LUNDS UNIVERSITET
Lunds Tekniska Högskola

Electrical and Information Technology, LTH
Professor Bjorn Landfeldt
Chair of Network Architecture and Services

SDN project part I

This is the first part of a project in network configuration and testing using Software Defined Networks (SDN). All questions in this description have to be answered satisfactorily in order to pass this part.

Report

The questions below should be answered in clear and concise English, illustrations need to be clear and the report submitted in pdf format by each group. Font used should be 11 pt Times New Roman. After you have finished the tasks in this part, turn your attention to part II and append your answers from that part to this report.

Aim

This part will first familiarise you with the Mininet emulator and prepare you to start making actual experiments in later parts. Second, it will introduce you to the world of SDN as it is implemented in Linux and the fundamental virtualisation support that allows virtualisation in for example data centres to use the underlying network virtualisation to manage communication to and from different virtual services on servers. The idea is to familiarise yourself with the thinking, planning and execution involved in configuring and managing such environments. For the interested, we recommend to read up on Openstack to complete the picture of cloud based service provisioning, but it is not covered in this course.

Execution

The project is designed to be carried out on the student computers available in the E-building. If you want to work on your own computers you will have to prepare them to use ssh and x-windows remotely yourselves, no support will be given for such setups. You can also run the lab on your own Linux computer if you have the necessary software installed and kernel support. Finally, you can install Linux in a virtual machine on your own computer first and then install mininet, but you may have to make adjustments to the lab to cater for the fact that you start in a virtual environment.

Linux

If you are not familiar with Linux (heaven forbid), there are numerous tutorials and other sources of information on the web. The main point to understand is that you will work with the command line interface using the bash shell. There are many sources explaining how different shell commands work for those who seek. An introduction is available from: http://www.hep.lu.se/courses/MNXB01/2017/MNXB01-tutorial-1b.pdf. Many of the commands you will use require superuser privileges, i.e. you have more rights than ordinary users. We recommend that you use the command sudo to temporarily become superuser while executing commands and only become superuser permanently when executing a long series of commands in order to avoid mixing up user rights for files you create or save. In order to become superuser permanently, use “sudo su” and your passphrase.

The username on the lab computer is: “student”, and the passphrase: “cloudnetworking”.

Address: Box 118, 221 00 Lund *Visiting Address:* Ole Römers Väg 3 *Phone:* +46 46-222 44 07
Email: bjorn.landfeldt@eit.lth.se

Mininet

Mininet is a software package that emulates nodes and switches on a network. For more information about Mininet, see: mininet.org.

There is documentation and a walkthrough on which parts of this project part is based, see: http://mininet.org/walkthrough/

Mininet takes advantage of inbuilt support in the Linux kernel for virtual switches and therefore also gives insight into the world of open SDN platforms. It is entirely possible to build powerful networking equipment on off the shelf hardware combined with Linux.

Notation

In the remainder of this document we use the following prefixes for commands:

> is used to indicate the prompt in a bash shell (Linux)

`mininet>` is used to indicate the command line inside Mininet

Part 1, Basic mininet commands

After logging into the server, start mininet by typing:

`> sudo mn -c` (enter the password for the computer)

This will clean up the mininet framework and reset it to the initial state. Start mininet again but without cleaning it up by:

```
> sudo mn
```

You should get a mininet prompt as follows: **mininet>**

Mininet is now initiated with a minimum configuration consisting of two hosts (computers) which are attached to a switch (layer 2 device) and a SDN controller. To list the nodes type:

```
mininet> nodes
```

To list the topology type:

```
mininet> net
```

The mininet syntax is as follows. If the initial string in a command is a node, switch or controller, the following command is executed on that entity, e.g.

`mininet> h1 ifconfig -a` (which lists all network interfaces on node h1)

Question 1: Draw the Mininet minimal topology and include the IP and MAC addresses for hosts and the switch. To which switch ports are the hosts and controller connected?

Test connectivity between nodes

Use ping to test if a host can be reached from another host, e.g.:

```
mininet> h1 ping -c 5 h2
```

This makes node h1 ping node h2 5 times.

Question 2: Record the round trip times for each ping. What are they?

Question 3: Why is the ping time for the first packet much larger than the following packets?

You can test the connectivity between all nodes using:

```
mininet> pingall
```

If you want to run commands on a specific node you start an xterm on that node, e.g.

```
mininet> xterm h1
```

Python interpreter

If the first string in a command is py, mininet will interpret the remainder as Python input. For example:

```
mininet> py h1.IP()
```

will return the IP address of host h1 using the inbuilt Python classes in mininet. Python is used to manipulate objects and to construct network scenarios in mininet.

Exiting

In order to exit mininet, simply type:

```
mininet> exit
```

Don't forget to clean up after each run using > `sudo mn -c.`

Part 2, custom topologies

In this part you will build a custom network using the Python API. You will program your Python code in a text editor and save the file in the home directory on the mininet server. You can use the inbuilt ubuntu text editor via xwindows by typing:

```
> gedit &
```

The Python code is implemented as a class called firstTopo which extends the Topo class provided with the mininet API. Save the class implementation in a file called firsttopo.py. This is the code for the firstTopo class:

```
from mininet.topo import Topo

class FirstTopo( Topo ):
     "Simple topology example."
     def __init__( self ):
         "Create custom topo."
         # Initialize topology
         Topo.__init__( self )
         # Add hosts and switches
         h1 = self.addHost( 'h1' )
         h2 = self.addHost( 'h2' )
         h3 = self.addHost( 'h3' )
         h4 = self.addHost( 'h4' )
         leftSwitch = self.addSwitch( 's1' )
         rightSwitch = self.addSwitch( 's2' )
         # Add links
         self.addLink( h1, leftSwitch )
         self.addLink( h2, leftSwitch )
         self.addLink( leftSwitch, rightSwitch )
         self.addLink( rightSwitch, h3 )
         self.addLink( rightSwitch, h4 )
```

```
topos ={ 'firsttopo': ( lambda: FirstTopo() ) }
```

Question 4: Draw the network and label the IP and MAC addresses for the hosts. Also label the interface names and MAC addresses of the switch ports that are connected to the hosts.

Question 5: Now extend the topology by adding a switch called midSwitch between rightSwitch and leftSwitch, connect it to the two other switches and add two new hosts to it called ‘h5’ and ‘h6’. Call the new class “secondTopo”. Repeat Question 4 for secondTopo.

Next, you will run an automated test run of a simple single switch topology using the following Python code:

```
#!/usr/bin/python

from mininet.topo import Topo
from mininet.net import Mininet
from mininet.util import dumpNodeConnections
from mininet.log import setLogLevel

class SingleSwitchTopo(Topo):
    "Single switch connected to n hosts."
    def build(self, n=2):
        switch = self.addSwitch('s1')
        # Python's range(N) generates 0..N-1
        for h in range(n):
            host = self.addHost('h%s' % (h + 1))
            self.addLink(host, switch)

def simpleTest():
    "Create and test a simple network"
    topo = SingleSwitchTopo(n=4)
    net = Mininet(topo)
    net.start()
    print "Dumping host connections"
    dumpNodeConnections(net.hosts)
    print "Testing network connectivity"
    net.pingAll()
    net.stop()

if __name__ == '__main__':
```

```
    # Tell mininet to print useful information
    setLogLevel('info')
    simpleTest()
```

Save the code to a file called simplest.py and run it using the following command:

```
> sudo python simplest.py
```

Question 6: Draw the resulting network.

Question 7: You will now change the code to define performance parameters and run performance tests on the network. In the code above, substitute for the following:

```
# Each host gets 50% of system CPU
host = self.addHost( 'h%s' % (h + 1), cpu=.5/n )
# 10 Mbps, 5ms delay, 2% loss, 1000 packet queue
self.addLink( host, switch, bw=10, delay='5ms', loss=2,
                            max_queue_size=1000, use_htb=True )
```

Also, add the following imports to the beginning of the script:

```
from mininet.node import CPULimitedHost
from mininet.link import TCLink
```

Finally, substitute the simpleTest function with function perfTest():

```
def perfTest():
    "Create network and run simple performance test"
    topo = SingleSwitchTopo( n=4 )
    net = Mininet( topo=topo,
                host=CPULimitedHost, link=TCLink )
    net.start()
    print "Dumping host connections"
    dumpNodeConnections( net.hosts )
    print "Testing network connectivity"
    net.pingAll()
    print "Testing bandwidth between h1 and h4"
    h1, h4 = net.get( 'h1', 'h4' )
    net.iperf( (h1, h4) )
    net.stop()
```

and finally:

```
if __name__ == '__main__':
    setLogLevel( 'info' )
    perfTest()
```

Save the code in a file called perftest.py and test that it runs properly. Now, change the bandwidth on the links to 100 Mbps (bw = 100) and test the throughput. Is the result the expected? Explain why/why not.

Note, perf can be used to generate background traffic in a network by setting the transmission rate and the duration for the test flow as follows:

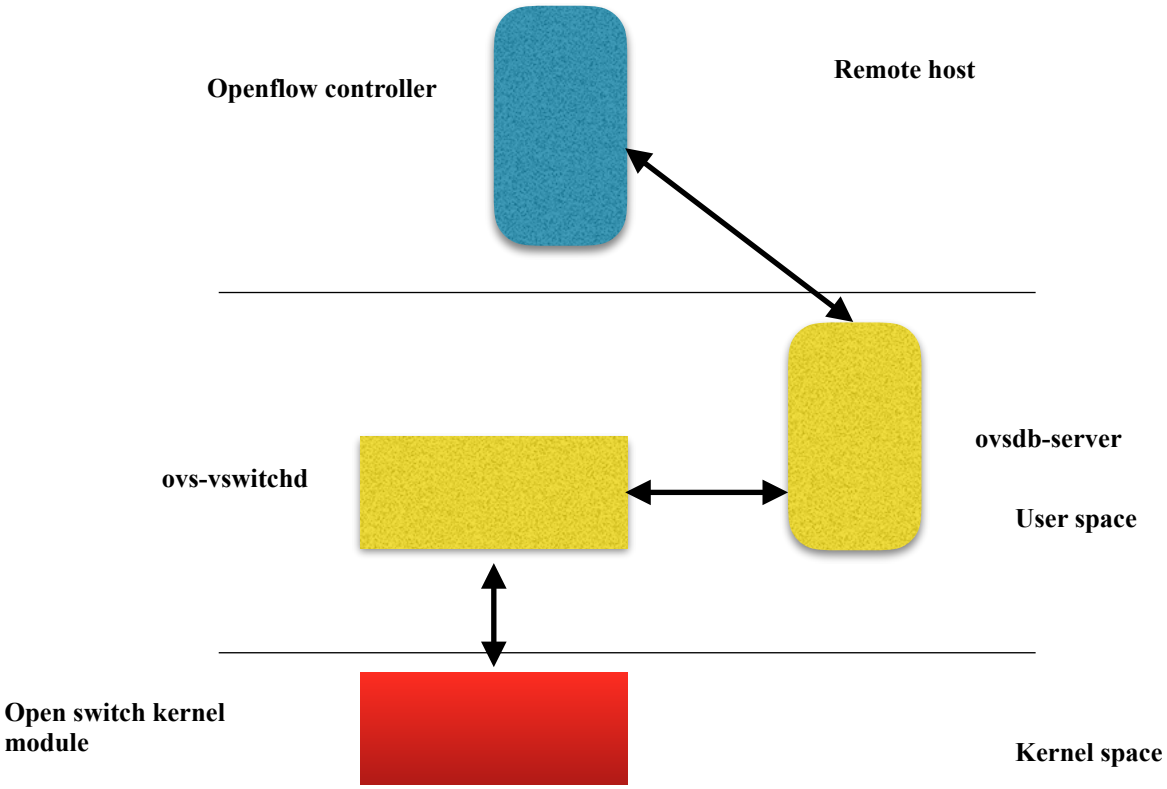
```
net.iperf( (h1, h4),  l4Type='UDP', udpBw='10M', seconds=100 )
```

Part III, managing the Linux virtual switch with Openflow

As stated earlier, mininet makes use of the inbuilt SDN support in Linux and since this is a platform that is used to implement SDN functionality in many actual production networks we will look a bit closer at this underlying technology.

In this part, you will familiarise yourself with the inbuilt Linux support for SDN using Openflow, which is an open protocol for communication between the controller and switch (data plane). Openflow is used to manage and control switches and their flow tables. You will use the ovs-ofctl management utility in Linux to manually configure flow tables.

The different components in the Linux SDN platform are illustrated below.



The kernel module is inspecting incoming packets and applies rules for forwarding, priority etc. It is controlled by the ovs-vswitchd daemon which stores configuration states in the ovsdb server. The ovsdb server is also the interface to external controllers which store persistent configuration data if the virtual switch should fail (in which case all configurations may be lost).

In order to interact directly with ovs-vswitchd, the command **ovs-vsctl** is used. Example uses are:

```
> ovs-vsctl show (prints out a summary of all switches, ports and
interfaces)
> ovs-vsctl add-br ‘sw1’ (adds a new switch to the system called ‘sw1’)
> ovs-vsctl add-port sw1 p1 (adds a new port called p1 to sw1)
> ovs-vctl del-br sw1 (deletes sw1)
```

In order to interact with the switching system using Openflow one uses the **ovs-ofctl** command. Example uses are:

```
> ovs-ofctl show sw1 (prints information about this switch)
> ovs-ofctl dump-flows sw1 (prints information about all installed flow
rules on sw1)
> ovs-ofctl add-flow sw1 action=normal (adds a new flow to switch sw1)
```

The next Linux command needed to configure virtual switches is the **ip** command. **ip** is a command line interface to manage network interfaces and configure IP addressing. Let’s assume you have created a switch sw1, and a port p1 on that switch. In order to start using the port and actually be able to send traffic through it, the first thing to do is to configure the IP address and net mask for the port. This is done using:

```
> sudo ip adds add 192.168.1.2/24 dev p1
```

which will configure p1 with IP address 192.168.1.2 and net mask 255.255.255.0 (24 bit). The next step it to make the port active. This is done using:

```
> sudo ip link set p1 up
```

you can now check that the port is configured and active using:

```
> ip a
```

which will print out a table of all physical and virtual network interfaces on the computer. There should now be an entry listing p1 with IP address 192.168.1.2/24 and a mac address.

Finally, let us go through a complete example defining and configuring a simple switch with two ports. Make sure you don’t have any existing virtual switches in the system by deleting any existing ones. We use the following commands to set up our simple switch:

```
> sudo ovs-vsctl add-br sw1
> sudo ovs-vsctl add-port sw1 p1 – set Interface p1 type=internal (– is
                                                          double minus)
> sudo ovs-vsctl add-port sw1 p2 – set Interface p2 type=internal
> sudo ip addr add 192.168.1.1/24 dev p1
> sudo ip addr add 192.168.1.2/24 dev p2
> sudo ovs-vsctl show (testing, should show sw1, p1 and p2)
```

```
> sudo ovs-vsctl list-ports sw1 (alternative testing, should list p1,p2)
> sudo ip link set p1 up
> sudo ip link set p2 up
> ip a (check that both interfaces have been configured)
> ping 192.168.1.1
> ping 192.168.1.2 (check that both interfaces are up and running)
```

Now you have created your first working virtual switch in the Linux environment and it is time to introduce the concept of namespaces.

Linux namespaces

Namespaces is a powerful concept that allows different processes to have different views of the routing framework on a physical platform. It enables a system administrator to manage which resources should be available to which processes and virtual environments. As an illustration, consider the old standard way of implementing routing on a server. Assume that the server has a single ethernet interface called eth0. There is one common routing table containing a default gateway “0.0.0.0” and a configured IP address, e.g. “192.168.1.16” for eth0. Every process running on this computer would see the same routing information and automatically send any traffic through eth0 in the same fashion. Let us now assume that you want to separate traffic so some processes send their data through a virtual tunnel “vtn1” and some other processes send their data through another virtual tunnel “vtn2” through different gateways. Namespaces allow you to easily set up different routing tables (with the different tunnels and gateways) where you can assign different processes to the different namespaces. You can also assign different performance parameters to different namespaces and prioritise traffic differently, all using namespaces and virtualisation.

In order to create a namespace simply type:

```
> sudo ip netns add <new_namespace_name>
```

for example:

```
> sudo ip netns add space1
```

You can now check that the namespace was created using:

```
> sudo ip netns list
```

In order to execute a command in a specific namespace use:

```
> sudo ip netns exec <command>
```

for example:

```
> sudo ip netns exec space1 ip link show
```

The next step is to create virtual links and assign these to the namespaces or assign physical interfaces as needed. A word of caution for this lab. If you start playing around with the physical interfaces you may lose connectivity with the outside world which is bad if you connect to one of the Linux servers in the lab. Don’t do that!!!!!!!! We will now follow a full example to construct the simple default network topology Mininet creates when fired up (answer to question 1). This will illustrate the different steps taken to virtualise different environments for different processes on a server. In the following we assume that you become superuser by typing in:

```
> xterm &
```

in the new terminal type:

```
> sudo su
```

and passphrase to avoid having to use sudo every time. Remember to start with a clean system to avoid problems.

```
# Start by creating host namespaces
ip netns add h1
ip netns add h2
# Create switch
ovs-vsctl add-br s1
# Create virtual links
ip link add h1-eth0 type veth peer name s1-eth1
ip link add h2-eth0 type veth peer name s1-eth2
ip link show
# Move host ports into namespaces
ip link set h1-eth0 netns h1
ip link set h2-eth0 netns h2
# View links
ip netns exec h1 ip link show
ip netns exec h2 ip link show
# Connect switch ports to switch
ovs-vsctl add-port s1 s1-eth1
ovs-vsctl add-port s1 s1-eth2
ovs-vsctl show
# Set up Openflow controller
ovs-vsctl set-controller s1 tcp:127.0.0.1
controller ptcp: &
ovs-vsctl show
# Configure network
ip netns exec h1 ip addr add 10.0.0.1/8 dev h1-eth0
ip netns exec h2 ip addr add 10.0.0.2/8 dev h2-eth0
ip netns exec h1 ip link set h1-eth0 up
ip netns exec h2 ip link set h2-eth0 up
ip link set s1-eth1 up
ip link set s1-eth2 up
# Test network
ip netns exec h1 ping -c1 10.0.0.2
```

Question 8: Draw the system you just created including all IP addresses, port names and MAC addresses.

This concludes part 1 of the project.