IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received November 22, 2021, accepted January 4, 2022, date of publication January 14, 2022, date of current version January 21, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3143160

CAPTON–Centralized All-Path Transmission for Over-Subscribed Datacenter Networks

NAN ZHOU AND JACK Y. B. LEE, (Senior Member, IEEE)
Department of Information Engineering, The Chinese University of Hong Kong, Hong Kong

Corresponding author: Jack Y. B. Lee (jacklee@computer.org)

This work was supported in part by the General Research Fund from the Hong Kong Research Grant Council under Grant GRF/14206521, and in part by the Chinese University of Hong Kong under Direct Grant 4055156.

ABSTRACT This work addresses an often-overlooked challenge in designing datacenter traffic control schemes. Specifically, most existing schemes were designed for network topologies with non-blocking network core. While this enabled the development of elegant solutions, core networks in practice are more likely to be *over-subscribed* by the access layer network due to cost considerations. Consequently, existing traffic control schemes may perform sub-optimally in real-world over-subscribed datacenter networks. This work proposes a new centralized flow-based scheduling scheme called CAPTON which schedules flow according to remaining flow size and link capacities to avoid congestion in over-subscribed networks. CAPTON employs all-path transmission to exploit all available paths and redundant links between source and destination hosts to maximize resource utilization and eliminate the need for routing. Unlike switch-based approaches, CAPTON does not require additional support from network switches and thus can be readily deployed in current datacenters equipped with commodity switches.

INDEX TERMS Datacenter, networking, scheduling, congestion, over-subscription.

I. INTRODUCTION

Datacenter is one of the essential building blocks for today's global Internet services. Unlike the wider Internet, networks *within* a datacenter are highly structured and optimized for very high bandwidth (10's to 100's Gbps) with very short delay (<100 us) [2]. While conventional Internet protocols such as TCP/IP can and do work in a datacenter network, their performances are often far from optimal [4], [10], [14]. For this reason, researchers have developed specialized protocols [4], [10] and novel traffic engineering schemes [1]–[3], [5]–[9], [11]–[14] to improve applications' network performance within a datacenter.

In this work, we investigate a practical issue in datacenter networks that have received relatively little attention in the literature. Specifically, most of the existing work was developed based on the assumption that the core network, i.e., the network interconnecting top-of-rack (ToR) switches, is *non-blocking*, otherwise known as the "big switch" model [1]–[3] (Fig. 1). This model removes the core network from resource allocation calculations, thereby enabling the development of many elegant solutions for traffic engineering [1]–[3], [14], [28].

The associate editor coordinating the review of this manuscript and approving it for publication was Chi-Tsun Cheng.

However, in large datacenters, the core network is far more likely to be *over-subscribed* due to cost considerations (e.g., 3:1 over-subscribed according to [24] and our conversations with CDN service providers). In over-subscribed datacenter networks, the combined access bandwidth of all host links will exceed the bisection bandwidth of the core and hence, core network congestion can no longer be ignored.

Our investigation reveals that core network over-subscription can substantially degrade the performance of current datacenter traffic control schemes, including the state-of-the-arts such as Pfabric [1], PIAS [2], and Hyline [14]. For example, the mean Flow Completion Time (FCT) of Pfabric for the websearch workload in a 3:1 over-subscribed topology could increase by ∼300% compare to non-over-subscribed topology (c.f. Section III).

This study develops a new traffic control scheme called CAPTON - Centralized All-Path Transmission for Over-subscribed Datacenter Networks to address the challenge due to over-subscription. There are four salient features in the design of CAPTON.

First, CAPTON adopts *all-path transmission* where packets of a flow are transmitted over *all* available equal-cost paths between the source and the destination to reduce flow completion time (FCT). This *eliminates* the need for routing and exploits the availability of multiple alternative paths between sender and receiver. Second, CAPTON employs a central

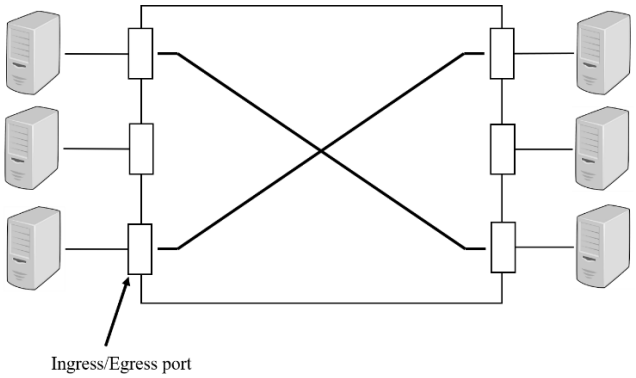

FIGURE 1. The big switch model for datacenters assumes full bisection bandwidth, congestions happen only at *first and* last hops (ingress/egress port) of the topology.

controller to schedule the flows' transmission order to reduce FCT and to prevent congestion. Third, CAPTON implements a scheduling threshold where flows smaller than that (e.g., 10 MSSs) are transmitted without scheduling to eliminate the performance penalty due to scheduling delay. Last but not least, CAPTON does not require any support from or runtime configuration of the network switches (e.g., forwarding table) and thus could be readily deployed in current datacenter networks using commodity switches.

We implemented CAPTON in NS2 [25] and evaluated its performance using two standard datacenter workloads [27], [28]. For example, compared to Pfabric [1] – the benchmark for near-optimal performance, CAPTON can reduce the FCT under the websearch workload by 18% and 45% at core network load of 50% and 100% respectively in a 3:1 over-subscribed network.

The rest of the paper is organized as follows: Section II reviews some previous related work; Section III demonstrates the over-subscription challenge by evaluating current state-of-the-art schemes in an over-subscribed network topology; Section IV presents the architecture and design of CAPTON; Section V evaluates CAPTON's performance and compares it to current state-of-the-art; and Section VI summarizes the study and outlines some future work.

II. RELATED WORK

We review some previous related work in this section by dividing them into distributed and centralized approaches.

A. DISTRIBUTED APPROACHES

In distributed approaches, hosts perform traffic control independently based on either implicit or explicit feedback from the network. Early work focused on a unique problem in datacenter – the *in-cast* problem where large number of senders transmit data to the same receiver simultaneously [4], [10]. This transmission pattern is a result of common datacenter workloads such as map-reduce [27] and web search [4]. In these scenarios, the aggregate data rate of multiple data flows could cause serious congestion at the network links leading to the receiver.

The best-known work to tackle the in-cast problem is the Datacenter TCP (DCTCP) proposed by Alizadeh *et al.* [4]. DCTCP makes use of the ECN feature in supported network switches to detect and response to congestion more quickly than TCP's built-in congestion control algorithm. In another approach, Wu *et al.* developed ICTCP [10] to dynamically adjust the TCP receive window according to the estimated bandwidth left on the receiving interface to limit the transmission window of the sender to avoid congestion.

Beyond the in-cast problem, another group of previous work focused on optimizing FCT and related performance metrics. For example, PDQ [9] adopted a distributed preemptive scheduling approach to schedule flows according to their level of criticalness. Compared to ordinary TCP, PDQ could achieve shorter FCT and meet the flows' deadlines.

In another work, Alizadeh *et al.* developed the well-known Pfabric [1] scheme which implemented the Ideal algorithm [1] that approximates the Shortest Remaining Processing Time First (SRPT) scheduling discipline using priority queueing at network switches. Bar-Noy *et al.* [20] subsequently proved that Pfabric can achieve FCT performance to within 2 times the optimal. However, Pfabric is difficult to deploy as it requires network switches with an infinite number of priority levels. Nevertheless, due to its proven $2\times$-optimal property, Pfabric is often used as the benchmark for comparison.

Subsequent studies had further employed the SRPT principle, e.g., Munir *et al.* proposed PASE [8] which adopted SRPT and the distributed switch-based flow scheduling approach in PDQ. In PASE, each link in the datacenter has an arbiter to perform priority scheduling, and an arbitration message from the end host traverses all arbiters along the path to obtain scheduling decisions before data transmission begins. PASE was shown to outperform even Pfabric in simulations. However, it requires an arbiter for each link and custom processing in network switches, making deployment more challenging.

In contrast, Montazeri *et al.* proposed Homa [3], which does not require separate per-link arbiters by extending the end-host-based scheduling approach in pHost [5]. It also adopted SRPT as the approximation target and employed an RPC-based protocol to coordinate flow scheduling entirely between end hosts. This is possible because Homa assumes that the host access links are the only bottlenecks, i.e., the big-switch model, where their utilizations are known to the hosts.

In another approach, Bai *et al.* developed a more readily deployable scheme called PIAS [2], which employed priority queues available in existing switches to implement a Multiple Level Feedback Queue (MLFQ) mechanism to approximate the Least Attained Service First (LAS) discipline. Unlike SRPT, LAS does not require prior knowledge of the flow size before transmission takes place and is thus easier to deploy. The tradeoff is potentially lower FCT performance compared to Pfabric due to the choice of LAS vs SRPT.

TABLE 1. Average FCT (ms) For Existing Schemes With and Without Core Network Oversubscription.

	Pfabric		PIAS		Hyline	
Access:Core Ratio	1:1	3:1	1:1	3:1	1:1	3:1
Overall	3.52	14.6	4.51	10.9	2.76	27.6
(0~100KB]	0.113	0.118	0.162	0.206	0.121	0.132
(100KB~10MB]	4.34	6.77	7.04	15.2	3.44	16.3
10MB~	56.1	403	49.1	150	42.6	750

B. CENTRALIZED APPROACHES

There are several different centralized approaches targeted at different problems in datacenter networks. For example, routing-specific approaches such as Hedera [11] collect switch statistics to find elephant flows to re-route the traffic to achieve load balance and to avoid potential congestion. Rate-specific approaches such as SDTCP [12] employed Software Defined Networking (SDN) where SDN-enabled switches report congestion status back to the controller via OpenFlow [19] so that the controller can regulate flows to avoid congestion.

In another study, Perry *et al.* proposed a fine-grained scheduler called Fastpass [7] where a centralized arbiter performs packet-level scheduling and routing to achieve zero queuing in the network fabric. The tradeoff is higher complexity due to the packet-level scheduling/routing processing required [6]. In a follow-up work, Perry *et al.* proposed Flowtune [12] where individual flow is sub-divided into flowlets. The central controller then schedules flowlets instead of individual packets to reduce its complexity. Flowlet pre-calculates routes and employs the network utility maximization (NUM) framework developed by Nagaraj *et al.* [6] to optimize the congestion control parameters according to various utility functions.

In a recent work, Abbasloo *et al.* proposed Hyline [14] which employed a central controller to schedule and route flows to optimize FCT performance. The Hyline scheduler approximates SRPT by taking into link capacity constraints in routing and scheduling individual flows. It also adopted a scheduling threshold so that flows smaller than it can transmit without incurring the scheduling delay.

In addition to FCT which measures the performance of individual flows, another class of studies – co-flow [15]–[18], focused on the performance of not one, but a group of flows that need to be all completed before processing can proceed further. Not surprisingly, the resultant routing and scheduling problem is far more complex, but it is orthogonal to this work. The principles adopted in the proposed CAPTON could be further extended to support co-flow scheduling and is a subject for future work.

The above-related works have all significantly advanced the performance of datacenter networks. Nevertheless, the commonly employed big-switch model poses a significant challenge to their real-world deployment as datacenter networks are more likely to be over-subscribed due to cost and scalability considerations. As over-subscription breaks the assumption of access-link-only bottlenecks, the existing schemes' performance in such environments is far from well-understood. We address this open problem in the next section.

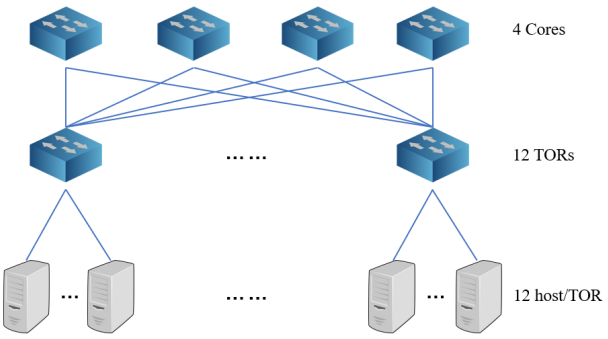

FIGURE 2. Topologies used in our experiments.

III. IMPACT OF CORE OVER-SUBSCRIPTION

In this section, we first investigate the impact of core network over-subscription on three state-of-the-art DCN traffic control schemes, namely Pfabric [1], PIAS [2], and Hyline [14]. Fig. 2 depicts the DCN topology employed – a common 2-stage leaf-spine topology with 144 hosts (12 per rack), 12 TOR switches, and 4 core switches. All access links are 10 Gbps, totaling 120 Gbps per rack. By configuring the core links between 10 Gbps and 30 Gbps, we can then setup an access-to-core bandwidth ratio of 3:1 (120 Gbps access to 40 Gbps core) and 1:1 (120 Gbps access to 120 Gbps core) respectively.

We adopted the websearch workload [27] where 11/12 of the traffics are inter-rack that need to traverse the network core. The offered total traffic load was set to 32.76% of the total access link bandwidth, resulting in an inter-rack traffic load close to 30%. We recorded the FCT of all flows completed and summarized the average FCT for all three schemes in Table 1.

Without core network over-subscription, this level of traffic load is well within the system's capacity so the FCT performances of all three schemes are excellent, with Hyline achieving the shortest average FCT, followed by Pfabric and PIAS. By contrast, once we switched to the 3:1 over-subscribed network setup, all three algorithms' FCT performance degraded significantly. Interestingly, the three schemes' relative performances are now reversed, with PIAS achieving the shortest average FCT, followed by Pfabric and Hyline.

Generally, the performance degradation increased with larger flow size in the over-subscribed setup as shorter flows were scheduled with higher priority. Longer flows with lower priority thus suffered more from the congestions in the network core. Overall, the average FCT of Pfabric, PIAS, and Hyline were increased by 315%, 142%, and 900% respectively under 3:1 over-subscription.

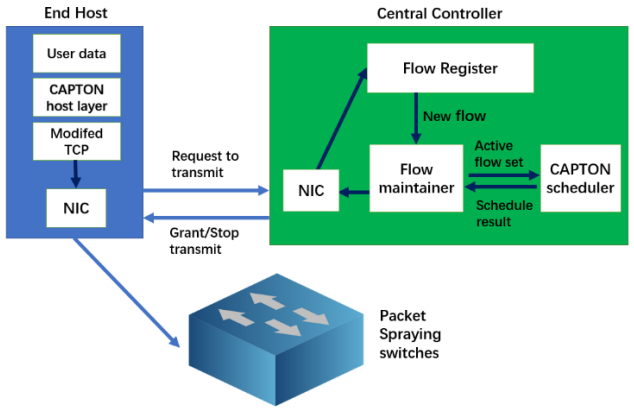

FIGURE 3. System architecture and design.

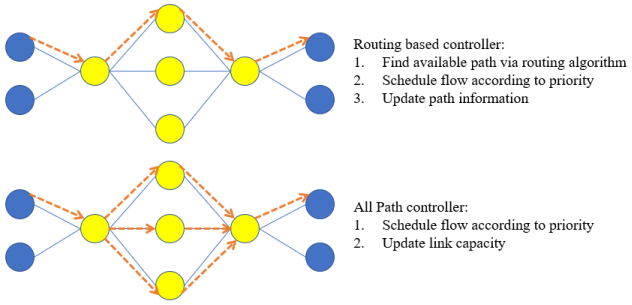

FIGURE 4. Difference between all-path controller and routing-based controller transmission.

It is worth noting that even at 3:1 oversubscription, the core network is running at 90% average utilization which is still *within* the core's capacity. However, randomness in the traffics means that the short-term load could momentarily exceed the core capacity, leading to packet losses and timeouts. The above results clearly demonstrate that core network over-subscription could not be ignored and should be incorporated into the traffic control scheme.

IV. ARCHITECTURE AND SYSTEM DESIGN

In this section, we first present CAPTON's design principles and then delve into the details in the sub-sections. There are four key design principles in CAPTON:

- It employs a central controller to schedule flows subject to access and core link capacity constraints to prevent congestion in both access and core links. This addresses the core network over-subscription problem discussed earlier.
- It adopts an all-path transmission scheme by sending packets of each flow over all available equal-cost shortest paths simultaneously. This effectively eliminates the need for dynamic routing and path selection, as the available paths between all source-destination pairs are known and can be pre-calculated. This not only enables utilization of all available links but also eliminates the complexity and scalability challenges in dynamically routing a large number of flows in a datacenter.
- To reduce the impact of scheduling delay, CAPTON implements a scheduling threshold to allow flows smaller than a pre-set size to bypass the central scheduler to transmit immediately. Longer flows are still scheduled by the central controller so that congestions are kept at a low level.
- CAPTON is designed to make use of dumb switches, without the need for any advanced features such as ECN, priority queueing, or even dynamic forwarding table updates. This not only reduces the network switch costs but also allows CAPTON to be deployed in a wide range of current and future datacenter networks.

A. SYSTEM OPERATION

Fig. 3 depicts the architecture of CAPTON, the design of the end host and the central controller. When an application creates a new flow, it is intercepted by the CAPTON host layer sitting in-between the application and the underlying transport. The host layer then checks if the data size exceeds the pre-defined scheduling threshold, and if so, sends a request to the central controller to schedule transmission. Otherwise, it proceeds to transmit the flow immediately without involving the central controller.

For unscheduled flows, the host always transmits data at full line rate. In contrast, the transmission rate for scheduled flows is set by the central controller. In both cases, data are sent over all available links including redundant links so as to maximize throughput to reduce FCT. To support this, the host's TCP layer is modified to allow transmission at a controlled data rate. Moreover, as opposed to conventional TCP operations where multiple TCP flows may transmit simultaneously and share the host's link bandwidth, CAPTON effectively *serializes* TCP flows within a host so that only one flow is transmitting at any given time. This contributes to approximating the SRPT scheduling discipline [32].

The central controller keeps track of all active flows and schedules the start and stop of flow transmissions by sending GRANT/STOP messages to the host's CAPTON layer, taking into consideration the set of active flows and their remaining flow size, the size of the new flow, access and core link capacities as well as their current utilization. Note that the central controller does not need to perform routing due to the use of all-path transmission, which reduces its complexity considerably.

B. ALL-PATH TRANSMISSION

The goal of all-path transmission is to make full use of *all* alternative equal-cost paths between the source and the destination for transmitting data of a flow. Datacenter ToR switches are typically inter-connected via multiple core switches for redundancy as well as for capacity scaling. In routing approaches [14], [28], one of the alternative paths between the source and destination will be chosen for a flow. This has two potential limitations. First, this single-path approach limits the per-flow bandwidth to that of a single core

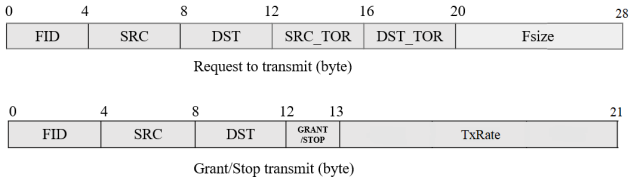

FIGURE 5. Control messages.

link. Although core links typically have higher bandwidth, their available bandwidth will be reduced during demand peaks. Splitting a flow across multiple core links allows the available bandwidth from multiple core links to be aggregated for potentially higher transfer rate for shorter FCT.

Second, implementing dynamic flow-level routing is a non-trivial problem in its own right. In addition to the obvious overheads and potential scalability challenge in computing routes for all the flows in the network, to implement the routes the controller will need to send forwarding table updates to all switches along the path and wait for them to complete the update before the flow can begin transmission. The delay in updating switch forwarding tables is often ignored in previous studies but its magnitude could be significant (e.g., in ms [33]) compared to the flow duration.

Therefore, we propose all-path transmission as depicted in Fig. 4 to split the data of each and every flows across all equal-cost paths to maximize the utilization of all available access and core links. As a result, switch forwarding tables entries for all destinations could be pre-computed offline, eliminating the need for and delay in switch forwarding table updates. All-path transmission in the core switches can be implemented using per-destination round-robin packet spraying which is widely supported in off-the-shelf switches [21], [27].

A potential issue in all-path transmission is packet reordering as most flows are transported over TCP where out-of-order packet arrivals could trigger spurious retransmissions. However, this problem is more severe in cases where the alternative paths have unequal costs (e.g., delay, bandwidth). Datacenter networks are typically far more symmetric such that alternative paths have similar, if not the same, delay and bandwidth [21]. Moreover, CAPTON's all-path transmission scheme further ensures that the loads on alternative paths are the same so that even queueing delays will be similar. Therefore, packet reordering is negligible in CAPTON. In fact, we did not observe *any* spurious retransmission due to packet reordering in all of our experiments in Section V.

C. END HOSTS

The end host implements two CAPTON components. First, the CAPTON host layer is responsible for intercepting flow data from applications and communicating with the central controller to schedule their transmission. In the Linux operating system, this can be implemented using API hooks on the socket's library so that no modification of the application is needed - an important, if not essential, feature for practical deployment. Most applications perform computation and then generate data in a burst for transmission. The CAPTON host layer first buffers the data to determine the flow size. If it is equal to or smaller than the preset scheduling threshold then it simply passes the data on to the transport layer for transmission. Otherwise, it sends a Request-to-Transmit (RT) message, as depicted in Fig. 5, to the central controller which contains the flow ID (FID), the source/destination switch port, source/destination rack, and flow size for the flow.

The central controller then schedules a time and data rate for the host to begin transmitting data for the flow. Note that the central controller does not send to the host the scheduled time to begin transmission. Instead, at the scheduled time, it sends a Grant message to the host to start the transmission. This approach avoids the need to synchronize the clocks of the hosts with the central controller which is a non-trivial problem in its own right. The tradeoff is a slight delay in delivering the Grant message, which is in the order of 100 us in datacenters (see Section V). Conceivably, the delay could be compensated for in the schedule as the path's propagation delay is known but our results show that even with the delay, CAPTON still outperforms existing schemes over a wide range of setups.

In addition to granting transmission, the central controller could also suspend an ongoing transmission by sending the host a Stop message for the flow. This may occur when a new flow enters the system with a higher priority than the ongoing flow (see Section IV-D), thus pre-empting the lower priority flow.

The second CAPTON component in the host is the TCP module. Technically, it is not a newly created module but a modified version of the existing TCP module. For example, in Linux, TCP is implemented as a pluggable congestion control module that can be loaded/unloaded at runtime as there are multiple TCP variants to choose from. CAPTON requires modifications to the chosen TCP variant, e.g., the default TCP variant known as Cubic [31], to control transmission at the data rate set by the controller.

One way to implement this modification is to override TCP's transmission window by the maximum value allowed[1] so that TCP's Slow-Start and AWnd limit are both bypassed. Outgoing packets' transmission rate can then simply be implemented using Linux's built-in pacing mechanism.

D. CENTRAL CONTROLLER

The central controller is the brain of the CAPTON platform as it keeps track of the network topology and orchestrates all data flows' transmissions to prevent congestion and to improve FCT performance. It consists of a flow register for processing new flows, a flow maintainer for maintaining the active flow list, and a scheduling module running the CAPTON scheduler to calculate the transmission schedule

[1]Linux kernel stores transmission window size as a 32-bit unsigned integer which has a maximum value of 4,294,967,295.

```
# CAPTON Scheduler #
N = number of hosts
F = scheduled flow set
Flow data structure: {
   .size;    // total flow size.
   .left;    // remaining flow size.
   .grant;   // 0=stop; 1=transmit.
   .rate;    // assigned transmission rate.
   .end;     // completion time.
   .sal_id;  // source access link ID.
   .dal_id;  // destination access link ID.
   .scl_id;  // source core link ID.
   .dcl_id;  // destination core link ID.
   .stor_id; // source top-of-rack switch ID.
   .dtor_id; // destination top-of-rack switch ID.
} f ∈ F;
A = set of access links.
C = set of core links.
Link data structure: {
   .cap;   // link capacity.
   .inuse; // current capacity in use.
} al ∈ A, cl ∈ C;

 Initialization:
 F.grant = 0;
 A.inuse = 0;
 C.inuse = 0;
 n = 0;
 Sort F in increasing order of remaining flow size.

1.  for each flow f in F:
2.     if n == N
3.        return F
4.     end if
5.     if A[f.sal_id].inuse == 0 and A[f.dal_id].inuse == 0:
6.        f.rate = min(A[f.sal_id].cap, A[f.dal_id].cap)
7.        if f.stor_id = f.dtor_id:  //intra rack traffic
8.           f.grant = 1
9.           A[f.sal_id].inuse = f.rate
10.          A[f.dal_id].inuse = f.rate
11.          n = n + 1
12.       else:  //inter-rack traffic
13.          if ((C[f.scl_id].inuse + f.rate)
14.              <= C[f.scl_id].cap) and
15.             ((C[f.dcl_id].inuse + f.rate)
16.              <= C[f.dcl_id].cap)):
17.             f.grant = 1
18.             A[f.sal_id].inuse = f.rate
19.             A[f.dal_id].inuse = f.rate
20.             C[f.scl_id].inuse += f.rate
21.             C[f.dcl_id].inuse += f.rate
22.             n = n + 1
23.          end if
24.       end if
25.    end if
26. end for
27. return
```

FIGURE 6. Pseudocodes for the CAPTON Scheduler.

for all data flows. In this section, we first discuss the principle of the CAPTON scheduling algorithm, and then explain how the CAPTON scheduler coordinates with other components inside the central controller.

The CAPTON scheduler is a centralized, flow-based scheduler that aims at approximating the SRPT discipline. Fig. 6 lists the pseudocodes for the CAPTON scheduler. The inputs are the set of flows to be scheduled, and the output is the transmission scheduled for the flows, including flows permitted to transmit at a set data rate and flows to be suspended.

The algorithm begins by initializing all flow states to be suspended and all link states to be unused. Note that due

FIGURE 7. CDF of workload size (in packets) used in the experiments.

to all-path transmission, multiple access and core links are simply aggregated into a virtual link in the algorithm. Next, it updates the remaining flow size, denoted by *f.size*, of all flows transmitting since the last iteration *i*-1 by

$$f.size = f.size - f.rate \cdot (T_{cur} - T_{i-1}) \tag{1}$$

where *f.rate* is the transmission rate assigned in the previous iteration *i*-1, T_{cur} and T_{i-1} are the current time and past execution time of iteration *i*-1, respectively.

It then iterates through each flow in increasing order of their remaining flow size. It first checks if both the source and the destination access links have bandwidth available to accommodate the flow. Note that CAPTON allows only one flow to transmit/receive at a time even if there is more than one flow pending at the host. However, a host could transmit and receive simultaneously as links are full-duplex.

Next, it checks if the flow is intra-rack which does not need to traverse core links and so can be marked to transmit. Otherwise it checks if the flow's data rate can be accommodated by all the core links along the path, and if so, marks the flow to transmit. The algorithm terminates when all flows have been processed or the number of flows marked to transmit equals to the number of hosts in the network.

The CAPTON algorithm is executed each time a Request-to-Transmit message is received or when a flow completes transmission. For flows that are scheduled to transmit, their completion time, denoted by *f.end* can be calculated from their remaining flow size *f.size* divided by the assigned transmission rate *f.rate*:

$$f.end = \frac{f.left}{f.rate} \tag{2}$$

Thus the earliest completion time among all flows determines the latest time, denoted by T_{i+1}, for the next iteration $i+1$ of the CAPTON algorithm to execute:

$$T_{i+1} = T_{cur} + \min\{f.end | \forall f \in F\} \tag{3}$$

where T_{cur} is the current time and F is the scheduled flows set. Note that if a new Request-to-Transmit request arrives before T_{next} then the scheduler will run immediately.

TABLE 2. Ratio of bytes retransmitted for websearch.

Load	Pfabric	PIAS	Hyline_Access	Hyline_Core	CAPTON
18.2%	0.67%	0.0073%	0.0516%	0.0231%	0
21.8%	0.82%	0.0068%	0.0691%	0.026%	0
25.5%	0.93%	0.006%	0.059%	0.028%	0
29.1%	1.1%	0.0052%	0.0697%	0.0861%	0
32.7%	3%	0.004%	0.0832%	0.028%	0
36.4%	5.5%	0.004%	0.0781%	0.0342%	0

TABLE 3. Simulation parameters.

	Pfabric	PIAS	Hyline	CAPTON
RTT (µs)	85.2	85.2	85.2	85.2
Buffer	2 BDP	2 BDP	2 BDP	2 BDP
RTO	2×RTT	200 ms	4 ms	200 ms
Routing	ECMP	ECMP	ECMP	All-Path Tx

E. SCHEDULING THRESHOLD

An inherent tradeoff in centralized flow scheduling is the additional delay incurred in sending a request to the controller and waiting for the response before transmission can take place. Datacenter networks typically have very short propagation delay, e.g., RTT within 100 us [13], but nonetheless could be significant for short flows. For example, a flow of 10 MTUs (i.e., 15 KB) in theory can be transmitted in a mere 1.2 us over a 10-Gbps link so the additional delay incurred in centralized scheduling is significant for very short flows.

One approach to tackle this problem is to relax the scheduling requirement by allowing flows smaller than a given size – *scheduling threshold*, to transmit right away without being scheduled. The intuition is that given the small flow size, it is unlikely to cause congestion at the switches even if the flow collides with another flow. This technique has also been employed in other centralized schedulers such as Hyline [14].

The challenge is in determining the scheduling threshold to be used. Too small a threshold then few flows could benefit from the delay reduction. Too large a threshold then the unscheduled flows could cause congestion. The optimal scheduling threshold depends on a multitude of factors, including workload characteristics (e.g., flow size and arrival pattern), network topology, traffic load, and even the scheduling algorithm adopted for the scheduled flows. This is clearly a non-trivial problem and previous work such as Abbasloo *et al.* [14] has proposed mathematical models to optimize the scheduling threshold.

Nevertheless, our experiments show that existing model (e.g., [28]) for optimizing the scheduling threshold could be too aggressive in networks with core over-subscription. Therefore, based on our experimental results, we adopted a simple fixed scheduling threshold of 10 MSS's. Despite its non-adaptive nature, we found that the fixed scheduling threshold works well across a wide range of setups (see Section V), thereby simplifying CAPTON's implementation and deployment significantly.

V. PERFORMANCE EVALUATION

In this section, we report results obtained from simulated experiments to evaluate the performance of CAPTON and compare it to three state-of-the-art schemes, namely Pfabric [1] - a widely used benchmark, PIAS [2] – a distributed approach, and Hyline [14] – a centralized approach.

We implemented CAPTON in NS2 [25] and employed the original NS2 implementations from Pfabric [1], PIAS [26], and Hyline [29] for comparisons. Thus all four approaches were simulated using the same simulator (NS2) to improve the performance results' consistency and comparability.

The datacenter network simulated is a leaf-spine topology with 144 hosts, 12 TORs, and 4 core switches, with 12 hosts under each TOR switch, resulting in an over-subscription ratio of 3:1. All links in the topology are 10-Gbps links. The end-to-end RTT between a source-destination pair across the core fabric is 85.2 us [2]. For all schemes compared, per-flow-based ECMP was adopted as the routing strategy except for CAPTON where routing is not needed due to its all-path transmission scheme.

The buffer size for all links was set to 2 BDP, The RTO for Pfabric was set to 3 × RTT as suggested in [1], the ones for PIAS and CAPTON were 200 ms (the simulators' default), and 4 ms for Hyline as suggested in [14]. For centralized schemes (i.e., Hyline and CAPTON), we accounted for scheduling delays by including the time required for the host to communicate with the central controller before the flow can start its transmission.

The scheduling threshold for Hyline was chosen according to the model proposed in [28]. A key parameter in optimizing the threshold is the *network load*. As core over-subscription was not considered in the original Hyline study, its network load is equivalent to both access-link load and core-link load. However, this is no longer true in over-subscribed networks so the choice of access versus core load as the network load parameter will lead to different scheduling thresholds.

To cover both cases, we ran two sets of simulations for Hyline in each experiment, one using scheduling threshold calculated from access load (denoted by Hyline_Access) and the other using scheduling threshold calculated from core load (denoted by Hyline_Core) as summarized in Table 4. By contrast, CAPTON employs a fixed scheduling threshold of 10 MSS's for all experiments.

We adopted two datacenter workloads widely used by previous works to generate the flow traces: the websearch workload [4] and the datamining workload [27]. Fig. 7 plots their cumulative distributions, showing that the datamining workload have considerably more short flows. In the experiments, we allow all flows to run until completion. Finally, we excluded delays due to switch forwarding table updates which are needed in routing-based schemes such as Hyline.

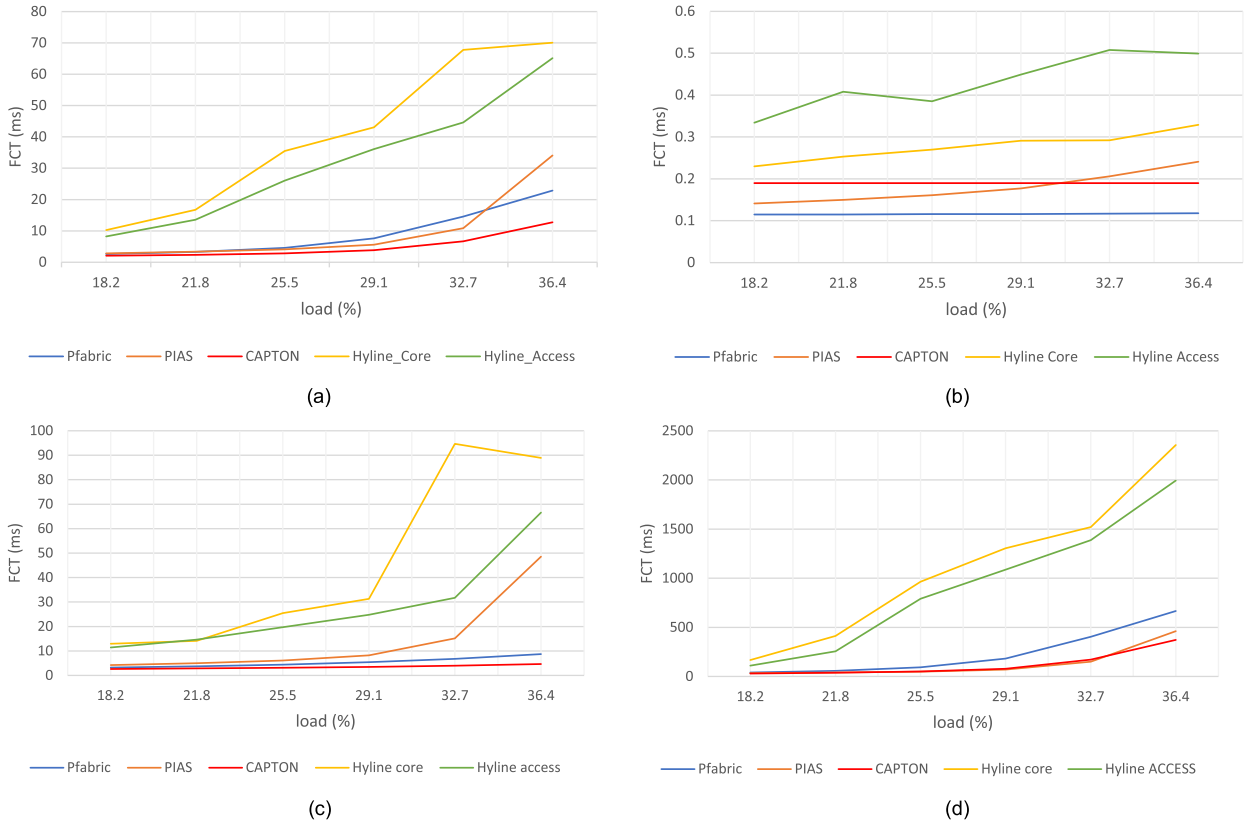

FIGURE 8. Average FCT (ms) of Pfabric PIAS, Hyline and CAPTON in websearch workload for a) All flows, b) flows in (0∼100 KB], c) flows in (100 KB∼10 MB], d) flows in (10 MB∼).

TABLE 4. Scheduling threshold for hyline [28].

Access Load (%)	Core Load (%)	Websearch (By Access,MB)	Datamining (By Access,MB)	Websearch (By Core,MB)	Datamining (By Core,MB)
18.2	50	9.18	35.1	5.72	20.9
21.8	60	7.45	28.05	5.2	18.82
25.5	70	7.45	28.05	4.8	17.21
29.1	80	7.45	28.05	4.47	15.93
32.7	90	6.42	23.77	4.2	14.88
36.4	100	6.42	23.77	4.2	14.88

A. WEBSEARCH WORKLOAD

In the first experiment, we generated a trace of 100,000 flows using the Websearch workload over a range of access loads from 18.2% to 36.4%, resulting in corresponding core loads of 50% to 100%.

We compare the performance of Pfabric, PIAS, Hyline, and CAPTON in terms of average FCT. Fig. 8a shows the overall average FCTs for all load scenarios. The result shows that CAPTON achieved the shortest overall FCT performance across all loads tested. Both Pfabric and PIAS performed closely at lighter loads (e.g., 18.2%) but the gap widens as load increases. For example, at access load of 29.1%, Pfabric's mean FCT at 7.61 ms is already 99% higher than CAPTON at 3.82 ms. Both Hyline setups exhibited substantially higher FCT. At the 29.1% access load, Hyline_Core and Hyline_Access achieved FCTs of 43.06 ms and 36.122 ms, which are 1127% and 945% higher than CAPTON.

Next, we divide flows into three groups based on flow size, i.e., small (0∼100 KB], medium (100 KB ∼ 10 MB], and large (10 MB or longer) which reveal the different tradeoffs attained by the four algorithms. There are three observations. First, Pfabric achieved the best FCT for small flows (Fig. 8b). This is to be expected as, by design, Pfabric assigns higher priority to shorter flows. Its performance at medium (Fig. 8c) and long flows (Fig. 8d), while not the best, is still very good. This shows why Pfabric is often employed as the benchmark

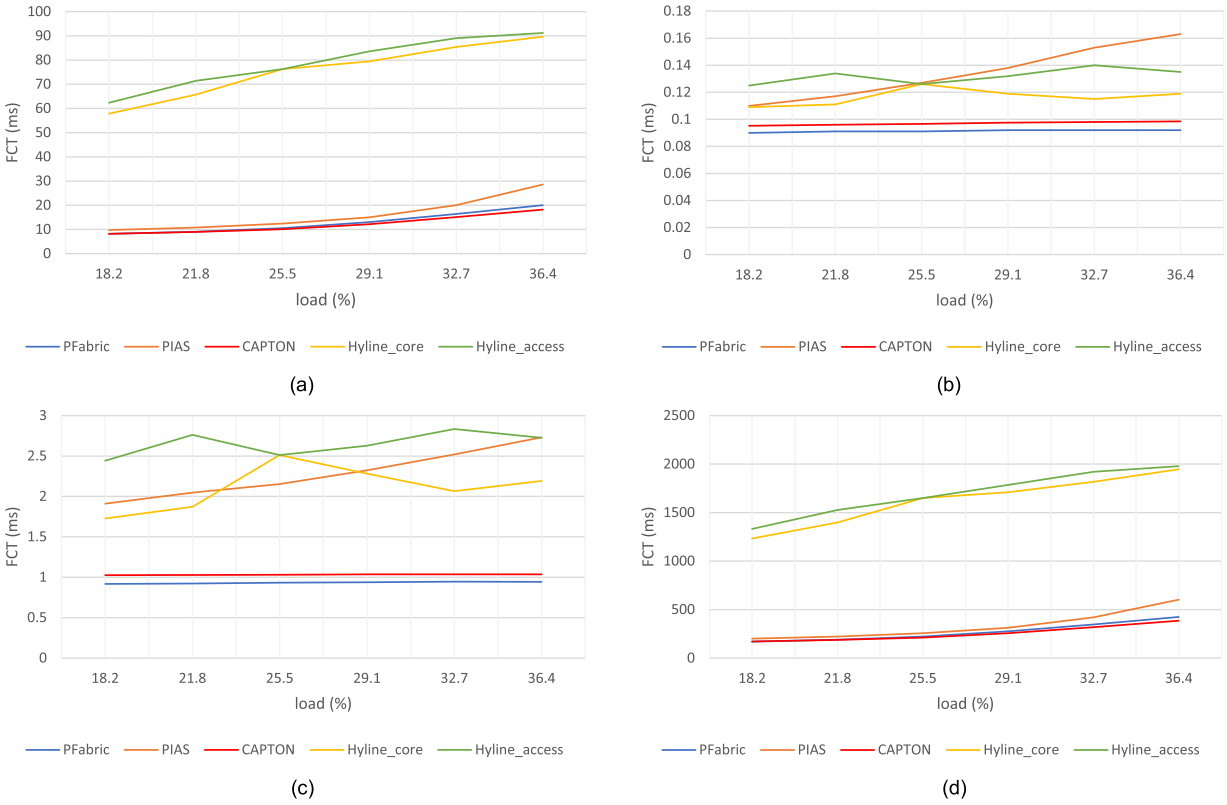

FIGURE 9. Average FCT (ms) of Pfabric PIAS, Hyline and CAPTON in datamining workload for a) All flows, b) flows in (0∼100 KB], c) flows in (100KB∼10 MB], d) flows in (10 MB∼).

for performance comparison, even though it was not designed for practical deployment.

Second, PIAS also performed very well, except at higher loads as shown in Fig. 8a. The results in Fig. 8b to 8c reveal that the primary contributor to PIAS's higher FCT at high loads is the substantial FCT increases in medium-sized flows (Fig. 8c).

Third, Hyline exhibited substantially higher FCTs in all flow sizes. The performance is noticeably lower than the results in its original studies [14], [28]. The primary difference for the experiments was the 3:1 over-subscription in the core network where Hyline did not account for. In particular, we observe that Hyline's model sets the scheduling threshold to a relatively large value (c.f. Table 4). As a result, over 87% (Hyline_Access) and 90% (Hyline_Core) of the flows ended up transmitting unscheduled (see Table 5). In comparison, CAPTON's fixed scheduling thresholds of 10 MSS's resulted in only 17.9% of the flows transmitting unscheduled.

The large amount of unscheduled flows in Hyline reduced its central controller's ability to schedule and route flows to prevent congestion. This results in congestion inside the core network which not only triggers retransmissions but also timeouts in more severe cases. Therefore, further investigations are needed to explore if and how Hyline could be extended to support over-subscribed network topologies.

TABLE 5. Unscheduled flow % for websearch.

Load	Hyline_Access	Hyline_Core	CAPTON
18.2%	92.8%	89.5%	17.9%
21.8%	91.4%	88.9%	17.9%
25.5%	91.4%	88.3%	17.9%
29.1%	91.4%	87.8%	17.9%
32.7%	90.4%	87.3%	17.9%
36.4%	90.4%	87.3%	17.9%

TABLE 6. Unscheduled flow % for datamining.

Load	Hyline_Access	Hyline_Core	CAPTON
18.2%	97.9%	97.4%	80.2
21.8%	97.7%	97.3%	80.2
25.5%	97.7%	97.2%	80.2
29.1%	97.7%	97.1%	80.2
32.7%	97.5%	97%	80.2
36.4%	97.5%	97%	80.2

By contrast, CAPTON exhibited zero retransmission in all test cases because it schedules all flows longer than 10 MSS's, and the unscheduled short flows are sufficiently short to not cause congestions at the core even under the highest load tested.

There are also tradeoffs in CAPTON. Flows longer than the scheduling thresholds will incur additional scheduling delay. This is more significant in short flows as shown in Fig. 8b where CAPTON's FCT is higher than Pfabric precisely by the extra scheduling delay. PIAS also achieved lower FCT than CAPTON for short flows until the load exceeded 30.9%.

In contrast, CAPTON's gains in FCT reduction can more than compensate for the additional scheduling delay in medium and long flows as shown in Fig. 8c and 8d. Thus the workload characteristics, in particular, the proportion of short flows, will impact the relative performance of different algorithms. We further investigate this using the datamining workload in the next section.

B. DATAMINING WORKLOAD

In this section, we investigate the algorithms' performance under the datamining workload. Compared to the websearch workload, the datamining workload has a larger proportion of small flows as shown in Fig. 7. In fact, more than 80% of the flows are smaller than 10 MSS's. For CAPTON this means over 80% of the flows will *not* be scheduled by the CAPTON scheduler. Thus, it will be interesting to see how CAPTON performs under such challenging workloads.

Fig. 9a to 9d plot and compare the algorithms' FCT performances. In terms of overall FCT, CAPTON continued to achieve the lowest FCT over the range of load tested. The gap between CAPTON and the benchmark Pfabric considerably narrowed compared to the same under the websearch workload. This is because the datamining workload's heavy proportion of short flows - in particular, flows shorter than CAPTON's scheduling threshold, reduced the among of traffics being scheduled, thereby forgoing the SRTF optimization of the CAPTON Scheduler.

For small flows in Fig. 9b, CAPTON's FCT performance in fact improved over the websearch workload. This somewhat counterintuitive result is due to the increased proportion of unscheduled flows under the datamining workload, which did not incur extra scheduling delays. Pfabric still performed better though as it employs priority queueing for different flow sizes.

For medium flows in Fig. 9c, CAPTON now underperformed Pfabric slightly, due to the increased queueing delay caused by the larger number of unscheduled flows. Nevertheless, CAPTON achieved substantially lower FCT than PIAS and Hyline even under the lightest load of 18.2%.

Finally, for large flows in Fig. 9d, CAPTON outperformed the other algorithms, with Pfabric being a close second, followed by PIAS. Hyline's FCTs are significantly higher across all load levels. This is not surprising as, under the datamining threshold, over 97% of the flows were smaller than its scheduling threshold and were thus not being scheduled at all.

C. SCHEDULING DELAY

Centralized approaches such as CAPTON and Hyline require host to send their flow information to the controller for scheduling purposes so they will incur extra scheduling delay as discussed earlier. For CAPTON the scheduling delay has three components: (a) the transmission time for the control messages; (b) the propagation delay between the host and the CAPTON controller; and (c) the execution time of the CAPTON Scheduler. We provide an estimate of the scheduling delay below.

The transmission time for control message of size 68 Byte (inclusive of Ethernet header of 40 bytes) over a 10-Gbps link is 0.0544 us. The round-trip propagation delay was set to 85.2 us following [2]. The CAPTON Scheduler runtime can vary, depending on the topology, the set of flows to be scheduled, and the hardware it runs on.

To obtain an estimate of the runtime, we sampled the actual runtime of the CAPTON Scheduler executed on one CPU core of the Intel I7-6700 CPU, during the websearch experiments under an access load of 36.4%. The samples showed a maximum of 640 active flows, resulting in a runtime up to 10 us. Thus, altogether the scheduling delay is 95.2 us. This is also close to the assumption (100 us) in the Hyline study [14]. Therefore, we simply adopted 100 us as the scheduling delay in simulating both Hyline and CAPTON.

VI. SUMMARY AND FUTURE WORK

This work demonstrated the impact of over-subscription in datacenter networks on the performance of traffic control schemes. It is clear that over-subscription must be accounted for and incorporated into the traffic control scheme to achieve consistent performance in practical datacenter networks. The proposed CAPTON scheme is relatively simple to deploy, due to the use of dumb switches and the elimination of routing altogether and performed consistently well under the wide range of scenarios tested.

This work is only the first step though as it shows that existing traffic control schemes should be revisited in light of over-subscription. This not only applies to centralized schemes, but also distributed schemes as well. More work is thus warranted to investigate this new line of research.

ACKNOWLEDGMENT

The authors wish to thank the Associate Editor and the anonymous reviewers for their suggestions in improving this article to its final form.

REFERENCES

[1] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker, "PFabric: Minimal near-optimal datacenter transport," in *Proc. ACM SIGCOMM Conf. (SIGCOMM)*, Aug. 2013, pp. 435–446.

[2] W. Bai, L. Chen, K. Chen, D. Han, C. Tian, and H. Wang, "PIAS: Practical information-agnostic flow scheduling for commodity data centers," *IEEE/ACM Trans. Netw.*, vol. 25, no. 4, pp. 1954–1967, Aug. 2017.

[3] B. Montazeri, Y. Li, M. Alizadeh, and J. Ousterhout, "Homa: A receiver-driven low-latency transport protocol using network priorities," in *Proc. Conf. ACM Special Interest Group Data Commun.*, Aug. 2018, pp. 221–235.

[4] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data center TCP (DCTCP)," in *Proc. ACM SIGCOMM Conf. SIGCOMM (SIGCOMM)*, 2010, pp. 63–74.

[5] P. X. Gao, A. Narayan, G. Kumar, R. Agarwal, S. Ratnasamy, and S. Shenker, "PHost: Distributed near-optimal datacenter transport over commodity network fabric," in *Proc. 11th ACM Conf. Emerg. Netw. Exp. Technol.*, Dec. 2015, pp. 1–12.

[6] K. Nagaraj, D. Bharadia, H. Mao, S. Chinchali, M. Alizadeh, and S. Katti, "NUMFabric: Fast and flexible bandwidth allocation in datacenters," in *Proc. ACM SIGCOMM Conf.*, Aug. 2016, pp. 188–201.
[7] J. Perry, A. Ousterhout, H. Balakrishnan, D. Shah, and H. Fugal, "Fastpass: A centralized zero-queue 'datacenter network,'" in *Proc. ACM Conf. SIGCOMM*. New York, NY, USA, Aug. 2014, pp. 307–318.
[8] A. Munir, G. Baig, S. M. Irteza, I. A. Qazi, A. X. Liu, and F. R. Dogar, "Friends, not foes: Synthesizing existing transport strategies for data center networks," in *Proc. ACM Conf. SIGCOMM*, Aug. 2014, pp. 491–502.
[9] C.-Y. Hong, M. Caesar, and P. Godfrey, "Finishing flows quickly with preemptive scheduling," in *Proc. ACM Comput. Commun. Rev. (SIGCOMM)*, Aug. 2012, vol. 42, no. 4, pp. 127–138.
[10] H. Wu, Z. Feng, C. Guo, and Y. Zhang, "ICTCP: Incast congestion control for TCP in data-center networks," *IEEE/ACM Trans. Netw.*, vol. 21, no. 2, pp. 345–358, Apr. 2013.
[11] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks," in *Proc. NSDI 7th USENIX Conf. Netw. Syst. Design Implement.*, vol. 10, no. 8, pp. 89–92, Apr. 2010.
[12] J. Perry, H. Balakrishnan, and D. Shah, "Flowtune: Flowlet control for datacenter networks," *Proc. 14th USENIX Symp. Networked Syst. Design Implement.*, Boston, MA, USA, Mar. 2017, pp. 421–435.
[13] Y. Lu, Z. Ling, S. Zhu, and L. Tang, "SDTCP: Towards datacenter TCP congestion control with SDN for IoT applications," *Sensors*, vol. 17, no. 12, p. 109, Jan. 2017.
[14] S. Abbasloo, Y. Xu, and H. J. Chao, "HyLine: A simple and practical flow scheduling for commodity datacenters," in *Proc. IFIP Netw. Conf. (IFIP Netw.) Workshops*, May 2018, pp. 1–9.
[15] M. Chowdhury, Y. Zhong, and I. Stoica, "Efficient coflow scheduling with varys," in *Proc. ACM Conf. SIGCOMM*, Aug. 2014, pp. 443–454.
[16] Y. Zhao, K. Chen, W. Bai, M. Yu, C. Tian, Y. Geng, Y. Zhang, D. Li, and S. Wang, "Rapier: Integrating routing and scheduling for coflow-aware data center networks," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr. 2015, pp. 424–432.
[17] S. Agarwal, S. Rajakrishnan, A. Narayan, R. Agarwal, D. Shmoys, and A. Vahdat, "Sincronia: Near-optimal network design for coflows," in *Proc. Conf. ACM Special Interest Group Data Commun.*, Aug. 2018, pp. 16–29.
[18] L. Chen, K. Chen, W. Bai, and M. Alizadeh, "Scheduling mix-flows in commodity datacenters with karuna," in *Proc. ACM SIGCOMM Conf.*, Aug. 2016, pp. 174–187.
[19] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling innovation in campus networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Apr. 2008.
[20] A. Bar-Noy, M. M. Halldórsson, G. Kortsarz, R. Salman, and H. Shachnai, "Sum multicoloring of graphs," *J. Algorithms*, vol. 37, no. 2, pp. 422–450, Nov. 2000.
[21] A. Dixit, P. Prakash, Y. C. Hu, and R. R. Kompella, "On the impact of packet spraying in data center networks," in *Proc. IEEE INFOCOM*, Apr. 2013, pp. 2130–2138.
[22] A. Aggarwal, S. Savage, and T. Anderson, "Understanding the performance of TCP pacing," in *Proc. IEEE INFOCOM. Conf. Comput. Communications. 19th Annu. Joint Conf. IEEE Comput. Commun. Societies*, Mar. 2000, pp. 1157–1165.
[23] Y. Chen, S. Alspaugh, and R. H. Katz, "Design insights for MapReduce from diverse production workloads," UC Berkeley, Berkeley, CA, USA, Tech. Rep. EECS-2012-17. [Online]. Available: https://www2.eecs.berkeley.edu/Pubs/TechRpts/2012/EECS-2012-17.html
[24] (Jun. 2018). *Huawei DCN Design Guide*. [Online]. [Online]. Available: https://support.huawei.com/enterprise/en/doc/EDOC1100023542?section=j00o
[25] *Network Simulator 2 (NS2)*. Accessed: Jul. 31, 2020. [Online]. Available: https://www.isi.edu/nsnam/ns/
[26] *PIAS Simulation Source*. Accessed: May 22, 2021. [Online]. Available: https://github.com/HKUST-SING/PIAS-NS2
[27] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "VL2: A scalable and flexible data center network," in *Proc. ACM SIGCOMM Conf. Data Commun. (SIGCOMM)*, 2009, pp. 51–62.
[28] S. Abbasloo, Y. Xu, and H. J. Chao, "To schedule or not to schedule: When no-scheduling can beat the best-known flow scheduling algorithm in datacenter networks," *Comput. Netw.*, vol. 172, May 2020, Art. no. 107177.
[29] *Hyline NS2 Simulation Source*. Accessed: Jul. 13, 2021. [Online]. Available: https://github.com/Soheil-ab/HyLine
[30] *CISCO Link Aggregation Control Protoco*. Accessed: Jul. 31, 2020. [Online]. Available: https://www.cisco.com/c/en/us/td/docs/ios/12_2sb/feature/guide/gigeth.html
[31] S. Ha, I. Rhee, and L. Xu, "CUBIC: A new TCP-friendly high-speed TCP variant," *ACM SIGOPS Operating Syst. Rev.*, vol. 42, no. 5, pp. 64–74, Jul. 2008.
[32] H. B. Mor, S. Bianca, B. Nikhil, and A. Mukesh, "Size-based scheduling to improve web performance," *ACM Trans. Comput. Syst.*, vol. 21, no. 2, pp. 207–233, Jan. 2002.
[33] C. N. R. Sarrar, S. Uhlig, and R. A. W. S. Moore, "OFLOPS: An open framework for OpenFlow switch evaluation," in *Proc. Int. Conf. Passive Act. Netw. Meas.*, Berlin, Germany, Mar. 2012, pp. 85–95.

NAN ZHOU received the B.Eng. and M.Sc. degrees in information engineering from The Chinese University of Hong Kong, Hong Kong, in 2016 and 2017, respectively, where is currently pursuing the Ph.D. degree with the Department of Information Engineering. He has participated in the research and development of datacenter networking technology at the Department of Information Engineering, The Chinese University of Hong Kong.

JACK Y. B. LEE (Senior Member, IEEE) received the B.Eng. and Ph.D. degrees in information engineering from The Chinese University of Hong Kong, Hong Kong, in 1993 and 1997, respectively. He is currently an Associate Professor with the Department of Information Engineering, The Chinese University of Hong Kong. His research group focuses on research in multimedia communications systems, mobile communications, protocols, and applications. He specializes in tackling research challenges arising from real-world systems. He works closely with the industry to uncover new research challenges and opportunities for new services and applications. Several of the systems research from his laboratory have been adopted and deployed by the industry.