Improving Data Center Network Performance through Path Diversity

Undergraduate Project Proposal

by

Charles Joshua Alba
2013-06878
B.S. Computer Engineering

Kyle Gomez
2013-25650
B.S. Computer Engineering

Rene Josiah M. Quinto
2013-14854
B.S. Computer Engineering

Adviser:

Professor Roel Ocampo
Professor Isabel Montes

University of the Philippines, Diliman
November 2017

Abstract

Improving Data Center Network Performance through Path Diversity

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 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. With this, we hypothesize an increase in throughput and decrease in the flow completion time. We will test this through a comprehensive analysis of different testbeds with varying parameters. Initial tests in a smaller network topology show that throughput between two end-hosts in a network increases which also implies that network utilization also increases. It was also observed that the overhead of establishing multiple subflows in an MPTCP connection using the TCP handshake penalizes short flows.

Contents

List of Figures

List of Tables

Chapter 1

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.

1.1 Servicing Increasing Demand with Datacenter Networks

Several companies and institutions (Google, Amazon, and Microsoft, to name a few) provide a wide variety of online services such as video streaming, online gaming, VoIP, file sharing, and simple web searching. These online services usually differ in their workload profiles [2, 3]—faster connections result to better performance for file sharing services, while connection stability is the main concern for video streaming services.

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 [4]. To keep up with demand that increases in both quantity and complexity, companies need to expand and upgrade their resources, increasing costs [5, 6]. 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) [7]. 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 [4].

Secondly, an organized and redundant network infrastructure eases management [4]. Finally, distributing requests to different servers increases server utilization, proving to be more cost-effective and scalable [8].

1.2 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 [9], the network topology must guarantee a speedy and reliable connection between local end-hosts. These properties can be characterized with sufficient interconnection bandwidth [10, 11] and fault-tolerance [12].

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 [7]. End-hosts also need to keep their internal data structures fresh [7] through frequent large transfers of data. Therefore, datacenters must also guarantee low latency for short flows, and high throughput to service large flows [7, 3].

Current datacenter network topologies guarantee the previous characteristics through different approaches. Switch-centric topologies [11, 13] 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 [14, 15] allow servers to forward packets as well, however software routing may be slow and can affect host-based applications [16].

Topologies can be also designed to be modular [15] or hierarchical [13] to increase network capacity through scalability [10]. This introduces multiple redundant links and paths between servers within the network, establishing a connection-rich nature [3, 17].

1.3 Better Network Utilization through Multiple Paths

Host-to-host connectivity within a datacenter was originally facilitated using the Transmission Control Protocol (TCP) [7, 18]. TCP ensures robustness in the delivery of packets between end-hosts in the presence of communication unreliability [19].

TCP connections make use of a single network path between end-hosts [20]. 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 [21], 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 [22, 23], which increases throughput and resiliency of the network [21].

1.4 Outline of the Document

The rest of the document is presented as follows. The next chapter will discuss different improvements suggested for MPTCP based on different metrics such as reliability, and a global view of the network. Additionally, some issues in unmodified MPTCP are also discussed and studied in depth. Chapter 3 discusses the problem statement, objectives, and scope and limitations of the project. Section 4 and 5 discuss the steps taken to create the project.

Chapter 2

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. The nature of datacenter network traffic and topology presents two potential setbacks for MPTCP. First, short packet streams (short flows) might experience a higher latency due to MPTCP's connection establishment mechanism. Second, large packet streams (large flows) might suffer from lower throughput due to MPTCP's congestion and subflow mechanism. In addition to the presented solutions to these problems, a closer look is given to switch-based packet spraying, combined with host-based reordering resiliency, as a potential improvement over MPTCP.

2.1 Connection Establishment of MPTCP Penalizes Short Flows

MPTCP, being designed as an extension for TCP [22], inherits its congestion control mechanisms. One of which is Slow Start [24], 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 [25] and are of a bursty nature [7], MPTCP would cause these flows to stay in the Slow Start phase [26], 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 [27].

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 [27], 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.

2.2 Decreased Throughput due to Redundant Retransmissions

MPTCP uses Equal-Cost Multipath (ECMP), a switch-based algorithm, to forward packets through the network [18]. ECMP bases its forwarding on a 5-tuple (source IP, destination IP, source port, destination port, protocol number) [28] 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. Full switch queues will then lead to dropped packets, which in turn, decrease mean network utilization [10].

Duplicating packets across multiple paths (bicasting/multicasting) in an MPTCP connection can be used to mitigate the decrease in throughput due to dropped packets [29]. In full bicasting, each connection transmits a copy of all packets sent across multiple subflows, while selective bicasting only duplicates packets that are sent as a retransmission of lost packets.

However, the increase in throughput for individual subflows brought about in both full and selective bicasting may not be optimal in a datacenter. This is, in part, attributed to the duplicating nature of bicasting wherein redundant data is sent through the network's links and consumes resources for other subflows. In addition to redundant data packets, control packets such as the ACK packets in TCP add to network traffic. This added traffic to the network can be perceived as a loss in throughput.

2.3 Diffusing Network Hotspots through MPTCP with Packet Spraying

As previously mentioned, network hotspots create areas of congestion in datacenter networks, throttling the throughput of the connections that pass through these hotspots. One way to avoid this is to use packet spraying [30]. 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 [30]. 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 [19]. 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 [27], 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 [31]. 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.

2.4 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 [32, 33, 34]. This enables network protocols to make use of global

network information, such as link utilization and end-to-end delays [35, 36], 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 [35].

These benefits, however, comes at the price of controller overhead [35]. 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) [37], 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 [35].

Chapter 3

Problem Statement and Objectives

3.1 Problem Statement

Common topologies in datacenter networks exhibit symmetry and path diversity [3]. This ensures that multiple paths of equal costs exist between two end-hosts [30]. In addition, an ideal datacenter network must guarantee high throughput for large flows, and low latency for short flows [7, 3].

Multipath TCP (MPTCP), an experimental protocol by the IETF, is a TCP extension that uses multiple paths over many connections [22]. By using multiple paths simultaneously, MPTCP aims to increase network utilization, specifically throughput, and increase "resilience of connectivity" [23]. This is done by employing Equal Cost Multi-Path (ECMP) [18], 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 [10], usually exceeding the link capacity. Because of this, the network experiences a drop in network utilization due to sender backoff [30].

Random packet spraying, an alternative to ECMP, can be used to resolve hotspots as it allows for a greater granularity for load balancing [3]. However, this can result in reordered packets at the receiver, triggering false retransmissions upon receipt of three duplicated packet acknowledgements [21], in addition to drastically reducing sender throughput [38]. 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 [38].

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 [2] 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 [30] and slow start [24], 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 [27]. However, this means that it inherits problems from both implementations altogether.

Better network utilization can be also achieved using Software-Defined Networking [35, 36] 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 [36].

To combat delay and lower throughput caused by multiple timeouts on wireless networks, retransmission redundancy was implemented using full and partial bicasting over MPTCP [29]. This may not necessarily be effective in the datacenter setup as redundant packets may cause more false retransmissions.

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. In addition, creating reorder-resilient end hosts will combat the negative effects of packet spraying such as decreased throughput. 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.

3.2 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.

3.3 Scope and Limitation

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.

Chapter 4

Methodology

Considering that Multipath TCP (MPTCP) penalizes short flows due to unnecessary overhead due to connection complexity, as well as creates hotspots due to ECMP routing, the metrics to be observed should be flow completion time, and end-host throughput, for short and large flows respectively.

To characterize the behavior of the routing protocol, the end-hosts, and the network, we execute a number of tests varying different parameters related to each. The control group would have end-hosts running TCP without any kernel or network stack modifications.

4.1 Behavioral Considerations

MPTCP is available as an ns-3 model [**?**] or a kernel patch [**?**]. Juggler [2], which modifies the GRO function to fix packet ordering, is only readily available as a kernel patch. Preference is then given to kernel patches, assuming that there are little to no conflicts between both MPTCP and Juggler.

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, which require a routing table storing multiple next-hops. In consideration, the switch must be capable of the three previously described algorithms, with preference to easier switch customization.

4.2 Data Gathering

Scripts will be programmed with the use of network performance tools to properly gauge the throughput, and flow completion time. The different network setups will then be evaluated using the measured performance metrics, and validated by replication on another testbed running on a separate device.

To determine the effects of implementing MPTCP and packet spraying in datacenters, a network will be emulated using Mininet with end-hosts configurable to run TCP or MPTCP, and network switches configurable to run ECMP or Packet Spraying. Next, end-hosts will be configured with and without Juggler. These changes will be weaved together to form multiple setups with varying configurations.

Chapter 5

Preliminary Work

To further understand the current implementation of MPTCP on servers, smaller tests were used and characterized. As the protocol is still in active development, a lot of parameters are available to play around it, resulting in different performances. Future work might focus on automatically setting these parameters to optimally cater to the network activity, completely transparent to the user.

5.1 Setting Up the Test Environment

Initially, three separate testbeds were created using Ubuntu 16.04.3 with a Linux kernel version of 4.10.0-28-generic. Mininet was chosen as the network emulator due to the researcher's comfortability with Python scripting, in addition to prior Mininet experience.

Since Mininet uses the host OS to emulate hosts and switches, only the host OS of the testbench needs kernel preconfiguration. MPTCP v0.93 was then used by patching the kernel of the testbeds.

We based our testing topology from an MPTCP laboratory experiment from an SDN

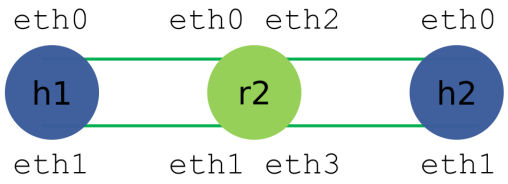

Figure 5.1: Mininet topology for MPTCP tests. Links are of 10 Mbps speed. Topology copied from [1]

Routing Protocol	Path Manager	Initiates subflow creation	Number of used IP address pairs	Number of used port number pairs	Number of subflows
TCP	None	N/A	1	1	1
MPTCP	Default	No	As sender: 1 As receiver: based on sender	As sender: 1 As receiver: based on sender	As sender: 1 As receiver: based on sender
MPTCP	Full mesh	Yes	All, combination	Configurable, default is 1	No. of source IP-port pairs * No. of destination IP-port pairs
MPTCP	Ndiffports	Yes	1	Corresponds to number of subflows	Configurable, default is 2

Table 5.1: Comparison of MPTCP path managers and TCP.

class available online [1]. An overview of the topology can be found in Figure 5.1. Because we only considered a very simple topology for preliminary work, we made do with using a host as a router[1] by configuring it to be able to forward packets. This should be replaced with a configurable router in the actual experimentation.

5.2 Managing Multipath TCP Subflows

Multipath TCP handles the discovery, setup, and termination of subflows through the heuristics of a path manager [19]. We consider three of the four available path managers MPTCP provides [39].

First, the default path manager does not initiate nor announce different IP addresses, but will still accept subflows from end-hosts that do. Next, the full mesh path manager creates flows using all combinations of interfaces (device within end-host to connect to links [40]) of both end-hosts (i.e. assuming all interfaces correspond to unique IP addresses, two end-hosts with three interfaces each will have 9 subflows). Finally, the N-different-ports path manager allows the control over the number of subflows in a connection through the use of different ports. The fourth path manager called binder, isn't considered as it was designed for mobility of devices, a trait not present in datacenter networks.

In summary, we see the comparisons of MPTCP path managers in Table 5.1. To serve as a control group, TCP was characterized as well.

Considering the topology described in Figure 5.1, assuming all links are rated at 10 Mbps (i.e., the minimum supported transmission rate of both devices connected to the link is 10 Mbps), we expect to see TCP, Default MPTCP, and Ndiffports MPTCP to have a sender rate of 10 Mbps, whereas Full Mesh MPTCP can have a sender rate of up to 20 Mbps. This is because the first three would only use 1 IP address pair, and thus is limited to 1 subflow. But, the Full Mesh MPTCP establishes four subflows, because both hosts have two available interfaces, fully utilizing all possible links. This was validated experimentally and the results are shown in Table 5.2.

[1]From this point onwards, we use the terms switch and router interchangeably to mean devices that do layer 2 forwarding and layer 3 routing.

Routing Protocol/Path Manager	Sender throughput (in Mbits/sec)
TCP	9.78
Default MPTCP	9.43
Ndiffports MPTCP	9.46
Full mesh MPTCP	19.1

Table 5.2: Comparison of sender throughput between MPTCP path managers and TCP. The topology used was described in Figure 5.1. Throughput values were taken using command line tool *iperf*.

Routing protocol/Path Manager	FCT (Mean, in ms)	FCT (Standard Deviation, in ms; Relative Standard Deviation)
TCP	7.86	1.53 (19.51%)
Default MPTCP	8.39	1.10 (13.11%)
Full Mesh MPTCP	10.5	3.86 (36.70%)

Table 5.3: Mean, standard deviation, and relative standard deviation of flow completion time for different MPTCP path managers and TCP.

However, more subflows utilized sometimes may come at a cost. For short flows, opening multiple subflows may not be necessary, as the packets may have already been sent even before a new subflow has been established. To prove that MPTCP does indeed introduce some overhead for short flows, the flow completion time was measured between TCP, Default MPTCP, and Full mesh MPTCP.

This experiment once again uses the topology in Figure 5.1. The left host (h1) will request for the web page from the right host (h2), which has a web server running. The web page to be fetched is a default directory index page, and because the web server sends a stream of small packets to complete the request, it counts as a short flow. The tests were ran 10 times with TCP, Default MPTCP, and Full Mesh MPTCP, and the flow completion time (FCT) were noted. We defined the flow completion time as the time difference between the first SYN packet up to the last ACK packet for the last FIN packet.

A summary of results are shown on Table 5.3. Here, we see that TCP has the fastest flow completion time. As Default MPTCP works much like TCP, but has an added establishment overhead over TCP, it has a slower flow completion time. Lastly, since Full Mesh MPTCP opens multiple subflows while transmitting data, packets for connection establishment of other subflows compete for the same available links, resulting in the slowest flow completion time. In addition, since flows are terminated like TCP connections, extra packets are sent to terminate all open subflows before finishing.

As the topology considered for the previous experiments is relatively simpler compared to actual datacenter topologies, we hypothesize that the observed behavior will still hold true, and the effects are amplified to a certain extent.

Chapter 6

Project Schedule and Deliverables

6.1 Halfway Point Deliverables

The following are the deliverables for the project halfway point:

- Mininet script that generates a custom datacenter network (fat-tree) topology
- Initial experiment testbed (Mininet topology, MPTCP-enabled network)
- Performance metric scripts/tools (throughput, flow completion time)
- Software switches capable of ECMP
- MPTCP performance metrics (throughput, flow completion time) for a fat-tree topology
- TCP performance metrics (throughput, flow completion time)

6.2 Final Deliverables

The following are the final deliverables for the project:

- Software switches capable of Packet Spraying
- Juggler-enabled end-hosts in experiment testbed
- MPTCP performance metrics (throughput, flow completion time) for a fat-tree topology with switches running either ECMP or Packet Spraying, and end-hosts with or without Juggler enabled.

6.3 Gantt Charts

In addition to the following tasks, all researchers must document their work, and continue working on the paper for the entire research period.

Tasks	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Compile Linux kernel for environment																	
MPTCP configuration and testing	■	■															
Create a datacenter network (fat-tree) topology on mininet																	
Collect and design performance metric tools			■	■													
Implement ECMP on switches					■	■	■										
Initial simulations (TCP vs MPTCP)																	
Implement Packet Spraying on switches								■	■	■							
Compile, add, and test Juggler to kernel											■	■					
Design of datacenter traffic simulation												■					
Extensive Simulation and Comparisons													■	■	■		
Buffer																■	■

Table 6.1: Charles Alba's Tasks

Tasks	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Compile Linux kernel for environment	■	■															
MPTCP configuration and testing	■	■															
Create a datacenter network (fat-tree) topology on mininet																	
Collect and design performance metric tools			■	■													
Implement ECMP on switches																	
Initial simulations (TCP vs MPTCP)					■	■	■										
Implement Packet Spraying on switches								■	■	■							
Compile, add, and test Juggler to kernel											■	■					
Design of datacenter traffic simulation												■					
Extensive Simulation and Comparisons													■	■	■		
Buffer																■	■

Table 6.2: Kyle Gomez's Tasks

Tasks	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Compile Linux kernel for environment	■	■															
MPTCP configuration and testing	■	■															
Create a datacenter network (fat-tree) topology on mininet			■	■													
Collect and design performance metric tools																	
Implement ECMP on switches					■	■	■										
Initial simulations (TCP vs MPTCP)																	
Implement Packet Spraying on switches								■	■	■							
Compile, add, and test Juggler to kernel											■	■					
Design of datacenter traffic simulation												■					
Extensive Simulation and Comparisons													■	■	■		
Buffer																■	■

Table 6.3: Rene Quinto's Tasks

Bibliography

[1] C.-H. Ke, "Multipath tcp test." Website, Dec. 2017.

[2] 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.

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

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

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

[14] 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.

[15] 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.

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

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

[18] 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.

[19] J. Postel *et al.*, "Transmission control protocol rfc 793," 1981.

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

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

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

[23] A. Ford, "Multipath tcp architecture: Towards consensus."

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

[25] 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.

[26] 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.

[27] 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.

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

[29] 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*, pp. 9–12, IEEE, 2016.

[30] 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.

[31] 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.

[32] 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.

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

[34] 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.

[35] 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.

[36] 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.

[37] 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.

[38] 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.

[39] U. catholique de Louvain, "Multipath tcp - linux kernel implementation : Users - configure mptcp browse." Website, Aug. 2017.

[40] R. Coltun, D. Ferguson, J. Moy, and A. Lindem, "Rfc 5340, ospf for ipv6," *IETF. July*, vol. 24, 2008.