PADERBORN UNIVERSITY

Advanced Networked Systems SS24

Lab3: Data Center on Your Computer

Maximum points:	10
Submission:	zipped source code on PANDA
Deadline:	05.06.2024 23:59
Contact:	lin.wang@upb.de

1 Introduction

In the previous labs, we have learned how to build a simple network in Mininet and also how to generate a fat-tree topology. Now, it is time to turn our gained experience into something more serious, specifically a small data center on your computer. The goal of this lab is to build a small fat-tree network, implement different routing algorithms on it, and compare the performance of the different routing algorithms.

Please run `git pull` in the labs codebase to retrieve the code template for this lab. You will see a new folder `lab3` and you are supposed to work in that directory for this lab. You are provided a code template.

⚠ We have identified several cases of possible plagiarism in the previous lab submissions. This is the last warning: Please do not copy code from any other group and do not give out your code to anyone outside of your group. If such cases happen again, they will be reported directly to the examination committee. Also, if you show code that you cannot explain in the interview, the interview will be terminated and you get a zero directly.

2 Fat-tree in Action

The first task of this lab is to build a fat-tree network in Mininet. 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 `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. Please use only letters/numbers (e.g., `s1`) and avoid special characters (‘ `_` ’, ‘ `.` ’,‘ `/` ’) in the switch names (specified with `addSwitch(name)`, without `dpid`), since the controller may have problems parsing switch names with such characters. Note that you should assign the correct IP addresses to the servers according to the original design described in the fat-tree paper. For the link properties, you can use a bandwidth of 15 Mbps and a latency of 5 ms for all links. Once you have done your implementation, you can test it by running

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

If you encounter weird outcomes, please run the following command before the above one to clean up possible previous Mininet runs.

```
sudo mn -c
```

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 since the forwarding tables of all switches are still empty 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.

3 Shortest-Path Routing: It Just Works

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 and router 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 with this template, 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 in a situation where 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 on this page.

⚠ The APIs above rely on events `EventSwitchEnter` to register the switches and links. We notice that sometimes these events are not very reliable, leading to delayed/incomplete switch/link discovery. If this happens in your case, try to allocate more resources (e.g., more CPU cores) to the VM and try again.

The above functions provide the port mappings between switches, but they do not give the switch ports that are used to connect the servers. Again, you cannot assume static mappings at the controller and you need to find a way to obtain such information at runtime.

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 you should restrict the routing to valid IP packets only.

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 purposes:

```
$ 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 can reach each other and there are 0% packet drops.

4 Two-Level Routing to the Rescue

While the shortest-path routing works, it can only utilize one out of the many paths available between any pair of servers in a fat-tree network. To improve the network efficiency, the fat-tree design also comes with a routing scheme called two-level routing, which is tailored for the fat-tree topology.

The two-level routing scheme is the default routing scheme described in the fat-tree paper (see Section 3.5, 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.

5 Performance Comparison

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 similar techniques. Here, the challenge is choosing the server pairs for testing. Think about why two-level routing is more favorable over shortest-path routing and under which scenarios you will be able to observe the difference. Upon completion, please plot your results in a reasonable fashion and bring the figure to the interview.

⚠ To ensure the reliability of the measurement results, please assign resources (CPU cores and memory) to the VM as much as you can. We recommend having at least 8 CPU cores and 8 GB of memory allocated for the VM.

6 Grading Criteria

The grading will be done based on an in-person interview-style oral examination. The detailed schedule for the interview will be announced when the submission deadline approaches. For fairness consideration, you must upload your code in a zip file. Please use the naming convention `Lab3_GroupX_LastName1_LastName2.zip` and rename the folder before you zip it. Here `X` is your group number and `LastName2` can be omitted if you are alone in the group. by the specified deadline and use the uploaded version for the interview. No interview will be scheduled for you if there is no code upload; this is a strict rule.

During the interview, you are supposed to show and explain the following:

- All servers are reachable under shortest-path routing and the flow rules on switches are appropriately set. You are supposed to explain in detail how you managed the port discoveries and ARP handling. (3 points)
- All servers are reachable under shortest-path routing and the flow rules on switches are appropriately set according to the description in the fat-tree paper. You are supposed to explain in detail how you managed the port discoveries and ARP handling. (4 points)
- Explain in detail what experiments you have designed and why you think they are appropriate. Show your experimental results figure and explain your findings. (3 points)