NUFTCP: Towards Smooth Network Updates in Software-Defined Datacenter Networks

Abdul Basit Dogar[⋆§], Yiran Zhang[†§]
[⋆]Department of Computer Science and Technology, Tsinghua University
[†]Institute for Network Sciences and Cyberspace, Tsinghua University
[§]Beijing National Research Center for Information Science and Technology (BNRist)
Email:{bas15,zyr17}@mails.tsinghua.edu.cn

***Abstract*—Since the widespread deployment of the Software Defined Network in the datacenter, network updates become more and more common. However, modern TCP designs are not resilient to network updates, which introduce flow reroutes. Frequent and inconsistent network updates lead to severe performance degradation such as packet drop (caused by transient congestion) and out-of-order packets problems. In this paper, we verify and show the poor performance of common TCP implementations when there are frequent network updates. We analyze the reasons and propose a simple variant of DCTCP as a countermeasure, named NUFTCP (Network Update Friendly TCP). We evaluate NUFTCP with simulations. The results demonstrate that NUFTCP can more efficiently handle the problems of packet drop and out-of-order packets caused by network updates.**

***Index Terms*—network updates, packet drop, out-of-order packets, SDN, software-defined datacenter networks, DCTCP**

I. Introduction

The emerging software-defined network (SDN) offers significant benefits for DCN to orchestrate data transmission by fast-growing enterprises such as Microsoft [1] and Google [2] etc. The benefit of SDN often comes with the accompaniment of more frequent network updates due to its fast rescheduling of flows to meet various requirements at the data plane using a logically centralized controller [3]–[6].

When executing consecutive network updates often consist of multiple operations usually in a nonblocking manner, consistency properties must also be preserved [7]. Frequent flow rescheduling in SDN scenarios incurs serious network update problems refer to network confusions triggered by careless updating methods [8], including forwarding loop, forwarding blackhole, network policy violation, and link congestion. These problems cause inconsistency in network updates. Consequently, packet drop and out-of-order packets problems happen. The bulk of earlier research provides the solutions of forwarding loop, forwarding blackhole [1], [3], [5], [9]–[12], link congestion [1], [3], [5], [6], [9], [13]–[16] and policy violation [4], [12] to maintain the consistency properties during network updates.

According to [17], the measurements unveil that 99.91% of traffic in datacenter comprises Transmission Control Protocol (TCP) traffic. Meanwhile, the reroute breaks the assumption of TCP and has an impact on TCP traffic that significantly degrades the network performance. The packet drop and out-of-order packets problems are indeed severe when reroute. We can classify the expected problems of TCP during network updates as follows.

1) There will be disorders, which will trigger unexpected retransmission and window size suppression. Many researchers try to solve the problems that are cost-based either on timer or DUPACK threshold estimation.
2) TCP always starts from a ”slow start” mechanism. However, when the network reroutes a flow, it will introduce a ”quick start” on the new path, which may incur severe congestion and out-of-order packets as well, especially when the update is inconsistent and frequent respectively or the flow scheduling is not perfect.

Considering theses TCP problems (e.g. packet drop and out-of-order packets), we can easily infer that TCP has poor performance when there are network updates and becomes even poorer when network updates are frequent. Hedera [18] uses 5 *sec* frequency of network updates. They trust that sub-second and potentially sub-100 *ms* updates are possible such as authors in [19] uses 1 *ms*, 5 *ms*, and 1000 *ms*.

In this paper, we propose a TCP modification, called Network Update Friendly TCP (NUFTCP) based on DCTCP, to solve the above problems caused by network updates in software defined datacenter network: packet drop and out-of-order packets. We argue that TCP needs to get involved in the network updates because the network state changes are not very transparent to TCP.

We systematically analyze the effects of network updates and demonstrate with simulations to justify that our model successfully captures the effects of network updates. Our approach does not affect the performance of DCTCP while being resilient to high-frequency and inconsistent network updates. The results show NUFTCP performs better than DCTCP.

We make the following main contributions in this paper:

1) We perform comprehensive experiments and analysis to understand the behavior of existing state-of-the-art TCP designs used in datacenters during frequent and inconsistent network updates.
2) We highlight the problems of packet drop and out-of-order packets triggered during network updates that affect real data transmission significantly.
3) We introduce the TCP implementation called NUFTCP, which is an extension of DCTCP to deal network updates

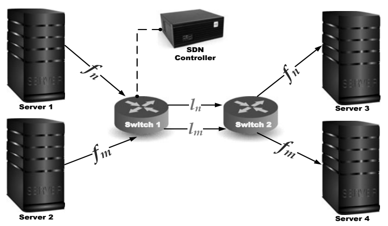

Fig. 1. Network topology of network updates for TCP variants.

TABLE I
EXPERIMENTATION PARAMETERS

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 Sec - 300 Sec
ACK Delay Threshold	2 Packets
ACK Delay Timeout	200 ms
Reroute Time	1 Sec

smoothly.

4) We use simulations to evaluate NUFTCP that shows the better performance during frequent and inconsistent network updates in software-defined datacenter network.

II. Analysis of TCP variants over Network Updates

In order to observe the behavior and the performance of different TCP variants, we conduct a series of experiments. To the best of our knowledge, this is the first study on the TCP performance focuses on the case of frequent and inconsistent network updates. (Note: Figures omitted due space limitations)

A. Experimental Setup

Fig. 1 depicts the topology we used in the experiments. It consists of four servers Intel Xeon X5650 with 6 cores at 2.67GHz, which act as the senders (*Server 1* and *Server 2*) and the receivers (*Server 3* and *Server 4*). We use Linux kernel 4.9. The two switches H3C S6800 (*Switch 1*) and H3C S6300 (*Switch 2*) are connected to the senders and the receivers respectively. Both the switches support OpenFlow 1.3 with 10Gbps link speed and connect to each other with two ports. Therefore, there are two paths from one sender to one receiver. SDN H3C VCF Controller is connected to the switch with a third port.

The controller controls the actual path used by each flow. We use two types a number of flows in the experiments ($f_n : Server_1 \rightarrow Server_3$ and $f_m : Server_2 \rightarrow Server_4$).

Fig. 1 is the initial network state. In the normal state, the flows are forwarded like this as follows:

f_n ($f_1, f_2, f_3 ... f_n$): *Server 1* to *Server 3* via Link l_n (*Switch 1* to *Switch 2*)

f_m ($f_1, f_2, f_3 ... f_m$): *Server 2* to *Server 4* via Link l_m (*Switch 1* to *Switch 2*)

After the network update, routes change, the traffic at link l_n shifts to link l_m and vice versa. The other scenario is, if link l_n failure happens, the network state is as follows:

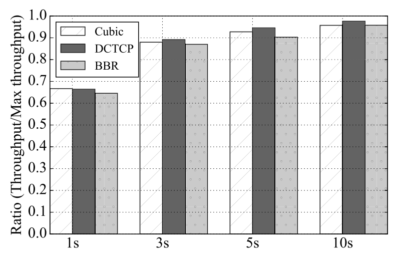

Fig. 2. An effect of network updates on the performance of TCP variants.

$f_n + f_m$: *Server 1* and *Server 2* to via Link l_m (*Switch 1* to *Switch 2*)

There are two types a number of flows f_n and f_m, which are initiated by *Server 1* and *Server 2* respectively. These flows are forwarded towards *Switch 1*. When the controller launches a network update, just to examine the behavior of TCP variants, we reroute the flow traffic initiated by *Server 1* towards link l_m and *Server 2* towards link l_n. We set a one-second threshold level to shift the traffic routes after every second for five minutes of experiment execution and other parameters for experiment are given in Table I. It may introduce frequent and inconsistent network state, consequently; it encounters out-of-order packets, link congestion and packet drop problems. This will result in significant throughput degradation due to the unnecessary retransmissions and flow throughput suppression.

B. Performance Analysis during Consistent Network Updates

The consistency requires that flows are to be forwarded to the new routing paths or reroutes seamlessly, which enforces dependencies among rules in flow tables along a routing path across multiple switches [20]. Thus, the correct order of updates is the most important to keep the consistency [21]. DCN updates happen frequently, whether activated by the operators, applications, or sometimes even failures [9]. Keeping high network utilization requires frequent network updates, which demands high-efficiency [1], as tens of thousands of flows in a few milliseconds happen.

We can observe the effect of frequency of network updates on the overall performance of the network as shown in Fig. 2, with TCP variants. This experiment runs for 1 minute to realize the relationship between the frequency of network updates and overall throughput performance of the network. It takes the frequencies of network updates with 1 second, 3 seconds, 5 seconds and 10 seconds, which shows the performance loss of the network as an estimated more than 30%, 10%, 5% and 2% respectively. Due to frequent flow dynamics, the data plane and controller should frequently update in time [22]. Therefore, we consider more frequent network updates.

In the experiment, the SDN controller launches the network update in terms of rerouting for every second by setting as a threshold. Now, we perform the experiment using two scenarios with the same topology; one is with large buffer size and the other one with small buffer size.

We conclude the most important and noticeable points. In the first scenario, using a large buffer size, TCP Cubic and DCTCP show a higher average number of retransmissions;

however, they have no packet drop. Therefore, we can conclude that the out-of-order packets problem happens during frequent network updates.

In the second scenario with small buffer size, BBR has a very low average number of retransmissions whereas the rest of TCP variants have a medium level of extent. BBR suffers from the most serious packet loss problem and the lowest packet drop rate is observed in DCTCP, while TCP Cubic shows packet drop about half of times lesser than BBR and near about half of times more than DCTCP.

C. Analysis for Inconsistent Network Updates

Inconsistencies emerge during data plane state changes on the desecration of policies or rule updates due to varying delayed coordination or no coordination across multiple switches or between controller and switch [20]. We introduce the inconsistency of 100 *ms* update delay by SDN controller in switches to examine the TCP variants using same metrics with the first scenario of large buffer size.

In inconsistent network updates with the first scenario with large buffer size, the situation is almost similar, as we have seen in the second scenario of frequent network updates as discussed before. Therefore, we can easily infer that situation with a small buffer size about the performance of TCP variants would become worse.

To recapitulate, the existing TCP variants are not good enough when there are frequent and inconsistent network updates. Thereby, we propose NUFTCP to deal with network updates in a smooth manner.

III. NUFTCP Design

We devise a design of NUFTCP. The key goals of NUFTCP are to avoid the packet drop during network updates in software-defined datacenter network by limiting the window size with queue management and also coping with the out-of-order packets problem. It consists of two major parts, which require modifications. The first part is to limit the window size on the transmit side to avoid the packet drop during an inconsistent network state. The second part describes the delay duplicated ACKs on the receiver side to alleviate the impact of out-of-order packets. Here, Table II represents the symbols used in the design part.

A. Limit the Window Size and Queue Management

1) Why do we care about Packet Drop instead of Congestion Avoidance?: NUFTCP does not avoid congestion. Instead, it only avoids severe congestion, which leads to packet drop. TCP fills the queue until a packet drop occurs, whereas the length of an average queue is unsurprisingly high. Thereby packet drop chances are very low until a queue becomes full and then arriving packets from all flows can be lost. In the worst cases, packet drop can reason severe defacement of received data or even the complete nonexistence of a link. That is why we are more concerned about packet drop. NUFTCP realizes this goal through the inherent negative feedback.

TABLE II
Key Notations

Symbol	Meaning
f_i	The set of flows
s_i	The set of data size
t_i	The set of round-trip time (RTT)
r_i	The set of flows throughput
$\hat{t}$	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

2) The relationship among Data Size, Flow Throughput and RTT: In TCP, there is a relationship among RTT t, data size s transmit in t and flow throughput r: $s = r \times t$. If s is fixed, r will decrease whereas t increases. Assume t is fixed, when a link begins to be congested, the queue of the link effects in terms of starting increment in the length. Consequently, the queuing latency of the packets, which are transferring through this link turns out to be longer. As a result, the RTT of the corresponding flow will be longer, and the throughput will be lower to decrease the congestion extent. Therefore, when we transmit the data size in one RTT, the throughput harnesses by a negative feedback. As long as the buffer size of the queue is large enough, there will be no packet drop.

3) Queue Management: We will discuss about derived stable equation to take Fig. 3 as an example. Before the update, $s_1 = r_1 \times t_1$ and $s_2 = r_2 \times t_2$ (s_i, r_i and t_i denote the corresponding variables of flow f_i). In this case, $r_1 = r_2 = c$ (c is the link capacity)and $t_1 = t_2 = \hat{t}$. ($\hat{t}$ is the round trip time without queuing latency).

When the network begins to update, it presents inconsistent network state, this is the reason why f_n network state updates, and it changes its route whereas f_m network state remains the same. So, during the inconsistent network state, f_n shifts the flow towards that intermediate node which is already used by f_m. Both flows share the same link and the queue is overwhelmed that can cause the packet drop. For simplicity, we assume the queue length (in bytes) in the stable state is l. Then the queuing latency is equal to the queue length by the link capacity ($\tau = l/c$). At the stable state, $s_1 = r_1' \times t_1'$ and $s_2 = r_2' \times t_2'$. $t_1' = t_2' = \hat{t} + \tau$ and $r_1' + r_2' = c$. Through the above equation, we have $l = \Sigma s_i - c\hat{t}$. Because $\Sigma s_i \leq 2c$, we always have $l \leq c\hat{t}$. Therefore, the maximal queue length should be $l_{max} < l + \Sigma s_i$ (the queue length will not be increased if the queue length in the last RTT exceeds l). This analysis can be easily extended to more flows. The conclusion is

$$l_{max} < 3c\hat{t} \tag{1}$$

For example, if $c = 10Gbps$ and $\hat{t} = 250\mu s$, $l < 937.5KB$. That means if the buffer size is 1MB, there should not be packet drop. The $equation$ (1) guarantees to circumvent the packet drop and stability during network updates.

4) How NUFTCP limits the Window Size?: NUFTCP takes advantages of the above negative feedback. NUFTCP tries to limit the window size w because $w \approx s$ for one TCP flow when there is a congestion. However, if the flow does not encounter a congestion, w may be much larger than s because w is increasing all the time. Because congestion avoidance increases the window size continuously but does so linearly. In order to avoid this problem, NUFTCP restricts the upper limit of window size. So, NUFTCP counts the average actual data size acknowledged $\overline{s}$ (EWMA) in the past $\hat{t}$ (preconfigured according to the actual value). Then it will set the upper limit of window size in the next $\hat{t}$ to $1.5s'$ (we use the factor 1.5 because the window size must be able to increase).

Note that NUFTCP is based on DCTCP, which means that packets will not drop by Random Early Detection (RED).

B. Delay Duplicated ACKs

In order to cope with the out-of-order problem, NUFTCP delays the return of duplicated ACKs when there is a network update. Only if the duplicated ACK is delayed for time T and the corresponding data packet has still not arrived yet, it will be actually returned. Otherwise, the duplicated ACK is canceled. Here T is a preconfigured time threshold which equals to the maximal RTT difference on different paths, mostly smaller than 1 or 2 milliseconds. Since the packet drop is reduced by limiting the window size, the impact of delaying the fast recovery of the transmit side is tolerable. The only problem is how NUFTCP detects network update.

NUFTCP requires the cooperation of switches (similar to ECN). A flow table entry in the switches is associated with a version number. Each time the controller recalculates a flow table entry, the version number should be increased by 1. For each packet matching the entry, it will be tagged with the current version number of the entry (if the version number is larger than the one carried in the packet). The first 4 bits in the TTL field is used for version number tagging because it is useless for intra-datacenter traffic whose maximal hops is smaller than 15. In this way, NUFTCP can detect network updates by detecting the TTL changes. NUFTCP can explicitly detect the network update and delay duplicated ACKs.

IV. Evaluation

In this section, we present simulation parameters, network topology and collected results that demonstrate the performance of NUFTCP. Existing network updates solutions [1], [3]–[5], [9], [10], [18], [21] are considering consistency properties violation problems and frequency of network updates, thereby NUFTCP cannot be compared with them. Because, NUFTCP is focusing on how the network updates affect the TCP performance that leads to packet drop and out-of-order problems. Therefore, we can only compare our solution with existing state-of-the-art TCP variant DCTCP that performs better than others as shown in Section II.

A. Network Topology and Simulation Parameters

We evaluate NUFTCP through simulations (based on NS3) using two scenarios with the large buffer size and the small buffer size. We compare NUFTCP with the DCTCP using

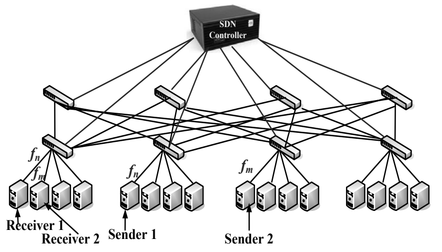

Fig. 3. Software-Defined Datacenter Network topology

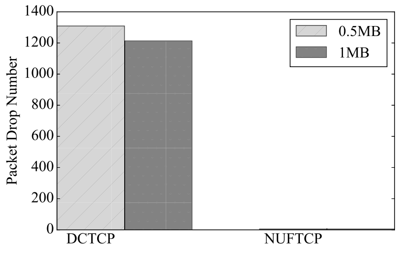

Fig. 4. Number of packet drop comparison of DCTCP and NUFTCP during network updates with large and small buffer size scenarios

software-defined datacenter networks. We are using state-of-the-art typical 2-tier Clos as network topology used in DCN and connected via SDN controller as shown in Fig. 3. The network uses ECMP. All the links are 10Gbps with $\hat{t} \approx 500\mu s$. The sizes of the large buffer and the small buffer are 1 MB and 0.5 MB respectively. For every 25 *ms*, we assume that one of the switches in the aggregation layer is down (or upgrading) and the controller sends corresponding update information to edge switches. In this format, flows that exactly go through the down (or upgrading) switches have to change their forwarding paths back and forth. According to our design, NUFTCP uses 4 bits of TTL field of the packet header to identify the version number, so the maximum number of different versions in our simulation is 16.

According to our topology, f_n flows are sent from *sender1* to *receiver1* and f_m flows are sent from *sender2* to *receiver2* at the same time. During the update, there is inconsistency among switches, so we simulate that there are $0 \sim 1$ *ms* (the difference of update time is generated randomly as authors in [19] uses 1 *ms* as well) update delay in the switches.

B. Results and Analysis

Here, Scenario I and Scenario II present NUFTCP evaluation with the large buffer size and the small buffer size respectively. (Note: Figures omitted due space limitations)

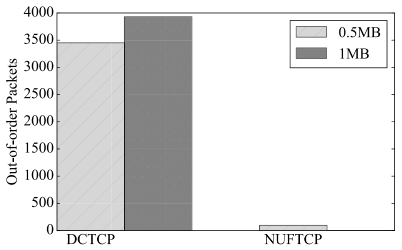

Fig. 5. Out-of-order packets comparison of DCTCP and NUFTCP during network updates with large and small buffer size scenarios

1) Scenario I: NUFTCP outperforms DCTCP by comparing actual throughputs. Results of DCTCP having packet drop at the start that shows an adverse effect on the throughput of flows, after a while, again packet drop happens, subsequently, there is convergence time of throughputs which is somehow longer. As a contrast , NUFTCP shows the steady throughput because there is no packet drop, and the out-of-order is well handled. We performed simulations many times, every time, DCTCP results in packet drop with different numbers (1188, 1294, 1159 etc.) and we show the average number of packet drop (1213) in Fig.4. Since, we are using a random delay to maintain inconsistency during network updates, whereas, there is no packet drop using NUFTCP every time, it shows stable throughput and performance. DCTCP also displays a high average number of out-of-order packets (3930) with large buffer size as we can observe in Fig. 5, the large buffer can hold more packets; therefore, the possibility is also high for out-of-order packets, while NUFTCP outclasses the DCTCP without out-of-order. Therefore, the simulation results advocate that NUFTCP is much better for SDN flow rescheduling in a software-defined datacenter network.

2) Scenario II: NUFTCP performs better than DCTCP with actual throughputs using small buffer size. NUFTCP depicts stability in throughput and similarity. It deals with packet drop and out-of-order problems in an efficient manner as it can observe in Fig. 4 that there is no packet drop. On contrary, we can observe the DCTCP shows poor performance that packet drop initiates at the earlier stage and continues for longer than before, then convergence time throughputs. After multiple simulations, every time, we find the packet drop problem (1381, 1366, 1181 etc.) in DCTCP, then we take the average number of packet drop (1309) of DCTCP as is shown in Fig. 4, whereas NUFTCP has no packet drop every time. In Fig. 5, DCTCP reveals a large number of out-of-order packets (3451) in average with small buffer size, but less than large buffer because small buffer can create a situation of early packet drop, in contrast, NUFTCP depicts a very low number out-of-order packets (96). Therefore, we can conclude that the performance of NUFTCP is reasonable to handle the frequent and inconsistent network updates with small buffer size as well in a software-defined datacenter network.

V. Conclusion

In this paper, we perform extensive experiments on existing TCP variants in the scenario of frequent and inconsistent network updates with the large buffer size and the small buffer size. We observe that existing TCP variants are insufficient to handle the frequent and inconsistent network updates. We propose NUFTCP, that can solve the problems of packet drop and out-of-order during network updates in the software-defined datacenter network. The evaluation shows the performance of NUFTCP is much better than DCTCP when the network updates are frequent and inconsistent.

References

[1] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer, "Achieving high utilization with software-driven wan," in *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4. ACM, 2013, pp. 15–26.

[2] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, and others, "B4: Experience with a globally-deployed software defined WAN," in *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM*. ACM, pp. 3–14.

[3] T. D. Nguyen, M. Chiesa, and M. Canini, "Decentralized consistent updates in sdn," in *Proceedings of the Symposium on SDN Research*. ACM, 2017, pp. 21–33.

[4] S. Vissicchio and L. Cittadini, "Flip the (flow) table: Fast lightweight policy-preserving sdn updates," in *Computer Communications, IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on*. IEEE, 2016, pp. 1–9.

[5] X. Jin, H. H. Liu, R. Gandhi, S. Kandula, R. Mahajan, M. Zhang, J. Rexford, and R. Wattenhofer, "Dynamic scheduling of network updates," in *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4. ACM, 2014, pp. 539–550.

[6] J. Zhu, J. Hua, M. Liu, Y. Li, and K. Cao, "Trus: Towards the real-time route update scheduling in sdn for data centers," *IEEE Access*, vol. 8, pp. 68 682–68 694, 2020.

[7] G. Li, Y. R. Yang, F. Le, Y.-s. Lim, and J. Wang, "Update algebra: Toward continuous, non-blocking composition of network updates in sdn," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 1081–1089.

[8] 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.

[9] H. H. Liu, X. Wu, M. Zhang, L. Yuan, R. Wattenhofer, and D. Maltz, "zUpdate: Updating data center networks with zero loss," in *Proceedings of SIGCOMM*. ACM, pp. 411–422, 00103.

[10] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger, and D. Walker, "Abstractions for network update," in *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*. ACM, pp. 323–334, 00385.

[11] S. Liu, T. A. Benson, and M. K. Reiter, "Efficient and safe network updates with suffix causal consistency," in *Proceedings of the Fourteenth EuroSys Conference 2019*, 2019, pp. 1–15.

[12] B. Finkbeiner, M. Gieseking, J. Hecking-Harbusch, and E.-R. Olderog, "Model checking data flows in concurrent network updates," in *International Symposium on Automated Technology for Verification and Analysis*. Springer, 2019, pp. 515–533.

[13] J. Zheng, H. Xu, G. Chen, H. Dai, and J. Wu, "Congestion-minimizing network update in data centers," *IEEE Transactions on Services Computing*, 2019.

[14] S. Luo, H. Yu, L. Luo, and L. Li, "Customizable network update planning in sdn," *Journal of Network and Computer Applications*, vol. 141, pp. 104–115, 2019.

[15] J. Zheng, Q. Ma, C. Tian, B. Li, H. Dai, H. Xu, G. Chen, and Q. Ni, "Hermes: Utility-aware network update in software-defined wans," in *2018 IEEE 26th International Conference on Network Protocols (ICNP)*. IEEE, 2018, pp. 231–240.

[16] Y. Chen, H. Zheng, and J. Wu, "Consistency, feasibility, and optimality of network update in sdns," *IEEE Transactions on Network Science and Engineering*, vol. 6, no. 4, pp. 824–835, 2018.

[17] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data center tcp (dctcp)," in *ACM SIGCOMM computer communication review*, vol. 40, no. 4. ACM, 2010, pp. 63–74.

[18] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, A. Vahdat *et al.*, "Hedera: dynamic flow scheduling for data center networks." in *NSDI*, vol. 10, no. 8, 2010, pp. 89–92.

[19] A. Basta, A. Blenk, S. Dudycz, A. Ludwig, and S. Schmid, "Efficient loop-free rerouting of multiple sdn flows," in *Proc. IEEE/ACM Transactions on Networking (ToN)*, 2018.

[20] R. Mahajan and R. Wattenhofer, "On consistent updates in software defined networks," in *Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks*. ACM, 2013, p. 20.

[21] W. Wang, W. He, J. Su, and Y. Chen, "Cupid: Congestion-free consistent data plane update in software defined networks," in *Computer Communications, IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on*. IEEE, 2016, pp. 1–9.

[22] H. Xu, Z. Yu, X.-Y. Li, C. Qian, L. Huang, and T. Jung, "Real-time update with joint optimization of route selection and update scheduling for sdns," in *Network Protocols (ICNP), 2016 IEEE 24th International Conference on*. IEEE, 2016, pp. 1–10.