but all the three TCP variants suffer from the problem of dropped packets. This indicates that the problem of out-of-order packets is more serious when the network is updated inconsistently and frequently. It may also be reasonably inferred from the results that the performance of TCP variants will deteriorate in the case of small buffer size.

III. NUFTCP DESIGN

The previous simulation experiments have revealed that the state-of-the-art TCP variants are ineffective, particularly when the network is updated inconsistently and frequently. This motivates us to design a better solution that is capable of coping with network updates smoothly. Specifically, we develop the NUFTCP design, which aims to mitigate the problem of dropped packets during network updation by restricting the window size, managing the queue and handling out-of-order packets in SD-DCN. More specifically, two modifications are introduced. The first one restricts the size of the transmit side window to mitigate dropped packets in inconsistent network environments. This is because the transmission window on the sending side decides the amount of data that can be sent before the receiver sends an acknowledgment, ensuring that it does not exceed the receiver's buffer. The second one diminishes out-of-order packets by delaying duplicated ACKs on the recipient side. The main symbols utilized in the design are listed in Table II.

A. Limit the Window Size and Queue Management

1) Why NUFTCP cares about Packet Drop not Congestion Avoidance: NUFTCP does not focus on ordinary congestion

avoidance; instead, it avoids severe congestion, which leads to packet drop. A TCP protocol keeps adding packets to the queue until a packet is dropped. Then TCP extends queues to accommodate transient traffic bursts, and hence the average queue length is quite long. Therefore, the rate of dropped packets is incredibly low until a queue is full, thereafter it is possible that all the incoming packets from all flows are lost. Consequently, packet drop slows down all the TCP flows. In the worst-case scenario, a packet drop can result in significant data corruption or possibly the loss of a link completely. For this reason, we are more apprehensive about packet drops. NUFTCP utilizes the inherent negative feedback to prevent the problem of packet drops triggered by severe congestion.

2) Relationship among Data Size, Flow Throughput and RTT: Since TCP is not aware of the congestion of the underlying network, it relies on a time limit, which shows the duration a sender waits before retransmitting a lost segment to estimate whether the forwarded segment is dropped or not. Specifically, if an acknowledgment is not received by the sender side within the defined time limit, it is considered a segment drop. In the context of standard TCP, the time limit is referred to as the retransmission timeout (RTO). The RTO is not a fixed value, but rather it is dynamically adjusted based on the estimated round-trip time (RTT). The RTO is maximum amount of time that TCP will wait for an ACK for a segment before retransmitting it. The RTO is typically set to a few times the estimated RTT, which is the time it takes for a segment to travel from the sender to the receiver and back. It is helpful to account for factors such as network delays and processing times. Further, if TCP retransmits a segment too many times without receiving an ACK, it assumes that the connection is congested and will slow down its sending rate. This is known as congestion control. The reason TCP relies on time limits or timeouts to estimate congestion is because it does not have direct access to network information such as buffer occupancy and queue lengths. Therefore, the RTO is one of the common ways to determine the packet drop. Accordingly, TCP keeps track of RTT for each segment. A TCP connection receives an estimated maximum data size that the receiver side can accommodate. Let RTT be t and flow throughput be r. Then data size s transmitted in t is given by $s = r \times t$. In other words, the flow throughput is proportional to the data size and inversely proportional to the RTT. When a link begins to exhibit congestion, the queue length for the link is increased, which induces the rise in the queuing latency of the packets going through the link. This implies that the RTT of the corresponding flow will be longer and the throughput will be lower in order to alleviate the congestion. Therefore, when we transmit the data size in one RTT, the throughput is controlled by negative feedback. There will be no packet drop as long as the buffer size of the queue is sufficiently large.

3) Queue Management: We will use the example of Fig. 14 to illustrate queue management. Prior to the update, assume that we have $s_1 = r_1 \times t_1$ and $s_2 = r_2 \times t_2$, where s_1, r_1 and t_1 are related to flow f_n, while s_2, r_2 and t_2 are related to flow f_m. In this case, $r_1 = r_2 = c$ and $t_1 = t_2 = \widehat{t}$.

Assume that when the network first commences updating, it offers an inconsistent network state. Subsequently, the network

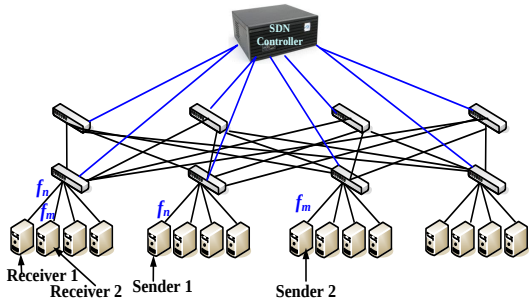

Fig. 14. Topology of a software defined data center network.

state of f_n is updated and its route is altered, while the network state of f_m remains unchanged. Specifically, flow f_n is routed to the intermediate node that f_m is currently using. Both the flows now utilize the same link, and the queue is overburdened, resulting in packet loss. For the sake of simplicity, suppose that the queue length (in bytes) in the stable state is l. Dividing the queue length by the link capacity yields the queuing latency $\tau = l/c$. After the inconsistent network update, $s_1 = r_1' \times t_1'$ and $s_2 = r_2' \times t_2'$ with $t_1' = t_2' = \widehat{t} + \tau$ and $r_1' + r_2' = c$. Hence, we have $l = \sum_i s_i - c \times \widehat{t}$. Because $\sum_i s_i \leq 2 \times c$, we have $l \leq c \times \widehat{t}$. Thus, the maximal queue length should be $l_{\max} < l + \sum_i s_i$ (if the queue length in the previous RTT is more than l, the queue length will not be increased.). Therefore, we can draw the conclusion:

$$l_{\max} < 3 \times c \times \widehat{t} \tag{5}$$

For example, if $c = 10\,\text{Gbps}$ and $\widehat{t} = 250\,\mu\text{s}$, $l < 937.5\,\text{KB}$. It implies that if the buffer size is 1 MB, no packets should drop. Eq. (5) ensures that packets are not dropped and network stability is maintained throughout network updates. The conclusion can be extended to more general cases. Algorithm 1 describes the queue management process of NUFTCP.

Algorithm 1 NUFTCP Queue Management Algorithm

1: **Input:** l (queue length in bytes), c (link capacity), $\hat{t}$ (time between RTTs), S_i (Data size for flow f_i), t_i (Transmission time for flow f_i), τ (Queuing latency)
2: **Output:** l_{max} (maximum queue length)
3: $l_{max} \leftarrow 0$ ▷ Initializing variable
4: $l \leftarrow \sum_i s_i - c \times \hat{t}$ ▷ Update queue length
5: $l_{max} \leftarrow 3 \times c \times \tau$ ▷ Calculating maximum queue length
6: **if** $l_{max} > l$ **then**
7: Set $l \leftarrow l_{max}$ ▷ Updating queue length
8: Drop packets until queue length is reduced to l_{max}
9: **end if**
10: $s_i \leftarrow s_i - (\frac{\tau \times s_i}{l_{max}})$ ▷ Calculating data size for each flow (si):
11: Transmit data for each flow (s_i)
12: Repeat steps 5-11 until queue is empty

4) How NUFTCP limits Window Size: Benefiting from negative feedback, NUFTCP aims to limit the window size w when a congestion occurs. As aforementioned, congestion avoidance protocol grows the window size constantly and linearly, which may cause large number of packet drops in

TABLE III
PERFORMANCE COMPARISON OF DIFFERENT TCP VARIANTS

Feature	BBR [27]	CUBIC [25]	DCTCP [23]	The Proposed NUFTCP
Objective	High bandwidth utilization	Fairness and high throughput	Data center congestion control	Addresses packet drop problem during frequent consistent network updates in SD-DCN
Congestion Control Type	Delay-based	Window-based	ECN-based	ECN-based on inherent negative feedback
Congestion Signal	RTT	Packet loss	ECN markings	Inherent negative feedback
Response to Congestion	Proactive, aims to prevent congestion	Reactive, reduces window size upon congestion	Proactive, utilizes ECN	Proactive, utilizes inherent negative feedback
Awareness of Frequent Network Updates	Low	Low	Moderate	High
Throughput Behavior	Aims for high throughput	Tends to oscillate	Balances throughput and low latency	High throughput
Throughput Optimization	Yes	Yes	Yes	Yes
Buffer Utilization	Efficient use of buffers	May lead to buffer bloat	Designed to avoid buffer bloat	Designed to avoid buffer bloat
Buffer Bloat Mitigation	Yes	No	Yes	Yes
Scalability	Yes	Yes	Yes	Yes
Use Case	Broadband connections	General-purpose	Data center environments	SD-DCN environments

severe congestion situation. NUFTCP limits the maximum window size to avoid this issue. When there is a congestion, the window size w is nearly equal to one TCP flow, i.e., $w \approx s$. If the flow does not face congestion, w may be significantly greater than s. TCP uses an exponentially weighted moving average (EWMA) to keep track of the fluctuations of RTT over time, and it places approximately 20 percent of the weight on the most recent RTT measurement. Therefore, NUFTCP keeps track of the average actual data size acknowledged $\overline{s}$ in the past $\widehat{t}$ using EWMA. The upper window size limit is then adjusted to $1.5 \times \overline{s}$ in the following $\widehat{t}$, where the factor 1.5 is employed because the window size must be able to grow gently.

Since NUFTCP is based on DCTCP, packets will not be discarded due to random early detection (RED), which is founded on statistical probabilities and is more balanced than the tail drop. However, DCTCP will decrease the slow start threshold when the transmit side receives packets with an explicit congestion expected (ECE) flag, which is part of an explicit congestion notification (ECN) protocol. If NUFTCP does not receive a packet with the ECE flag in 10 consecutive RTTs, it will reset the slow start threshold. This is because to prevent the excessive window size limit, avoiding congestion, and proactively addressing potential packet loss in the network. Algorithm 2 presents the congestion control and window size adjustment of NUFTCP.

Algorithm 2 Congestion Control and Window Size Adjustment in NUFTCP

Variables: s (Average actual data size acknowledged in the past using EWMA), ECE_flag (Explicit congestion expected flag)
Output: w (Window size)
$w \leftarrow s$ ▷ Initialize congestion window w
$SST \leftarrow s$ ▷ Initialize slow start threshold
$ECE\_flag\_counter = 0$ ▷ Initialize ECE flag counter
while true **do**
  if congestion detected **then**
    $w \leftarrow s$ ▷ Update window size
  end if
  if congestion is not detected **then**
    if $w < SST$ **then**
      $w \leftarrow min(w + 1, SST)$ ▷ Gradually increase window size
    else
      $w \leftarrow min(w + 1.5s, w_{max})$ ▷ Update w using weight $= 1.5$
    end if
  end if
  if an ECE_flag is received **then**
    $SST \leftarrow s$ ▷ Update slow start threshold
    $ECE\_flag\_counter \leftarrow 0$ ▷ Reset $ECE\_flag\_counter$
  else
    $ECE\_flag\_counter++$ ▷ Increment ECE_flag_conter
  end if
  if $ECE\_flag\_counter == 10$ **then**
    $SST \leftarrow s$ ▷ Reset slow start threshold
    $ECE\_flag\_counter \leftarrow 0$ ▷ Reset $ECE\_flag\_counter$
  end if
  $w \leftarrow min(w, SST)$ ▷ Update window size
  Transmit data using window size w
end while
Repeat steps 6-29 until congestion is resolved or slow start threshold is reached

B. Delay Duplicated ACKs

NUFTCP can rectify the out-of-order issue by delaying the delivery of duplicate ACKs during the network update. The duplicated ACK will only be returned if it is postponed for period T and the accompanying data packet has not yet been delivered. A duplicated ACK is rejected in all the other cases. Here, T is a preconfigured time threshold that equals to the maximal RTT difference on different paths, typically smaller than 1 or 2 ms. Since the window size limit avoids the packet drop, the impact of delaying the fast recovery is tolerable.

To be able to correctly delay duplicate ACKs, NUFTCP needs to first identify network update. The cooperation of switches, like ECN, is necessary for NUFTCP. A version number is assigned to each flow table entry in the switches. Each time the controller recalculates a flow table entry, the version number will be raised by one. Each packet that matches the entry will be marked with the current version number of the entry (if the version number is greater than the one carried in the packet). The first 4 bits in the TTL field are used for version number tagging because it is generally futile for intra-data center traffic whose maximal hops are smaller than 15. By tracking TTL modifications, NUFTCP has the capacity to explicitly identify network updates and defer repeated ACKs. Table III presents the performance comparison of BBR [27], CUBIC [25], DCTCP [23], and the proposed NUFTCP. Algorithm 3 describes the duplicate ACK delay

mechanism of NUFTCP.

Algorithm 3 NUFTCP Duplicate ACK Delay Mechanism

Input: Ack (Acknowledgment packet), T (Preconfigured time threshold for delaying duplicate ACKs), $packet\_received$ (Flag indicating whether the corresponding data packet has been received), $current\_version$ (Current version number of the flow table entry), $packet\_version$ (Version number carried in the ACK packet), t (time since ACK reception)
Output: $processed\_ACK$ (Boolean flag indicating whether the ACK was processed or not)
while $processed\_ACK == False$ **do**
  if $current\_version > packet\_version$ **then**
    ACK indicates a network update
  end if
  if If an ACK is duplicate **then**
    if If $t > T$ and $packet\_received$ == False **then**
      $processed\_ACK \leftarrow True$
      $packet\_received \leftarrow True$
    else
      $processed\_ACK \leftarrow False$ ▷ Discard the ACK
    end if
  end if
  if ACK is not a duplicate **then**
    $processed\_ACK \leftarrow True$
  end if
end while
Return $processed\_ACK$ flag

IV. Evaluation

The currently available network updating methods [2], [3], [7], [9]–[11], [20], [29] take into account the violations of consistency properties and network update frequency. Hence the comparison between NUFTCP and these methods is unsuitable, as NUFTCP is concerned with how dropped packets and out-of-order packets issues of network updates influence TCP performance. For this reason, we compare NUFTCP with the state-of-the-art TCP variant DCTCP, which has been shown to outperform other TCP variants as demonstrated in the experiments of Section II.

A. Simulation Network Topology and Parameters

We assess NUFTCP using NS3 based simulations under two different conditions of large and small buffer sizes. We use SD-DCN topology to compare NUFTCP and DCTCP. As illustrated in Fig. 14, we utilize an advanced conventional 2-tier Clos network topology in DCN attached through an SDN controller. The network makes use of equal-cost multipath routing (ECMP). Each link has a 10 Gbps capacity with $\widehat{t} \approx 500\,\mu$s RTT without queuing latency. The large and small buffers have 1 MB and 0.5 MB in size, respectively. The controller transmits the updated information towards the edge switches every 25 ms, assuming one of the aggregation layer switches is down (or upgrading). Flows passing precisely via the down (or upgrading) switches must modify their transmitting pathways backward and forward in this format. NUFTCP strategy utilizes four bits of the packet header's TTL field to indicate the version number, and hence in our simulations, the utmost multitude of distinct versions is sixteen.

Our network architecture states that f_n flows are transmitted from *Sender 1* to *Receiver 1* and f_m flows are transmitted from *Sender 2* to *Receiver 2* simultaneously. Because there is inconsistency across switches during the update, we simulate update delay of $0 \sim 1$ ms in the switches. The variation in update time is produced at random, similar to [24].

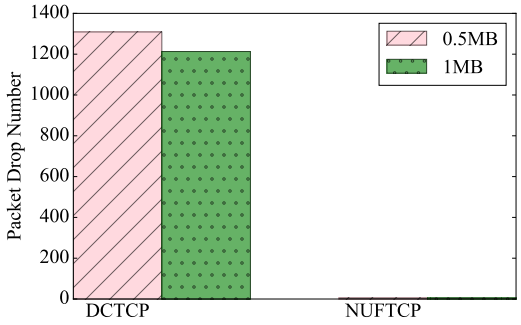

Fig. 15. Comparison of packet drops for DCTCP and NUFTCP during network updates with large and small buffer sizes.

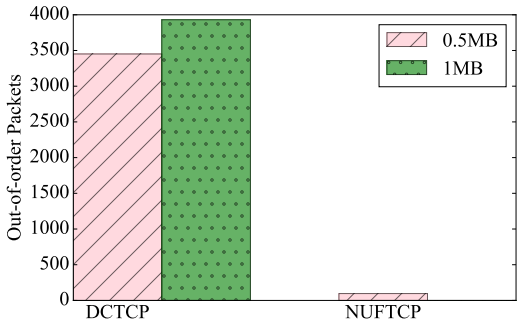

Fig. 16. Comparison of out-of-order packets for DCTCP and NUFTCP during network updates with large and small buffer sizes.

B. Results and Discussion

1) Packet Drops and Out-of-Order Packets: A randomized delay is used to preserve inconsistency throughout the network updates. In the scenario of large buffer size, we execute several simulations and find that the numbers of packet drops caused by DCTCP are 1188, 1294, 1159, and so on. The average number of dropped packets is 1213, which is displayed in Fig. 15. By contrast, NUFTCP never drops a packet as can be seen from Fig. 15. As shown in Fig. 16, the typical number of out-of-order packets for DCTCP is 3930, which is very high. NUFTCP on the other hand performs much better than DCTCP with no out-of-order packets.

In the scenario of small buffer size, after several simulations, the numbers of packet drops caused by DCTCP are found to be 1381, 1366, 1181, and etc. The average number of packet drops by DCTCP is 1309 as depicted in Fig. 15. Because a small buffer may lead to early packet drops, DCTCP exhibits a higher number of packet drops than in the large buffer scenario. Again NUFTCP does not suffer from packet loss. In Fig. 16, DCTCP exhibits a high number of out-of-order packets (3451) but is less than in the large buffer case. It is rare for NUFTCP to suffer from the problem of out-of-order packets and in this case, it only has 96 out-of-order packets.

2) Throughput: For the case of large buffer size, Figs. 17 and 18 depict the throughput achieved by DCTCP and NUFTCP, respectively, during the network update. Observe from Fig. 17 that for DCTCP, because of packet drops in the beginning, both its flows' throughput are reduced to almost zero. Then after some time, one of its flow throughput is reduced to almost zero owning to packet drops occurring again. Finally, there is a prolonged converging time of its two

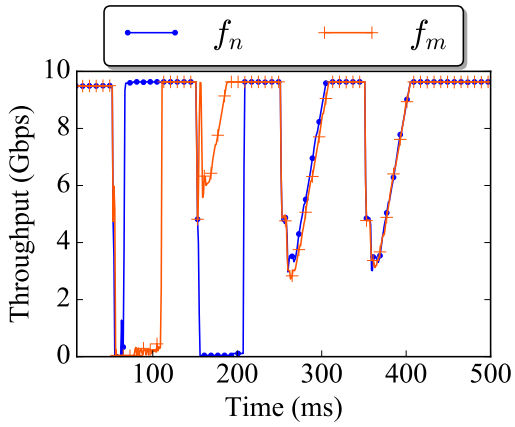

Fig. 17. DCTCP throughput during network updates with large buffer size.

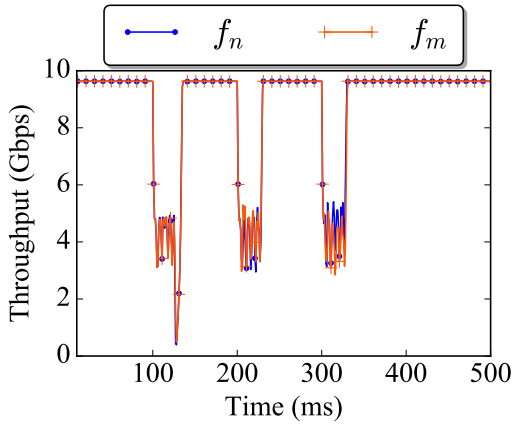

Fig. 18. NUFTCP throughput during network updates with large buffer size.

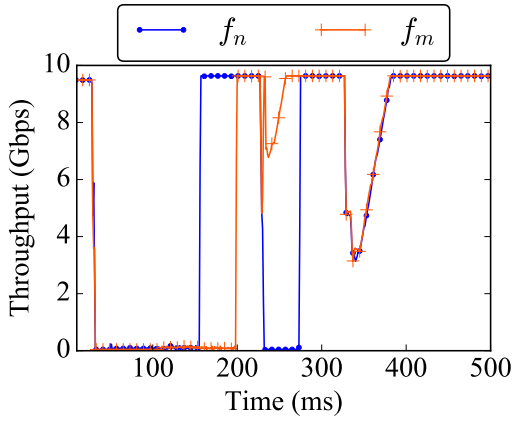

Fig. 19. DCTCP throughput during network updates with small buffer size.

flows' throughput. As can be seen from Fig.18, NUFTCP by contrast offers more consistent throughput since packets are not dropped and out-of-order problem is dealt with effectively.

In the scenario of small buffer size, the throughput achieved by DCTCP and NUFTCP during the network update are depicted in Figs. 19 and 20, respectively. It can be seen from Fig. 19 that DCTCP performs even worst than in the large buffer case, and there exists a long period of almost zero throughput for its two flows. NUFTCP on the other hand exhibits more consistent and stable throughput throughout the whole update period, just as in the case of large buffer. This

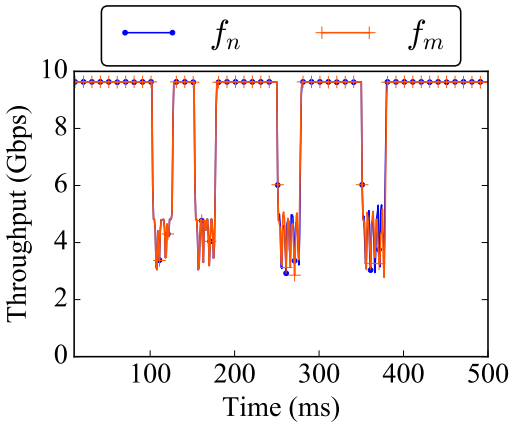

Fig. 20. NUFTCP throughput during network updates with small buffer size.

clearly demonstrates that NUFTCP can effectively maintain a stable network throughput during network update.

3) Discussion: The aforementioned experimental results again confirm that the state-of-the-art TCP variants, such as DCTCP, suffer from the serious problem of packet drops and out-of-order packets during frequent and inconsistent network updates, which significant degrades their achievable throughput, particularly during the early period of update. The simulation results also validate that our proposed NUFTCP achieves its design goals. Specifically, NUFTCP can effectively deal with the issues of packet drops and out-of-order packets, and consequently it achieves better and more consistent throughput during inconsistent and frequent network updates. Based on this evaluation, we may draw the conclusion that our NUFTCP design offers an effective means for handling inconsistent and frequent network updates in SD-DCN. However, NUFTCP is unable to completely mitigate the issue of TCP Incast congestion. Incast congestion refers to a decrease in throughput when multiple senders simultaneously communicate with a single receiver, thereby exceeding the receiver's buffer capacity. Hence, our future work will seek to address this issue.

V. Related Work

A. TCP designs

All TCP designs offer some kinds of congestion algorithms to avoid and resolve congestion problems and to attempt to mitigating the issues of out-of-order packets and packet drops. The relevant contemporary TCP implementations are reviewed in this subsection.

Several delay based-TCP variants [33]–[35], have been proposed, which rely on packet delay measurements as a signal of congestion [36]. These schemes aim to reduce queue lengths and congestion at routers. However, queuing delays in data centers, are comparable to sources of noise in the system, thus, unable to provide a reliable congestion signal. Moreover, delay signals are not accurate enough to compute appropriate congestion window to reduce congestion at routers [37]. TCP Reno [38], [39] presents a fast recovery mechanism using available bandwidth designated by the arrival of DUPACK but it is intolerant to connections with long delays. TCP NewReno [40], [41] is a loss-based congestion algorithm that

is an extension of TCP Reno with a modified fast recovery algorithm. CUBIC [25] introduces a cubic function of elapsed time for window growth when the loss occurs, and hence it improves the friendliness of binary increase congestion control (BIC). Data Center TCP (DCTCP) [23] alters ECN, so that switches mark packets in accordance with the current queue length and senders modify the size of their send window according to the estimated fraction of marked packets. Linux TCP now has a novel congestion control technique called bottleneck bandwidth and RTT (BBR) [27]. BBR finds a better operating point that takes care of bandwidth and RTT by estimating the round-trip propagation delay and bandwidth, and it sets *cwnd* to a small multiple of the estimated BDP. TCP-PR [42] and TCP-RR [43] were developed for persistent out-of-order packets but they are not suitable for the DCN update scenario. TCP-RR relies on DSACK, which is not supported by all servers, and TCP-PR needs to maintain tables in memory which imposes high computation costs.

The aforementioned TCP variants are ineffective to deal with the issues of out-of-order packets and packet drops occurred in inconsistent and frequent updates of SD-DCNs. By contrast, our proposed NUFTCP design is capable of dealing with the problems of out-of-order packets and packet drops effectively and, consequently, ameliorates TCP performance when networks are updated inconsistently and frequently.

B. Network Updates

The literature of network updates provides state-of-the-art solutions for mitigating the problems of forwarding loops, forwarding blackhole, link congestion, and policy violation, which cause inconsistent network updates. Since inconsistent network updates may lead to the issues of out-of-order packets and packet drops, these solutions aim to maintain consistency.

A single switch can handle the difficulties of forwarding loop and forwarding blackhole, as mentioned in a method by Reitblatt *et al.* [20]. To distinguish the old and new packets, they are stamped with version numbers to implement old and new rules. zUpdate [2] provides a solution to the problems of congestion, forwarding loop, and forwarding blackhole in the data center, and it uses ECMP to split the traffic equally using multiple redundant paths. Hedera [3] handles frequent network updates in data centers by allocating the paths for large flows based on the estimated demand using an annealing based algorithm. SWAN [7] achieves high network capacity utilization of inter-DC links in SDN in the presence of traffic volume variations, and it leaves a small amount of scratch capacity on the link that can be used for updating. TRUS [12] provided a timely route updating technique that reduces network congestion while meeting the bandwidth needs of delay-sensitive traffic. Dionysus [11] supports fast and consistent network updates in SDN using dynamic scheduling during updates at switches individually. It can be applied to both SDN WAN and DCN environments. FLIP [10] proposes an algorithm that ensures forwarding correctness and forwarding policies using a fast and lightweight algorithm. Cupid [29] emphasizes on consistent flow tables and data plane updating to maintain the throughput of flows, and it outperforms Dionysus. ez-Segway [9] presents a decentralized consistent update mechanism, which completes network updates quickly by utilizing sophisticated coordinating actions in the switches.

The authors of [21] proposed suffix causal consistency (SCC) motivated by a consistency model for shared-memory systems for rule updates. The method ensures consistency properties to avoid blackhole loops, bounded loops etc. The approach of [22] is based on on temporal logic and model checking for data flow correctness verification and concurrent updates using Petri nets to make sure the absence of loops. The authors in [15] devised algorithms to mitigate transient congestion, reduce update time, and minimize control overhead. Their algorithms optimize the intermediate stages after finding the optimal route at each middle stage to minimize the temporary congestion efficiently. The authors of [16] proposed customizable update planner (CUP) which adopts the existing designs to achieve the congestion avoidance and optimize the update speed. CUP introduces generic linear programming models to schedule network updates to user-specified needs, and it offers a solution to the transient congestion problem. Hermes [17] provided a utility-aware network update system that maximizes the total utility by a rate-limiting scheme before the update. It ensures congestion-free property during network updates. The authors in [18], [19] used resource dependency graph to formulate network update problems, approximation algorithms to utilize bandwidth resources, spare-path-assisted algorithms for consistent flow migration, and rate-limiting-flow to resolve deadlocks. Their method ensures fast network updates with consistency properties. In [44], the authors emphasized that the real-time communication should remain invariant by diverting the traffic to uninvolved devices during network updates. The authors of [14] introduced a framework based on abstract algebra that enables controllers to combine the fast composition of numerous network updates with persistent and non-blocking modifications in the network by efficiently modeling the data plane operations.

Most of the aforementioned network updating methods focus on avoiding the violations of consistency properties in network updates. By contrast, our NUFTCP design is developed to alleviate the impact of inconsistent and frequent network updates on TCP performance so that network updates can happen smoothly.

VI. Conclusions

The contribution of this paper has been twofold. Firstly, we have conducted comprehensive experiments to evaluate the performance of the state-of-the-art TCP variants in the presence of frequent and inconsistent network updates in SD-DCNs. Our findings have confirmed that current TCP variants are incapable of handling frequent and inconsistent network updates, and they suffer from the problems of out-of-order packets and packet drops, which leads to significant performance degradation in terms of network throughput. Secondly, we have proposed a network update friendly TCP modification, called NUFTCP, which is an extension to DCTCP. Our NUFTCP design can tackle the issues of packet drops and out-of-order packets throughout frequent and inconsistent

network updates in SD-DCNs, which have not been resolved by the previous works. Our evaluation results have validated that NUFTCP performs substantially better than the state-of-the-art DCTCP, when the network is updated frequently and inconsistently. Our NUFTCP therefore offers a useful design to smoothly handle network updates in SD-DCNs.

REFERENCES

[1] D. Li, S. Wang, K. Zhu, and S. Xia, "A survey of network update in SDN," *Frontiers of Computer Science*, vol. 11, no. 1, pp. 4–12, 2017.

[2] H. H. Liu, *et al.*, "zUpdate: Updating data center networks with zero loss," in *Proc. SIGCOMM 2013* (Hong Kong, China), Aug. 12-16, 2013, pp. 411–422.

[3] M. Al-Fares, *et al.*, "Hedera: Dynamic flow scheduling for data center networks," in *Proc. NSDI 2010* (San Jose, CA, USA), Apr. 28-30, 2010, pp. 1–15.

[4] T. Benson, A. Anand, A. Akella, and M. Zhang, "MicroTE: Fine grained traffic engineering for data centers," in *Proc. CoNEXT 2011* (Tokyo, Japan), Dec. 6-9, 2011, pp. 1–12.

[5] K.-T. Foerster, S. Schmid, and S. Vissicchio, "Survey of consistent software-defined network updates," *IEEE Communi. Surveys & Tutorials*, vol. 21, no. 2, pp. 1435–1461, Secondquarter 2019.

[6] U. Haider, M. Waqas, M. Hanif, H. Alasmary and S. M. Qaisar, "Network load prediction and anomaly detection using ensemble learning in 5G cellular networks," in *Computer Communications*, Elsevier, vol. 197, pp. 141-150, Jan. 2023.

[7] C.-Y. Hong, *et al.*, "Achieving high utilization with software-driven WAN," in *Proc. SIGCOMM 2013* (Hong Kong, China), Aug. 12-16, 2013, pp. 15–26.

[8] S. Jain, *et al.*, "B4: Experience with a globally-deployed software defined WAN," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 3–14, Oct. 2013.

[9] T. D. Nguyen, M. Chiesa, and M. Canini, "Decentralized consistent updates in SDN," in *Proc. SOSR 2017* (Santa Clara, CA, USA), Apr. 3-4, 2017, pp. 21–33.

[10] S. Vissicchio and L. Cittadini, "FLIP the (flow) table: Fast lightweight policy-preserving SDN updates," in *Proc. INFOCOM 2016* (San Francisco, CA, USA), Apr. 10-14, 2016, pp. 1–9.

[11] X. Jin, *et al.*, "Dynamic scheduling of network updates," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4, pp. 539–550, Oct. 2014.

[12] J. Zhu, *et al.*, "TRUS: Towards the real-time route update scheduling in SDN for data centers," *IEEE Access*, vol. 8, pp. 68682–68694, 2020.

[13] J. Zhang, B. Gong, M. Waqas, S. Tu and Z. Han, "A Hybrid Many-Objective Optimization Algorithm for Task Offloading and Resource Allocation in Multi-Server Mobile Edge Computing Networks," in *IEEE Transactions on Services Computing*, vol. 16, no. 5, pp. 3101-3114, Sept.-Oct. 2023.

[14] G. Li, *et al.*, "Update algebra: Toward continuous, non-blocking composition of network updates in SDN," in *Proc. INFOCOM 2019* (Paris, France), Apr. 29-May 2, 2019, pp. 1081–1089.

[15] J. Zheng, *et al.*, "Congestion-minimizing network update in data centers," *IEEE Trans. Services Computing*, vol. 12, no. 5, pp. 800–812, Sep.-Oct. 2019.

[16] S. Luo, H. Yu, L. Luo, and L. Li, "Customizable network update planning in SDN," *J. Network and Computer Applications*, vol. 141, pp. 104–115, Sep. 2019.

[17] J. Zheng, *et al.*, "Hermes: Utility-aware network update in software-defined WAN," in *Proc. ICNP 2018* (Cambridge, UK), Sep. 25-27, 2018, pp. 231–240.

[18] Y. Chen, H. Zheng, and J. Wu, "Consistency, feasibility, and optimality of network update in SDNs," *IEEE Trans. Network Science and Engineering*, vol. 6, no. 4, pp. 824–835, Oct.-Dec. 2018.

[19] M. Waqas, M. Zeng, Y. Li, D. Jin and Z. Han, "Mobility Assisted Content Transmission For Device-to-Device Communication Underlaying Cellular Networks," in *IEEE Transactions on Vehicular Technology*, vol. 67, no. 7, pp. 6410-6423, July 2018, doi: 10.1109/TVT.2018.2802448.

[20] M. Reitblatt, *et al.*, "Abstractions for network update," in *Proc. SIGCOMM 2012* (Helsinki, Finland), Aug. 13-17, 2012, pp. 323–334.

[21] S. Liu, T. A. Benson, and M. K. Reiter, "Efficient and safe network updates with suffix causal consistency," in *Proc. EuroSys 2019* (Dresden, Germany), Mar. 25-28, 2019, pp. 1–15.

[22] B. Finkbeiner, M. Gieseking, J. Hecking-Harbusch, and E.-R. Olderog, "Model checking data flows in concurrent network updates," in *Proc. ATVA 2019* (Taipei, Taiwan, China), Oct. 28-31, 2019, pp. 515–533.

[23] M. Alizadeh, *et al.*, "Data center TCP (DCTCP)," *ACM SIGCOMM Computer Communi. Review*, vol. 40, no. 4, pp. 63–74, Oct. 2010.

[24] A. Basta, *et al.*, "Efficient loop-free rerouting of multiple SDN flows," *IEEE/ACM Trans. Networking*, vol. 26, no. 2, pp. 948–961, Apr. 2018.

[25] S. Ha, I. Rhee, and L. Xu, "CUBIC: A new TCP-friendly high-speed TCP variant," *ACM SIGOPS Operating Systems Review*, vol. 42, no. 5, pp. 64–74, Jul. 2008.

[26] A. B. Dogar and Y. Zhang, "NUFTCP: Towards Smooth Network Updates in Software-Defined Datacenter Networks," in *17th International Conference on Network and Service Management (CNSM)*, Izmir, Turkey, 2021, pp. 365-369, doi: 10.23919/CNSM52442.2021.9615582.

[27] N. Cardwell, *et al.*, "BBR: Congestion-based congestion control," *Communications of the ACM*, vol. 60, no. 2, pp. 58–66, Feb. 2017.

[28] R. Mahajan and R. Wattenhofer, "On consistent updates in software defined networks," in *Proc. HotNets-XII* (College Park, MD, USA), Nov. 21-22, 2013, pp. 1–7.

[29] W. Wang, W. He, J. Su, and Y. Chen, "Cupid: Congestion-free consistent data plane update in software defined networks," in *Proc. INFOCOM 2016* (San Francisco, CA, USA), Apr. 10-14, 2016, pp. 1–9.

[30] H. Xu, *et al.*, "Real-time update with joint optimization of route selection and update scheduling for SDNs," in *Proc. ICNP 2016* (Singapore), Nov. 8-11, 2016, pp. 1–10.

[31] M. Hock, R. Bless, and M. Zitterbart, "Experimental evaluation of BBR congestion control," in *Proc ICNP 2017* (Toronto, ON, Canada), Oct. 10-13, 2017, pp. 1–10.

[32] N. Cardwell, *et al.*, "BBR: Congestion-based congestion control," *Queue*, vol. 14, no. 5, pp. 58–66, Feb. 2017.

[33] Verma, L.P. Sharma, V. K. Kumar, M. Kanellopoulos, and Dimitris,"A novel delay-based adaptive congestion control TCP variant" *Computers and Electrical Engineering*, vol. 101, pp. 108076, 2022.

[34] Chiang, C. Chan, Y. Chen and Ping-Lien,"Delay-based TCP with Pacing and ECN for solving incast problem in data center networks" in *IET International Conference on Engineering Technologies and Applications (IET-ICETA)*, 2022.

[35] Kim, Geon-Hwan,Cho and You-Ze,"Delay-aware BBR congestion control algorithm for RTT fairness improvement" *IEEE Access*, vol. 8, pp. 4099–4109, 2022.

[36] Agarwal, N. Varvello, M. Aucinas, A. Bustamante, F. Netravali and Ravi,"Mind the delay: the adverse effects of delay-based TCP on HTTP" in *Proceedings of the 16th International Conference on emerging Networking EXperiments and Technologies*, pp. 364–370, 2020.

[37] Ma, Huihui, Xu and Du,"INT-based TCP window modulator for congestion control in data center networks" *Journal of Network and Computer Applications*, vol. 8, pp. 103688, 2023.

[38] M. Allman, V. Paxson, and E. Blanton, "TCP congestion control," RFC 5681, 2009. https://doi.org/10.17487/rfc5681

[39] J. Mo, R. J. La, V. Anantharam, and J. Walrand, "Analysis and comparison of TCP Reno and Vegas," in *Proc. INFOCOM 1999* (New York, NY, USA), Mar. 21–25, 1999, pp. 1556–1563.

[40] S. Floyd, A. Gurtov, and T. Henderson, "The NewReno modification to TCP's fast recovery algorithm," RFC 3782, Apr. 2004.

[41] T. Henderson, *et al.*, "The NewReno modification to TCP's fast recovery algorithm," RFC 6582, Apr. 2012.

[42] S. Bohacek, *et al.*, "TCP-PR: TCP for persistent packet reordering," in *Proc. 3rd Int. Conf. Distributed Computing Systems* (Providence, RI, USA), May 19-22, 2003, pp. 222–231.

[43] M. Zhang, B. Karp, S. Floyd, and L. Peterson, "RR-TCP: A reordering-robust TCP with DSACK," in *Proc. 11th IEEE Int. Conf. Network Protocols* (Atlanta, GA, USA), Nov. 4-7, 2003, pp. 95–106.

[44] S. U. N. Prottoy, D. Saucez, and W. Dabbous, "NUTS: Network updates in real time systems," in *Proc. SOSR 2019* (San Jose, CA, USA), Apr. 3-4, 2019, pp. 160–161.