CS5229 Project Report

Chen Yu (A0242092M)
Zijian Luo (A0224725H)
Zhengdao Zhan (A0229934U)

1 Summary of completion

Figure Name	Figure1(a)	Figure1(b)	Figure1(c)	Figure2(a)	Figure2(b)	Figure9
State	Fully	Fully	Fully	Fully	Fully	Fully

Table 1: Part 2

Figure Name	Figure 1(c)	TCP 1 flow	TCP 8 flows	MPTCP 8 subflows
State	Partially	Fully	Fully	Partially

Table 2: Part 3

2 Part 2: Numerical Reproduction

2.1 Figure 1(c) Path Lengths

Figure 1(c) shows the comparison of path length distribution between 686-server Jellyfish and same-equipment fat-tree topology. The X-axis represents the length of the shortest path between two server nodes, ranging from 2 to 6, while the Y-axis shows the number of server pairs that fit the path length over all pairs. The reproduction process is listed as follows.

1. Calculate the per-switch port number and overall switch number to support a 686-server fat-tree topology. According to the definition of fat-tree,
$$server = \frac{port^3}{4} \tag{1}$$
We can derive that
$$port = \sqrt[3]{4 \times server} \tag{2}$$
After calculating the per-switch port number, the switch number can be easily derived as follows.
$$switch = (port/2)^2 + port^2 \tag{3}$$
Since both two topologies share the same equipment, these two values are also the number of switches and ports in Jellyfish.

2. Build Jellyfish and fat-tree topology graph with NetworkX. For fat-tree, we set the calculated per-switch port number and build the network with the function `fat_tree_topology` from `fnss`. For Jellyfish, we use our own function which takes in switch number, per-switch port number and per-switch host number. To get an ideal theoretic result, the number of hosts per switch is set to $\lceil server/switch \rceil$, which means that servers spread over the switches.

3. Analyze the path-length distribution. In this step, we fetch the server nodes from the generated graph, permute all possible server pairs to find their shortest path length and record the number of server pairs that fit in the targeted path length. The result is shown as follows.

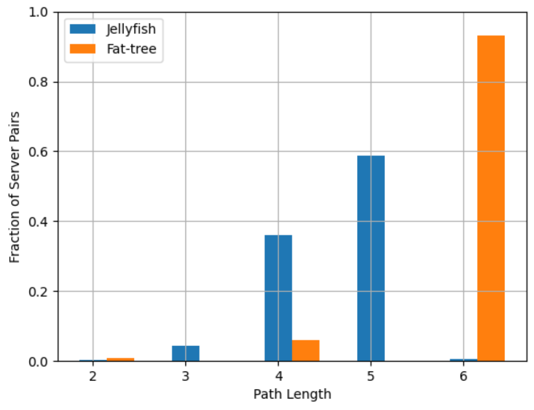

Figure 1: Reproduced Figure 1(c)

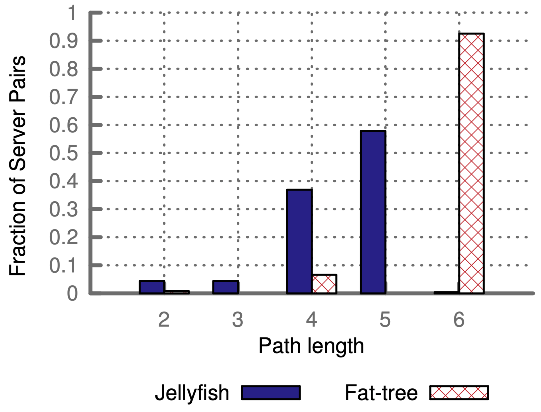

Figure 2: Original Figure 1(c)

2.2 Figure 2(a) Bisection Bandwidth

Figure 2(a) shows the normalized full bisection bandwidth of two network topologies with different switch number and per-switch port number. The X-axis is the number of servers in thousands, while the Y-axis is the normalized bisection bandwidth. Bisection bandwidth is the worst-case bandwidth spanning any two equal-size partitions of a network. Since the fat-tree topology follows a tree structure, there is only one path from one partition to the other. Hence, for fat-tree, normalized bisection bandwidth is always 1. However, Jellyfish follows a very different calculation strategy, which is listed as follows.

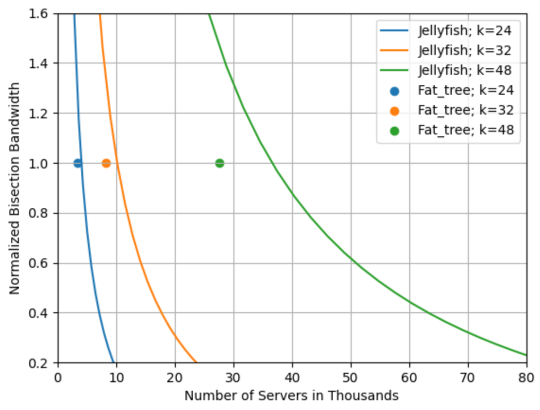

Figure 3: Reproduced Figure 2(a)

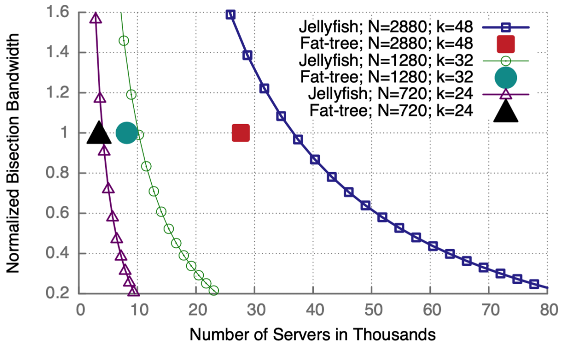

Figure 4: Original Figure 2(a)

1. First, the number of server in the Jellyfish network is uncertain because we only fix the number of switches and the per-switch port number while missing the number of ports that are connected to the hosts, namely the value $(k - r)$ in the paper. Therefore, we need to calculate the range value of the server count. Typically, the minimum possible number of server is
$$min\_server = switch \times 2 \tag{4}$$
The maximum possible server count is
$$max\_server = switch \times \lceil port/2 + 5 \rceil \tag{5}$$
Same as the original figure, there is an assumption that the point used for calculation represents the multiplication of switch count in the target range.

2. Then as described in the paper, since the network is an r-regular graph with N nodes, the lower bound bisection bandwidth is

$$N \times \left(\frac{r}{4} - \frac{\sqrt{r \ln 2}}{2} \right) \tag{6}$$

Here N is the number of switches and r is the number of free ports for switch-to-switch connection. Ideally, due to the assumption that the number of servers is the multiplication of switch count, r is set as

$$r = port - server/switch \tag{7}$$

To obtain the normalized bandwidth, the value needs to be divided by the switch count in the partition.

2.3 Figure 2(b) Equipment Cost

Figure 2(b) compares the cost of building a full bisection-bandwidth network between the two topologies. The X-axis represents the number of servers in thousands, while the Y-axis is the total number of ports in thousands, which simply equals the number of switches (in thousands) times the degree of each switch.

$$Y = N \times k \tag{8}$$

For a JellyFish network topology, the total number of servers are given by $N \times (k - r)$, where N is the number of switches, k is the degree of the switch and r is the number of ports of a switch used to connect other switches. Therefore, in this case, the expression for the X coordinate is

$$X = N \times (k - r) \tag{9}$$

Since Figure 2(b) represents the equipment cost vs the number of servers at full bisection bandwidth. The Y value of 1.0 in Figure 2(a) represents full bisection bandwidth. According to Figure 2(a), it can be easily derived that the normalized bisection bandwidth of the network is 1. Thus, the value of bisection bandwidth equals half the number of servers.

$$N \times \left(\frac{r}{4} - \frac{\sqrt{r \ln 2}}{2} \right) = \frac{X}{2} \tag{10}$$

As mentioned before, $X = N \times (k - r)$, therefore, we can further derive that

$$\frac{r}{4} - \frac{\sqrt{r \ln 2}}{2} = \frac{k - r}{2} \tag{11}$$

Then

$$\sqrt{r} = \frac{2\sqrt{ln2} + \sqrt{4ln2 + 24k}}{6} \tag{12}$$

Combining Equation 8 and Equation 9, we can get

$$Y = \frac{X \times k}{k - r} \tag{13}$$

Substituting k with r shown in Equation 12, we can get the relationship between Y and X only regarding to k, from which the lines in Figure 2(b) can be easily drawn.

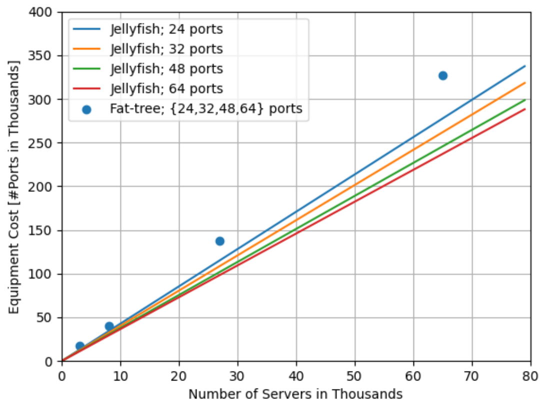

Figure 5: Reproduced Figure 2(b)

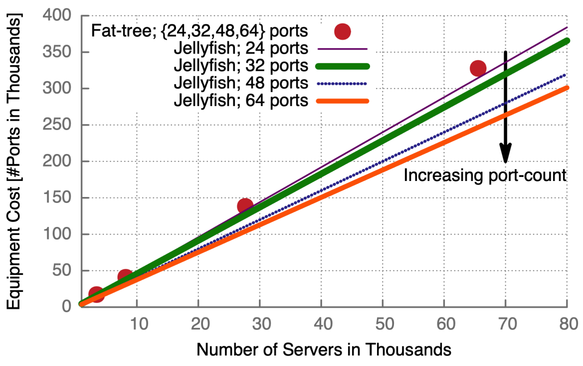

Figure 6: Original Figure 2(b)

2.4 Figure 9 Path Diversity

Figure 9 shows the number of distinct paths on each link in the Jellyfish networks. The graph compares three types of routing algorithms including 8 shortest paths MPTCP, 64-way ECMP, and 8-way ECMP. The X-axis is the rank of links, starting from the link with the lowest diversity to the link with the highest diversity. The Y-axis represents the number of distinct paths that go through the link. The reproduction procedure is listed as follows.

1. Build the 686-server Jellyfish network. Since we are only given the server number, we need to calculate the number of per-switch port and switch count ourselves before passing the parameters to establish a Jellyfish network. This part is the same as `Step 1` of Figure 1(c).

2. Calculate the diversity value of each link in the network. For convenience, we add an attribute to all the edges called *"diversity"* to record the number of path that a link is on. Its initial value is 0. Then for each routing policy, we permute all the possible server pairs in the graph, go through the k shortest paths between two servers and add one to the *"diversity"* attribute of a link when there is a path on that link.

   Note that the idea to find the k shortest paths of ECMP and MPTCP is a bit different. We use the NetworkX method `nx.shortest_simple_paths` to find k shortest paths of MPTCP, which only returns path with no repeated nodes. However, for ECMP, the NetworkX method is replaced with `nx.all_shortest_paths`, which computes all shortest paths in the graph.

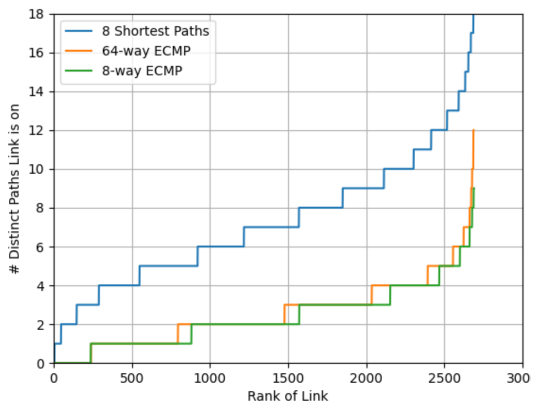

Figure 7: Reproduced Figure 2(a)

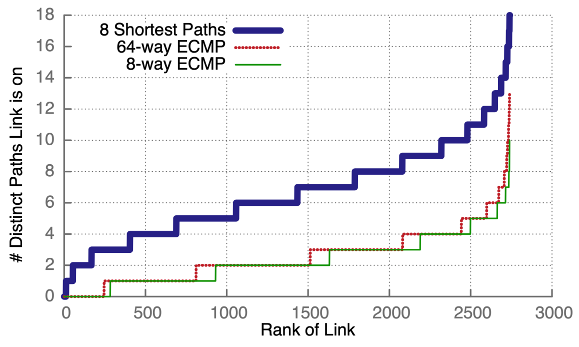

Figure 8: Original Figure 2(a)

3 Part 3: Mininet Reproduction

3.1 Figure 1(c)

Our implementation is mainly based on mininet and POX Openflow Controller. The routing algorithm is directly adopted from POX forwarding methods. To reproduce Figure 1(c), we measure the path length with the pair-wise ping time cost, which makes the result more precise compared to that in Part 2 that is mainly based on ideal estimation.

3.2 Table 1

3.2.1 ECMP Routing

In this part, the routing rules of the whole network is generated by our script. We utilize the functions in `NetworkX` to set the routing flow, then deploy these flows to the switches with Openflow controller. Similar as the process in Figure 9 reproduction, the program uses `nx.all_shortest_paths` to first find all possible ECMP paths between two servers, then chooses the K shortest result to register in the network.

3.2.2 MPTCP Configuration

We tried to configure MPTCP directly on the AWS server but failed. The instance always crashes when it is switched from the original kernel to the compiled MPTCP kernel. Having no idea about what's going on, we tried another workaround method, through which it is possible to get the package from repo release without local compilation. The detailed procedures are in our private GitHub Repo, https://github.com/InanisV/CS5229-JellyFish/blob/master/MPTCP%20Setting.pdf.

4 Overall learnings and takeaways

There are some overall takeaway ideas from us about this project.

1. Open source project sometimes does cause problem. We should doubt, debug and do it ourselves quickly once a bug pops up. For example, we found that we cannot set up the network connection when using a certain flag `forwarding.l2_multi` with the POX OpenFlow controller. Actually, the problem is led by POX itselves. After digging into the source code, we solved the issue by fixing a bug in ARP packet handling.

2. Multi-threading is really a good idea to reduce running time of a program that costs much time. For those repetitive actions that do not need to be done in sequence, it is nice to put the tasks into a thread pool.

5 Signed statement of contributions

- Chen Yu (A0242092M): Part 1, Part 2 Reproduction, Part 3 Controller & Routing
- Zijian Luo (A0224725H): Part 1, Part 3 Throughput Measurement & MPTCP Configuration
- Zhengdao Zhan (A0229934U): Part 1, Part 2 Network Establishment, Part 3 Server, Report

Signatures

- Chen Yu (A0242092M)
- Zijian Luo (A0224725H) LUO ZIJIAN
- Zhengdao Zhan (A0229934U) Zhan Zhengdao