Advanced Computer Networks (2021)

Lab3: Building A Data Center Network

Type: Group
Maximum points: 15
Submission: report and code on Canvas
Deadline: Monday December 6, 2021
Contact: vu.acn.ta@gmail.com

The goals of this lab are

- Write code to build a fat-tree network in Mininet
- Implement the shortest-path routing on the fat-tree network
- Implement the two-level routing on the fat-tree network
- Design an experiment to compare the performance of the network under the two routing schemes and write a report about it

Note: Please include your group number and all group member names at the beginning of your report to facilitate our grading. Also, please rename the folder to be submitted with name “lab2-groupx” where “x” is your group number and then compress the folder into a “.zip” file. Please do the folder renaming **before** the compression so that the folder name is correct when we decompress the folder. Please submit the zipped folder on Canvas to the assignment “Lab3”.

1. Build a fat-tree data center network in Mininet

In the previous labs, we have learnt how to build a simple network in Mininet and also how to generate a fat-tree topology. The first task of this lab is to build a fat-tree network in Mininet, i.e., building your own data center on your machine! Of course, due to resource limitations, we cannot run a large-scale network on a single machine, so we will limit our attention to a small-scale one, with 16 servers (#port = 4 in fat-tree).

Since we have the code for generating a fat-tree topology already, we can employ the generated topology directly. You may need to slightly modify your code for topology generation. In any case, please first make a copy of your fat-tree topology generation code (i.e., `topo.py` by default) from directory `lab2` to directory `lab3`. This is to make sure your `lab3` submission is self-contained and is not dependent on code in other directories.

You are provided with a code template in the file `lab3/fat-tree.py`. Please complete this template, more specifically the constructor method of the `FatTreeNet` class, to create a fat-tree network instance in Mininet. Note that you should assign correct IP addresses to the servers according to the original fat-tree design described in the fat-tree paper. For the link properties, you can use bandwidth `15Mbps` and latency `5ms` for all links. Once you have done your implementation, you can test it by running:

```
$ bash ./run.sh
```

A controller has also been specified as `RemoteController` running at `127.0.0.1:6653` in the code. When you run the above command, the network together with the controller should be brought up. However, you will not be able to reach another server from a server in the network since all the switches have not set up the correct forwarding tables and by default all packets that arrive at a switch are forwarded to the controller. Since you have not told the controller to do anything, all packets will be dropped.

2. Implement shortest-path routing on the fat-tree network

To enable reachability in the fat-tree network you have just built, you need to implement a routing scheme on the network through the controller. In this task, you are supposed to implement the shortest-path routing scheme on the fat-tree network. In the previous lab, you have already implemented algorithms (e.g., Dijkstra's algorithm) to perform shortest path calculation. You can simply reuse the code you already have.

Your job is to write a Ryu controller application just like the learning switch you implemented in `lab1`, which tells the switches to do shortest-path routing between any pair of servers. You are provided with a code template at `lab3/sp_routing.py`. Please work on this code, but feel free to change anything in the code if needed.

Note that you are not supposed to assume switches are interconnected with each other using static port mappings. In other words, you are under the situation that the switch ports used for connecting any two switches are unknown. You need to obtain such information with topology discovery APIs from Ryu at runtime. You can use API functions like `get_switch()` and `get_link()` to obtain the switches in the network and the network links including port mappings. You can find more details about how to use these functions at:
https://github.com/Ehsan70/RyuApps/blob/master/TopoDiscoveryInRyu.md.

The above functions provide the port mappings between switches, but they do not give the switch ports that are used to connect the servers. You need to find your own way to obtain such information. A hint is to use ARP packets to do the job. Recall that when the controller receives a packet, it records the `in_port`, which can help you obtain the mapping between the IP address of the server and the switch port that the server is connected to. However, before you obtain the correct mapping, you probably need to do flooding for ARP resolution. You cannot do the flooding on the network simply because you have not set up forwarding entries on switches for routing. Think about how to handle ARP resolution without routing information.

Once you manage to calculate a forwarding entry for a switch, you should push a flow entry by invoking the `add_flow` function which is available in the code template already. If you have done `lab1` properly, you should know how to use this function. Here, we simply adopt forwarding based on the destination IP address following the shortest path. Note that for a flow entry that matches on the IP address, e.g., `dst_ip->port`, you may need to match on the `EtherType` of the Ethernet header as well. Please find out which `EtherType` code you should match for IP packets.

Once you have finished implementing your controller application, you can run the following command to bring up the controller:

```
$ ryu-manager ./sp_routing.py --observe-links
```

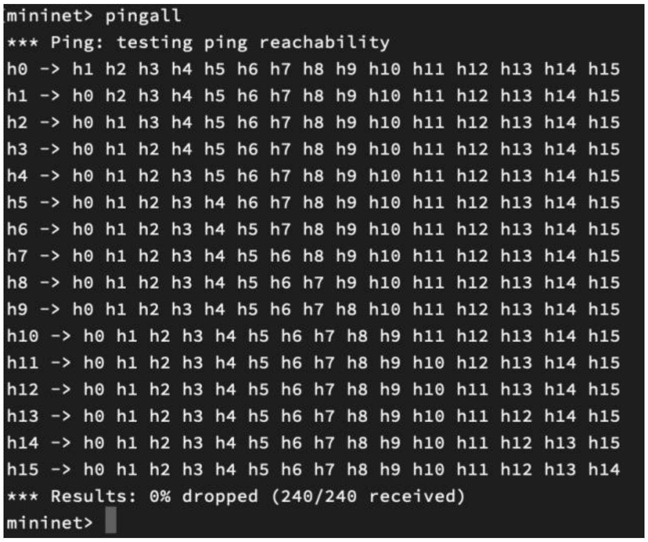
Note that the option `--observe-links` is needed for topology discovery APIs.

You may also want to use the following command to check the flow entries on switches for debugging purpose:

```
$ sudo ovs-ofctl dump-flows s1
```

where `s1` is the ID of the switch that you want to check.

To finally test the correctness of your control program, you can run the `pingall` command in Mininet. You should see that all the servers in the fat-tree network are able to reach each other and there are `0%` packet drops. See a screenshot below:



```
mininet> pingall
*** Ping: testing ping reachability
h0 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15
h1 -> h0 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15
h2 -> h0 h1 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15
h3 -> h0 h1 h2 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15
h4 -> h0 h1 h2 h3 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15
h5 -> h0 h1 h2 h3 h4 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15
h6 -> h0 h1 h2 h3 h4 h5 h7 h8 h9 h10 h11 h12 h13 h14 h15
h7 -> h0 h1 h2 h3 h4 h5 h6 h8 h9 h10 h11 h12 h13 h14 h15
h8 -> h0 h1 h2 h3 h4 h5 h6 h7 h9 h10 h11 h12 h13 h14 h15
h9 -> h0 h1 h2 h3 h4 h5 h6 h7 h8 h10 h11 h12 h13 h14 h15
h10 -> h0 h1 h2 h3 h4 h5 h6 h7 h8 h9 h11 h12 h13 h14 h15
h11 -> h0 h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h12 h13 h14 h15
h12 -> h0 h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h13 h14 h15
h13 -> h0 h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h14 h15
h14 -> h0 h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h15
h15 -> h0 h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14
*** Results: 0% dropped (240/240 received)
mininet> 
```

3. Implement two-level routing in the fat-tree network

Now, let us implement a different routing scheme instead of shortest-path routing.

The two-level routing scheme is the default routing scheme described in the fat-tree paper (see Section 3.5 in the fat-tree paper, please read it carefully). The high-level idea is to use (1) prefix matching for intra-pod traffic and (2) suffix matching for inter-pod traffic. You need to set up the flow entries correctly on all the switches according to the description in the paper. A template is given in `lab3/ft_routing.py`.

Note that you can use the `priority` of a flow entry and if multiple matching flow entries are present, only the one with the highest priority will be matched. For example, you can set up a higher priority for prefix matching entries to prevent intra-pod traffic from being routed outside the pod.

Once you have finished implementing your controller application, you can run the following commands to bring up the controller:

```
$ ryu-manager ./ft_routing.py --observe-links
```

Again, you can run the `pingall` command in Mininet to test the correctness of your control program. You should see that all the servers in the fat-tree network are able to reach each other and there are `0%` packet drops, i.e., the result should be the same as shown in the above screenshot.

4. Compare the performance of the two routing schemes

Once you have done the implementation of the two routing schemes, please design an experiment to compare the performance of the network when applying the two different routing schemes. In Lab0 we have already learned how to test the throughput and latency of a network in Mininet (by using commands like `ping` and `iperf`). You may want to follow the same techniques. Here, the challenge is on choosing the server pairs for testing. Please reason about your choice and document your findings after your experiments by writing a small report.

5. Assessment criteria

This lab will be assessed based on the following criteria:

- **Correctly generating the fat-tree network in Mininet: 4 points.** Your code should be able to generate the fat-tree network in the correct way: the number of servers, number of switches, and the links between all servers and switches should be correct. Note that you should not hard-code any numbers in your code, i.e., your code should be able to generate fat-tree networks in different sizes when supplied with different inputs, although for testing purpose we use #port = 4.
- **Correctly implementing shortest-path routing: 4 points.** Your code should be able to set up shortest-path routing on the fat-tree network. Make sure `pingall` is successful and all the flow entries are correctly set up on the switches.
- **Correctly implementing two-level routing: 4 points.** Your code should be able to set up two-level routing on the fat-tree network. Make sure `pingall` is successful and all the flow entries are correctly set up on the switches.
- **Experiment and report quality: 3 points.** In the report, you should at least explain your experiment plan, the system setup, and the results you obtain (in the form of table or figure). Also, please analyze the results and try to make reasonable conclusions from the results.