Improving Data Center Network Performance through Path Diversity

Charles Joshua Alba, Kyle Dominic Gomez, and Rene Josiah Quinto
Electrical and Electronics Engineering Institute
University of the Philippines - Diliman
Quezon City, Philippines
charles.alba@eee.upd.edu.ph, kyle.gomez@eee.upd.edu.ph, and rene.josiah.quinto@eee.upd.edu.ph

***Abstract*—Datacenter networks allow the operation of Internet services by partitioning requests to multiple servers, and aggregating results back to form the response sent to the users. Services, differentiated by their workload profiles, require a network that can guarantee high throughput, and low flow completion time. To achieve these, this paper focuses on Multipath TCP (MPTCP) as a transport protocol in the context of datacenters, by making better use of the network topology. In addition, changes in the packet forwarding protocol within network switches and the Generic Offload Receiver (GRO) layer within network servers will be employed to make up for MPTCP's undesired behavior within datacenters such as network link hotspots. We created a simulated datacenter environment wherein various tests were done to measure throughput, goodput, flow completion time, and mean network utilization, tweaking flow types, network traffic conditions, switch forwarding protocol, and reorder resiliency in hosts in the process. Analyzing the data, the study found that packet spraying turned out to be beneficial for the network in terms of throughput, goodput, and mean network utilization. Reorder resiliency, on the other hand, resulted in a decrease in the performance of the network.**

I. Introduction

To keep up with increasing demand for online services, requests and operations are usually serviced by partitioning tasks into multiple servers within a datacenter. These servers are then arranged in a special topology to ensure quick intercommunication between each other, regardless of packet stream length or size. As a consequence, servers have multiple paths between each other. There lies a promising performance benefit by taking advantage of this path diversity, in which the traditional transport protocol, TCP, cannot provide.

A. Servicing Increasing Demand with Datacenter Networks

The demand for online services has been increasing steadily due to their popularity. Moreover, the competitive market have pushed for innovations to improve services, resulting in increased computational load [1]. To keep up with demand that increases in both quantity and complexity, companies need to expand and upgrade their resources, increasing costs [2], [3]. One way to meet this demand is to distribute service requests to multiple servers, then aggregating responses to form the final result (also known as the partition/aggregate design pattern) [4]. Infrastructures built with this demand distribution property are called datacenter networks (DCNs).

Apart from meeting the increasing demand, this provides several benefits. Firstly, by having redundant servers, services can be maintained even during partial network maintenance [1]. Secondly, an organized and redundant network infrastructure eases management [1]. Finally, distributing requests to different servers increases server utilization, proving to be more cost-effective and scalable [5].

B. Connection-Rich Nature of Datacenter Network Topologies

Partitioned requests are distributed among servers within a datacenter (end-hosts) through packet streams (flows). Since majority of the flows stay within a datacenter [6], the network topology must guarantee a speedy and reliable connection between local end-hosts. These properties can be characterized with sufficient interconnection bandwidth [7], [8] and fault-tolerance [9].

Requests must be serviced quickly and aggregated back to the user, which means that the server response delays also add up and affect the performance of the service [4]. End-hosts also need to keep their internal data structures fresh [4] through frequent large transfers of data. Therefore, datacenters must also guarantee low latency for short flows, and high throughput to service large flows [4], [10].

Current datacenter network topologies guarantee the previous characteristics through different approaches. Switch-centric topologies [8], [11] rely solely on switches and routers to forward traffic across the network, which ensures speedy delivery due to specialized hardware at the expense of scalability. Server-centric topologies [12], [13] allow servers to forward packets as well, however software routing may be slow and can affect host-based applications [14].

Topologies can be also designed to be modular [13] or hierarchical [11] to increase network capacity through scalability [7]. This introduces multiple redundant links and paths between servers within the network, establishing a connection-rich nature [10], [15].

C. Better Network Utilization through Multiple Paths

Host-to-host connectivity within a datacenter was originally facilitated using the Transmission Control Protocol (TCP) [4], [16]. TCP connections make use of a single network path between end-hosts [17]. Since it is typical for two end-hosts in a datacenter network to have multiple paths, TCP is unable to fully utilize the available paths between end hosts [18], and may not be able to achieve optimal network utilization.

To make use of the available resources in a path-diverse network, Multipath TCP (MPTCP) was proposed as an experimental protocol by the IETF. MPTCP uses parallel TCP connections [19], [20], which increases throughput and resiliency of the network [18].

II. Review of Related Work

While MPTCP promises performance benefits such as increased throughput between two hosts, it suffers from problems which hinder or negate its advantages.

A. MPTCP's Connection Establishment Penalizes Short Flows

MPTCP, being designed as an extension for TCP [19], inherits its congestion control mechanisms. One of which is Slow Start [21], where each subflow starts sending a limited amount of packets, then exponentially increases its sending rate until a link capacity/threshold is reached. Exceeding link capacity can result to packet loss due to switch overflows, making the receiver fail to acknowledge the packet. This subsequent action will cause the sender to retransmit the packet due to its internal retransmission timeout, or due to receiving three duplicate acknowledgements (TCP Fast Retransmit) for a previously sent packet, in addition to reducing the sending rate of the sender.

Since majority of the flows on a datacenter are short flows [22] and are of a bursty nature [4], MPTCP would cause these flows to stay in the Slow Start phase [23], where the sending rate is initially limited, then is increased drastically. Therefore, when short flows lose packets due to congestion, retransmission is necessary, and this increases the flow completion time [24].

In addition, MPTCP also inherits TCP's handshake protocol, in order to create and establish subflows between end-hosts. Since MPTCP cannot discern between short and long flows, it cannot decide as to how many subflows should be established to optimally send packets across the network. Therefore, a mismatch in the number of subflows, specifically for short flows would further increase the flow completion time due to the extra packets required to establish a connection.

To minimize the flow completion time for short flows, connections can start by spraying packets over multiple paths. Connections then can switch back to MPTCP upon reaching a certain amount of packets [24], to make use of MPTCP's benefits over long flows such as increased throughput and better congestion control. However, the introduction of packet spraying in addition to using MPTCP would open this specific implementation to both problems associated with each.

B. Diffusing Network Hotspots through MPTCP with Packet Spraying

MPTCP uses Equal-Cost Multipath (ECMP), a switch-based algorithm, to forward packets through the network [16]. ECMP bases its forwarding on packet header fields [25] which is hashed and used to forward packets to paths. This means that all data belonging to one subflow will strictly follow one network path. Selected network paths by ECMP may traverse links that have already been chosen by another subflow, which will effectively decrease the maximum available capacity of the link. Areas in the network that are close to exceeding link capacity are called hotspots. Colliding subflows may exceed the link capacity for any given link in a datacenter network, resulting in full switch queues, leading to dropped packets, and ultimately decrease mean network utilization [7].

One way to avoid this is to use packet spraying [26]. Packet spraying spreads a sequence of packets through different candidate ports instead of forwarding packets of a single flow through a chosen port according to its hashed flow identifier as seen in ECMP. This scheme prevents the collision of multiple paths onto specific links (hash collision), and spreads the packets in the hopes of evenly distributing the load through all relevant links of the network.

However, distributing packets of a flow into different paths can lead to out-of-order packets if the paths have significantly different end-to-end delays [26]. End-hosts receiving packets that do not arrive in the order that they were sent may falsely perceive that the network is lossy and request for retransmissions [27]. The sending end-host will receive these retransmission requests and will falsely perceive that there is congestion in the network, cutting the congestion window in half which will then result in the reduction of throughput.

One way to avoid false congestion due to packet reordering is to adjust the time threshold of the end-hosts before they request for retransmission [24], effectively making end-hosts more tolerant to delays in the network. However, an improper configuration (mismatch) of this threshold will result in an increased delay before sending retransmission requests in actual congestion experienced in the network, greatly decreasing the flow completion time and penalizing short flows.

To increase the resiliency of the end-hosts against out-of-order packets while avoiding a threshold mismatch, one solution would be to increase the buffer size of the Generic Receive Offload (GRO) layer of the end-hosts [28]. The GRO layer works underneath the transport layer, and is responsible for bundling packets to decrease computational overhead versus individual per-packet processing. Increasing the buffer size enables it to accept packets that arrive earlier than those preceding it thereby increasing its resilience towards out-of-order packets. This is based on the assumption that, in a datacenter, delay differences between paths are significantly less than in non-datacenter networks.

C. Drawbacks Caused by MPTCP with a Global View

With Software Defined Networking (SDN), a global view of the network state can be provided to a central controller [29], [30], [31]. This enables network protocols to make use of global network information, such as link utilization and end-to-end delays [32], [33], to create more informed decisions. SDN approaches have introduced several benefits to MPTCP, such as increased throughput and network resiliency, by controlling the number of subflows and scheduling the routes of different connections to avoid both network hotspots and out-of-order packets [32].

These benefits, however, comes at the price of controller overhead [32]. SDN controllers need to communicate regularly with the network switches in order to issue forwarding rules to adapt with the situation, causing delays. Since most congestion events that happen in datacenter networks are because of traffic bursts that only lasts for a few microseconds (microbursts) [34], controller overhead becomes significant, and so SDN implementations cannot respond fast enough. Due to controller overhead, SDN implementations can also increase the flow completion time of connections, penalizing short flows. Moreover, SDN implementations generally do not scale with a large number of connections as the controller overhead also increases with the number of active connections [32].

III. Problem Statement and Objectives

A. Problem Statement

Common topologies in datacenter networks exhibit symmetry and path diversity [10]. This ensures that multiple paths of equal costs exist between two end-hosts [26]. In addition, an ideal datacenter network must guarantee high throughput for large flows, and low latency for short flows [4], [10].

Multipath TCP (MPTCP), an experimental protocol by the IETF, is a TCP extension that uses multiple paths over many connections [19]. By using multiple paths simultaneously, MPTCP aims to increase network utilization, specifically throughput, and increase "resilience of connectivity" [20]. This is done by employing Equal Cost Multi-Path (ECMP) [16], a switch-based forwarding algorithm. ECMP hashes subflows into separate paths using the packet's header information. However, hotspots in the network may occur when flows are hashed to paths with one or more links overlapping with each other [7], usually exceeding the link capacity. Because of this, the network experiences a drop in network utilization due to sender backoff [26].

Random packet spraying, an alternative to ECMP, can be used to resolve hotspots as it allows for a greater granularity for load balancing [10]. However, this can result in reordered packets at the receiver, triggering false retransmissions upon receipt of three duplicated packet acknowledgements [18], in addition to drastically reducing sender throughput [35]. This can be minimized by dynamically changing the retransmission threshold. However, a positive mismatch on the threshold may mean a slower response time for actual packet drops [35].

Minimizing retransmission due to out-of-order packets can be done by supplementing the function of the Generic Receive Offload (GRO) layer to fix the order of packets [36] in addition to merging packets together. This not only reduces computational overhead compared to per-packet processing, but makes the end-hosts more tolerant to packet reordering, and thus reduces retransmissions. However, reordered packets must arrive within a short delay of each other since the switch queues are limited and timeouts are smaller.

Also, to maintain backwards compatibility with TCP, MPTCP also suffers from the complexity in connection establishment [26] and slow start [21], which in turn penalizes short flows through higher flow completion times. To maintain lower flow completion times and lower latency for short flows, a specified amount of packets are sprayed through multiple paths initially before switching to MPTCP [24]. However, this means that it inherits problems from both implementations altogether.

Better network utilization can be also achieved using Software-Defined Networking [32], [33] with MPTCP. With a global view of the network, it can better utilize path diversity, have a better gauge on the number of subflows per connection, and ideally minimize the receipt of out-of-order packets. This solution benefits large data flows, but due to the added controller overhead, it may penalize short flows. In addition, it may also have some scalability issues [33].

We hypothesize that an increase in overall throughput and network utilization in MPTCP can be achieved through implementing packet spraying in the network instead of ECMP. Since packet spraying can introduce an eventual decrease in throughput, we hypothesize that this can be mitigated by creating reorder-resilient end hosts. By comparing the results we see from different experiments, we strengthen the basis for considering a reorder-resilient network for future datacenter networks, as well as potentially contribute to the growth of MPTCP as an experimental protocol.

B. Objectives

The objectives of this work are as follows: First is to experimentally prove that MPTCP benefits large flows, but penalizes short flows. Next, understand and prove that network hotspots occur due to ECMP-based switches. Lastly, observe and analyze the effects of packet spraying switches, as well as reorder-resilient hosts, to minimize network hotspots, and in turn benefit both short and large flows.

C. Scope and Limitations

The project will focus on datacenter networks, and assume ideal working conditions. More specifically, this project does not consider the possibility of switch failures, host failures, link damages that could cause degradation of performance or even disconnections. Among datacenter topologies, only fat tree topologies and possible variants will be considered.

While the nature and topology of datacenter networks are highly distinct from the vast majority of the Internet, or wide area networks (WANs), the results and observations presented in this paper may potentially apply to WANs as well.

As this paper relies heavily on experiments done through network simulations, this project cannot guarantee the realization of actual or realistic datacenter network traffic, but tests will be made to mimic certain datacenter network conditions such as worst-case scenarios.

IV. Methodology

Taking into consideration the penalties incurred on short flows from MPTCP connection establishment complexity, as

Technical Specifications	Value
Processor	Intel(R) Core i7-3612QE
Processor Speed	2.10 GHz
Number of Cores	4
RAM	16 GB
Operating System	`Ubuntu 16.04.3 LTS`
Linux Kernel	`Linux version 4.9.60.mptcp`

Table I
TESTBED TECHNICAL SPECIFICATIONS.

well as the penalties incurred on long flows due to the network hotspots that arise as a consequence of ECMP routing, we chose to measure flow completion time, and end-host throughput and goodput, respectively. Mean network utilization was also measured to gauge the effectivity of each protocol in terms of how well the available network routes were used.

To properly attribute each change in the above metrics, several possible permutations of the routing behavior, transport protocol, flow types (or payload sizes) end-host pairs, and network conditions were tested. These metrics were extracted and compared against each other considering the varying parameters between each test.

A. Building the Test Environment

Mininet was chosen to orchestrate the virtual network, its switches, and its hosts. The network topology is generated programmatically, and switch behavior can be swapped through configuration files. It also ensures that all virtual hosts are running on the same configuration as the server it is running in. This ease of use allows the researchers to implement multiple changes at once.

MPTCP is available as an ns-3 model [37] or a kernel patch [38]. Juggler [36], which modifies the GRO function to fix packet ordering, is only readily available as a kernel patch. Since the experiments require hosts that require both features to be present, it was decided that the kernel patches were to be used. The MPTCP and Juggler custom kernels were merged together and applied to the server to virtualize certain hosts.

Switches, on the other hand, can be described with its normal forwarding algorithm, which stores only one next-hop for all destinations in its routing table. However, the switches must also be capable of Equal Cost Multi-Path (ECMP), and Packet Spraying (PS), which require a routing table storing multiple next-hops. Since these forwarding behavior require inspection into packet headers and metadata, we utilized and modified software switches enabled by P4, a programming language used to process packets [39].

1) Testbed Specifications: Experiments and tests were ran on two SuperMicro bare metal servers with the specifications listed below. One server will be patched with the MPTCP custom kernel only, and the other with the merged MPTCP and Juggler kernel.

2) Setting up the Network Topology: For this experiment, the fat tree topology was chosen, a common and scalable data center network topology. Like all DCN topologies, the fat tree topology provides several equal cost paths between any two end-hosts. Moreover, because of the symmetry of the topology even with scaling, a Mininet topology can be easily set up with unique addressing [8]. Mininet was used for the simulation of the nodes, and Python was used to construct the topology. A $K = 4$ fat tree topology can be seen in 1, complete with the conventions used.

3) Switch Behavior: The P4 `behavioral-model` repository [40] was forked to get its target executables (e.g., `simple_router`) and modified the source to extend their functionalities. Each behavior is defined by tables and actions coded in C++, and compiled to a JSON using the `p4c-bm` tool [41]. The JSON is fed to the target executable, and the tables are filled with entries during initialization.

B. Test Design

To review, the goals of the tests are to measure the network performance given varying configurations as previously discussed. In particular, the study will test mainly throughput and flow completion time, with goodput and mean network utilization as extra data points.

For the purposes of this section, a topology with $K = 4$ will be considered. This means that there are 16 hosts, 16 switches in the edge and aggregate layers, and 4 switches in the core (see Figure 1).

With this in mind, this allows for 4 unique paths between two hosts.

$$Given\ k = 4,\ paths_{max} = \left(\frac{K}{2}\right)^2 = \left(\frac{4}{2}\right)^2 = 4 = n \quad (1)$$

Considering that each host uses only one interface to communicate to the entire network, preference was given to using `ndiffports` (with $n = 4$) as MPTCP's path manager in this experiment. This can be done through the `sysctl` feature. The transport protocol can be changed through `net.mptcp.mptcp_enabled`. Furthermore, we can control the MPTCP path manager using `net.mptcp.mptcp_path_manager`. In this case, we set the path manager of MPTCP-enabled hosts to `ndiffports`.

1) Network Traffic Conditions: Tests were done in two simulated network conditions. The first condition assumed a network without any activity, allowing for two hosts to communicate using all of the available resources of the network (hereinafter referred to as a *silent network*). The second condition assumed some form of simulated traffic, comparable to the performance in a live network (hereinafter referred to as a *noisy network*).

These were done to observe the changes in the design metrics in different network conditions. Note that the simulation of traffic was an approximation and may not be representative of the actual datacenter network traffic.

In the *silent network*, host pairs took turns at sending data to each other, without any other activity in the network. Pairs were chosen by random, ensuring that all hosts became a server or client at some point. The clients requested a payload of certain size from the server, one after another. This request was then repeated a certain number of times. The designations

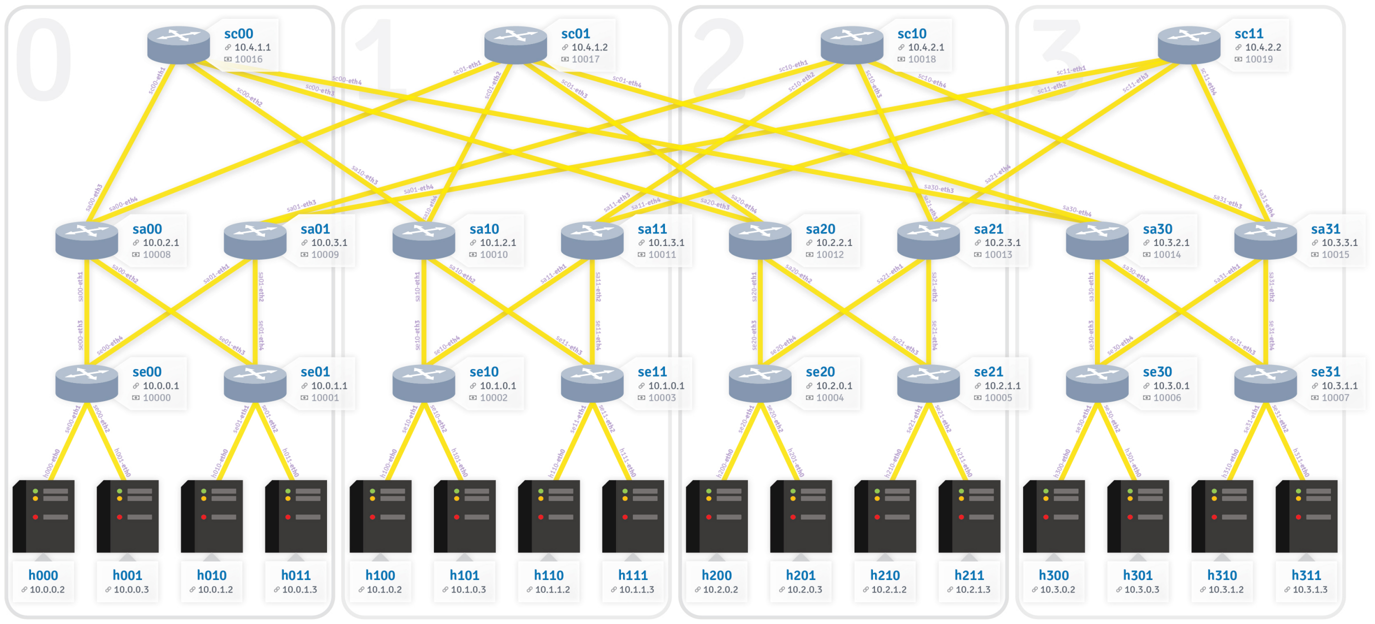


Figure 1. Fat tree (`k = 4`) topology in Mininet, including naming and addressing conventions.

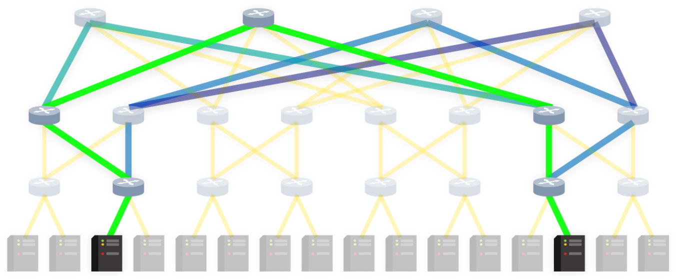


Figure 2. Two hosts communicating in a *silent network* setup. Note that there are no other network activity, and that there are four paths between the two hosts (indicated by the green ang bluish lines).

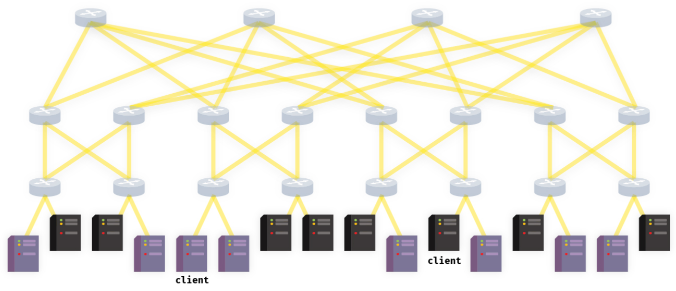


Figure 3. Two host groups in a *noisy network* setup, colored in black and purple. Note that one host in each group is designated as the client, to communicate with the rest of the group.

were fixed for each iteration of the test. This was done through a network bandwidth measurement tool named *iperf*.

In the *noisy network*, two hosts requested data from multiple hosts at the same time, polluting the network with activity. The network was split between two groups with one client each, and the rest acting as servers. Considering again a fat tree topology with $K = 4$, one client was chosen at random at the start of the test and seven were chosen to be file servers. Like the *silent network*, designations were fixed for each iteration of the test. This was done through an *HTTP* server in Python, and a tool to transfer data called *curl*.

In both network conditions, multiple server-client pairs were used to obtain a thorough assessment of the different links in the network. Analysis was done through inspection of the packet capture output of Mininet. Each test is also repeated to obtain a sufficient number of data points.

2) Test Parameters: The testbed can have varying permutations of network and host configurations. To aid in this, the test can be run using different testing parameters namely, host reorder resiliency, switch behavior, transport protocol, and payload sizes.

Datacenters cater to requests of varying sizes. To have an approximation of these requests and how they affect the performance of the changes to the network stack, we included the option to test with different payload sizes. Three flow types are to be used and tested for these experiments: `query`, `short`, and `long` flows [4].

Note that for the silent network, a query flow is defined with a flow size of 128KB as the tool used did not allow for the payload size to go below 128KB.

As a baseline comparison for the improvements of MPTCP, a TCP option was included in the testing environment. These results served as a form of control for the performance of MPTCP as a protocol. In addition, the performance of the

Flow	Silent Network	Noisy Network
query	128 kiB	10 kiB
short	500 kiB	500 kiB
long	25 MiB	25 MiB

Table II
FLOW DEFINITIONS, AND THEIR CORRESPONDING SIZES ON BOTH SIMULATED NETWORK TRAFFIC CONDITIONS.

changes in the network stack with TCP as its transport protocol was also measured and analyzed.

3) Performance Metrics: To review, throughput, goodput, flow completion time, and mean network utilization between end-hosts were measured to assess the effects of reorder resiliency in end-hosts and packet spraying in routers.

a) Throughput and Goodput: We define throughput to be the rate of the bytes transferred between two hosts in the network (i.e., total bytes transferred / flow completion time). In contrast, goodput is the rate of the actual data (only the size of the payload) transferred over a period of time (i.e., payload size in bytes / flow completion time).

A comparison between goodput and throughput can be used to indicate how much overhead affects the rate of transmission of actual data. If throughput is higher than goodput, this can be an indication to the presence of added MPTCP headers or packet retransmissions. This translates to no perceptible improvement from the perspective of the hosts.

b) Flow Completion Time: We define flow completion time to be the amount of time that it takes for a request between two hosts to complete. This includes the first request packet sent from the client to the server and the last `FIN` packet sent from server to the client.

An increase in flow completion time can be attributed to the overhead of connection establishment (for MPTCP), packet retransmissions, and/or `ACK` timeouts.

c) Mean Network Utilization: For this paper, we consider mean network utilization to be the ratio of used links between two hosts. For a fat tree network topology with $K = 4$, there are at most 4 paths that connect two hosts. Ideally, mean network utilization is maximized if all of paths have the near-equal (if not equal) minimum amount of bytes transferred in them (i.e., all paths have a balanced load with regards to the flow).

4) Statistical Tests: There are two independent variables: the switch behavior combined with the underlying transport protocol (`ecmp-mptcp`, `ps-mptcp`), as well as the inclusion/non-inclusion of Juggler (`juggler`, `vanilla` respectively). There are six test clusters, grouped by network traffic conditions (*silent network*, *noisy network*), permuted with the three payload sizes (`query`, `short`, `long`).

There are four dependent variables pertaining to the performance metrics discussed previously: throughput, goodput, flow completion time, and mean network utilization.

Each test cluster was subjected to a two-way MANOVA to verify the significance of the differences, and ranked via comparison of means. In addition, a comparison of percent differences of goodput and throughput was executed to inspect the effects of hotspots in the network.

Another experiment was also executed that includes TCP alongside MPTCP (the addition of `static-tcp` and `ps-tcp`), to serve as control. For the majority of the next section, we will be discussing the experiment concerning MPTCP tests only.

V. RESULTS AND ANALYSIS

A proper multivariate analysis of variance (MANOVA) shall be executed by meeting the following assumptions [42]: dependent variables should be continuous, independent variables should consist of two or more categorical groups, there should be independence between observations, and there should be a sufficient number of data points. These four assumptions were met by virtue of the experimental setup.

Other assumptions for MANOVA, if unmet, reduce the validity of the results. Some tests show that our data set does not meet some of these assumptions, which means that the following results should be interpreted with reservations.

A. Tests of Between-Subjects Effects and Best Performing Independent Variables

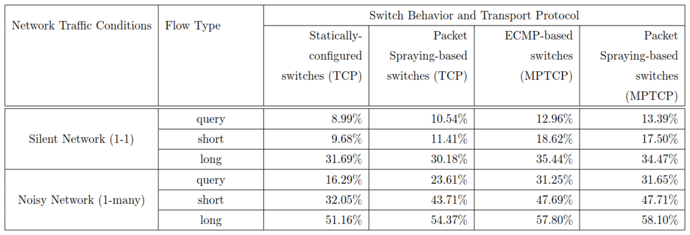
We chose a confidence interval of 95% ($\alpha = 0.05$) to determine the significance of the various test groups. For this section, we will be focusing solely on the tests of between-subjects effects.

For a given link in the network, bandwidth is limited to 20 Mbps. Tests within a silent network, specifically for throughput, goodput, and flow completion time are inconclusive as they can maximize the bandwidth regardless of switch behavior and transport protocol. The throughput and goodput for a given flow naturally reach the bandwidth cap in a silent network as there are no other flows competing for resources.

B. Gap Between Goodput and Throughput as an Indicator of Network Hotspots

Network hotspots, defined as switches that have links that are near to exceed their capacity, result in an increase of dropped packets and increased packet retransmissions. This results in an increase in flow completion time, as well as the total amount of bytes transferred. Therefore, a significant gap between throughput and goodput may be noticed if network hotspots are in the network. In this experiment, we took the observed means of goodput and throughput for statically-configured switches (with hosts running TCP), ECMP-based switches (with hosts running MPTCP), and PS-based switches (with hosts running TCP and hosts running MPTCP) and calculated the percentage of traffic in excess of the actual flow size.

If we assume that the percentage of excess traffic is caused by retransmission, then we can see that statically-configured switches has significantly less retransmissions over both ECMP-based switches and PS-based switches for query and short flows. However, during long flows, the difference between all the switch behaviors seem insignificant. In all



Network Traffic Conditions	Flow Type	Switch Behavior and Transport Protocol			
		Statically-configured switches (TCP)	Packet Spraying-based switches (TCP)	ECMP-based switches (MPTCP)	Packet Spraying-based switches (MPTCP)
Silent Network (1-1)	query	8.99%	10.54%	12.96%	13.39%
	short	9.68%	11.41%	18.62%	17.50%
	long	31.69%	30.18%	35.44%	34.47%
Noisy Network (1-many)	query	16.29%	23.61%	31.25%	31.65%
	short	32.05%	43.71%	47.69%	47.71%
	long	51.16%	54.37%	57.80%	58.10%

Figure 4. Percentage of traffic in excess of "good" traffic (i.e., actual payload size).

cases, ECMP-based switches and PS-based switches (with hosts running MPTCP) performed comparably equally, implying that packet spraying did not mitigate the hotspot problem.

C. Flow Completion Time Performed Worse with Reorder-Resilient Hosts; Inconclusive Effects on Throughput

Since Juggler, the tool to enable reorder-resiliency in the hosts, was created to increase tolerance between TCP packet arrival times, it might not perform as well with MPTCP. Based on our results, Juggler had a negative effect on flow completion time for a silent network with query and short flows, as well as for a noisy network for short flows.

This can also explain the inconclusive results for throughput. Results are mixed, with Juggler increasing throughput for short flows (on both silent and noisy networks), but failing to increase throughput for query flows on a silent network and long flows on a noisy network.

This leads us to believe that Juggler cannot be simply placed in a network without negatively affecting its performance.

D. Packet Spraying Generally Performed Better than ECMP

As for the switch behavior, packet spraying increased mean network utilization for all test groups compared to ECMP. In addition, packet spraying performed better than ECMP for goodput and throughput tests in a noisy environment.

This behavior may be preferred as most datacenter networks are high in network activity.

E. Effects of Combining Packet Spraying-based Switches with Reorder-Resilient Hosts

A combination of packet spraying and reorder resiliency improves throughput for both silent and noisy networks. However flow completion time increases (due to the presence of Juggler as seen in the Juggler-only analysis).

Goodput results turn out to be inconclusive as tests performed better with PS-based switches, but some preferred the inclusion of Juggler while others don't. Mean network utilization remains significantly higher when compared to the rest which can be attributed to the packet spraying behavior.

VI. Conclusion and Recommendations

Based on preliminary tests, we found that MPTCP increases throughput significantly compared to TCP, especially with using multiple paths. However the connection establishment overhead needed to create multiple subflows penalizes the speed at which flows are created and consequently completed.

In addition, the paper promised to experimentally prove that ECMP-based switches causes hotspots in the network. By comparing the throughput and goodput of ECMP-based switches without reorder resiliency, we saw that there are was no improvements to the hotspots in the network. However hotspots in the network may be a consequence of the experiment setup. This could either be attributed to the amount of traffic in the noisy network or due to the limited bandwidth at the single interface of the host for the silent network.

It was also found that packet spraying improves throughput, goodput, and mean network utilization for a noisy environment. Considering that most datacenters will, more often than not, operate with a lot of concurrent flows in the network, packet spraying may be beneficial.

Packet spraying and reorder resiliency show a significant difference between groups (except for a noisy network with long and query-length flows and silent network with short flows). Looking at a noisy network with short flows, throughput is improved and flow completion time worsens.

From this, we can say that packet spraying is better in terms of throughput, goodput, and mean network utilization. Flow completion time for packet spraying turned out inconclusive data.

A. Recommendations and Future Work

Further analysis may be done on TCP combined with packet spraying and reorder resilient hosts. According to initial testing, TCP with reorder resilient hosts and packet spraying showed an increase in throughput and goodput. This may be promising as there would be no need to implement a new transport protocol.

Network hotspots may also be studied further by creating a special testbed specifically built to measure the amount of traffic going through specific nodes in the network. We hypothesize that packet spraying may still improve network hotspots caused by ECMP.

Data collected in an datacenter network with real traffic would be beneficial for the validity of the tests since the simulated traffic in this experiment is merely an approximation.

References

[1] L. A. Barroso, J. Clidaras, and U. Hölzle, "The datacenter as a computer: An introduction to the design of warehouse-scale machines," *Synthesis lectures on computer architecture*, vol. 8, no. 3, pp. 1–154, 2013.

[2] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowtron, "Better never than late: Meeting deadlines in datacenter networks," in *ACM SIGCOMM Computer Communication Review*, vol. 41, pp. 50–61, ACM, 2011.

[3] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: amazon's highly available key-value store," *ACM SIGOPS operating systems review*, vol. 41, no. 6, pp. 205–220, 2007.

[4] 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*, vol. 40, pp. 63–74, ACM, 2010.

[5] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel, "The cost of a cloud: research problems in data center networks," *ACM SIGCOMM computer communication review*, vol. 39, no. 1, pp. 68–73, 2008.

[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*, pp. 267–280, ACM, 2010.

[7] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, “Hedera: Dynamic flow scheduling for data center networks.,” in *NSDI*, vol. 10, pp. 19–19, 2010.

[8] M. Al-Fares, A. Loukissas, and A. Vahdat, “A scalable, commodity data center network architecture,” in *ACM SIGCOMM Computer Communication Review*, vol. 38, pp. 63–74, ACM, 2008.

[9] R. Niranjan Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat, “Portland: a scalable fault-tolerant layer 2 data center network fabric,” in *ACM SIGCOMM Computer Communication Review*, vol. 39, pp. 39–50, ACM, 2009.

[10] J. He and J. Rexford, “Toward internet-wide multipath routing,” *IEEE network*, vol. 22, no. 2, 2008.

[11] C. E. Leiserson, “Fat-trees: universal networks for hardware-efficient supercomputing,” *IEEE transactions on Computers*, vol. 100, no. 10, pp. 892–901, 1985.

[12] C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, and S. Lu, “Dcell: a scalable and fault-tolerant network structure for data centers,” in *ACM SIGCOMM Computer Communication Review*, vol. 38, pp. 75–86, ACM, 2008.

[13] 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,” *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 4, pp. 63–74, 2009.

[14] B. Lebiednik, A. Mangal, and N. Tiwari, “A survey and evaluation of data center network topologies,” *arXiv preprint arXiv:1605.01701*, 2016.

[15] S. Rost and H. Balakrishnan, “Rate-aware splitting of aggregate traffic,” tech. rep., Technical report, MIT, 2003.

[16] 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*, vol. 41, pp. 266–277, ACM, 2011.

[17] J. F. Kurose, *Computer networking: A top-down approach featuring the internet, 3/E.* Pearson Education India, 2005.

[18] A. Ford, C. Raiciu, M. Handley, S. Barre, and J. Iyengar, “Architectural guidelines for multipath tcp development,” tech. rep., 2011.

[19] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure, “Tcp extensions for multipath operation with multiple addresses,” tech. rep., 2013.

[20] A. Ford, “Multipath tcp architecture: Towards consensus.”

[21] W. R. Stevens, “Tcp slow start, congestion avoidance, fast retransmit, and fast recovery algorithms,” 1997.

[22] 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*, vol. 39, pp. 51–62, ACM, 2009.

[23] R. Barik, M. Welzl, S. Ferlin, and O. Alay, “Lisa: A linked slow-start algorithm for mptcp,” in *Communications (ICC), 2016 IEEE International Conference on*, pp. 1–7, IEEE, 2016.

[24] M. Kheirkhah, I. Wakeman, and G. Parisis, “Mmptcp: A multipath transport protocol for data centers,” in *Computer Communications, IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on*, pp. 1–9, IEEE, 2016.

[25] C. E. Hopps and D. Thaler, “Multipath issues in unicast and multicast next-hop selection,” 2000.

[26] A. Dixit, P. Prakash, Y. C. Hu, and R. R. Kompella, “On the impact of packet spraying in data center networks,” in *INFOCOM, 2013 Proceedings IEEE*, pp. 2130–2138, IEEE, 2013.

[27] J. Postel *et al.*, “Transmission control protocol rfc 793,” 1981.

[28] H. Zhang, J. Zhang, W. Bai, K. Chen, and M. Chowdhury, “Resilient datacenter load balancing in the wild,” in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pp. 253–266, ACM, 2017.

[29] B. A. A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, and T. Turletti, “A survey of software-defined networking: Past, present, and future of programmable networks,” *IEEE Communications Surveys & Tutorials*, vol. 16, no. 3, pp. 1617–1634, 2014.

[30] H. Kim and N. Feamster, “Improving network management with software defined networking,” *IEEE Communications Magazine*, vol. 51, no. 2, pp. 114–119, 2013.

[31] S. H. Yeganeh, A. Tootoonchian, and Y. Ganjali, “On scalability of software-defined networking,” *IEEE Communications Magazine*, vol. 51, no. 2, pp. 136–141, 2013.

[32] A. Hussein, I. H. Elhajj, A. Chehab, and A. Kayssi, “Sdn for mptcp: An enhanced architecture for large data transfers in datacenters,” in *Communications (ICC), 2017 IEEE International Conference on*, pp. 1–7, IEEE, 2017.

[33] S. Zannettou, M. Sirivianos, and F. Papadopoulos, “Exploiting path diversity in datacenters using mptcp-aware sdn,” in *Computers and Communication (ISCC), 2016 IEEE Symposium on*, pp. 539–546, IEEE, 2016.

[34] S. Ghorbani, Z. Yang, P. Godfrey, Y. Ganjali, and A. Firoozshahian, “Drill: Micro load balancing for low-latency data center networks,” in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pp. 225–238, ACM, 2017.

[35] M. Zhang, B. Karp, S. Floyd, and L. Peterson, “Rr-tcp: a reordering-robust tcp with dsack,” in *Network Protocols, 2003. Proceedings. 11th IEEE International Conference on*, pp. 95–106, IEEE, 2003.

[36] Y. Geng, V. Jeyakumar, A. Kabbani, and M. Alizadeh, “Juggler: a practical reordering resilient network stack for datacenters,” in *Proceedings of the Eleventh European Conference on Computer Systems*, p. 20, ACM, 2016.

[37] M. Kheirkhah, “Multipath tcp (mptcp) implementation in ns-3.” Github, May 2014.

[38] U. catholique de Louvain, “Linux kernel implementation of multipath (mptcp).” Github, Nov. 2017.

[39] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, *et al.*, “P4: Programming protocol-independent packet processors,” *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87–95, 2014.

[40] p4lang, “behavioral-model: Rewrite of the behavioral model as a C++ project without auto-generated code (except for the PD interface),” Mar. 2018. original-date: 2015-01-26T21:43:23Z.

[41] p4lang, “p4c-bm: Generates the JSON configuration for the behavioral-model (bmv2), as well as the C/C++ PD code,” Mar. 2018. original-date: 2015-08-07T15:48:22Z.

[42] “How to perform a two-way MANOVA in SPSS Statistics | Laerd Statistics.”