Practical DCB for Improved Data Center Networks

Brent Stephens, Alan L. Cox
Rice University

Ankit Singla
UIUC

John Carter, Colin Dixon, Wesley Felter
IBM Research, Austin

***Abstract*—Storage area networking is driving commodity data center switches to support lossless Ethernet (DCB). Unfortunately, to enable DCB for all traffic on arbitrary network topologies, we must address several problems that can arise in lossless networks, e.g., large buffering delays, unfairness, head of line blocking, and deadlock. We propose TCP-Bolt, a TCP variant that not only addresses the first three problems but reduces flow completion times by as much as 70%. We also introduce a simple, practical deadlock-free routing scheme that eliminates deadlock while achieving aggregate network throughput within 15% of ECMP routing. This small compromise in potential routing capacity is well worth the gains in flow completion time. We note that our results on deadlock-free routing are also of independent interest to the storage area networking community. Further, as our hardware testbed illustrates, these gains are achievable today, without hardware changes to switches or NICs.**

I. Introduction

New developments in commodity Ethernet hardware, driven by Fibre Channel over Ethernet (FCoE), have led to wide support in data center Ethernet switches and NICs for an enhanced Ethernet standard called Data Center Bridging (DCB)[1] [1]. This new standard, in part, augments standard Ethernet with a per-hop flow control protocol that uses "backpressure" to ensure that packets are never dropped due to buffer overflow.

Currently DCB is typically only used, for good reasons, to connect servers to the SAN network via the data network, and even then just for the first-hop from the server to the top-of-rack (ToR) switch, which connects to the SAN. Enabling DCB more broadly, e.g., for data traffic or multi-hop storage traffic, could provide significant benefits, but doing so on arbitrary network topologies introduces many problems, e.g., deadlock, large buffering delays, unfairness, and head of line (HoL) blocking. In this paper we present two solutions to these problems that, when combined, not only enable the use of DCB for all traffic on networks with arbitrary topologies, but also achieve shorter flow completion times than achieved by previous work [2]–[4].

A key goal of a data center network is to provide high throughput for large background tasks and low latency and burst tolerance for user applications and management tasks [3]. With DCB, it is possible further this goal. If packets are never dropped, than TCP slow start can be eliminated, which allows flows to immediately transmit at a multigigabit line rate. DCB also improves flow completion time by preventing incast [5], TCP congestion collapse that occurs in short, barrier synchronized, latency sensitive workloads.

Additionally, the DCB standard is driven by the demand for converged Storage Area Network (SAN) and LAN fabrics, which have significant cost, performance, and management advantages over maintaining separate SANs and LANs [6]. Thus it is desirable to build a network for all traffic with completely converged storage and data networks, not just a bridge between the LAN and SAN, to reduce cost, simplify management, and exploit the new traffic priority classes included with DCB.

Unfortunately, a naive implementation of a network where all traffic exploits DCB and TCP slow start is disabled has significant undesirable consequences. As we demonstrate on real hardware, DCB can lead to increased latency, unfairness in flow rates, head-of-line blocking, and even deadlock, which quickly renders the network unusable until it is reset. However, as we shall show in this paper, none of these are irresolvable problems, and they can be addressed using simple mechanisms already supported on commodity hardware. Additionally, our solutions to these problems do not compromise the improvements we achieved for flow completion times.

To address DCB's problems, we present *TCP-Bolt*, a TCP variant that is designed to achieve shorter flow completion times in data centers while avoiding head-of-line blocking and maintaining near TCP fairness, and a *novel routing algorithm* that uses *edge-disjoint spanning trees* (EDSTs) to prevent deadlock. TCP-Bolt avoids the negative properties of DCB by using DCTCP [3] to maintain low queue occupancy while relying on DCB to prevent throughput collapse due to incasts, which occur on a timescale shorter than an RTT. TCP-Bolt is able to aggressively and optimistically stress the network by eliminating slow start, which results in faster flow completions. By using edge-disjoint spanning trees (EDSTs), our routing algorithm guarantees that there are no cyclic dependencies between routes, and thus no deadlocks. Our EDST routing mechanism forwards over many different EDSTs, so it performs multipath forwarding with a minimal performance impact. Also, since each spanning tree is edge disjoint, all paths used in the network are guaranteed to be isolated.

As motivation for our work, we illustrate the gains achievable on actual DCB-enabled hardware by comparing the performance of a single flow over a simple two-hop network using TCP with an initial congestion window as large as the network's bandwidth-delay product[2] running over DCB (TCP-

[1]These enhancements are also referred to as Converged Enhanced Ethernet (CEE) and Data Center Ethernet (DCE), but we will refer to them as DCB.

[2]Using an initial congestion window equal to or larger than the bandwidth-delay product effectively disables slow start and is the mechanism we use for that feature in TCP-Bolt.

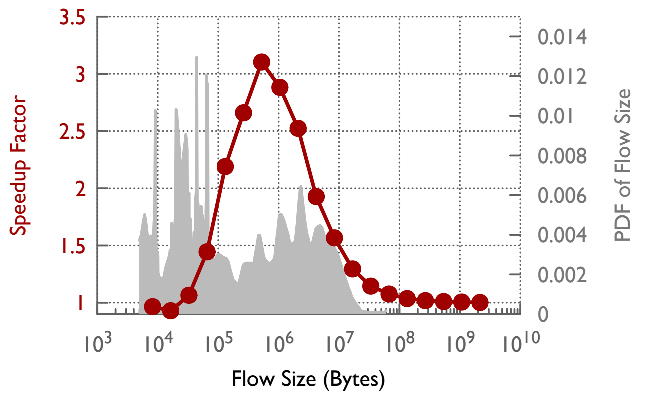

Fig. 1: TCP with a large initial congestion window running over DCB achieves significant speedup with standard TCP over Ethernet.

DCB) against default TCP over normal Ethernet (TCP-Eth). Figure 1 plots a *speedup factor* of how much faster flows of different sizes finish with TCP-DCB compared to TCP-Eth. The speedup factor is superimposed over a probability density function of background flow sizes from a production data center with 6000 servers [3]. For flow sizes between 100 KB and 10 MB, which represent a significant fraction of the flows, this experiment shows that TCP-DCB can achieve speedups of 1.5–3x. This is important because flows in this range have been characterized as being time-sensitive short message flows [3].

While there is a significant body of work [3], [4], [7]–[10] on reducing flow completion times in the data center, none of these past proposals can achieve gains similar to ours *on commodity hardware*. We also note that our proposed modifications to TCP for use over DCB cover most of the functionality that the IEEE 802.1Qau working group [11] is attempting to achieve through hardware modifications beyond DCB. This is significant because in the absence of this functionality, deploying DCB over non-trivial multi-hop topologies runs the risk of *congestion spreading* [12], which is closely related to the head-of-line blocking problem we solve.

Our key contributions are:

- Demonstrating on real hardware both the problems that exist with DCB and solutions that avoid these pitfalls.
- Showing that using DCB to eliminate TCP slow start reduces average completion times for flow sizes between 10 KB and 1 MB by 50 to 70%; 99.9^{th} percentile flow-completion times are reduced by as much as 90%. Additionally, we show that TCP-Bolt outperforms prior work [2] that also eliminates TCP slow start.
- We introduce a simple and efficient deadlock-free EDST routing scheme for commodity Ethernet that makes DCB feasible over irregular topologies. This also makes past work on lossless ethernet more generally applicable.

The remainder of this paper is organized as follows. Section II provides background on lossless Ethernet, including how it works and its implications. We describe the design of TCP-Bolt and the EDST routing algorithm in Section III. In Section IV we describe our experimental methodology, followed by our performance evaluation in Section V. Section VI briefly discusses related work. Finally, we conclude and suggest avenues for possible future research in Section VII.

II. Data Center Bridging 101

DCB is designed to avoid losses caused by buffer overruns. To prevent buffer overrun, a DCB NIC or switch port anticipates when it will not be able to accommodate more data in its buffer[3] and sends a *pause frame* to its directly connected upstream device asking it to wait for a specified amount of time before any further data transmission. Once this pause request expires, either buffer space will be available or the NIC/switch port will renew the request by sending another pause frame. Expiration is typically on the timescale of a few packets worth of transmission time.

In effect, pause frames exert *backpressure* because a persistently paused link will cascade pauses back into the network until, ultimately, traffic sources themselves receive pause frames and stop sending traffic.

A. Implications of Backpressure

DCB's backpressure paradigm has some non-obvious implications that can degrade throughput and latency when used with TCP. We detail these problems here, and discuss our solutions in Section III.

1. Increased queuing (bufferbloat): In the experiments on our physical testbed, we observed that when TCP is run over DCB, round trip times increase from 240 μs to 1240 μs, more than a 5x increase. This occurs because by eliminating packet losses, DCB effectively disables TCP congestion control. If TCP sees no congestion notifications (i.e., losses), its congestion window grows without bound. When congestion occurs, buffers become fully utilized throughout the network before pause frames can propagate, which adds substantial queueing delays, both in the switches and end hosts.

2. Deadlocks in routing: Routing deadlocks arise from packets waiting indefinitely on buffer resources. If a cyclic dependency occurs across a set of buffers, with each buffer waiting on another buffer to have capacity before it can transmit a packet, deadlock results. If routes are not carefully picked, lossless networks such as DCB or InfiniBand can suffer from this problem. Traditional Ethernet avoids deadlocks by dropping packets when buffer space is not available.

A simple example of a deadlock resulting from a cycle of such wait dependencies is shown in Figure 2. The left half of the figure has 3 flows running over 3 switches A, B, and C. The example shows input-port buffers, but using output queues is equivalent. Flow f_{ABC} starts at a host (not shown) attached to A, passes through B, and ends at a host attached to C. Likewise, there are two other flows, f_{BCA} and f_{CAB}. Note that individual routes are loop-free. However, as shown on the right, if the packet at the head of the A's input queue is

[3]A DCB-enabled NIC or switch port must conservatively estimate how much data the upstream device could send before receiving and processing a pause frame, and issue pause frames while it has enough buffer space to accommodate this data.

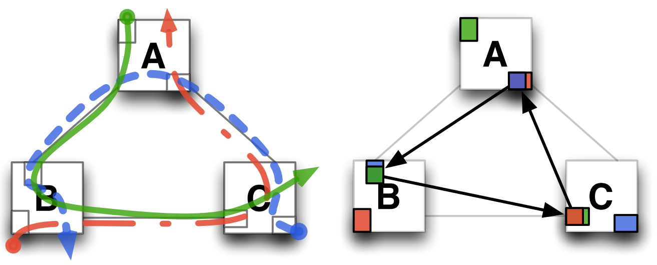

Fig. 2: A cycle of buffer dependencies that could cause a routing deadlock. Note that each individual route is loop-free.

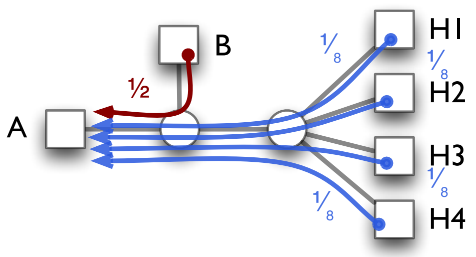

Fig. 3: Because DCB enforces per-port fairness, the $B \rightarrow A$ flow gets half the bandwidth while the other flows share the remainder.

destined for the host at B, the packet at B's input is destined for the host at C, and the one at C's input is destined for the host at A, a cyclic dependency develops between the buffers at A, B, and C: f_{ABC} waits for f_{BCA}, which waits for f_{CAB}, which waits for f_{ABC}. While this simple example may seem easy to avoid, deadlocks can arise from complex interactions involving many flows and switches.

3. Throughput unfairness: Under stable operation, TCP achieves max-min fairness between flows sharing a bottleneck link. This occurs because every host receives and reacts appropriately to congestion notifications, typically packet losses.

In contrast, DCB propagates pause frames hop-by-hop, without any knowledge of the flows that are causing the congestion. Switches do not have per-flow information, so they perform round robin scheduling between competing ports, which can lead to significant unfairness at the flow level. Figure 3 illustrates one such situation. Without DCB, TCP would impose per-flow fairness at the congested link (incoming at A), resulting in each flow receiving $\frac{1}{5}$ of the link. However, when DCB is employed naively, fairness is enforced at a port granularity at the left (congested) switch. The $B \rightarrow A$ flow gets $\frac{1}{2}$ of the congested link while the other four flows share the remainder because of DCB's per-input-port fairness policy.

Figure 4 and Figure 5 present what happens when we implemented the scenario shown in Figure 3 on our hardware testbed using TCP over Ethernet and TCP over DCB, respectively. Hosts $H\{1,2,3,4\}$ are attached to one switch, while hosts A and B are connected to another switch. There is a single link between the two switches. There are five flows: $B \rightarrow A$, and $H\{1,2,3,4\} \rightarrow A$. Two flows, $B \rightarrow A$ and $H1 \rightarrow A$, last the entire experiment. $H2 \rightarrow A$ lasts from $t = 2$–12, $H3 \rightarrow A$ lasts from $t = 4$–10, and $H4 \rightarrow A$ lasts from $t = 6$–8. Figure 4 presents the results for TCP over Ethernet;

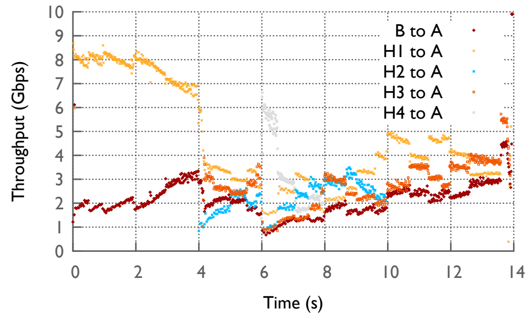

Fig. 4: The throughput of competing normal TCP flows in the topology shown in Figure 3. TCP roughly approximates per-flow fair sharing of the congested link.

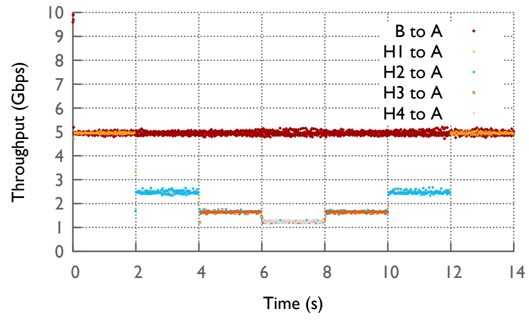

Fig. 5: The throughput of competing normal TCP flows in the topology shown in Figure 3 with DCB enable on the switches. Because DCB provides per-port fair sharing, the A to B flow gets half the bandwidth while the other flows share the remaining bandwidth.

each flow's bandwidth converges to roughly its fair share soon after any flow joins or leaves, but there is substantial noise and jitter. Figure 5 presents the results for TCP over DCB; almost all jitter is eliminated, but bandwidth is shared per-port rather than per-flow. The $B \rightarrow A$ flow gets half of the shared link's capacity, while the flows that share the same input port on the second switch share the remaining capacity uniformly.

4. Head-of-line blocking: DCB issues pause frames on a per-link (or per virtual-lane) granularity. If two flows share a link, they will both be paused even if the downstream path of only one of them is issuing pauses. This scenario is illustrated in Figure 6, where the $A \rightarrow D$ flow suffers head-of-line blocking due to sharing the virtual lane with $A \rightarrow C$, even though its own downstream path is uncongested. Further, as these pause frames are propagated upstream to both flows, periodicities in the system may cause one flow to repeatedly be issued pauses even as the other occupies the buffer each time a few packets are transmitted. This latter anomaly is similar to the 'TCP Outcast' problem [13].

Figure 7 presents the results on our testbed when we ran a set of workloads that induced head of line blocking on a similar configuration (the results for $H3 \rightarrow C$ are elided). In this scenario, the flow from $A \rightarrow C$ exists from time $t = 0{-}10$ and then completes. During the initial ten seconds, the flow from $A \rightarrow D$ is unable to achieve full (10 Gbps) bandwidth because of the head of line blocking induced by the $A \rightarrow C$ flow's congestion. During this period, all flows achieve only

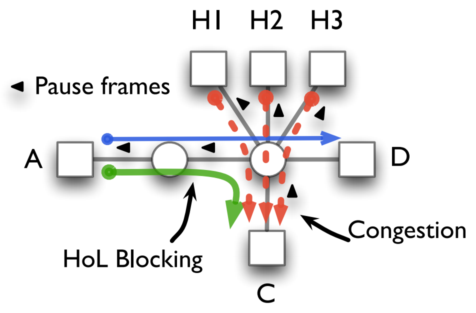

Fig. 6: The $A \rightarrow D$ flow suffers head-of-line blocking due to sharing bottlenecked link with $A \rightarrow C$.

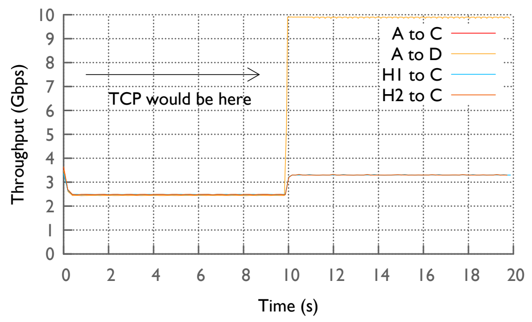

Fig. 7: Throughput of normal TCP flows over the topology seen in Figure 6 with DCB-enabled switches. The $A \rightarrow D$ flow is unable to use the full bandwidth available to it because it is being blocked along with the $A \rightarrow C$ flow.

2.5Gb/sec. When the $A \rightarrow C$ flow stops at time $t = 10$, flow $A \rightarrow D$ ramps up to its full 10 Gbps potential, while the remaining flows evenly share the 10 Gbps link to C.

III. Practical DCB

As described above, DCB can have negative consequences for throughput and latency. In this section, we show that these problems are not irresolvable, and provide solutions to each of the four problems that we discussed.

A. TCP-Bolt

One mechanism solves three of our four problems: increased queuing, throughput unfairness, and head-of-line blocking. The solution is to use ECN [14] in conjunction with DCB. Packets are marked with ECN bits by each switch at a configurable ECN threshold, allowing TCP to function normally even as DCB prevents losses. However, using only ECN results in low throughput, because TCP halves its window upon receipt of a single ECN marked packet. Thus we base TCP-Bolt on DCTCP [3], a recently proposed TCP variant that responds proportionally to the fraction of ECN marked packets. This makes DCTCP's response to congestion more stable and less conservative. DCTCP also reduced buffer occupancy compared to normal TCP, which further improves the completion time of short flows.

Although DCTCP has low flow completion times for short (2–8 KB) flows, we have observed that using DCTCP can increase the tail end of the flow completion times of medium (64 KB–16 MB) flows, a range of flow sizes that still includes

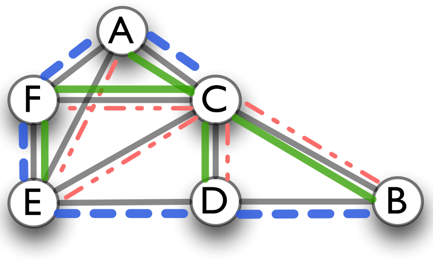

Fig. 8: Three example edge-disjoint spanning trees (EDSTs) in a small network. The EDSTs are in color, the links are in gray.

latency-sensitive flows [3]. This problem could be addressed by using DCTCP with bandwidth-delay product sized congestion windows, but doing so is unsafe and can increase the tail of flow completion times because congestion collapse is still possible. However, by using DCB and a bandwidth-delay product sized congestion window DCTCP in TCP-Bolt, we reduce flow completion times for short and medium flows by eliminating slow start while preventing congestion collapse because DCB eliminates packet loss.

B. DFR for commodity Ethernet

Routing deadlocks are a well known problem in high-performance computing, so a variety of deadlock-free routing algorithms of varying complexity, efficiency, and generality have been developed [15]. However, these routing schemes are targeted at networks where nodes typically have small routing tables due to memory constraints, whereas our switches allow over 100,000 routing entries [16]. In addition Ethernet, supports VLANs, which allow routes on the same path to be distinguished. We exploit these features to build a simple and easy to deploy deadlock-free routing (DFR) algorithm.

We exploit two simple facts in our deadlock-free routing algorithm: (i) shortest-path routing is deadlock-free on trees and (ii) edge-disjoint trees are guaranteed to be isolated from one another. Each route in a tree only goes up or down, never using the same link twice. Further, cyclic dependencies across routes are impossible because the topology has no loops. Breaking up a given network's physical topology into a forest of *edge-disjoint spanning trees* (EDSTs) and using these trees exclusively for routing creates isolated routing sub-graphs on which traffic does not interact because buffer-space is allocated per-edge.

Clearly, the number of such trees that one can find is limited for each topology. We define T as the number of such trees for a given topology. Having a larger number of such trees is desirable, as it provides a greater diversity of routes. Thus, we use *virtual lanes* to increase the number of EDSTs possible. The DCB standard (802.1Qbb Priority Flow Control [17]) creates eight separate virtual lanes per physical link and allows these virtual links to be paused and restarted independently, which makes them deadlock-independent of one another. Thus, we can create as many as $8T$ EDSTs. Note that we need not simply replicate a single set of T trees eight times—that would not yield greater path diversity. Instead, we prefer to find eight distinct sets of roughly T trees.

Figure 8 illustrates using two different sets of EDSTs in a small network. There are three trees shown in dashed blue, dotted red, and solid green lines. The dashed blue and dotted red line trees are edge-disjoint without the need for virtual lanes. However, with the addition of a second virtual lane, we can add the solid green line tree that is not disjoint from the other trees, which increases path diversity.

After obtaining our EDSTs, the DFR algorithm assigns a VLAN identifier to each tree. It then computes shortest path routes on each tree (VLAN) to obtain a next hop entry at each node for each destination (MAC). During routing, either the ingress-switch or the host (if routing information is exposed to hosts) decides which tree is used to reach a particular destination. This choice can be statically encoded in the switch's routing table, or, for multipath routing, can be a per-flow randomized choice from a set of desirable trees . The tree is identified by a VLAN identifier in the packet header, and the virtual layer corresponding to the tree is identified in the QoS bits per the DCB standards.

For an example, consider an unstructured Jellyfish network [18], which is a random r-regular graph (RRG) at the switch layer[4]. In theory one can, with high probability, obtain $\lfloor \frac{r}{2} \rfloor$ EDSTs on a Jellyfish topology [19]. We used a heuristic randomized spanning tree selection process that consistently found roughly this number of EDSTs. With 8 virtual layers, $\sim 8\lfloor \frac{r}{2} \rfloor$ EDSTs are available. By running the randomized heuristic eight times, we typically obtain eight significantly different sets of trees. Consider a Jellyfish network built using 64-port switches, assuming 32 ports are used for the network (i.e., $r = 32$) and the other 32 attach hosts. For such a system, we can obtain 128 distinct EDSTs. Even if all 128 EDSTs were used for routing to all destinations, the routing state for a 1000-switch Jellyfish network (i.e., 32,000 end hosts) would fit in our layer-2 forwarding tables [20].

IV. Experimental Methodology

A. TCP-Bolt

1) Physical Testbed: Our testbed consists of 4 IBM G8264 [20] 48-port, DCB-enabled, 10 Gbps switches, and 20 hosts. For the presented experiments, we have them arranged in a line. The RTT of the network is approximately 240 μs, with most latency coming from the host's network stacks resulting in near-constant RTTs regardless of path hop counts. Each switch has nine megabytes of buffer space for packets, which is shared between all ports on the switch. Each host in the testbed runs Ubuntu 12.04 Linux with the 3.2 kernel. Except when noted, we use the default Linux TCP implementation, TCP Cubic, with an initial congestion window size of 10 MSS. Our DCTCP implementation is based on TCP NewReno and is forward ported to the 3.2 kernel from the implementation made available by the DCTCP authors [3]. For TCP bolt, we modified TCP to disable slow start by setting the initial congestion window to allow for line-rate transmission for our network.

[4]On folded Clos-networks, such as fat-trees, finding EDSTs is trivial.

2) ns-3 TCP Simulations: To consider the effects of larger networks and more complicated congestion dynamics, we also performed simulations in ns-3 [21]. This simulation code is available upon request.

In our simulations, all of the TCP variants are based on TCP NewReno, the default TCP initial congestion window is set to 10 segments, and the bandwidth-delay product congestion window is set to 200 segments. To increase the performance of the baseline TCP, we set the minimum retransmit timeout to the low value of 2ms, as suggested by Vasudevan et al. [22]

We also compare TCP-Bolt against pFabric [2], a recently proposed congestion scheme that uses line-rate initial congestion windows, small priority queues, and a short fixed retransmit timeout. We set the initial pFabric congestion window to 200 and the minimum retransmit timeout to 600 μs, roughly three times the minimum RTT, as recommended by the authors.

We simulated a full bisection bandwidth 3-tier fat-tree topology with 54 hosts. All links in the network operate at 10 Gbps, and the network delays are set so that the RTT is 240 μs. Each port in the network has 225 KB of buffer space unless pFabric is enabled, in which case we use 22.5 KB of buffer per port, similar to the author's suggestions for implementing pFabric. When DCTCP is used, the marking threshold is set to 22.5 KB. These parameters were chosen to emulate our physical testbed.

We use two different load balancing schemes in the simulations. The first is ECMP. The second is packet spraying, suggested by DeTail [4], which routes each packet along a random minimal path. When packet spraying is enabled, we disable TCP Fast Retransmit so that packet reordering does not have an adverse impact on TCP behavior.

3) Workload: To evaluate TCP-Bolt, we use a workload based on the characterization of a production data center [3] that is similar to the workloads used in previous work [4], [7], [23]. Short partition-aggregate jobs, or incasts, arrive according to a Poisson process at each host. For each incast job, the origin host, or aggregator, requests a server request unit (SRU) from 10 other randomly chosen hosts. The SRU is randomly chosen from 2 KB, 4 KB, and 8 KB with equal probability. The average arrival rate is set to 200 incasts per second per host—about 1% of the total network throughput.

In addition to incast flows, each host also averages sending one background flow to another randomly chosen host. These background flows are sized randomly from 64 KB to 32 MB and represent non-query, non-aggregate traffic in the network, which includes both short message and background traffic in a real data center [3].

B. EDST Simulations

For the EDST analysis, we make a number of assumptions. We assume the use of 64-port layer-2 commodity switches that support DCB and have tens or hundreds of thousands of L2 forwarding entries that map (Destination MAC, VLAN)

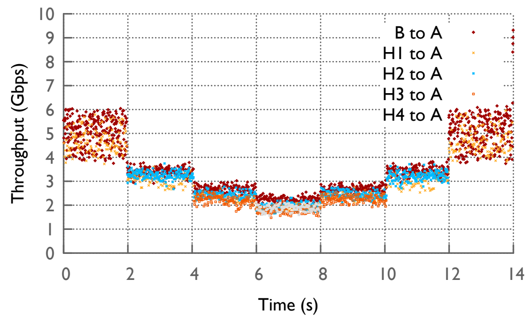

Fig. 9: The throughput of competing TCP-Bolt flows with DCB enabled on the network switches using the topology shown in Fig. 3. DCTCP dynamics keep buffer occupancy low so that per-flow (instead of per-port) fair sharing emerges.

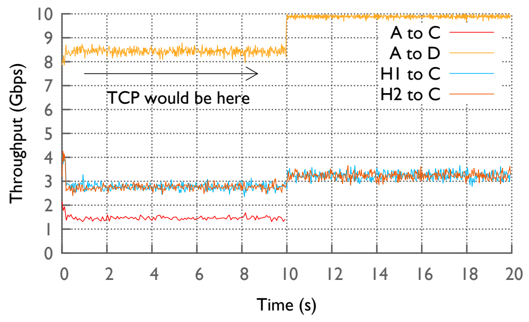

Fig. 10: The throughput of TCP-Bolt flows using the topology show in Fig. 6. DCTCP dynamics prevent any head-of-line blocking, but also cause slight unfairness.

tuples to output ports, e.g., IBM's G8264 [20]. We assume that forwarding tables route on MAC addresses, are computed offline, and are statically populated at switches. We assume that there are 8 virtual lanes per output port. For scalability beyond a few thousand hosts, we assume route aggregation at the ingress switches, which can be achieved without compromising layer-2 semantics [24], [25] and can reduce routing state by 20–1000x depending on the number of hosts attached to a switch and the number of virtual machines per host.

To perform our EDST analysis, we wrote a custom simulator that models flows, omitting the TCP dynamics modeled in the ns-3 simulator. Instead, we assume that each flow immediately receives its max-min fair share bandwidth of the bottleneck link on its path. The simulated workload is a random permutation, with each host acting once as a sender and once as a receiver, transmitting at maximum rate. All results were averaged over 20 runs on the same Jellyfish network.

V. Evaluation

In this section, we evaluate TCP-Bolt and our EDST routing algorithm. First, we show that TCP-Bolt mitigates the potential fairness and head-of-line blocking pitfalls of DCB that were described in Section II. Second, we demonstrate that TCP-Bolt improves flow completion times at scale across the full range of flow sizes using experiments performed on our testbed and ns-3 simulations. Finally, we show that using our novel EDST routing algorithm for DFR only minimally affects performance compared to deadlock-oblivious routing approaches.

A. Solving DCB's Pitfalls

Although DCB's per-hop flow control can cause per-port fairness and head-of-line blocking, TCP-Bolt's use of DCTCP ensures that in the common case switch buffers are not full and thus pause frames are uncommon. This in effect allows us to use pause frames for safety, while using DCTCP's mechanisms to adapt to the correct long-term transmission rate.

1) Fairness: Fig. 9 shows the throughput achieved using TCP-Bolt for flows $B \rightarrow A$ and $H\{1,2,3,4\} \rightarrow A$ on the same configuration (Fig. 3) used for the fairness experiments described in Section II-A. The flows clearly come much closer to fairly sharing available bandwidth than normal TCP either with (Figure 5) or without (Figure 4) DCB. While there is small variance in throughput, the average per-flow throughput is very close to the per-flow fair levels of 5, 3.33, 2.5 and 2 Gbps as the number of competing flows increase from 2 to 5. Also, note that compared to the normal TCP result shown in Figure 4, TCP-Bolt exhibits far less noise.

2) Head-of-Line Blocking: Perhaps even worse, DCB can prevent full utilization of available bandwidth if packets of some otherwise-unencumbered flows are stuck behind packets of flows crossing a bottleneck. Fig. 10 shows the throughput of TCP-Bolt flows in the same configuration (Fig. 6) as the head-of line blocking experiments described in Section II-A.

The DCTCP dynamics of TCP-Bolt allow the flow from $A \rightarrow C$ to once again be able to use all of the spare capacity not claimed by the flow from $A \rightarrow D$. However, the flow from $A \rightarrow D$ now only receives half of its fair share of the link to D. Unfortunately, this is a consequence of DCTCP dynamics. A DCTCP flow will receive a certain rate of ECN-marked packets for each bottleneck link it crosses. Thus, a flow which crosses two such links will receive approximately twice the number ECN signals and thus back off twice as often converging to half of its expected fair-share. While the end-result does not achieve the max-min per-flow fairness that might be desired, it is a significant improvement over the bandwidth wasted by head-of-line blocking.

B. TCP-Bolt Performance

1) Testbed performance: To show that TCP-Bolt works on current commodity hardware, we ran the partition-aggregation workload on our testbed, although we omit the results due to space. As expected, using bandwidth-delay product sized congestion windows with standard TCP hurts the performance of the incast flows, and, surprisingly, did not significantly improve the throughput of background flows. On the other hand, TCP-Bolt consistently achieved half the flow completion time of standard TCP for the incast flows, and for the background flows TCP-Bolt achieved a speedup curve similar to that shown in Figure 1, achieving a 2x speedup at the peak of the curve. However, due to the lack of complicated congestion, both DCTCP and TCP with DCB and slow start disabled matched the performance of TCP-Bolt when run on a topology with a single switch.

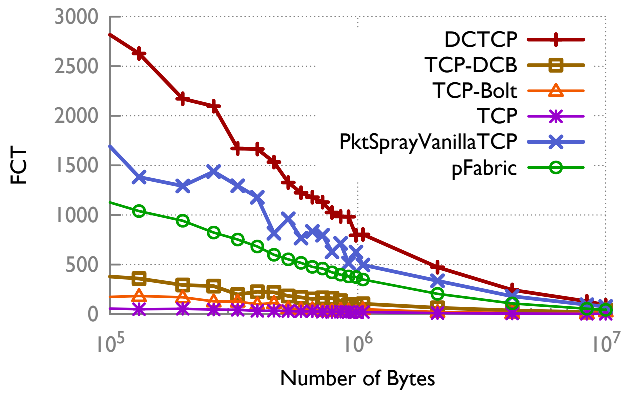

Fig. 11: Simulation comparison of the normalized 99th percentile medium flow completion times for different TCP variants. Variants of TCP and DCTCP wit slow start disabled are omitted for clarity—results with these variants are very similar to TCP's performance.

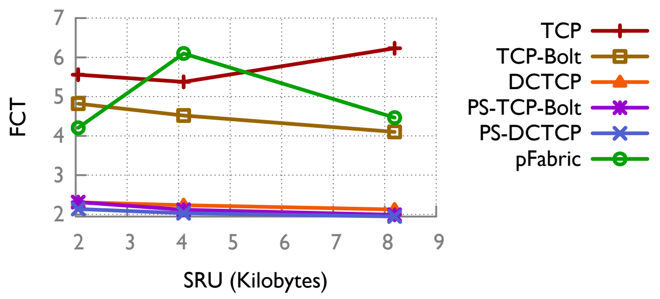

Fig. 12: Simulation comparison of the normalized 99.9th percentile incast completion times for different TCP variants.

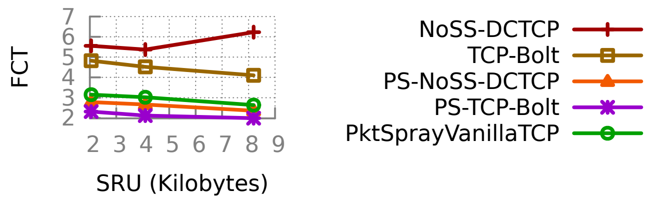

Fig. 13: Comparison of the normalized 99.9th percentile incast completion times for lossy and lossless TCP-Bolt variants.

2) Performance at scale: In networks with larger and more complex topologies, the head-of-line blocking and fairness problems of DCB may be more problematic than on our testbed, and congestion can be more complex. We used ns-3 to consider such a scenario. From our results, we find that TCP-Bolt consistently achieves fast flow completion times across the full range of flow sizes, noting that packet scattering is necessary for achieving the shortest incast flow completion times. TCP-DCB, which prevents TCP congestion control, performs about 20x worse than the best performing algorithm for the short incast flows, and 2x worse for the medium sized flows. DCTCP, on the other hand, achieves short flow completion times, but DCTCP is roughly 10x worse than the best performing variant for the medium sized flows.

There are many individual aspects of TCP-Bolt that all contribute to decreased flow completion times, such as DCTCP, disabling slow start, DCB, and packet scattering. Our results indicate that none of the individual benefits of these elements dominates the others, and that all of them are necessary for short flow completion times.

Fig. 11 shows the 99th percentile flow completion time for the short message and background flows, which represent the range of latency-sensitive background flows that TCP-Bolt benefits. The number of bytes transferred in the flow is on the x-axis, and the y-axis is the flow completion time, which is normalized to the optimal completion time.

We first note that the conservative DCTCP performs consistently more than 10x worse than TCP-Bolt for all of the flow sizes in the figure. DCTCP-DCB has very similar performance to DCTCP and is omitted from the figure. pFabric also does not achieve low flow completion times for short message flows because incasts transmit more data than the switch buffer sizes, so pFabric is almost guaranteed to incur a retransmit timeout.

TCP-DCB, where TCP is unable to perform rate control because packets are never dropped or marked, performs significantly better, but it still more than 2x worse than TCP-Bolt. Although TCP-Bolt has consistently low completion times, both TCP and the TCP variants that disable slow-start outperform TCP-Bolt for background flows. This is because they gain their performance at the expense of the incast flows.

Fig. 12 shows the 99.9th percentile flow completion time for the incast flows. The x-axis is the SRU for the incast, and the y-axis is the normalized flow completion time for the 10 replies to be sent to the aggregator. We first note that TCP-DCB is omitted from this figure because its normalized flow completion time was always greater than 40. As expected, TCP performs the worst of all the presented variants, and TCP without slow start, which is omitted for clarity, matches the performance of TCP. pFabric also performs similarly to TCP. Both TCP-Bolt and DCTCP without slow start, which is also omitted, fail to match the performance of DCTCP, taking about 2x as long instead. Fortunately, as we can see from PS-TCP-Bolt, which stands for TCP-Bolt with packet spraying enabled, increasing the congestion window does not hurt performance if the incast load is balanced over the network.

Due to omissions for clarity, it may not be clear that DCB is necessary for reducing flow completion times instead of just enabling DCTCP with packet spraying and without slow start. Fig. 13 shows the 99.9th percentile flow completion times for the incast flows with the variants of the DCTCP, packet spraying (PktSpray), and disabling slow start (NoSS). From this figure we can see that, with packet spraying, TCP-Bolt provides a 20% improvement over the equivalent TCP variant without lossless Ethernet. Although omitted for space, it is worth noting that packet spraying no slow start DCTCP performs over 5x worse than TCP-Bolt for some of the small background flow sizes. This is because enabling packet spraying requires that TCP fast retransmit be disabled, so

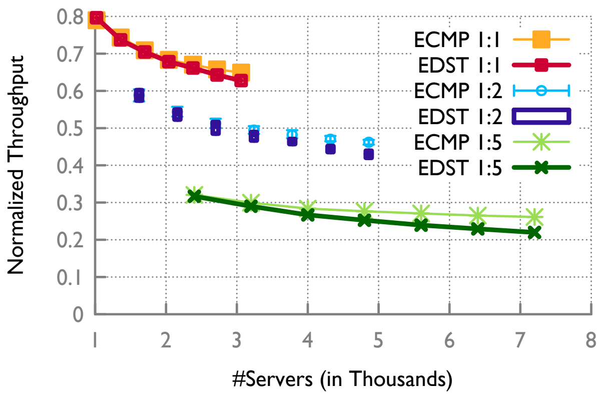

Fig. 14: Comparison with ECMP in terms of aggregate throughput achieved at various sizes.

all packet losses incur a retransmit timeout. When DCB is enabled, packet losses, and subsequently retransmit timeouts, do not happen.

C. Impact of Deadlock Free Routing

While we have demonstrated that DCB significantly improves TCP flow completion times, it is also important to show than enabling DCB on realistic data center topologies does not significantly affect aggregate throughput. To evaluate this issue, we use a custom flow simulator.

Fig. 14 compares the aggregate throughput for random permutation traffic using our EDST deadlock-free routing scheme to using deadlock-oblivious equal cost multiple path (ECMP) routing on Jellyfish networks of increasing sizes. Each curve represents a fixed bisection ratio: 1:1, 1:2, and 1:5 in normalized terms. For the EDST results, we used a routing algorithm similar to ECMP that randomly selects a tree from among the trees that offer the shortest routes. We note that ECMP itself is much higher-performance than traditional Ethernet, which uses a single spanning tree.

Even in the worst case, 7200 servers, our EDST-based DFR scheme achieves at least 85% of ECMP's throughput. The efficiency gap between our EDST scheme and ECMP increases marginally with network size because the number of EDSTs in a topology is a function of network radix and does not increase with network size. In contrast, increasing the number of virtual lanes reduces this efficiency gap. As a result, high performance EDST routing requires more virtual lanes as the network scale increases for a given switch radix, but determining this relationship is left for future work. While there may exist DFR that perform better than our EDST approach on very large networks, we only claim that DFR is feasible to implement over commodity Ethernet without crippling inefficiencies. We leave a comparison across known DFR schemes and the development of better algorithms to future work.

D. Discussion

In Section V we showed that packet spraying is necessary for fast completion times for the incast workload. Although packet spraying sounds exotic, it can easily be implemented on existing hardware and networking stacks that support ECMP by randomizing currently unused Ethernet or IP type fields per packet. Similarly, hosts may easily perform packet scattering by using MPTCP [26], or, when our DFR algorithm is used, hosts may randomize the VLANs that they use per-packet.

Also, even though TCP-Bolt relies on features only found in modern data center switches, it is still possible to use TCP-Bolt and safely communicate with hosts on the global internet. Existing networking stacks already have support for different segment sizes per subnet, and support can also be added for different initial congestion window and ECN settings per subnet. If ECN is disabled and the TCP initial congestion window is left at the default value, then traffic to the outside world will behave as normal.

VI. Related Work

Our discussion of related work focuses on efforts to reduce flow completion times in the data center. We exclude a lengthy discussion of deadlock-free routing literature because our only DFR-related claim is feasibility. For those who are interested, Flich et al. [15] provide a recent survey of deadlock-free routing techniques. We note, however, that Ethernet hardware—large forwarding tables, support for VLANs, and virtual lanes—has allowed us to develop a simple deadlock-free routing algorithm. The same tools are not all available in other settings, e.g., Infiniband and on-chip networks.

Like TCP-Bolt, zOVN [23] observes that enabling DCB can reduce flow completion times. However, their primary focus is enabling DCB support in vSwitches. Further, they use standard TCP variants, which we have shown can perform poorly, and they do not explore the possibility of disabling slow start.

Also like our work, DeTail [4] reduces the tail of flow completion times in data centers with a new network stack that uses DCB and packet spraying to balance network load. However, unlike our work, DeTail continues to use the default TCP initial congestion window on top of DCB and does not address deadlock free routing.

pFabric [2] also abandons TCP slow start, relying on a prioritization scheme to ensure that high priority flows get short flow completion times. However, pFabric can suffer from congestion collapse at high loads and larger flows can potentially suffer from starvation. Further, pFabric requires changes to hardware to support its queuing and prioritization mechanisms.

Complementary to our work are transport protocols that introduce mechanisms to prioritize traffic [7]–[9], [27]. These protocols approach reducing flow completion times by applying either a shortest-flow-first, an earliest-deadline-first, or a least-attained-service schedule.

Remy [28], a new TCP variant, uses machine learning to design TCP congestion control algorithms. We expect that extending Remy to support ECN could automatically generate a congestion control algorithm that outperforms DCTCP, which would improve the performance of TCP-Bolt.

Two other recent proposals for achieving faster flow completion times are notable: TDMA in the data center [29] and HULL [10]. The former uses the pause-frame primitive to implement a TDMA packet schedule for low latency [29]. However, as the authors acknowledge, it is unclear whether an effective centralized controller can be built to handle arbitrary topologies and workloads. HULL [10] reduces latency at the cost of total network throughput but does not attempt to improve overall flow completion times.

VII. Conclusions

In this paper, we demonstrate that it is practical to enable DCB in a fully converged network, despite the many associated pitfalls, and that doing so provides flow completion time benefits. We present both TCP-Bolt, an immediately implementable TCP variant that utilizes DCB, DCTCP, and bandwidth-delay product sized congestion windows to achieve shorter flow completion times, and a novel EDST deadlock-free routing scheme that works with current commodity Ethernet switches.

In our evaluation of TCP-Bolt, we show that using DCB and disabling TCP's conservative initial congestion window on an uncongested network can reduce flow completion times by 50 to 70%. Next, we use physical hardware to demonstrate that using DCTCP with DCB resolves the problems of increased latency, fairness, and Head-of-Line blocking.

We then evaluate the performance of TCP-Bolt against other TCP variants under a realistic workload. Using physical hardware, we demonstrate that TCP-Bolt offers 2x lower flow completion times than TCP. Using ns-3 simulations, we find that TCP-Bolt performs the best of all of the variants, reducing flow completion times by up to 90% compared to DCTCP for medium flow sizes, while simultaneously matching the performance of DCTCP for short, latency-sensitive flows.

Lastly, while lossless networks come with a risk of deadlocks, we present a novel deadlock-free routing scheme that works with current commodity Ethernet switches. This routing scheme reduces aggregate throughput by less than 15%. Combined, these techniques offer the possibility for higher-performance data center networks using commodity switches.

In addition to devising more efficient deadlock-free routing algorithms, this work can be extended in a number of interesting ways. The DCB feature Priority Flow Control [17] can be used to ensure the prioritization of latency-sensitive flows. While we currently use the priority bits for our virtual lanes, they could serve both purposes or we could use extra VLAN tag space for virtual lanes. DCB also provides for an intermediate switch on a path to send congestion notifications [11], which should allow hosts be able to react to congestion faster than waiting for an ACK with an ECN bit set. Lastly, the receipt of pause frames at an ingress switch or host indicates congestion along a path, and this information could be used to shift flows to alternative paths probabilistically.

References

[1] Data Center Bridging Task Group, http://goo.gl/v4evY.

[2] M. Alizadeh, S. Yang, S. Katti, N. McKeown, B. Prabhakar, and S. Schenker, "Deconstructing datacenter packet transport," *HotNets*, 2012.

[3] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "DCTCP: Efficient packet transport for the commoditized data center," in *SIGCOMM*, 2010.

[4] D. Zats, T. Das, P. Mohan, D. Borthakur, and R. H. Katz, "DeTail: Reducing the flow completion time tail in datacenter networks," EECS Department, University of California, Berkeley, Tech. Rep., 2012. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2012/EECS-2012-33.html

[5] A. Phanishayee, E. Krevat, V. Vasudevan, D. G. Andersen, G. R. Ganger, G. A. Gibson, and S. Seshan, "Measurement and analysis of TCP throughput collapse in cluster-based storage systems," in *FAST*, 2008.

[6] Forrester Consulting, "Benefits of SAN/LAN Convergence, 2009." http://goo.gl/gqlDy.

[7] B. Vamanan, J. Hasan, and T. N. Vijaykumar, "Deadline-aware datacenter TCP (D2TCP)," in *Proceedings of the ACM SIGCOMM*, 2012.

[8] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowstron, "Better never than late: Meeting deadlines in datacenter networks," in *SIGCOMM*, 2011.

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

[10] 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," *NSDI*, 2012.

[11] N. Finn, "IEEE 802.1: 802.1Qau - Congestion Notification," http://goo.gl/37kPH, 2005.

[12] M. Alizadeh, B. Atikoglu, A. Kabbani, A. Lakshmikantha, R. Pan, B. Prabhakar, and M. Seaman, "Data center transport mechanisms: Congestion control theory and IEEE standardization," in *Communication, Control, and Computing*, 2008.

[13] P. Prakash, A. Dixit, Y. Hu, and R. Kompella, "The TCP outcast problem: exposing unfairness in data center networks," *NSDI*, 2011.

[14] K. K. Ramakrishnan, S. Floyd, and D. L. Black, "The addition of explicit congestion notification (ECN) to ip," Internet Requests for Comments, RFC Editor, RFC 3168, 2001. [Online]. Available: http://www.rfc-editor.org/rfc/rfc3168.txt

[15] J. Flich, T. Skeie, A. Mejía, O. Lysne, P. López, A. Robles, J. Duato, M. Koibuchi, T. Rokicki, and J. Sancho, "A Survey and Evaluation of Topology Agnostic Deterministic Routing Algorithms," *IEEE Transactions on Parallel and Distributed Systems*, 2011.

[16] B. Stephens, A. Cox, W. Felter, C. Dixon, and J. Carter, "PAST: Scalable ethernet for data centers," in *CoNext*, 2012.

[17] Data Center Bridging Task Group, "Proposal for Priority Based Flow Control," http://goo.gl/wh2Ns.

[18] A. Singla, C.-Y. Hong, L. Popa, and P. B. Godfrey, "Jellyfish: Network Data Centers Randomly," *NSDI*, 2012.

[19] E. Palmer, "On the Spanning Tree Packing Number of a Graph: A Survey," *Discrete Mathematics*, 2001.

[20] "IBM Rackswitch G8264," http://goo.gl/YEFJd.

[21] "The ns-3 discrete-event network simulator," http://www.nsnam.org.

[22] V. Vasudevan, A. Phanishayee, H. Shah, E. Krevat, D. G. Andersen, G. R. Ganger, G. A. Gibson, and B. Mueller, "Safe and Effective Fine-grained TCP Retransmissions for Datacenter Communication," in *Proceedings of ACM SIGCOMM*, Barcelona, Spain, Aug. 2009.

[23] D. Crisan, R. Birke, G. Cressier, C. Minkenberg, and M. Gusat, "Got loss? get zOVN!" in *SIGCOMM*, 2013.

[24] J. Mudigonda, P. Yalagandula, J. Mogul, B. Stiekes, and Y. Pouffary, "NetLord: A Scalable Multi-tenant Network Architecture for Virtualized Datacenters," *SIGCOMM*, 2011.

[25] 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," *SIGCOMM*, 2009.

[26] D. Wischik, C. Raiciu, A. Greenhalgh, and M. Handley, "Design, implementation and evaluation of congestion control for multipath TCP," *NSDI*, 2011.

[27] 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 *INFOCOM*. Turin, Italy: IEEE, April 2013.

[28] K. Winstein and H. Balakrishnan, "Tcp ex machina: Computer-generated congestion control," in *SIGCOMM*, 2013.

[29] B. Vattikonda, G. Porter, A. Vahdat, and A. Snoeren, "Practical TDMA for Datacenter Ethernet," *EuroSys*, 2012.