A comparative study of the Multipath TCP for the Linux kernel

Shubham Saini, Korhan Demirkaya
University of California, San Diego
shubhamsaini@ucsd.edu, kdemirka@ucsd.edu

ABSTRACT

Multipath TCP (MPTCP) is an extension to TCP, with the ability to distribute network traffic across multiple connections. Data is transmitted across the available connections simultaneously, thus providing reliability and congestion control mechanisms. While there appear to exist some obvious benefits to using MPTCP, we are still a long way from adopting it on an internet-scale. In this work we validate some of the goals of MPTCP and compare it against TCP. We identify some areas where MPTCP might not honor some of its key principles and suggest some solutions for the same.

1. INTRODUCTION

The Transmission Control Protocol (TCP) was designed in 1974 as part of Internet Protocol Suit, with the idea of single device using a single network interface. A TCP operation consist of three phases: Connection establishment, data transfer and connection release. The TCP connection establishment process is presented in Figure 1. Connection is established with 3-way handshake by the sender sending SYN packet. The receiver acknowledges by sending SYN+ACK packet. Then the sender sends an ACK to complete the establishing the connection and starts sending data. The data is sent in segments with sequence numbers. ACKs are sent back. After the communication is done, TCP connection is closed either by sending FIN or RST packet. In TCP connection is established between two IP addresses, thus restricting an application to restrict creation of one TCP connection through a single link. This leads to severe link under utilization, reliability and efficiency issues.

To get around these issues, a multipath congestion control algorithm [Barré 2011] [Wischik et al. 2011] was proposed that improves throughput and fairness compared to single-path TCP. Multipath TCP (MPTCP) is an extension to TCP, with the ability to distribute network traffic across multiple connections. Data is transmitted across the available connections simultaneously, thus providing reliability and congestion control mechanisms. Currently this protocol is under active development, and is being increasingly deployed by major commercial entities like Apple for their Siri product and Samsung on their Edge devices.

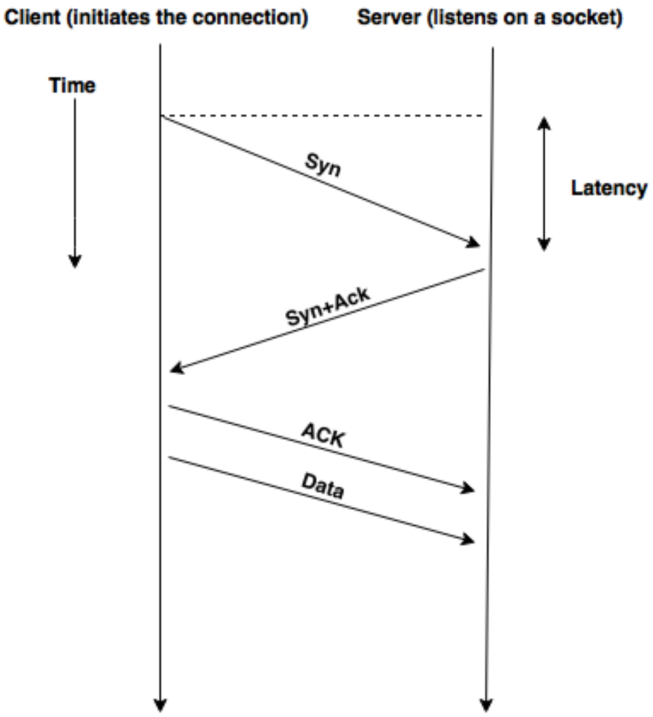

Figure 1: Three way handshake of TCP

MPTCP allows multiple subflows to be set up for a single MPTCP session. When an initial subflow is established like a regular TCP flow with MP_CAPABLE handshake, additional subflows can be added to the MPTCP session with MP_JOIN packets. This process is shown in Figure 2 Unlike TCP the data transfer can happen in any of the subflows with special data sequence numbers.

MPTCP differs from regular TCP in the following ways:

1. While TCP establishes a connection between two interfaces, MPTCP is designed to establish a connection between two hosts.

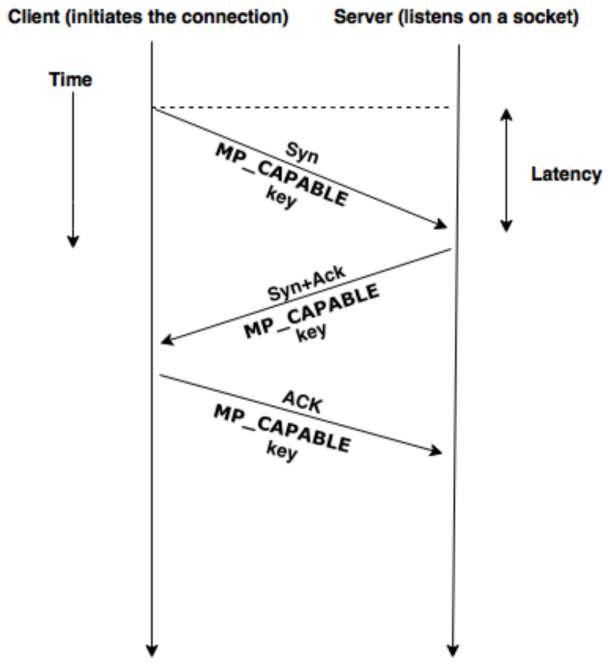

Figure 2: Handshake mechanism of MPTCP

2. TCP works with a single path at any given time, but MPTCP can create flows across multiple available paths.

3. TCP has no link handover capability, but MPTCP has an added reliability guarantee feature by using all available paths.

The primary goal of this project is the verify the three advantages of MPTCP as mentioned previously. Specifically,

- **Efficiency:** MPTCP should send traffic on the most efficient (least congested, least RTT) path.
- **Reliability:** MPTCP should provide reliable communication across the available paths.
- **Bandwidth Aggregation:** MPTCP should provide bandwidth aggregation capabilities by making use of all available resources.
- **Fairness to TCP:** MPTCP should use as much resources as available, while being fair to TCP

Remainder of the paper is organized as follows: In Section 2 we cover mention some work related to the Multipath TCP. Next in Section 3 we describe our testbed and experiments. We present our findings in Section 4. Finally, we conclude with some insights in Section 5.

2. RELATED WORK

MPTCP works by distributing load across multiple subflows. One simple way to control congestion is to apply standard TCP Reno algorithm, but that would give MPTCP unfairly high share of link bandwidth. Several congestion control algorithms for algorithms have been proposed that solves the fairness issue and helps MPTCP transfer traffic through least congested path.

2.1 Linked Increase Congestion Control

Alias Linked Increase (LIA) algorithm works by using the congestion information on all subflows and then adjusting the congestion window. This is active on the additive increase part of the congestion avoidance phase.

For each ACK received for flow i, increase congestion window $cwnd_i$:

$$Min\left\{\frac{\alpha \cdot B_{ack} \cdot M_{SS_i}}{\sum_{i=0}^{n} cwnd_i}, \frac{B_{ack} \cdot M_{SS_i}}{cwnd_i}\right\} \quad (1)$$

Where,
α describes the aggressiveness of the multi-path flow
B_{ack} is number of acknowledged bytes
M_{SS_i} is the max segment size of flow i
n is the number of flows

Further, for each loss on window i, decrease window size by $cwnd_i/2$

Alpha is computed by taking the observed behaviors of all the multi-path subflows:

$$\left(\sum_{i=0}^{n} \text{cwnd}\right) \frac{Max\left\{\frac{cwnd_i}{RTT_i^2}\right\}}{\left(\sum_{i=0}^{n} \frac{cwnd_i}{RTT_i}\right)^2} \quad (2)$$

This algorithm ensures that MPTCP is fair to competing TCP flows and helps moving traffic away from congested paths.

2.2 Opportunistic Alias Linked Congestion Control

One issue with LIA is its inability to achieve both its goals at the same time due to trade off between resource pooling and responsiveness. This leads to fairness issues for competing TCP flows. OLIA [Khalili et al. 2013] was proposed as an extension to LIA to solve this problem.

For each ACK received for flow i, increase congestion window $cwnd_i$:

$$\frac{\frac{cwnd_i}{RTT^2}}{\left(\left(\sum_{i=0}^{n} cwnd_i\right) \cdot \left(\frac{cwnd_p}{RTT_p}\right)\right)^2} + \frac{\alpha_i}{cwnd_i} \quad (3)$$

Where,
$cwnd_p$ is window size of a path p with largest congestion window.
RTT_p is RTT of a path p with largest congestion window.

i is adjust parameter for a path i.
n is number of subflows.

The first part of the equation provide optimal resource pooling while the second part guarantees responsiveness and non-floppiness.

Further, for each loss on window i, decrease window size by $cwnd_i/2$

2.3 Balanced Linked Adaptation Congestion Control

Both LIA and OLIA suffer from unfairness to TCP or unresponsive to network condition changes, including when all paths have same RTT. BALIA [Walid et al. 2015] generalizes these algorithms to provide a good balance between their problems. The adaptation process is as follows:

$$\left(\frac{x_i}{RTT_i \cdot \left(\sum_{i=0}^{n} x_k\right)^2}\right) \cdot \left(\frac{1+a_i}{2}\right) \cdot \left(\frac{4+a_i}{5}\right) \quad (4)$$

Where,
$x_i = cwind_i/RTT$
$a_i = Max(x_k)/x_i$
n is number of subflows.

Further, for each loss on window i, decrease window size by

$$\left(\frac{cwnd_i}{2}\right) \cdot Min\left\{a_i, 1.5\right\} \quad (5)$$

The multiplicative decrease path of BALIA differs from TCP Reno by a factor of [1,1.5] which ensures faster recovery and higher responsiveness.

3. DESIGN

3.1 Topology

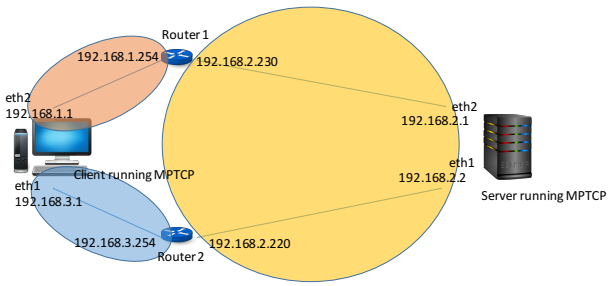

Figure 3: Testbed Topology

A local network has been setup using VirtualBox virtualization solution, as presented in Figure 3. It consists of a client and a server running Lubuntu 14.04, two routers running Ubuntu Server 14.04. The routers run an instance of Quagga Routing Suite [Jakma and Lamparter 2014] software that adds routing capabilities to any system. Each colored region represents a subnet, implemented using VirtualBox internal network interface feature. Each interface on a device within a colored region is only visible to other interfaces in the same region. Further, MPTCP for Linux [Paasch et al. 2013] has been installed on the client and the server.

3.2 MPTCP for linux

MPTCP for the linux kernel is developed using the standard RFC6824 [Ford et al. 2013], which describes how the protocol should work and is maintained by the Internet Engineering Task Force (IETF). It supports a variety of other platforms including MacOS, Fedora, CentOs, Android, OpenWRT etc. MPTCP for Linux includes several congestion control algorithms (LIA, OLIA, BALIA, wVegas), and two schedulers (Lowest RTT and Round Robin). For the purposes of this project, we use different congestion control algorithms which we describe in the following sections.

Due to the absence of actual physical links, the total bandwidth across all links remains the same. As a result MPTCP sends traffic on just a single link the bandwidth on a single link is same as bandwidth across all links. Therefore bandwidth was limited to 2 MBPS for router interfaces using wondershaper for the experiments.

Bandwidth measurements were taken using iperf3, that establishes a service for traffic transfer between a client and server and measures the traffic performance. We can set the number of flows, target bandwidth, segment size among other parameters using iperf3.

3.3 Experiments

3.3.1 Efficiency

One of the goals of MPTCP is to send most of the traffic across the most efficient path, in terms of congestion and RTT. To validate this experiment we start with two paths of equal capacity, congestion and RTT. During the transfer, we introduce congestion on one of the links and monitor if there are any changes in the MPTCP traffic. We next move the congestion away from previously selected path to another path to see if MPTCP quickly adapts to changing conditions.

3.3.2 Reliability

Because of its ability to use multiple paths at the same time, MPTCP has an obvious benefit over TCP in terms of reliability. We test this aspect of the protocol by transferring a file from the server to client across multiple connections and breaking one of the connections during the transfer. We next repeat this experiment with a single path TCP connection.

3.3.3 Bandwidth Aggregation

An added advantage of the reliability property of MPTCP is bandwidth aggregation. By making use of

all available resources, MPTCP provides higher total throughput between two hosts. We measure the total bandwidth and transfer times of a large file using MPTCP with two paths and compare the results to a single path TCP. We add additional connections during the transfer to monitor the MPTCPs adaptation to changing conditions.

3.3.4 *Fairness*

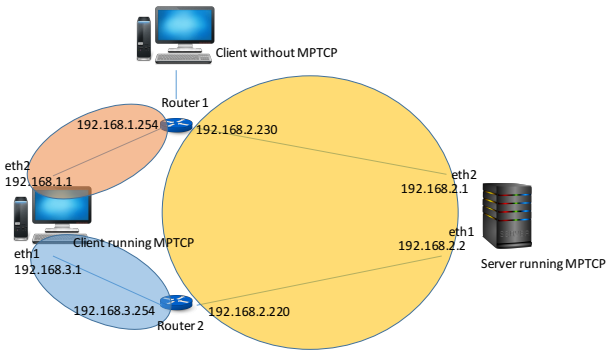

Figure 4: Topology to test fairness

Perhaps the most important goal of MPTCP is to use as much resources as available, while being fair to competing TCP flows. Specifically, MPTCP should give at least as much bandwidth as TCP would on the best path while taking no more capacity on any link than a single path TCP would. To validate this goal we add a new client C2 to router 1, as shown in Figure 4. We start a single path TCP flow between C2 and the server. We next add a multi-path flow from the older client and see how it affects the throughput of the TCP flow from C2.

3.3.5 *HTSIM Simulation*

For any experiments that we find anomalous behavior for, we run the simulation using HTSIM [Raiciu et al. 2010] to corroborate our findings or debug any errors in our implementation. While HTSIM does not exactly replicate the MPTCP behaviour due to absence of reordering mechanism at host level and using congestion window size in bytes, it does reveal the underlying traffic patterns across links.

4. RESULTS

We know present the results for the experiments mentioned in the previous section.

4.1 Efficiency

The results of the MPTCP efficiency performance is given in Figure 5. We noticed that MPTCP sends traffic using both the available channels initially. Since the total bandwidth of the virtual links remains the same

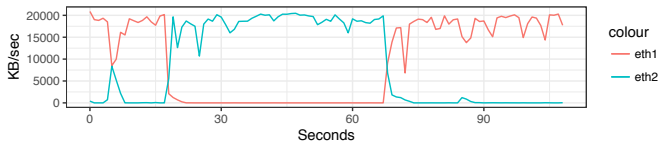

Figure 5: MPTCP Efficiency

in the absence of capacity restrictions, MPTCP moves all the traffic over to link 1 within the initial 10 seconds. Next, as we introduce congestion on link 1 at the 70 second mark, MPTCP quickly adapts to it and moves the traffic over to link 2. This shows that MPTCP favors sending most of the traffic on the most efficient path.

4.2 Reliability

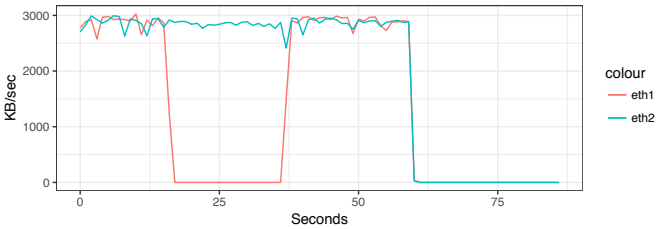

Figure 6: MPTCP Reliability

When the link capacities were restricted to 2 MBPS, MPTCP sends the traffic across both the available links to achieve maximum throughput, as shown in Figure 6. When we artificially brought the link 2 down at the 15 second mark, MPTCP maintained continuous file transfer without any interruption. Further, MPTCP started using link 2 again when it was brought back alive at the 40 second mark.

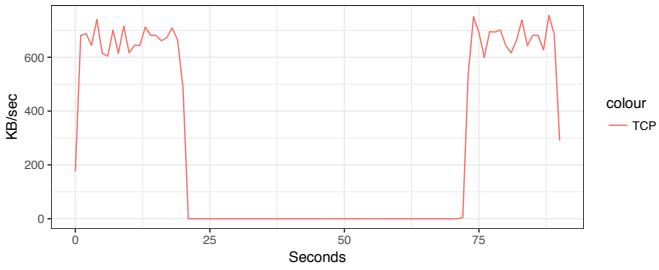

Figure 7: TCP Reliability

In contrast, TCP did not offer any reliability in data transfer as shown in Figure 7. As we bring the link under use down, TCP halted the file transfer and did not use the other available path.

4.3 Bandwidth Aggregation

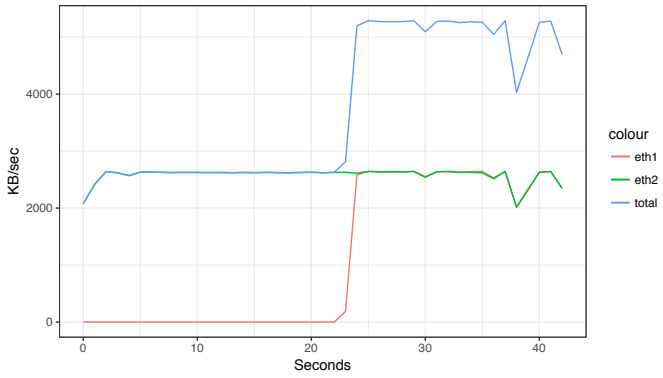

Figure 8: MPTCP Bandwidth Aggregation

MPTCP has a huge advantage over TCP when it comes to bandwidth aggregation. Because of its ability to use of all available resources, MPTCP provides higher total throughput between two hosts. The results of this experiment is shown in Figure 8. Initially we have just one active link between the hosts. When we bring another link alive at the 20 second mark, MPTCP adapts to this change and starts using both the links simultaneously, thereby adding more bandwidth.

4.4 Fairness to TCP

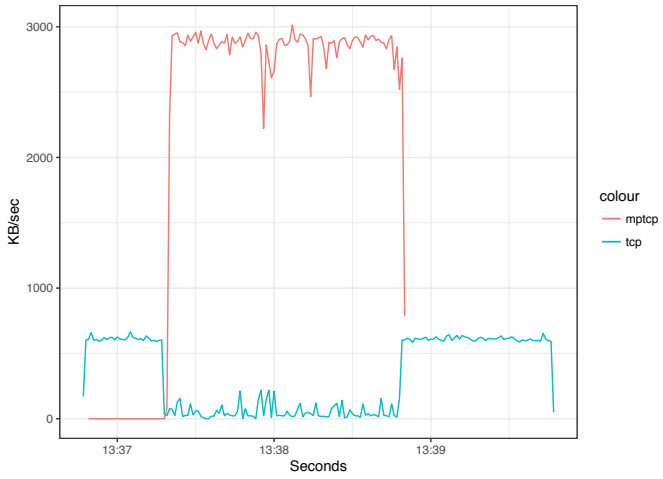

Figure 9: Using TCP Reno congestion control at multi path client

The most interesting results came out of our experiments to validate the MPTCP fairness goal towards competing TCP flows. We start a TCP flow from client2 which does not have multi path capability, and start a multi path flow from client1 after 30 seconds. Figure 9 shows the result when client1 uses TCP Reno congestion control algorithm. As evident from the figure, the competing TCP flows bandwidth is reduced severely (by a factor of 4), while the MPTCP flow grabs a massive share of the link capacity.

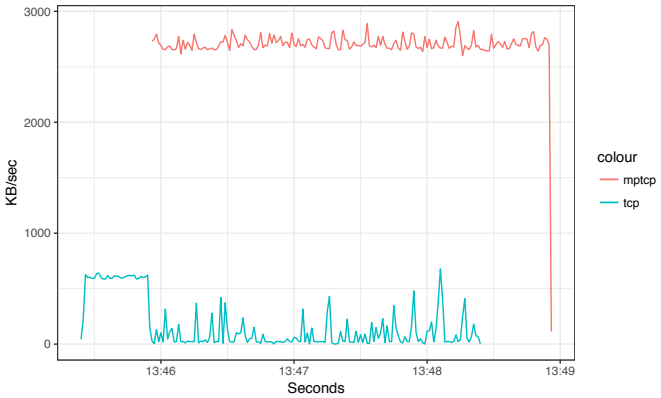

Figure 10: Using BALIA congestion control at multi path client

We next changed the congestion control algorithm at client1 to BALIA. The results are in Figure 10. We notice that the competing TCP flow gets higher bandwidth than previously, but it oscillates between high and low bandwidth share. As in the previous case, MPTCP flow uses the bulk of the link capacity, thereby not honoring other TCP flows on the link.

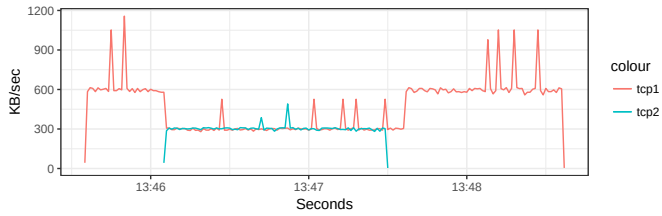

Figure 11: Comparing two competing TCP flows

To check if there was some misconfiguration on the multi path enabled client, we switch the client to standard TCP mode. As show in Figure 11, client1 now uses its fair share of the bandwidth.

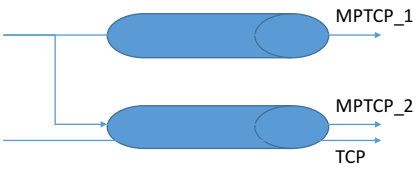

Figure 12: HTSIM Topology

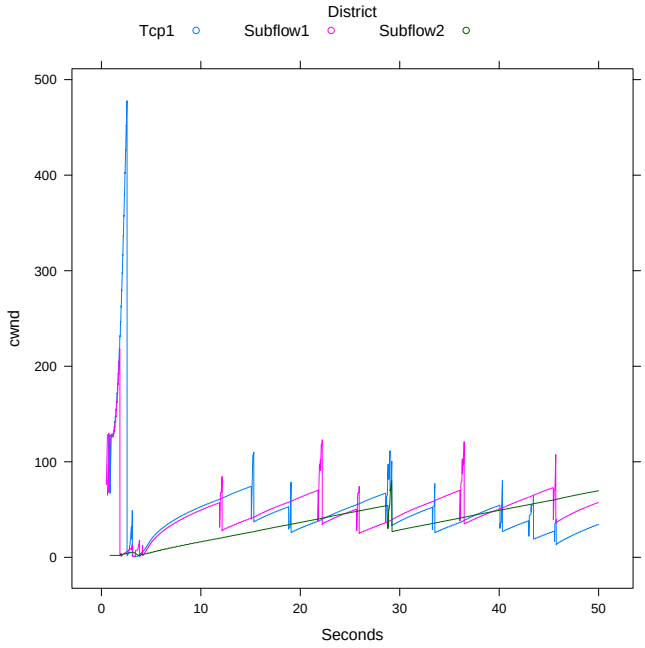

Figure 13: HTSIM Simulation

To verify our results we used HTSIM simulator that allows testing for a large number of TCP flows concurrently. It has a simple MPTCP implementation with congestion control coupling. We implemented a simple bottle-neck topology as shown in Figure 12, with the bottleneck rate of 500 packets/sec where each packet is 1000 bytes. We studied the change in congestion window size for each of the flow, as shown in Figure 13. Contrary to our earlier experiments, HTSIM simulation showed fairness of MPTCP towards TCP flows, as the congestion window for the TCP flow increases for periods of 10 seconds before reducing to half. While we were unable to extract the bandwidth information due to complex writer API offered by the simulator, we have good reason to believe that increasing congestion window sizes for long durations correspond to higher bandwidth.

5. CONCLUSION AND DISCUSSION

We have presented a verification of MPTCP protocol along with testing multiple distinct MPTCP congestion control algorithms. Our experimental design setup using a simple topology resulted into the following key findings:

1. MPTCP selects efficient paths for sending traffic.
2. MPTCP offers bandwidth aggregation, and out performs standard TCP when looking at throughput.
3. MPTCP offers reliable communication methods that is immune to link failures.
4. The default congestion control algorithm supplied with MPTCP for Linux does not offer fairness towards other TCP flows.

Some unexpected observations in our experiments that require further analysis:

1. The HTSIM simulator works differently from the MPTCP implementation for Linux kernel.
2. MPTCP is not fair to other TCP flows sharing a single bottleneck.

MTTCP is under aggressive development currently, and a long way from being adopted across Internet. We believe our project reinforces a lot of key goals of MPTCP, and will prove helpful in future development of the protocol. Due to time constraints and lack of prior relevant experience, we were unable to troubleshoot some unexpected findings and hope to get back to those some time in future.

6. DATA AND CODE AVAILABILITY

All the data generated during this project can be found on our GitHub repository `https://github.com/shubhamsaini/cse222a_multipath_tcp`. The repository includes key instructions to setup a Multipath TCP client and server and write routing tables. We also include a link to download our testbed in the form of a VirtualBox application.

7. REFERENCES

Sébastien Barré. 2011. Implementation and assessment of modern host-based multipath solutions. *Universite catholique de Louvain, Louvain-la-Neuve, Belgium* (2011).

Alan Ford, Costin Raiciu, Mark Handley, and Olivier Bonaventure. 2013. *TCP extensions for multipath operation with multiple addresses.* Technical Report.

Paul Jakma and David Lamparter. 2014. Introduction to the quagga routing suite. *IEEE Network* 28, 2 (2014), 42–48.

Ramin Khalili, Nicolas Gast, Miroslav Popovic, and others. 2013. Opportunistic linked-increases congestion control algorithm for MPTCP. (2013).

Christoph Paasch, Sébastien Barré, and others. 2013. Multipath TCP in the Linux kernel. *Available from {www. multipath-tcp. org.}* (2013).
Costin Raiciu, Christopher Pluntke, Sebastien Barre, Adam Greenhalgh, Damon Wischik, and Mark Handley. 2010. Data center networking with multipath TCP. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*. ACM, 10.
Anwar Walid, Qiuyu Peng, Jaehyun Hwang, and Steven H Low. 2015. Balanced linked adaptation congestion control algorithm for MPTCP. *IETF, Individual Submission, Internet Draft draft-walid-mptcp-congestion-control-03* (2015).
Damon Wischik, Costin Raiciu, Adam Greenhalgh, and Mark Handley. 2011. Design, Implementation and Evaluation of Congestion Control for Multipath TCP.. In *NSDI*, Vol. 11. 8–8.