Improving Data Center Network Performance through Path Diversity

Charles Joshua M. Alba, Kyle Dominic A. Gomez, Rene Josiah M. Quinto
Electrical and Electronics Engineering Institute
University of the Philippines - Diliman
Quezon City, Philippines
Email: charles.alba@eee.upd.edu.ph, kyle.gomez@eee.upd.edu.ph, 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. This requires servers within the datacenter to communicate efficiently and rapidly. Services, differentiated by their workload profiles, require a network that can guarantee high throughput, and low flow completion time. To achieve these, this paper focused on the effects of 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 were employed, providing increased reordering resiliency, to make up for MPTCP's undesired behavior within datacenters such as network link hotspots. With this, the researchers hypothesized an increase in throughput and decrease in the flow completion time. To test this, a simulated datacenter environment was created tweaking flow types, network traffic conditions, switch behavior, transport protocol, and host-based reordering resiliency. Link throughput and goodput, as well as flow completion time and mean network utilization was then measured. Analyzing the data, the study found that packet spraying turned out to be generally beneficial for the network in terms of all metrics (throughput, goodput, flow completion time, and mean network utilization). The change in the GRO layer, on the other hand, provided mixed results. By simply comparing the means, a simulated network with traffic designed to be close to an actual data center generally performed better in all metrics with switches that perform packet spraying, as well as MPTCP-enabled servers with the GRO layer modification.**

I. Introduction

Common topologies in datacenter networks exhibit symmetry and path diversity [1]. This ensures that multiple paths of equal costs exist between two end-hosts [2]. In addition, an ideal datacenter network must guarantee high throughput for large flows, and low latency for short flows [3], [1].

Multipath TCP (MPTCP), an experimental protocol by the IETF, is a TCP extension that uses multiple paths over many connections [4]. By using multiple paths simultaneously, MPTCP aims to increase network utilization, specifically throughput, and increase "resilience of connectivity" [5]. This is done by employing Equal Cost Multi-Path (ECMP) [6], 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 [2].

II. Review of Related Literature

Random packet spraying, an alternative to ECMP, can be used to resolve hotspots as it allows for a greater granularity for load balancing [1]. However, this can result in packets that come out-of-order at the receiver, triggering false retransmissions upon receipt of three duplicated packet acknowledgements [8], in addition to drastically reducing sender throughput [9]. 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 [9].

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 [10] 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 [2] and slow start [11], 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 [12]. However, this means that it inherits problems from both implementations altogether.

Better network utilization can be also achieved using Software-Defined Networking [13], [14] 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 flows, but due to the added controller overhead, it may penalize short flows. In addition, it may also have some scalability issues [14].

To combat delay and lower throughput caused by multiple timeouts on wireless networks, retransmission redundancy was implemented using full and partial bicasting over MPTCP [15].

This may not necessarily be effective in the datacenter setup as redundant packets may cause more false retransmissions.

III. Problem Statement and Objectives

An increase in datacenter network performance (higher throughput, lower flow completion time) is ideal not only for the company but for its users as well. Employing MPTCP in a network alone will bring a comparable difference compared to TCP (higher throughput due to its use of multiple paths), however it comes with its drawbacks (longer flow completion time due to connection establishment complexity; appearance of network hotspots).

The researchers hypothesized 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, the researchers hypothesized that this can be mitigated by creating reorder-resilient end-hosts. By comparing the results we see from different experiments, the basis for considering a reorder-resilient network for future datacenter networks can be strengthened, as well as have the results potentially contribute to the growth of MPTCP as an experimental protocol.

A. Objectives

The objectives of this work were as follows: First was to experimentally prove that MPTCP benefits large flows, but penalizes short flows. Next was to 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.

B. Scope and Limitation

The project focused on datacenter networks, and assumed ideal working conditions. More specifically, this project did 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 were 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 relied heavily on experiments done through network simulations, this project cannot guarantee the realization of actual or realistic datacenter network traffic, but tests were made to mimic certain datacenter network conditions such as worst-case scenarios.

IV. Methodology

A. Validating the Behavior of MPTCP Path Managers

Preliminary work was executed to investigate the performance between three out of four MPTCP path managers (default MPTCP, `fullmesh` MPTCP, and `ndiffports` MPTCP) compared to the performance of TCP. Based on the results, TCP had the fastest flow completion time out of the four, being beneficial to short flows. `Fullmesh` MPTCP achieved the highest throughput overall, being able to take advantage of multiple paths, making it ideal for large flows. However, it had the slowest flow completion time, which may be attributed to extra packets being sent.

B. Building the Test Environment

Mininet was chosen to orchestrate the virtual network, its switches, and its hosts. It ensured that all virtual hosts are running on the same configuration as the server it was running in. This allowed the researchers to implement multiple changes at once. The network topology was generated programmatically, and switch behavior was swapped through configuration files.

MPTCP was available as an ns-3 model [16] or a kernel patch [17]. Juggler [10], which modifies the GRO function to increase packet reordering resiliency, was only readily available as a kernel patch. Since the experiments required hosts that needed both features, a custom kernel was prepared with the MPTCP kernel and Juggler kernel merged together. This was then applied to the server to virtualize certain hosts.

Switches, on the other hand, were described with its normal forwarding algorithm, which stores only one next-hop for all destinations in its routing table. The switches were also capable of Equal Cost Multi-Path (ECMP), and Packet Spraying (PS), which required 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 [18].

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

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 (MPTCP only)	4.9.60.mptcp
Linux kernel (MPTCP + Juggler)	4.14.24

TABLE I
TECHNICAL SPECIFICATIONS OF THE TESTBED.

2) Setting up the Network Topology: For this study, the fat tree topology was chosen because it was 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 the topology maintains its symmetry regardless of its size, a Mininet topology can be easily set up with unique addressing [19]. Mininet was used for the simulation of the end-hosts and switches, and Python was used to construct the topology. A $K = 4$ fat tree topology can be seen in Figure 1, complete with the conventions used.

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

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

Considering that each host used 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.

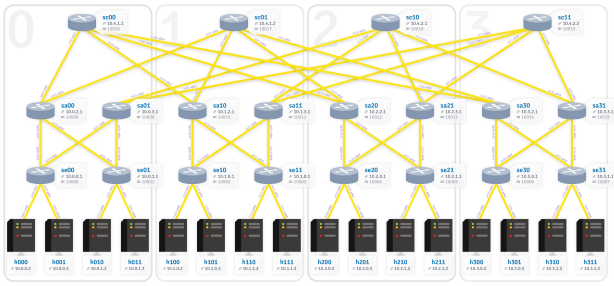

Fig. 1. Fat tree (k = 4) topology in Mininet, including naming and addressing conventions. Pods were the large groups of end-hosts and switches numbered 0 to 3.

C. Test Design

1) Network Traffic Conditions: Tests were done in two simulated network conditions/setups. 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`).

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 (this defined the flow type), one after another. This request was then repeated a certain number of times. The designations were fixed for each iteration of the test. The payload transfer was done through a network bandwidth measurement tool named `iperf`.

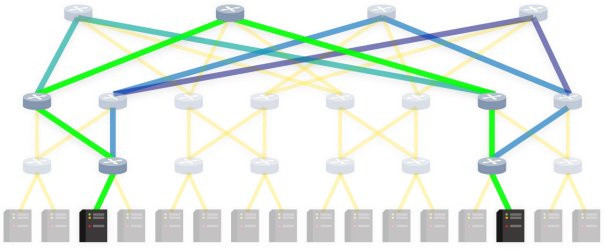

Fig. 2. Two hosts communicating in a `silent network` setup. Note that there are no other network activity, and that there are four paths between the two hosts (indicated by the green and bluish lines).

In the `noisy network`, two hosts requested data from multiple hosts at the same time, polluting the network with activity. For a fat tree topology with $K = 4$, this meant that the network was split between two groups, with each having one client chosen at random, and seven chosen to be servers. Like the `silent network`, designations were fixed for each iteration of the test. This behavior mimics the partition-aggregate design pattern of DCNs [3].The payload transfer was done through an `HTTP` server in Python, and a tool to transfer data called `curl`.

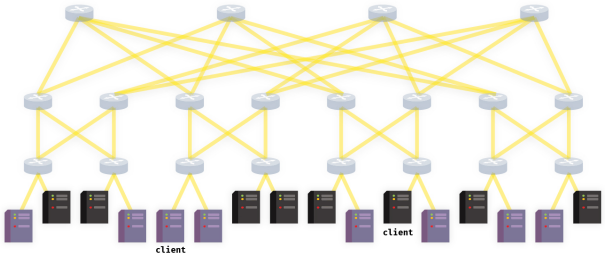

Fig. 3. Two host groups in a `noisy network` setup, colored in black and purple. Note that one host in each group is designated as the client, to communicate with the rest of the group.

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 was also repeated to obtain a sufficient number of data points.

2) Test Parameters: 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 flow types, done by defining payload sizes. Three flow types are to be used and tested for these experiments: `query`, `short`, and `long` (previously referred to as `large`) flows [3].

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

Flow	Examples	Silent Network	Noisy Network
query	Quick, bursty connections	128 KB	10 KB
short	Web page requests	500 KB	500 KB
long	Streaming, VoIP, file hosting and transfer	25 MB	25 MB

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

As a baseline comparison for the improvements of MPTCP, a TCP option was included in the testing environment. These results can serve as a control group for the comparison of the performance of MPTCP as a protocol. In addition, the performance of the changes in the network stack with TCP as its transport protocol was also measured and analyzed.

3) Performance Metrics: Packets were aggregated to flows (based on the 5-tuple) and were grouped together by links.

By tracing the links traversed by flows between two end-hosts, the following were discovered: the unique shortest paths used, the total amount of bytes transferred within the interface associated with the flow, and the time it took for the flow to complete. This allowed for the following performance metrics to be derived.

a) Throughput and Goodput: Throughput was to be the rate of the bytes transferred between two hosts in the network (i.e., total bytes transferred in all flows / flow completion time). In contrast, goodput was 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: Flow completion time was defined to be the amount of time that a request between two hosts took to complete. This included the first request packet sent from the client to the server and the last `FIN` packet sent from server to the client. For instances with multiple flows, the flow that took the longest time to complete was considered.

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, mean network utilization was considered to be the ratio of used links between two hosts. Knowing both used and unused unique shortest paths, mean network utilization was considered to be maximized if all interfaces with the smallest number of bytes transferred per path had about the same value. For example, if two end-hosts with four unique shortest paths between them used only one path out of four, the mean network utilization will be 25% (0.25).

4) Statistical Tests: All in all, the test clusters and independent variables was summarized in Table III.

			Test Clusters					
			Silent Network (1-1)			Noisy Network (1-many)		
Independent Variables	ecmp-mptcp	vanilla	query	short	long	query	short	long
		juggler	query	short	long	query	short	long
	ps-mptcp	vanilla	query	short	long	query	short	long
		juggler	query	short	long	query	short	long

TABLE III
TEST CLUSTERS AND INDEPENDENT VARIABLES USED IN THIS EXPERIMENT

There were 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 were six test clusters, grouped by network traffic conditions (`silent network`, `noisy network`), permuted with the three flow types/payload sizes (query, short, long).

There were 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 the performance of the independent variables were 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 which included TCP tests serving as control groups (the addition of `static-tcp` and `ps-tcp` to the existing switch behavior-transport protocol pairs) was also executed. For the majority of the next chapter however, the experiments discussed MPTCP tests only.

V. RESULTS AND ANALYSIS

A proper multivariate analysis of variance (MANOVA) meets the following assumptions [20]: 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 not listed above, if unmet, reduce the validity of the results. Some tests did show that the data set did not meet some of these assumptions, which meant that the following results should be interpreted with reservations.

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

A confidence interval of 95% ($\alpha = 0.05$) was chosen to determine the significance of the various test groups. For this chapter, focus is given solely on the tests of between-subjects effects. The independent variables with the highest mean for outcomes that had a significant result were then identified.

For a given link in the network, bandwidth was limited to 20 Mbps. Tests within a silent network, specifically for throughput, goodput, and flow completion time were mostly inconclusive. This may be attributed to the nature of the silent network–with all configurations able to maximize the network capacity, the results may turn out with little difference with each other. With the throughput and goodput for a given flow for all configurations naturally reaching the bandwidth cap, flow completion time may not differ greatly between setups.

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, 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) were taken. The percentages of traffic in excess of the actual flow size were then calculated.

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%

TABLE IV
PERCENTAGE OF TRAFFIC IN EXCESS OF "GOOD" TRAFFIC (I.E., ACTUAL PAYLOAD SIZE).

The results show that statically-configured switches had significantly less excess traffic 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 seemed insignificant. In all 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.

By assuming that excess traffic correlates to multiple retransmissions and the presence of hotspots, increasing the number of paths used may not always result in an increase in goodput especially if these paths shared common links (i.e., hosts only had one interface connected to the entire network, resulting in clogs).

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

Based on the results, Juggler had a negative effect on flow completion time as well as goodput for a silent network with query and short flows, as well as for a noisy network for short flows. The increase in false retransmissions may be attributed to Juggler where at times, it may have incorrectly bundled MPTCP packets as if they were TCP packets, as MPTCP was built on top of TCP [4], with its own separate system of determining the order of packets. This may also explain the inconclusive results for throughput. Results were 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.

D. Packet Spraying Generally Performed Better than ECMP

As for the switch behavior, packet spraying performed better than ECMP in all significant cases. It increased mean network utilization for all test groups compared to ECMP, and it performed mostly better than ECMP for goodput, throughput, and flow completion time 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

Studying the combined interaction of both switch behavior and reorder-resiliency led to less significant results. Packet spraying without the Juggler modification increased goodput for query flows on a silent network (1.65% greater than ECMP without the Juggler modification), and for short flows on a noisy network (1.38% greater than packet spraying with the Juggler modification). Interestingly, ECMP with the Juggler modification increased goodput for long flows on a silent network (1.01% increase versus packet spraying with the Juggler modification). Packet spraying with the Juggler modification, on the other hand, increased throughput for long flows on a silent network (4.27% increase versus ECMP without the Juggler modification), and short flows on a noisy network (21.84% increase versus ECMP with the Juggler modification).

Packet spraying without the Juggler modification decreased the flow completion time for query flows on a silent network (1.70% decrease versus ECMP without the Juggler modification), and short flows on a noisy network (1.86% decrease versus ECMP without the Juggler modification). In addition, it also increased the mean network utilization for long flows on a silent network (0.15% increase versus packet spraying with the Juggler modification).

By visually comparing the results, ignoring all measures of statistical significance, it can be seen that that a noisy network with packet spraying and the Juggler modification provided a more desired performance in all four metrics (througput, goodput, flow completion time, and mean network utilization). Packet spraying and the Juggler modification also improved the mean network utilization in all cases. On the other hand, if flow completion time were to be prioritized (where the majority of the flows can be classified as query flows or short flows), the network may be better off without the Juggler modification.

VI. CONCLUSION AND RECOMMENDATIONS

Based on preliminary tests, MPTCP increased throughput significantly compared to TCP, especially whenusing multiple paths. However, the connection establishment overhead needed to create multiple subflows penalized 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. As a basis for comparison, two network traffic conditions designed to mimic both the best case (silent network) and worst case (noisy network) scenarios were used. By comparing the throughput and goodput of ECMP-based switches without reordering resiliency, it can be seen that that there were 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 both network setups.

It was also found that packet spraying generally improved all four metrics (throughput, goodput, flow completion time,

and mean network utilization) for a noisy environment. However, inspecting the effects of the GRO layer modification (Juggler) had mixed results. Comparisons including the combined effects of both the switch behavior and Juggler turned out with less statistically significant results, however when this is ignored, packet spraying and Juggler-modified hosts provided a more desired performance in all metrics (goodput, throughput, flow completion time, and mean network utilization) for noisy networks. Considering that most datacenters will, more often than not, operate with a lot of concurrent flows in the network, packet spraying (tied with the GRO layer modification) may be beneficial.

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 in the network.

As for MPTCP, the other path managers can be used and tweaked to gain more insights as to how the total number of subflows would affect the performance of the different flow types. Specifically, an investigation can be done analyzing if incrementing/reducing subflows would affect network performance. The addition of a flow type classifier as well as an automatic knob on the number of subflows can potentially help grow MPTCP as a protocol.

Further tests can also be done studying network topologies other than fat tree, as used in this study. Fat tree is commonly preferred due to its cheap and scalable nature, but the fact that each end-host is connected to the rest of the network through only one interface causes a significant bottleneck, limiting the benefits of the network's supposed path diversity. It is also possible that different results may arise from having a larger fat tree ($K > 4$).

Juggler, the modification to the Generic Receive Offload (GRO) layer, can also be investigated further, specifically with its effects on both TCP and MPTCP. Since Juggler has not been maintained for quite a while now, its kernel modifications may have been completely overwritten, skipped, or negated by updated kernel patches.

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. In some cases (like in a different network topology), packet spraying may still improve network hotspots caused by ECMP.

Lastly, data collected in a datacenter network with real traffic would be beneficial for the validity of the tests since the simulated traffic in this experiment was merely an approximation.

REFERENCES

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

[2] 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*. IEEE, 2013, pp. 2130–2138.

[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*, vol. 40, no. 4. ACM, 2010, pp. 63–74.

[4] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure, "Tcp extensions for multipath operation with multiple addresses," Tech. Rep., 2013.

[5] A. Ford, "Multipath tcp architecture: Towards consensus." [Online]. Available: https://www.ietf.org/proceedings/77/slides/imptcp-2.pdf

[6] 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, no. 4. ACM, 2011, pp. 266–277.

[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, 2010, pp. 19–19.

[8] A. Ford, C. Raiciu, M. Handley, S. Barre, and J. Iyengar, "Architectural guidelines for multipath tcp development," Tech. Rep., 2011.

[9] 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*. IEEE, 2003, pp. 95–106.

[10] 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*. ACM, 2016, p. 20.

[11] W. R. Stevens, "Tcp slow start, congestion avoidance, fast retransmit, and fast recovery algorithms," 1997.

[12] 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*. IEEE, 2016, pp. 1–9.

[13] 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*. IEEE, 2017, pp. 1–7.

[14] 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*. IEEE, 2016, pp. 539–546.

[15] M. Fukuyama, N. Yamai, and N. Kitagawa, "Throughput improvement of mptcp communication by bicasting on multihomed network," in *Student Project Conference (ICT-ISPC), 2016 Fifth ICT International*. IEEE, 2016, pp. 9–12.

[16] M. Kheirkhah, "Multipath tcp (mptcp) implementation in ns-3," Github, May 2014. [Online]. Available: https://github.com/mkheirkhah/mptcp

[17] U. catholique de Louvain, "Linux kernel implementation of multipath (mptcp)," Github, Nov. 2017. [Online]. Available: https://github.com/multipath-tcp/mptcp

[18] 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.

[19] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 4. ACM, 2008, pp. 63–74.

[20] "How to perform a two-way MANOVA in SPSS Statistics | Laerd Statistics." [Online]. Available: https://statistics.laerd.com/spss-tutorials/two-way-manova-using-spss-statistics.php