TECHNISCHE UNIVERSITÄT BERLIN
Fakultät IV – Elektrotechnik und Informatik
FG INET
Prof. Dr. Thomas Zinner
Theresa Enghardt, Apoorv Shukla, Damien Foucard, Thomas Krenc

Routerlab SoSe 2018
Worksheet 10: OpenFlow, Mininet, and Software Defined Networking

Today we are leaving the safe realm of established network standards and looking at a new standard of tomorrow called *OpenFlow*. We will be investigating the basic concepts behind OpenFlow, using the Mininet network emulation tool. Using mininet, you will create network topologies, deploy a software controller, and optionally, the brave ones can try building their own simple OpenFlow controller. Finally, you will watch a video from Professor Scott Shenker about the motivations behind OpenFlow and see how OpenFlow enables the network to become "Software Defined."

OpenFlow Primer

Info 10.1

We have been working quite a bit with the managed switches in our Routerlab from Cisco, and configuring, e.g., ACLs and VLANs on them. Internally, those switches consist of two core parts: An embedded processor running an operating system (Cisco IOS), for configuration and managing a forwarding table, and a specialised ASIC chip providing the *forwarding path* that actually forwards the packets from one port to another, based on some internal *forwarding table*. These two parts are tightly coupled together in a *closed box*, i.e., you cannot buy one without the other, and you can configure it, but not program it according to your wishes. If you want something that is not supported by the firmware (e.g., reflexive ACLs :)), you are very much out of luck.

Some vendors are living very very well on this tight coupling between the hardware forwarding (switch box) and intelligence (firmware), but many users, for instance Telcos, and large datacenter owners, are unhappy with it. Enter OpenFlow! OpenFlow is a refactoring of the classical switch architecture to separate the mechanism of forwarding from the process of controlling the content of the forwarding table. Instead of an internal embedded processor in a closed OS, an OpenFlow switch exposes a forwarding table called the *flow table* via a standardized protocol to an external software *controller*, e.g., a piece of *software* running on a standard PC server. The controller makes the "intelligent" forwarding decisions, while the switch hardware performs the mechanism of forwarding the packets in hardware and at line rate.

OpenFlow is currently under active development and adoption. There's a mixed academy/industrial consortium which led to its development, originating at Stanford university. Today, the Open Networking Foundation, of which which Deutsche Telekom is a founding member, is in charge of defining the future direction of the evolution of the OpenFlow protocol.

Question 1: (10 Points) *OpenFlow intro*

Familiarize yourself with OpenFlow 1.0. Have a look at the introduction and white-paper at http://www.opennetworking.org/wp-content/uploads/2013/04/openflow-spec-v1.0.0.pdf. Then answer the following questions.

(a) What is the function of the OpenFlow controller? When does it communicate with a switch?

(b) Describe the difference between the data plane and the control plane in an Open Flow network? What traffic goes over the data plane? What traffic goes over the control plane?

(c) What is a flow table, what purpose does it serve? When does it get modified, and what modifies it?

(d) Against which parts of an Ethernet frame can a flow table entry match?

(e) What actions can be taken for matching frames? Explain 3.

(f) Can you think of practical situations where OpenFlow is useful? Which real-life practical problems based on your previous worksheets can you solve with it (that are difficult to solve otherwise)? Explain 2!

(g) What new problems might OpenFlow introduce into networks? Do you see any limitations? Pick 2 and explain.

(h) Could you build an IPv4 router using an OpenFlow v1.0 switch and an appropriate controller? What required mechanisms from the IP router would be missing? Hint: Look at the supported flow-table actions for OF v1.0
http://www.opennetworking.org/wp-content/uploads/2013/04/openflow-spec-v1.0.0.pdf

Question 2: (5 Points) *Mininet intro*

Familiarize yourself with Mininet. Have a look at the documentation at http://www.mininet.org/. Then answer the following questions.

(a) What is the biggest architectural difference between Mininet and a network simulator such as NS3?

(b) Upon which mechanisms in the Linux Kernel does Mininet rely to virtualize a network? Name at least two and explain what function each serves.

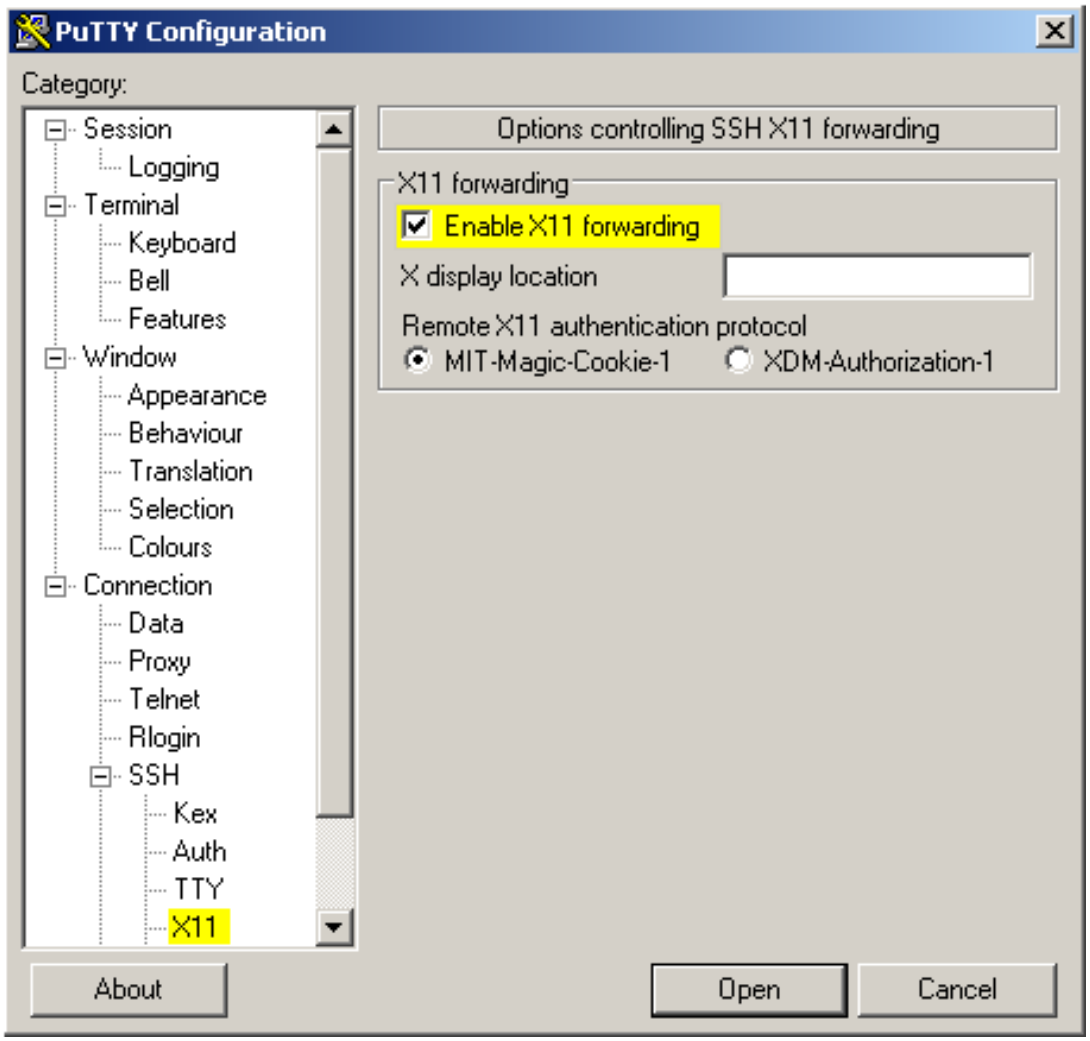
(c) What is the relationship between Mininet and OpenFlow? What (hint: two) parts of a Mininet system actually implement the OpenFlow protocol?

(d) Which version(s) of OpenFlow can Mininet support?

(e) How would you interface a system running Mininet with a hardware OpenFlow switch?
Look at Download/Get Started With Mininet http://mininet.org/download/ and follow the instruction there using option 1.
(If you find your mouse stuck inside the virtualbox window, press the lower right ctrl key on your keyboard once and you will have control of your mouse and be able to leave the window).
Once you have the system up and running, log into the VM, the default username and password are both mininet. We highly recommend doing the Mininet Walkthrough http://mininet.org/walkthrough/ (also found in step 5 of the above option 1 instructions).
Note that if you plan to work with the mininet environment in the future to experiment with writing or modifying a Software Defined network controller in the future you will want to be able to connect into your virtual machine from your local machine so you can move files, run X11, etc. If you do not ever plan on going further, you MAY be able to skip getting multiple interfaces to work and only use the mininet command prompt to complete the remainder of this lab. If you just want to accomplish that skip down to Question 3. However, if you want to "play in the sandbox" with the SDN research community, you will want to make sure you set up the two interfaces by following the directions at VirtualBox: Setting up a Host-Only Network https://github.com/mininet/openflow-tutorial/wiki/VirtualBox-specific-Instructions where one of the steps will be to edit /etc/network/interfaces to include: auto eth0 iface eth0 inet dhcp auto eth1 iface eth1 inet dhcp
(You will not need this link if you get the interfaces to work but in case you have trouble here is more help http://www.brianlinkletter.com/set-up-mininet/)

For Macs you will need to do your own research on how to make this work on a Mac. To be able to ssh from your laptop to your running VM and use X11 on a computer running Windows, you need to download and install the required SSH and X-server software (recommend PuTTY and XMing).

Windows OS Instructions: In order to use X11 applications such as xterm and wireshark, the Xming server must be running, and you must make an ssh connection with X11 forwarding enabled. First, start Xming (e.g. by double-clicking its icon.) No window will appear, but if you wish you can verify that it is running by looking for its process in Windows' task manger. Second, make an ssh connection with X11 forwarding enabled. If you start up puTTY as a GUI application, you can connect by entering your VM's IP address (that will be the 192.168.56.101 or similar address on eth0 which is your NAT interface not the host only interface) and enabling X11 forwarding. To enable X11 forwarding from puTTY's GUI, click puTTY;then Connection;then SSH;then X11, then click on Forwarding;then "Enable X11 Forwarding", as shown below:



Question 3: (30 Points) *Running Mininet*

After installing and getting set up:

(a) Lets begin by creating the simplest mininet topology. From your Putty window connected to your VM, give the command sudo mn (Note, sudo would not be required if you were running as root). Describe what happens. Draw the resulting topology and include the ip addresses assigned to each of the virtual hosts.

(b) What tcp:ip port is the OpenFlow controller and OpenVSwitch instance using? (hint: use "dump" command at the mininet prompt or "c0 netstat" so as to run `netstat` on the controller c0). Which network interfaces do you see from h1's namespace? (hint "h1 ifconfig")

(c) From the Mininet prompt, execute the command sh ovs-ofctl dump-flows s1 Explain what this command does. What do you see, and what does it mean? Now send a ping from h1 to h2. Within 15 seconds of sending the ping, execute the dump-flows command again. What do you see now? Explain in detail.

(d) Also have a look at the flow statistics. What kind of stats are gathered?

(e) Exit Mininet and start a new Mininet topology using the command `sudo mn --co none` Send a ping from h1 to h2. What happens and why? What needs to happen before pings will be forwarded again? When finished exit mininet.

Question 4: (15 Points) Optional Bonus: Dissecting the OpenFlow protocol with tshark

Now let's take a closer look at what's going on in the open flow control channel. We'll take the practical network engineer approach and capture the packets and then try and make some sense out of them. After doing sudo mn Fire up tcpdump on the loop back interface of the control plane c0, and have it capture all traffic where the src or dst port is 6633 and write out the entire packets (including payloads) to some file. Send a single ping from h1 to h2. Exit tcpdump. tcpdump should report that some packets have been captured. You will want to use the option -s 0 to make sure tcpdump captures entire packets, not just headers.

(a) Use `tshark` with the OpenFlow dissector available in the mininet VM to read and analyze the dumpfile. You should see the individual OF protocol messages now in clear text. Provide us with an excerpt of these messages from tshark. Explain the individual messages that you see (you will want to reference the OpenFlow spec http://www.opennetworking.org/wp-content/uploads/2013/04/openflow-spec-v1.0.0.pdf). Specially, why do some messages have to be flooded to all ports?

Question 5: (20 Points) The Bigger Picture: From OpenFlow to Software Defined Networking.

So far, you've become familiar with OpenFlow, however this protocol is just a small architectural component of a much bigger picture: Software Defined Networking (SDN). Grab some popcorn, and head over to this (1 hour long) video lecture, "An attempt to motivate and clarify Software Defined Networking" from Professor Scott Shenker. https://www.youtube.com/watch?v=WVs7Pc99S7w. While you watch this video, keep in mind, and then answer the following questions:

(a) What is the relationship between OpenFlow and SDN? What role specifically does OpenFlow play?

(b) What are at least two motivating factors behind SDN?

(c) Keeping in mind all the previous worksheets you have completed, what do you expect are the benefits of SDN with respect to the tasks that you carried out? Please elaborate on at least 3 different benefits.

(d) Around minute 39 of the talk, Professor Shenker mentions that "SDN isn't just a better mechanism, but rather it is an instantiation of fundamental abstractions." Explain in detail what he means by this and give an example of one of the key abstractions that SDN builds upon.

Question 6: (15 Points) Important Question.

This part of the worksheet is very challenging and your group may choose not to attempt it. It is intended for advanced students who want a challenge. Starting with the POX learning switch module, implement a controller for the default single- switch, two-host mininet topology. The requirements of the controller are as follows: The controller must forward all pings just as the default learning switch module does. In addition, the controller must allow http requests from h1 to h2, but not in the other direction. It must also allow ssh connections to be established from h2 to h1, but not in the other direction. Once you have built your controller, run it with the default mininet topology. Demonstrate that your controller meets our requirements by performing pings, http and ssh connections between h1 and h2. Finally, provide us with the source code of your controller.

You can use any existing module to build your own controller, for example, the "simple" **l2_learning.py** module already in the mininet virtual machine (which you will install) in the /pox/pox/forwarding directory. To get started you will want to look at the document https://noxrepo.github.io/pox-doc/html/ For some additional information on forwarding.l2_learning.py the following may be helpful:
http://blog.pythonicneteng.com/2013/02/openflow-tutorial-with-pox.html

Here is a warm up exercise you may find helpful to get started on Question 6: Overview: The network you'll use in this warm up exercise includes 3 hosts and a switch with an OpenFlow controller (POX):

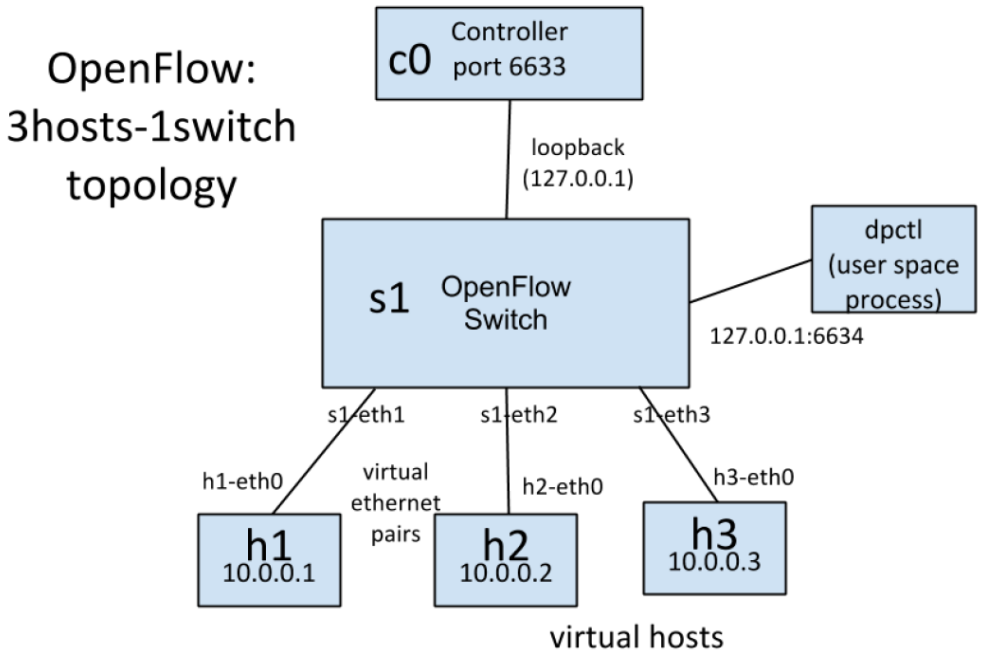


Figure 1: Topology for the Network under Test

POX is a Python based SDN controller platform geared towards research and education. For more details on POX, see About POX or POX Documentation on NOXRepo.org. We are not going to be using the reference controller anymore, which is the default controller that Mininet uses during it simulation. If you are using windows:
Start Xming (you will not see anything)
use putty to open an ssh session with your vm (login mininet and password mininet). In this ssh window you will run the controller

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

The POX controller comes pre-installed with the provided VM image. Now, run the basic hub example:
use putty to open a second ssh window to your vm. In this window type:

```
/pox/pox.py log.level --DEBUG forwarding.hub
```

This tells POX to enable verbose logging and to start the hub component. The switches may take a little bit of time to connect. When an OpenFlow switch loses its connection to a controller, it will generally increase the period between which it attempts to contact the controller,

up to a maximum of 15 seconds. Since the OpenFlow switch has not connected yet, this delay may be anything between 0 and 15 seconds. If this is too long to wait, the switch can be configured to wait no more than N seconds using the –max-backoff parameter. Alternately, you exit Mininet to remove the switch(es), start the controller, and then start Mininet to immediately connect. Wait until the application indicates that the OpenFlow switch has connected. When the switch connects, POX will print something like this: INFO:openflow.of_01:[Con 1/1] Connected to 00-00-00-00-00-01
DEBUG:samples.of_tutorial:Controlling [Con 1/1]

Verify Hub behavior with tcpdump

Now verify that hosts can ping each other, and that all hosts see the exact same traffic - the behavior of a hub. To do this, we'll create xterms for each host and view the traffic in each. In the
Mininet console, start up three xterms: `mininet> xterm h1 h2 h3`
Arrange each xterm so that they're all on the screen at once.
This may require reducing the height to fit a cramped laptop screen. In the xterms for h2 and h3, run tcpdump, a utility to print the packets seen by a host:
`# tcpdump -XX -n -i h2-eth0`
and respectively:
`# tcpdump -XX -n -i h3-eth0`
In the xterm for h1, send a ping:
`# ping -c 1 10.0.0.2`
The ping packets are now going up to the controller, which then floods them out all interfaces except the sending one. You should see identical ARP and ICMP packets corresponding to the ping in both xterms running tcpdump. This is how a hub works; it sends all packets to every port on the network. Now, see what happens when a non-existent host doesn't reply.
From h1 xterm:
`# ping -c 1 10.0.0.5`

You should see three unanswered ARP requests in the tcpdump xterms. If your code is off later, three unanswered ARP requests is a signal that you might be accidentally dropping packets. You can close the xterms now.
Now, let's look at the hub code:

```
from pox.core import core
import pox.openflow.libopenflow_01 as of
from pox.lib.util import dpidToStr
log = core.getLogger()
def _handle_ConnectionUp (event):
  msg = of.ofp_flow_mod()
  msg.actions.append(of.ofp_action_output(port = of.OFPP_FLOOD))
  event.connection.send(msg)
  log.info("Hubifying %s", dpidToStr(event.dpid))
def launch ():
  core.openflow.addListenerByName("ConnectionUp", _handle_ConnectionUp)
  log.info("Hub running.")
```

Table 1. Hub Controller

Useful API of POX :
connection.send(...) function sends an OpenFlow message to a switch.
When a connection to a switch starts, a ConnectionUp event is fired. The above code invokes a_handle_ConnectionUp () function that implements the hub logic.
ofp_action_output class This is an action for use with ofp_packet_out and ofp_flow_mod. It specifies a switch port that you wish to send the packet out of. It can also take various "special" port numbers.
An example of this, as shown in Table 1, would be OFPP_FLOOD which sends the packet out all ports except the one the packet originally arrived on.
Example. Create an output action that would send packets to all ports:

out_action = of.ofp_action_output(port = of.OFPP_FLOOD)
ofp_match class (not used in the code above but might)
Objects of this class describe packet header fields and an input port to match on. All fields are optional – items that are not specified are "wildcards" and will match on anything. Some notable fields of ofp_match objects are:
dl_src - The data link layer (MAC) source address
dl_dst - The data link layer (MAC) destination address
in_port - The packet input switch port
Example. Create a match that matches packets arriving on port 3: match = of.ofp_match()
match.in_port = 3
ofp_packet_out OpenFlow message (not used in the code above but might be useful)
The ofp_packet_out message instructs a switch to send a packet. The packet might be one constructed at the controller, or it might be one that the switch received, buffered, and forwarded to the controller (and is now referenced by a buffer_id). Notable fields are:
buffer_id - The buffer_id of a buffer you wish to send. Do not set if you are sending a constructed packet.
data - Raw bytes you wish the switch to send. Do not set if you are sending a buffered packet.
actions - A list of actions to apply (for this tutorial, this is just a single ofp_action_output action).
in_port - The port number this packet initially arrived on if you are sending by buffer_id, otherwise OFPP_NONE.
Example. send_packet() method:

```
  If buffer_id is a valid buffer on the switch, use that.  Otherwise,
  send the raw data in raw_data.
  The "in_port" is the port number that packet arrived on.  Use
  OFPP_NONE if you're generating this packet.
  """
msg = of.ofp_packet_out()
msg.in_port = in_port
if buffer_id != -1 and buffer_id is not None:
  # We got a buffer ID from the switch; use that
  msg.buffer_id = buffer_id
else:
  # No buffer ID from switch -- we got the raw data
  if raw_data is None:
    # No raw_data specified -- nothing to send!
    return
  msg.data = raw_data

action = of.ofp_action_output(port = out_port)
msg.actions.append(action)

# Send message to switch
self.connection.send(msg)
```

Table 2: Send Packet

ofp_flow_mod OpenFlow message

This instructs a switch to install a flow table entry. Flow table entries match some fields of incoming packets, and executes some list of actions on matching packets. The actions are the same as for ofp_packet_out, mentioned above (and, again, for the tutorial all you need is the simple ofp_action_output action). The match is described by an ofp_match object. Notable fields are:
idle_timeout - Number of idle seconds before the flow entry is removed. Defaults to no idle timeout.
hard_timeout - Number of seconds before the flow entry is removed. Defaults to no timeout.
actions - A list of actions to perform on matching packets (e.g., ofp_action_output)

priority - When using non-exact (wildcarded) matches, this specifies the priority for overlapping matches. Higher values are higher priority. Not important for exact or non-overlapping entries.
buffer_id - The buffer_id of a buffer to apply the actions to immediately. Leave unspecified for none.
in_port - If using a buffer_id, this is the associated input port.
match - An ofp_match object. By default, this matches everything, so you should probably set some of its fields! Example. Create a flow_mod that sends packets from port 3 out of port 4.
fm = of.ofp_flow_mod()
fm.match.in_port = 3
fm.actions.append(of.ofp_action_output(port = 4))

Verify Switch behavior with tcpdump

This time, let us verify that hosts can ping each other when the controller is behaving like a Layer 2 learning switch. Kill the POX controller by pressing Ctrl-C in the window running the controller program and run the l2_learning example:
`$ /pox/pox.py log.level --DEBUG forwarding.l2_learning`

At this point the mininet network is assumed still running in the other ssh window, if not after starting the controller start the network again with:
`$ sudo mn --topo single,3 --mac --switch ovsk --controller remote`

Like before, we will create xterms for each host and view the traffic in each. In the Mininet console, start up three xterms:
mininet¿ xterm h1 h2 h3
Arrange each xterm so that they are all on the screen at once.
This may require reducing the height of to fit a cramped laptop screen. In the xterms for h2 and h3, run tcpdump, a utility to print the packets seen by a host:
`# tcpdump -XX -n -i h2-eth0` and respectively:
`# tcpdump -XX -n -i h3-eth0` In the xterm for h1, send a ping:
`# ping -c 1 10.0.0.2`
Here, the switch examines each packet and learn the source-port mapping. Thereafter, the source MAC address will be associated with the port. If the destination of the packet is already associated with some port, the packet will be sent to the given port, else it will be flooded on all ports of the switch. You can close the xterms now. The code for l2_learning application is provided under /pox/pox/forwarding and and may also be found at
https://github.com/CPqD/RouteFlow/blob/master/pox/pox/forwarding/l2_learning.py
You will want to reference
https://noxrepo.github.io/pox-doc/html/#openflow-in-pox to better understand pox code and the statements you will need to use to implement your own modified controller.

SUMMARY:

So in summary to run the existing forwarding.l2_learning controller you first start it in an ssh window connected to your virtual machine with the command:
`sudo /pox/pox.py log.level --DEBUG forwarding.l2_learning`

Next to create a topology in another ssh window connected to your virtual machine with just two hosts: `sudo mn --topo single,2 --mac --switch ovsk --controller remote`

One approach is to use the l2_learning.py file as a starting point and modify it to implement the controller specified in this last part of the worksheet assignment.

Note:

To make sure that you are starting clean each time you change something you should use the following process to stop the controller and to stop the mininet network:

Make sure that a controller is not running in the background (usually a contol C in the window will stop it clean but just in case):
From the ssh window your controller is running in:
`$ ps -A | grep controller`

If so, you should kill it either press Ctrl-C in the window running the controller program, or from the other SSH window:
`$ sudo killall controller`

To shutdown Mininet and to make sure that everything is clean and using the faster kernel switch (this you may want to do often):
From you Mininet console:
`mininet> exit`
`$ sudo mn -c`

Question 7: (5 Points) And finally...

We would like to know how you liked this lab course. Was it fun? What should be improved next time? Which worksheet was particularly hard or way too easy? Of course, we will not grade the content of your comments. We will hand out the feedback form at the BGP Debriefing session next week, you have to return feedback forms on OpenFlow debriefing and claim your 5 points. The evaluation will then be carried out anonymously by a blind deaf monkey sitting in a black box locked away on a distant island... or something similar. We definitely will not hold your comments against you :-)

Thanks for participating in the routerlab course and have a nice semester break!

Submission details (more in ISIS):
Please submit an archive (.tar.gz or .zip) containing a *directory*, which contains all files you want to submit. Please have *your group number* in the file name and the directory name.
A report (one single PDF file, named *worksheet(num)-group(num).pdf*) containing the following elements is mandatory:

- Your group number on the first page
- Topology map with relevant routers, switches, *loadgens*, and interfaces, IPs and subnet masks (CIDR).
- For each question, the written answers with the **relevant** portions of output from all commands such as ping, tcpdump, etc in a text format. **No** screenshots of terminal windows are accepted. For ping 3-4 lines of ping requests are usually sufficient.
- For each question all commands needed to configure the *loadgens*.
- For each question all **changed parts** in the configuration of routers and switches (differences to the default config).
- **Never** include the full verbatim switch or router configuration in the pdf report.
- For all questions, state your assumptions, say what you did, describe what you observed, explain your conclusions.

Additionally, please include your config files in the archive.
For each question, please provide the full switch and router configuration in a separate text file named after the device and question, e.g.: *q01-config-sc1.txt*. This makes it easier for us to reproduce your configuration and understand what you did.
We can only grade what we find in your submission and what we understand. Please state your assumptions and observations as clearly as possible.
Due Date: Thursday, July 12, 23:55 PM