2011

On the Efficacy of Fine-Grained Traffic Splitting Protocols in Data Center Networks

Advait Dixit
Purdue University, dixit0@purdue.edu

Pawan Prakash
Purdue University, pprakash@purdue.edu

Ramana Rao Kompella
Purdue University, kompella@cs.purdue.edu

Report Number:
11-011

On the Efficacy of Fine-Grained Traffic Splitting Protocols in Data Center Networks

Advait Dixit, Pawan Prakash, Ramana Rao Kompella
Department of Computer Science
Purdue University

***Abstract*—Multi-rooted tree topologies are commonly used to construct high-bandwidth data center network fabrics. In these networks, switches typically rely on equal-cost multipath (ECMP) routing techniques to split traffic across multiple paths, where each flow is routed through one of the available paths, but packets within a flow traverse the same end-to-end path. Unfortunately, since ECMP splits traffic based on flow-granularity, it can cause load imbalance across multiple paths resulting in poor utilization of network resources. More fine-grained traffic splitting techniques are typically not preferred because they can cause packet reordering that can, according to conventional wisdom, lead to severe TCP throughput degradation. In this paper, we revisit this fact in the context of regular data center topologies such as fat-tree architectures. We argue that packet-level traffic splitting, where packets belong to a given flow are sprayed through all available paths, would lead to a better load-balanced network, which in turn leads to significantly more balanced queues and much higher throughput compared to ECMP. We conduct extensive simulations to corroborate this claim.**

I. Introduction

In the recent years, data centers have become the cornerstones of modern computing infrastructure. Many distributed processing applications (*e.g.*, search, social collaboration, high-performance computing) are routinely run in large-scale data centers, that may contain upwards of 100,000 servers. Because of the inherently distributed nature of computation, the network fabric that connects these different servers becomes quite critical in determining the performance of these applications. To scale the data center network to provide the level of connectivity required, most data center network fabrics are organized in the form of a multi-rooted tree topology. Further, they use multipathing between servers so that load is balanced among several alternate paths and the chances of congestion are reduced.

One popular multipathing mechanism used in data centers today is equal-cost multipath (ECMP), where different flows (as identified by the TCP 4-tuple) between a given pair of servers are routed via different paths. The switches basically compute all the available shortest paths using regular routing protocols (such as OSPF) and select a path by hashing the flow key to index into one of the possible next hops. By spreading flows across different paths, ECMP ensures that the load in the network is balanced. However, because all flows are not identical either in their size (number of bytes) or in their duration, this simple strategy is not sufficient to prevent the occurrence of hot-spots in the network. Particularly, two long-lived flows may hash to the same path for a long time, and hence may observe reduced performance, while there may be spare capacity through alternate paths in the network that could have been utilized.

Many recent works have identified the load imbalance that can arise under ECMP and suggested different approaches for making the load more balanced across the different available paths. In particular, Hedera [3] relies on a centralized flow scheduler that periodically obtains information about 'elephant' flows in a dynamic fashion and schedules these elephants so that they do not conflict with each other as much as possible. Another recent approach, called multipath-TCP (MP-TCP) focuses on solving this problem at the TCP-level by breaking a flow into several sub-flows that can be striped across several ECMP paths and the receiver performs the re-assembly of the stream from these sub-flows. By maintaining a separate congestion window across the sub-flows, MP-TCP splits traffic in proportion to the levels of congestion (*i.e.*, a lightly loaded path carries more traffic than a heavily congested flow). Both these schemes, however, have their disadvantages. Hedera requires centralized knowledge making it harder to scale, while MP-TCP requires a complete replacement of TCP at all end-hosts, which may not be feasible in certain environments (*e.g.*, public cloud platforms).

We consider an alternate approach that essentially does not require replacing TCP with a new protocol, or centralized knowledge for scheduling heavy hitters. Yet, this approach is quite simple to implement, and is in fact, already implemented to some extent in many commodity switches (*e.g.*, cisco [1]). The approach is based on packet-level traffic splitting (PLTS) where each packets that belong to a single flow are forwarded across multiple paths that are available between the source and the destination. This idea of spreading packets within a flow across different paths is not novel by itself. However, this approach is not typically preferred because conventional wisdom suggests that TCP will confuse reordered packets with lost packets, that will in turn result in the reduction of the congestion window significantly, thereby losing out on throughput.

In this paper, we revisit this myth surrounding TCP's poor interaction with reordering. Specifically, we argue that if the switches were to spray packets within a given flow across all available paths, and further, if they apply this mechanism to each and every flow in the network, because of the nearly perfect load balancing in the network, the impact on TCP may potentially be quite minimal. By revisiting this conventional

wisdom, our goal in this paper is to suggest that, packet reordering may not be that harmful when all flows in the network uniformly spray packets across all available paths. The simplicity of implementation of this approach is a sufficiently strong motivation to, at the least, study these approaches in more detail, rather than clinging on to the conventional wisdom that TCP interacts poorly with reordering.

Specifically, in this paper, we set out to show using simulations that for several traffic patterns, PLTS achieves much higher overall network throughput compared to ECMP. Further, because potentially most links are similarly loaded, the amount of reordering is quite small. Some flows do suffer slightly, but that is true in ECMP as well due to the possibility of a flow getting routed through a congested path. In that sense, our scheme is not worse than ECMP, but has the effect of making the data center network evenly loaded and more predictable.

Thus, the main contributions of the paper include the following.

- We revisit the conventional wisdom that packet-level traffic splitting is inherently harmful to TCP. Specifically, our observation is grounded on the fact that many popular data center network designs such as the fat-tree, or more generally, multi-rooted tree topologies are symmetric in their architecture, and by spraying packets across different paths leads to a more balanced and predictable network architecture, that interacts well with TCP.
- We propose different variants of the per-packet load balancing algorithms in this paper that exploit the basic idea, but vary depending on the amount of state each of the solution maintains in the routers. We also propose different algorithms based on a hypervisor-based solution that can offer some assistance to mitigate any negative effects of reordering caused by the striping packets within a flow across different packets.
- Using simulations, we compare these various strategies in terms of TCP throughput achieved. We also compare them against ECMP on various network parameters like latency, queue sizes, link utilization etc. Specifically, we show that per-packet load balancing outperforms ECMP in all the dimensions—1.5$\times$ better throughput and 6$\times$ better latency at the 99^{th} percentile.

The rest of the paper is organized as follows. We first start with background and motivation in Section II. We discuss the various packet-level traffic splitting (PLTS) schemes in Section III. In Section IV, we discuss our simulation setup and evaluation results.

II. Background and Motivation

Many data center applications today require significant network bandwidth. For example, MapReduce [11], a commonly used data center application relies on shuffling massive amounts of data from various nodes and hence the job completion time is directly dependent on the available bandwidth between these servers. Large-scale online services are another class of applications that are hosted in massive data centers. For instance, Google, Microsoft, Facebook rely on data centers to host their critical services such as search, recommendation and social collaboration applications and other Web services. These services are typically organized in a three-tier fashion with load balancers, front-end web servers and other backend servers such as SQL servers. Thus, a single client query incurs potentially transferring significant amounts of data between these various tiers. Network performance thus directly impacts the end-to-end query latency, which translates to user satisfaction and ultimately money for the service provider.

While network performance is clearly important, data center operators also want the flexibility of placing any server at any data center node. This gives data center operators more choices in optimizing various resources such as power, storage and computation. Such flexibility, however, means that the data center network design cannot pre-assume a given traffic matrix and optimize the routing and forwarding to that given matrix. In fact, recent papers characterizing data center traffic have found that there exists a tremendous amount of spatial and temporal variation in traffic volumes [15], [19], [7]. Thus, the recent trend towards network fabric designs that can achieve full bi-section bandwidth such as the fat-tree architecture [2].

While in theory the fat-tree architecture provides full bi-section bandwidth, achieving this is hard in practice because it is dependent on the underlying routing scheme. Traditional single-path forwarding mechanisms are inadequate since the full bi-section bandwidth guarantee assumes that all paths that exist between servers can be fully utilized. To mitigate this problem, a simple multipathing algorithm based on equal-cost multipath (ECMP) has been used as the *de facto* routing algorithm. ECMP essentially routes different flows through different paths, so that all the available paths are utilized thus achieving the full bisection bandwidth.

Unfortunately, ECMP only provides coarse-grained load balancing and can lead to many scenarios where paths are not fully balanced. This occurs partly because of the presence of long-lived flows that carry a significant amount of traffic. For example, in [19] study, the authors find that 90% of the traffic volume is actually contained in 10% of the flows (the heavy-hitters). If two long-lived flows hash to the same path, then this can cause significant dip in the performance. For example, we can see the existence of such collisions in Figure 1 where two long flows, A and C, collide along the link $A1 - C1$, while the remaining paths are less loaded. If A and C choose link-disjoint paths, then both would potentially achieve better performance. (In this example, the destinations of all the flows are outside of the pod consisting of the aggregate switches $A1$ and $A2$, and ToR switches $T1$ and $T2$, which required going through the core routers.)

The goal of this paper is to mitigate this problem with the help of a more fine-grained traffic splitting approach, one that involves load-balancing at the granularity of a single packet. Specifically, we start with the basic ECMP traffic splitting that hashes the flow key of a packet to identify the next hop for forwarding. Since all packets for a given flow share the same flow key, this mechanism ensures all packets for that

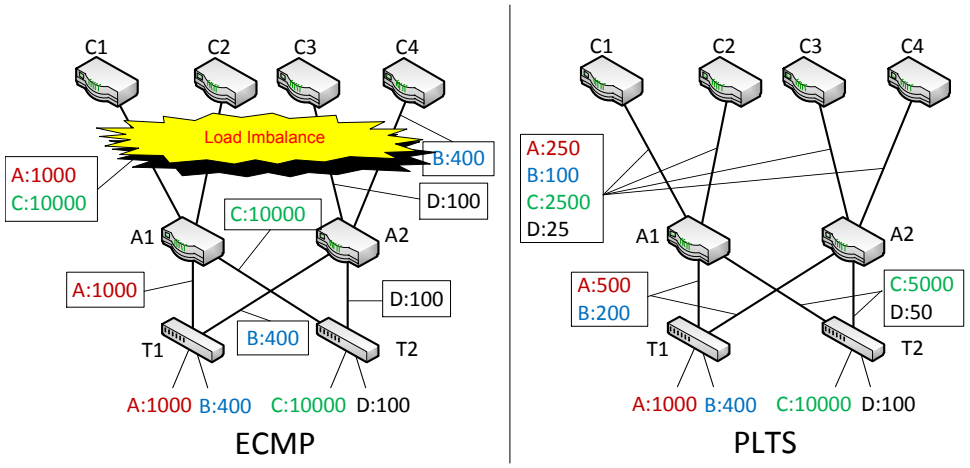


Fig. 1. Motivating example comparing ECMP with packet-level traffic splitting techniques. In ECMP, the four flows $A-D$ are routed by hashing and it could happen that flows A and C traverse the same link while C the link between switches $A2$ and $C2$ is idle. In PLTS, we can observe that all the four links between aggregation switches $A1$, $A2$ and $C1-C4$ are equally loaded with fractions of all the flows across the different paths.

flow traverse the same end-to-end path. Our approach in this paper is to study traffic splitting mechanisms that eliminate this restriction and splits traffic equally among all the paths even within the flow. In the ideal case, we have perfect load balancing as shown in Figure 1 (right side). For the same example as before, we can see now that a quarter of A's and C's traffic is carried on all the four core routers, in addition to a quarter of B' and D's traffic. Within the pod, however, links $T1-A1$ and $T1-A2$ are in one equivalence class, with each carrying half of A's and B's traffic, while the other links are in another equivalence class carrying C's and D's traffic.

PLTS however suffers from the obvious downside of packet reordering on TCP. This is because back-to-back packets within a given flow may now traverse a completely different route to the destination. Depending on the traffic conditions along these individual paths, the packets may arrive out of order at the receiver. While it is well-known that TCP interacts poorly with packet reordering, we argue that these fine-grained traffic splitting schemes improve performance on regular data center network architectures such as the fat-tree. Our intuition is rooted in three basic observations.

1) *Low queue sizes.* Per-packet traffic splitting avoids uneven queue build-up along any path to the destination. This leads to fewer packet drops and uniformly low round-trip times for all flows.
2) *Better load balancing.* All flows between a source (or sources within the same rack) and a destination (or destinations with the same rack) traverse the same set of paths and experience similar network conditions, leading to better and fairer sharing of resources.
3) *Independence from flow-size distribution.* Algorithms that split traffic at a packet granularity are not affected by changes in flow size distribution.

Of course, PLTS will also suffer from reordering due to the potentially different latencies of packets along different paths. This reordering will typically cause the TCP receiver to send duplicate ACKs that may trigger congestion avoidance activity at the TCP sender, causing it to reduce its congestion window significantly thus leading to a reduction in the achieved throughput. We argue, however, that when all flows are split equally between all possible paths, queue lengths along all the paths are typically equally loaded which causes a small number of reorderings, but not enough to cause congestion window reductions and fast retransmits. We discuss these issues in more detail in the next section.

III. Packet-Level Traffic Splitting

In this section, we discuss different techniques that employ packet-level traffic splitting (PLTS) at switches. We also discuss the adverse effects of PLTS on the TCP throughput in the context of fat-tree network topologies. Finally, we discuss different techniques that can be utilized to mitigate the negative effects of PLTS using simple solutions that can be deployed at the receiver TCP stack or within the end-host hypervisor.

A. Mechanisms

We use three simple techniques—random, counter-based and round-robin—to demonstrate the power and limitations of PLTS. All these techniques essentially split traffic within a flow across all available paths, but differ based on the amount of state maintained at the switches. Random picks an arbitrary port at random (among the multiple next hops) to forward a packet, and hence requires no state to implement. At the other end of the spectrum, we discuss a per-flow round robin technique where packets within a flow are transmitted in a round robin fashion. However, it requires remembering the last port used on a per-flow basis making it more complicated than the random. A variant of this algorithm, where packets for the same destination are transmitted in a round-robin fashion, is already implemented in many commercial switches (*e.g.*, Cisco). Intuitively speaking, per-flow round robin achieves better load-balancing than destination-based; hence, we chose this in our list of techniques.

In addition to the above, we also propose a new algorithm that splits traffic based on local port counters. This reduces the amount of state required significantly, but also results in slightly worse load balancing in the network (as we shall see) and consequently, a slight loss in throughput. While this list of algorithms is not meant to be exhaustive, we chose these three schemes as potential candidates since they can be implemented in a rather straight-forward fashion. We discuss these algorithms in more detail in the next few paragraphs.

1) Random path selection: One of the simplest techniques is to randomly forward the packets of a given flow along one of the available paths to the destination. In this scheme called PLTS-Random, for any incoming packet, the switch uses a random number generator to decide the port on which the packet should be forwarded. The forwarding decision about the next incoming packet for the same flow is independent of the previous packets and switch does not need to maintain any flow level state about the port on which the last packet was forwarded.

A uniform random number generator guarantees that all paths will be equally loaded over a long interval of time. However, this scheme may suffer from some short-term imbalance due to the completely random choice. This imbalance may result in momentary increases in queue lengths along one of the paths while leaving some other path underutilized ultimately resulting in less efficient use of the network resources. But this approach is simple to implement; thus, we focus on analyzing its performance in this paper.

2) Counter-based load balancing: We can address the short-term load imbalance due to the random choices by explicitly storing a counter for each port and choose the next-hop that is least loaded. This approach tries to keep the local queue sizes comparable across all the ports at a switch. Specifically, the switch maintains a counter corresponding to each output port. While forwarding a packet, it selects the output port with the smallest counter value and increments the counter by the corresponding packet size (in bytes). Clearly, this technique results in a keeping all the ports as balanced as possible within the switch. For implementing this scheme (referred to as PLTS-Counter), the only extra state required are the counters corresponding to ports of the switch.

PLTS-Counter however can potentially lead to slightly sub-optimal traffic splitting. For example, in Figure 2, we show an example scenario where the load may be slightly imbalanced compared to the ideal. In the example, there are three flows marked A, B and C of equal size. Among the three, B's destination is within the pod. Let us suppose that B's packets and A's packets are perfectly interleaved at the switch $T1$, in which case, A's packets go via $T1-A1$ link and B's packets go via $T1-A2$ link. (In general, the skew may depend on the level of interleaving.) Because of this interleaving, all of A's packets are routed to $A1$ which means only two paths are available for these packets. Meanwhile, since B's destination is within the pod, B's packets do not compete for resources at the links between $A2$ and the core switches $C3$ and $C4$. In this scenario, we can see that first two core links $A1-C1$ and

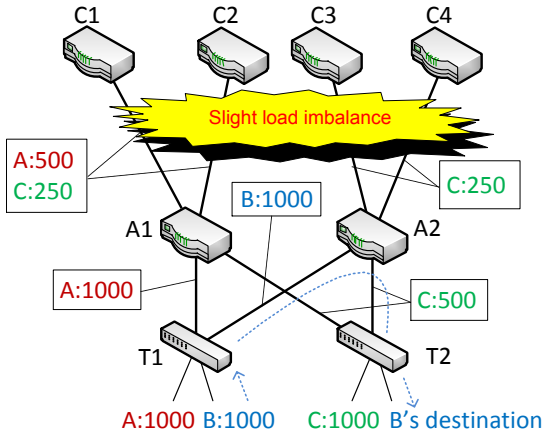


Fig. 2. One scenario where counter-based approach may lead to load imbalance.

$A1-C2$ carry half of A's packets and a quarter of C's packets each, while the other two carry a quarter of C's packets each. In the ideal case where we do perfect traffic splitting of every flow, as the next algorithm achieves, we would observe that each of the core links would carry a quarter of A's packets and a quarter of C's packets, similar to the scenario discussed in Figure 1 example.

3) Round-robin across paths: The last mechanism we discuss, that will address the sub-optimality associated with the previous scheme at the cost of extra state in the router, is the per-flow round robin scheme (referred to as PLTS-RR). In PLTS-RR, each switch uses a round-robin policy to select the port on which the packet corresponding to a particular flow has to be forwarded. To do this, it maintains the next hop taken by the previous packet of that flow. This information needs to be consulted and updated for every packet corresponding to the flow.

One small issue with round-robin based traffic splitting is that packets can suffer from small delays due to synchronization problems. For example, the two incoming flows at a switch can get synchronized in the selection of port using round robin technique. They choose, say, port 1 for their first packet and then in a round robin fashion port 2 for their second packet and so on. But for this to be a major problem, the packet arrival rates of the two flows has to also synchronize. We can solve this problem to some extent by introducing a little bit of randomness in the way round-robin operates. It, however, requires a more detailed analysis that is outside the scope of this paper.

While we have presented this algorithm assuming switches can store one piece of extra state per-flow to remember the last port that was used to forward a packet for that flow, this may not be feasible in practice as the number of flows may be huge. We note however that weaker variants of this algorithm are easily possible. For example, since all servers within a rack are part of the same destination subnet, instead of maintaining the state on a per-flow basis, we could maintain

the port information on a per-prefix basis. Since forwarding may be prefix-based anyways, this requires just one extra counter per entry in the forwarding table. While we do not compare this scheme in this paper, we note that we believe the performance of this variant will lie somewhere between the pure per-flow round robin and the counter-based approach. (In our evaluation, the gap between the two is about 10%.)

B. Adverse effects of packet-level traffic splitting

The packet forwarding techniques described above optimizes the use of network resource available. We discussed the various advantages of packet-level traffic splitting in Section II, but the obvious down-side of fine grained traffic splitting is packet reordering in a flow which can adversely affect TCP performance. Specifically, there are two types of reordering that can happen—forward-path and reverse-path—reordering that we explain next.

1) Forward-Path Reordering: Forward-path reordering is the reordering of data packets, which results in the generation of duplicate acknowledgements (DUPACKs). If the TCP receiver receives packets out of order, it would generate a DUPACK that correspond to the last packet received in order. While eventually the receiver would obtain all the packets and would advance the acknowledgement number in ACKs, the sender upon seeing three DUPACKs assumes that there was a packet loss in the network because the window is too high. Correspondingly, it would perform fast-retransmit and reduce the window by half (other variants may reduce it by a different amount). Unfortunately, both these actions would not have actually been necessary and in fact cause two major problems. First, the unnecessary retransmissions would waste precious bandwidth resources in the network. Second, and perhaps more importantly, the reduction in the congestion window causes TCP sender to be not as aggressive and thus would not fully utilize the available bandwidth. Both these ill-effects are the main reason why ECMP (and other mechanisms such as Hedera [3]) did not disturb the ordering of packets within a given flow.

In addition to these direct effects, there are also two indirect effects. First, RTT estimators for flows not using TCP timestamp option can, at least in theory, suffer from packet reordering. But as discussed in [6], it turns out that this is not a big problem in practice. Second, the TCP receiver is forced to buffer out-of-order data packets and data is released to the application in bursts. Of course, buffering itself is not such a big deal as end hosts typically are reasonably well-provisioned in terms of memory. Bursty release of data to the application, however, can be a bit of a concern for multimedia applications (which use UDP anyway precisely for this reason), but not so much for bulk transfer applications.

2) Reverse-Path Reordering: If ACKs are reordered and one ACK arrives early and acknowledges a large amount of data, TCP sender is likely to transmit a series of new segments at once. So we may see bursts of TCP segments from the sender. One potential fix to this problem is that we can choose to do ECMP-based forwarding for ACK packets. On the other hand, there may not be a need for such a fix since back-to-back packets may now take completely different routes; so, the bursts are likely to be spread out throughout the network.

In the context of data center networks, PLTS scheme ensures even-sized queues at the switches (see section IV), thus keeping the network free of any hot-spots. The packets of a given flow travelling on parallel paths to the receiver would encounter similar queue sizes along their paths. Thus, even if some of the packets get reordered, not all of them trigger the congestion control mechanism of TCP. The ill-effects of packet reordering, which can be pretty severe in case of the Internet, are relatively quite mild in data center networks. The latencies observed across multiple paths are similar, the receiver does not have to buffer a large number of out-of-order packets and not all the reordered packets lead to fast retransmission. In order to further reduce the adverse effects of PLTS, we discuss a few potential techniques that can be implemented at the end host.

C. Reducing spurious fast retransmissions

We can potentially mitigate the ill-effects of spurious fast retransmits and improve TCP's performance if we can somehow detect and suppress spurious DUPACKs that are generated due to reordering. While some research has been done to distinguish packet reordering from a packet loss event [8], the problem remains considerably difficult in the general setting. We can use some simple mechanisms to prevent the TCP sender from reacting to DUPACKs too aggressively, since if we implement PLTS, most DUPACKs will be due to transient reordering. We discuss two techniques—the first involves changes in the receiver TCP stack, while the second focusses on a hypervisor-based solution at the end host.

1) Adjusting DUPACK threshold for Fast Retransmission: One straightforward technique, that was studied in prior work RR-TCP[24] and TCP-PR [9], we can employ is to dynamically adjust the TCP DUPACK threshold. In RR-TCP, the receiver uses TCP-DSACK [13] to report the receipt of duplicate segments to the TCP sender. Even without TCP-DSACK, a TCP sender can assume that a fast retransmission is spurious if it receives an acknowledgement for a retransmitted segment in less than a fraction RTT_{min} [5]. In either case, when a TCP sender detects a spurious retransmission, it can undo the congestion window reduction that occurred in the fast recovery phase. [8] lists many modifications to TCP to make it robust to packet reordering that can also be used in our setting. In data center networks, where the effect of packet reordering is not so severe, this simple adjustment to the DUPACK threshold can help mitigate most of the ill-effects due to PLTS.

2) An agent to filter DUPACKs: In a public cloud environment, an end host runs multiple guest virtual machines (VM). These VMs are rented out to tenants, who may be using customized protocols for end to end communication. For PLTS to be practical in such a setting, we cannot make any assumptions about the higher layer protocols. Fortunately, the hypervisor running below the VMs and the entire data

center network is administered by a central entity, and provides the opportunity to design and deploy techniques that require interactions between intermediate switches and end hosts' hypervisors to counter the effects of packet reordering. Such a technique would be completely transparent to the guest VMs belonging to clients.

We assume the presence of an agent that runs in the hypervisor layer (either sender's or receiver's) that can inspect every incoming packet. The agent essentially drops, modifies or delays duplicate acknowledgements to control the TCP stack's view of packet reordering. If the agent blindly stops all the DUPACKs, then it prevents the TCP fast recovery in case of an actual packet loss. This can cause TCP timeout resulting in a drastic decrease in TCP throughput—a situation we want to avoid. Thus, the agent essentially needs to store information about the ACK packet last seen for each flow, using which it can easily detect the DUPACKs in a flow. The actions of the agent are dependent on whether the switches in the network employ RED [12] and ECN [21] or use droptail queueing policy.

RED/ECN queue management policy. In a network where routers use RED and ECN for congestion control, the TCP receiver echos any network congestion back to the sender by copying the CE (Congestion Encountered) bit of the data packet onto the corresponding acknowledgement packet as the ECE (ECN-Echo) bit. In case there is congestion, the TCP sender acknowledges it by setting the CWR (Congestion Window Reduced) bit in data packets. The filtering agent can then use it to differentiate a packet reordering event from a packet loss event. The agent can keep dropping DUPACKs until it sees a packet with the ECE bit set. This behavior assumes that any packet drop would happen only after some kind of congestion in the network. If the agent receives a DUPACK with ECE bit set, then it allows it go through so that TCP can take the required action necessitated by the congestion in the network. It would keep on allowing the DUPACKs until their ECE bit are set. We implemented this simple agent, but found that while it performs much better compared to ECMP, it does not yield any significant benefits beyond the basic PLTS schemes. Perhaps other techniques that can better exploit these markings may exist; a more detailed study however is outside the scope of this paper.

DropTail queue management policy. In case of drop tail queue management policy, it is difficult to differentiate a packet reordering from a packet loss event. However, we can use a scheme similar to RR-TCP [24] to adjust the DUPACK threshold dynamically. Specifically, the agent maintains multiple counters corresponding to different numbers of consecutive sequences of DUPACKs (*i.e.*, number of times it received 3, 4 or 5 consecutive DUPACKs). When the agent receives the first two consecutive DUPACKs, it delivers them to the guest VM. It drops the following DUPACKs until the length of the current sequence of consecutive DUPACKs reaches the top 90^{th} percentile of sequence lengths seen earlier. When this DUPACK is delivered to the VM, it triggers a fast retransmission at the TCP sender. The algorithm has the effect of avoiding 90% fast retransmissions and allowing only those that would have resulted from the longest sequence of consecutive duplicate acknowledgements. PLTS enabled with this simple agent gives slightly (by about 5%) better performance than PLTS alone as shown in section IV.

IV. Evaluation

We now evaluate the various packet-level traffic splitting schemes we have discussed in Section III. Specifically, our evaluation goals are three-fold. First, we wish to compare the various variants of PLTS (PLTS-Random, PLTS-Counter and PLTS-RR) with other prior schemes such as ECMP and MPTCP [14] in terms of their overall throughput achieved. Second, we wish to understand other characteristics such as packet latencies, queue length and fairness properties of PLTS. Finally, we want to study the impact of different traffic patterns and packet sizes on PLTS.

A. Experimental setup

Most of our experiments in this paper are based on simulations using QualNet [23], a packet-level discrete event simulator. Given the large number of TCP variants in the wild, we chose to focus on one of the most widely used TCP variants, namely NewReno. We used 1MB TCP receive and send windows. Typically, operating systems reserve a few hundred kilobytes for TCP windows. However, we inflated this value slightly so that the TCP window sizes do not limit the network throughput. This ensures that changes in the TCP congestion window size always affect the network throughput. We summarize the most significant simulation parameters for TCP in Table I.

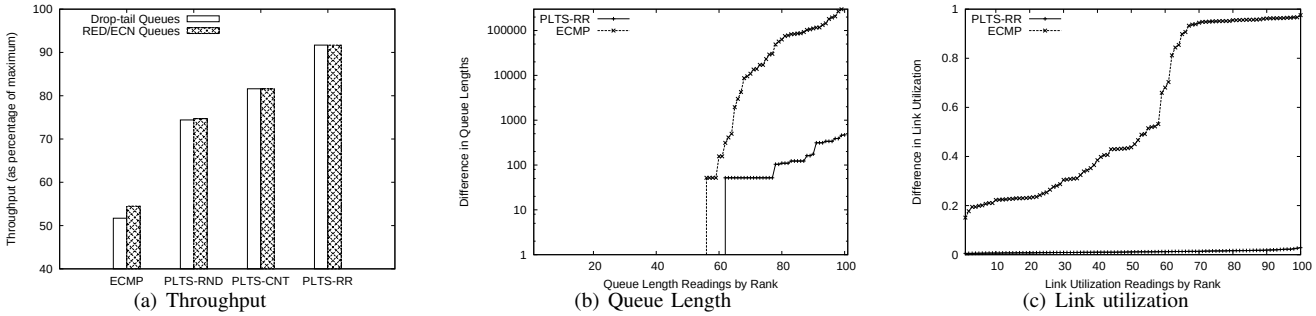
TABLE I
Simulation Parameters.

Parameter	Value
Algorithm	NewReno
Send Window	1MB
Receive Window	1MB
Delayed ACKs	Disabled
Nagle's Algorithm	Disabled
Link Bandwidth	100Mbps
Output Port Buffers	512KB

For evaluating PLTS, we focus mainly on fat-tree architecture. However, the results shown should hold good in any topology with multiple equal-cost paths between end hosts. We do note that topologies BCube provide a large number of paths between end hosts, but some paths may traverse more links than others and hence their cost may be different.

B. Performance comparison between PLTS and ECMP

We compare PLTS and ECMP with respect to different network parameters. The goal of these comparisons is to identify the benefits of PLTS technique over ECMP in different dimensions which are of interest to network operators.



(a) Throughput (b) Queue Length (c) Link utilization

Fig. 3. Throughput comparison across different routing mechanisms. We also show the queue length and link utilization comparisons across PLTS and ECMP.

1) Throughput comparison: We first compare the throughput of PLTS compared to other schemes such as ECMP and MP-TCP. For this evaluation, we simulate a 6-pod fat tree topology with 54 end-hosts and create 27 FTP applications which run for the entire duration of the simulation. The end hosts for each FTP application are chosen at random while ensuring that each end host executes either one FTP server or client. Since a fat-tree topology offers full bisection bandwidth, ideally each FTP application should be able to transfer data at full link capacity. We have evaluated the TCP throughputs with two queue management policies at the routers—DropTail and RED with ECN. Figure 3(a) demonstrates that the throughput values are almost identical for PLTS in both the cases. Henceforth, for brevity reasons, all the experimental results for PLTS are shown with DropTail policy. The results for RED with ECN policy are similar.

Figure 3(a) shows the average throughput observed by FTP applications, under different schemes, as a percentage of the ideal throughput. The throughput achieved by ECMP is similar to that reported in [20]. The low average throughput in case of ECMP-based forwarding can be attributed to the fact that two or more ECMP flows may be forward over the same core link which becomes a bottleneck. And for the entire flow duration that link remains the host spot in the network while leaving other links underutilized. Due to static allocation of paths in ECMP, if some of the flows are unlucky and are routed through a congested link, then they suffer permanently for the entire duration resulting in a poor throughput.

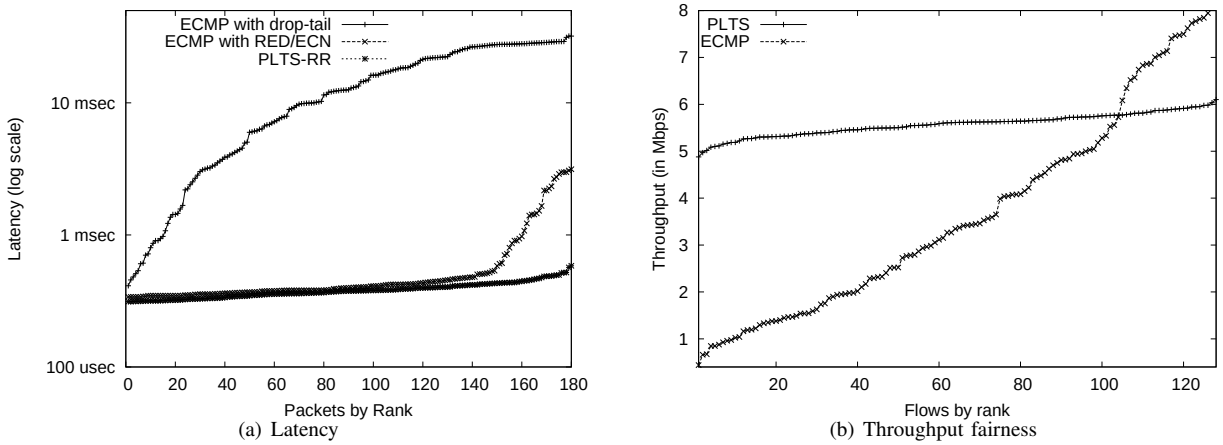
All the three PLTS techniques achieve better throughput than ECMP as shown in Figure 3(a). [14] reports that MP-TCP achieves almost 90% utilization for the same experimental setup, which is comparable to what the PLTS-RR achieves. Among packet-level traffic splitting, PLTS-RND attains the least throughput because skews in short sequences of random numbers increases variability in latencies across different path. This leads to more packet reordering and fast retransmissions. Hence, we will not explore PLTS-RND further. PLTS-CNT attains around 81% of the ideal throughput. In PLTS-CNT technique, packets from a given flow may not be scattered across all the paths always. It tries to make the best *local* decision by choosing the smallest queue-size path at the router. Depending on queue sizes at intermediate routers, packets of a particular flow may momentarily favor one of the available paths. After some time, changes in network conditions can change the distribution of packets across available paths. So we observe small oscillations among the subset of paths a flow is forwarded to.

PLTS-RR ensures that all the paths between source and destinations are utilized. In case the round robin policy is used for all the flows at all the routers, the traffic load in similar across all the paths and packets for a given flow are believed to encounter similar queues and latencies across multiple parallel paths. This evenness and uniformity in the network usage prevents any hot spots in network and helps all the flows to fully utilize the network capacity. Figure 3(a) confirms this behavior and PLTS-RR is able to achieve almost 92% of the ideal throughput.

2) Comparison of queue lengths and link utilization: It is well known that ECMP based forwarding mechanism can result in uneven load on network components [3], [14]. Even in case of a uniform traffic matrix, different part of the network could be very differently loaded. The result is uneven queue sizes and link utilization across the network. In contrast, as our intuition suggests, we expect that PLTS should keep the queue sizes and utilization balanced throughout the network, which we evaluate next.

In a 6-pod fat tree data center network, carrying 54 active flows in the network, we isolate a total of 9 links originating from all the 9 core switches and connecting them to a particular pod of the fat tree. We want to analyze the variation in queue lengths and link utilizations of these links which are bringing traffic into this pod. We divide an interval of 1 second into hundred 10 milliseconds sub-intervals. At the end of each sub-intervals, we record the queue lengths and the link utilization values across all the 9 links selected above. For both the metrics, we take the difference between the maximum and minimum of the 9 values for a sub-interval. This difference gives us the maximum variation in queue lengths and link utilizations for that sub-interval. At the end of 1 second interval, we sort these 100 values and plot them to demonstrate the variability of these metrics over a period of 1 second.

Figure 3(b) demonstrates that in case of PLTS, the difference between the queue lengths of longest and the smallest



(a) Latency (b) Throughput fairness

Fig. 4. Packet Latency values for a sample of 180 packets in ECMP versus PLTS. We also compare the throughput achieved by different flows for fairness.

queue never exceeded 500 bytes in the analysis interval of 1 second. In ECMP, there are times when the difference in queue sizes has increased all the way to 100KB. PLTS does not only has smaller average queue length but it also has lesser variation across multiple queues under consideration. Figure 3(c) plots the variability in link utilization across the 9 links. It demonstrate a similar behavior where ECMP suffers from huge difference of workload across different links while PLTS maintains uniformity. The graphs support our claim that packets will experience similar queueing delays along all paths. The small difference in link utilizations indicates that PLTS is less prone to hot-spots than ECMP.

3) Comparison of latencies: In the same setup as described in the previous section, we compare the latencies experienced by a sample of 180 packets of a long-lived flow between two hosts in different pods. We measure the latencies between the top-of-the-rack switches connected to the end points of the flow. This isolates the latencies experienced in the network from the latency at the access links. We compare the latencies experienced by the sampled packets in three scenarios: PLTS-RR and ECMP with drop-tail queues and ECMP with RED/ECN active queue management. The latencies for the sampled packets are ranked and plotted in figure 4(a).

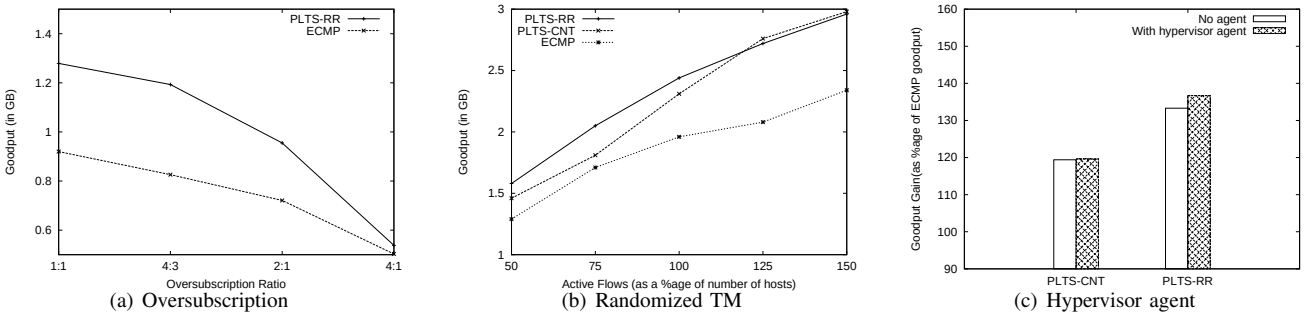
For ECMP, the latency values for drop-tail policy are worse than the latency values of RED/ECN (as drop-tail allows large queue build ups). PLTS with the drop-tail policy shows significantly better latency values experienced by the sampled packets as compared to ECMP with RED/ECN. While for most of the packets the latency values for the two situations are comparable, a packet in PLTS experiences only one-third of the packet latency of ECMP with RED/ECN at the 90^{th} percentile and one-sixth of the packet latency at the 99^{th} percentile. In most of the partition-aggregate type workload today [4], the latency at the 99^{th} percentile is of utmost importance. It decides how well a service can perform within SLA (Service Level Agreements) up time [4], which ultimately would affect customer experience in public cloud. The results indicate the opportunities which PLTS could provide to significantly improve the operation of public clouds.

4) Throughput fairness: In PLTS, flows achieve better TCP throughput on an average as compared to flows with ECMP forwarding. But we also need to analyze how the increased throughput performance is shared among all these flows. Given the uniform nature of PLTS, we expect high fairness among different flows simultaneously enabled in the data center network. We show that the gains in throughput offered by PLTS are evenly distributed among all flows in the network. In an 8-pod fat tree network with 128 FTP applications, we analyze the throughput achieved by each individual flow over the duration of the simulation. Figure 4(b) shows the throughput observed by all flows. Flow throughputs for ECMP show a huge variability as compared to those of PLTS. There is an order of magnitude of difference between the highest and lowest throughput seen by ECMP. So the number of *unlucky* and *lucky* flows are large in case of ECMP forwarding. In case of PLTS we see high degree of fairness among all the flows. Hence, PLTS provides more predictability in flow throughput.

C. Effect of Over Subscription

Most of the evaluation results have shown that PLTS outperforms ECMP in all dimensions in a data center network with full bisection bandwidth. In this section, we compare the two techniques in more constrained networks with higher oversubscription ratios. iAn oversubscription ratio of 4:1 at aggregate switches means that the uplink capacity of aggregate switches to the core are four times less than the uplink capacity of top of rack switches to the aggregate switches. We study the effect of oversubscription on the performance of ECMP and PLTS.

We begin with an 8-pod fat tree (which has 16 core switches and 128 end hosts) topology which has 1:1 oversubscription ratio, meaning that we have a full fat-tree network. In order to constrain this network topology, we remove 4 core switches (out of the 16 core switches) and the corresponding links to get an oversubscription ratio of 4:3 at the aggregate switches. We further remove 4 core nodes to get a ratio of 2:1 and so on. We simulate 128 FTP applications in the network. Each



(a) Oversubscription (b) Randomized TM (c) Hypervisor agent

Fig. 5. Performance of PLTS under cases of over-subscribed data center network, randomized traffic matrix and with different packet sizes.

end host act as an FTP server for one application and client for another. The FTP clients and servers are paired up at random (while ensuring that they are in different pods). The FTP applications run for the entire duration of the simulation. Figure 5(a) shows the goodput (total bytes transferred) by PLTS-RR and ECMP. With a full bisection bandwidth, PLTS achieves almost 1.5 times the goodput offered by ECMP. However, as the oversubscription ratio increases, the capacity in the core of as well as the number of parallel paths in the data center decreases. The gain of PLTS over ECMP begins to decrease when fewer paths are available. But even in case of 4:1 oversubscription PLTS performs better.

D. Goodput comparison with randomized traffic matrix

In most of the evaluation experiments, we have used a uniform traffic load where each end host is sending or receiving or doing both at a time. This is not the case in real traffic load, where some of the end hosts may not be involved in communication at all, or some have multiple TCP connections simultaneously. In order to simulate such behavior we did an experiment similar to one described in [14]. To simulate a more realistic scenario, we started a fixed number of FTP flows between randomly chosen hosts in a 6-pod fat tree data center network. Flow sizes were chosen from the distribution described in [15], which are representative of real flows in data center networks. When a flow finishes, it was replaced with another flow between two randomly chosen hosts. The experiment was run for 10 seconds and we calculate the total number of bytes transferred by all the flows in the entire duration.

Figure 5(b) shows the total number of bytes transferred by the PLTS-CNT and PLTS-RR technique. The X-axis is the number of active flows (which remains constant throughout the simulation) as a fraction of the number of hosts. We observe that PLTS outperforms ECMP consistently, and more so in case there are larger number of active flows in the network. Since it was a different kind of traffic workload, we also want to compare the two PLTS techniques. PLTS-CNT and PLTS-RR have comparable performance in most of the cases. Note that since the traffic in each experimental run is created randomly, the performance of a specific technique would also depend on the kind of traffic load it has to work with. So the general conclusion from this experiment is that PLTS performs better than ECMP with both RR and CNT schemes having comparable performance.

E. Effect of variable packet sizes

We do most of the evaluation by simulating FTP applications in which all flows have same packet sizes (though data and ACK packets differ in size). We also get near optimal performance of PLTS in such scenarios without needing any assistance from the agent. In this section, we vary the packet sizes across different flows and evaluate the performance of PLTS with and without the agent. Since we are introducing variability in packet sizes, we expect more variability in latencies experienced by packets travelling across different paths. A flow may experience more reordering and in such a situation we would like to evaluate the performance of our agent which tries to handle packet reordering in a given flow.

To vary packet size in the network, we set the MSS for every flow (total 54 active flows in a 6-pod fat tree) randomly between 64 to 1500 bytes. Figure 5(c) shows goodput gain observed in this simulation environment. Even with the increased variability, PLTS still outperforms ECMP. PLTS-RR without any agent is approximately 1.35 times better. In such a scheme, an agent, which dynamically adjusts the threshold of duplicate acknowledgement for fast retransmission, helps to improve the performance of PLTS.

V. Related Work

The most related to our work are those mechanisms that rely on flow-level traffic splitting such as ECMP and Hedera [3]. Mahout [10] is a recent scheme that uses end-host mechanisms to identify elephants, and uses flow scheduling schemes similar to Hedera. BCube [17] proposes a server-centric network architecture and source routing mechanism for selecting paths for flows. When a host needs to route a new flow, it probes multiple paths to the destination and selects the one with the highest available bandwidth. Techniques like Hedera, Mahout and BCube which select a path for a flow based on current network conditions suffer from a common problem: When network conditions change over time, the selected path may no longer be the optimal one. To overcome this problem, they periodically re-execute their path selection algorithm. VL2[15] and Monsoon[16] propose using Valiant

Load Balancing (VLB) at a per-flow granularity, but they too do not split an individual flow across multiple paths.

Two research efforts propose traffic splitting at a sub-flow granularity. The first effort is MPTCP[14] splits a TCP flow into multiple flows at the end hosts. [20] evaluates the performance benefits of using MPTCP for load balancing in a data center network. The ECMP protocol running at each router may route each TCP sub-flow over different paths in the network. The receiving end host aggregates the TCP sub-flows and resequences packets. The second effort, although in the context of the Internet, is FLARE [22]. FLARE exploits the inherent burstiness of TCP flows to break up a flow into bursts called flowlets, and route each flowlet along a different path to the destination. However, FLARE requires each router to maintain some per-flow state and estimate the latency to the destination. We did experiment with some simple variants of FLARE, such as keeping a small number of packets of a flow go through the same path. But we observed that any simple variant of FLARE does not achieve as good a throughput as our PLTS, since bursts of packets may actually lead to disparity in queue lengths across different paths, which in turn causes much more packet reordering and reduction in throughput.

VI. Conclusion

With many data center applications requiring large amount of intra-cluster bandwidth, there is a lot of interest in designing network fabrics that provide full bisection bandwidth. Multi-rooted tree topologies have emerged as the architecture of choice for many such environments. Unfortunately, however, default multipath routing protocols such as ECMP can lead to significant load imbalance resulting in underutilizing the available network resources. Previous solutions such as Hedera and MP-TCP address this problem to some extent, but are typically complicated to implement or deploy. In contrast, we show that simple packet-level traffic splitting mechanims can, somewhat surprisingly, yield significant benefits in keeping the network load balanced resulting in better overall utilization, despite the well-known fact that TCP interacts poorly with reordered packets. These schemes are also readily implementable and are of low complexity making them an appealing alternative to ECMP and other complicated mechanisms. While more work needs to be done, we believe that the myth that TCP interacts poorly with reordering, while may be true in more general settings, does not seem to hold true in regular data center topologies.

References

[1] "Per packet load balancing," http://www.cisco.com/en/US/docs/ios/12_0s/feature/guide/pplb.html.

[2] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat, "A scalable, commodity data center network architecture," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 4, pp. 63–74, 2008.

[3] Mohammad Al-fares, Sivasankar Radhakrishnan, Barath Raghavan, Nelson Huang, and Amin Vahdat, "Hedera: Dynamic flow scheduling for data center networks," in *In Proc. of Networked Systems Design and Implementation (NSDI) Symposium*, 2010.

[4] Mohammad Alizadeh, Albert Greenberg, David A. Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan, "Data Center TCP (DCTCP)," in *ACM SIGCOMM'2010*, Aug. 2010.

[5] Mark Allman and Vern Paxson, "On estimating end-to-end network path properties," 1999.

[6] Jon C. R. Bennett, Craig Partridge, and Nicholas Shectman, "Packet reordering is not pathological network behavior," *IEEE/ACM Trans. Netw.*, vol. 7, pp. 789–798, December 1999.

[7] Theophilus Benson, Ashok Anand, Aditya Akella, and Ming Zhang, "Understanding data center traffic characteristics," in *WREN*, 2009, pp. 65–72.

[8] Ethan Blanton and Mark Allman, "On making tcp more robust to packet reordering," *ACM Computer Communication Review*, vol. 32, pp. 2002, 2002.

[9] Stephan Bohacek, João P. Hespanha, Junsoo Lee, Chansook Lim, and Katia Obraczka, "Tcp-pr: Tcp for persistent packet reorderin," in *ICDCS*, 2003, pp. 222–.

[10] Andrew Curtis, Wonho Kim, and Praveen Yalagandula, "Mahout: Low-overhead datacenter traffic management using end-host-based elephant detection," in *Proc. of IEEE Infocom*, Apr. 2011.

[11] Jeffrey Dean and Sanjay Ghemawat, "Mapreduce: Simplified data processing on large clusters," December 2004, pp. 137–150.

[12] Sally Floyd and Van Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Trans. Netw.*, vol. 1, pp. 397–413, August 1993.

[13] Sally Floyd, Jamshid Mahdavi, Matt Mathis, and Matthew Podolsky, "An extension to the selective acknowledgement (sack) option for tcp," RFC 2883, IETF, July 2000.

[14] Alan Ford, Costin Raiciu, and Mark Handley, "Tcp extensions for multipath operation with multiple addresses," Internet-draft, IETF, Oct. 2009.

[15] Albert Greenberg, James R. Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A. Maltz, Parveen Patel, and Sudipta Sengupta, "Vl2: a scalable and flexible data center network," in *SIGCOMM '09*, New York, NY, USA, 2009, pp. 51–62, ACM.

[16] Albert Greenberg, Parantap Lahiri, David A. Maltz, Parveen Patel, and Sudipta Sengupta, "Towards a next generation data center architecture: scalability and commoditization," in *Proceedings of the ACM PRESTO*, New York, NY, USA, 2008, pp. 57–62, ACM.

[17] Chuanxiong Guo, Guohan Lu, Dan Li, Haitao Wu, Xuan Zhang, Yunfeng Shi, Chen Tian, Yongguang Zhang, and Songwu Lu, "Bcube: a high performance, server-centric network architecture for modular data centers," New York, NY, USA, 2009, SIGCOMM '09, pp. 63–74, ACM.

[18] Scott Hogg, "10ge and network oversubscription ratios," http://www.networkworld.com/community/node/48965.

[19] Srikanth Kandula, Sudipta Sengupta, Albert G. Greenberg, Parveen Patel, and Ronnie Chaiken, "The nature of data center traffic: measurements & analysis," in *Internet Measurement Conference*, 2009, pp. 202–208.

[20] Costin Raiciu, Christopher Pluntke, Sbastien Barr, Adam Greenhalgh, Damon Wischik, and Mark Handley, "Data centre networking with multipath tcp," in *Ninth ACM Workshop on Hot Topics in Networks (HotNets-IX)*, Monterey, California, US, October 2010.

[21] K. K. Ramakrishnan, Sally Floyd, and David Black, "The addition of explicit congestion notification (ecn) to ip," RFC 3168, IETF, Sept. 2001.

[22] Shan Sinha, Srikanth Kandula, and Dina Katabi, "Harnessing TCPs Burstiness using Flowlet Switching," in *3rd ACM SIGCOMM Workshop on Hot Topics in Networks (HotNets)*, San Diego, CA, November 2004.

[23] Scalable N. Technologies, "QualNet," http://www.scalable-networks.com/.

[24] Ming Zhang, Brad Karp, Sally Floyd, and Larry L. Peterson, "Rr-tcp: A reordering-robust tcp with dsack," in *ICNP*, 2003, pp. 95–106.