Multi-Tenant Multi-Objective Bandwidth Allocation in Datacenters Using Stacked Congestion Control

Chen Tian[†], Ali Munir[‡], Alex X. Liu[†‡], Yingtong Liu[♯], Yanzhao Li[†], Jiajun Sun[†], Fan Zhang[§], Gong Zhang[§]
[†]*State Key Laboratory for Novel Software Technology, Nanjing University, China*
[‡]*Department of Computer Science and Engineering, Michigan State University, USA*
[♯]*Department of Computer Science, University of California, Irvine, USA*
[§]*Future Network Theory Lab, Huawei, Hong Kong, China*

***Abstract*—In datacenter networks, flows can have different performance objectives. We use a tenant-objective division to denote all flows of a tenant that share the same objective. Bandwidth allocation in datacenters should support not only performance isolation among divisions but also objective-oriented scheduling among flows within the same division. This paper studies the Multi-Tenant Multi-Objective (MT-MO) bandwidth allocation problem. To our best knowledge, no existing practical work support performance isolation and objective scheduling simultaneously. We propose Stacked Congestion Control (SCC), a distributed host-based bandwidth allocation design, where an underlay congestion control (UCC) layer handles contention among divisions, and a private congestion control (PCC) layer for each division optimizes its performance objective. Via the tenant-objective tunnel abstraction, SCC achieves weighted bandwidth sharing for each division in a distributed and transparent way. By adding a rate-limiting send queue in the ingress of each tunnel, mechanisms between performance isolation and objective scheduling are completely decoupled. We evaluate SCC both on a small-scale testbed and with large-scale NS-2 simulations. Compared to the direct coexistence cases, SCC reduces latency by up to 40% for Latency-Sensitive flows, deadline miss ratio by up to 3.2× for Deadline-Sensitive flows, and average flow-completion-time by up to 53% for Completion-Sensitive flows.**

I. Introduction

Motivation: In datacenter networks, flows can have different performance objectives. A private datacenter is shared by various tenants, such as search engine, advertising and e-Business applications. Each tenant can run many service entities (*e.g.*, Virtual Machines, Containers, Java processes) that communicate over the underlying network. The flows generated by these services have different performance objectives due to their service requirements. Some flows are *Latency-Sensitive (**LS**):* service can enqueue copies of a task in multiple servers to combat computation time variability [1]; to minimize resource wastage, a cancelation message should be sent to the counterpart servers as soon as the first replica is finished. On the other hand, some flows are *Deadline-Sensitive (**DS**)*: the partition-aggregate architecture of Online Data Intensive applications (OLDI) [2], [3] and real-time analytic [4], [5] enforce deadline semantics for every leaf-to-parent flow. Furthermore for many other applications, minimizing average flow-completion-time (AFCT) can significantly improve their performance [6], [7], [8], and we call these flows as *Completion-Sensitive (**CS**)* flows. We use a *tenant-objective division* to denote all the flows of a tenant that share the same performance objective.

Bandwidth allocation in datacenters should support not only performance isolation among divisions but also objective-oriented scheduling among flows within the same division. Bandwidth allocation design, in essence, defines how flows behave when congestion happens. Most datacenter networks are oversubscribed [9] and congestion is not uncommon: packet drops due to congestion can be observed when the whole network utilization is around only 25% [10]. To achieve performance isolation, administrators can assign weights to different divisions that share the underlying network [11]. For example, upon congestion, an administrator may prefer tenant A's DS flows over tenant B's DS flows, or all tenants' LS flows over their CS flows. Various techniques can be used to support objective-oriented flow scheduling: some reduce tail latency of messages [12], [13], [14], [15], some add deadline awareness [7], [16], [17], [18], and others focus on reducing AFCT [7], [19], [20], [6], [8], [21], [22], [23], [24].

This paper studies the Multi-Tenant Multi-Objective (MT-MO) bandwidth allocation problem in datacenter networks. To our best knowledge, *no existing work supports performance isolation and objective scheduling simultaneously.*

Limitations of Prior Art: Many of the existing objective-oriented approaches [7], [12], [13], [14], [15], [19], [20], [6], [8], [21], [22], [23], [24] are designed to achieve only a single performance objective at a time, and there could be severe interference if approaches of different objectives coexist without isolation. This happens because these approaches may detect congestion differently (*e.g.*, packet drop, or ECN) or react to congestion differently (*e.g.*, the ECN co-existence problem in production Cloud [25], [26]). pFabric [6] and Karuna [27] evaluate the coexistence of the DS and CS flows by setting absolute priority to DS flows over CS flows. However, performance isolation among flows with the same objective but of different tenants is not considered. Furthermore, existing performance isolation approaches cannot optimize performance objectives for individual tenant-objective division. Neither bandwidth guarantee [28], [29], [30], [31], [32], [33] nor proportional sharing [11] can perform bandwidth allocation at flow-level granularity.

Bandwidth allocation design should be practical and readily-deployable. Many works either require non-trivial switch modifications [13], [7], [19], [6], [34], or assume non-blocking

978-1-5090-5336-0/17/$31.00 ©2017 IEEE

network, which is not widely available in production datacenter [24]. A centralized scheduling mechanism (*e.g.*, Fastpass [21]) may perform optimization for all divisions, but requires an ideally scalable control plane and time synchronization for arbitration.

Proposed Approach: In this paper, we propose Stacked Congestion Control (SCC), a distributed host-based bandwidth allocation framework. SCC can support both *performance isolation* and *objective scheduling*, and is readily-deployable. SCC has two layers to perform congestion control: (i) an underlay congestion control (UCC) layer handles contention among divisions (*i.e.* performance isolation), and (ii) a private congestion control (PCC) layer for each division to optimize its performance objective (*i.e.* objective scheduling). UCC supports the following abstraction: a weight (obtained via administrator policy or utility function maximization [35]) is attached with each division; each division should get a bandwidth share that is proportional to its weight.

Technical Challenges and Solutions: The first challenge is how to achieve weighted bandwidth allocation by controlling the active flows within each division in a distributed and transparent way. Irrespective of the tenant-objective division semantic, TCP (and its variants) only allocates bandwidth proportional to the contending flows. SCC thus introduces *tenant-objective tunnel* as an edge-based abstraction (between every eligible pair of source and destination hosts, flows of each division are aggregated inside a dedicated point-to-point tunnel). Each tunnel is assigned a weight (dynamically updated by the division's PCC) and SCC allocates the bandwidth share to a tunnel in proportion to its weight. Because of the state-of-the-art multi-path load balancing schemes (*e.g.*, Presto [36]) in datacenter (*e.g.*, Clos-network), the share of a tunnel in each network tier is statistically proportional to its weight. Our key insight is that the division's bandwidth share can be achieved by cooperatively scaling each tunnel's network weight if the tunnels in the same division ensure that the sum of their weights equals the global weight of the division.

The second challenge is how to support two congestion control mechanisms (UCC and PCC) simultaneously. For distributed host-based bandwidth allocation, two factors define congestion control behaviour: (i) *congestion signal*, *i.e.*, what method is used to detect network congestion, and (ii) *control law*, *i.e.*, what rate increase (decrease) law is used to grab (yield) network bandwidth. The dilemma is that, to respect performance isolation, all tunnels should follow the same UCC mechanism; there is only one uniform set of congestion signal available from the network, which reflects the contention among tunnels; it cannot be directly exploited for objective-oriented congestion control inside each tunnel. Our technical contribution here is to add a rate-limiting queue in the ingress of each tunnel to completely decouple two congestion control mechanisms. UCC uses ECN as the congestion signal and uses weighted network sharing algorithms (similar to Seawall [11]) to derive the tunnel rate according to network condition. Moreover, each sender queue can locally generate desired congestion signal for PCC layer.

Further, we develop PCC algorithms for every objective to dynamically derive the per-tunnel weight, so that each tunnel is allocated a weight according to the demand of its flows as compared to other flows within the same division. Besides existing objective-oriented scheduling approaches, we also develop new in-tunnel scheduling algorithms to exploit the extra flexibility provided by SCC.

We implement SCC as a Linux kernel module using NetFilter. We evaluate SCC on a small-scale testbed with 16 Dell servers and a commodity PICA-8 Gigabit Ethernet switch with ECN enabled. To complement our small-scale testbed experiments, we further conduct large-scale simulations using NS-2. SCC can meet requirements of performance objectives, when flows of different objectives coexist in the datacenter networks. Compared to the direct coexistence cases, SCC reduces latency by up to 40% for Latency-Sensitive flows, deadline miss ratio by up to 3.2× for Deadline-Sensitive flows, and average flow-completion-time by up to 53% for Completion-Sensitive flows.

II. Background and Motivation

In this section, we first summarize the existing work done to improve the performance of applications with different objectives (§II-A). Next, via toy scenarios, we demonstrate that, when sharing the same network, existing approaches can interfere with each other and degrade the performance (§II-B). Lastly, we discuss a range of techniques that share the high-level similarities but are insufficient to support coexisting performance objectives in datacenter networks (§II-C).

A. Related Work

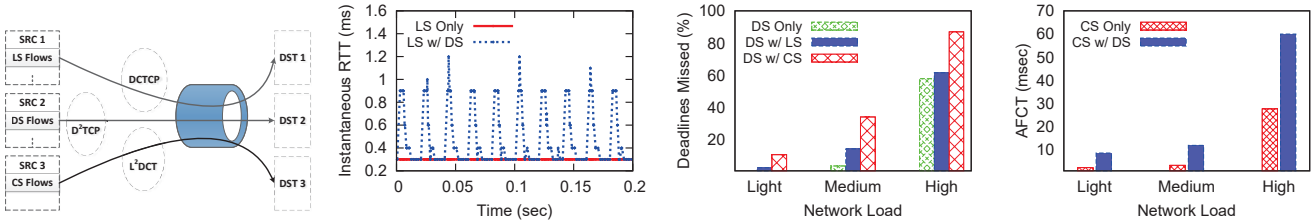
Latency Sensitive (LS): Silo [14] proposes a placement algorithm and a hypervisor-based packet pacing to provide latency guarantees. HULL [13] trades bandwidth for ultra-low latency and it requires modification to switch ASIC. Fastpass [21] uses a centralized arbiter to decide when and which path each packet should be sent. QJUMP [15] uses multiple physical switch queues to provide different combinations of latency and throughput. DCTCP can keep switch queue length at a low level to reduce packet latency [12].

Deadline Sensitive (DS): D^3 pioneers the idea of incorporating deadline awareness into network scheduling [16]. However, D^3 and RACS [37] are not deployment friendly as they require modifications to the commodity switch hardware. D^2TCP [17] is a distributed deadline-sensitive protocol, which uses both ECN feedback and deadlines to modulate the congestion window.

Completion Sensitive (CS): PDQ [7] uses preemptive flow scheduling to minimize average FCT. pFabric [6] and PASE [8] assume flow size is known a priori, and attempt to approximate Shortest Job First (SJF), which is the optimal scheduling for minimizing average FCT in a single bottleneck scenario. L^2DCT [20] and PIAS [22], without prior knowledge of flow size information, reduce AFCT by approximating the Least Attained Service (LAS) scheduling discipline.

B. Interference when Sharing Network

To demonstrate interference, when flows with different objectives coexist in the same network, we use a single bottleneck topology as shown in Fig. 1(a). We set the switch ECN



(a) Example topology (b) LS flows' RTT (c) DS flows' missing ratio (d) CS flows' finish time

Fig. 1. Interference demonstration, when alone and coexist.

marking threshold to 20, the switch buffer size to 250 packets and the round trip time to 300 us. We consider traffic from a LS application (from $SRC1$ to $DST1$), a DS application (from $SRC2$ to $DST2$), and a CS application (from $SRC3$ to $DST3$). We use the instantaneous end-to-end packet RTT, fraction of missed deadlines, and average FCT (AFCT) as the performance metrics for evaluating the LS, DS and CS flows, respectively. We use NS-2 for performance evaluation.

We consider state-of-the-art deployment-friendly protocols for each of the performance goals and implement them in NS-2. Specifically, we use DCTCP for LS flows, D^{2}TCP for DS and L^{2}DCT for CS flows. For LS flows, we generate small (2 KB) flows that arrive at intervals following a Poisson distribution and require an aggregate bandwidth of 400 Mbps on average. For DS and CS, the flows are generated uniformly within the range 2 KB-198 KB with deadlines in the range 5-25 ms (for DS only). Flows arrive at intervals generated following Poisson distribution and incurring low (20%), medium(50%) and high (80%) load on the link.

1) Performance in Isolation: We first evaluate the application performance in a scenario where each protocol has exclusive access to the underlying network. When used in isolation, we represent flows as Base-LS, Base-DS, and Base-CS respectively. In these experiments, the bottleneck link capacity is 500 Mbps.

Base-LS: As shown in Fig. 1(b), the LS flow packets experience almost zero queuing delay and the round-trip latency is very close to the RTT (300 us).

Base-DS: Fig. 1(c) demonstrates the deadline missing rate of DS flows across a range of loads. We can see that for light to moderate loads, almost no flows miss deadlines. However, at high load many flows miss deadlines.

Base-CS: Fig. 1(d) shows the AFCT under different network load conditions.

2) Coexistence of Different Objectives: Next, we evaluate the scenarios with two performance objectives coexisting in the network. The bottleneck link capacity is 1000 Mbps, which is large enough to meet the demands of both types of flows.

Coexist-LS/DS: In this scenario, the performance of LS flows is severely affected as shown in Fig. 1(b). LS flows experience large queuing delays and, as a result, their tail latency increases by more than 300%; the average latency increases by more than 50%. On the other hand, DS flows also experience performance degradation and many flows miss their deadlines, as shown in Fig. 1(c).

Coexist-DS/CS: We consider the coexistence of DS and CS flows in the network. In this scenario, DS flows experience performance degradation, as shown in Fig. 1(c): even at low loads, some of the flows miss their deadline due to interference from CS flows and the degradation increases up to 4x for medium and higher network loads. Similarly, the completion times of CS flows is also affected and flows take longer to finish compared to the scenario with only CS flows in the network (Fig. 1 (d)). Specifically, at higher loads, the AFCT of CS flows increases by more than 2x.

3) Same Objective Different Tenants: We also consider the scenarios where multiple tenants with the same performance objective coexist. We evaluate both *Coexist-LS/LS* and *Coexist-DS/DS* scenarios, where the bottleneck capacity is not large enough to meet the requirements of all flows. Lacking the ability to favor one tenant over another, flows of both the tenants experience performance degradation. We omit results due to space limitation.

C. Current Isolation won't Work

An intuitive solution is to segregate application flows with different objectives to separate physical queues in a switch. Such an segregation approach requires a large number of traffic classes, which are not supported by currently available CoS tags in packet formats; existing commodity switches also do not have sufficient number of queues [11].

Bandwidth Guarantee: Several bandwidth guarantee approaches have been proposed, which provide network abstraction models and placement algorithms for tenants or applications to express their bandwidth requirements [28], [11], [29], [30], [31], [32]. Bandwidth guarantee are coarse grained: they should meet the peak requirements, which is either inefficient, or is suboptimal in terms of flow objectives.

Proportional Sharing: Seawall provides per-entity weight enforcement for each VM-to-VM tunnel [11]. However, it does not support bandwidth allocation at flow-level granularity, hence cannot optimize the performance objectives of individual flow divisions. Further, it does not support tenant level network sharing.

III. SCC Design

A. Design Overview

Network Mode: This paper targets the prevalent Clos-network fabrics. We assume the adoption of state-of-the-art load balancing works such as Presto [36]: a large flow can be divided into fine-grained, near-uniform units of data (*i.e.*, *flowcells*) and load is balanced across almost every available path between two neighboring tiers of network switches.

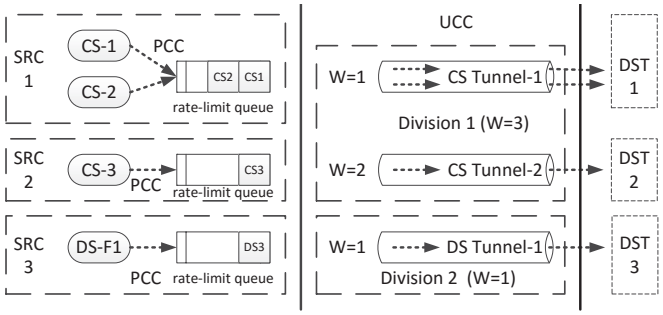


Fig. 2. SCC framework with tenant-objective tunnel abstraction.

Allocation of network weights among divisions is outside the scope of this paper. We assume each division weight is given either as input by an algorithm that enforces administrator policy or by a utility maximization function [35], [38].

Data Path: An example data path is shown in Figure 2. There are two CS tunnels and one DS tunnel among three pairs of hosts: division 1 contains two CS tunnels; division 2 contains one DS tunnel; all CS/DS flow(s) enter the CS/DS tunnel(s). Each tunnel has a rate-limiting queue at it's ingress port in the endhost. A tunnel weight W_t is attached with each tunnel.

Note that tunnel is only an logical abstraction: it does not require any additional packet headers, hence NIC offload features such as TCP Segmentation Offload (TSO) are affected.

B. Underlay Congestion Control

Congestion Signal: UCC uses ECN as the congestion signal and uses weighted network sharing algorithms to derive the contemporary tunnel rate. For all outgoing packets in a tunnel, the ECN-enable bits in IP headers are set by the tunnel. In the receiver end of the tunnel, the ECN mark ratio is recorded; the destination host periodically sends ECN feedback; for all incoming TCP acknowledgement packets, ECN bit is removed before transferring to upper transport protocols. Note that if the upper transport protocol supports ECN, UCC should pass this information to PCC layer.

Tunnel Rate: SCC provides the tunnel with a bandwidth share that is proportional to its weight; This allocation is an end-to-end and work-conserving design. Each tunnel has a rate limiting queue attached to the ingress port in the source host. Based on the ECN feedback, each tunnel estimates the available bandwidth and adjusts the rate of the send queue. Similar to Seawall, we mimic the DCTCP behaviour in tunnel level congestion control. If there is no congestion in the network, the rate R_t increases as:

$$R_t = (1 + W_t) * R_t, \tag{1}$$

where, W_t is the weight assigned to a tunnel. Upon detecting congestion, the rate reduces as:

$$R_t = (1 - \alpha * W_t) * R_t, \tag{2}$$

where α is the level of congestion in the network.

Scaling Tunnel Weight: As a distributed solution, each tunnel in SCC, periodically generates its reference tunnel weight W'_t according to its PCC layer algorithm. This calculation is objective-oriented, and the corresponding algorithms are presented in Section IV. All tunnels in the same division should ensure that the global weight W_d allocated to this division is respected, by cooperatively scaling each tunnel's self-derived reference network weight.

A tracker is elected from all the source hosts in each division, and each source host periodically sends W'_t to this tracker. A scale factor are calculated as:

$$\lambda_d = W_d / \sum_{\forall t \in d} W'_t, \tag{3}$$

and λ_d is periodically sent back to hosts in the division. Accordingly, each source hosts derive its contemporary weight value as follows:

$$W_t = \lambda_d * W'_t. \tag{4}$$

In this way, the division weight is dynamically divided among all its constituent tunnels. As shown by the example in Figure 2, the sum of weights for CS tunnel 1 and tunnel 2 equals 3, which is equal to the division 2's allocated weight.

Due to the impact of load balancing, in each tier of the network, the share of bandwidth obtained by a tunnel is statistically proportional to its weight. Since the sum of their weights equals the global weight allocated to this division, the division's bandwidth share is achieved.

C. Private Congestion Control

Congestion Signal: SCC is completely transparent to the upper layer transport protocols. The congestion signal for objective-oriented scheduling is generated from the underlying host send queue instead of the network. Each division tunnel has full control of using its own congestion signal and control law; it can either rely on existing objective-oriented transport protocols, or directly schedule flows in the tunnel. The upper transport protocols may enable ECN support, such as in D^{2}TCP, DCTCP and L^{2}DCT. Note that in SCC, ECN bits are used for division-level congestion control and before handing over to upper layers, all ECN marks are removed. For this scenario, SCC remarks the ECN bit based on concurrent queue length and properties of flows inside the tunnel.

It requires no operation if the transport protocols rely on round trip time. Otherwise, if the transport protocols rely on packet drop, a queue length should be set (*e.g.*, 100 packets).

Generate Reference Tunnel Weight and In-Tunnel Scheduling: To optimize the performance objective, there are two algorithm options for a particular division: (i) generate reference tunnel weight W'_t according to contained flows, and relies on upper layer transport protocols to schedule flows inside the tunnel; and (ii) integrated design of both reference tunnel weight and in-tunnel scheduling. We present our PCC algorithms in Section IV. Note that SCC is a general architecture: researchers can developed algorithms for different objectives, or algorithms also target LS/DS/CS flows but with better performance.

IV. PCC Tunnel Algorithms

In this section, our designs of tunnel algorithms, for DS/CS/LS divisions are presented. To optimize the performance objective, we adopt an integrated design method of both reference tunnel weight and in-tunnel scheduling. Even if upper layer transport protocols are preferred to schedule flows

inside the tunnel, the reference tunnel weight algorithms can still be used.

A goal of algorithms is that, for a division, using a datacenter network exclusively, the control mechanism should achieve comparable performance with and without SCC. Assume four DS flows with the flow priority $f_1 > f_2 > f_3 > f_4$; f_1 and f_2 share the same tunnel, and f_3 and f_4 share the same tunnel. Prior to SCC, and without the concern of coexistence, all DS flows may compete independently. With SCC , now f_1/f_2 tunnel competes with another tunnel that contains f_3/f_4. This metric reflects the overhead/gain of SCC algorithms.

We omit the detailed proofs for the optimality of the algorithms due to space limitation. The symbols used are listed in Table I.

$M_s(t)$	remaining data size for session s at time t
$\delta_s(t)$	remaining time till deadline for session s at time t
$x_s(t)$	requested source rate for session s at time t
$Q_s(t)$	buffer status for session s at time t
λ_s	average data arrival rate for session s
Q_s^*	target average queue length
$BW_{req}(t)$	request from the host on the bandwidth at time t
$BW_{res}(t)$	response from the coordinator on thebandwidth at time t
τ	decision slot duration

TABLE I
NOTATION TABLE

A. Deadline-Sensitive Tunnel

Deadline-sensitive flows require to transfer a flow of certain size within a deadline. It is well known that for a single flow session s with remaining size M_s and deadline δ_s, a guaranteed bandwidth x_s no less than M_s/δ_s can meet its deadline. Therefore, SCC dynamically updates the required minimum bandwidth of each DS tunnel based on the sizes and deadlines of all active flows in the tunnel (step 1 and 2, Algo. 1). This requirement is updated periodically ($\approx$ msec).

Algorithm 1: Rate Control for DS Tunnel

At the beginning of each decision slot, the tunnel
Step 1: Updates the flow information, $\{M_s(t), \delta_s(t)\}$
Step 2: Estimates the rates $x_s^*(t) = \frac{M_s(t)}{\delta_s(t)}$ and $BW_{req}(t) = \sum_s x_s^*(t)$.
Step 3: Let $W_t' = BW_{req}(t)$, gets W_t from W_t', and set $BW_{res}(t) = W_t$.
In-tunnel scheduling
if $BW_{req}(t) < BW_{res}(t)$ **then**
  Send data at $x_s^*(t)$ calculated in Step 2.
else
  Satisfy the first $N(t)$ sessions $\{r(1), r(2), \ldots, r(N(t))\}$ with $\sum_{s=1}^{N(t)} \frac{M_{r(s)}(t)}{\delta_{r(s)}(t)} \leq BW_{res}(t)$ and $\sum_{s=1}^{N(t)+1} \frac{M_{r(s)}(t)}{\delta_{r(s)}(t)} > BW_{res}(t)$, where $r(s) = 1, 2, 3, \ldots$ and $\frac{M_{r(1)}(t)}{\delta_{r(1)}(t)} \leq \frac{M_{r(2)}(t)}{\delta_{r(2)}(t)} \leq \frac{M_{r(3)}(t)}{\delta_{r(3)}(t)} \leq \cdots$.

The tunnel responds to the allowed bandwidth, which may or may not be equal to the required bandwidth (step 3, Algo. 1). PCC can choose to either use or not use in-tunnel scheduling algorithm, irrespective of the upper layer transport protocols, . We demonstrate later in evaluation, that in-tunnel scheduling can provide significant performance improvement, which accredits to the flexibility of the SCC framework.

B. Completion-Sensitive Tunnel

The bandwidth sharing mechanism for CS tunnels is similar to the DS tunnels, except that only flow sizes are used to compute required bandwidth. To mimic the Shortest-Flow-First (SFF) strategy, the inversion of flow size is used as each flow's weight (step 2, Algo. 2). Inside the CS tunnel, flows are simply scheduled by SFF.

Algorithm 2: Rate Control for CS Tunnel

At the beginning of each decision slot, the tunnel
Step 1: Updates the flow information, $\{M_s(t)\}$
Step 2: Estimates the weight $W_t' = \sum_s 1/M_s(t)$.
Step 3: Gets W_t from W_t'.
In-tunnel scheduling
Shortest-Flow-First

C. Latency-Sensitive Tunnel

For LS tunnels, the buffer length in the host/network queues defines the flow latency. This algorithm considers queuing at the end-hosts only, not from the switches in the network. We rely on tunnel level UCC protocols to maintain switch queues at a low-level, and an additional option is to put LS flows into a higher physical queue.

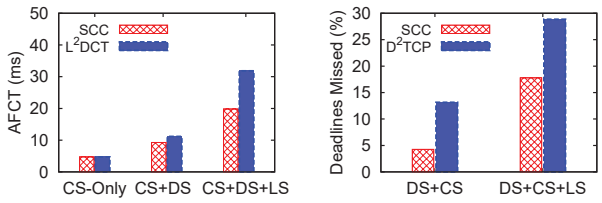
PCC maintains small queues in the tunnel by using ECN based notification. To compute bandwidth requirement, PCC leverages insights from how existing transport protocols work. For example, most of the existing datacenter transport protocols start with initial congestion window of 10 packets (such as DCTCP, L^{2}DCT, etc.,), and to avoid timeouts, they need at least one ACK per RTT. Therefore, we set average bandwidth of a tunnel to be at least $10 \times N_f$, where N_f is the number of LS flows in each tunnel. Additionally, we also consider the queue size to determine the target rate for the LS flows as listed in Algo. 3.

Algorithm 3: Rate Control for LS Tunnel

At the beginning of each decision slot, the tunnel
Step 1: Update the current queue status $\{Q_s(t)\}$
Step 2: Estimates the rates $x_s^*(t) = \frac{Q_s(t) - \epsilon_s Q_s^*}{\tau} \mathbf{1}\left[Q_s(t) > \epsilon_s Q_s^*\right]$ and $BW_{req}(t) = \sum_s x_s^*(t)$.
Step 3: Let $W_t' = BW_{req}(t)$, gets W_t from W_t', and set $BW_{res}(t) = W_t$.
In-tunnel scheduling
if $BW_{req}(t) < BW_{res}(t)$ **then**
  Send data at $x_s^*(t)$ calculated in Step 2.
else
  Satisfy the first $N(t)$ sessions $\{r(1), r(2), \ldots, r(N(t))\}$ with $\sum_{s=1}^{N(t)} \frac{Q_s(t) - \epsilon_s Q_s^*}{\tau} \mathbf{1}\left[Q_s(t) > \epsilon_s Q_s^*\right] \leq BW_{res}(t)$ and $\sum_{s=1}^{N(t)+1} \frac{Q_s(t) - \epsilon_s Q_s^*}{\tau} \mathbf{1}\left[Q_s(t) > \epsilon_s Q_s^*\right] > BW_{res}(t)$, where $r(s) = 1, 2, 3, \ldots$ and $V_{r(1)}(t) \geq V_{r(2)}(t) \geq V_{r(3)}(t) \geq \cdots$ with $V_s(t) = Q_s(t) + Q_s^* - 2\epsilon_s Q_s^*$.

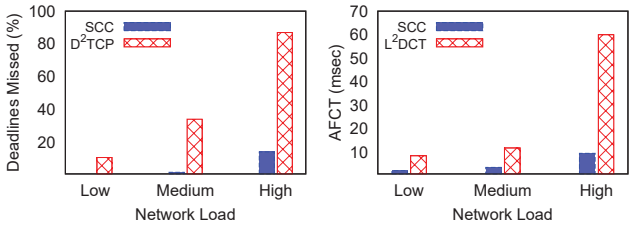
V. TESTBED VERIFICATION

In this section, we first discuss SCC system implementation and testbed evaluation. Then, we compare simulation results to testbed results.



(a) CS flows (b) DS flows

Fig. 3. Testbed Results w/ & w/o SCC.



(a) DS (b) CS

Fig. 4. Performance under CS/DS coexistence scenario

System Implementation: We implement the SCC shim layer as a Netfilter kernel module in Linux. Two hash based flow tables, one for send and one for receive, are used for packet classification and for tracking per-flow state. To enforce accurate rates over short timescales, we use Linux high-resolution kernel timer, HRTIMER, for our rate limiters.

At the sender, we use the LOCAL_OUT hook to intercept all outgoing packets to enforce virtual tunnel abstraction. To support ECN, the ECN-capable (ECT) codepoint is marked in every packet's IP header. The packet is then forwarded to a rate limited per-tunnel queue.

At the receiver end, we use the LOCAL_IN hook to intercept all incoming packets. Each packet is matched against the receive table, and its ECN bit is checked. The receiver shim layer calculates the fraction of ECN marking packets and delivers this information back to the sender that uses it to perform tunnel-level congestion control. We have not implemented in-tunnel algorithms for the testbed.

Testbed Experiments: For Testbed experiments, we use a single rack with 16 DELL servers (with 1G NICs) and PICA-8 Gigabit switch with ECN support enabled. For evaluation, we consider three kind of traffic, latency-sensitive (D-CTCP), deadline-sensitive (D^{2}TCP) and completion-sensitive (L^{2}DCT). We use same settings as Base-DS, Base-LS and Base-CS for each traffic.

First, we consider CS application in isolation, with and without SCC support. Flows are generated such that it incurs an average load of 400 Mbps (low load). Figure 3(a) (*i.e.* only scenario) shows that performance of CS flows remain the same with or without SCC support in this scenario. The reason is that, in a network with low load, flows do not experience interference and finish quickly.

Next, we consider coexistence of CS/DS (medium load) and CS/DS/LS (high load) flows in the network. With SCC support enabled (Figure 3(a)), the AFCT of L^{2}DCT flows is improved by $17\%, 53\%$ for CS/DS and CS/DS/LS coexistence scenarios, respectively. With SCC, deadline missing of DS flows is reduced by $2\times, 1\times$ for CS/DS and CS/DS/LS coexistence scenario, respectively.

Verification using Simulations: In this section, we verify NS2 simulations via testbed results, when the CS and DS flows coexist in the network, under the motivation example scenario. We use same settings as Base-DS, and Base-CS for each application and vary network load. In this scenario, as shown earlier, DS flows and CS flows experience degraded performance (Fig. 1(c) and (d)). Figure 4 shows that with SCC support, fewer flows miss their deadlines, more specifically, at higher loads, we observe $4\times$ performance improvement compared to when D^{2}TCP is used. Similarly, completion time of flows is improved by 5x at higher loads. This is due to the in-queueing at the end-hosts and the adaptive ECN marking at shim layer. We observe similar performance trends as testbed at low and medium loads, while simulations show better improvement at higher loads.

VI. Large-scale NS-2 Simulations

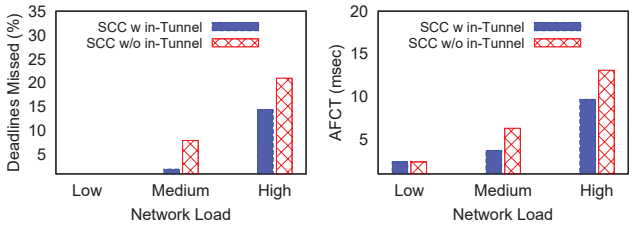
Our evaluation addresses the following questions:

- **Can SCC improve the performance of different flows when they coexist?** We evaluate the scenarios where flows with different performance objectives coexist in the data center network. SCC is able to meet the requirements of objectives like CS, DS or LS simultaneously across a wide range of workloads such as Web-search [12], Data-mining [39] and MapReduce [40]. SCC improves performance by up to $3.2\times$ for DS flows and 40% for CS flows as compared to state-of-art protocols like pFabric [6].
- **Can SCC achieve the same or even better performance when only flows of the same objective exist in the network?** In our evaluation, we show that SCC can achieve similar or better performance for protocols when they exist in the network alone. At high loads, SCC can reduce the deadline miss rate by up to $2\times$ and achieve similar AFCTs.
- **Does SCC consistently perform well?** We test SCC performance in oversubscribed settings and with different transport protocols. The results demonstrate that with SCC enabled, most protocols deliver better performance.

Evaluation Setup: We use a Clos topology for NS2 simulations, unless specified otherwise. The capacity of edge links is 1 Gbps and core links is 10 Gbps. We assume ECN capable switches with 250 KB buffering and maximum end-to-end RTT of 300 usec [8].

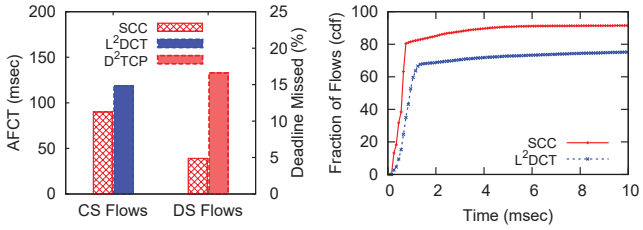
We model two traffic classes. Class-I traffic belongs to deadline-sensitive small message application and requires deadline guarantees. Class II traffic is similar to Class-I, but has different objective than deadline, such as minimizing flow completion times or latency-guarantees. We replicate Web-search [12], Data-mining [39] and MapReduce [40] workloads for Class-I and Class-II traffic. This setting coarsely models the workload for OLDI applications and distributed storage.

In our experiments, deadline-sensitive traffic uses D^{2}TCP [17], latency-sensitive uses DCTCP [12], and completion-sensitive traffic uses L^{2}DCT [20] transport protocol at the endhosts. We use default parameters, from



(a) DS flows (b) CS flows

Fig. 5. Benefits of In-tunnel algorithm



(a) CS and DS (b) CDF of CS flows

Fig. 6. Coexistence of CS/DS/LS flows with datamining workload

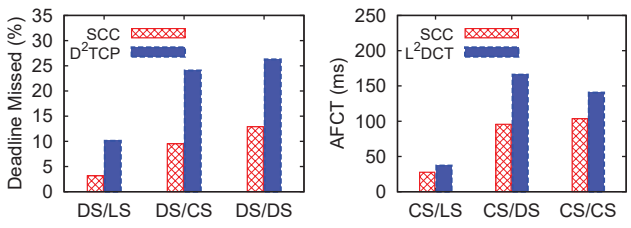
respective papers, for each of the protocol. The applications generate traffic according to a poisson process, such that the average bandwidth requirement of each class is 300 Mbps to meet its requirements. For deadline flows, the deadlines are exponentially distributed using guidelines from [20].

A. Comparison in Coexistence Scenario

In this section, we evaluate the impact of interference, on application performance, when flows with different performance objectives coexist in the network.

Coexistence Performance w/o In-Tunnel Algorithms: In this section, we show the benefits of using in-tunnel scheduling algorithms on top of objective-tunnel scheduling mechanism. To illustrate the benefits, we reuse the simulation setup of motivation example and show the performance when DS and CS flows coexist in Fig. 5. As shown in Figure 5(a), the deadline miss ratio of DS flows reduces by up to 8% when in-tunnel scheduling is used. Similarly, as shown in Figure 5(b), the AFCT performance of CS flows reduces by up to 6% when in-tunnel scheduling is used. This shows that simple in-queue scheduling can further provide benefits on top of the tunnel level abstraction. We observe similar gains in other scenarios. In all other experiments, we use in-tunnel scheduling algorithms.

Coexist-LS/DS/CS Performance: In the first setup, we consider data-mining workload and assume that the flows of the three objectives (i.e., latency-sensitive, deadline-sensitive and completion-sensitive) coexist in the network. The aggregate workload generated by traffic from three objectives is 900 Mbps, generating 90% network load. Figure 6 shows that when the three objectives coexist without SCC support, they affect each other's performance. Data-mining workload is more skewed and more than 80% of the flows are less than 2 KB, SCC completes 3.2× more flows than without SCC, in Figure 6(a). SCC mitigates network interference and improves the AFCT by 40% compared to network without SCC support, in Figure 6(b). SCC, improves latency of latency-sensitive flows' by up to 40%. We omit results due to space limitation.



(a) DS (b) CS

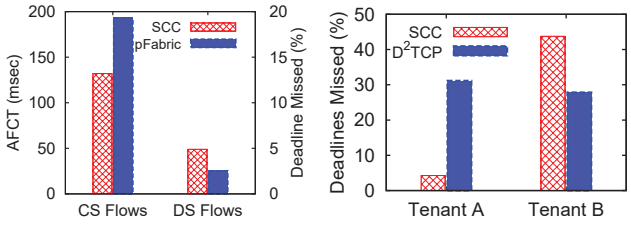
Fig. 7. Coexistence with Data-mining workload

Coexistence performance of different objectives also hurts in the presence of interference traffic without SCC. To evaluate the impact of interference, we consider same settings as above and use Data-mining workload. When coexisting, DS flows miss lots of deadlines because of interference in Figure 7(a). While coexistence with CS and LS flows, SCC reduces the deadline miss ratio by up to 3×. When coexisting, CS performance affects more in the presence of DS flows in the network, in Figure 7(b). However, when coexisting with only LS flows, the AFCT is not degraded. The reason is that there is sufficient capacity available in the network and flows do not experience queuing delays.

Comparison with State-of-the-Art: We also compare the SCC performance with state-of-the-art protocols such as pFabric [6] that consider the coexistence of different objectives like CS/DS. We consider coexistence of CS and DS flows using data-mining workload and each application generates 40% load in the network. For pFabric, we assign high priority to DS flows as compared to the CS flows. Figure 8(a) shows that SCC improves the performance of the CS flows by 30%, compared to pFabric. However, SCC misses 2% more deadlines for DS flows. This is expected as, in this scenario, SCC does not enforce any specific prioritization policy in the network.

B. Comparison in Policy Enforcement

In this section, we consider the coexistence performance of same tenants or objectives but with constraints on the available network bandwidth and weight. We consider two tenants (A and B) with DS objective and both the tenants require 600 Mbps bandwidth to meet the deadline requirements of their applications. We assume that tenant A has more priority in the network, thus operator assigns more bandwidth to A and assigns rest of the available bandwidth to the tenant B. Figure 8(b) shows that SCC meets almost all the deadlines of Tenant A, however, the performance degrades for Tenant B. SCC meets more deadlines for both the Tenants combined.



(a) pFabric (b) Admin policy

Fig. 8. Comparison under Data-mining workload

C. Comparison in Isolation Scenario

In this setup, we evaluate performance of SCC for deadline-sensitive and completion-sensitive objectives when they have exclusive access to the network and compare it to the performance of D^{2}TCP and L^{2}DCT protocols respectively. The goal is to evaluate if, in the absence of coexistence, SCC can achieve similar performance as the transport protocols across a range of network loads. We use data-mining workload and use same network setup as in section VI.

For deadline-sensitive applications, SCC meets more deadlines across all the network loads and reduces the deadline missing by up to $2.5\times$ (Figure 9(a)). This shows that SCC not only eliminates interference among flows of different objectives, but also improves performance via the abstraction of objective-tenant congestion-free tunnel.

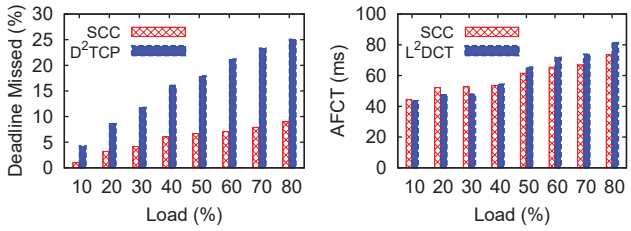
For completion-sensitive applications, SCC minimizes the AFCT and achieves performance similar to L^{2}DCT, as shown in Figure 9(b). SCC achieves similar performance as L^{2}DCT at low loads and improves AFCT at higher loads by eliminating the inter-flow interference via its priority sub-queues.

D. SCC Dynamics

In this section, we evaluate SCC performance under different workloads, network oversubscription, different transport protocols, and strict priority.

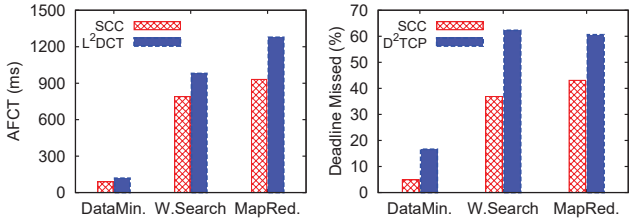
Coexist-LS/DS/CS Performance under different workloads: Next, we evaluate CS/DS/LS co-existence performance under different workloads in the network. The aggregate workload generated by traffic from three objectives is 900 Mbps, generating 90% network load.

Figure 10 shows that when the three objectives coexist without SCC support, they affect each other performance. SCC, reducers AFCT of CS flows' by up to $40\%, 24\%, 32\%$ for Data-mining, Web-search, and MapReduce Figure 10(a). SCC reduces deadline missing ratio by $3.2\times/1.8\times/0.8\times$ for the Data-mining, Web-search, and MapReduce workloads respectively, Figure 10(b). We observe similar trends for Web-search and MapReduce workloads.



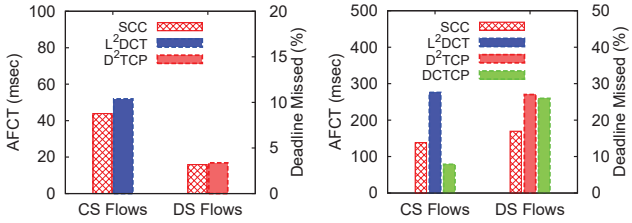
(a) DS flows' missed deadline (b) CS flows' AFCT

Fig. 9. Isolation scenario.



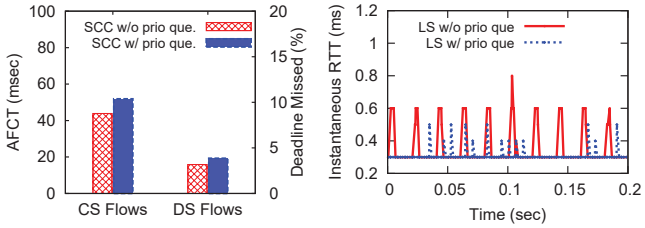
(a) AFCT of CS flows (b) missed deadline of DS flows

Fig. 10. Coexistence with and without SCC support for different workloads



(a) Over-subscribed (b) Different transport

Fig. 11. SCC Dynamics.



(a) CS/DS (b) LS latency

Fig. 12. Objective isolation using priority queues

Over-subscribed Network Scenario: We evaluate SCC on a 3:1 oversubscribed network. In this topology, we have 4 ToR switches, each connecters to 30 hosts with 1 Gbps links and connects to core switches using 10 Gbps links. We generate east-to-west traffic: all traffic is inter-rack. Note that load is calculated based on network core load compared to previous scenarios, where the load was generated based on the edge links. We consider three applications, CS, DS and LS in the network. All the flows follow the Web-search workload and have average network load of 300 Mbps each. Figure 11(a) shows that SCC reduces AFCT of CS flows by 20% and reduces deadline missing ratio of DS traffic only marginally.

Different Transport Protocol: Figure 11(b) shows that even with DCTCP as transport protocol, SCC improves the application performance compared to the original protocols. The AFCT of CS flows improves by $3\times$ as these flows get equal share of the network bandwidth as DS flows. DCTCP is more aggressive in increasing rate than the L^{2}DCT, for large flows DCTCP increases congestion window by 1 pkt/RTT whereas L^{2}DCT increases by the factor k, where $k \in 0.5, 1$, which depends on the flow size. The performance of DS flows is also better than D^2TCP protocol, however, the deadline performance degrades compared to the scenario where SCC uses D^{2}TCP as the transport protocol. The reason is that now both CS and DS flows share network fairly and benefits of D^2TCP at the transport layer are lost.

Isolating Tenants using Priority Queues: In this section, we compare and contrast the tradeoffs of using priority queues inside the switches, to isolate LS flows from other objectives to meet its requirements. To illustrate the benefits, we consider a network with two priority queues and coexistence of LS, DS and CS flows. Here, LS flows are mapped to the higher priority queue and CS/DS flows share the lower priority queue. Figure 12(a) shows that the performance of DS and CS flows is hurt a little when LS flows are prioritized over them. However,

latency requirements of LS flows are better met when strict priority is enforced in switch queues, in Figure 12(b).

VII. Conclusions

There exist two kinds of stakeholders in a datacenter networks: the tenants and the administrators. An individual tenant's congestion control tries to achieve its own performance objective. With a global view of the network, the administrators need to maximize its utility function. In this paper, we provide a framework to decouple the congestion control among different performance objectives, from the congestion control among flows with the same objective. This work demonstrate that simple solutions can be adopted to eliminate the interference among different performance objectives and different tenants. This work is inspired by the Application-Driven Network (ADN) [41].

Acknowledgment

The authors would like to thank anonymous reviewers for their valuable comments. This work is partially supported by the National Natural Science Foundation of China under Grant Numbers 61602194, 61402198, 61472184 and 61321491, the National Science Foundation under Grant Numbers CNS-1318563, CNS-1524698, CNS-1421407, and IIP-1632051, the Collaborative Innovation Center of Novel Software Technology and Industrialization, and the Jiangsu Innovation and Entrepreneurship (Shuangchuang) Program.

References

[1] J. Dean and L. A. Barroso, "The tail at scale," *Communications of the ACM, 2013*.

[2] V. Jalaparti, P. Bodik, S. Kandula, I. Menache, M. Rybalkin, and C. Yan, "Speeding up distributed request-response workflows," in *ACM SIGCOMM 2013*.

[3] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: amazon's highly available key-value store," in *in ACM SIGOPS 2007*.

[4] S. Melnik, A. Gubarev, J. J. Long, G. Romer, S. Shivakumar, M. Tolton, and T. Vassilakis, "Dremel: interactive analysis of web-scale datasets," *in Proc. of the VLDB 2010*.

[5] C. Engle, A. Lupher, R. Xin, M. Zaharia, M. J. Franklin, S. Shenker, and I. Stoica, "Shark: fast data analysis using coarse-grained distributed memory," in *ACM SIGMOD 2012*.

[6] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker, "pfabric: Minimal near-optimal datacenter transport," in *ACM SIGCOMM 2013*.

[7] C.-Y. Hong, M. Caesar, and P. Godfrey, "Finishing flows quickly with preemptive scheduling," in *Proc. ACM SIGCOMM 2012*.

[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 *ACM SIGCOMM 2014*.

[9] N. Farrington and A. Andreyev, "Facebook data center network architecture," in *IEEE Optical Interconnects Conf.* Citeseer, 2013.

[10] A. Singh, J. Ong, A. Agarwal, G. Anderson, A. Armistead, R. Bannon, S. Boving, G. Desai, B. Felderman, P. Germano *et al.*, "Jupiter rising: A decade of clos topologies and centralized control in google's datacenter network," in *Proc. ACM SIGDC 2015*.

[11] A. Shieh, S. Kandula, A. G. Greenberg, C. Kim, and B. Saha, "Sharing the data center network," in *NSDI*, 2011.

[12] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data center tcp (dctcp)," in *in Proc. ACM SIGCOMM 2011*.

[13] 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*. USENIX, 2012, pp. 19–19.

[14] K. Jang, J. Sherry, H. Ballani, and T. Moncaster, "Silo: Predictable message latency in the cloud," in *in Proc. ACM SIGDC, 2015*.

[15] M. P. Grosvenor, M. Schwarzkopf, I. Gog, R. N. Watson, A. W. Moore, S. Hand, and J. Crowcroft, "Queues don't matter when you can jump them!" in *NSDI*. USENIX, 2015.

[16] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowtron, "Better never than late: Meeting deadlines in datacenter networks," in *ACM SIGCOMM*, 2011, pp. 50–61.

[17] B. Vamanan, J. Hasan, and T. Vijaykumar, "Deadline-aware datacenter tcp (d2tcp)," in *ACM SIGCOMM*, 2012, pp. 115–126.

[18] H. Zhang, K. Chen, W. Bai, D. Han, C. Tian, H. Wang, H. Guan, and M. Zhang, "Guaranteeing deadlines for inter-datacenter transfers," in *Proceedings of the Tenth European Conference on Computer Systems*. ACM, 2015, p. 20.

[19] D. Zats, T. Das, P. Mohan, D. Borthakur, and R. Katz, "Detail: reducing the flow completion time tail in datacenter networks," in *ACM SIGCOMM*, vol. 42, no. 4, 2012, pp. 139–150.

[20] A. Munir, I. A. Qazi, Z. A. Uzmi, A. Mushtaq, S. N. Ismail, M. S. Iqbal, and B. Khan, "Minimizing flow completion times in data centers," in *in Proc. IEEE INFOCOM, 2013*.

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

[22] W. Bai, L. Chen, K. Chen, D. Han, C. Tian, and H. Wang, "Information-agnostic flow scheduling for commodity data centers," in *NSDI*. USENIX, 2015.

[23] R. Mittal, N. Dukkipati, E. Blem, H. Wassel, M. Ghobadi, A. Vahdat, Y. Wang, D. Wetherall, D. Zats *et al.*, "Timely: Rtt-based congestion control for the datacenter," in *Proc. ACM SIGDC 2015*.

[24] 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 *Proceedings of the CoNEXT*, 2015.

[25] A. Sivaraman, K. Winstein, S. Subramanian, and H. Balakrishnan, "No silver bullet: extending sdn to the data plane," in *in ACM HotNets 2013*.

[26] G. Judd, "Attaining the promise and avoiding the pitfalls of tcp in the datacenter," in *USENIX NSDI 2015*.

[27] L. Chen, K. Chen, W. Bai, and M. Alizadeh, "scheduling mix-flows in commodity datacenters with karuna," in *ACM SIGCOMM, 2016*.

[28] C. Guo, G. Lu, H. J. Wang, S. Yang, C. Kong, P. Sun, W. Wu, and Y. Zhang, "Secondnet: a data center network virtualization architecture with bandwidth guarantees," in *ACM CoNext 2010*.

[29] H. Rodrigues, J. R. Santos, Y. Turner, P. Soares, and D. Guedes, "Gatekeeper: Supporting bandwidth guarantees for multi-tenant datacenter networks," in *WIOV*, 2011.

[30] V. Jeyakumar, M. Alizadeh, D. Mazieres, B. Prabhakar, C. Kim, and A. Greenberg, "Eyeq: practical network performance isolation at the edge," in *NSDI*. USENIX, 2013.

[31] L. Popa, P. Yalagandula, S. Banerjee, J. C. Mogul, Y. Turner, and J. R. Santos, "Elasticswitch: practical work-conserving bandwidth guarantees for cloud computing," in *ACM SIGCOMM 2013*.

[32] J. Lee, Y. Turner, M. Lee, L. Popa, S. Banerjee, J.-M. Kang, and P. Sharma, "Application-driven bandwidth guarantees in datacenters," in *ACM SIGCOMM*. ACM, 2014, pp. 467–478.

[33] S. Hu, W. Bai, K. Chen, C. Tian, Y. Zhang, and H. Wu, "Providing bandwidth guarantees, work conservation and low latency simultaneously in the cloud," in *Proceedings of infocom*, 2016.

[34] F. R. Dogar, T. Karagiannis, H. Ballani, and A. Rowstron, "Decentralized task-aware scheduling for data center networks," in *ACM SIGCOMM*.

[35] L. Popa, G. Kumar, M. Chowdhury, A. Krishnamurthy, S. Ratnasamy, and I. Stoica, "Faircloud: sharing the network in cloud computing," in *ACM SIGCOMM 2012*.

[36] K. He, E. Rozner, K. Agarwal, W. Felter, J. Carter, and A. Akella, "Presto: Edge-based load balancing for fast datacenter networks," *ACM SIGCOMM Computer Communication Review, 2015*.

[37] A. Munir, I. A. Qazi, and S. Bin Qaisar, "On achieving low latency in data centers," in *ICC*. IEEE, 2013, pp. 3721–3725.

[38] K. Nagaraj, D. Bharadia, H. Mao, S. Chinchali, M. Alizadeh, and S. Katti, "Numfabric: Fast and flexible bandwidth allocation in datacenters," in *ACM SIGCOMM 2016*.

[39] 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 *ACM SIGCOMM 2009*.

[40] Y. Chen, A. Ganapathi, R. Griffith, and R. Katz, "The case for evaluating mapreduce performance using workload suites," in *IEEE MASCOTS'11*.

[41] Y. Wang, D. Lin, C. Li, J. Zhang, P. Liu, C. Hu, and G. Zhang, "Application driven network: providing on-demand services for applications," in *ACM SIGCOMM 2016 Poster*.