DCTCP-FQ: Enhancing Fairness and Convergence Time in Data Center Congestion Control

Haoyu Wang*, Xiaoqian Zhang†, Allen Yang*, Bo Sheng*
*Department of Computer Science, University of Massachusetts Boston, Boston, USA
†Department of Computer Science, University of Nebraska Omaha, Omaha, USA

***Abstract*—Efficient congestion control is vital for the performance of modern data centers, which host diverse and latency-sensitive applications. Data Center TCP (DCTCP) is a well-established algorithm that utilizes Explicit Congestion Notification (ECN) to achieve low latency and high throughput. However, DCTCP has drawbacks regarding convergence time and fairness, especially when managing flows that start at different times when there is already congestion. To address these limitations, we introduce DCTCP-FQ, an enhanced version of DCTCP that incorporates an active queue management strategy. DCTCP-FQ employs selective packet marking at congestion points, targeting flows with transmission rates exceeding the average sustainable rate. This strategy enables faster convergence for later-started flow and improved fairness without compromising the throughput and latency benefits of DCTCP. Extensive simulations demonstrate that DCTCP-FQ achieves significant improvements in convergence time and flow fairness compared to traditional DCTCP in the scenario where a congestion point adds a new flow. According to our evaluations, DCTCP-FQ archives about an average of 64.82% faster in flow completion time than DCTCP for smaller size flow ranging from 200KB to 1000kb under a 1Gbps congestion point and about 43.91% increase in Jain's fairness index in multi-hop multi-bottleneck flows topology for mixed starting flows, making it a robust solution for next-generation data center networks. This paper details the design, implementation, and evaluation of DCTCP-FQ, highlighting its potential to enhance network performance and fairness in data center environments.**

I. Introduction

In modern data centers, efficient and fair network congestion control is critical for optimizing performance and ensuring the smooth operation of various applications. Data Center TCP (DCTCP) [1] has been widely adopted due to its ability to maintain low latency and high throughput by leveraging Explicit Congestion Notification (ECN) for fine-grained congestion signaling. However, as data center networks continue to evolve, there is a growing need for congestion control mechanisms that sustain high performance and achieve faster convergence and fairness among competing flows.

This paper presents DCTCP-FQ, an enhancement of the traditional DCTCP algorithm. DCTCP-FQ integrates an active queue management approach designed to address DCTCP's limitations, specifically targeting convergence time and fairness in flow handling. Through extensive simulations, we demonstrate that compared to DCTCP, our solution, DCTCP-FQ, maintains comparable throughput and latency performance, significantly reducing convergence time and improving fairness between flows that start at different times and with different round trip times.

The proposed DCTCP-FQ algorithm employs a fair queuing strategy that ensures newer flows receive equitable bandwidth allocation without compromising the performance of ongoing flows. This results in a more balanced network environment, reducing the chances of flow starvation and improving overall network efficiency. Our evaluation shows that DCTCP-FQ can achieve these improvements with minimal additional overhead, making it a viable option for deployment in existing data center networks.

The remainder of this paper is structured as follows: Section II explains the background and motivation for our design. Section III details the design and implementation of the DCTCP-FQ algorithm. Section IV presents the simulation setup and evaluation metrics. Section IV discusses the simulation results, highlighting the convergence time and fairness improvements. Section V reviews related work in congestion control algorithms for data centers. Finally, Section VI concludes the paper and outlines potential directions for future research.

II. Background and Motivation

In this section, we will first introduce the traffic patterns of a data center environment and then the existing congestion control algorithms and their limitations, which motivate our design of DCTCP-FQ.

A. Data Center Traffic Pattern

Data center traffic is significantly different from wide area networks [2] and is often characterized by a mix of large, long-lived flows (elephant flows) and small, short-lived flows (mice flows). Elephant flows typically handle bulk data transfers, such as backups or big data analytics, while mice flows correspond to latency-sensitive tasks like web searches and database queries. These various flow sizes and durations create difficulties for congestion control and resource allocation. ([3] and [2].)

Incast is a common traffic pattern in data centers where many servers simultaneously send data to a single receiver, causing a sudden burst of traffic that can overwhelm the receiver's buffer and lead to packet loss. This pattern is prevalent in storage and caching systems, where multiple servers respond to a single client's request [4]. Quantitatively according to [1] measurements, data center traffic consists of query traffic (2KB to 20KB in size), delay-sensitive short

979-8-3315-2096-0/25/$31.00 ©2025 IEEE

messages (100KB to 1MB), and throughput-sensitive long flows (1MB to 100MB). As discussed in [5] and [6], the query traffic experiences incast impairment in the context of storage networks. Both Query and delay-sensitive short messages experience long latency due to long flows consuming some or all of the available buffer in the switches. This is because the congestion point does not maintain a persistently small queue size while keeping the high throughput, and also, the congestion point treats high rate flow and low rate flow in terms of marking or dropping with the same weight so that the short flow is very slow to grab a fair share of traffic bandwidth at the congestion point.

B. Limitations of Existing Congestion Control Schemes

Many congestion control algorithms have been published over the years to address the problems described above. They mainly fall into two large categories: delay-based protocol and active queue management (AQM) based protocol. The delay-based protocol uses round-trip time (RTT) measurements for congestion control, providing rapid response to congestion without requiring changes to network switches, for example, [7]. However, in environments with fluctuating RTTs due to varying path lengths or network load, the scheme's performance can degrade. These schemes also face scalability issues in very large data centers where RTT measurements can become less reliable and more variable [8]. For AQM-based protocol, DCTCP has been the most successful and popular one, it uses the ECN for fine-grained congestion feedback and adjusts the sending rate proportionally to the congestion level to achieve high throughput and low buffer occupancy. However, it trades off the convergence time of later-started short flows like query traffic and short message traffic.

III. Design of DCTCP-FQ

The convergence time is the time required for a new flow to grab its share of bandwidth from an existing flow with a large window size, which the design of DCTCP traded off for achieving high throughput and low delay. It is not considered a major concern in DCTCP, and it is even slightly slower in DCTCP compared to traditional TCP.

The goal of the DCTCP-FQ algorithm is to achieve high burst tolerance, low latency, high throughput, and quicker convergence time. To achieve this, DCTCP-FQ reacts to congestion in proportion to the extent of congestion just like what DCTCP does, but DCTCP only reacts to those flows already with large throughput to reduce the large flow's bandwidth to accommodate smaller newer flows' need of increasing throughput bandwidth so that the congestion point can maintain a low queue at the same time giving a fair share of the bandwidth to all flows pass through.

The DCTCP-FQ algorithm has three main components:

1) ***The Selective Marking at the Switch****:* The selective marking strategy at the congestion point is the key to our DCTCP-FQ algorithm to achieve a quicker converge time and fair stable distribution of the valuable bottleneck bandwidth. We employ an active queue management scheme with consideration of the different rates of the flow passing by. The following parameters are considered to determine if a packet should be marked or not when a queue is deemed as in a congestion state:

- K: the marking threshold ,
- R_{avg}: an estimation of the average sending rate based on the total number of different flows for the total bandwidth of the switch link and the latest transmitting rate,

When a packet arrives at the switch waiting for enqueue, DCTCP-FQ checks the two parameters to determine if this packet should be marked with the CE codepoint for indicating congestion. The first one is if the current queue length is larger than the threshold K; the second is if the flow rate of the current packet is above the average rate R_{avg} at the queue; if both conditions meet this packet will be marked to indicated congestion and the flow should reduce speed. It has to be pointed out here that the average rate R_{avg} setting should take into consideration the RTT time needed for the reduction of sending rate taking effect at the switch after a congestion signal is sent out. So often the parameter R_{avg} should be set proportionally less than the estimation of the actual average rate to avoid queue length growing much larger than the setting threshold due to the reaction time of RTT.

The marking condition check is simple, but the accurate estimation of the average rate and the instant transmitting rate for each flow at the switch are difficult to obtain. There are two challenges to address:

1) How to define a flow? This would be the hardest challenge for the wide area network since there can be numerous flows past a given congestion point, but for a data center environment the congestion point switch most likely knows the server flow it is handling, such as the server IP, the service port, protocol, etc.
2) Estimate the instant transmitting rate and the average rate. To accurately estimate the instant transmitting rate for each flow, we should maintain a counter queue to store all the packets dequeued within the latest RTT where each flow packet is identified with the packet dequeue time so that when the dequeue time of the packet is outside of the latest RTT window, it is removed from the counter queue. Within the counter queue, we have the parameter for the total number of different flow N_f, the total number of packets N_p, the total number of packets for each flow, and the ideal average number of packets for each flow N_{avg}.

   In fact, our design can be simplified by removing the requirement of the dequeue time for each packet because when a packet needs to be marked, the overall transmitting speed will always reach the full link speed, and the total number of packets within the counter queue will stay the same as $RTT * Bandwidth$. **And when the overall transmitting speed does reach the full link speed, we know the queue will be empty and there is no need to mark any packets, thus even if the counter queue may not reflect a more accurate estimation of each**

flow speed, it does not matter.

Algorithm 1: Flow Rate Estimation

Data: B_n: Bandwidth at the bottleneck
$Size_p$: A preset packet size
P_i: arriving Packet for flow i
Q: an FIFO queue
CQ: a counter map
$N_p \leftarrow RTT * B_n/Size_p$: the total number of packets allowed in the queue
$N_f \leftarrow 0$: the total number of flows
$N_{avg} \leftarrow 0$: the ideal average packet supposed int the queue
Result: No return

```
CQ[fi] ++
if CQ[fi] == 1 then
    Nf ++; Navg = Np/Nf
end
while len(Q) > Np do
    px = Deque(Q)
    CQ[px] − −
    if CQ[px] == 0 then
        Nf − −; Navg = Np/Nf
    end
end
```

After addressing these two challenges, we are ready to explain our marking algorithm.

Algorithm 2: Marking at Enqueue

Data: $FR \leftarrow 3/5$;
N_q the current number of packet in queue;
K The marking threshold;
P_{fi} the packet of flow_i trying to enqueue;
Result: y

```
y = false;
if Nq > K&&CQ[pX] > Nave * FR then
    y = true # Marking the packet with ECN;
end
```

To determine whether an incoming packet gets marked or not at the bottleneck, the algorithm will check both conditions: if the current queue size is larger than the marking threshold, and if the flow transmitting rate of the current packet is larger than the ideal average flow rate maintained at the Algorithm 1. We set a factor of $3/5$ on the average flow rate to accommodate the RTT needed for the sender to tame its sending rate, which means when the flow rate is about to reach the ideal average flow rate, it should begin to mark its packets for not increasing more sending rate. During the feedback RTT time the sending rate is still increasing to reach to the ideal sending rate.

*2) **ECN-Echo at the Receiver**:* Just like the DCTCP does at the receiver, DCTCP-FQ will ACK every packet back to the sender with the ECN-Echo flag whenever an arriving packet has a marked CE codepoint. In this way, the sender gets the exact portion of marked packets with a window to act upon.

*3) **Controller at the Sender**:* The controller at the send maintains the same settings as DCTCP.

IV. Evaluation

This section will demonstrate the simulation performance of DCTCP-FQ on the same simulation condition that DCTCP used for comparison using the ns3 simulator [9]. We have published our implementation and evaluation data results here [10]. Specifically, we will use a standard dumbbell topology with a bottleneck bandwidth of $1Gbps$ and $10Gbps$ respectively corresponding to a marking threshold of $k = 20$ for $1Gbps$ and $k = 65$ for $10Gbps$; the leaf node bandwidth is set as $1Gbps$ for all nodes; the RTT is set to be $160\mu s$ where the leaf link is delay is $30\mu s$ and bottleneck delay is $20\mu s$; in all experiment estimation gain g is set to be $1/16$. O

A. DCTCP-FQ *Queue Size at Bottleneck*

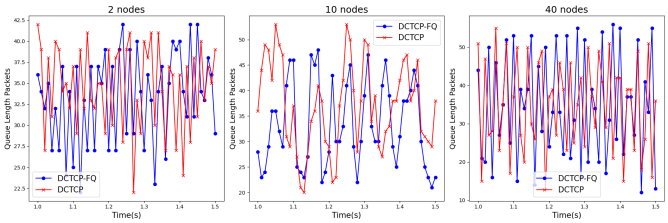

Fig. 1. Comparisons between the queue size at the bottleneck

We have evaluated the queue size at the bottleneck in this experiment, with the above settings on different numbers of flows $N = 2, 10, 40$ long-lived DCTCP flows and DCTCP-FQ flows. As is shown, these two flows have the same general seesaw shape of increasing and decreasing around the marking threshold at the bottleneck which proves that DCTCP-FQ will maintain the same latency performance as the DCTCP under the same environment settings. Additionally, both types of flow achieve the maximum throughput of the bottleneck and the link utilization is 100%. This is understandable since the queue length at the bottleneck never goes below zero in our simulation.

B. DCTCP-FQ *Convergence time for short small-sized flow under congestion*

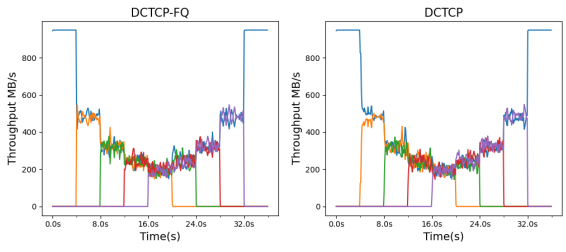

Fig. 2. Comparisons between the convergence performances

In this section, we will conduct simulations to demonstrate the convergence time performance and Jain's fairness for

DCTCP and DCTCP-FQ. First, we do the simulation of the settings on the DCTCP paper [1], in which we set up 10 host nodes via $1Gbps$ links to a switch of $1Gpbs$, as a standard dumbbell topology where there are 5 nodes for sending traffic and 5 nodes for receiving. We start a single long-lived flow, and then we sequentially start and then stop at the other senders gaped by 4 seconds. The time series depicting the overall flow throughout is shown in Figure 2. As all the flows come and go, they quickly converge to their fair share, both types of flows demonstrated the same convergence performance at the scales of seconds.

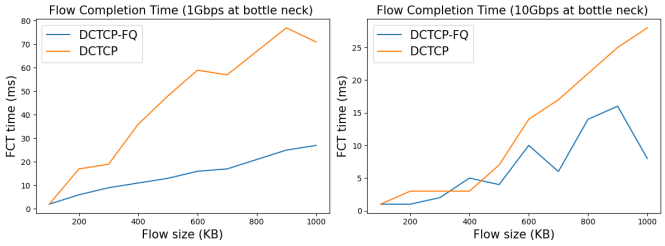

Fig. 3. The flow completion time of small-sized flows within congestion

As noted in the DCTCP paper [1] the data center has many sorts of short flows such as **Query Traffic**, **Background Traffic**. Query traffic is very short and latency-critical flows. Background traffic consists of both large and small flows. For all these short small traffic, their flow completion time and latency can be heavily affected by the congestion status of the switch and how the active queue algorithm is treating them. Since the marking algorithm employed by DCTCP is a one-slate marking strategy where it marks all the incoming packets equally when there is congestion; as a result, even if the later started flow have very low sending rate, it still can revive marking packets and reduce its sending rate; although the sending rate will eventually catch up for DCTCP flows, it doesn't matter too much, since the short query flows are smaller in size and it finishes in the order of ms. So, for DCTCP, even if it can offer low latency at the bottleneck, it can be relatively unfair to later start short flow, where the convergence time for them can be longer even compared to traditional TCP. This trade-off is well documented in the paper of DCTCP.

Our algorithm, DCTCP-FQ, is a very good solution to this trade-off while maintaining the advantages of DCTCP. In this experiment, we show the comparisons of flow-completion time for the short-size flows in a congested switch of $1Gbps$ and $10Gbps$. From Figure 3, we can see that for small-sized flow ranges from $200KB - 1000KB$ DCTCP-FQ achieves obviously better results than that of DCTCP. To calculate the average difference, we use this equation $\frac{1}{n}\sum_{i=1}^{n}|a_i - b_i|$, and for $1Gbps$ and $10Gbps$, DCTCP-FQ is faster on average by $29.46ms$ and $5.91ms$; to turn the difference in ratio then, it is 64.82% and 20.19% faster for $1Gbps$ and $10Gbps$.

C. DCTCP-FQ *Mutlihop, multi-bottle neck topology*

To evaluate the DCTCP performance in a multihop, multi-bottleneck environment, the DCTCP paper has done an experiment on the topology shown in Figure 4 The senders in group S1 and S3 each with 10 flows all send to the receiver group R1; there are another 20 senders in group S2 send to receiver group of R2. There are two bottlenecks in this topology, the $10Gbps$ link between Triumph 1 and the Scorpion and the $1Gbps$ link between Triumph2 and R1. Only the senders in group S1 encounter both these bottlenecks. We use these same settings on NS3 simulation on both DCTCP and DCTCP-FQ (we use the base DCTCP example written by Shravya and Tom Henderson etc).

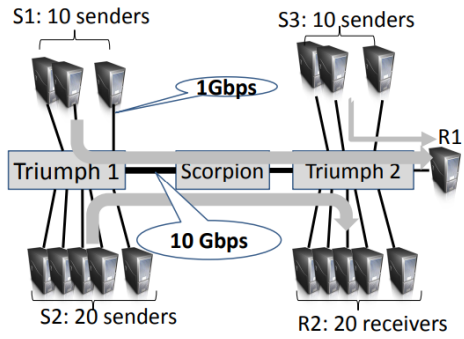

Fig. 4. Multihop topologgy

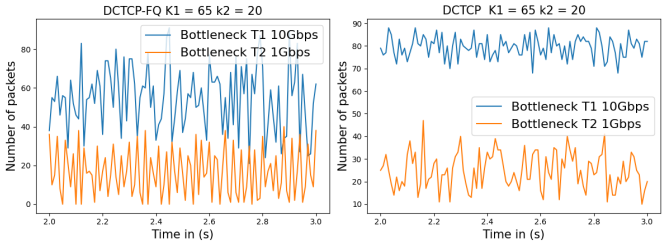

Fig. 5. Multihop queue size at bottleneck T1 and T2

We find that for DCTCP: the average throughput for S1-R1 is 16.60Mbps with Jain's fairness index of 0.779; the average throughput for S2-R2 is 473.57Mbps with Jain's fairness index of 0.983; the average throughput for S3-R1 is 79.81Mbps with Jain's fairness index of 0.998; Aggregate user-level throughput for flows through T1: 9.637 Gbps; Aggregate user-level throughput for flows to R1: 0.964 Gbps; average queue size at bottleneck 1 is 81.2 packets; average queue size at bottleneck2 is 21.2. So for DCTCP both of the bottlenecks are working at the full link speed, and flows within each group archives high Jain's fairness index. We have to point out that the simulation has some performance differences between the bandwidth distributions on S1 and S3, which may be due to there being different settings of parameters since the DCTCP has not published all the parameters settings. However, the results still show that DCTCP is able to achieve high link utilization and low queueing delay.

For comparison for DCTCP-FQ: the average throughput for S1-R1 is 46.22 Mbps with Jain's fairness index of 0.993; the average throughput for S2-R2 is 458.75 Mbps with Jain's fairness index of 1.000; the average throughput for S3-R1 is 50.19 Mbps with Jain's fairness index of 0.996; Aggregate user-level throughput for flows through T1: 9.637 Gbps;

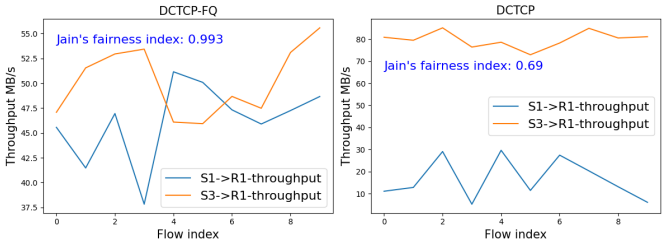

Fig. 6. Multihop throughputs for S1 and S3 to R1

Aggregate user-level throughput for flows to R1: 0.964 Gbps; average queue size at bottleneck 1 is 62.3 packets; average queue size at bottleneck2 is 18.4. So for DCTCP-FQ both of the bottlenecks are also working at the full link speed, and flows within each group archive high Jain's fairness index and better than DCTCP. And percentage increase of Jain's fairness index from DCTCP to DCTCP-FQ is 43.91%.

We also stress that not only did DCTCP-FQ achieve higher Jain's fairness within each sender group, it also archives significantly higher fairness between longer RTT and two bottlenecks sender group S1 and short RTT single bottleneck sender group S3 in Figure 6 which manifests that the DCTCP-FQ's design goal of giving a fair share of the bottleneck bandwidth to the smaller longer flow.

V. Related Work

Congestion control in data centers has been an active area of research due to the increasing demands for low latency and high throughput in cloud-based applications. Traditional TCP congestion control mechanisms, designed for wide-area networks, often fall short in the high-speed, low-latency environments of data centers. This has led to the development of several specialized congestion control algorithms. For example, DCTCP, introduced by Alizadeh et al., significantly improves upon traditional TCP by using ECN to provide multi-bit feedback on network congestion [1].

Various enhancements and alternatives to DCTCP have been proposed to address its limitations. For example, HULL (High-bandwidth Ultra-Low Latency) combines DCTCP with a rate limiter to achieve low latency [11]. TIMELY uses round-trip time measurements to adjust transmission rates, aiming for low latency and high throughput without requiring switch modifications [7]. PIAS (Priority-based In-Network Scheduling) prioritizes short flows to reduce their completion times, thereby improving fairness among flows of different sizes [12]. Several influential packet-scheduling-based schemes schedule data transmissions by assigning different levels of priority to packets. These schemes aim to ensure fairness and improve network performance, by maximizing the deadline meet rate for deadline flows and minimizing average flow completion time for non-deadline flows [13]–[15].

DCTCP-FQ builds on the foundation of DCTCP, introducing a selective marking strategy to address its fairness and convergence time issues. Through our evaluation, we show that DCTCP-FQ maintains DCTCP's desirable properties while offering significant improvements in convergence time and fairness, making it a compelling solution for modern data center networks.

VI. Conclusion

In this paper, we proposed a new variant of DCTCP, called DCTCP-FQ, which FQ stands for fair queue. The proposal of the DCTCP-FQ is motivated by the trade-off of convergence time DCTCP has for achieving both high throughput and low delay. This trade-off penalizes short small-sized flows that widely exist in the data center environment to grab a fair share of bandwidth from the long-lived large flows at the bottleneck. To alleviate this trade-off we develop a selective marking algorithm at the congestion point where low-rate flow packets are not marked when congestion happens, only the large transmitting rate flow packets are marked for the sender to reduce the sending rate. Multiple simulation results show that DCTCP-FQ can achieve a much quicker flow completion time by two to three times than that of DCTCP and still maintains overall high throughput and low latency at the bottlenecks as DCTCP which meets its design goals.

References

[1] M. Alizadeh, A. Greenberg, D. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data center TCP (DCTCP)," in *SIGCOMM '10*, 2010.

[2] A. G. P. P. S. Kandula, S. Sengupta and R. Chaiken, "The nature of data center traffic: measurements analysis," in *ACM IMC*, 2009.

[3] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *ACM IMC*, 2010.

[4] D. S. D. Nagle and A. Matthews, "The effects of incast on data center networks," in *IEEE LANMAN*, 2004.

[5] L. X. Y. Wu, M. H. H. Weissman and J. Yang, "Understanding tcp incast throughput collapse in datacenter networks," in *Proceedings of the ACM SIGCOMM 2013 conference*. ACM, 2013, pp. 13–24.

[6] G. V. R. R. V. Vasudevan, D. Borthakur and S. Shenker, "Safe and effective fine-grained tcp retransmissions for datacenter communication," in *Proceedings of the ACM SIGCOMM 2009 conference*. ACM, 2009, pp. 303–314.

[7] R. Mittal, V. Lam, N. Dukkipati, E. Blem, H. M. G. Wassel, M. Ghobadi, A. Vahdat, Y. Wang, D. Wetherall, and D. Zats, "TIMELY: RTT-based congestion control for the datacenter," *Comput. Commun. Rev.*, vol. 45, pp. 537–550, 2015.

[8] R. Mittal, V. Shukla, S. Ratnasamy, and S. Shenker, "Timely: Rtt-based congestion control for the datacenter," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4, pp. 537–550, 2015.

[9] ns 3 project, *ns-3: Network Simulator 3*, 2021, available at https://www.nsnam.org/.

[10] *DCQCN-FQ: NS3 Simulation Implementaion*, available at https://github.com/Haoyu2/ns-3-dev-git/tree/my/src/my/model.

[11] M. Alizadeh, A. Kabbani, T. Edsall, B. Prabhakar, A. Vahdat, and M. Yasuda, "Less is more: Trading a little bandwidth for ultra-low latency in the data center," in *NSDI, '12*, 2012.

[12] W. Bai, K. Chen, H. Wu, W. Wang, D. Li, and Y. Jiao, "Pias: Practical information-agnostic flow scheduling for data center networks," in *Proceedings of the ACM SIGCOMM 2015 conference*. ACM, 2015, pp. 221–232.

[13] J. Perry, A. Ousterhout, H. Balakrishnan, D. Shah, and H. Fugal, "Fastpass: a centralized "zero-queue" datacenter network," in *SIGCOMM 2014*, 2014.

[14] P. Gao, A. Narayan, G. Kumar, R. Agarwal, S. Ratnasamy, and S. Shenker, "phost: distributed near-optimal datacenter transport over commodity network fabric," in *The 11th ACM Conference on Emerging Networking Experiments and Technologies*, 2015.

[15] M. Handley, C. Raiciu, A. Agache, A. Voinescu, A. Moore, G. Antichi, and M. Wójcik, "Re-architecting datacenter networks and stacks for low latency and high performance," *ACM Special Interest Group on Data Communication*, 2017.