Performance Evaluation of Multi-Path TCP for Data Center and Cloud Workloads

Lucas Chaufournier
UMass Amherst
lucasch@cs.umass.edu

Ahmed Ali-Eldin
UMass Amherst
ahmeda@cs.umass.edu

Prateek Sharma
Indiana University
prateeks@iu.edu

Prashant Shenoy
UMass Amherst
shenoy@cs.umass.edu

Don Towsley
UMass Amherst
towsley@cs.umass.edu

ABSTRACT

Today's cloud data centers host a wide range of applications including data analytics, batch processing, and interactive processing. These applications require high throughput, low latency, and high reliability from the network. Satisfying these requirements in the face of dynamically varying network conditions remains a challenging problem. Multi-Path TCP (MPTCP) is a recently proposed IETF extension to TCP that divides a conventional TCP flow into multiple subflows so as to utilize multiple paths over the network. Despite the theoretical and practical benefits of MPTCP, its effectiveness for *cloud applications* and environments remains unclear as there has been little work to quantify the benefits of MPTCP for real cloud applications. We present a broad empirical study of the effectiveness and feasibility of MPTCP for data center and cloud applications, under different network conditions. Our results show that while MPTCP provides useful bandwidth aggregation, congestion avoidance, and improved resiliency for some cloud applications, these benefits do not apply uniformly across applications, especially in cloud settings.

ACM Reference Format:
Lucas Chaufournier, Ahmed Ali-Eldin, Prateek Sharma, Prashant Shenoy, and Don Towsley. 2019. Performance Evaluation of Multi-Path TCP for Data Center and Cloud Workloads. In *Tenth ACM/SPEC International Conference on Performance Engineering (ICPE '19), April 7–11, 2019, Mumbai, India.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3297663.3310295

1 INTRODUCTION

Data centers host a wide range of complex applications with variable and evolving requirements. Most of these applications are distributed in nature with their components communicating over the data center network. Consequently, data center networks must support a diverse array of requirements and workloads ranging from latency-sensitive applications such as web servers to throughput-intensive ones such as distributed data processing and virtual machine live-migration.

The diverse mix of applications, workload burstiness, and dynamic placement of application components on data center servers results in highly dynamic and bursty network traffic [6]. This high variability in network traffic can cause congestion on certain parts of the data center network, and can cause network *imbalance*. This imbalance and congestion adversely affects both throughput and latency, and can severely hurt application performance. In some latency-sensitive applications such as web-search, it even affects correctness and monetary revenue [3].

Given the undesirable nature of network imbalance, network load balancing has received significant attention from both industry and academia [2, 3]. Traditionally, one of two approaches have been used to address network load dynamics within data center networks. The first approach uses network topologies such as fat-trees [1, 18] to provide multiple paths between servers and network switches, thereby providing high aggregate network bandwidth. The second approach detects overloaded network elements and makes appropriate packet switching and forwarding decisions to alleviate network hot-spots. This approach pushes network management *down* into the network switches, and is implemented by switch-based techniques such as ECMP and Software Defined Networks (SDNs) [27].

A less explored alternative for handling network dynamics is the use of *multi-path networking* where packets to and from an application traverse multiple network paths *concurrently*. Multi-path networking has many potential benefits. In addition to increased reliability, multi-pathing enables network resources from multiple paths to be aggregated—potentially providing increased bandwidth. Multi-path networking can also mitigate congestion within the data center network or handle network failures by exploiting redundant paths [43]. Multi-path networking is *feasible* in today's data centers since (i) common data center network topologies such as fat-trees and clos provide multiple paths to each server, and (ii) servers themselves have multiple network interfaces (NICs), enabling them to exploit redundant paths to other servers.

Multi-path TCP (MPTCP) is an end-to-end protocol, proposed as an IETF extension to TCP, to provide multi-path networking capabilities without the need for complex in-network hardware [16]. MPTCP divides a conventional TCP flow into multiple sub-flows that traverse multiple paths between servers utilizing the available NICs on a server [37]. It is a drop-in replacement for TCP that theoretically can enhance data center application performance by providing network bandwidth aggregation across multiple paths, increased reliability and resiliency against network failures, and better performance with congestion [37]. In the context of data center networks, MPTCP performance has been evaluated using simulations and simple network measurement applications like iperf [34], but not using real-world data center and cloud workloads. Similarly, MPTCP's reliability benefits have been studied for mobile environments such as cellphones with multiple interfaces (cellular and WiFi) [11], a different environment from data center networks. We build on this prior work and address the following question in this paper: *How do modern data center and cloud applications perform under MPTCP in real-world settings from the perspective of bandwidth aggregation, multi-tenancy, network congestion and reliability?* To address this broad question, we experimentally study MPTCP performance to answer the following research questions.

(1) How much benefit does MPTCP's promise of aggregated bandwidth provide to applications in uncongested networks? How do these benefits change for multi-tenant applications with competing network traffic?
(2) What is the application performance under congested network settings using MPTCP? Does MPTCP provide reliability benefits when compared to TCP for handling network failures?
(3) For each of the above questions, what type of application workloads see the greatest benefits from MPTCP? Are there workloads or scenarios where MPTCP is detrimental to performance when compared to TCP?
(4) How well do these benefits, measured in a private cloud environment, carry over to a public cloud infrastructure?

In answering these questions, our paper makes the following research contributions:
1. We conduct an empirical performance study evaluating the performance of MPTCP beyond simple benchmark applications used in prior work. Our study considers a wide range of representative data center workloads under different scenarios including non-congested networks, multi-tenancy, congestion, and in public clouds.
2. In uncongested single-tenant scenarios, we find MPTCP to be ineffective over TCP since other resources become a bottleneck well before the network saturates. In uncongested multi-tenant scenarios, MPTCP provides benefits for select workload classes but not for others. Specifically, memory-transfer intensive workloads such as VM migration can see up to 2X higher throughput, while bulk disk transfers workloads see benefits whenever SSD read speeds exceed allocated network rates. No benefits are seen for Spark workloads on slower HDDs or for request-response workloads.
3. In the presence of network congestion, MPTCP offers benefits for a broad set of applications, with the up to 20 to 50% better performance over TCP in the presence of congestion—due to its ability to exploit multiple paths. In the presence of network failures, MPTCP's robustness properties allow us to exploit additional network paths upon a failure on the first path, providing significant robustness advantages over TCP.
4. While the above benefits apply to private cloud data centers, we find that today's public cloud platforms employ network topologies and policies such as throttling that may prevent applications from fully exploiting MPTCP's capabilities.

2 BACKGROUND

Over the past decade, data center applications have evolved considerably from legacy applications to today's cloud-native applications.

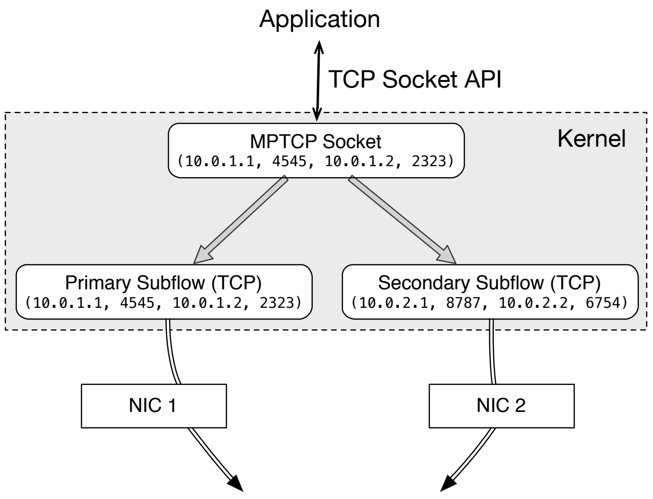


Figure 1: MPTCP creates multiple TCP subflows for a single TCP connection.

Cloud workloads today range from micro-services with microsecond latency requirements to batch applications and distributed data processing jobs running for hours or even days [38]. In parallel to the application evolution, data center networks have seen a parallel evolution to provide high network performance for these applications.

Moreover, application dependence on network performance can vary considerably, since applications may be bottlenecked by other resources such as CPU and I/O. For example, the importance of high-speed networks for distributed data processing workloads such as Hadoop and Spark has recently been questioned [30, 41]. While prior work on MPTCP has focused on network performance, it is also important to understand the benefits of MPTCP for end-to-end application performance.

Multi-Path TCP. MPTCP is a set of proposed protocol extensions to conventional TCP, that enables TCP to effectively use multiple paths by dividing a TCP flow into multiple sub-flows. Subflows take advantage of the multiple NIC's available on each server to traverse multiple available paths between servers [37]. Since today's data center network architectures (like clos and fat-trees) already provide some form of link redundancy, MPTCP can be used in data centers to provide bandwidth aggregation, resiliency, and higher performance when some network paths are congested.

MPTCP is carefully designed to be robust against middle-boxes such as NATs. Applications and users create MPTCP connections using the existing BSD socket API (`connect, bind, listen, read, write`, etc.). MPTCP is implemented as a layer on top of the TCP stack in the kernel. With an MPTCP enabled kernel, applications need no modifications, as the MPTCP layer in the kernel treats TCP sockets as MPTCP sockets. MPTCP has many desirable properties that may provide many benefits in data centers. These include the following:
Bandwidth aggregation via Resource Pooling: One of MPTCP's design principles is to provide *resource pooling* at the network layer. Essentially, MPTCP exposes multiple network paths as a single

abstract connection. Each subflow sends data on one network path, and the network resources on these paths are aggregated and pooled. As an example, when two servers are each connected using two 10 Gbps interfaces, an MPTCP connection enables throughput of up to 20 Gbps, the aggregate of the two connections.

Improved Resiliency: Using multiple paths presents an obvious reliability advantage. If one path is inaccessible, then the MPTCP connection can continue data transfer on the remaining subflows, resulting in improved resiliency against network faults to applications.

Operational Readiness: MPTCP operates at the end-host level, and does not require support from the underlying network, in contrast to other approaches for data center networking that rely on switch-level management [2, 47]. MPTCP is both network *and* application transparent since it presents a standard TCP socket abstraction to applications. Further, MPTCP support is available both in server operating systems such as Linux, and in mobile OSes such as iOS. Hence, users of data centers and clouds can use MPTCP without any network or application modification, and avail these features.

MPTCP implementation. The implementation of MPTCP in Linux and other OSes can be understood from the perspective of data and control planes, and congestion control.

Control plane: MPTCP uses TCP connections to send data on multiple paths. It retains and extends TCP's four-tuple (`source-ip, source-port, destination-ip, destination-port`) model of connection establishment. To the application, it presents the abstraction of a TCP connection. Underneath, it sets up multiple *subflows* for a single connection (see Figure 1). Each subflow represents a distinct four-tuple, and data is sent on each subflow using TCP. As part of MPTCP's connection establishment, it advertises and negotiates the subflows based on the IP addresses available on each host.

Each subflow represents a separate path between the source and the destination. For example, if each host has two network interfaces (and hence two IP addresses), MPTCP establishes up to four subflows. MPTCP creates a *primary subflow* that corresponds to the TCP connection four-tuple requested by the application. The primary subflow is established first, followed by the secondary subflows on the other paths. Subflows are said to have been established once their TCP connection is also established and ready to send/receive data. If a path becomes (in)accessible, its corresponding subflow is removed/added by MPTCP. Note that the number of network links shared by the paths (and hence subflows) depends on the network topology. Subflows can also use disjoint network paths offered by common data center network topologies such as fat-trees.

Data plane: Since MPTCP uses TCP for the underlying subflows, its data transfer mechanisms are based on TCP sequence numbers, windows, etc. The existence of multiple paths requires that the MPTCP layer must tackle the problem of *scheduling* packets on to subflows, and reassembling packets from different subflows at the receiver [5, 32]. MPTCP supports multiple scheduling policies including, for example, round-robin. The default, more sophisticated scheduling policy considers the available window sizes and round trip times (RTTs) of the subflows. Each subflow maintains its own TCP sequence numbers, and the data from multiple subflows is reassembled by MPTCP to provide the application with an ordered bytestream.

Congestion control: MPTCP implements its own congestion control that works across all subflows [35]. One of its design goals is to be fair to existing TCP connections not causing other connections to starve. Accordingly, its congestion control scheme uses a single congestion window shared by all subflows, and changes the global congestion window based on congestion on each subflow. In addition to this *coupled* policy, other congestion control policies have been proposed in the literature [22, 28, 44], including *uncoupled* policies, which treat each subflow independently and is not fair to other TCP traffic [13].

VM	vCPU	Mem (GiB)	SSD (GiB)	NICs	Bandwidth
D4 v2	8	28	400	8	6000 (Mbps)
D13 v2	8	56	400	8	6000 (Mbps)

Table 1: Azure VMs used in our public cloud experiments.

3 EXPERIMENTAL METHODOLOGY

In this section, we describe our experimental methodology, including our system setup, network setup and applications workloads used in our empirical study.

3.1 System setup

Our experiments assume two distinct environments, a cluster environment in a private cloud and a production public cloud. We conduct our initial set of experiments in the private cluster environment, which enables controlled experimentation with varying network loads. We also conduct additional experiments in a public cloud to see how well these results apply to those settings.

Cluster Environment for Private Cloud. Our private cloud cluster consists of eight Dell PowerEdge R430 servers with two 2.10 GHz Intel E5-2620 v4 CPUs, 64GB of RAM, 1 TB SATA HDD, and a Samsung Evo 850 250 GB SSD. The servers are equipped with both 10 GigE and 1 GigE cards, Intel X710 and Broadcom NetXtreme BCM5720 NICs, respectively. We configure our servers to run Ubuntu 16.04 with Linux kernel v4.10. The cluster network is configured as a simplified two-level fat-tree topology [1, 12] as shown in Fig 2. Nodes are divided between two pods/racks. The servers are interconnected by 10 GigE Netgear M4300-12x12F switches. We use two 10Gb ethernet links for each server unless explicitly specified—i.e., applications using MPTCP have an available aggregate bandwidth of 20Gbps.

Public Cloud Setup. Our public cloud experiments are run on the Azure public cloud. The experiments are performed on the Azure East-US region using two different VM types, namely, Standard D4 V2 and Standard D13 V2 VM types. Table 1 summarizes each VM type and their available resources. Both VM types are identical except for their memory sizes. We configure each VM with two network interfaces, with each interface assigned an IP address on a different subnet.

MPTCP Setup. Our machines run Linux with kernel version 4.10 that is configured to run MPTCP v0.93 [1]—the latest version as of Jan. 1, 2018 [31]. While we have experimented with different MPTCP kernel configurations and congestion control algorithms, unless otherwise specified, our results assume the default kernel

[1] We performed the same experiments with the three previous versions of MPTCP, but do not report there results due to space constraints.

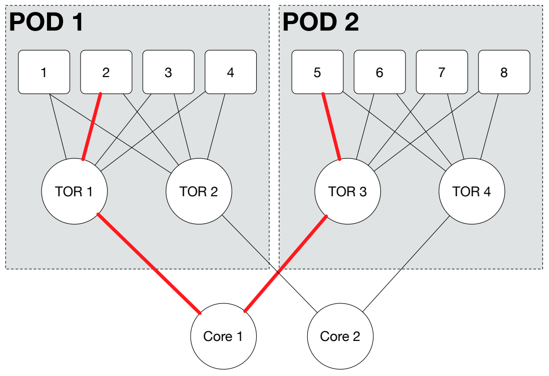


Figure 2: Two-level fat-tree network used in our private cluster experiments.

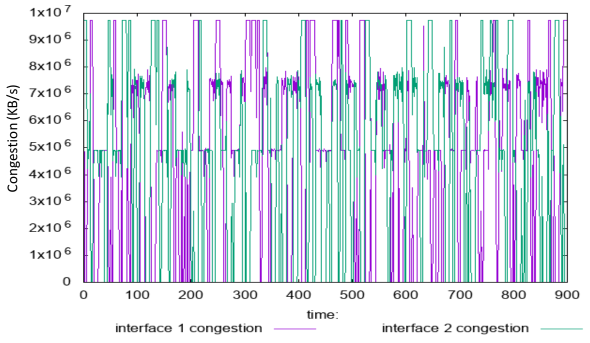


Figure 3: Alternating congestion on the two interfaces.

options and the LIA congestion control, which is the recommended congestion control algorithm for MPTCP. To reduce unnecessary CPU overhead, we turn off MPTCP header checksumming while leaving the conventional TCP checksums enabled even when using MPTCP. In most experiments, receive buffers are set according to RFC6182 [15] which specifies the multiplication of the maximum round-trip time across all paths by the total bandwidth available as the suggested size of the buffers. This results in receive buffers being much larger (256MB) than the default (8KB).

3.2 Application Workloads

Our experiments assume two broad classes of cloud workloads, namely memory-intensive and disk-intensive, both of which generate network traffic of varying characteristics (e.g., packet sizes, network bandwidth use). For memory-intensive workloads, our experiments use the following:

- Bulk memory data transfer microbenchmarks using iperf
- Live virtual machine migration which involves network-intensive memory to memory data transfers
- Redis, an in-memory key-value store, with the Yahoo Cloud serving Benchmark (YCSB) to generate requests

For disk-intensive workloads, our experiments use the following:

- Bulk disk data transfer microbenchmarks using rsync and FTP
- Distributed data processing using Spark

The specific setup of each of these workload is described along with the experiments in sections 4 and 5.

3.3 Network Conditions

For both memory and disk bound workloads, we first run experiments in an uncongested network where the full 10 GigE interface bandwidth is available to the application and there is no background traffic on the switches. We then repeat the experiment in a multi-tenant environment by assuming that the server hosts multiple applications and rate limit an application's network bandwidth allocation to smaller values (e.g., 1 Gb/s, 2Gb/s and so on); such rate limits are common in public cloud environments (as we later discuss in Section 6). Next, we introduce congestion in the network and measure the application performance in the presence of congestion. Finally, we introduce link failures of certain links on switches and measure application performance in the presence of failures. By gradually progressing from no congestion to network failures, we are able to systematically evaluate MPTCP under a broad set of network conditions.

Since MPTCP's use of multiple paths can provide congestion and failure resilient capabilities *without* the need for custom network modifications, we run experiments congesting the network paths to study how MPTCP is robust under congestion conditions. To study the performance of MPTCP versus TCP under a realistic congestion profile, we built a network congestion generator that randomly congests a network interface. We introduce congestion for a variable amount of time and bandwidth over all available paths, from two neighboring servers, to mimic background traffic seen in data center networks. The congestion per interface is independent of the presence of congestion on the other interfaces, i.e., for MPTCP, the two interfaces can become congested during the same time period. An example congestion profile on two network interfaces is shown in Figure 3, which shows a snapshot of the traffic generated on the interfaces per second for a period of 15 minutes. The congestion generator is used in all our congestion experiments.

4 PERFORMANCE OF MEMORY-INTENSIVE WORKLOADS

We first begin our experimental study by considering memory-intensive workloads that perform memory-to-memory network I/O with different characteristics. Our premise is that memory-to-memory data transfers represent the "best case" for exploiting the available network bandwidth under MPTCP (e.g., for exploiting the available aggregate bandwidth on both paths) since memory is often not a bottleneck for such workloads, leaving them flexibility to fully utilize the available network bandwidth. Consequently we begin our experiments with applications such as memory data transfers, live VM migration, and in-memory key-value stores that represent this class of workloads. We begin with a description of the setup for each workload and then describe our results.

Memory data transfer microbenchmark. iperf is a memory-to-memory data transfer microbenchmark used to stress test networks and measure available bandwidth. Thus it is inhrently capable of using all available bandwidth between a pair of servers (and that on multiple paths for MPTCP). We configure iperf to perform bulk data transfers of various sizes ranging from as small as 1KB to as large as 10 GB. This microbenchmark has also been used in prior work on MPTCP performance [5, 34, 48] and provides a baseline result for our more complex workloads.

Live Virtual Machine Migration. Cloud data centers are virtualized and host applications inside virtual machines. The VMs are often migrated from one physical server to another for re-balancing load and alleviating hot-spots [46]. Live VM migration is therefore a common internal data center workload that involves iterative memory transfer of VM state from one machine to another via a process called pre-copying, i.e., dirty pages, at the source server are recopied to the destination continuously until the rate of re-copied pages is greater than the page dirtying rate. As memory bandwidth is several times larger than network bandwidth, live migration is usually network bound. In our experiments, we use KVM to migrate VMs with different memory sizes (ranging from 16GB to 64GB memory sizes) from one machine to another. The VMs run a kernel compile task during the migration, which causes significant dirtying of pages and stresses the migration process. The VM disk state is assumed to be stored on network storage and is not transferred (a common assumption in data centers).

Latency-sensitive In-Memory Key-value stores. In-memory key-value stores, such as memcached and Redis, are key components of many cloud applications, since providing low latencies is an important performance requirement for these workloads. We use Redis, a widely used in-memory key-value store as a representative example of such a workload and use the Yahoo Cloud Serving Benchmark (YCSB) v0.12 [9] to generate read and write requests to this Redis server. We use one of YCSB's core workloads, `workloada` [2], that has a mix of 50:50 reads and writes to generate the client workload. We also use a second read-intensive workload with a 99:1 reads/write ratio. Both workloads are configured to request objects of two different sizes 1KB and 100KB from the in-memory Redis store.

4.1 Performance in Uncongested Networks

We first present experimental results for a single-tenant uncongested network where there is no background traffic from other applications, and thus no network interference.

Memory Bulk Transfer Microbenchmarks. We configure iperf to perform in-memory network transfers of various sizes ranging from kilobytes to megabytes to ten gigabytes. We determine the total latency to perform these data transfers under MPTCP and compare it to vanilla TCP. Figure 4 depicts the latency of these in-memory transfers for both MPTCP and TCP, and reports the mean value for each data point over 15 runs. The figure shows that both MPTCP and TCP exhibit very similar latencies for up to 100 MB of network transfers. However, MPTCP begins to outperform TCP beyond this point and shows 2X lower latencies for transfer sizes between 1GB to 10GB (and beyond). This result is in line with our expectation (and prior work [34]). Since the servers are equipped with 10 GigE network interface cards, flows up to 100MB take anywhere from tens of micro-seconds to tens of milliseconds to complete (depending of transfer size). This is too short of a duration for MPTCP to begin

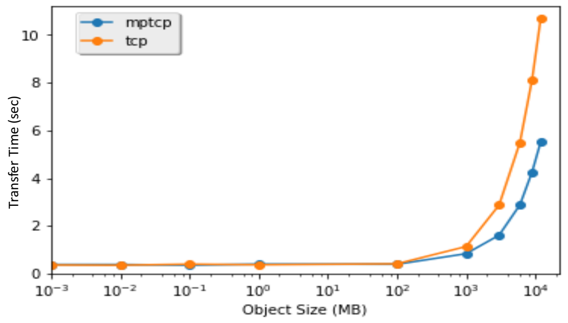


Figure 4: Latencies for memory-to-memory bulk data transfers. MPTCP shows up to 2X benefits due to its bandwidth aggregation but only for transfer sizes exceeding 100MB.

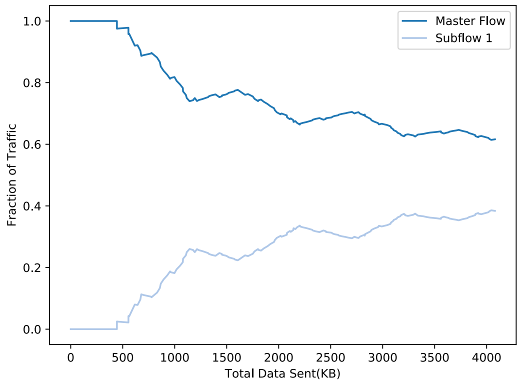


Figure 5: MPTCP's subflow does not start sending packets until 500Kb are sent on the main flow.

exploiting its second path, causing MPTCP behavior to resemble TCP behavior on a single path. For large data transfers of 1GB or greater, which are longer lived flows, MPTCP can begin to exploit both network paths allowing it to exhibit network aggregation benefits. We note that even a 10GB transfer takes only about a second for TCP and half a second for MPTCP on 10 GigE interfaces. As has been observed in prior work, flows need to be "long" and sufficiently network-intensive for MPTCP to exploit its aggregation capability, and these microbenchmarks confirm those insights for 10 GigE environments. These overheads are due to MPTCP's more complex sequence of events during connection establishment, and due to MPTCP's scheduler, which chooses a subflow for outgoing packets.

To validate the above insights, we captured a packet trace of iperf sending ten gigabytes of data and plotted the fraction of traffic sent over each subflow. Figure 5 shows the amount of traffic sent over each of MPTCP's subflows for the first several KBs. We see that the master subflow is used to send all traffic until 500KB of data is sent, at which point the second subflow is established and begins transferring data. The figure shows that while the second subflow starts when around 500KB of data has been transmitted, it is not until at least 1000KB of data is sent that the second subflow is effective.

[2]See https://github.com/brianfrankcooper/YCSB/wiki/Core-Workloads

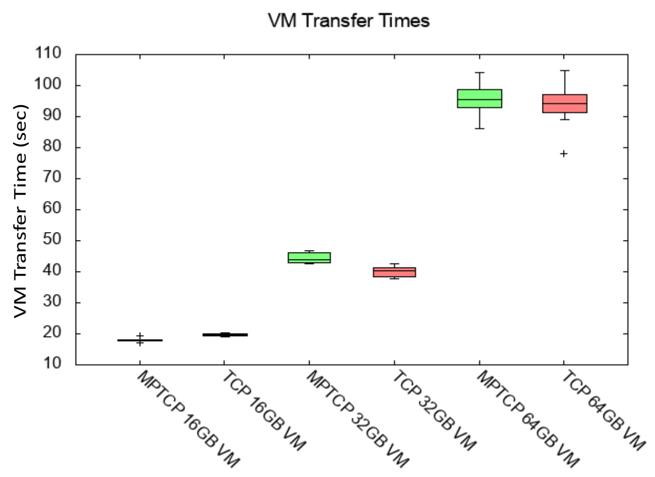


Figure 6: Live VM migration times for different VM memory sizes. Surprisingly, MPTCP does not show any improvement over TCP in uncongested networks.

Thus for workloads where the total amount of data sent is less than 1MB, MPTCP may be ineffective for bandwidth aggregation due to the delay in setting up and using the second path. Further, our results indicate that the flow has to be sufficiently long ($>$100MB in our experiments) to see meaningful bandwidth aggregation benefits in high-speed LAN settings.

VM Live Migration. VM live migration involves copying the memory state of the VM from one server to another using an iterative copying mechanism. It is therefore a pure memory-to-memory bulk transfer workload, and one would expect that, in theory, it should provide benefits similar to the iperf microbenchmark for large VM memory sizes. We performed a range of live VM migrations for VM memory sizes ranging from 16GB to 64GB. The migrated VMs execute a kernel compilation task, which changes and dirties a significant number of pages. Figure 6 shows the mean and the 95% confidence intervals of the migration times under MPTCP and TCP over 15 runs for each memory size. Surprisingly, we find that despite involving data transfers of tens of gigabytes (a scenario where iperf shows clear benefits for MPTCP), the mean migration times for MPTCP is similar to TCP and both have overlapping confidence intervals, showing no statistical advantage for MPTCP. We attribute this behavior to several factors. Specifically the VM migration process in QEMU is performed by a single thread, which also performs operations such as checksumming. Our results show that for an uncongested 10 GigE network, we are *unable to perform these CPU operations at line speeds*. Consequently the VM migration process is unable to fully utilize the bandwidth of 10 GigE links, much less the aggregated 20 Gb/s bandwidth available to MPTCP. Due to these software bottlenecks, both TCP and MPTCP are able to use only a portion of the available network bandwidth and exhibit similar migration latencies.

In-memory Key-Value Stores. Our final real-world cloud application is Redis, an in-memory data-store, which has strict average and tail latency requirements. We use YCSB to generate the client workload for Redis using different ratios of read and write requests and different object sizes as discussed earlier. Since the in-memory network transfer sizes are in the range of kilobytes for this workload [39], in theory, we should expect behavior similar to the small data transfer sizes for our initial iperf experiment. Since MPTCP's performance is known to depend on the send and receive buffer sizes [8, 14], we ran our experiments using the default socket buffer size of 4MB in Linux and a large buffer size of 256MB that is more appropriate for high speed interfaces; both buffer sizes are adequate for the request sizes in this workload and showed similar results; consequently we only report results for the 256MB socket buffer size below.

Protocol	**TCP**		**MPTCP**	
Buffer	**256MB**		**256MB**	
Workload	**1KB**	**100KB**	**1KB**	**100KB**
Non-Congested Latency	**253**	3094	**305**	3611
Non-Congested Throughput	123652	10060	102711	8546
Congested Latency	369	∞	369	**3635**
Congested Throughput	93181	0	76974	7868

Table 2: MPTCP performance for Redis workloads. MPTCP shows higher latency in uncongested networks but performs better under congestion.

Table 2 depicts the mean read latencies and mean throughput for YCSB under MPTCP and TCP for the two object sizes. We see that both MPTCP and TCP exhibit similar latency and throughput with MPTCP under-performing TCP by around 10-20%. We attribute this slight degradation in performance to the more complex connection setup overhead of MPTCP when compared to TCP. This result also reveals an important insight—since the majority of network flows in data centers are smaller than 1MB in size [39], it is not advisable to make MPTCP the default protocol for all applications (since it will slightly worsen performance for many applications). Instead it is better to selectively use MPTCP only for applications characterized by long-lived flows that transfer significant amounts of data. Of course, if resiliency and congestion-avoidance are desired, than MPTCP is still suitable.

4.2 Multi-tenancy Performance

Our previous experiments assumed a single-tenant uncongested network with no competing applications and full NIC bandwidth available to the running application. In practice, each data center or cloud server is virtualized hosting multiple VMs (tenants). The network interface bandwidth will be partitioned across the co-located tenants, with a certain fraction of the bandwidth of each NIC dedicated to a tenant. Consequently, in practice, only a fraction of each 10 GigE NIC's bandwidth may be available to a tenant, while the rest of the NIC bandwidth may be used by other competing tenants. For now, we assume that co-located tenants do not fully congest the network (Section 4.3 considers multi-tenancy with congestion), We repeat our VM live migration experiments using with varying bandwidth allocation of 1, 2, 4, and 8 GB/s for the VM migration process.

Figure 7 shows the migration latency for MPTCP and TCP for a 32 GB VM. Since the amount of memory state transferred is tens of Gigabytes, MPTCP shows the expected 2x reduction in latency with

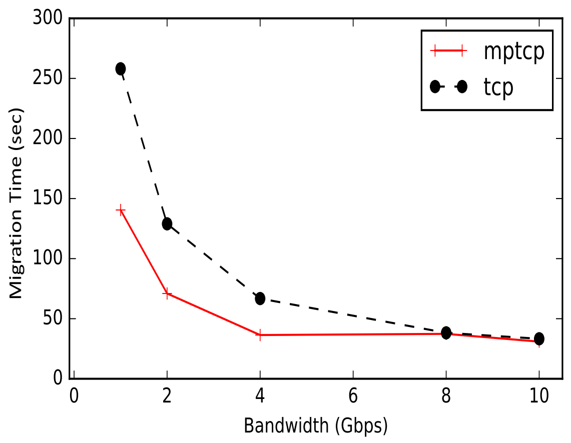


Figure 7: MPTCP yields lower migration times than TCP when the bandwidth allocated to the tenant VM between 1 and 8 Gbps.

double bandwidth aggregation when it is allocated 1 Gb/s per NIC. The migration process is able to perform per-page CPU operations at line speeds, allowing MPTCP to fully exploit the bandwidth on both interfaces. As the allocated NIC bandwidth is increased to 2 GB and beyond, MPTCP shows diminishing benefits over TCP and the advantage of bandwidth aggregation disappears at 8 GB/s and beyond. We attribute this to the relative speed of CPU versus network—as the network becomes faster, the CPU is not able to keep up and consequently MPTCP's aggregation benefits are diminished.

We also repeated our experiments with YCSB and Redis in a multi-tenant scenario with limited bandwidth allocations. In this case, since the requested object sizes are small (1KB and 100 KB), MPTCP is unable to exhibit any benefit from its second flow even at 1 GB/s (as shown in Figure 4, transfer sizes need to be 1 MB or greater to see any benefits).

4.3 Performance in Congested Networks

Our next set of experiments quantify the benefits of MPTCP in congested settings. Our premise is that MPTCP should be more robust to network congestion on some paths since it can utilize the bandwidth available on other paths [34] in contrast to TCP. We run experiments to quantify these benefits.

VM migration. We run the congestion generator described in Section 3.3 to introduce congestion on all interfaces and repeat our live migration experiments. We measure migration latencies for 16, 32, and 64 GB VMs.

Figure 8 shows a box-plot of the observed latencies with the mean, and the 95th percentile (the whisker) values marked. Our results show that MPTCP provides significant latency reduction over TCP in all cases, with reduction of 30% to 50% compared to TCP across the runs. The benefits vary due to the variable nature of the generated congestion. These benefits are due to MPTCP's ability to send more data on the less congested path(s) when some path(s) are

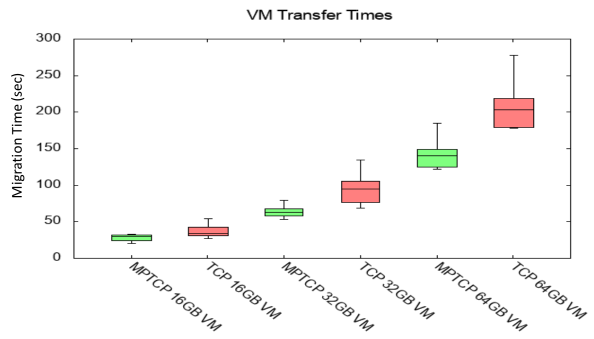


Figure 8: Under congestion, MPTCP provides better VM migration time by up to 33% compared to TCP.

congested as well as the bandwidth aggregation (with higher overall available bandwidth).

Next, we repeat the Redis and YCSB experiments under congestion. As shown in Table 2, for 1 KB object sizes, MPTCP offers no benefit over TCP, since at least 500KB flows are needed for MPTCP to activate the second flow. Both TCP and MPTCP latency and throughput worsen significantly due to congestion.

However, for 100KB objects, MPTCP offers significant benefits—it is able to provide similar (~3.6 seconds) latency when compared to no congestion. TCP requests, on the other hand, begin to time-out, resulting in YCSB request failures and infinite latencies. Thus, in the presence of congestion, MPTCP performance is more robust than TCP.

4.4 Robustness in the presence of link failures

In data centers, link failures can occur due to a variety of reasons, including, hardware, and software failures [17]. Since MPTCP utilizes multiple NICs and multiple paths, failure along one of the subflow paths can still enable MPTCP to communicate using the other path, yielding greater robustness. In this experiment, we test MPTCP's ability to provide fault-tolerance in the face of link failures with the VM migration workload [3]. We repeat our live-migration experiments described previously with both TCP and MPTCP, migrating a VM with 16 GB memory, running Kernel compile. After the migration operation starts, we induce a failure on one of the two links.

Table 3 shows the migration latency seen under MPTCP. TCP yields a migration latency of "infinity" on account of TCP connection/timeout failures that are seen after the link failure. MPTCP, on the other hand, is able to successfully complete the migration every time despite the failure, albeit with a degradation in migration completion time when compared to the no failure case. This experiments reveals a key benefit, namely MPTCP provides robustness to link failures and enables applications to continue execution, while TCP sees connection failures (unless it is manually switched to the other redundant path).

[3]We run similar experiments with other workloads but do not include them due to space limitations.

Table 3: VM Migration times for 16GB VMs in the presence of network failures. MPTCP is more robust to network failures

	No Failure	Failure
TCP	98 Sec	∞
MPTCP	94 Sec	110 sec

5 PERFORMANCE OF DISK-INTENSIVE WORKLOADS

While the previous section considered data center workloads that are memory-intensive with no disk I/O, disk intensive applications are an important class of data center workloads. Bulk data transfer applications such as disk backups, database-driven web applications, and distributed data processing platforms such as Spark and Hadoop, are all examples of workloads that perform significant disk I/O operations while also relying on network I/O. Our next class of applications, discussed in this section, involve disk-intensive workloads.

Bulk-data transfer Microbenchmarks. To establish a baseline for MPTCP performance for disk-intensive applications, we consider a microbenchmark where two simple tools, namely, *FTP* and *rsync* are used to perform bulk copy of large VM disk images from one server to another. In our experiments, We assume that this data is being transferred from one SSD to another. Note that, in uncongested settings, a 10GigE network is no longer the limiting resource—the SSD has a maximum read speed of only 550 MB/s, well below the bandwidth of a 10 GigE link.

Spark Distributed Data Processing. Distributed data-intensive workloads such as Spark are popular data center and cloud workloads. These workloads often require nodes to load data from disk, run computations, and then perform a shuffle operation to aggregate results, which mostly takes place in memory. For applications such as these, MPTCP's bandwidth aggregation and congestion control can be useful features as applications like Spark rely on fast networks to transfer data between the distributed nodes.To evaluate the effect of bandwidth aggregation, we use Spark 1.6.2, and run the TeraSort [24] benchmark with an input file size of 100 GB. Input files to Spark are stored on magnetic disks on HDFS with a replication factor of 3.

Similar to our experiments for memory-intensive workloads, we evaluate MPTCP's performance compared to TCP in (i) uncongested networks, (ii) multitenant settings, and finally (iii) congested networks.

5.1 Performance in Uncongested Networks

Our first experiment evaluates performance in uncongested networks with no interference from other applications.

Disk Bulk Transfer Microbenchmarks We use rsync and FTP to transfer a VM disk image file from an SSD disk on one server to another. In our experiments, we vary the disk image size from 1 GB to 150 GB and measure the total transfer time under MPTCP and TCP. Since TCP and MPTCP performance can be sensitive to send and receive socket buffer sizes, we repeat each experiment for a range of buffer sizes [8, 14].

Figure 9(a) shows the mean file transfer times for rsync, while Figure 9(b) depicts the mean transfer time seen by FTP versus rsync for a 150GB file copy. Since SSD read speeds are far lower than the full 10GigE network bandwidth (our SSD has a maximum sequential read speed of 550 Mb/s), bulk data transfers are disk-bound and unable to exploit the available network bandwidth of a single NIC, let alone the aggregated bandwidth of the dual NICs. Thus, TCP and MPTCP transfer times are nearly identical for a range of file sizes. We also note that, while a larger buffer improves performance of high speed links, it provides these benefits for both TCP and MPTCP without any notable differences between the two.

When using ssh transport, rsync performs data encryption, which adds to its overhead, while FTP performs unencrypted bulk data transfers. Consequently, Figure 9(b) demonstrates slightly lower completion times for FTP over rsync due to the elimination of encryption overheads. However, the disk bound nature of the transfers still dominates, causing TCP and MPTCP performance to be near identical.

Spark TeraSort workload. We use Spark TeraSort to sort a 100 GB data set under MPTCP and TCP in uncongested network. Figure 10 shows a box plot of completion times across 15 run for both TCP and MPTCP. Note that our Spark experiments uses traditional magnetic drives (HDDs), which are slower than SDDs, making the workload even more disk bound than the SDD case. As a result, neither TCP not MPTCP can utilize the full bandwidth of 10 GigE in uncongested settings. The disk bound nature of TeraSort results in similar completion times under MPTCP and TCP (around 12 minutes for a 100 Gb sort) with MPTCP unable to exploit any benefits of bandwidth aggregation. Our analysis of the Spark job shows that the vast majority of time (around 11 minutes out of 12) is spent on disk reads, while the actual sort takes between 10 and 30 seconds only. The network shuffle phase is thus a small fraction of the total completion time, and a faster network can only improve this part of the job, yielding insignificant improvements to the total completions time.

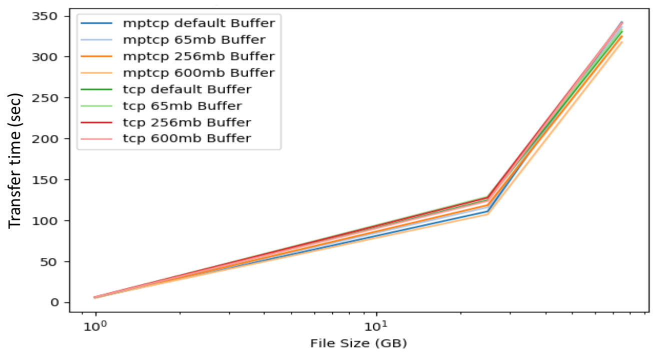
5.2 Multi-tenancy performance

We repeat our previous experiments in a multi-tenant setting by assuming that each tenant is dedicated 1 Gb/s bandwidth per NIC.

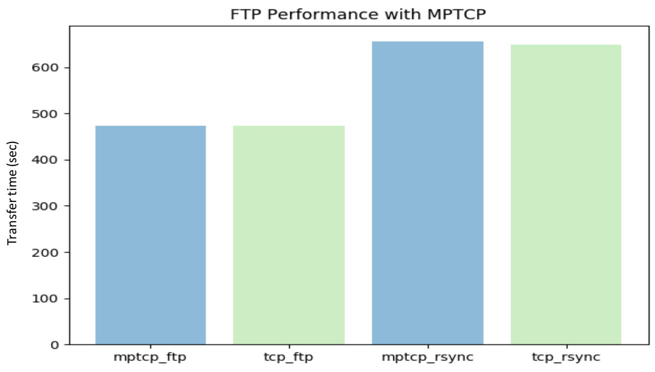
Figure 11 shows the file transfer times to transfer a 30 GB file using both MPTCP and TCP. Note that on a 1 Gb/s network link, the network, rather than the disk, is the bottleneck since the maximum SSD read rate of 540 MB/s ($\approx$ 4 Gb/s) is around 4x the maximum network rate. Consequently, MPTCP's bandwidth aggregation across two 1 Gb/s paths yields double the network throughput compared to TCP as shown in Figure 11. However, the Figure shows that this advantage disappears when the network speed is increased to 10 Gb/s since the network becomes faster than the disk and the disk speed becomes the bottleneck.

We also repeat the Spark TeraSort workloads under 1 Gb/s network bandwidth limits. In this case however, there was no noticeable difference between MPTCP and TCP. This is because the disk I/O dominates the completion times and the short network shuffle does not provide any noticeable improvements for MPTCP under 1 Gb/s speeds. Since TeraSort does not stress the network, we omit the graphs for space consideration.

Our results show that, in multi-tenant settings, MPTCP can improve performance of large SSD streaming reads and replicating



(a) Using rsync with SSD disks, I/O bandwidth is the bottleneck and MPTCP provides no performance improvement.



(b) While FTP provides better performance compared to rsync, MPTCP results in no improvement over TCP.

Figure 9: Disk-bound bulk transfer workloads do not see MPTCP improvements in uncongested settings.

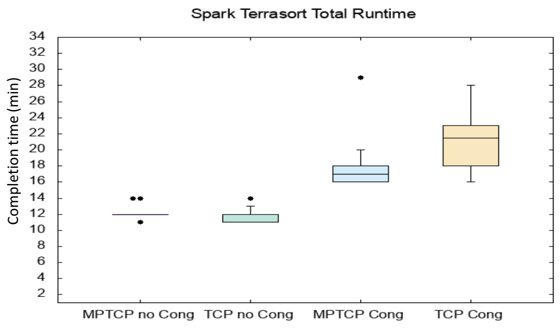


Figure 10: MPTCP can improve Spark TeraSort performance by 20% under network congestion. No benefits are seen in uncongested settings.

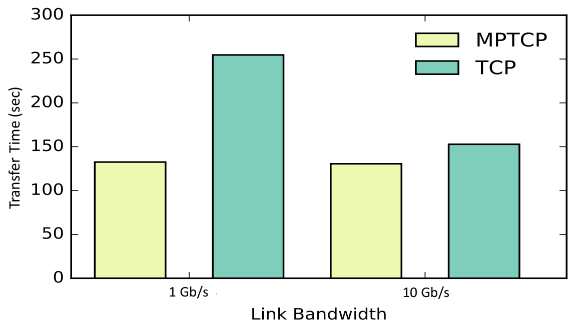


Figure 11: MPTCP shows better performance compared to TCP when the network bandwidth allocated a tenant is a fraction of the 10GigE NIC bandwidth.

large objects when the network bandwidth allocation is low. However, these benefits disappear under slower HDDs or under higher network bandwidth allocation.

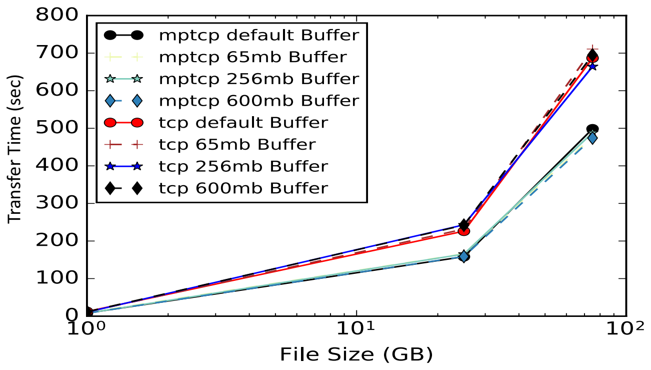


Figure 12: MPTCP speeds-up file transfers during congestion by up to 30% compared to TCP.

5.3 Performance Under Network Congestion

Rsync. Our final experiments introduces congestion using our congestion generator and measures the performance of bulk disk transfers and Spark TeraSort under MPTCP and TCP in the presence of congestion.

Figure 12 shows the file transfer times using rsync for files ranging from 1 GB to 100 GB, under MPTCP and TCP using different socket buffer sizes. In the presence of congestion, MPTCP improves file transfer times by up to 30% over TCP by exploiting both bandwidth aggregation and use of less congested paths. We also observe that larger socket buffers offer a small marginal benefit for MPTCP.

Spark. To test the performance of Spark under congestion, we run our congestion generator described earlier on all machines in the cluster, see Figure 2, with machines on the left side of the network acting as sources of congestion while machines on the right-side of the network acting as receivers. This generated traffic can be viewed as background network traffic on the whole network. We note that the congestion generator is very light-weight and does not consume any significant CPU or memory resources on the source or destination.

The two right most box plots in Figure 10 show the performance of MPTCP vs TCP under high congestion. MPTCP is on average 20% faster than TCP. When looking at the per phase speed, the sort phase in some runs grew to over four minutes under TCP, while it grew to around two minutes for MPTCP. The distribution in running times is also much less dispersed with MPTCP compared to TCP. This is expected as MPTCP's multiple links allows more data to be transferred under extreme congestion.

6 PERFORMANCE IN PUBLIC CLOUDS

Our previous experiments assumed a private cloud that enabled controlled network experiments. Our final experiment examines how our observations of MPTCP performance comes over to public clouds.

Major cloud providers, such as Amazon EC2, Microsoft Azure, and IBM Softlayer, all employ a virtualized data center architecture and offer VMs to end-customers. These VMs can be configured with multiple logical NICs—for example, a private IP NIC and a public IP NIC, or in other cases, multiple NICs of each type.

Although the internal data center details of public clouds are not public knowledge, it is reasonable to assume they employ modern 10 GigE or even 40 GigE LANs, and redundant LAN topologies. Thus, in principle, utilizing the multiple logical NICs of a VM via MPTCP should offer similar benefits (and similar drawbacks) as that seen in our previous experiments.

As discussed earlier, we conduct our evaluation on Azure public cloud instances in the US East region. In particular, we use Standard D4 V2 and Standard D13 V2 VM types. We configure each VM with two network interfaces, with each interface assigned an IP on a different subnet. We examine the effectiveness of MPTCP's bandwidth aggregation using iperf to measure the bandwidth between two VMs, each with two network interfaces.

We ran the iperf memory-to-memory transfer benchmark on the two Azure VMs. Figure 13 shows the bandwidth observed on the two virtual interfaces at the receiver VM over time for both MPTCP and TCP. Our experiment yields an unexpected result. First, we observe that TCP is able to outperform MPTCP by a significant margin. Second, we observe that, under MPTCP, the bandwidth of the two subflows fluctuate significantly over time, but the sum of the two bandwidth, i.e, aggregate MPTCP bandwidth, shows much less variance. In particular, when the bandwidth utilized by one subflow falls, there is a simultaneous corresponding increase on the other interface, keeping the total bandwidth relatively stable.

Since the internal cloud configuration is not public information, we can only infer the cause of this surprising behaviour. It is clear that the total bandwidth usage of the VM *across all NICs* is being throttled. This is why the sum of the bandwidth across flows is relatively stable. It is possible that the two logical NICs maybe mapped internally to the same physical NIC—in that case, the two MPTCP flows are competing with one another and a rise in one flow's bandwidth results in a drop in the other.

Even if the VMs logical NICs were to be mapped to different physical NICs, the aggregate bandwidth usage appears to be capped, which inherently prevents MPTCP from exploiting the presence of

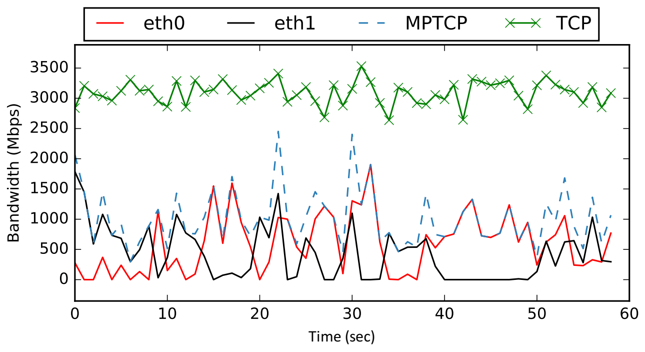


Figure 13: Time-varying bandwidths seen in the Azure public cloud. Interface bandwidths vary significantly, and TCP vastly outperforms MPTCP.

multiple NICs and multiple paths. Under these conditions, bandwidth aggregation as a negative impact on application performance. [4]

Thus, under current configurations, our private cloud results *do not* carry over to the public cloud (at least for the Azure cloud). However, if public cloud providers were to remove bandwidth throttling, or increase the total allocation significantly, we expect cloud VMs to behave like our private cloud VMs. In that case, similar insights to our private cloud results will apply with regard to MPTCP's benefits and drawbacks.

7 LESSONS LEARNED AND DISCUSSION

In this section, we summarize our findings and discuss their implications for today's clouds and data centers.

Uncongested Networks: In an uncongested network, we find that, for 10 GigE links and beyond, the network is typically not the bottleneck for today's applications. Other resource bottlenecks prevent today's unoptimized applications from exploiting multiple paths and bandwidth aggregation benefits of MPTCP.

For memory intensive workloads, we find that the "ideal" iperf benchmark can exploit the bandwidth aggregation benefits of MPTCP since it is fully network bound. Live VM migration, which is the closest memory-to-memory transfer workload, however is unable to transmit pages at line speed to extract these benefits. Redis in memory databases typically do not have flow sizes that are large enough for MPTCP to provide any benefits. For disk-intensive workloads, the slower speeds of modern SSDs and HDDs relative to network speeds leave disk-intensive workloads unable to exploit MPTCP benefits.

Multi-tenant workloads. In multi-tenant scenarios where the network bandwidth allocated to a tenant is smaller than full link benefits, MPTCP shows some benefits in selected scenarios. For memory intensive workloads, live migrations see MPTCP's bandwidth aggregation benefits with a lower bandwidth allocation while small requests in Redis do not see any benefits. For disk intensive workloads running on SSD, MPTCP yields benefits whenever the allocated network bandwidth is lower than the disk speeds. However, HDDs continue

[4]We note that Amazon's AWS also does not allow bandwidth aggregation when using multiple NICs: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-eni.html

to be a bottleneck even with a 1GB/s network allocation speed and do not see any benefits from MPTCP.

Congested Network and Failures. MPTCP's advantages over TCP increase in the presence of network congestion since MPTCP can exploit redundant paths. Both memory intensive and disk intensive workloads see benefits of 20 to 50% compared to TCP. Even "small flows" such as Redis requests see benefits at high levels of congestion.

Finally, MPTCP's robustness to network failures is a major advantage. While networked applications fail with TCP, they execute successfully, albeit with a performance degradation, when using MPTCP.

Implications. Due to these mixed results, whether and when to use MPTCP in data centers and clouds depends on the scenario. In purely uncongested, single tenant situations, our results do not show any benefits of deploying MPTCP. For multi-tenant scenarios, our results argue for selective deployment of MPTCP for certain applications. Specifically, MPTCP should be deployed for bulk data transfers, e.g., VM migrations or fast SSD bulk transfers, and not be deployed for request-response server applications where small requests dominate. If the data center network is characterized by frequent congestion in the network or experiences frequent link failures, our results argue for a "blanket" deployment of MPTCP for all applications to exploit its robustness properties.

In the case of public clouds, unless the cloud provider allows customers to fully exploit the bandwidth available over multiple NICs (by removing bandwidth limits), customers will not see the above benefits.

Beyond 10 GigE networks. Our current experiments utilize a 10 GigE network. Today, 40 GigE networks are already seeing early deployments, thus it is instructive to understand how our insights carry over to 40 GigE networks and beyond. Broadly speaking, our findings are generally applicable to 40 GigE networks and beyond, but with some caveats. First, our congestion and failure results do not depend on network speeds and emphasize robustness—they will apply to faster networks as well when they experience congestion or failures. Our multi-tenant results depend largely on the bandwidth allocated to a tenant rather than the full link speed, thus, a tenant that is rate limited to 1 Gb/s should see similar behaviour on faster networks. Similarly, when the behaviour depends on disk speeds versus network speed, the tenant's allocation largely determines the behaviour. If tenants are allocated more bandwidth on 40 Gig LANs, the benefits will diminish. In uncongested single tenant setting, the network speed does matter, but we find MPTCP to be ineffective since 10 Gig networks are not the bottleneck, and faster networks will be even less so.

8 RELATED WORK

The earliest work discussing multi-path networking dates back to 1975 when the concept of "dispersity routing" was discussed. Since then, multiple solutions have been proposed with many surveys discussing multi-pathing techniques and benefits [21, 29, 33]. Resource pooling for network resources [43], motivates the need for multi-path at the transport layer and lays the groundwork for MPTCP.

The seminal work on MPTCP in data centers by Raiciu et al. [34], shows the shortcomings of ECMP and proposes MPTCP to improve data center performance and robustness using state-of-the-art data center network topologies. The evaluation in that work is network-centric rather than application-centric, with a bias towards network simulations. Work on MPTCP [26, 35–37] focused on establishing its feasibility and fairness properties. In this paper, we assume that the fairness properties hold and have performed an application-centric empirical study in different network environments.

MPTCP has high appeal in mobile environments, where WiFi and cellular networks can be combined to provide high throughput, improved connectivity, and lower costs [7, 11]. Some of this work raise concerns over MPTCP's short flow performance and how the primary subflow can impact overall connection-level performance[7, 11]. Path prioritization for MPTCP is explored in [20] for video streaming applications to reduce energy and cellular network usage.

Path selection is crucial in multi-path networking [25]. Path-capture [45] is a well-known exploration-exploitation trade-off. Path characteristics can only be obtained by sending data, but this "exploration" can degrade performance if the secondary paths have very poor performance, especially with coupled congestion control. Initial path selection and the RTT difference between paths can have drastic and unexpected effects [4]. In data centers, this problem is not very prominent with uniform links. To address these issues, an alternative to MPTCP's packet striping using fountain-codes is proposed in [10].

Mitigating congestion and improving load balancing in data centers is an active research area. Most approaches involve the use of network switches to perform "intelligent" load balancing. For example, *decentralized* approaches to traffic management rely on implementing congestion-detection and flow-management policies in either custom or off-the-shelf network switches [2, 19]. MPTCP on the other hand requires no extra support from the network, and works independently of other network management components. Distributed approaches such as Conga [2] and others [23, 42, 47] use high-speed programmable switches to implement congestion-mitigating policies in switches. A good comparison of load balancing approaches including ECMP and MPTCP can be found in [23]. Finally, characterization of network traffic in production data centers can be found in [6, 18, 40].

9 CONCLUSION

In this paper, we empirically study MPTCP's performance with multiple data center workloads and network conditions. Our results show that for 10 Gbps networks, many data center applications are constrained by resources other than network bandwidth, limiting the efficacy of MPTCP. In several cases, this can result in worsening performance compared to the performance with TCP. We demonstrate that MPTCP may result in better performance when flow sizes are sufficiently large to compensate for the overheads of MPTCP and to saturate the capacity of more than one interface. Therefore, performance can improve when using MPTCP on networks with limited bandwidth, e.g., VMs with rate limited network allocations. However, due to the current public cloud network architectures, benefits are not seen in practice. We conclude that while performance improvements from bandwidth aggregation are unlikely, MPTCP may still provide desired congestion and resiliency performance for some application in data centers.

Acknowledgment This research was supported in part by NSF grants CNS-1413998, 1763834, 1802523, 1836752, and 1405826, a gift from Huawei, and a grant from Microsoft Azure.

REFERENCES

[1] M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. In *ACM SIGCOMM Computer Communication Review*, volume 38, pages 63–74. ACM, 2008.

[2] M. Alizadeh, T. Edsall, S. Dharmapurikar, R. Vaidyanathan, et al. Conga: Distributed congestion-aware load balancing for datacenters. In *ACM SIGCOMM Computer Communication Review*, volume 44, pages 503–514, 2014.

[3] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan. Data center tcp (dctcp). In *ACM SIGCOMM computer communication review*, volume 40, pages 63–74. ACM, 2010.

[4] B. Arzani, A. Gurney, S. Cheng, R. Guerin, and B. T. Loo. Deconstructing mptcp performance. In *Network Protocols (ICNP), 2014 IEEE 22nd International Conference on*, pages 269–274. IEEE, 2014.

[5] S. Barré, C. Paasch, and O. Bonaventure. Multipath tcp: from theory to practice. *NETWORKING 2011*, pages 444–457, 2011.

[6] T. Benson, A. Akella, and D. A. Maltz. Network traffic characteristics of data centers in the wild. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pages 267–280. ACM, 2010.

[7] Y.-C. Chen, Y.-s. Lim, R. J. Gibbens, E. M. Nahum, R. Khalili, and D. Towsley. A measurement-based study of multipath tcp performance over wireless networks. In *ACM IMC*, pages 455–468, 2013.

[8] Y.-C. Chen and D. Towsley. On bufferbloat and delay analysis of multipath tcp in wireless networks. In *Networking Conference, 2014 IFIP*, pages 1–9. IEEE, 2014.

[9] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears. Benchmarking cloud serving systems with ycsb. In *Proceedings of the 1st ACM symposium on Cloud computing*, pages 143–154. ACM, 2010.

[10] Y. Cui, L. Wang, X. Wang, H. Wang, and Y. Wang. Fmtcp: A fountain code-based multipath transmission control protocol. *IEEE/ACM Transactions on Networking (TON)*, 23(2):465–478, 2015.

[11] S. Deng, R. Netravali, A. Sivaraman, and H. Balakrishnan. Wifi, lte, or both?: Measuring multi-homed wireless internet performance. In *ACM IMC*, pages 181–194, 2014.

[12] Z. Ding, R. R. Hoare, A. K. Jones, and R. Melhem. Level-wise scheduling algorithm for fat tree interconnection networks. In *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, page 96. ACM, 2006.

[13] S. Ferlin, Ö. Alay, T. Dreibholz, D. A. Hayes, and M. Welzl. Revisiting congestion control for multipath tcp with shared bottleneck detection. In *IEEE INFOCOM*, pages 1–9, 2016.

[14] S. Ferlin-Oliveira, T. Dreibholz, and Ö. Alay. Tackling the challenge of bufferbloat in multi-path transport over heterogeneous wireless networks. In *IEEE IWQoS,*, pages 123–128, 2014.

[15] A. Ford, C. Raiciu, M. Handley, S. Barre, and J. Iyengar. Architectural guidelines for multipath TCP development. RFC 6182, Mar. 2011.

[16] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure. TCP extensions for multipath operation with multiple addresses. RFC 6824, 2013.

[17] P. Gill, N. Jain, and N. Nagappan. Understanding network failures in data centers: measurement, analysis, and implications. In *ACM SIGCOMM Computer Communication Review*, volume 41, pages 350–361. ACM, 2011.

[18] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta. Vl2: a scalable and flexible data center network. In *ACM SIGCOMM computer communication review*, volume 39, pages 51–62. ACM, 2009.

[19] M. P. Grosvenor, M. Schwarzkopf, I. Gog, R. N. Watson, A. W. Moore, S. Hand, and J. Crowcroft. Queues don't matter when you can jump them! In *NSDI*, pages 1–14, 2015.

[20] B. Han, F. Qian, L. Ji, V. Gopalakrishnan, and N. Bedminster. Mp-dash: Adaptive video streaming over preference-aware multipath. In *ACM CoNEXT*, pages 129–143, 2016.

[21] H. Han, S. Shakkottai, C. V. Hollot, R. Srikant, and D. Towsley. Multi-path TCP: A joint congestion control and routing scheme to exploit path diversity in the Internet. *IEEE/ACM Transactions on Networking*, 14:1260–1271, 2006.

[22] S. Hassayoun, J. Iyengar, and D. Ros. Dynamic window coupling for multipath congestion control. In *IEEE ICNP*, pages 341–352, 2011.

[23] K. He, E. Rozner, K. Agarwal, W. Felter, J. Carter, and A. Akella. Presto: Edge-based load balancing for fast datacenter networks. *ACM SIGCOMM Computer Communication Review*, 45(4):465–478, 2015.

[24] E. Higgs. Spark Terasort. https://github.com/ehiggs/spark-terasort.

[25] P. Key, L. Massoulié, and D. Towsley. Path selection and multipath congestion control. In *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*, pages 143–151. IEEE, 2007.

[26] R. Khalili, N. Gast, M. Popovic, U. Upadhyay, and J.-Y. Le Boudec. MPTCP is not pareto-optimal: Performance issues and a possible solution. In *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies (CoNext)*, pages 1–12. ACM, 2012.

[27] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1):14–76, 2015.

[28] T. A. Le, C. S. Hong, M. A. Razzaque, S. Lee, and H. Jung. ecmtcp: an energy-aware congestion control algorithm for multipath tcp. *IEEE communications letters*, 16(2):275–277, 2012.

[29] M. Li, A. Lukyanenko, Z. Ou, A. Yla-Jaaski, S. Tarkoma, M. Coudron, and S. Secci. Multipath transmission for the internet: A survey. *IEEE Communications Surveys Tutorials, vol. PP*, (99):1–41, 2016.

[30] K. Ousterhout, R. Rasti, S. Ratnasamy, S. Shenker, B.-G. Chun, and V. ICSI. Making sense of performance in data analytics frameworks. In *NSDI*, volume 15, pages 293–307, 2015.

[31] C. Paasch and S. Barre. Multipath TCP in the Linux kernel. http://www.multipath-tcp.org.

[32] C. Paasch, S. Ferlin, O. Alay, and O. Bonaventure. Experimental evaluation of multipath tcp schedulers. In *Proceedings of the 2014 ACM SIGCOMM workshop on Capacity sharing workshop*, pages 27–32. ACM, 2014.

[33] J. Qadir, A. Ali, K.-L. A. Yau, A. Sathiaseelan, and J. Crowcroft. Exploiting the power of multiplicity: a holistic survey of network-layer multipath. *IEEE Communications Surveys & Tutorials*, 17(4):2176–2213, 2015.

[34] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley. Improving datacenter performance and robustness with multipath tcp. In *ACM SIGCOMM Computer Communication Review*, volume 41, pages 266–277. ACM, 2011.

[35] C. Raiciu, M. Handly, and D. Wischik. Coupled congestion control for multipath transport protocols. RFC 6356, Oct 2011.

[36] C. Raiciu, D. Niculescu, M. Bagnulo, and M. J. Handley. Opportunistic mobility with multipath TCP. pages 7–12, 2011.

[37] C. Raiciu, C. Paasch, S. Barre, A. Ford, M. Honda, F. Duchene, O. Bonaventure, and M. Handley. How hard can it be? Designing and implementing a deployable multipath TCP. 2012.

[38] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch. Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In *Proceedings of the Third ACM Symposium on Cloud Computing*, SoCC '12, pages 7:1–7:13, New York, NY, USA, 2012. ACM.

[39] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren. Inside the social network's (datacenter) network. In *ACM SIGCOMM Computer Communication Review*, volume 45, pages 123–137. ACM, 2015.

[40] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren. Inside the social network's (datacenter) network. In *ACM SIGCOMM Computer Communication Review*, volume 45, pages 123–137. ACM, 2015.

[41] A. Trivedi, P. Stuedi, J. Pfefferle, R. Stoica, B. Metzler, I. Koltsidas, and N. Ioannou. On the [ir]relevance of network performance for data processing. In *8th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 16)*, Denver, CO, 2016. USENIX Association.

[42] E. Vanini, R. Pan, M. Alizadeh, T. Edsall, and P. Taheri. Let it flow: Resilient asymmetric load balancing with flowlet switching. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. USENIX Association.

[43] D. Wischik, M. Handley, and M. B. Braun. The resource pooling principle. *ACM SIGCOMM Computer Communication Review*, 38(5):47–52, 2008.

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

[45] D. Wischik, C. Raiciu, and M. Handley. Balancing resource pooling and equipoise in multipath transport. *ACM SIGCOMM*, 2010.

[46] T. Wood, P. J. Shenoy, A. Venkataramani, and M. S. Yousif. Black-box and gray-box strategies for virtual machine migration. In *NSDI*, 2007.

[47] K. Zarifis, R. Miao, M. Calder, E. Katz-Bassett, M. Yu, and J. Padhye. Dibs: Just-in-time congestion mitigation for data centers. In *Proceedings of the Ninth European Conference on Computer Systems*, page 6. ACM, 2014.

[48] J. Zhao, J. Liu, H. Wang, and C. Xu. Multipath tcp for datacenters: From energy efficiency perspective. In *INFOCOM 2017-IEEE Conference on Computer Communications, IEEE*, pages 1–9. IEEE, 2017.