National University of Singapore
School of Computing
CS5229: Advanced Computer Networks
Semester I, 2021/2022

Term Project - Research Reproducibility

Last updated at: 2021/10/23 14:24:13 (Singapore Time)

This document provides detailed information about the term project. For the term project, you would work in teams to gain an in-depth understanding of a state-of-the-art research paper and reproduce the main results from it. This project is inspired from the "learning through reproducibility" practice at the Stanford University [1]. Note that this document would be periodically updated as the project progresses and you would be notified of the updates.

Learning Objectives

The main goal of this project is for the students to a gain a detailed, in-depth understanding of a significant research paper, its key ideas, and its key results. At the end of the project, your understanding of the research paper is expected to be beyond the "third pass" [2] of simply reading the research paper. Besides, you would pick up valuable skills in building system infrastructure for research experimentation, running research experiments, processing data and plotting graphs. You would also learn to make sense of the various graphs and gain expert intuition about the system that you are running (including the experimentation platform).

Logistics and Timeline

The project needs to be carried out in **teams of 3**. Each team will work independently of the other teams (no inter-team collaboration). Table 1 shows the timeline including the deliverables.

Table 1: Project Timeline

Date-time	**Milestone**	**% grade**
12 Sep 23:59	Form Teams of 3	-
24 Sep 23:59	Submit Part 1 (Coursemology)	5
19 Nov 23:59	Submit Parts 2 and 3: Complete source code (Github) AWS AMI Final Report	25

More detailed submission instructions would be updated in this document in the due course. The **most urgent** task right now is to form teams and get started ASAP.

Team Formation

Please use LumiNUS and/or Microsoft Teams to quickly form teams of 3. Appoint one member of the team as the *team representative*. Send an email to us (rajjoshi, ayush, benleong @comp.nus.edu.sg) with the subject "CS5229 Team Registration" while cc'ing all the team members. In the email, mention the registered names and emails of all the team members. Also mention which team member is going to be your team representative.

Team Representative

The team representative will use his Coursemology account for any submissions related to the project. The team representative will also create an AWS account and load AWS credits in his account (details below). To allow other members to access the provided AWS credits, the team representative should add them as IAM users to his/her AWS account with appropriate permissions.

AWS Credits

Thanks to the sponsorship from AWS Educate, we are able to provide each team with AWS credits worth USD 130. Once you register your team, we will be sending you instructions on how to add these credits to the team representative's AWS account.

You would use AWS EC2 instances mainly for part-3 of the project. USD 130 should be more than sufficient for this project, if you use them wisely. For example, don't use AWS instances for code development. For smaller scale experiments (to check correctness), try to use smaller and cheaper EC2 instaces (e.g. 4 vCPUs). Only while running larger experiments, use powerful EC2 instances (e.g. 8 vCPUs). Also, never leave an EC2 instance running if you are not using it. Each team would be responsible to monitor its own AWS billing and avoid incurring charges to the team representative's credit/debit card. We would NOT be able to provide you with additional AWS credits if you happen to exhaust USD 130.

Intra-team Disputes

You are all graduate students and so we would NOT entertain any intra-team disputes e.g. team representative acting on his own and making incorrect/incomplete submissions without consent from other members. Your final report will have a clear statement of contributions signed by all the team members. In general, we expect all team members to contribute equally. Unless serious descrepancy, the grading will be done for the team as a whole and all members will receive the team's grade.

Consultation

At this point, we do NOT plan to hold any dedicated consultation hours for the project. So feel free to tag the TAs and ask your questions/queries on the Microsoft Teams channel "Reproducibility Project". If you have a question/query that you think should be asked privately, then with high probability you should NOT be asking that question/query. This is because answering such questions/queries privately to you would be unfair to the other project teams. Note that the project is **partially open-ended**, since we would be reproducing the paper's results on a more realistic platform (Mininet) and not the one

used in the paper. Therefore, we are open to hold on-demand consultation(s) if/when a majority of teams seem to run into trouble.

Plagiarism

Using the Internet for guides and resources is generally fair and in fact expected in this project. However, any attempt of "copying" across teams or from the Internet would be dealt as per the NUS plagiarism policies. Your source code as well as final report would be subjected to rigorous plagiarism checks which includes any publicly available source code on the Internet. When in doubt, please ask on the Teams channel.

Research Paper(s)

The research paper chosen for this semester is ***Jellyfish: Networking Data Centers Randomly*** [3] from Usenix NSDI 2012. This paper takes a radical new approach towards incremental datacenter network topologies. Your main task would be to build an in-depth understanding of this paper and reproduce some of the experimental results from it (details provided below). However, the results from any research paper are compared to a baseline which is often the state-of-the-art at the time of the paper's submission/publication. In this case, the baseline used for most of Jellyfish paper's results is the fat tree datacenter network. Therefore, you would also need to understand the fat tree paper [4] to a very good extent. Also, no network runs without proper routing and transport. For this reason, you would also need to build a good understanding of Equal Cost Multipath Routing (ECMP), k-shortest path routing as well as multi-path TCP (MPTCP) [5, 6].

Project Overview

The project is divided into 3 parts. The first part is the lightest and it will only test your understanding of the concerned research papers. This part needs to be submitted early (see Table 1). This is to ensure that you remain on track for parts 2 and 3. Part 2 is moderately heavy and requires you to implement topology construction and topology traversal algorithms which would play a key role in part 3. You will also reproduce several numerical results in part 2. Part 3 is where we expect you to spend majority of your time. Part 3 involves building the experimental infrastructure in Mininet and running experiments to reproduce the target results.

Reproducibility

Bear in mind that it may not be very straightforward to reproduce a result (figures/tables) because every single detail (e.g. parameters) may not be spelled out clearly in the paper. Therefore, try your best to reproduce a result as closely as possible. We do NOT expect you to get exactly identical results. From the grading perspective, you will be given full credit even for a different result as long as your different result is in itself reproducible and you can explain what exactly you did and justify your parameter choices. Although rare, it is possible that the results in the paper could be entirely wrong! If you come across such a situation and are confident about your finding, please contact us before writing in your final project report.

Final Report

We **highly recommend** that you keep writing your final report as you progress in the project. For each reproduced result (figure/table), please note down what assumptions or parameter choices you had to make and justify the same. Please also note down any of your learnings while reproducing the result. In case you end up with a different looking result (figure/table), please mention in full details how exactly you reproduced your result, why did you choose certain parameters, etc.

Part 1: Understanding the paper(s)

In this part, you would answer a few questions which are intended to test your understanding of Jellyfish and other related papers. A solid conceptual understanding would set you up for success in parts 2 and 3. The following are all short answer questions. Please be concise and limit your responses to a maximum of 5 sentences each. Submit your answers on Coursemology through your team representative's account. Any other submissions will NOT be considered for evaluation.

Questions

1. What is the main problem that Jellyfish is trying to solve?
2. Was the problem not solved before? Using examples of one or two closest competitors of JellyFish, describe where and how existing[1] works fall short?
3. What is the key idea of JellyFish? What is the intuition behind JellyFish being able to provide high bandwidth along with flexibility?
4. Calculate the number of servers and switches in a JellyFish network with the following parameters: (i) N=30, k=10, r=1; (ii) N=30, k=10, r=4.
5. What is the bisection bandwidth of a network?
6. Suppose you have 4-port switches and you want to build a fat tree network. How many total switches would you need and how many servers would the network support? Explain with calculations.
7. What is ECMP routing? Will ECMP routing work well with a fat tree network? Explain why or why not? What about ECMP routing with JellyFish?
8. Suppose you are building a real-world datacenter network. Would you use JellyFish (Yes/No)? Explain the reasons behind your answer.
9. What is a dual-homed host/server?
10. Is it necessary to have a dual-homed host/server in order to run MPTCP in a datacenter network (Yes/No)? In other words, can MPTCP work with a single-homed host? Explain why or why not.

Part 2: Numerical Reproduction

In this part, you would be reproducing some of the results from the JellyFish paper numerically i.e. without running any network experiments. For this part, you would NOT require any EC2 instances. Save your AWS credits for part 3.

[1] existing at the time of JellyFish's publication

Please complete the following tasks and submit on Coursemology. To help you get started, we have provided some starter code in **proj-starter.html** (or the corresponding Jupyter notebook **proj-starter.ipynb**) on Coursemology.

1. **Topology construction:**
   (i) Reproduce Figure 1(a) and submit on Coursemology. Not by drawing digitally but by programmatically constructing the graph/topology and then plotting it. There is no need to draw/plot the concentric circles. The provided starter code should help you do this right away.
   (ii) JellyFish graph construction: Using the networkX library, implement the function `build_jellyfish_graph` and submit on Coursemology. The function template is provided in the starter code and is also available on Coursemology. You would be using this function to build the JellyFish topology for the rest of the project (including in Part 3). Therefore, be sure to add the attribute 'type' to each node in your JellyFish graph. The attribute 'type' can have a value as either 'switch' or a 'host'. This will come very handy while converting your JellyFish graph into a Mininet topology in Part 3. On Coursemology, we have provided some basic tests to help you check your implementation of `build_jellyfish_graph`. Our tests rely on your implementation specifying the type of each node correctly. Please refer to the resources section for help on using the networkX library.
   (iii) Reproduce Figure 1(b) and submit on Coursemology.
2. **Topology analysis:** Reproduce the following figures and submit on Coursemology. You can use various tools for plotting graphs (see resources section below). Since we are doing almost everything in Python, the recommended way is to use Python-based tools such as Matplotlib or Plotly (see resources).
   (i) Figure 1(c) (Path Lengths): Notice that the parameters for JellyFish are not mentioned anywhere. However, JellyFish is compared to the "same equipment" fat-tree. This is a good enough hint to figure out the JellyFish parameters.
   (ii) Figure 2(a) Bisection Bandwidth: The bisection bandwidth for a JellyFish network with given parameters is not fixed. This is because the wiring is random. Therefore, you will have to resort to plotting the lower bound of the *expected* bisection bandwidth of JellyFish. **Hint:** The key to computing the lower bound of JellyFish's bisection bandwidth may not be near the description of Figure 2(a) in the paper.
   (iii) Figure 2(b): Equipment Cost
   (iv) Figure 9: Path Diversity

Part 3: Mininet Reproduction

In this part, you would use Mininet to experimentally reproduce the following results from the JellyFish paper:

- Figure 1(c) using Mininet
- Table 1 using Mininet

In order to do the above, you will first implement JellyFish and fat tree network topologies in Mininet. Then you will implement ECMP and 8-shortest path routing. You will also configure and run MPTCP[2] with 8 subflows.

Following are some "guideline" steps to help you with part-3. There is no strict ordering between the steps i.e. several of them could be distributed among team members and executed in parallel.

1. Getting started with AWS: account credits, access to other team members.
2. Setting up the VM environment: Python, Mininet, required libraries, OpenFlow controller (does not have to be FloodLight).
3. Topology construction + basic routing in Mininet: Fat tree, JellyFish.
4. Reproducing Figure 1(c).
5. Random permutation traffic in Mininet: generate and measure random permutation traffic at the server-level.
6. Implementing ECMP routing: for both fat tree and JellyFish.
7. Implementing 8-shortest paths routing for JellyFish.
8. MPTCP setup: configuring and getting it to run correctly with ECMP and 8-shortest paths routing.
9. Reproducing Table 1.

Parametrized implementation. It is important that the experimentation infrastructure that you build is parametrized so that you could run experiments at different scales. For example, while development and testing, you could work with a smaller sized network (k=6 fat tree and equivalent JellyFish) and on your local Linux machine or on a small AWS instance (save your AWS credits!). For the final experiment run, you could simply change the parameter and run a large network on a large AWS instance.

Below we provide more details on the "guideline" steps.

Getting started with AWS

Redeeming Credits. [Team representative] Login to AWS console as a root user. Go to Billing Management Console and the Credits tab in it. Then click "Redeem Credit" and submit your voucher code. Follow this video if you are still can't figure out.

Adding team members to the representative's AWS account. [Team representative] You have to go to Identity and Access Management (IAM) console from your AWS main console and add one IAM user for each team member. The team representative can continue using the root account and does not necessarily have to add an IAM user account. Follow this video if you want more guidance. Following are a few key things while adding IAM user accounts for your team members:

- Give both Programmatic and AWS Management Console access
- For permissions, simply attach existing policies directly. Select the following existing policies: **AmazonEC2FullAccess**, **AWSBillingReadOnlyAccess**

[2]Since built-in MPTCP support in the latest Linux kernel could be limited, you may need to configure MPTCP all by yourself.

Requesting for increased vCPU limits. [Team representative] Every AWS account has a default limit on the total number of vCPUs that you can have across all your VM instances. The team representative needs to send a request to increase this limit so that you may be able to run larger scale experiments later. From the AWS main console, search and go to the "Service Quotas" console. Be sure to check your region is Singapore (top-right corner). From the "Service Quotas" dashboard go to "Amazon Elastic Compute Cloud (Amazon EC2)". Search for "Running On-Demand Standard (A, C, D, H, I, M, R, T, Z) instances". If both your applied and default quota values are less than 16, then request quota increase to 16. It might take 2-3 working days for your request to be processed. You might even get some email communication from the AWS customer support team. Meanwhile, you can start working with your current vCPU quota.

Managing AWS Credits and Billing. All team members should be able to monitor the balance credits in the "Credits" tab of the Billing Management Console. All team members should also be able see the incurred charges in the "Bills" tab of the Billing Management Console (typically after certain hours of use).

Setting up the VM environment

AWS VMs are all managed from the EC2 console. Before doing anything in the EC2 console, be sure to change the AWS region (see top-right corner when on EC2 console) to ap-southeast-1 (Singapore).

Initially you could start with a smaller sized VM instance with 4 vCPUs. You can check the costs for different AWS on-demand instances here. Be sure to select the region as Singapore. You can also filter by the number of vCPUs.

Starting an EC2 instance. You can follow this tutorial for launching an EC2 instance. Essentially you do the following steps:

1. **Select an AMI:** We recommend Ubuntu Server 20.04 LTS. We also recommended to go with the default 64-bit (x86) architecture. You can choose 64-bit Arm if you want cheaper instance types in the next step. It is rare, but still possible that you may run into some software compatibility issues on an Arm architecture.
2. **Select instance type:** For larger-scale experiments, it is recommended to have an instance with at least 8 vCPUs, 16GB RAM and high network performance (e.g. c5.2xlarge). For smaller-scale development and testing, you may use less powerful and cheaper instance (e.g. t3.xlarge).
3. **Configure instance:** Use default settings.
4. **Add storage:** You would likely not need much. 30-40 GB should suffice. Please do NOT encrypt the storage.
5. **Add tags:** Just skip.
6. **Configure Security Group:** Keep simply ssh for now. You may add TCP port 80 and 8080 to access OpenFlow controller GUI, etc.
7. **Review and launch:** Create a new key pair. Download the .pem file that will be used to ssh into your VM instance.

On the EC2 dashboard, you can see your instance to be running. Selecting the instance you can see its public IPv4 address and Public IPv4 DNS in the details below. You could use either of them to ssh into your VM as following:

```
ssh -i <path to .pem file> ubuntu@<public IPv4 address/DNS>
```

The default user account "ubuntu" has sudo priviledges and requires no password. It is recommended that you use the default user account "ubuntu" among all team members. For this you could either share the .pem file or add ssh public keys of other team members to the VM's ubuntu user account (i.e. in the file /home/ubuntu/.ssh/authorized_keys).

Setting up the necessary software. To setup the software you could follow steps from the Teams post to setup VM on M1 Mac (starting from step 2). You would likely not need any GUI. Mininet can be controlled completely without GUI including the individual hosts in the network (see this). So only steps 4 and 5 are relevant. To use the latest version (2.3.0) of Mininet, please skip Mininet installation from Ubuntu packages and instead use pip3 to install Mininet. You will also need to install fnss, networkx, and other required Python libraries. Please stick to Python3 for this project.

For the OpenFlow controller, it is not necessary to stick to FloodLight. FloodLight does not support ECMP out of box. riplpox supports ECMP. But it is old and you might need to resolve some version issues with the latest Mininet. For 8-shortest paths routing, one way could be to do reactive routing. The first TCP packet (typically SYN) will miss the table in the first hop switch and will be sent to the controller. The controller would then install rules so as to enforce 8-shortest paths routing scheme. ECMP could be done in a similar way too. Such reactive routing requires OpenFlow's Packet-In and Packet-Out messaging. This may not be very straightforward with FloodLight. Therefore the POX OpenFlow controller is another good option to consider.

Topology construction and basic routing

The fnss library can help to convert a fnss topology to a Mininet topology. See example here. Similarly, for converting your JellyFish networkx graph to a Mininet topology, you can either try using the fnss library or write your own function to build a Mininet topology from a given networkx graph.

For basic routing, if you are using a controller like FloodLight, it should be able to discover the shortest paths between the hosts and install default routing rules accordingly. Even with POX, you should be able to setup network forwarding with basic L2 learning.

Reproducing Figure 1(c)

For this task, you will need to figure out how to measure path lengths in a Mininet network experimentally. There are several ways to do this. You could use traceroute or the ttl value reported by ping. But these require some support from the switches/hosts in the network and may not be supported out of box. You would likely need to add relevant match-action rules to the switches.

In the simplest form, you could setup your link delays such that based on the ping RTT measurements, you could reliably tell the path lengths. If you are using ping RTT measurements, then be sure to send ping packets at short intervals using the -i option and also send a fixed large number of ping probes using the -c option. This should give you more accurate measurements.

Random permutation traffic in Mininet

Computing a random permutation of hosts should be easier given you have the graph representations of your network topologies. Note that the class `fnss.DatacenterTopology` is derived from the `networkx.Graph` class and so has all the members and methods of the `networkx.Graph` class. It is not scalable to manually open the xterms for each pair of hosts and start traffic between them. So please use Mininet APIs to run Linux commands (e.g. start/stop iperf/iperf3 server or client) on individual hosts. You would need to do some testing to figure out how to read back the output e.g. throughput reported by iperf/iperf3. Note that for a large number of hosts in the network, it might take some time to start iperf/iperf3 traffic among all random permutation pairs. So the throughput over the initial 10s of seconds would be required to be ignored.

Some useful iperf3 options for this task would be `--omit`, `--get-server-output`, `--json`, `--logfile`. Note that this is an untested suggestion and this may or may not help in fully realizing the solution.

Implementing ECMP routing

This is one of the challenging parts of the project. ECMP routing is typically done by the dataplane of switches in production networks. The switch dataplane first finds that for a given packet there are multiple output ports to forward it to its destination. Then it uses a hash function on the 5-tuple[3] of the packet to choose one of the possible output ports.

Now, in Mininet we are using OpenVSwitch (OVS) which is a software switch and as far as we know OVS does not support ECMP in the switch dataplane. There are a few possible ways to do this.

The *dynamic* way would be for all switches to send the first packet of a TCP flow to the controller and let the controller dynamically add a forwarding rule. For example, when a TCP flow starts, the first switch will send the TCP SYN packet to the controller, and the controller will do the hash computation on the 5-tuple and choose one of the several possible output ports. Accordingly, the controller will add a rule with the 5-tuple match and output port action to the first switch. The same will happen when the packet reaches the next switch and so on. The process will also follow in the reverse direction for the TCP SYN-ACK packet. Once all switches have the rules added, the subsequent packets of the TCP flow will go through the established path without going to the controller. Do this for all the TCP flows and the traffic would be ECMP-balanced in the network. This is approach is most flexible, but requires appropriately using OpenFlow's Packet-In and Packet-Out messaging.

The *static* way would be to configure routes on all the switches in advance. This is possible since the random permutation traffic is already computed i.e. we know all <src, dst> pairs of servers that would be sending traffic in the experiment. Further, using iperf3 it is possible to choose both the TCP source and destination port numbers deterministically. From programming assignments you already know how to fix iperf3 server's (receiver's) port. Refer to this post to see how you can bind an iperf3 client (sender) to a specific source port. Given this, you already know the 5-tuples for all the TCP flows (both TCP data and ACK directions) that would run in the network. Then for a given flow starting from the source host, at each switch hop you can do ECMP routing

[3] 5-tuple includes the following fields from a packet: IP src, IP dst, IP proto, TCP src port, and TCP dst port

computation (offline) and decide on the next hop. Accordingly, you can pre-populate match-action rules for all flows on all switches. Please note that iperf3 also creates its own TCP connection (with different port numbers) which exchanges some data before and after the specified TCP flow runs. You do not need to worry about this, other than ensuring that there is a default route (need not be shortest or load balanced) between any two pair of hosts. This could be populated by the OpenFlow controller by default. Note that doing it in the static way is more fail proof since there is nothing happening at runtime. You also have more flexibility to choose the OpenFlow controller since the controller need not support Packet-In and Packet-Out messaging.

Implementing 8-shortest paths routing

Once you have figured out how to implement ECMP, implementing 8-shortest paths routing would be quite similar. Instead of doing the ECMP hashing and modulo to figure out the next path, you would do 8-shortest paths routing. Similar to ECMP this could also be done in a dynamic or static way and the trade-offs between the two ways are also similar.

Setting up MPTCP

Configuring and getting to run MPTCP on hosts with a single network interface (non-multihomed) is yet another challenging part of the project. Reportedly, multi-subflow MPTCP is supported in Linux kernel 5.7.x, but there are still issues with it (see this). In this project, we not only need multi-subflow MPTCP, but we need multi-subflow MPTCP to run with a single network interface. In other words, we need a single MPTCP connection to create multiple MPTCP subflows using different port numbers. This way the routing scheme (ECMP or 8-shortest paths) running in the network could hash the subflows to different paths. Now, the port numbers used by MPTCP for different subflows may not configurable/deterministic. In this case, the static way of doing ECMP or 8-shortest paths routing will come into trouble unless you can find out some other *deterministic* fields in the TCP packets (with supported OpenFlow match) which can help to distinguish packets belonging to different MPTCP subflows.

Fall back: As a fall back, it is okay to relax the project's requirement and allow the hosts to be multi-homed i.e. each host has eight network interfaces (with different IP addresses) to the ToR switch. You can add these extra interfaces to the switch's total number of ports but still consider it a "same equipment" network. Now, running MPTCP subflows over different interfaces will produce TCP packets with different pairs of source-destination IP addresses for each subflow. For ECMP or 8-shortest paths routing in the network, you do not need to necessarily use all 5 fields of the "standard" 5-tuple. You could only use 3 (IP source, IP destination and IP proto) and deterministically map different MPTCP subflows to different paths without worrying about the TCP source and destination ports used by MPTCP. This way, you can do ECMP and 8-shortest paths routing in a static way and still be able to run MPTCP.

As of today (23 Oct 2021), the latest Linux kernel running on Ubuntu 20.04 is 5.11.x and you could check what level of built-in MPTCP support it has. There are apparently pre-built MPTCP AMIs in certain AWS regions (see this). However, these AMIs likely do not have the latest MPTCP version which is v0.95.1. Your best bet would be to use the latest MPTCP from https://multipath-tcp.org/. There are a few easy ways to install MPTCP. Please refer to the MPTCP Setup subsection below for more resources on setting up MPTCP. Your best bet would be to install a pre-built MPTCP Linux kernel and be able

to successfully boot and run your machine with the MPTCP kernel. It it possible that to do so, you may need to move back to an older Linux kernel and/or an older Ubuntu distribution. But if nothing works, then building the MPTCP-based Linux kernel from source could be the last resort. Do **NOT** use your local machine while trying to setup MPTCP. Things are very likely to break since you are messing around with the Linux kernel. Please use a fungible AWS instance which you could terminate if things go wrong and then start over again.

Submission Instructions

Project Report

[More details would be updated later.]

Evaluation Script and Code Packaging

[More details would be updated later.]

Submission of Amazon Machine Image (AMI)

[More details would be updated later.]

Resources

Following are some additional (but non-exhaustive) resources to help you with the project. These are simply pointers to what you would otherwise find through web search. You do NOT have to read them all. Just pick up whatever is necessary.

NetworkX

- NetworkX
- NetworkX tutorial

Tools for plotting Graphs

Please use **offline** tools only such that you can provide us a high-level script in your code submission that can run your experiments and also generate the plots.

- Gnuplot, Gnuplot Demos
- Matplotlib, Matplotlib Tutorials
- Plotly

Fat Tree and JellyFish

- A blog post on the JellyFish paper
- Slides from Cornell CS5413 on FatTree
- Slides from Costin Raiciu on Datacenter Network Topologies. Use the slideshow tab and not the auto-generated video.
- JellyFish conference talk + slides

ECMP

- ECMP on Wikipedia
- RFC2991 Multipath Issues in Unicast and Multicast Next-Hop Selection (short article)
- Short lecture video by David Wetherall (University of Washington)
- ECMP Video Tutorial from Palo Alto Networks

MPTCP

- An Overview of Multipath TCP (blog article)
- Overview of Multipath TCP and its applications - interesting talk by Costin Raiciu at Microsoft Research
- SIGCOMM'11 talk slides for the paper – Improving datacenter performance and robustness with multipath TCP. Use the slideshow tab and not the auto-generated video.
- SIGCOMM'11 talk video for the paper – Improving datacenter performance and robustness with multipath TCP. The talk video is under the "Supplemental Material" section.

MPTCP Setup

- MPTCP with a newer Linux kernel
  - Faster Internet with MPTCP (Multipath TCP)
  - https://github.com/onemarcfifty/mptcp-tools
  - https://www.openmptcprouter.com/
- Resources on setting up MPTCP with an older Linux kernel (some may not work). But certain configuration steps could come handy.
  - https://yongchaohe.github.io/linux/2019/09/29/Multipath-TCP.html
  - https://gist.github.com/BigNerd95/6bff4dd74c23f24a35d9e3e0c2f8c9de
  - https://gist.github.com/brandonheller/2730049
  - https://askubuntu.com/questions/624058/installing-new-kernel-with-apt
  - https://gist.github.com/tovask/316f0dc855f2459042af403688590a7f
- MPTCP on RHEL
- The fastest TCP connection with Multipath TCP (tutorial/blog).
- MPTCP support in Linux kernel

References

[1] Lisa Yan and Nick McKeown. Learning networking by reproducing research results. *SIGCOMM Computer Communication Review*, 47(2):19–26, 2017.

[2] Srinivasan Keshav. How to read a paper. *SIGCOMM Computer Communication Review*, 37(3):83–84, 2007.

[3] Ankit Singla, Chi-Yao Hong, Lucian Popa, and P Brighten Godfrey. Jellyfish: Networking data centers randomly. In *Proceedings of NSDI*, 2012.

[4] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. A scalable, commodity data center network architecture. In *Proceedings of SIGCOMM*, 2008.

[5] Alan Ford, Costin Raiciu, Mark Handley, Olivier Bonaventure, et al. RFC 6824: TCP extensions for multipath operation with multiple addresses. 2013.

[6] Costin Raiciu, Sebastien Barre, Christopher Pluntke, Adam Greenhalgh, Damon Wischik, and Mark Handley. Improving datacenter performance and robustness with multipath TCP. In *Proceedings of SIGCOMM*, 2011.