Advanced Computer Networks 2022
Vrije Universiteit Amsterdam
Lin Wang, George Karlos, Florian Gerlinghoff

Lab2: Data Center Network Topology

Type:	Group
Maximum points:	10 (+5 with bonus)
Submission:	Canvas upload (code and report)
Deadline:	Wednesday November 16, 2022, 23:59
Contact:	vu.acn.ta@gmail.com

The goals of this lab are

- Write code to generate fat-tree and jellyfish topologies
- Compare the properties of the two topologies
- Write a report to document your findings

Please pull the `lab2` directory from GitHub by running `git pull` in the `acn22-labs` directory. You are supposed to work in the `lab2` folder for this lab.

Note: Please use the template (i.e., `acn22-lab2-report.tex`) we provide for writing the report. There are specific instructions regarding what should be included in the report. For the submission, please put your report (in PDF) in the `lab2` folder and rename the `lab2` folder to `lab2-groupx` where `x` is your group number. (Please follow this naming convention **strictly** to make our life easier.) Then, please compress the folder into a `.zip` file. Please rename the folder before compression so that the folder name is correct when we decompress it. Please submit the zipped folder on Canvas to the assignment "Lab2".

1 Generating Fat-tree and Jellyfish Topologies

As we have learned in the lecture, there are different ways to construct a data center network topology (e.g., using a multi-rooted tree or a fat-tree). Different topologies come with different performance properties such as bandwidth and latency, and with different costs such as the number of switches needed to interconnect a given number of servers.

So far, the topologies we have learned for building a data center are regularly structured. Yet, there is another completely different way of interconnecting servers in a data center—randomly generating links between switches to form a network. One example is the Jellyfish topology described in the following paper.

[Jellyfish] Ankit Singla, Chi-Yao Hong, Lucian Popa, P. Brighten Godfrey. Jellyfish: Networking Data Centers Randomly. USENIX NSDI 2012. [pdf]

In the first part of this lab, you are supposed to write code to generate two different topologies: fat-tree and Jellyfish. The fat-tree topology is regularly constructed and you should be able to implement it by following closely the original fat-tree paper (one of the papers in the week three reading list, also discussed in the lecture). The generation of the Jellyfish topology is detailed in the Jellyfish paper linked above. Please read the papers thoroughly and make sure you understand the main ideas before you start coding.

You are requested to use **Python 3** in this lab since in the next lab, the topologies you generate now will be used by the Ryu controller which only supports Python. Using the same language across the two labs will provide you some convenience. You are also provided a code template. Please do not change the high-level structure in the template (e.g., class names) and keep the existing variables and functions. You are free to add extra variables and functions in the classes as you deem necessary.

In the report, please document the following:

- How we should run your code
- Your general approach to generating the topologies
- Some small-scale example topologies (e.g., plotted with `matplotlib`) you have generated

2 Comparing the Properties of Fat-tree and Jellyfish

The Jellyfish paper claims that the Jellyfish topology outperforms the fat-tree topology in various aspects. The paper verifies these claims by showing a variety of interesting figures. In this part, we are going to reproduce two of these figures. This also allows us to verify whether the results in the published paper are reproducible or not. Reproducibility is a very important step for open science and is especially valued in computer systems/networks research since the experiment setup is usually complex and reproducibility is hard to verify.

2.1 Reproducing Figure 1(c)

This figure compares the shortest path length distribution for server pairs in fat-tree and Jellyfish. For each pair of servers, find out the length of the shortest path between the servers. Once you do this for all possible server pairs, you can calculate the fraction of server pairs (y-axis in the figure) that use a shortest path of a specific length (x-axis in the figure). Note that you are supposed to implement all algorithms from scratch, e.g., you need to compute the shortest path by implementing the Dijkstra algorithm (or other algorithms if you prefer). You should not use any external Python libraries for this purpose.

Using the code you wrote in the first part, you can generate the two types of topologies, i.e., fat-tree and Jellyfish, in different sizes. In particular, Figure 1(c) assumes to interconnect 686 servers, which can be achieved by using 14-port switches in a fat-tree network. For Jellyfish, you should also use the same number of 14-port switches and distribute the servers evenly to these switches. Since Jellyfish is a random topology, you should perform multiple (ten in the paper) trials and average the results. Finally, please use bar-plots to depict the distribution as done in the paper and compare it to the original figure in the paper.

In the report, please include the following:

- How we should run your code
- Your general approach to reproducing the figure
- The figure you have reproduced
- Explanation for the comparison (e.g., reason about any differences if any)

2.2 Reproducing Figure 9

This figure compares the performance of the Jellyfish network when different routing schemes are employed. The performance is measured by the number of distinct paths each link is on (similar to edge betweenness centrality[1] in graph theory). The routing schemes under comparison are:

- **8 shortest paths:** uses the 8 paths with the least lengths (these paths may have different lengths, but they are ranked the lowest in terms of the path length)
- **8-way ECMP:** uses 8 paths that have an equal length to the shortest path (note that the number of paths could be less than 8 if the total number of paths that have an equal length to the shortest path is less than 8)
- **64-way ECMP:** uses 64 paths that have an equal length to the shortest path (note that the number of paths could be less than 64 if the total number of paths that have an equal length to the shortest path is less than 64)

You can use the Yen's algorithm to calculate k shortest paths simultaneously. The Yen's algorithm is documented at https://en.wikipedia.org/wiki/Yen%27s_algorithm. Please implement the algorithm by yourself, instead of using any external libraries.

The Jellyfish topology you use is the same as the one you have used when reproducing Figure 1(c): 686 servers interconnected with switches that would be used in a 14-port fat-tree topology. The traffic pattern we use in this experiment is called random permutation: each server sends traffic to exactly one other randomly chosen server. This can be achieved by performing a random permutation over the server index and mapping the server indices to the permuted indices. Note that the traffic is likely not symmetric.

In the report, please include the following:

- How we should run your code
- Your general approach to reproducing the figure
- The figure you have reproduced
- Explanation for the comparison (e.g., reason about any differences if any)

3 Assessment Criteria

This lab will be assessed based on the following criteria:

- **Correctly generating fat-tree and Jellyfish: 2 points for each topology.** Your code should be able to generate each of the topologies in different sizes correctly. No partial points for a topology will be given if your code cannot generate the topology or the generated topology is wrong.
- **Successfully reproducing Figure 1(c): 3 points.** Your code should be able to generate the distributions correctly and produce a figure similar to Figure 1(c). Note that hard-coding the distributions without code to generate them will be denied and will receive no points.
- **Successfully reproducing Figure 9: 3 points.** Your code should be able to generate the distributions correctly and produce a figure similar to Figure 9. Note that hard-coding the distributions without code to generate them will be denied and will receive no points.
- Code and report quality: For each of the tasks above, there is one point included in the total points, which is assigned to the quality of your code and report. Your code should be general, e.g., the code can be used to generate the desired topology at any size. The report should contain enough details including your methodology, results (e.g., plots), and analysis of the results in comparison to the original figures.

[1] https://en.wikipedia.org/wiki/Betweenness_centrality

4 Bonus

If you want to challenge yourself further, the following bonus worth **5 points** is available for you. You have to complete the following two tasks in order to claim this bonus. No partial points will be given even if you manage to do partial work.

4.1 Generating Other Topologies

You are expected to generate other two (inspiring but not widely used) data center network topologies (i.e., BCube and DCell) in addition to the aforementioned two. The papers that describe the two topologies are listed below.

[BCube] Chuanxiong Guo, Guohan Lu, Dan Li, Haitao Wu, Xuan Zhang, Yunfeng Shi, Chen Tian, Yongguang Zhang, Songwu Lu, Guohan Lv. BCube: A High Performance, Server-centric Network Architecture for Modular Data Centers. ACM SIGCOMM 2009. [pdf]

[DCell] Chuanxiong Guo, Haitao Wu, Kun Tan, Lei Shi, Yongguang Zhang, Songwu Lu. DCell: A Scalable and Fault-Tolerant Network Structure for Data Centers. ACM SIGCOMM 2008. [pdf]

4.2 Extending Figure 9 to All the Topologies

In reproducing Figure 9 of the Jellyfish paper, you have compared the performance of Jellyfish under different routing schemes. Now, try to carry out the same comparison for the other three topologies, i.e., fat-tree, BCube, and DCell. You should generate one plot for each of these topologies similar to that for Jellyfish and conduct a comparative analysis among the different topologies as to which routing scheme is more friendly to which topology.

In the report, please indicate if you are ready to claim this bonus. If you do so, please briefly explain your approaches to the above two tasks, your results, and discussion on the results.