Student Name: Marco Ermini  
Email: mermini@asu.edu  
Submission Date: 14th Jun, 2021  
Class Name and Term: CSE548 Summer 2021

Project Report 2 – SDN-Based Stateless Firewall

# Project Overview

In this lab we are exploring how to set up a software defined environment based on mininet and containernet. We also get to practice how to set up an OpenFlow based flow-level firewall on SDN. Finally, we need to set up and practice flow-based firewall filtering policies such as enabling the ability to accept, drop, or reject the incoming flows thus ensuring the safety of the system from malicious attacking network traffic.

All the files and configurations used for this lab have been uploaded on GitHub; references are provided throughout the text and in the Appendix A at the bottom of the File.

# Network Setup

Since I have experienced some issues with connecting the VM to my lab through my host PC running Windows 10, I have chosen to set up the VM in VirtualBox in a bridged network configuration.

In this way, I could avoid configuring a static IP address to the VM and simply opted to fetch an IP for the VM via DHCP; this IP is then assigned directly to my router, which can be useful to troubleshoot eventual connectivity issues. On the Ubuntu/Linux side this has brought no issues whatsoever, once the configuration commands are adjusted (e.g. use “dhclient br0” rather than assigning an IP address with “ifconfig br0”).

Because of the use of DHCP, depending on the lab run and when I restarted the VM, it may have assumed a different address in the 192.168.1/24 network. This does not affect the outcome of the lab exercises, but it may look inconsistent in the screenshots. Apologies about that.

Please find below the initial set-up of the virtual infrastructure as I have configured it in VirtualBox.

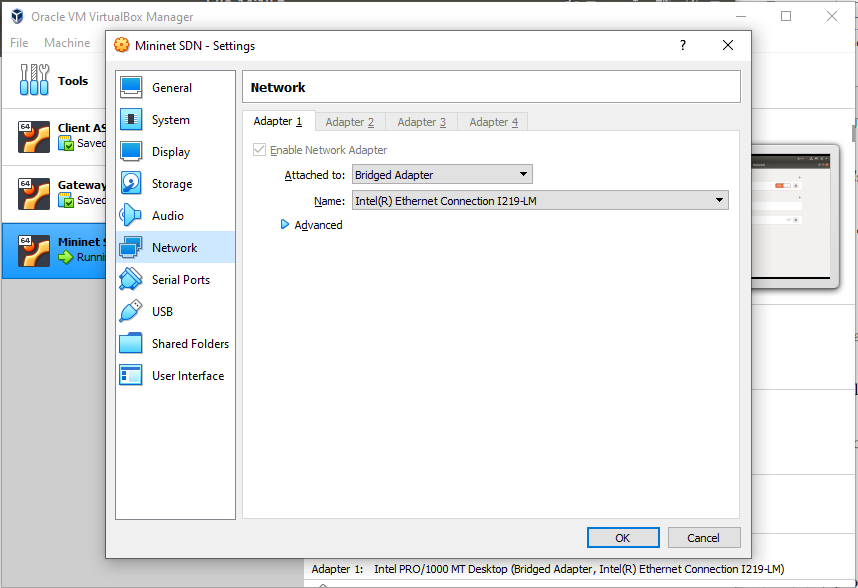


Figure - Bridged network setup in VirtualBox

# Software

For this first lab, the following software has been used:

* Various network tools (specifically, tcpdump, ping, traceroute, hping3, and nc – netcat)
* POX (GitHub link: <https://noxrepo.github.io/pox-doc/html/>
* Open vSwitch: <http://www.openvswitch.org/>
* Open vSwitch Cheat Sheet: <https://therandomsecurityguy.com/openvswitch-cheatsheet/>
* Containernet: <https://containernet.github.io/>
* Containernet tutorial: <https://github.com/containernet/containernet/wiki/Tutorial:-Getting-Started>

# Project Description

In this assignment, I have executed the various labs steps and obtained the proofs that the assignments have been completed.

## Lab “lab-cs-net-00006 OpenVirtual Switch”

As mentioned in the “Network Setup” section, the interesting part of this lab happens because a bridge is required, and I am using DHCP. The following screenshots illustrate how DHCP is used on the bridge and the MAC address of the VM external interface is assigned to the bridge.

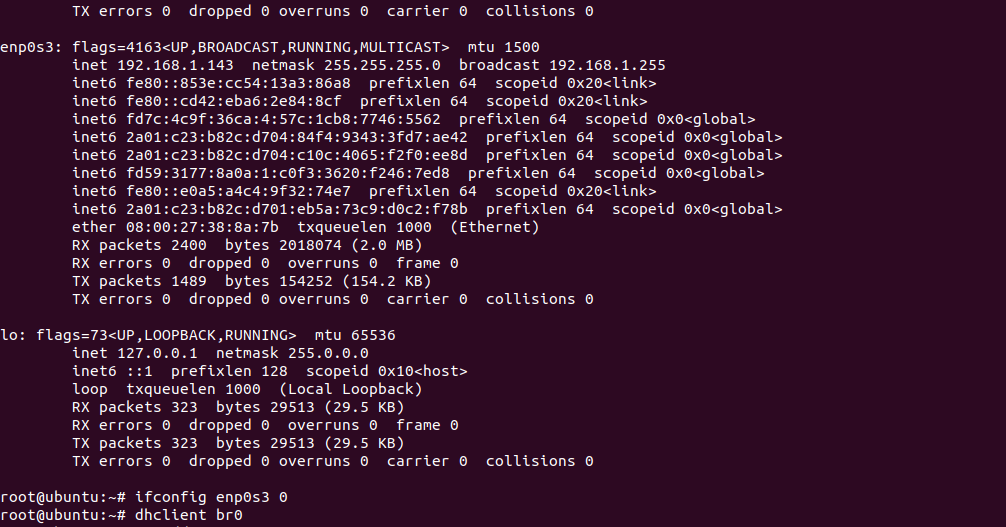


Figure - On the created bridge br0, dhclient is run

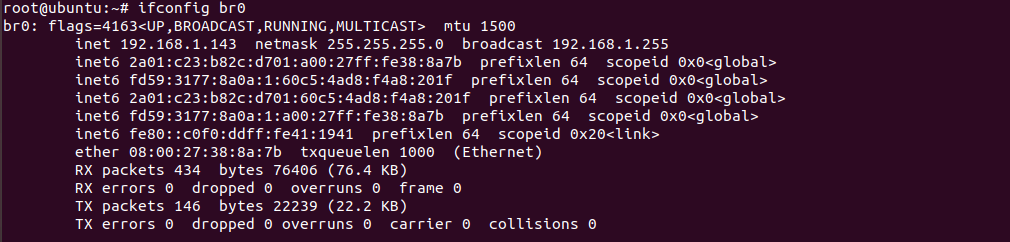


Figure - The bridge inherits the same IP of the enp0s3 interface

The bridge obtains the same IP from the router because it uses the same MAC address of the interface, and therefore the route is just reusing a DHCP lease already assigned.

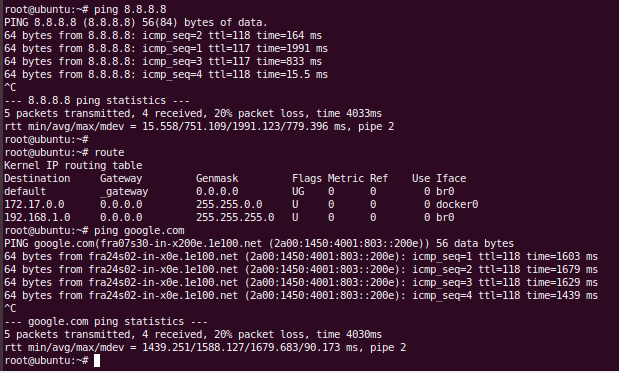


Figure - Pinging the outside world

As we can see, pinging on some Internet hosts works – even in IPv6! However, there is a performance penalty to be paid by the double software bridge (on the Linux VM and on the VirtualBox host), which causes packet losses. Generally speaking, it caused no issues – I could run *apt* and update and install packages as needed.

### Lab Assessment 1: The OVS…

* …can correctly show the created bridges via the open vswitch command:

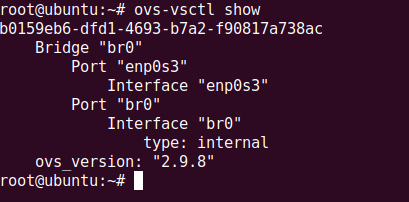


Figure - lab-cs-net-00006 - Lab 1-1

* … can show the correct bound between the br0 and ens33 (in my case, enp0s3):

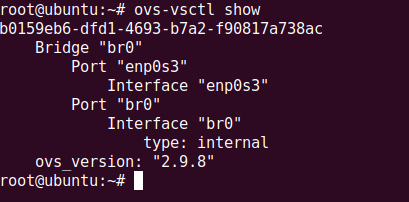


Figure - lab-cs-net-00006 - Lab 1-2

* …can show the correct IP address assigned to the br0 (in my case, it is obtained via DHCP):

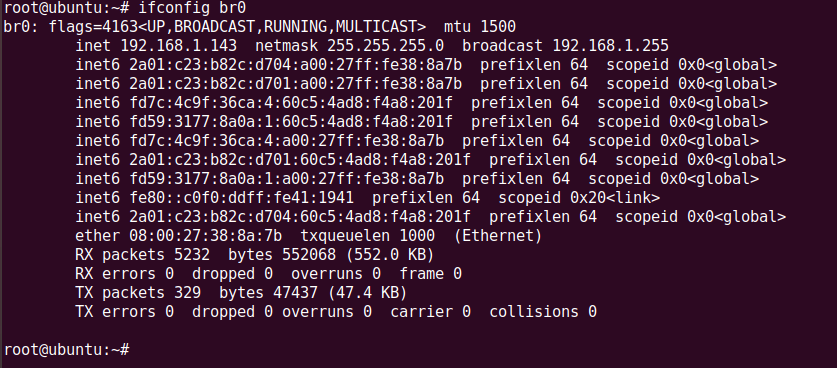


Figure - lab-cs-net-00006 - Lab 1-3/1

As a counterproof, I am showing that the ep0s3 interface has no (IPv4) address. To note that IPv6 is unaffected by “ifconfig 0” (I wonder if this has to do with the packet loss; note that both interfaces’ statistics do not report any transmission error):

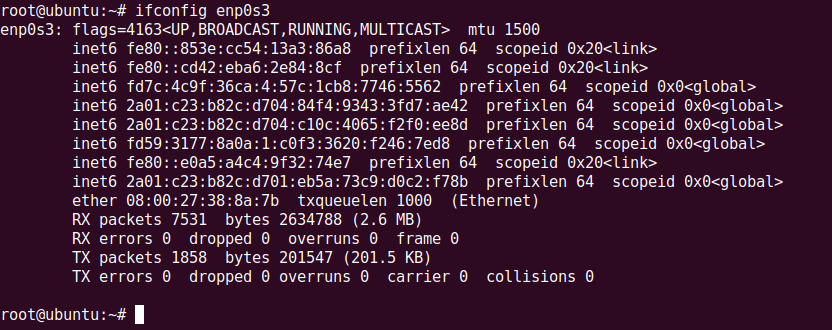


Figure - lab-cs-net-00006 - Lab 1-3/2

### Lab Assessment 2: After finishing the configuration of br0 in OVS, the machine…

* …can show the correct routing table for especially for the br0 and ens3 (in my case, enp0s3):

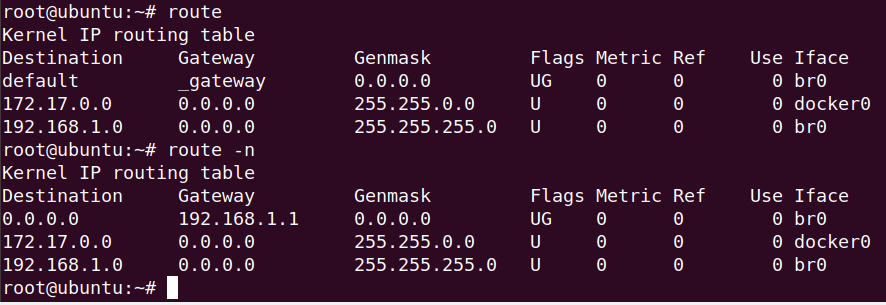


Figure - lab-cs-net-00006 - Lab 2-1

* …can ping outside such as ping 8.8.8.8 correctly:

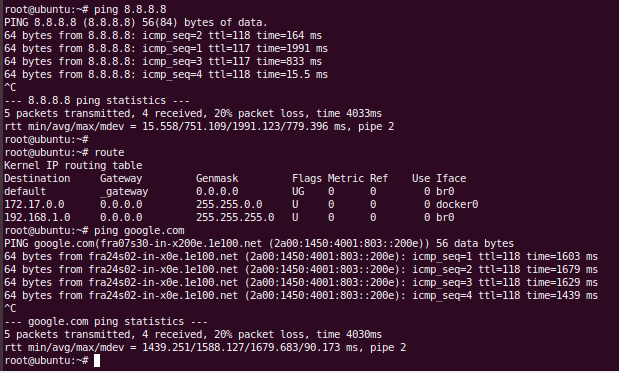


Figure - Ping outside

## Lab “lab-cs-net-00007 - Mininet”

### Setting up mininet and Running mininet topology…

* *…can correctly create the topology with a single switch and four hosts*

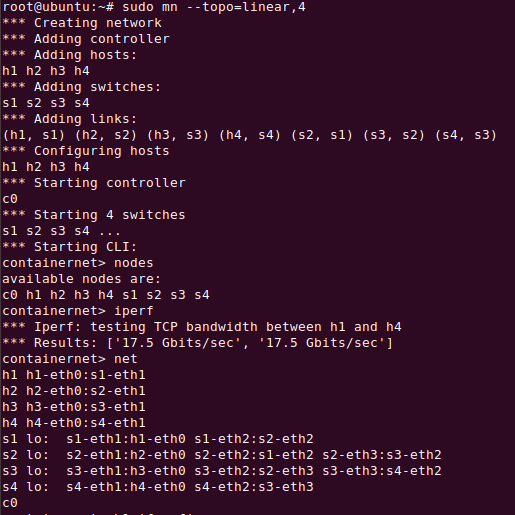


Figure - lab-cs-net-0007 - Lab 1-1

* …can correctly create the linear topology with five switch and one host for every switch:

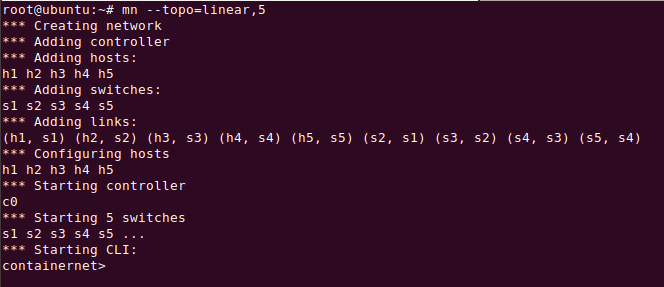


Figure - lab-cs-net-0007 - Lab 1-2

### For each topology above, they should be able to:

* show the correct number of nodes within the current topology

Single switch and four hosts:

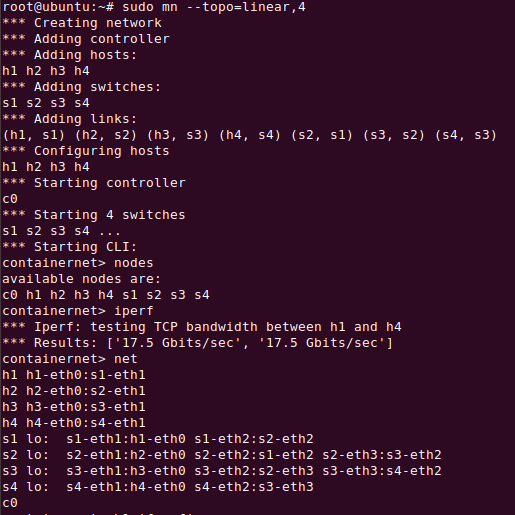


Figure - lab-cs-net-0007 - Lab 2-1 – topology 1

Linear topology of five switch and one host for each switch:

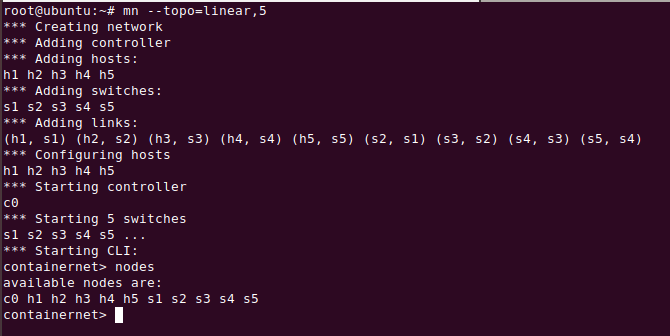


Figure - lab-cs-net-0007 - Lab 2-1 – topology 2

* …perform bandwidth measurement.

Single switch and four hosts:

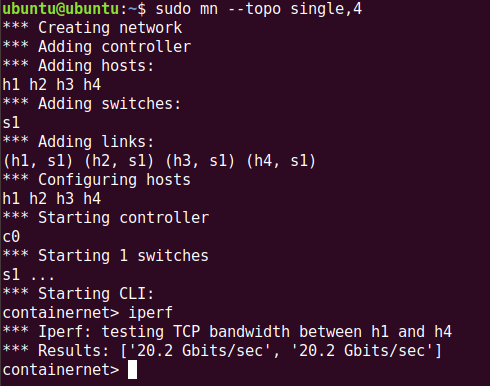


Figure - lab-cs-net-0007 - Lab 2-2 - Topology 1

Linear topology of five switch and one host for each switch:

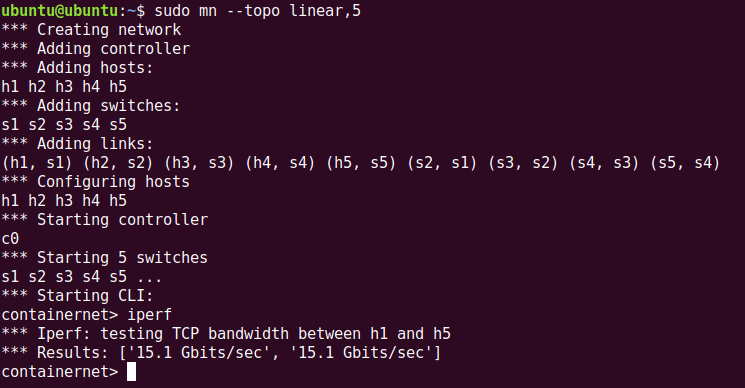


Figure - lab-cs-net-0007 - Lab 2-2 - Topology 2

* …display the correct link information among hosts and switches:

Single switch and four hosts:

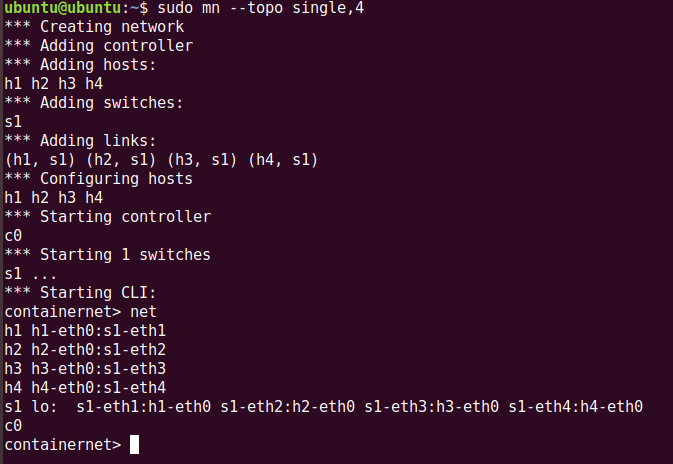


Figure - lab-cs-net-0007 - Lab 2-3 - Topology 1

Linear topology of five switch and one host for each switch:

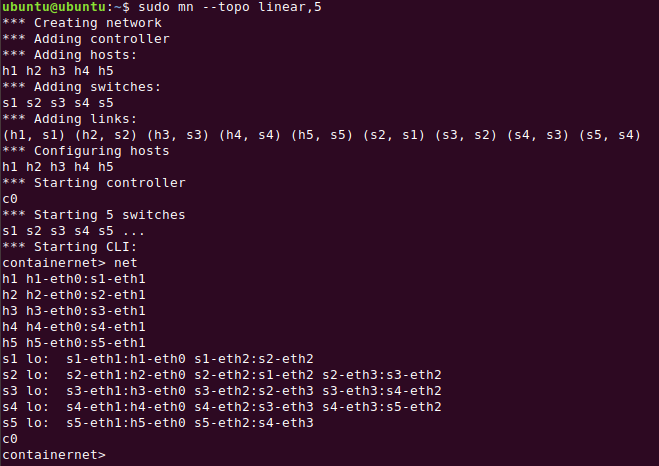


Figure - lab-cs-net-0007 - Lab 2-3 - Topology 2

### Create another tree topology of depth 2 and fanout 8.

* Startup with “*sudo mn --topo tree,depth=2 fanout=8*”:

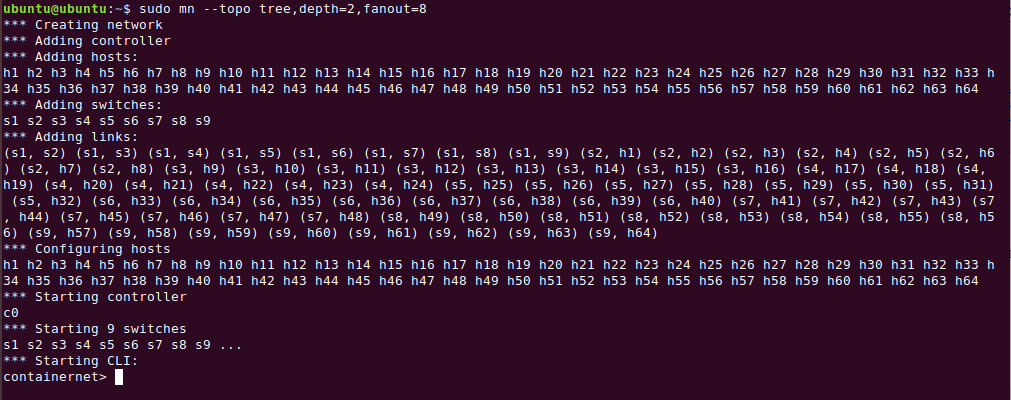


Figure - lab-cs-net-0007 - Lab 3-1

* the host1 can correctly ping the host64:

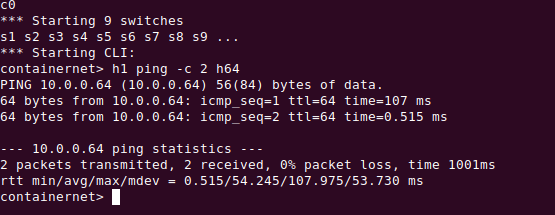


Figure - lab-cs-net-0007 - Lab 3-2

(End of “lab-cs-net-00007 - Mininet”)

## Lab “CS-NET-00008 – SDN Controller (POX)”

### The POX controller can:

* connect to a mininet topology:

The behavior of pox and mininet can be observed by switching between the two tabs where the programs are running, showing that mininet connected to openflow:

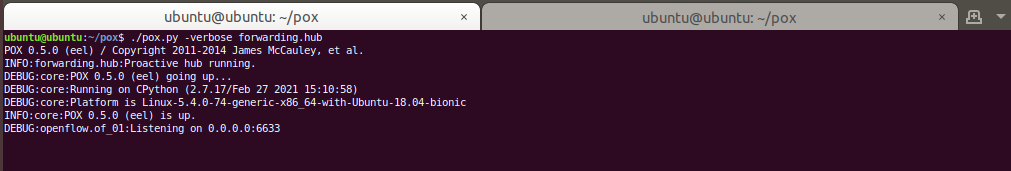


Figure - lab-cs-net-0087 - Lab 1-1/1

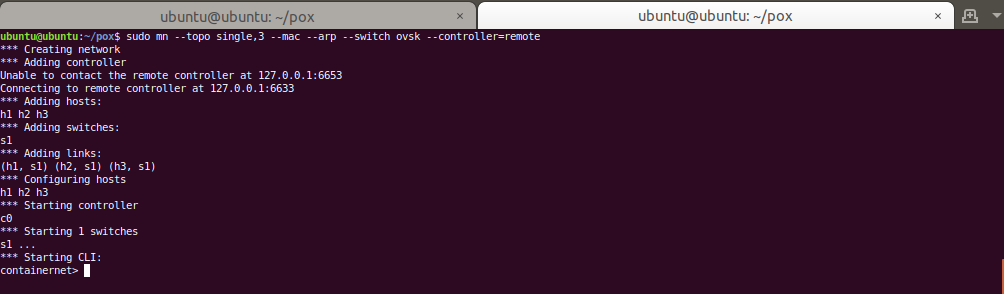


Figure - lab-cs-net-0087 - Lab 1-1/2

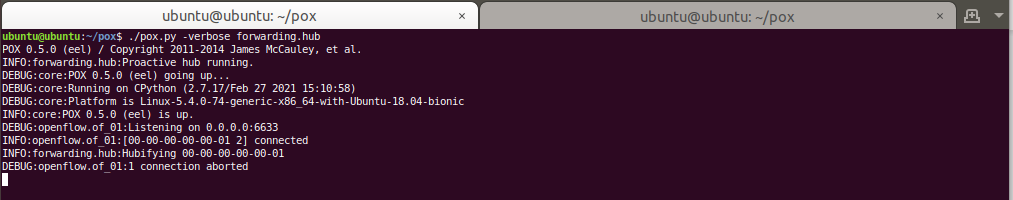


Figure - lab-cs-net-0087 - Lab 1-1/3

* start the forwarding.l3 learning component correctly

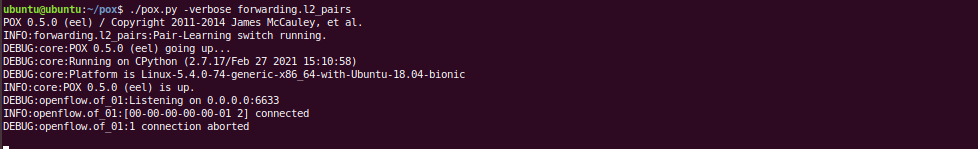


Figure - lab-cs-net-0087 - Lab 1-2/1

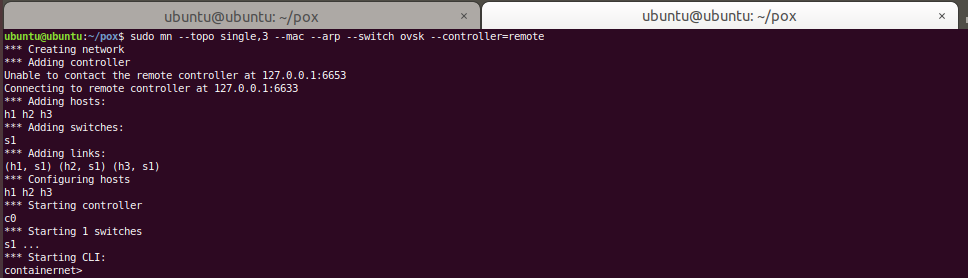


Figure - lab-cs-net-0087 - Lab 1-2/2

### While connect to the controller, the mininet topology can (Please take snapshots for each task)

* open the xterm terminal correctly
* enable the ping command between two different host in two different xterm terminal (two different hosts)

On this first image, I have opened the three xterm with the command “xterm h1 h2 h3” highlighted on the bottom. I have also run a couple of ping commands.

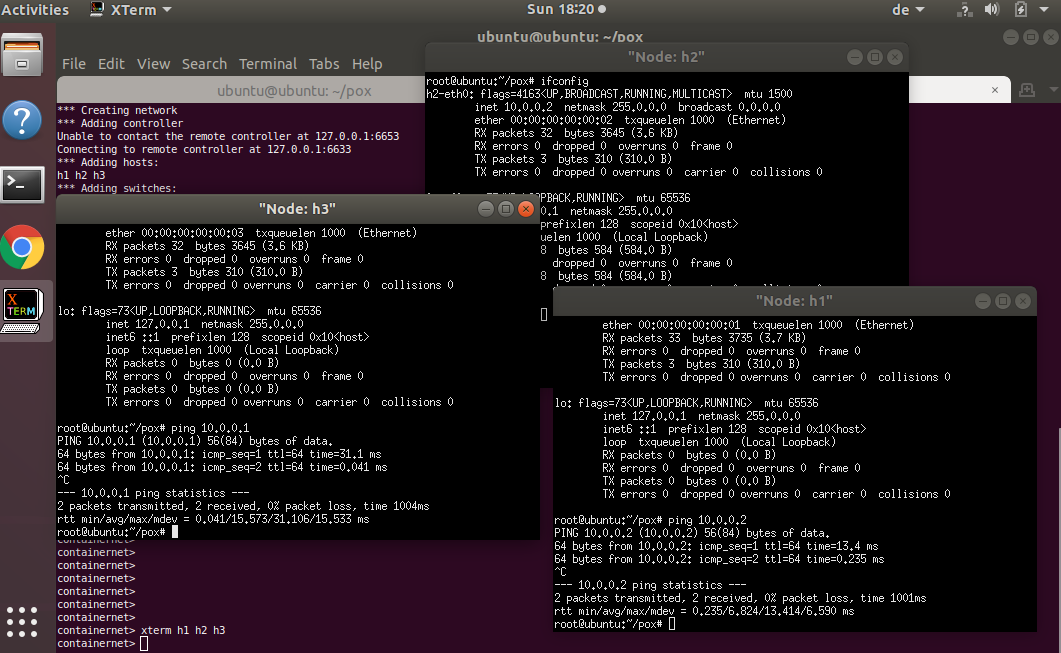


Figure - lab-cs-net-0087 - Lab 2-1

In this screenshot, I am running tcpdump on one xterm while executing a ping from another. In this way it is possible to notice the incoming packets.

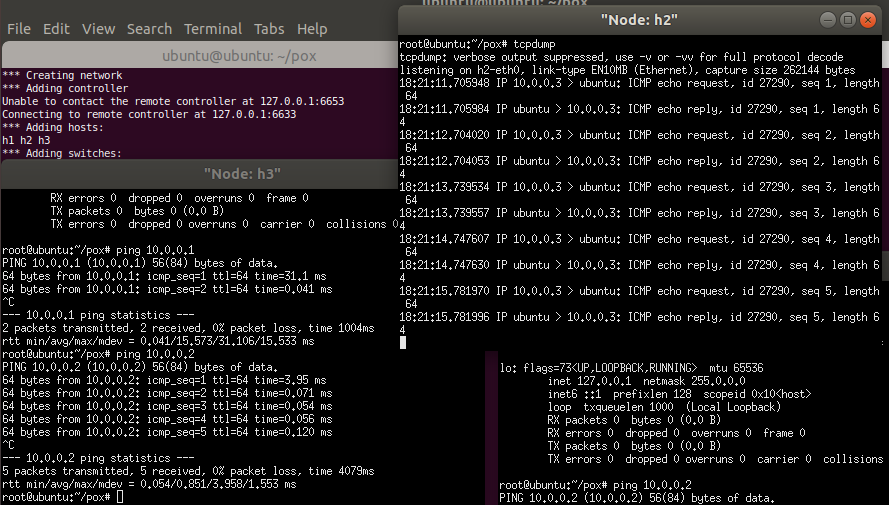


Figure - lab-cs-net-0087 - Lab 2-2

### Extra Activity

I have started the pox webserver connected to mininet to test its functionality.

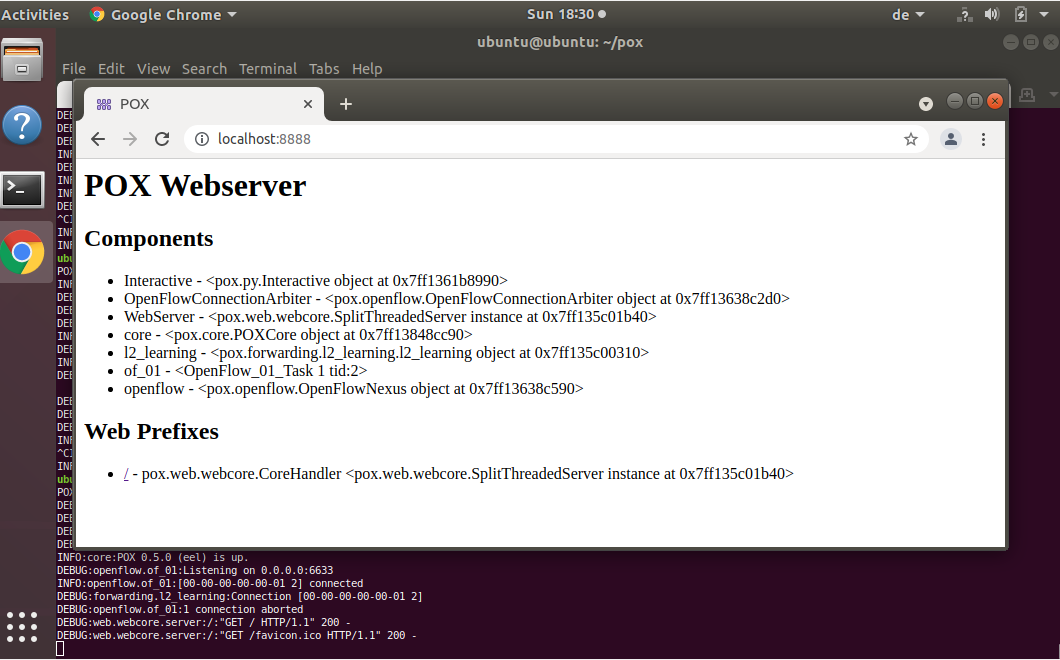


Figure - - lab-cs-net-0087 - Extra Activity

## Lab “CS-NET-00009 – Containernet”

### The containernet topology can:

* successfully implement the test ping command in a temporary topology

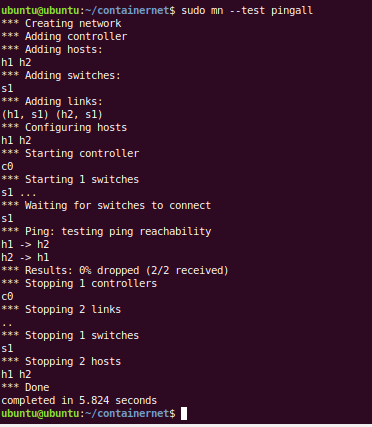


Figure - lab-cs-net-0009 – Lab 1-1

After started the scripts the container example.py can

* create the indicated topology

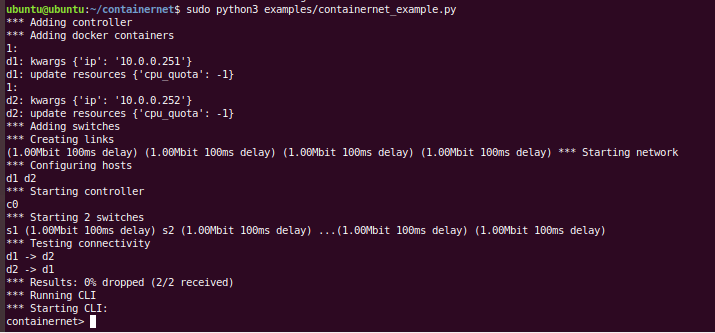


Figure - lab-cs-net-0009 – Lab 1-2

* enable the communication between the d1 and d2 (link up the two switches):

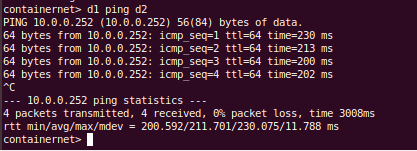


Figure - lab-cs-net-0009 – Lab 1-3

(End of “lab-cs-net-00009 - Containernet”)

## Lab “CS-CNS-00101” – OpenFlow Based Stateless firewall

In this lab, the students are required to verify working of stateless firewall and try adding different rules using

config files.

### Create a mininet based topology with 4 container hosts and one controller switches and run it.

* + *Add link from controller1 to switch 1.*
  + *Add link from controller2 to switch 1.*
  + *Add link from switch 1 to container 1.*
  + *Add link from switch 1 to container 2.*
  + *Add link from switch 1 to container 3.*
  + *Add link from switch 1 to container 4.*

This is accomplished by two steps: first, creating two controllers with pox (only one of the two will be really used). I have created a script, called “run\_pox.sh” (present on GitHub and the Appendices), which spawns two controllers, as show below.

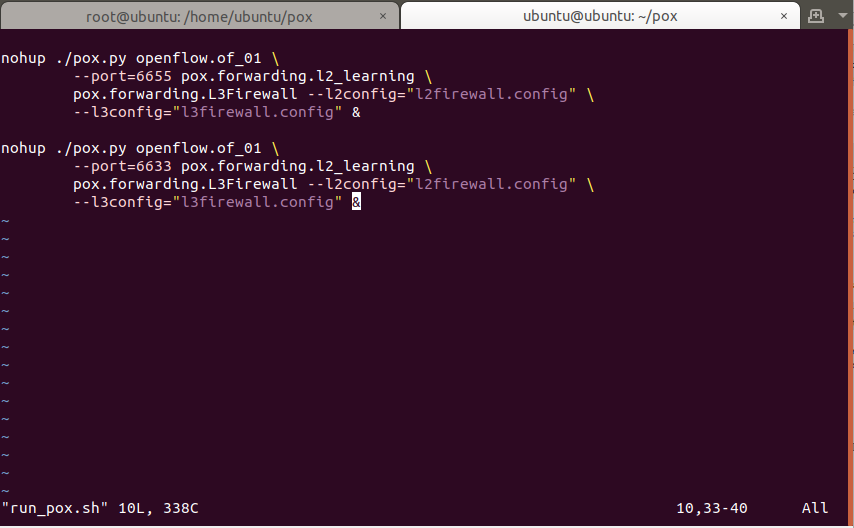


Figure - CS-CNS-00101 – run\_pox.sh

The controllers are identifiable as python programs from the Ubuntu process status:

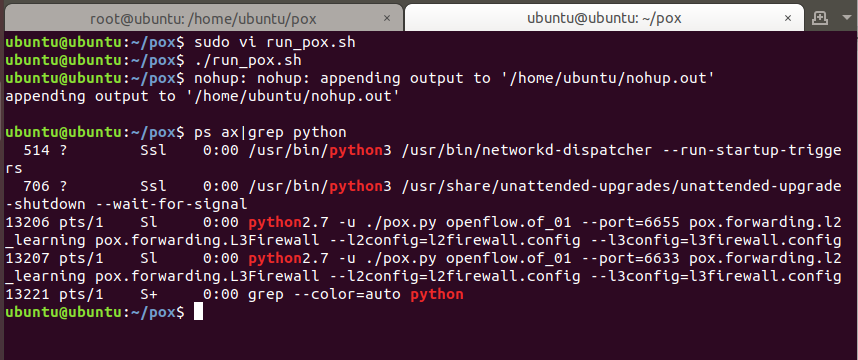


Figure - CS-CNS-00101 – run\_pox.sh running

The second step is to run another script from another Terminal tab or windows, launching mininet. This file, called “run\_lab.sh”, is also presented in the Appendices and GitHub.



Figure - CS-CNS-00101 – run\_lab.sh

After mininet starts, it will bind to the two pox controllers without errors (if everything goes smoothly):

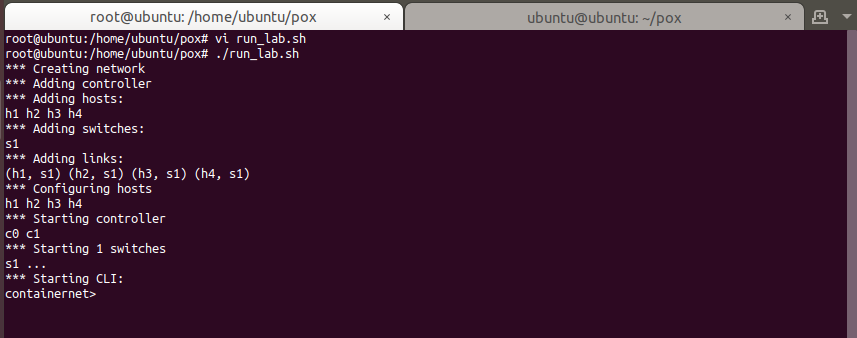


Figure - CS-CNS-00101 – run\_lab.sh running

At this point we can run the xterm and observe that the containers work.

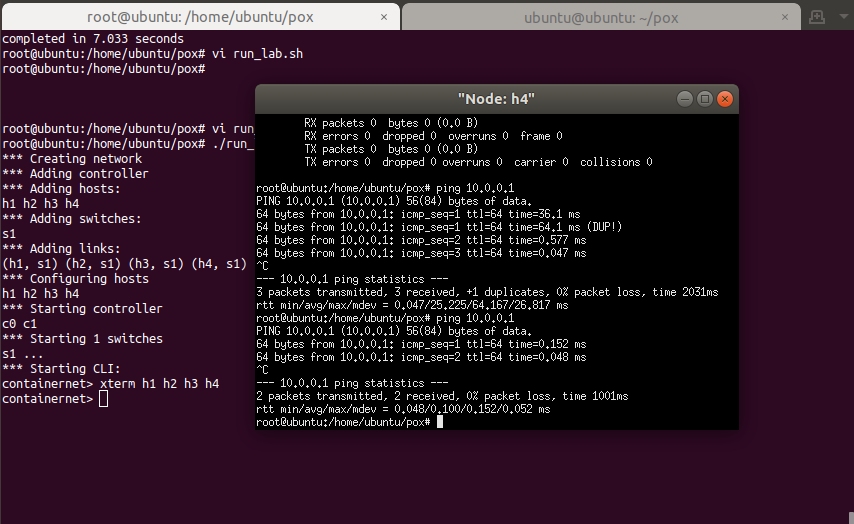


Figure - CS-CNS-00101 – Lab 1

### Make the interfaces up and assign IP addresses to interfaces of container hosts.

* *Assign IP address 192.168.2.10 to container host #1.*
* *Assign IP address 192.168.2.20 to container host #2.*
* *Assign IP address 192.168.2.30 to container host #3.*
* *Assign IP address 192.168.2.40 to container host #4*

The following commands are ran from the mininet CLI to assign them the addresses requested by the lab:

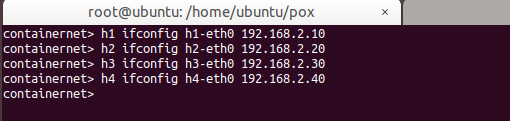


Figure - CS-CNS-00101 – Change IP addresses

The next screenshot to document that the containers have all assumed the correct IP addresses.

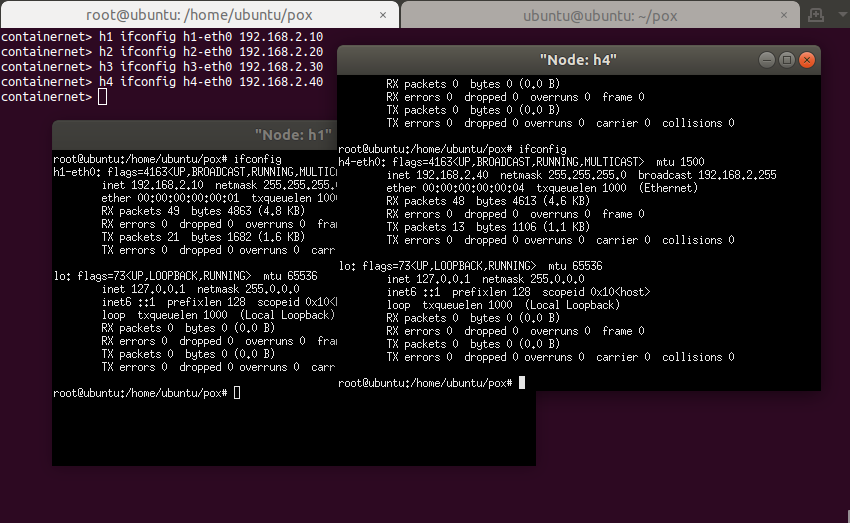


Figure - CS-CNS-00101 – Lab 2

### Add new rule to l3config file for blocking ICMP traffic from source IP 192.168.2.10 and destination IP 192.168.2.30.

### Add new rule to l3config file for blocking ICMP traffic from source IP 192.168.2.20 and destination IP 192.168.2.40.

### Add new rule to l3config file for blocking HTTP traffic from source IP 192.168.2.20.

### Add new rule to l2config file for blocking traffic from MAC address 00:00:00:00:00:02 to destination MAC address 00:00:00:00:00:04.

### Add new rule to l3config file for blocking tcp traffic from 192.168.2.10 to 192.168.2.20.

### Add new rule to l3config file for blocking udp traffic from 192.168.2.10 to 192.168.2.20.

The following screenshot documents the l3firewall.config:

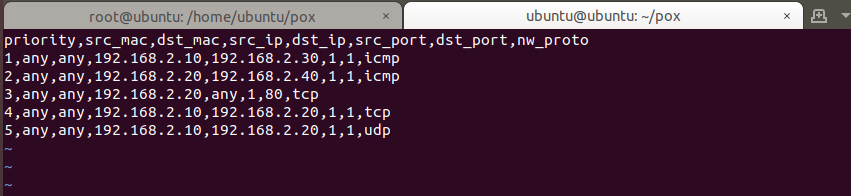


Figure - CS-CNS-00101 – l3firewall.config

The following screenshot documents the l2firewall.config:

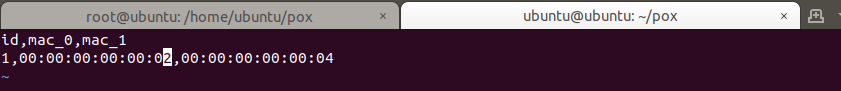


Figure - CS-CNS-00101 – l2firewall.config

At this point, we restart both pox and mininet.

From looking at the “nohup.out” generated by the run\_pox.sh, it is possible to see the Layer3 rules enabled (but not the Layer2):

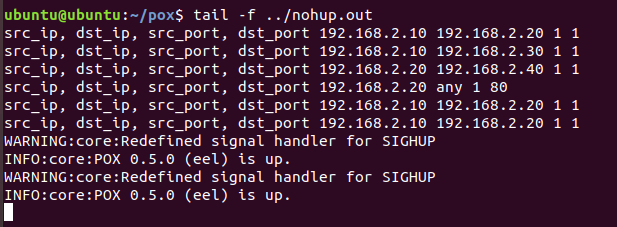


Figure - CS-CNS-00101 – checking nohup.out

After we restart mininet, we must re-assign the correct IP addresses (fortunately, the CLI history has stored the command, so we must just retrieve them with the arrows and feed them to the Open vSwitch pressing Enter):

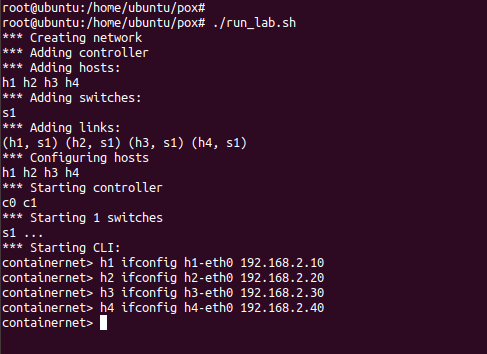


Figure - CS-CNS-00101 – Changing IP addresses again

Let’s now test out our rules to see if they work.

Let’s start from pinging h3 from h1. The ping should fail, but in fact, the first packet gets true, but all the subsequent are blocked. Seems like one rule must be triggered first before the vSwitch “loads” them in, but at this point they are all enabled.

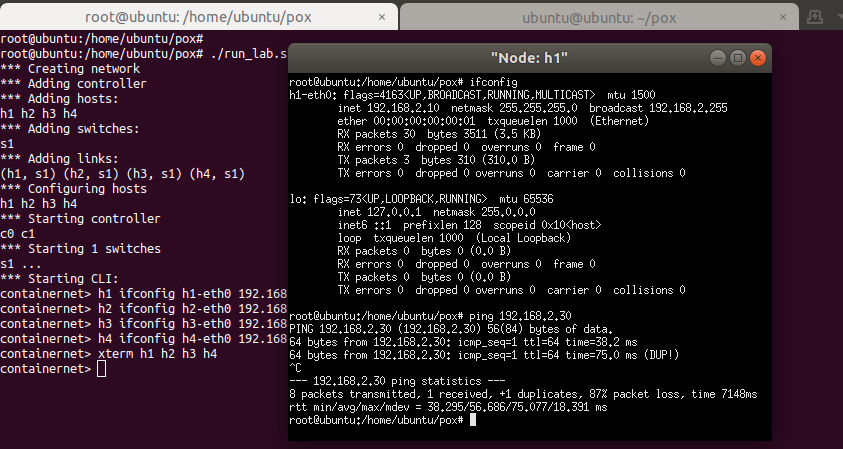


Figure - CS-CNS-00101 – first ping from h1 to h3 gets through, then the rest are blocked

In fact, pinging h4 from h2 fails (not even the first packet gets through):

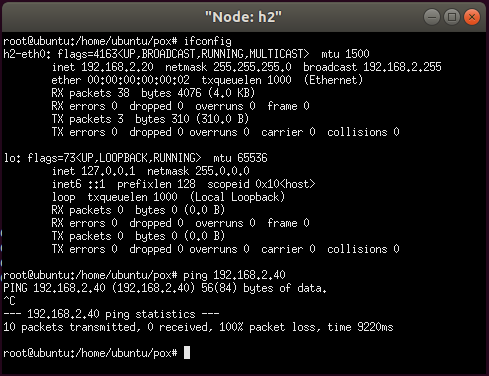


Figure - CS-CNS-00101 – pings blocked from h2 to h4

I start the Python SimpleHTTPServer on port 80 of node h3 and test fetching a page from the node h4, which works (as expected):

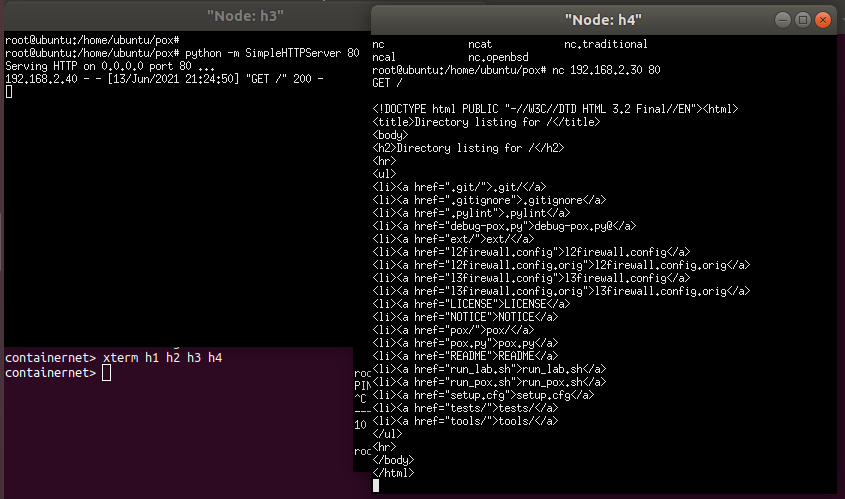


Figure - CS-CNS-00101 – h4 can reach web server on h3

However, trying to “browse” the web page at port 80 from node h2 – no matter if with netcat or curl – fails as the connection is dropped and, as we can see, it never reaches the node h3 – confirming the Layer3 rule works as intended:

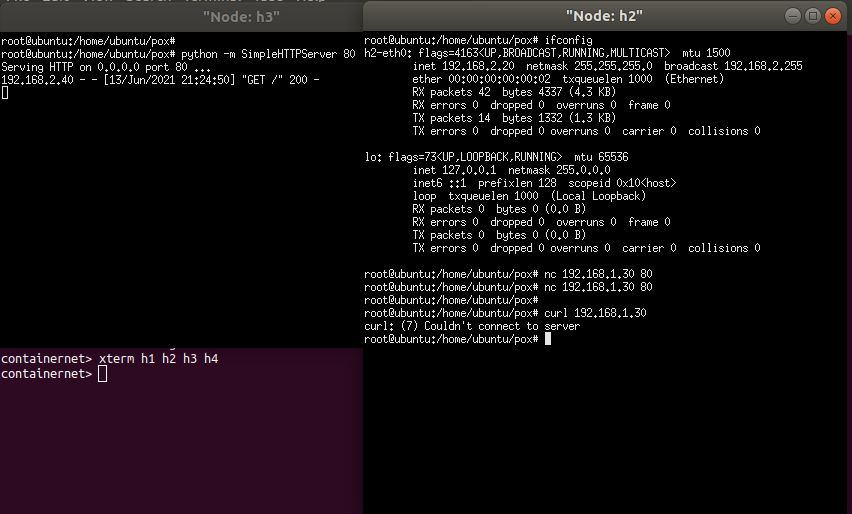


Figure - CS-CNS-00101 – h2 cannot reach web server on h3

Another test against the web server on h3, this time from h1, shows that effectively only h2 is blocked from reaching web pages.

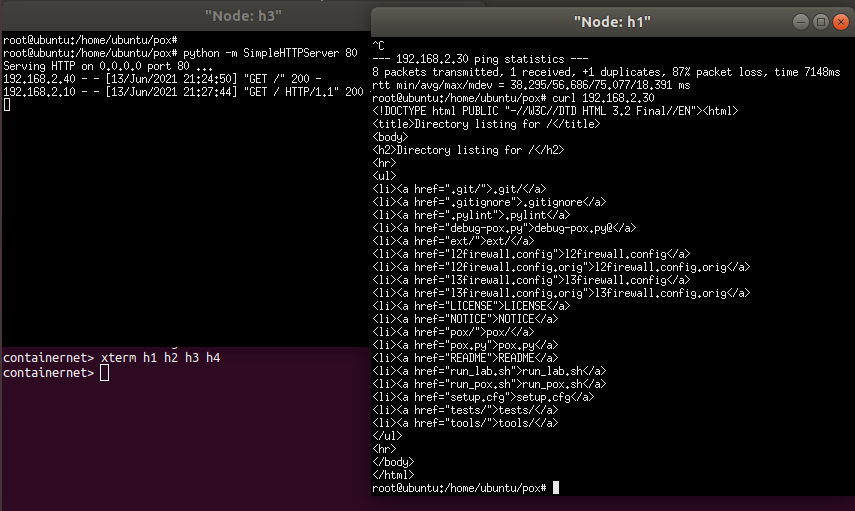


Figure - CS-CNS-00101 – h1 can reach web server on h3

Finally, between h2 and h4, the block should be at the Layer2 level. To test this, I have tried first using ping, then running tcpdump on h4, while doing scans from node h2.

The first is a UDP scan:

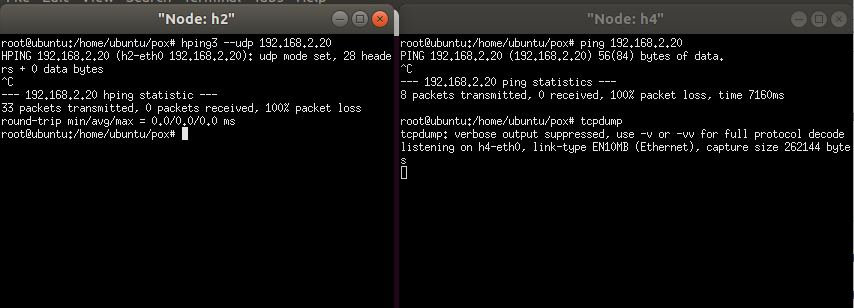


Figure - CS-CNS-00101 – h2 ping and UDP scan against h4 fail

The second is a TCP scan on some 8000 circa ports. As we can see, no packet ever reaches node h4 – be it ICPM, TCP or UDP.

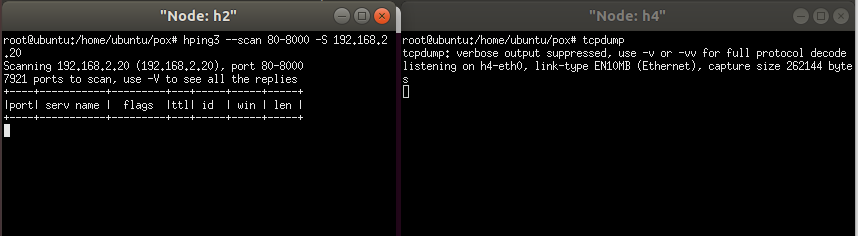


Figure - CS-CNS-00101 – h2 TCP scan against h4 fails

This is how hping3 spits out its failure against h4; the result from h4 shows that no packet has ever reached the interface.

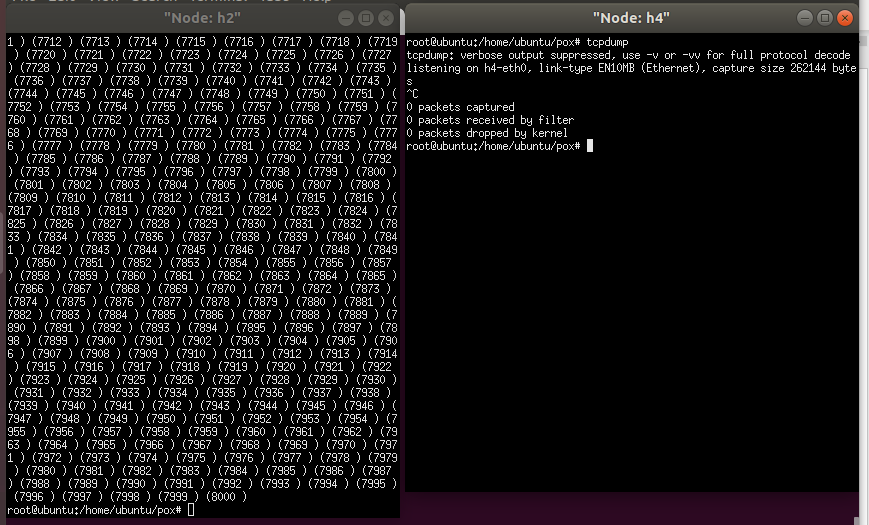


Figure - CS-CNS-00101 – h2 TCP scan against h4 fails

Finally, we can dump the flows directly from the vSwitch, to verify that rules are in place (packet counts only show a few rules because the screenshot has been taken throughout restarts):

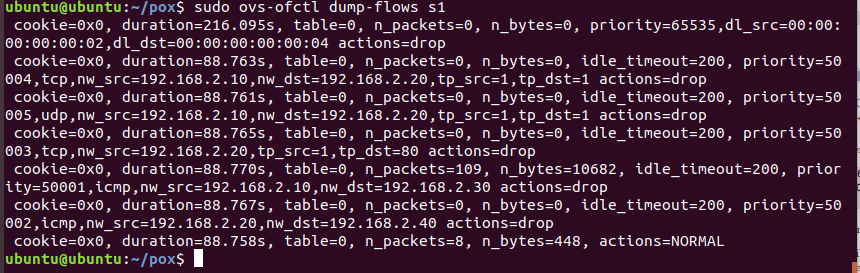


Figure - CS-CNS-00101 – ovs-ofctl dump-flows s1

# Appendix A: Files for the Lab

Please find the list of files created for this lab and mentioned throughout this document, plus their GitHub link for download.

The overall GitHub directory for the project is: <https://github.com/markoer73/CSE-548/tree/main/Project%202%20-%20SDN-Based%20Stateless%20Firewall>

|  |  |
| --- | --- |
| Project-Report-2 SDN-Based Stateless Firewall.docx | <https://github.com/markoer73/CSE-548/blob/main/Project%202%20-%20SDN-Based%20Stateless%20Firewall/Project-Report-2%20SDN-Based%20Stateless%20Firewall.docx> |
| l2firewall.config | <https://github.com/markoer73/CSE-548/blob/main/Project%202%20-%20SDN-Based%20Stateless%20Firewall/l2firewall.config> |
| l3firewall.config | <https://github.com/markoer73/CSE-548/blob/main/Project%202%20-%20SDN-Based%20Stateless%20Firewall/l3firewall.config> |
| run\_lab.sh | <https://github.com/markoer73/CSE-548/blob/main/Project%202%20-%20SDN-Based%20Stateless%20Firewall/run_lab.sh> |
| run\_pox.sh | <https://github.com/markoer73/CSE-548/blob/main/Project%202%20-%20SDN-Based%20Stateless%20Firewall/run_pox.sh> |

## File Content

### l3firewall.config

priority,src\_mac,dst\_mac,src\_ip,dst\_ip,src\_port,dst\_port,nw\_proto

1,any,any,192.168.2.10,192.168.2.30,1,1,icmp

2,any,any,192.168.2.20,192.168.2.40,1,1,icmp

3,any,any,192.168.2.20,any,1,80,tcp

4,any,any,192.168.2.10,192.168.2.20,1,1,tcp

5,any,any,192.168.2.10,192.168.2.20,1,1,udp

### l2firewall.config

id,mac\_0,mac\_1

1,00:00:00:00:00:02,00:00:00:00:00:04

### run\_pox.sh

nohup ./pox.py openflow.of\_01 \

--port=6655 pox.forwarding.l2\_learning \

pox.forwarding.L3Firewall --l2config="l2firewall.config" \

--l3config="l3firewall.config" &

nohup ./pox.py openflow.of\_01 \

--port=6633 pox.forwarding.l2\_learning \

pox.forwarding.L3Firewall --l2config="l2firewall.config" \

--l3config="l3firewall.config" &

### run\_lab.sh

mn --topo=single,4 \

--controller=remote,port=6633 \

--controller=remote,port=6655 \

--switch=ovsk --mac

# References

* Linux NAT Tutorial: <https://www.karlrupp.net/en/computer/nat_tutorial>
* Ubuntu “Basic Iptables HOWTO”: <https://help.ubuntu.com/community/IptablesHowTo>
* “Iptables Tutorial: Ultimate Guide to Linux Firewall”: <https://phoenixnap.com/kb/iptables-tutorial-linux-firewall>

VIII. Table of Figures

[Figure 1 - Bridged network setup in VirtualBox 1](#_Toc74549301)

[Figure 2 - On the created bridge br0, dhclient is run 3](#_Toc74549302)

[Figure 3 - The bridge inherits the same IP of the enp0s3 interface 3](#_Toc74549303)

[Figure 4 - Pinging the outside world 3](#_Toc74549304)

[Figure 5 - lab-cs-net-00006 - Lab 1-1 4](#_Toc74549305)

[Figure 6 - lab-cs-net-00006 - Lab 1-2 4](#_Toc74549306)

[Figure 7- lab-cs-net-00006 - Lab 1-3/1 4](#_Toc74549307)

[Figure 8 - lab-cs-net-00006 - Lab 1-3/2 5](#_Toc74549308)

[Figure 9 - lab-cs-net-00006 - Lab 2-1 5](#_Toc74549309)

[Figure 10 - Ping outside 5](#_Toc74549310)

[Figure 11 - lab-cs-net-0007 - Lab 1-1 6](#_Toc74549311)

[Figure 12- lab-cs-net-0007 - Lab 1-2 6](#_Toc74549312)

[Figure 13 - lab-cs-net-0007 - Lab 2-1 – topology 1 6](#_Toc74549313)

[Figure 14 - lab-cs-net-0007 - Lab 2-1 – topology 2 7](#_Toc74549314)

[Figure 15- lab-cs-net-0007 - Lab 2-2 - Topology 1 7](#_Toc74549315)

[Figure 16- lab-cs-net-0007 - Lab 2-2 - Topology 2 7](#_Toc74549316)

[Figure 17- lab-cs-net-0007 - Lab 2-3 - Topology 1 8](#_Toc74549317)

[Figure 18- lab-cs-net-0007 - Lab 2-3 - Topology 2 8](#_Toc74549318)

[Figure 19- lab-cs-net-0007 - Lab 3-1 9](#_Toc74549319)

[Figure 20- lab-cs-net-0007 - Lab 3-2 9](#_Toc74549320)

[Figure 21 - lab-cs-net-0087 - Lab 1-1/1 10](#_Toc74549321)

[Figure 22 - lab-cs-net-0087 - Lab 1-1/2 10](#_Toc74549322)

[Figure 23 - lab-cs-net-0087 - Lab 1-1/3 10](#_Toc74549323)

[Figure 24 - lab-cs-net-0087 - Lab 1-2/1 10](#_Toc74549324)

[Figure 25 - lab-cs-net-0087 - Lab 1-2/2 11](#_Toc74549325)

[Figure 26 - lab-cs-net-0087 - Lab 2-1 11](#_Toc74549326)

[Figure 27 - lab-cs-net-0087 - Lab 2-2 12](#_Toc74549327)

[Figure 28 - - lab-cs-net-0087 - Extra Activity 12](#_Toc74549328)

[Figure 29 - lab-cs-net-0009 – Lab 1-1 13](#_Toc74549329)

[Figure 30 - lab-cs-net-0009 – Lab 1-2 13](#_Toc74549330)

[Figure 31 - lab-cs-net-0009 – Lab 1-3 14](#_Toc74549331)

[Figure 32 - CS-CNS-00101 – run\_pox.sh 15](#_Toc74549332)

[Figure 33 - CS-CNS-00101 – run\_pox.sh running 15](#_Toc74549333)

[Figure 34 - CS-CNS-00101 – run\_lab.sh 16](#_Toc74549334)

[Figure 35 - CS-CNS-00101 – run\_lab.sh running 16](#_Toc74549335)

[Figure 36 - CS-CNS-00101 – Lab 1 16](#_Toc74549336)

[Figure 37 - CS-CNS-00101 – Change IP addresses 17](#_Toc74549337)

[Figure 38 - CS-CNS-00101 – Lab 2 17](#_Toc74549338)

[Figure 39 - CS-CNS-00101 – l3firewall.config 18](#_Toc74549339)

[Figure 40 - CS-CNS-00101 – l2firewall.config 18](#_Toc74549340)

[Figure 41 - CS-CNS-00101 – checking nohup.out 18](#_Toc74549341)

[Figure 42 - CS-CNS-00101 – Changing IP addresses again 18](#_Toc74549342)

[Figure 43 - CS-CNS-00101 – first ping from h1 to h3 gets through, then the rest are blocked 19](#_Toc74549343)

[Figure 44 - CS-CNS-00101 – pings blocked from h2 to h4 19](#_Toc74549344)

[Figure 45 - CS-CNS-00101 – h4 can reach web server on h3 20](#_Toc74549345)

[Figure 46 - CS-CNS-00101 – h2 cannot reach web server on h3 20](#_Toc74549346)

[Figure 47 - CS-CNS-00101 – h1 can reach web server on h3 21](#_Toc74549347)

[Figure 48 - CS-CNS-00101 – h2 ping and UDP scan against h4 fail 21](#_Toc74549348)

[Figure 49 - CS-CNS-00101 – h2 TCP scan against h4 fails 22](#_Toc74549349)

[Figure 50 - CS-CNS-00101 – h2 TCP scan against h4 fails 22](#_Toc74549350)

[Figure 51 - CS-CNS-00101 – ovs-ofctl dump-flows s1 22](#_Toc74549351)

Contents

[I. Project Overview 1](#_Toc74549358)

[II. Network Setup 1](#_Toc74549359)

[III. Software 2](#_Toc74549360)

[IV. Project Description 2](#_Toc74549361)

[A. Lab “lab-cs-net-00006 OpenVirtual Switch” 3](#_Toc74549362)

[1) Lab Assessment 1: The OVS… 4](#_Toc74549363)

[2) Lab Assessment 2: After finishing the configuration of br0 in OVS, the machine… 5](#_Toc74549364)

[B. Lab “lab-cs-net-00007 - Mininet” 6](#_Toc74549365)

[1) Setting up mininet and Running mininet topology… 6](#_Toc74549366)

[2) For each topology above, they should be able to: 6](#_Toc74549367)

[3) Create another tree topology of depth 2 and fanout 8. 8](#_Toc74549368)

[C. Lab “CS-NET-00008 – SDN Controller (POX)” 10](#_Toc74549369)

[1) The POX controller can: 10](#_Toc74549370)

[2) While connect to the controller, the mininet topology can (Please take snapshots for each task) 11](#_Toc74549371)

[3) Extra Activity 12](#_Toc74549372)

[D. Lab “CS-NET-00009 – Containernet” 13](#_Toc74549373)

[1) The containernet topology can: 13](#_Toc74549374)

[E. Lab “CS-CNS-00101” – OpenFlow Based Stateless firewall 15](#_Toc74549375)

[1) Create a mininet based topology with 4 container hosts and one controller switches and run it. 15](#_Toc74549376)

[2) Make the interfaces up and assign IP addresses to interfaces of container hosts. 17](#_Toc74549377)

[3) Add new rule to l3config file for blocking ICMP traffic from source IP 192.168.2.10 and destination IP 192.168.2.30. 17](#_Toc74549378)

[4) Add new rule to l3config file for blocking ICMP traffic from source IP 192.168.2.20 and destination IP 192.168.2.40. 17](#_Toc74549379)

[5) Add new rule to l3config file for blocking HTTP traffic from source IP 192.168.2.20. 17](#_Toc74549380)

[6) Add new rule to l2config file for blocking traffic from MAC address 00:00:00:00:00:02 to destination MAC address 00:00:00:00:00:04. 17](#_Toc74549381)

[7) Add new rule to l3config file for blocking tcp traffic from 192.168.2.10 to 192.168.2.20. 17](#_Toc74549382)

[8) Add new rule to l3config file for blocking udp traffic from 192.168.2.10 to 192.168.2.20. 17](#_Toc74549383)

[V. Appendix A: Files for the Lab 23](#_Toc74549384)

[A. File Content 23](#_Toc74549385)

[1) l3firewall.config 23](#_Toc74549386)

[2) l2firewall.config 23](#_Toc74549387)

[3) run\_pox.sh 23](#_Toc74549388)

[4) run\_lab.sh 23](#_Toc74549389)

[VI. References 24](#_Toc74549390)

[VIII. Table of Figures 24](#_Toc74549391)