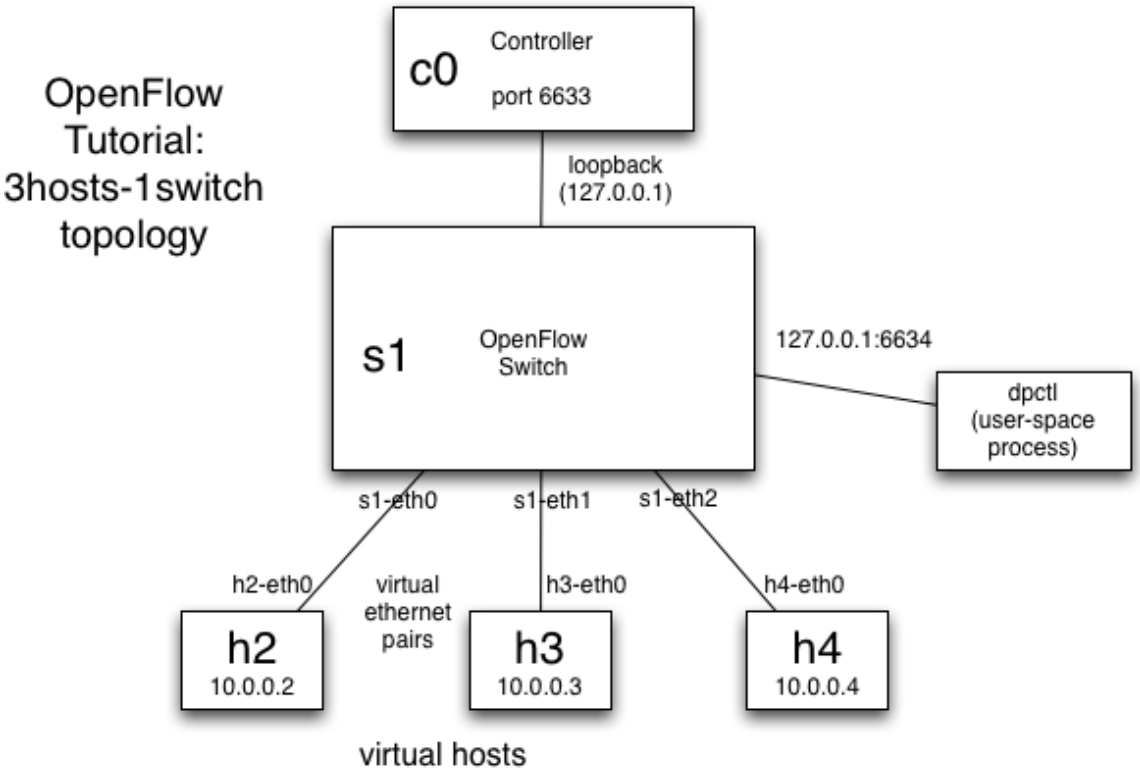
Exercise 8 – First Steps in Mininet

1. Start a Network (10P)

Let's get back to our basic OpenFlow network from the lecture:



Assume you want to realize this topology in Python. What would you probably do? Define classes and methods for hosts, controllers, switches, link these entities, implement a specific controller, etc. In sum, hundreds or thousands of lines of code. In Mininet, you can do this in one line as shown in the lecture:

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

This command creates a topology with a single switch and three hosts, sets the MAC address of the hosts to be easily readable and defines that the controller instance will be running remotely. In one line, Mininet thus created three virtual hosts (each has a separate IP address), one software switch (with three ports), connected the virtual hosts to the

software switch (note: the switch is running in the kernel) with a 'virtual Ethernet cable' and set the MAC address of each host appropriately. Consider if you had to do this on your own.

a. (10P) In the mininet console, check the ability to reach the other hosts in the network for each host h1, h2, h3. What do you observe? What is the reason for the result?
b. (Optional) Play around with some commands introduced in the lecture, for example: *help, nodes, ifconfig*, etc.
c. (Optional) Start xterm for the different devices as shown in the lecture. Play around with some commands here as well. What is the difference you observe when executing *ifconfig* at a host compared to executing the same command on the switch?

2. Flow tables (50P)

Currently, the network is not attached to a controller yet and the switch is not getting any instructions from any source. Have a look at the switches flow tables:

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

What you are doing here is using the Open vSwitch (ovs) OpenFlow (of) Control (ctl) command to dump the flow rules currently installed at s1. You will observe that there are currently no rules installed, so let's change that:

```
$ sudo ovs-ofctl add-flow <switch> <match>,<actions>
```

Will install the flow table entry <match>,<actions> into <switch>.

a. (20P) Add the following flow rules to s1:
    i. Any packet that arrives on port 1 should be forwarded via port 2
    ii. Any packet that arrives on port 2 should be forwarded via port 1

    To be able to correctly express these rules, have a look at the man-page of ovs-ofctl.

    To verify that your flows have been installed correctly, execute dump-flows for s1 again.

    (25P) Now test your network for connectivity among the hosts again. What do you observe? Please describe the OpenFlow messages exchanged within the network that have led to the current connectivity situation.

(5P) Dump the flows in s1 again. What has changed since the last dump-flows command? Why?

3. Let's analyze the network! (40P)

One popular tool to analyze networks in general is the protocol analyzer Wireshark. We can also analyze the behavior of OpenFlow with this tool. Start Wireshark by connecting to the VM with X forwarding enabled and then entering:

```
$ sudo wireshark &
```

A new window should show up. In Wireshark, we can now dissect our traffic. What we want to do is to get only the OpenFlow traffic to show up. In Wireshark, go to "Capture->Interfaces" and select the "lo" interface. This will cause Wireshark to capture the traffic that it sees on the loopback interface.

a. (10P) Why do we need to look at the loopback interface, rather than checking eth0 or eth1 for packets?

What we need to do next is to tell Wireshark that we only want to see the OpenFlow related traffic. For this, Wireshark offers the "filter" functionality. In the main Wireshark window, you should see a "filter" field. Into that field, simply enter:

```
of
```

To see some action, we will now start a controller. In your SSH console, enter:

```
$ controller ptcp:
```

What this does is to start a controller that acts as a simple learning switch (remember the Computer Networks lecture?). Now you should see some packets being passed through Wireshark. You can select single packets by clicking on them. Try this with, for instance, the **of_features_reply** packet. If you click on it, you will see some information about the packet in the section below. Here, you can peak into the packet headers (Ethernet, IP, TCP) and OpenFlow message contents.

b. (10P) Click on "OpenFlow" and have a look at the information you get. Given the information that a controller asks a switch for available ports with an **of_features_request** message, describe what you see in the **of_features_reply** packet.

One of the nice features of Mininet is that we can simply setup a new topology any time. To do that properly, we first have to shut down Mininet and clean-up the environment (e.g., remove external controllers like the one we started above). Stop the controller you have just started. Afterwards, proceed by:

```
mininet> exit
$ sudo mn -c
```

Now, we can start a new network. Start the most simple topology (two hosts, one switch, one OF reference controller) with:

```
$ sudo mn
```

Note that this time, in contrast to our example before, the controller runs inside the VM and is not located remotely. Our controller now is the OF reference controller. We now want to see how this controller automatically configures the switch.

Switch to Wireshark. You are probably seeing many echo-request/reply messages (do you have a guess what these are for?). We want to filter out these messages to be able to focus on the things that really happen in the network. For that, in the filter field of Wireshark, replace the "of" with:

```
of and not (of10.echo_request.type or of10.echo_reply.type)
```

In some (outdated) versions of Wireshark you may have to use the following instead:

```
of && (of.type != 3) && (of.type != 2)
```

c. (20P) Ping host h2 from host h1 at least three times (ping –c3). Have a close look at the latency of the ping command. What do you observe? Explain this phenomenon with the help of Wireshark.

4. Play around with some topologies! (50P)

Here are some more commands for creating a topology. Play around with them, do not forget to exit and cleanup mininet after each experiment:

```
$ sudo mn --topo linear,4
```

a. (10P) Which topology does this command create?
b. (20P) Start a linear topology with 256 switches. What can you observe when you try to ping host h2 from host h1, and then host h255 from host h1? Can you explain what has happened?

Try this topology:

```
$ sudo mn --link tc,delay=10ms
```

c. (20P) What do you observe when you ping host h2 from host h1 now? Explain your observations.