Data Center Networking with Multipath TCP

Costin Raiciu†, Christopher Pluntke†, Sebastien Barre‡, Adam Greenhalgh†,
Damon Wischik†, Mark Handley†

†University College London, ‡Universite Catholique de Louvain

ABSTRACT

Recently new data center topologies have been proposed that offer higher aggregate bandwidth and location independence by creating multiple paths in the core of the network. To effectively use this bandwidth requires ensuring different flows take different paths, which poses a challenge.

Plainly put, there is a mismatch between single-path transport and the multitude of available network paths. We propose a natural evolution of data center transport from TCP to multipath TCP. We show that multipath TCP can effectively and seamlessly use available bandwidth, providing improved throughput and better fairness in these new topologies when compared to single path TCP and randomized flow-level load balancing. We also show that multipath TCP outperforms laggy centralized flow scheduling without needing centralized control or additional infrastructure.

Categories and Subject Descriptors

C.2.2[**Computer-Comms Nets**]: Network Protocols

General Terms: Multipath TCP, Data Center Networks

1. INTRODUCTION

Recent growth in cloud applications from companies such as Google, Microsoft, and Amazon has resulted in the construction of data centers of unprecedented size. These applications are written to be distributed across machines numbering in the tens of thousands, but in so doing, they stress the networking fabric within the data center: distributed file systems such as GFS transfer huge quantities of data between end-systems (a point-to-point traffic pattern) while data processing applications such as MapReduce, BigTable or Dryad shuffle a significant amount of data between many machines. To allow maximum flexibility when rolling out new applications, it is important that any machine can play any role without creating hot-spots in the network fabric.

Data center networking has become focus of attention recently; in part this is because data centers are now important enough to be considered as special cases in their own right, but perhaps equally importantly, they are one of the few cases where researchers can dictate both the physical topology and the routing of traffic simultaneously. New topologies such as FatTree[1] and VL2[5] propose much denser interconnects than have traditionally been implemented, so as to allow operators to deploy application functionality in a location independent manner. However, while such dense interconnects can in principle support the full cross-sectional bandwidth of every host communicating flat out simultaneously, the denseness of interconnection poses a difficult challenge for routing. How can we ensure that no matter the traffic pattern, the load is distributed between the many possible parallel paths as evenly as possible?

The current wisdom seems to be to use randomised load balancing (RLB) to randomly choose a path for each flow from among the possible parallel paths. However, RLB cannot achieve full bisectional bandwidth because some flows will randomly choose the same path while other links randomly fail to be selected. Thus, RLB tends to be supplemented by centralized flow-scheduling for large flows.

In this paper we propose an alternative and simpler approach; the end systems in the data center should simply use multipath TCP (MPTCP), as currently under consideration in the IETF[4], to utilize multiple parallel paths for each TCP connection. The great advantage of this approach is that the linked congestion controller in each MPTCP end system can act on very short timescales to move its own traffic from paths it observes to be more congested, onto paths it observes to be less congested. Theory suggests that such behavior can be stable, and can also serve to load balance the entire network.

We evaluate how effective MPTCP is in comparison to alternative scheduling mechanisms across a range of different proposed data center topologies. We use a combination of large scale simulation and smaller scale data center experimentation for evaluation. Our conclusion is that for all the workloads and topologies we considered, MPTCP either matches or in many cases exceeds the performance a centralized scheduler can achieve, and is more robust.

Further, we show that single-path TCP cannot fully utilize capacity for certain topologies and traffic matrices, while multipath can. There is a close connection between topology, path selection, and transport in data centers; this hints at possible benefits from designing topologies for MPTCP.

Hotnets '10, October 20–21, 2010, Monterey, CA, USA.

2. DATA CENTER NETWORKING

From a high-level perspective, there are four main components to a data center networking architecture:

- Physical topology
- Routing over the topology
- Selection between the paths supplied by routing
- Congestion control of traffic on the selected paths

These are not independent; the performance of one will depend on the choices made by those preceding it in the list, and in some cases by those after it in the list. We will discuss each in turn, but it is worth noting now that MPTCP spans both path selection and congestion control, which is why it is able to offer benefits that cannot otherwise be obtained.

2.1 Topology

Traditionally data centers have been built using hierarchical topologies: racks of hosts connect to a top-of-rack switch; these switches connect to aggregation switches; in turn these are connected to a core switch. Such topologies make sense if most of the traffic flows into or out of the data center. However, if most of the traffic is intra-datacenter, as is increasingly the trend, then there is a very uneven distribution of bandwidth. Unless traffic is localized to racks, the higher levels of the topology become a serious bottleneck.

Recent proposals address these limitations. VL2 and FatTree are Clos[3] topologies that use multiple core switches to provide full bandwidth between any pair of hosts in the network. They differ in that FatTree uses larger quantities of lower speed (1Gb/s) links between switches, whereas VL2 uses fewer faster (10Gb/s) links. In contrast, in BCube[6], the hierarchy is abandoned in favor a hypercube-like topology, using hosts themselves to relay traffic.

All three proposals solve the traffic concentration problem at the physical level — there is enough capacity for every host to be able to transmit flat-out to another randomly chosen host. However the denseness of interconnection they provide poses its own problems when it comes to determining how traffic should be routed.

2.2 Routing

Dense interconnection topologies provide many possible parallel paths between each pair of hosts. We cannot expect the host itself to know which of these paths is the least loaded, so the routing system itself must spread traffic across these paths. The simplest solution is to use randomized load balancing, where each flow is assigned a random path from the set of possible paths.

In practice there are multiple ways to implement randomized load balancing in today's switches. For example, if each switch uses a link-state routing protocol to provide ECMP forwarding then, based on a hash of the five-tuple in each packet, flows will be split roughly equally across equal length paths. VL2 provides just such a mechanism over a virtual layer 2 infrastructure.

However, in topologies such as BCube, paths vary in length, and simple ECMP cannot access many of these paths because it only hashes between the shortest paths. A simple alternative is to use multiple static VLANs to provide multiple paths that expose all the underlying network paths[8]. Either the host or the first hop switch can then hash the five-tuple to determine which path is used.

In our simulations, we do not model dynamic routing; instead we assume that all the paths between a pair of endpoints are available for selection, whatever mechanism actually does the selection. For our experiments in Section 4, we use the VLAN-based routing solution.

2.3 Path Selection

Solutions such as ECMP or multiple VLANs provide the basis for randomised load balancing as the default path selection mechanism. However, as others have shown, randomised load balancing cannot achieve the full cross-sectional bandwidth in most topologies, nor is it especially fair. The problem, quite simply, is that often a random selection causes hot-spots to develop, where an unlucky combination of random path selection causes a few links to be underloaded and links elsewhere to have little or no load.

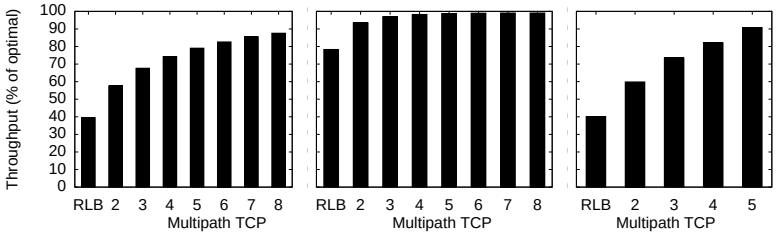
To address these issues, the use of a centralized flow scheduler has been proposed. Large flows are assigned to lightly loaded paths and existing flows may be reassigned to maximize overall throughput[2]. The scheduler does a good job if flows are network-limited, with exponentially distributed sizes and Poisson arrivals, as shown in Hedera [2]. The intuition is that if we only schedule the big flows we can fully utilize all the bandwidth, and yet have a small scheduling cost, as dictated by the small number of flows.

However, data center traffic analysis shows that flow distributions are not Pareto distributed [5]. In such cases, the scheduler has to run frequently (100ms or faster) to keep up with the flow arrivals. Yet, the scheduler is fundamentally limited in its reaction time as it has to retrieve statistics, compute placements and instantiate them, all in this scheduling period. We show through simulation that a scheduler running every 500ms has similar performance to randomised load balancing when these assumptions do not hold.

2.4 Congestion Control

Most applications use singlepath TCP, and inherit TCP's congestion control mechanism which does a fair job of matching offered load to available capacity on whichever path was selected. Recent research has shown there are benefits from tuning TCP for data center use, such as by reducing the minimum retransmit timeout[10], but the problem TCP solves remains unchanged.

In proposing the use of MPTCP, we change the partitioning of the problem. MPTCP can establish multiple subflows across different paths between the same pair of endpoints *for a single TCP connection*. The key point is that by linking the congestion control dynamics on these multiple subflows,



(a) FatTree (8192 hosts) (b) VL2 (11520) (c) BCube (1024)

Figure 1: Throughput for long running connections using a permutation traffic matrix, for RLB and varying numbers of MPTCP subflows

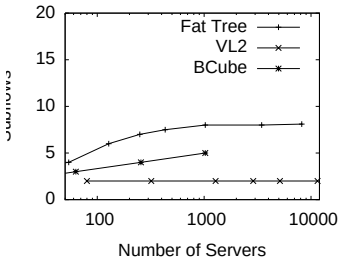


Figure 2: Subflows needed to reach 90% network utilization

MPTCP can explicitly move traffic away from the more congested paths and place it on the less congested paths.

The algorithm currently under discussion in the IETF is called "linked increases" because the slope of additive increase part of the TCP sawtooth is determined by that flow's fraction of the total window of traffic in flight. The faster a flow goes, the larger its fraction of the total, and so the faster it can increase. This algorithm makes MPTCP incrementally deployable, as it is designed to be fair to competing singlepath TCP traffic, unlike simply running multiple regular TCP flows between the same endpoints. In addition it moves more traffic off congested paths than multiple regular TCP flows would.

Our hypothesis is that given sufficiently many randomly chosen paths, MPTCP will find at least one good unloaded path, and move most of its traffic that way. In so doing it will relieve congestion on links that got more than their fair share of RLB-balanced flows. This in turn will allow those competing flows to achieve their full potential, maximizing the cross-sectional bandwidth of the network and also improving fairness. Fairness is not an abstract concept for many distributed applications; for example, when a search application is distributed across many machines, the overall completion time is determined by the slowest machine. Hence worst-case performance matters significantly.

3. ANALYSIS

To validate our hypothesis, we must examine how MPTCP performs in a range of topologies and with a varying number of subflows. We must also show how well it performs against alternative systems. To perform such an analysis is itself challenging - we really want to know how well such deployments will perform at large scale with real-world transport protocol implementations and with reasonable traffic patterns. Lacking a huge data center to play with, we have to address these issues independently, using different tools;

- Flow-level simulation to examine idealized large scale behavior.
- Packet-level simulation to examine more detailed medium-scale behavior.
- Real-world implementation to examine practical limitations at small-scale.

3.1 Large scale analysis

First we wish to understand the potential benefits of MPTCP with respect to the three major topologies in the literature: FatTree, VL2 and BCube. The baseline for comparison is randomized load balancing with RLB using singlepath TCP. MPTCP adds additional randomly chosen paths, but then the linked congestion control moves the traffic within each connection to the least congested subflows.

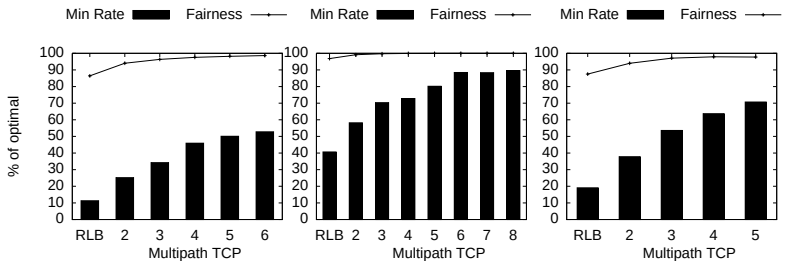
We use an iterative flow-level simulator to analyze topologies of up to 10,000 servers[1]. In each iteration the simulator computes the loss rates for each link based on the offered load, and adjusts the load accordingly. When the offered load and loss rate stabilize, the simulator finishes. This simulator does not model flow startup behavior and other packet level effects, but is scalable to very large topologies.

Fig. 1 shows the total throughput of all flows when we use a random permutation matrix where each host sends flat out (as determined by the TCP response function) to a single other host. In all three topologies, with the right path selection this traffic pattern should just be able to load the network to full capacity but no more.

What we observe is that RLB is unable to fill any of these networks. It performs best in the VL2 topology, where it achieves 77% throughput, but performs much worse in FatTree and BCube. The intuition is simple: to achieve 100% capacity with RLB, no two flows should ever traverse the same link. Obviously RLB cannot do this, but how badly it suffers depends on how overloaded links become. With FatTree, when two TCP flows that could potentially send at 1Gb/s end up on the same 1Gb/s link, each backs off by 50%, leaving other links underutilized. With VL2, when eleven 1Gb/s TCP flows end up on the same 10Gb/s link, the effect is much less drastic, hence the reduced performance penalty.

The benefits of MPTCP are clear; as additional subflows are added, the overall throughput increases. How many subflows are needed depends on the topology; the intuition is as before - we need enough subflows to overcome the traffic concentration effects of the random path allocation. One might think that *the power of two choices*[7] might apply here, providing good load balancing with very few subflows. However it does not because the paths are not disjoint. Each

[1]how many is determined by the need for a regular topology



(a) FatTree (b) VL2 (c) BCube

Figure 3: Minimum flow throughput and Jain fairness index for flows in Fig. 1

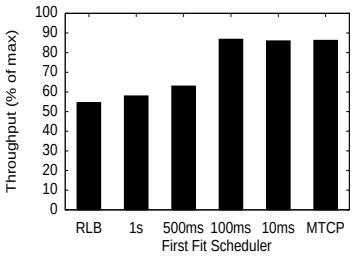


Figure 4: First-fit scheduling compared to RLB and MPTCP

subflow can encounter a congested bottleneck on a single link along its path, causing the other links along the path to be underutilized. Although such bottleneck links are load-balanced, with FatTree in particular, other links cannot be fully utilized, and it takes more than two subflows to spread load across sufficient paths to fully utilize the network.

This raises the question of how the number of subflows needed scales with the size of the network. We chose an arbitrary utilization target of 90% of the cross sectional bandwidth. For different network sizes we then progressively increased the number of subflows used. Fig. 2 shows the minimum number of subflows that can achieve 90% utilization for each size of network. The result is encouraging: beyond a certain size, the number of subflows needed does not increase significantly with network size. For VL2, two subflows are needed. For FatTree, eight are needed. This might seem like quite a high number, but for an 8192-node FatTree network there are 256 distinct paths between each host pair, so only a small fraction of the paths are needed to achieve full utilization. From the host point of view, eight subflows is not a great overhead.

We also care that the capacity is allocated fairly between connections, especially for applications where the final result can only be returned when the last node running a part of a calculation returns its results. Fig. 3 shows the throughput of the lowest speed flow (as a percentage of what should be achievable) and Jain's fairness index for the three topologies. Multipath always improves fairness, even for the VL2 topology which performed relatively well if we only examine throughput.

We have also run experiments in our packet-level simulator with a wide range of load levels. At very light load, there are few collisions, so MPTCP gives little benefit over RLB on FatTree or VL2 topologies. However on BCube, MPTCP excels because a single flow can use all the the host interfaces simultaneously.

At the other extreme, under overload conditions even RLB manages to fill the network, but MPTCP still gives better fairness. Fig. 5 shows the throughput of each individual flow in just such an overload scenario.

The results above use a permutation traffic matrix, which is useful as a benchmark because it enables a network designed for full bisection bandwidth to be loaded to 100% utilization with the right traffic distribution scheme. In practice less regular traffic and both lighter and heavier loads are of interest. Fig. 6 shows results when the source and destination is chosen randomly for varying numbers of flows.

FatTree show substantial improvements over single-path RLB, even for very light or very heavy loads. This shows the performance benefits of MPTCP are robust across a wide range of conditions.

The improvements for BCube are even greater at lower traffic loads. This is because BCube hosts have multiple interfaces, and MPTCP can use them all for a single flow - at light loads the bottlenecks are the hosts themselves.

The results for VL2 were a surprise, given that Fig. 1 shows improvements for this topology with the permutation matrix. MPTCP gives improvements over RLB of less than 1% for all loads we studied. On closer examination, it turns out that the host interface is almost always the bottleneck for VL2. Many flows collide on either the sending or receiving host, and MPTCP has no path diversity here. The 10Gb/s links are then not the bottleneck for the remaining flows under these load levels.

3.2 Scheduling and Dynamic Flow Arrivals

With single-path TCP is it clear that RLB does not perform sufficiently well unless the topology has been specifically tailored for it, as with VL2. Even with VL2, fluid simulations show that MPTCP can increase fairness and performance significantly.

RLB however is not the only singlepath path selection algorithm; Hedera proposes using a centralized scheduler to supplement RLB, with the goal of explicitly allocating large flows to paths. Specifically, Hedera flows start off using RLB, but are measured by the centralized scheduler. If, during a scheduling period, a flow's average throughput is greater than 10% of the interface speed, it is explicitly scheduled. How well does MPTCP compare with centralized scheduling?

This evaluation is more difficult; the performance of a scheduler can depend on lag in flow measurement, path configuration, and TCP's response to path reconfiguration. Similarly the performance of MPTCP can depend on how quickly

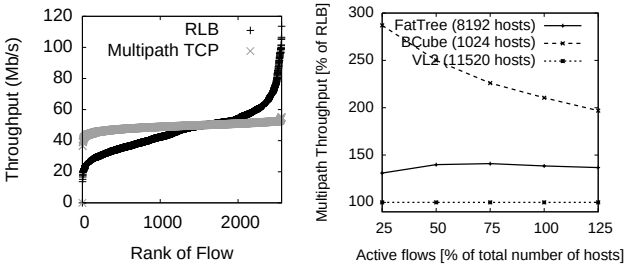
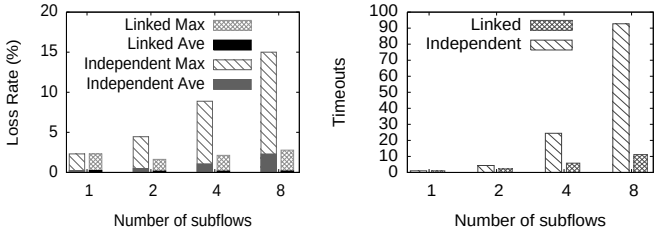


Figure 5: Flow Rates for an overloaded Fat Tree (128)

Figure 6: Random conns: Improvement vs. load



(a) Network loss rates

(b) Retransmit Timeouts

Figure 7: MPTCP vs. multiple independent TCP flows

new subflows can slowstart. None of these effects can be captured in a fluid flow model, so we have to resort to full packet-level simulation.

For our experiments we modified *htsim*[9], which was built from ground up to support high speeds and large numbers of flows. It models TCP very similarly to ns2, but performance is much better and simulation time scales approximately linearly with total bandwidth simulated.

For space reasons, we only examine the FatTree topology with 128 severs and a total maximum bandwidth of 128Gb/s. We use a permutation traffic matrix with closed loop flow arrivals (one flow finishes, another different one starts), and flow sizes distributed according to the VL2 dataset. We measure throughputs over 20 seconds of simulated time for RLB, MPTCP (8 subflows), and a centralized scheduler using the First Fit heuristic, as in Hedera [2].[2]

The average bisectional bandwidth achieved is shown in Fig. 4. Again, MPTCP significantly outperforms RLB. Centralized scheduler performance depends on how frequently it is run. In the Hedera paper it is run every 5 seconds. Our results show it needs to run every 100ms to approach the performance of MPTCP; if it runs as frequently as every 500ms there is little benefit because in the high bandwidth data center environment even large flows only take around a second to complete.

Host-limited Flows

Hedera's flow scheduling algorithm is based on the assumption that long-lived flows contribute most of the bytes and therefore it only needs to schedule those flows. Other flows are treated as background noise. It also assumes that flows which it schedules onto unused links are capable of increasing their transmit rate to fill that link.

Both assumptions can be violated by flows which are end-host limited and so cannot increase their transmission rate. For example, network bandwidth can easily exceed disk performance for certain workloads. Host-limited flows can be long lived and transfer a great deal of data, but never exceed the scheduling threshold. These flows are essentially invisible to the scheduler and can collide with scheduled flows. Perhaps worse, a host-limited flow might just exceed the threshold for scheduling and be assigned to an empty path which it cannot utilize, wasting capacity.

We ran simulations using a permutation matrix where each host sends two flows; one is host-limited and the other is not. When the host-limited flows have throughput just below the 10% scheduling threshold, Hedera's throughput drops 20%. When the same flows are just above the threshold for scheduling it costs Hedera 17%.

Scheduling	App Limited Flows	
Threshold	Over-Threshold	Under-Threshold
5%	-21%	-22%
10%	-17%	-21%
20%	-22%	-23%
50%	-51%	-45%

The table shows the 10% threshold is a sweet spot; changing it either caused too few flows to be scheduled, or causes even more problems when a scheduled flow cannot expand to fill capacity.

In contrast, MPTCP makes no such assumptions. It responds correctly to competing host-limited flows, consistently obtaining high throughput.

MPTCP vs. Multiple TCP Connections

Using multiple subflows clearly has significant benefits. However, MPTCP is not the only possible solution. Could we not simply use multiple TCP connections in parallel, and stripe at the application level?

From a network performance point of view, this is equivalent to asking what the effect is of the congestion control linkage within MPTCP. If, instead of using MPTCP's "linked increases" algorithm, we use regular TCP congestion control independently for each subflow, this will have the same effect on the network.

To test this, we use again the permutation traffic matrix and create 20 long running flows from each host. We measure network loss rates for MPTCP with Linked Increases and compare against running independent TCP congestion control on each subflow.

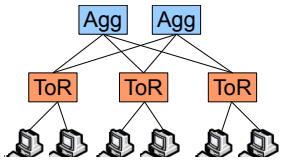
The results in Fig. 7(a) show that MPTCP does not increase network load, as measured by either mean or max loss rate. In contrast, independent congestion control for each

[2]We chose First Fit because it runs much faster than the Simulated Annealing heuristic; execution speed is really important to get benefits with centralized scheduling.

subflow causes the network to become increasingly congested as we add more subflows.

Fig. 7(b) shows the number of retransmit timeouts in this experiment. Unsurprisingly, independent TCP suffers many timeouts, which may impact application performance. With MPTCP, even though the loss rate does not increase as we add subflows, the probability of timeout does increase a little. This is because as we add subflows, each one becomes less aggressive, so uses a smaller window. In some subflows the congestion window may be less than the four packets needed to generate three dup-acks and trigger fast retransmit. This could be mitigated by reducing the dup-ack threshold, as reordering is uncommon in data center networks.

4. EXPERIMENTAL EVALUATION



To test with our Linux implementation of MPTCP we used the topology above, which is essentially a trimmed down FatTree with two paths between each pair of endpoints. Our aim is not to verify the simulation results (we don't have access to enough machines), but rather to examine the potential downsides to MPTCP that might arise through practical implementation issues.

Throughput. Ideally we'd like to examine the end-host performance of MPTCP to see how much more expensive it is for the operating system. Unfortunately our implementation is a work-in-progress and is not yet sufficiently tuned to get representative results[3]. Despite this, it is still worth examining performance.

We use two permutation connection matrices where each host sends to a single other host in a different rack. In TM1, each host sends to its second neighbor to the right. In TM2, each host establishes a bidirectional connection to its third neighbor to the right. File sizes are drawn from a Pareto distribution, and as soon as one file transfer completes, another is started.

With RLB, each connection picks one of the two paths to the destination by selecting a random VLAN source address. MPTCP creates two subflows, one on each path to the destination. For the centralized "scheduler", we hard code the paths to avoid overlap.

Experiment	RLB	Scheduler	MPTCP
TM 1	2.7Gb/s	4.7Gb/s	4.2Gb/s
TM 2	2.7Gb/s	2.7Gb/s	4.2Gb/s

[3]Currently it does not use DMA and the code is still optimized for the common case of in-order arrivals, whereas MPTCP would benefit from a different set of optimizations.

The total network goodput is shown above. After accounting for overhead and Acks, Scheduler gets 90% of the theoretical maximum on TM1, whereas MPTCP gets 80% due to the limitations of our implementation. This is still a substantial improvement over RLB.

The TM2 results are more surprising. The poor performance of Scheduler is a limitation of the VLAN-based routing, which forces bidirectional flows to take symmetric paths. It turns out that no collision-free schedule is possible for single-path TCP. In contrast, MPTCP uses all paths and so works well despite collisions.

5. CONCLUSIONS

MPTCP is a simple solution that can effectively utilize the dense parallel network topologies proposed for modern data centers, significantly improving throughput and fairness compared to singlepath TCP.

MPTCP unites path selection and congestion control, moving traffic to where capacity is available. This flexibility allows the creation of cheaper and better network topologies: for instance, cheap multiport NICs can be used between hosts and ToR switches allowing much higher throughputs when the network core is underutilized. It is hard for regular singlepath TCP to use such capacity. We will explore such topologies in our future work.

As with any system, there may be some costs too: under heavy load, per-packet latency may increase due to timeouts; a little more memory is needed for receive buffers; and some complexity will be added to OS implementations, particularly to deal with issues such as such as flow-to-core affinities. We intend to explore these in future work, together with ways of automatically tuning MPTCP to bring the best performance.

6. REFERENCES

[1] M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. In *Proc. SIGCOMM 2010*.

[2] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat. Hedera: Dynamic flow scheduling for data center networks. In *Proc. Usenix NSDI 2010*.

[3] C. Clos. A study of non-blocking switching networks. *Bell System Technical Journal*, 32(5):406–424, 1952.

[4] A. Ford, C. Raiciu, M. Handley, and S. Barre. TCP Extensions for Multipath Operation with Multiple Addresses. Internet-draft, IETF, 2009.

[5] A. Greenberg el al. VL2: a scalable and flexible data center network. In *Proc. ACM Sigcomm 2009*.

[6] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu. Bcube: a high performance, server-centric network architecture for modular data centers. In *Proc. SIGCOMM 2009*.

[7] M. Mitzenmacher. The power of two choices in randomized load balancing. *IEEE Trans. Parallel Distrib. Syst.*, 12(10):1094–1104, 2001.

[8] J. Mudigonda, P. Yalagandula, M. Al-Fares, and J. C. Mogul. Spain: Cots data-center ethernet for multipathing over arbitrary topologies. In *Proc. NSDI 2010.*

[9] C. Raiciu, D. Wischik, and M. Handley. htsim. http://nrg.cs.ucl.ac.uk/mptcp/implementation.html.

[10] V. Vasudevan et al. Safe and effective fine-grained tcp retransmissions for datacenter communication. In *Proc. SIGCOMM 2009*.