Report for hw6

In this lab, the working network included 7 machines. There is one web-server hosting **WordPress**; one firewall deployed **Snort**; one gateway server, one Openflow Controller and 3 machines as virtual switches. Also, in this lab, all of the machines are virtual machines created on **SAVI** (based on OpenStack).

The topology can be illustrated like this:

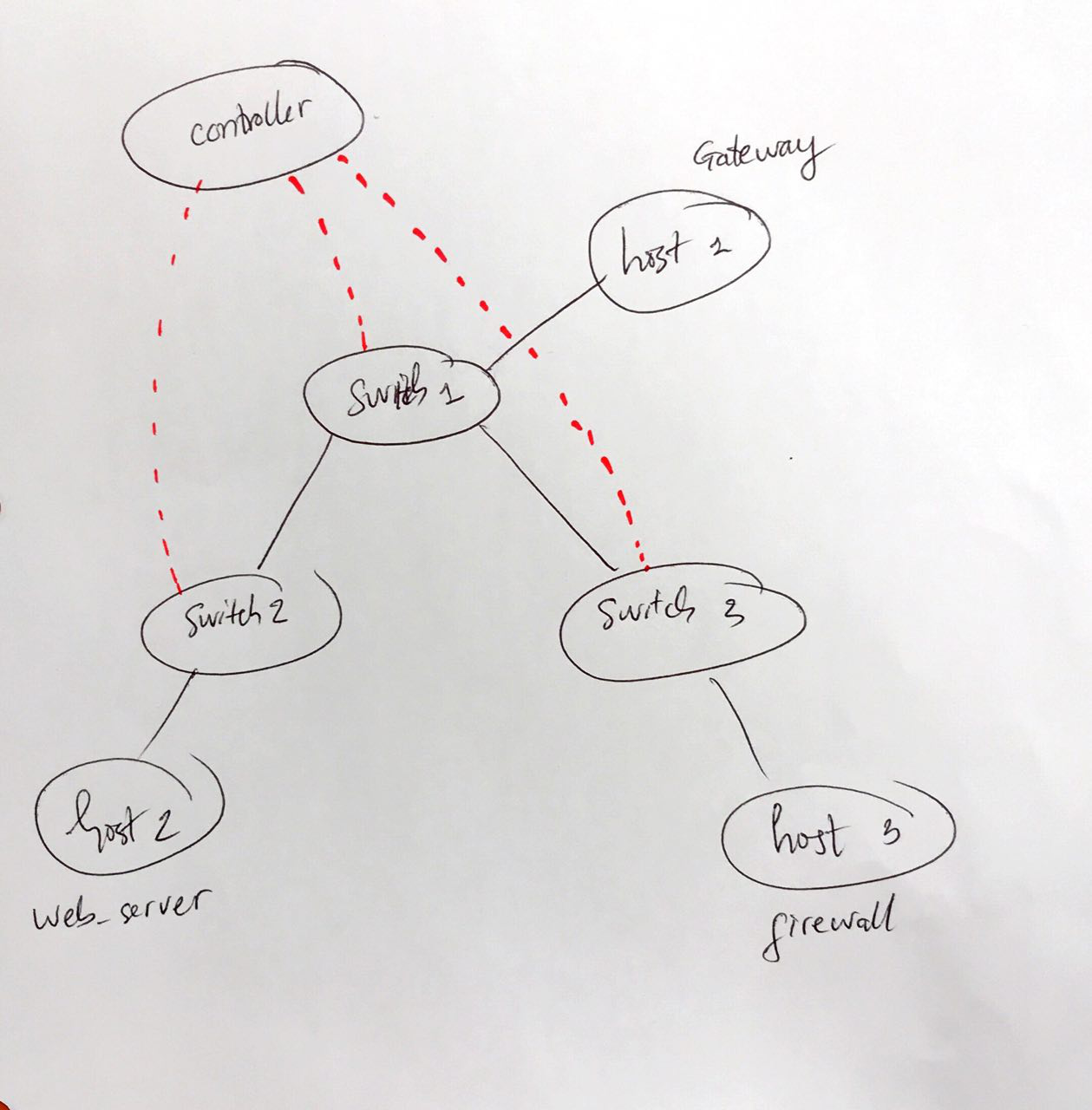


Figure 1 Topology of the network. The red dotted line indicate the connection between switches and the controller.

Host 1, host 2 and host 3 belongs to the same VXLAN with overlay IP addresses are:

1. Host 1 (h1): 192.168.200.11
2. Host 2 (h2): 192.168.200.12
3. Host 3 (h3): 192.168.200.13

Only the gateway – host 1 – is connected to the public network directly via IP address of underlay network.

## Part 1

In this part, the network with above topology is created using scripts from previous lab – **saviOverlay**.

**Remarks**:

1. Get multiple trials and errors with the SAVI platform to successful create the network. When deploying the topology on the platform, it shows the error (code: **500**) that the server cannot create the virtual machine. This could be the resource limitation problem or the RESTful API is not stable?
2. All the machine must be created using the correct security group, otherwise it is not allowed to access the webserver at port 80.

Steps have done in this part:

1. Create the openflow controller virtual machine on savi by this command:

savi-run-server m1.small ece361 netsoft43 netsoft43 netsoft43-controller

1. Setting up the Ryu OpenFlow controller as in guidance notes. This is the controller in the Figure 1.
2. Create the topology by using **saviOverlay deploy** command. Below are the results after all machines are created. It is noted that **controller43** is the virtual machine which runs the controller.

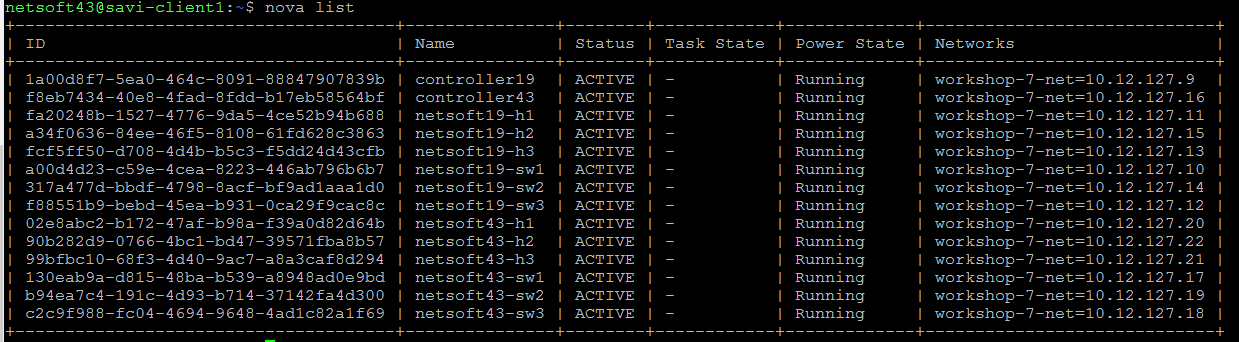
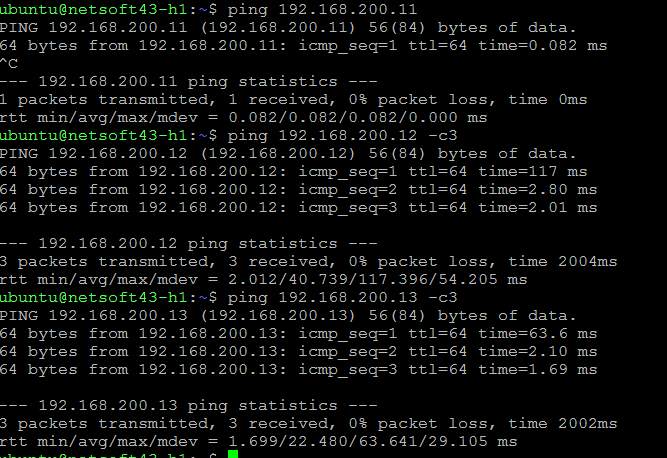


Figure 2 Details of each machine after running "saviOverlay deploy"

1. After successfully deploying the network. It is expected that all of them must be able to ping to each other. Below are the ping results from h1 to h2, and to h3.



## Part 2:

The goal of this part is to force the traffic to go under the overlay network only. In order words, h2 and h3 cannot be accessed via their underlay IP addresses anymore.

**Remarks**:

1. Understand that how the traffic is forced to run through over overlay network only.
2. Learnt to use the *iptables* on Linux to set rules.

All tasks have been done:

1. Configure h1 (the gateway) as a router to perform IP masquerading. All the packets received from h2 and h3 are forwards to public networks. In other words, it means that h2 and h3 can access the *Internet* via the gateway h1.
2. Configure h2 and h3 to use h1 as it only gateway.

Below are screenshot to show the correctness of the settings:

1. *iptables* rules from h1

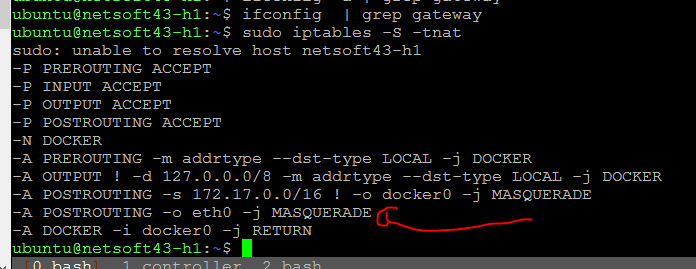


Figure 3 iptables entry (marked with red arrow) which turns h1 into a NAT router

1. Gateway changed at h2

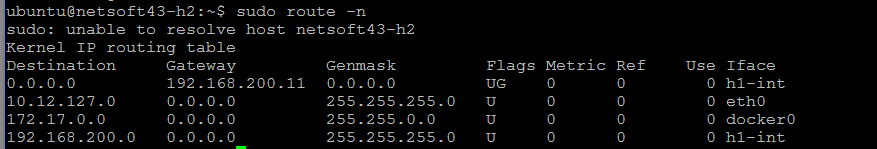


Figure 4 Routing table of h2. It shows that h1 is the default gateway

1. Gateway changed in routing table of h3

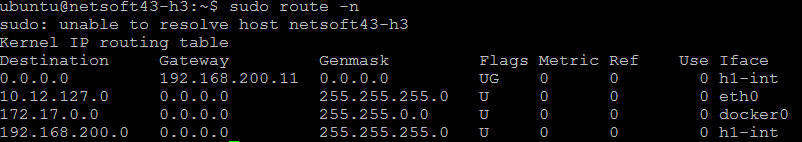


Figure 5 Routing table of h3. It shows that h1 is the default gateway

## Part 3

In this part, WordPress is deployed at webserver - *h2* and Snort firewall is deployed the firewall - *h3*.

**Remarks**:

* Must be careful to set the correct security group for the virtual machine. Otherwise, traffic of port 80 is blocked.

All the tasks have been done:

1. Deploy WordPress service on h2 using 2 docker containers. One container is for the WordPress itself. Another one is for the MySql server.
2. Configure gateway h1 to act as a reverse-proxy for the WordPress service. It is done by using the iptables rule to forward all TCP traffic at port 80 of h1 to the webserver h2 at port 80. This command is used:

sudo iptables -t nat -A PREROUTING -d 10.12.127.20/32 -p tcp -m tcp --dport 80 -j DNAT --to-destination 192.168.200.12:80

1. Deploy Snort at the firewall h3:
   1. Must create two interface for the firewall: one interface for incoming traffic and one for out-going traffic.



Figure 6 Oversimplified illustration of the firewall function.

The commands to create ingress and egress interfaces are:

* Ingress interface: sudo ovs-vsctl add-port h1-br snort-int -- set interface snort-int type=internal
* Egress interface: sudo ovs-vsctl add-port h1-br snort2-int -- set interface snort2-int type=internal

1. Get the port numbers of each interface by this command:

sudo ovs-ofctl show h1-br

Here is the result from the command. The number next to the interface name is the port number.

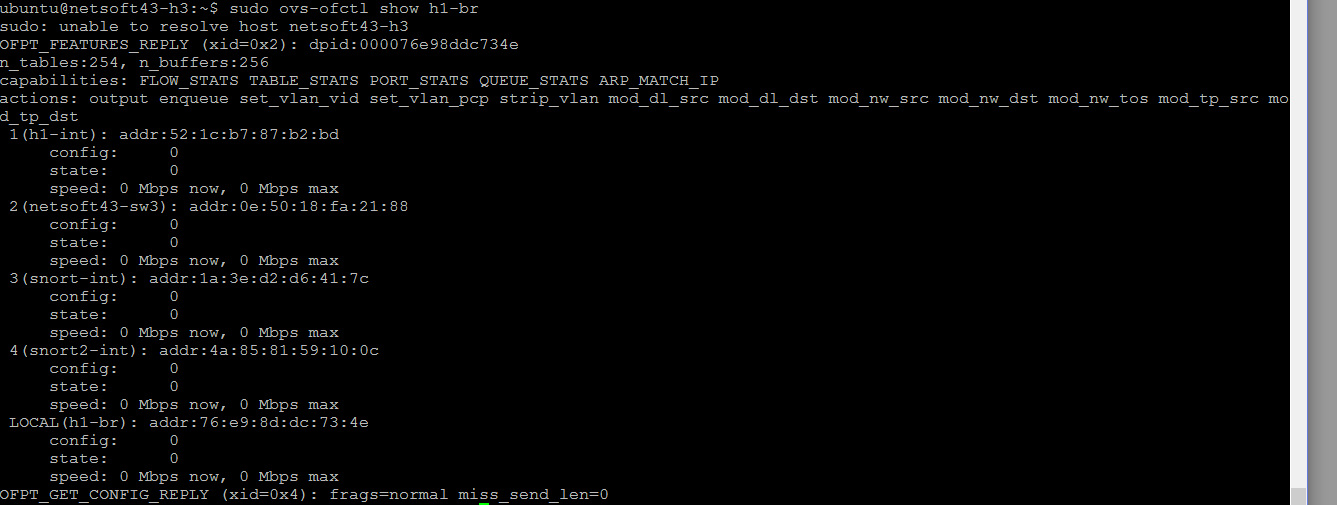


Figure 7 The port numbers of each interface of firewall h3.

In summary, here are the list of ports:

* Internal interface created by the saviOverlay script – *h1-int*: port 1
* VXLAN interface – *netsoft43-sw3*: port 2
* Ingress interface of firewall – *snort-int*: port 3
* Egress interface of firewall – **snort2-int**: port 4

By using these number, flow rules for the OVS are created.

1. Explain each rules:
2. *in\_port={port num of vxlan iface},priority={something high},dl\_dst={mac of internal iface created by saviOverlay},actions=output:{port num of internal iface created by saviOverlay}* 🡺 This rule says that for traffic coming from the VXLAN interface, with destination is the underlay interface MAC address will be forward to the that interface (h1-int).
3. *in\_port={port num of vxlan iface},priority={something high},dl\_dst=01:00:00:00:00:00/01:00:00:00:00:00,actions=output:{port num of internal iface created by saviOverlay }* 🡺 This rule says that for broadcast and multicast traffic from the VXLAN, it will be forward to the underlay interface *h1-int*.
4. *in\_port={port num of internal iface created by saviOverlay},actions=output:{port num of vxlan iface}* 🡺 This rule says that for the traffic from internal *h1-int* interface, it will be forward to the VXLAN interface.
5. *in\_port={port num of vxlan iface},priority={something lower than before},actions=output:{port num of snort ingress iface}* 🡺 This rules say that the traffic from VXLAN interface will be forward to the ingress interface of the firewall.
6. *in\_port={port num of snort egress iface},priority={something lower than before},actions=output:{port num of vxlan iface}* 🡺 This rule says that the traffic from the output interface of the firewall will be forward to the VXLAN interface.

Those rules can be summarized as followed:

* Rule 1: VXLAN 🡪 underlay
* Rule 3: VXLAN 🡨 underlay
* Rule 4: VXLAN 🡪 Ingress of firewall
* Rule 5: VLXAN 🡨 Egress of firewall
* Rule 2: VXLAN broadcast/multicast 🡪 underlay

The direction of 🡪 and 🡨 indicate the traffic from incoming port to outgoing port directions. From here, it is easy to understand why those rules are needed. First, they allow the bi-directional traffic between the VXLAN and the underlay network. Second, they force the bidirectional traffic in and out the firewall server. Lastly, it allows the broadcast and multicast from the VXLAN to the outside networks.

In details, those rules are defined as:

1. sudo ovs-ofctl add-flow h1-br in\_port=2,priority=60000,dl\_dst=52:1c:b7:87:b2:bd,actions=output:1

2. sudo ovs-ofctl add-flow h1-br in\_port=2,priority=60000,dl\_dst=01:00:00:00:00:00/01:00:00:00:00:00,actions=output:1

3. sudo ovs-ofctl add-flow h1-br in\_port=1,actions=output:2

4. sudo ovs-ofctl add-flow h1-br in\_port=2,priority=50000,actions=output:3

5. sudo ovs-ofctl add-flow h1-br in\_port=4,priority=50000,actions=output:2

Here are the results of *sudo ovs-ofctl dump-flows h1-br* at h3 after installing rules.

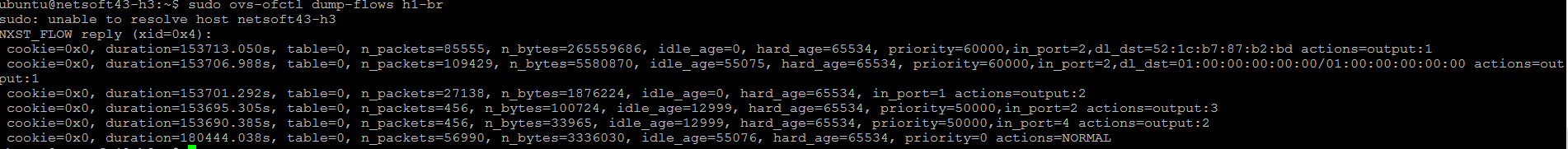


Figure 8 Flow rules at h3

1. **Snort rule:**

Snort service acts as a firewall in this network. To enable it, rules to filter the data must be defined. In general, the snort syntax rule format is like this:



Figure 9 Basic snort rule format with example values

This is the above rule written in plain form:

reject tcp any any -> any 80 (content:"inject"; nocase; msg:"accessed forbidden pages!!"; sid:5000000;)

## Part 4:

In this part, the OpenFlow controller uses flow rules to re-direct all request to web-server h2 to the firewall h3. By doing this, the webserver is hidden from the real word access and the firewall works as the proxy to the web service, it also allows or forbid certain requests to the webserver.

**Remarks**:

* The incoming package to the webserver does not contain the mac address of the real request machine (e.g., request originated outside of the VXLAN), it only acknowledges the mac address of the gateway.
* When the request is blocked by the firewall, basically, the web browser’s screen is blank.

All tasks have been done:

1. Collect the mac address of all hosts: h1 – gateway, h2 - webserver, and h3 – firewall server, by using this command:

ifconfig

1. Using the RYU api to get the DPIDs of each switch then identify their name based on the connected hosts. For instance, it is known by design that host h1 is connect to switch sw1 so that the retrieved DPID is of sw1. It also indicates which port the host is connected to the switch.

Commands used: ryu\_ofctl.getMacIngressPort(mac\_address)

1. Identify all in and out port numbers inside the topology using the links getting from RYU. Below is the updated figure of the topology.

Command used: ryu\_ofctl.listLinks()

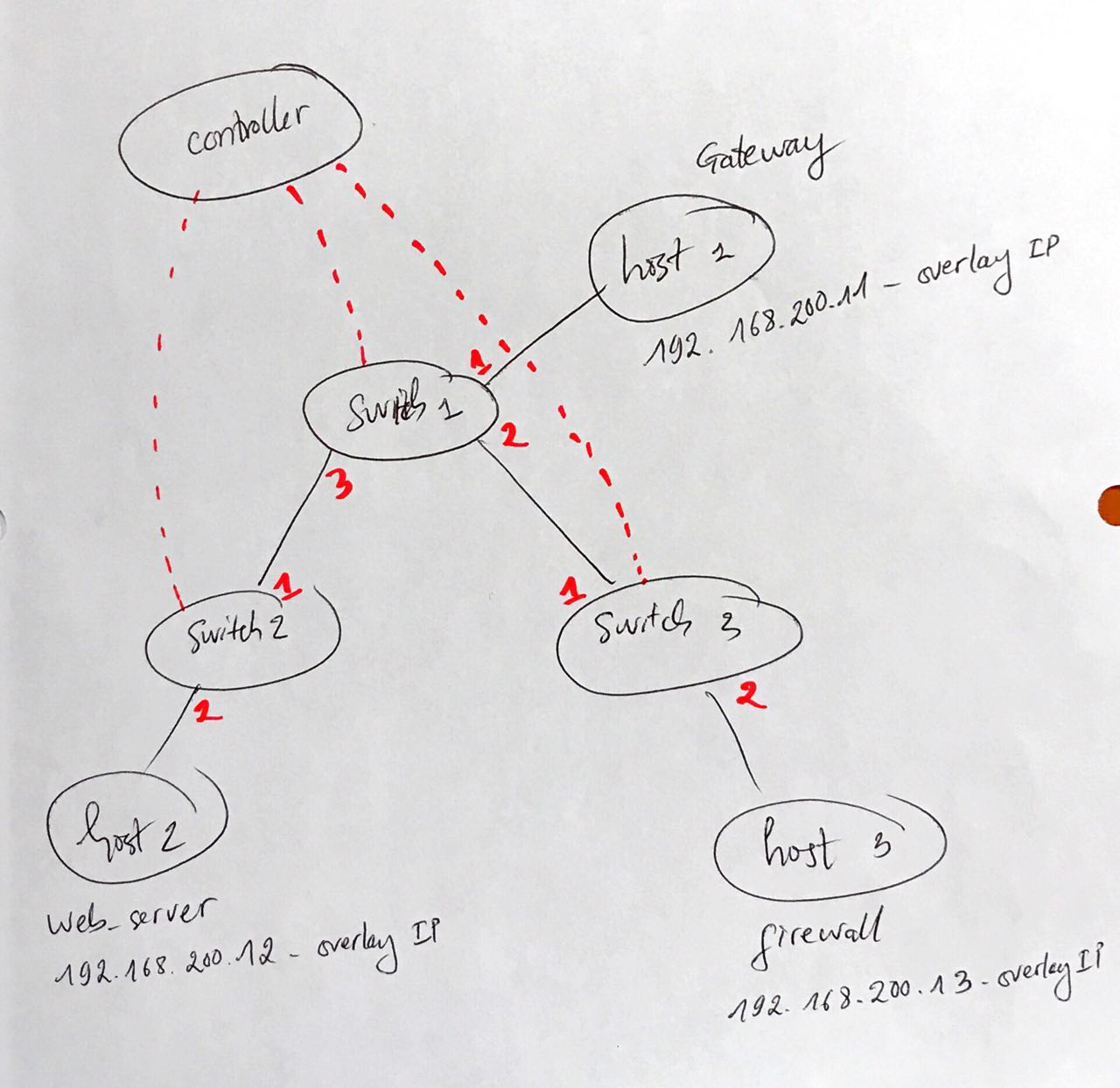


Figure 10 Network topology with port numbers in red.

1. Based on the above figure, here is the visualization of traffic flow (in orange line) showing how webserver – host 2 - is accessed from the outside (i.e., through the gateway).



Figure 11 Traffic flow from gateway to webserver, marked in orange color.

The traffic will follow this path: gateway 🡪 switch 1 🡪 switch 3 🡪 firewall 🡪 switch 3 🡪 switch 1 🡪 switch 2 🡪 webserver 🡪 switch 2 🡪 switch 1 🡪 switch 3 🡪 firewall 🡪 switch 3 🡪 switch 1 🡪 gateway. According to this route, the flow rules are defined and install to each switch. The code to define and insert rules to the switches are included in the Appendix part.

Here are the contents of the OVS flow-table after those flow rules are deployed:

* Flow rules at switch 1:

*ubuntu@netsoft43-sw1:~$ sudo ovs-ofctl dump-flows h1-br*

*sudo: unable to resolve host netsoft43-sw1 NXST\_FLOW reply (xid=0x4):*

*cookie=0x0, duration=125852.202s, table=0, n\_packets=278250, n\_bytes=14190750, idle\_age=400, hard\_age=65534, priority=65535,dl\_dst=01:80:c2:00:00:0e,dl\_type=0x88cc actions=CONTROLLER:51*

*cookie=0x0, duration=935.855s, table=0, n\_packets=107, n\_bytes=17178, idle\_age=351, priority=50000,tcp,in\_port=1,dl\_dst=6e:1d:53:54:d9:e7,tp\_dst=80 actions=output:2*

*cookie=0x0, duration=846.001s, table=0, n\_packets=107, n\_bytes=8253, idle\_age=351, priority=50000,tcp,in\_port=2,dl\_dst=6e:1d:53:54:d9:e7,tp\_dst=80 actions=output:3*

*cookie=0x0, duration=790.595s, table=0, n\_packets=0, n\_bytes=0, idle\_age=790, priority=50000,tcp,in\_port=3,dl\_src=6e:1d:53:54:d9:e7,tp\_dst=80 actions=output:2*

*cookie=0x0, duration=707.241s, table=0, n\_packets=0, n\_bytes=0, idle\_age=707, priority=50000,tcp,in\_port=2,dl\_src=6e:1d:53:54:d9:e7,tp\_dst=80 actions=output:1*

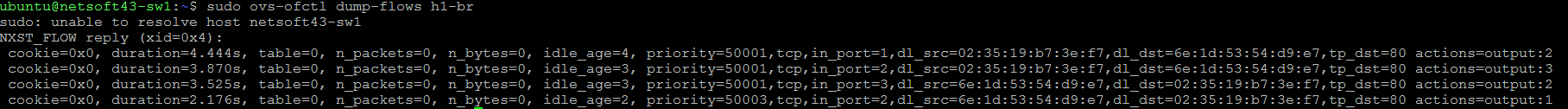


Figure 12 Flow rules at switch 1

* Flow rules at switch 2:

*ubuntu@netsoft43-sw2:~$ sudo ovs-ofctl dump-flows h1-br*

*sudo: unable to resolve host netsoft43-sw2 NXST\_FLOW reply (xid=0x4):*

*cookie=0x0, duration=125812.642s, table=0, n\_packets=139072, n\_bytes=7092672, idle\_age=463, hard\_age=65534, priority=65535,dl\_dst=01:80:c2:00:00:0e,dl\_type=0x88cc actions=CONTROLLER:51*

*cookie=0x0, duration=887.318s, table=0, n\_packets=107, n\_bytes=8253, idle\_age=413, priority=50000,tcp,in\_port=1,dl\_dst=6e:1d:53:54:d9:e7,tp\_dst=80 actions=output:2*

*cookie=0x0, duration=867.878s, table=0, n\_packets=0, n\_bytes=0, idle\_age=867, priority=50000,tcp,in\_port=2,dl\_src=6e:1d:53:54:d9:e7,tp\_dst=80 actions=output:1*

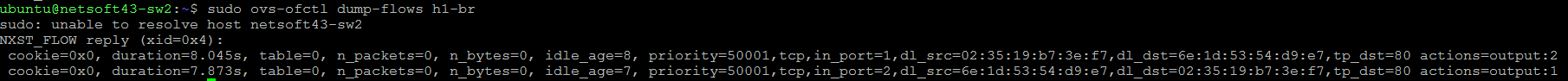


Figure 13 Flow rules at switch 2

* Flow rules at switch 3:

*ubuntu@netsoft43-sw3:~$ sudo ovs-ofctl dump-flows h1-br*

*sudo: unable to resolve host netsoft43-sw3 NXST\_FLOW reply (xid=0x4):*

*cookie=0x0, duration=125941.004s, table=0, n\_packets=139176, n\_bytes=7097976, idle\_age=498, hard\_age=65534, priority=65535,dl\_dst=01:80:c2:00:00:0e,dl\_type=0x88cc actions=CONTROLLER:51*

*cookie=0x0, duration=967.026s, table=0, n\_packets=107, n\_bytes=17178, idle\_age=448, priority=50000,tcp,in\_port=1,dl\_dst=6e:1d:53:54:d9:e7,tp\_dst=80 actions=output:2*

*cookie=0x0, duration=957.558s, table=0, n\_packets=107, n\_bytes=8253, idle\_age=448, priority=50000,tcp,in\_port=2,dl\_dst=6e:1d:53:54:d9:e7,tp\_dst=80 actions=output:1*

*cookie=0x0, duration=828.115s, table=0, n\_packets=0, n\_bytes=0, idle\_age=828, priority=50000,tcp,in\_port=1,dl\_src=6e:1d:53:54:d9:e7,tp\_dst=80 actions=output:2*

*cookie=0x0, duration=819.300s, table=0, n\_packets=0, n\_bytes=0, idle\_age=819, priority=50000,tcp,in\_port=2,dl\_src=6e:1d:53:54:d9:e7,tp\_dst=80 actions=output:1*

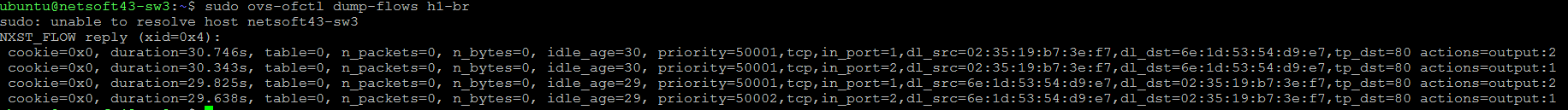


Figure 14 Flow rules at switch 3

1. Below are the screenshot of the browser after all rules are deployed.

* When accessing the home page via: [**http://localhost/?inject**](http://localhost/?inject)before deploying the rules



Figure 15 Webpage before deploying rules at switches

* When accessing the page with url [**http://localhost/?inject**](http://localhost/?inject)after deploying the rules

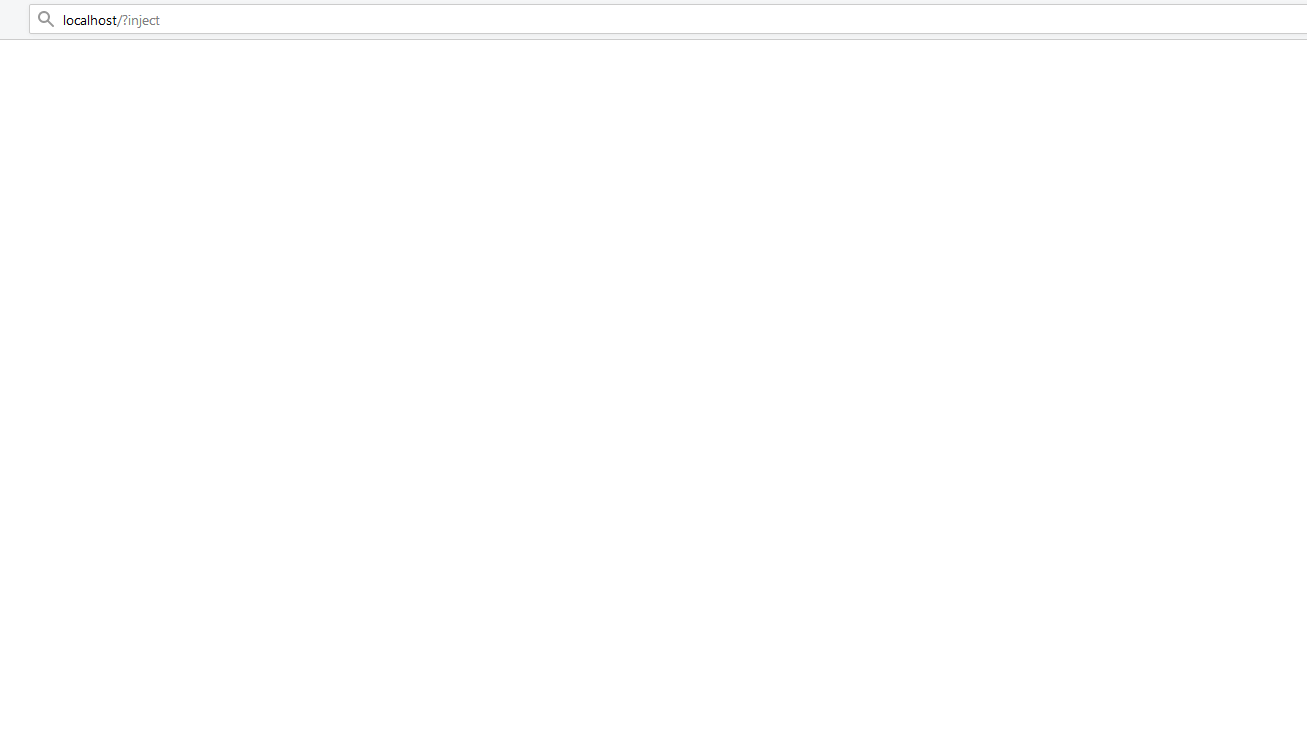


Figure 16 Webpage after deploying rules at switches

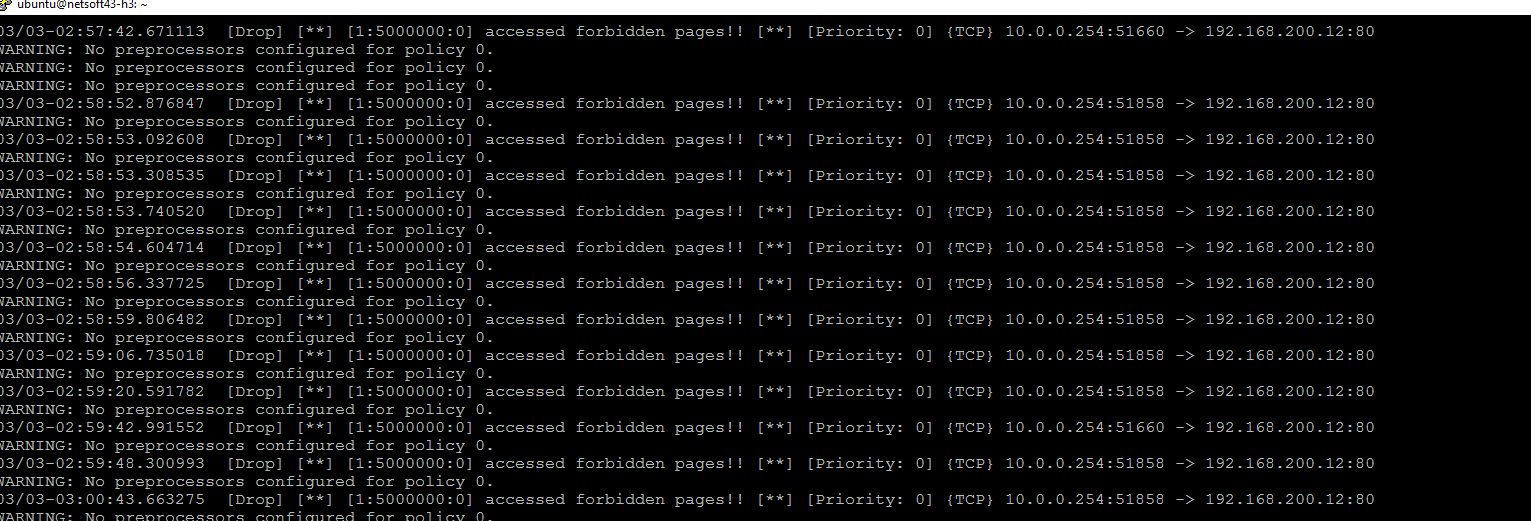


Figure 17 Message from Snort showed that the "inject" content is forbidden to access.

## Appendix:

import ryu\_ofctl  
  
def createFlow(dl\_src, dl\_dst, in\_port, out\_port, switch\_dpid, priority=50001):  
 *"""  
 create the flow and insert to the switch* ***:param*** *dl\_src: source mac, string* ***:param*** *dl\_dst: dest mac, string* ***:param*** *in\_port: in port, integer* ***:param*** *out\_port: out port for the action, integer* ***:return****: the FlowEntry object with the associate action of out\_port  
 """* flow = ryu\_ofctl.FlowEntry()  
 # for the constant  
 flow.priority = priority  
 # http protocol: tcp and port 80  
 flow.dl\_type = 0x800 # IPV4  
 flow.nw\_proto = 0x6 # TCP  
 flow.tp\_dst = 80 # webserver port  
 # for src and dest mac  
 flow.dl\_src = dl\_src  
 flow.dl\_dst = dl\_dst  
 # for in and out\_port  
 flow.in\_port = in\_port  
 act = ryu\_ofctl.OutputAction(out\_port)  
 # add the action to the flow  
 flow.addAction(act)  
 # insert the flow to the switch  
 print("the flow is")  
 flow.printMatch()  
 print(ryu\_ofctl.insertFlow(switch\_dpid, flow))  
 return flow  
  
# list the mac of each machine  
mac1 = "02:35:19:b7:3e:f7" # h1  
mac2 = "6e:1d:53:54:d9:e7" # h2  
mac3 = "52:1c:b7:87:b2:bd" # h3  
  
# list of dpids of the switches  
sw1 = '0000daed57ae3949'  
sw2 = '00002ad84265224b'  
sw3 = '00006e3e0e07ca45'  
  
# to get the ports  
ryu\_ofctl.getMacIngressPort(mac1)  
ryu\_ofctl.getMacIngressPort(mac2)  
ryu\_ofctl.getMacIngressPort(mac3)  
ryu\_ofctl.listLinks()  
  
  
# writing the flow for the f\*\* switches  
# building the topology diagram by checking the links  
  
# From h1 to h3: GW (gateway) to FW (firewall)  
  
h1\_s1 = createFlow(dl\_src=mac1, dl\_dst=mac2, in\_port=1, out\_port=2, switch\_dpid=sw1)  
s1\_s3 = createFlow(dl\_src=mac1, dl\_dst=mac2, in\_port=1, out\_port=2, switch\_dpid=sw3)  
# s2\_h3: no need a rules  
  
#createFlow(dl\_src=None, dl\_dst=, in\_port=, out\_port=, switch\_dpid=)  
# From h3 to h2: FW to WP - wordpress  
h3\_s3 = createFlow(dl\_src=mac1, dl\_dst=mac2, in\_port=2, out\_port=1, switch\_dpid=sw3)  
s3\_s1 = createFlow(dl\_src=mac1, dl\_dst=mac2, in\_port=2, out\_port=3, switch\_dpid=sw1)  
s1\_s2 = createFlow(dl\_src=mac1, dl\_dst=mac2, in\_port=1, out\_port=2, switch\_dpid=sw2)  
  
# From h2 to h3: WP -> FW  
h2\_s2 = createFlow(dl\_src=mac2, dl\_dst=mac1, in\_port=2, out\_port=1, switch\_dpid=sw2)  
s2\_s1 = createFlow(dl\_src=mac2, dl\_dst=mac1, in\_port=3, out\_port=2, switch\_dpid=sw1)  
s1\_s3 = createFlow(dl\_src=mac2, dl\_dst=mac1, in\_port=1, out\_port=2, switch\_dpid=sw3)  
  
# From h3 to h1: FW to GW  
h3\_s3 = createFlow(priority=50002, dl\_src=mac2, dl\_dst=mac1, in\_port=2, out\_port=1, switch\_dpid=sw3) # higher  
s3\_s1 = createFlow(priority=50003,dl\_src=mac2, dl\_dst=mac1, in\_port=2, out\_port=1, switch\_dpid=sw1)  
  
# to delete all flows  
#ryu\_ofctl.deleteAllFlows(sw1)  
#ryu\_ofctl.deleteAllFlows(sw2)  
#ryu\_ofctl.deleteAllFlows(sw3)