Advanced Networked Systems (SS25)

Lab1: Hey Switches and Routers

Maximum points:	16
Submission:	zipped source code on PANDA
Deadline:	13.05.2025 23:59
Contact:	lin.wang@upb.de

1 Introduction

In the lecture we walked you through the key concepts in networking and in particular we explained how network packets are forwarded and routed by different types of network devices including switches and routers. The goal of this lab is to get your hands dirty by implementing the basic functionalities of switches and routers in an emulated network in Mininet.

Please run `git pull` in the labs codebase to retrieve the code template for this lab. You will see a new folder `lab1` and you are supposed to work in that directory for this lab.

2 A More Realistic Network

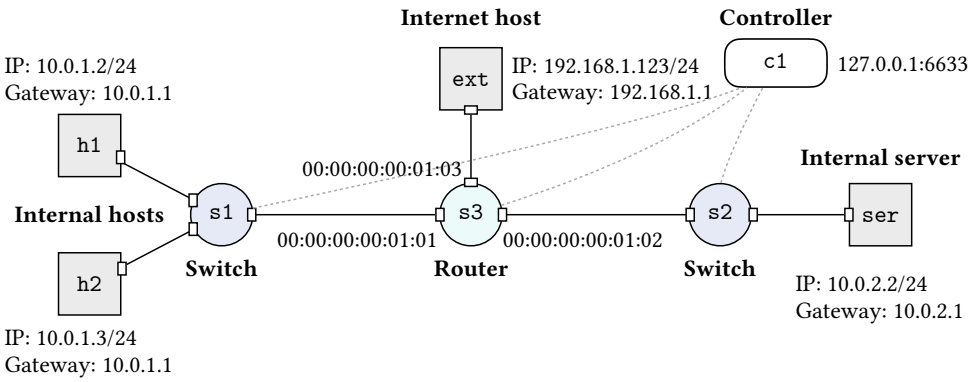

Figure 1: Network topology and configuration for this lab.

The topology for the network to implement is depicted in Figure 1. You can treat it as a typical but simplified network of a company. The network consists of two internal hosts `h1` and `h2` interconnected by a switch `s1`, an internal server `ser` connected to another switch `s2`, and a router `s3` connecting the two switched networks and an external host `ext` somewhere on the Internet. Note that the router device itself is not so different from a switch here; the real difference comes from the control logic implemented for them. The hosts are configured with IP addresses and default gateways (i.e., the router) as shown in the figure. As you can see the internal hosts, the internal server, and the external host are all on different subnets identified by their unique IP prefixes. The router is responsible for interconnecting these subnets. The MAC addresses for the three interfaces on the router are also labeled in the figure. You might not need to assign these MAC addresses explicitly to the switch/router ports, but consider them as virtual addresses (see Section 5). For all the links in the network, you can simply use the following link properties: (bandwidth: 15 Mbps, delay: 10 ms). Please implement this network in Mininet following the exercise in Lab0.

The switches and router are connected to an external controller (i.e., `c1`) which instructs them to make forwarding and routing decisions. More specifically, switches and routers maintain a forwarding table holding flow rules in the form of match-actions. An example flow rule could be to match on the destination IP address and

upon a match forward out the packet to a particular port. The controller is responsible for generating these flow rules and pushing them to the switches and router, following closely the protocols that are supposed to run on these devices (recall the lecture). This paradigm is called software-defined networking, which will be further explained in detail in an upcoming lecture. For now, staying at such a high level as above would be sufficient.

3 Fight Against Ryu

Ryu is a component-based control framework for software-defined networks. Ryu supports various protocols for managing network devices including the most popular one called OpenFlow. Throughout this lab, we will be using OpenFlow version v1.3.0 and you can check the APIs in the provided documentation. With Ryu, we can easily implement network control functionalities by writing scripts in Python.

Ryu has already been installed on the VM provided by us (via Vagrant) and you can use it out of the box. If you are using your own setup, you can easily install Ryu with the Python package manager by running

```
$ pip install ryu
```

❗ If you are running a Ubuntu version newer than 20.04, you may encounter an issue with the Ryu installed via the package manager later since some of the APIs of Python packages have changed. If that happens, you can fix this issue by installing Ryu manually using this repository. You can clone this repository and run the following commands:

```
$ cd ryu; pip install .
```

Now, let us test the Ryu controller with a simple switch function on a network. First, open a terminal and run Mininet with the following command. This starts a network emulation environment with one simple switch (of type Open vSwitch kernel switch) and three hosts. The switch will be managed by a remote controller.

```
$ sudo mn --topo single,3 --mac --controller remote --switch ovsk
```

Although the network is connected, if you run `pingall` among the hosts, you will notice that hosts are not reachable to each other. This is simply because the switch is supposed to behave as a dumb device and all its intelligence should come from the remote controller. However, we have not set up the controller yet.

To address this issue, we can open a new terminal and run a simple switch controller. This controller is a pre-implemented application of Ryu and provides the basic functionalities of an Ethernet switch. The controller handles new incoming packets at the switch and instructs the switch with flow rules created based on exactly how an Ethernet switch works.

```
$ ryu-manager ryu.app.simple_switch
```

After running the Ryu controller, you can test again the reachability between the three hosts in Mininet. Now, it should work. Meanwhile, you should be able to see `packet-in` messages at the terminal running the Ryu controller. Note that when you specify a default remote controller in Mininet (with option `--controller remote`), the controller is expected to run at `127.0.0.1:6653`. This is handled by Ryu automatically. To verify this, you can first start the Ryu controller and then start Mininet. You should be able to see the following message in the Mininet setup message:

```
Connecting to remote controller at 127.0.0.1:6653
```

If all the above went smoothly without errors, you should have a reliable setup for the tasks of this lab.

In the rest of this lab, instead of using the already implemented Ryu controller like the simple switch we just played with, you are supposed to implement your own Ryu controller for controlling the switches and router in

Figure 1. To that end, you will need to use the Ryu APIs which can be found on the Ryu documentation page. This page will be your best friend during your implementation and there is not so much you can get elsewhere except maybe a few related discussions on some online forums.

To help you get started, we have provided you with a template (i.e., `switch_router.py`) for the controller code in Python. The template already includes the basic structure of the controller function. The parts missing are the `packet_in` handler. By default, the switch/router will forward all packets for which it does not know how to handle to the controller and the function `_packet_in_handler` of the controller will be triggered consequently. That is where your custom controller logic should go. Note that you should clarify the functionalities of a switch and a router, respectively, and implement their logic separately, although they will be part of the same script and running at the same controller. For any knowledge gap in the understanding of the workings of switches and routers, please refer to the recommended textbooks on computer networks.

4 Everyone Is Learning, Including Switches

The first task is to implement a controller for the switches in Figure 1. You can easily test your switch implementation in the left part of the network (consisting of `h1`, `h2`, and `s1`). The goal is to enable the communication (e.g., `ping`) between the two hosts `h1` and `h2`.

Initially, the switch has only one flow rule installed which instructs the switch to forward all packets to the controller. The function `_packet_in_handler` will be triggered when a packet arrives at the controller generating a `PacketIn` event. With this function, we can extract a message from the event, which contains a numerical ID of the switch `datapath`, the switch port where the packet was received `in_port`, and the packet itself, among others. You can find more information about Ryu messages on this page.

Ethernet switches typically maintain a forwarding table holding match-action entries where they match on the destination MAC address and perform the action of forwarding the packet (more precisely the Ethernet frame) to a port. More importantly, Ethernet switches learn by themselves how to forward packets and populate the forwarding table automatically.

A naive approach is to rely on the controller to process all packets redirected by the switches and maintain the forwarding table at the controller. However, this approach will soon become inefficient and non-scalable since every packet will have to go through the controller which is usually quite slow. To avoid this problem, you can push the learned match-action entries (also known as flow rules) onto the forwarding table on the switch itself. The switch we use (Open vSwitch kernel switch) allows matching on a variety of packet header fields and performing actions including forwarding to a particular switch port or dropping the packet. To push a flow rule to the switch, you can simply use the `addflow()` function already provided in the controller code template.

❗ You are free to choose the format (mostly about which packet header fields to match) for the flow rules on the switch. However, you must ensure that the switch behavior is in line with that we discussed in the lecture and the number of flow rules is kept to a minimum.

Upon completion, you can test your controller with the following steps. In one terminal of the VM, you run

```
$ ryu-manager ans_controller.py
```

This will launch your controller at `127.0.0.1:6653`. In another terminal window, you launch the emulated network with the following command:

```
$ sudo python3 run_network.py
```

You can now test if the controller for the switch is working as expected or not. As mentioned, this can be done by performing a `ping` test between `h1` and `h2`, the two hosts attached directly to switch `s1`. To check the forwarding table entries that have been installed on the switch, you can run the following command in another terminal window of the VM:

```
$ sudo ovs-ofctl dump-flows <switch_id>
```

where `<switch_id>` is the name of the switch you want to check, e.g., `s1`.

While you may test your controller implementation only with `s1`, please keep in mind that your implementation should work for both switches (`s1` and `s2`) in the network. This will become important when the controller for the router is also implemented and the connectivity to the internal server `ser` must be established.

5 Routers Are Busy

The next task is to implement the controller logic for the router in the network. All the details needed for implementing router functionalities can be found in RFC 1812. The router serves as the gateway to the switched networks and all traffic going across the subnets (identified by their unique IP prefixes) will have to be routed by the router. Similar to switches, you can potentially process all incoming packets at the router with the controller, but a better way is to push flow rules to the router forwarding table so that for known flows the router can handle the forwarding itself without bothering the controller. For this lab, you are supposed to follow the latter.

Note that the router should learn the routes and other needed information by itself. In other words, you are not allowed to handcraft the flow rules for them. Nevertheless, You can assume that the router knows the IP and MAC addresses of its own ports. For example, you may make use of the following maps at your controller:

```
# Router port MACs assumed by the controller
port_to_own_mac = {
    1: "00:00:00:00:01:01",
    2: "00:00:00:00:01:02",
    3: "00:00:00:00:01:03"
}
# Router port (gateways) IP addresses assumed by the controller
port_to_own_ip = {
    1: "10.0.1.1",
    2: "10.0.2.1",
    3: "192.168.1.1"
}
```

Your router implementation must achieve the following objectives:

First, all hosts should be able to `ping` each other, except host `ext` which should not be able to `ping` any of the other (internal) hosts. This is to ensure that external hosts cannot easily discover the existence of the internal hosts. The `pingall` command should give the following result:

```
mininet> pingall
*** Ping: testing ping reachability
ext -> X X X
h1 -> X h2 ser
h2 -> X h1 ser
ser -> X h1 h2
*** Results: 50% dropped (6/12 received)
```

Second, all hosts should be able to reach each other with TCP/UDP connections, except that such connections between `ext` and `ser` are not allowed due to security reasons. You can perform a quick test for TCP connectivity with the following command:

```
mininet> iperf h1 h2
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['13.3 Mbits/sec', '16.2 Mbits/sec']
```

To thoroughly test the TCP and UDP connections with different options, you can launch `xterm` windows for the hosts and run the `iperf` client and server in these `xterm` prompts, as we already did in Lab0.

Finally, every host should be able to `ping` its own gateway, but not the gateways of others. An example is shown below:

```
mininet> h1 ping 10.0.1.1 -c1
PING 10.0.1.1 (10.0.1.1) 56(84) bytes of data.
64 bytes from 10.0.1.1: icmp_seq=1 ttl=64 time=44.9 ms

--- 10.0.1.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 43.154/43.154/43.154/0.000 ms

mininet> h1 ping 10.0.2.1 -c1
PING 10.0.2.1 (10.0.2.1) 56(84) bytes of data.

--- 10.0.2.1 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms
```

6 Grading Criteria

The grading will be done based on an in-person interview. 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 `Lab1_GroupX.zip` and *rename the folder before you zip it.* Here `X` is your group number. Your code must be uploaded on PANDA by the specified deadline and we will use the uploaded version for the interview. Any code changes made after the deadline and before the interview will not be considered. 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:

- You have constructed the network in Mininet as required by the lab. (4 points)
- Your learning switch implementation works as expected. (4 points)
- Your router implementation works as expected. (8 points)

Please keep in mind that details matter. Please stay with the protocol specifications (e.g., RFCs) as closely as possible so that your implementation can be applied not just in our small emulated network, but (hopefully) also in a large-scale, real network like the Internet.