HKA Assignment 5

Hayder Al Rubaye
T00613298

INTRODUCTION TO OPENSWITCH WITH FLOODLIGHT CONTROLLER

We first login to the GENI using the NCSA account. As seen in Figures 1-3

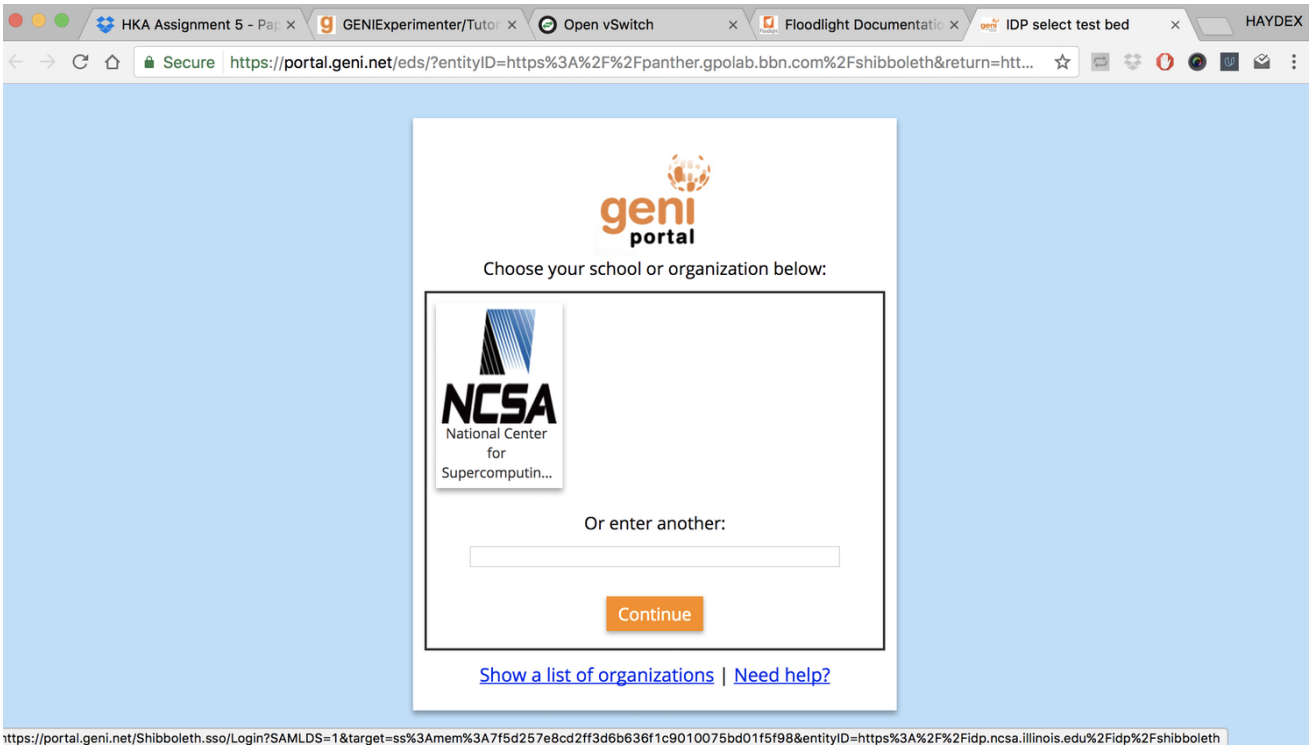

Figure 1

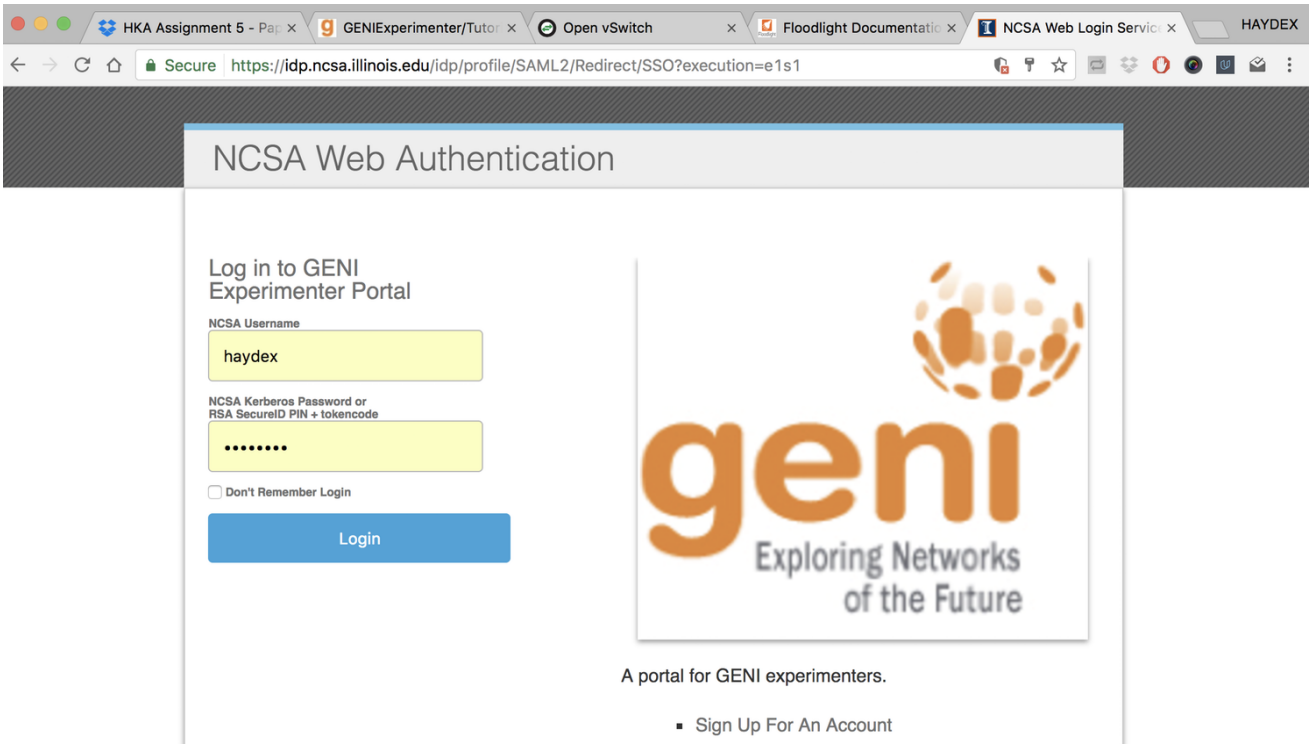

Figure 2

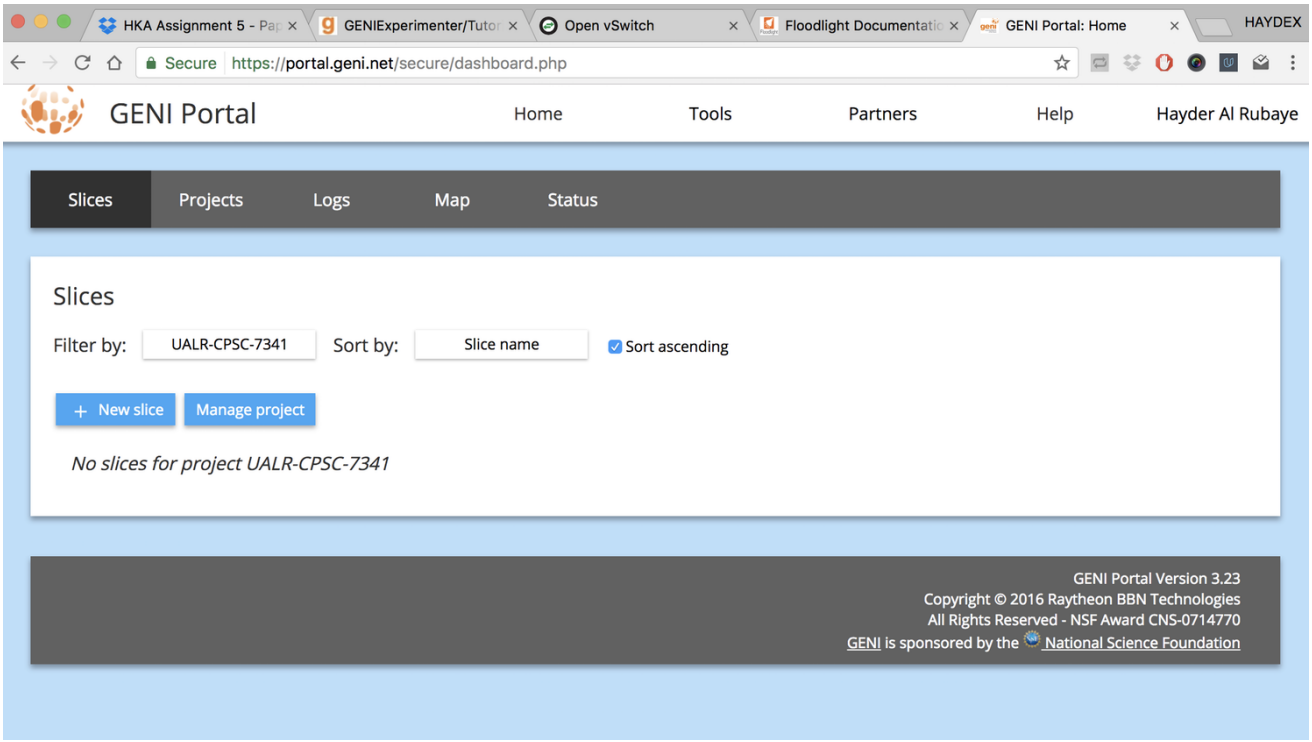

Figure 3

Step 1: We obtain the resources.

a) We *create the controller slice and reserve a VM that runs OpenFlow controller by loading its RSpec file from the given link. (Figures 4-8)*

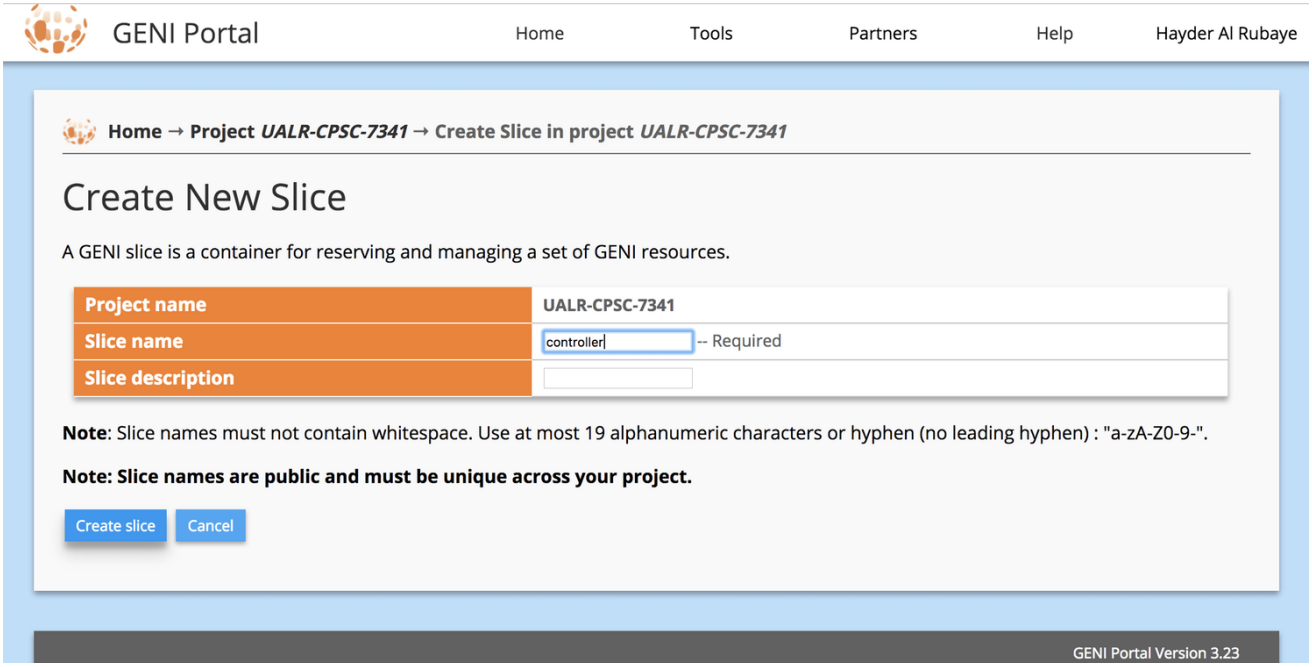

Figure 4 Controller Slice

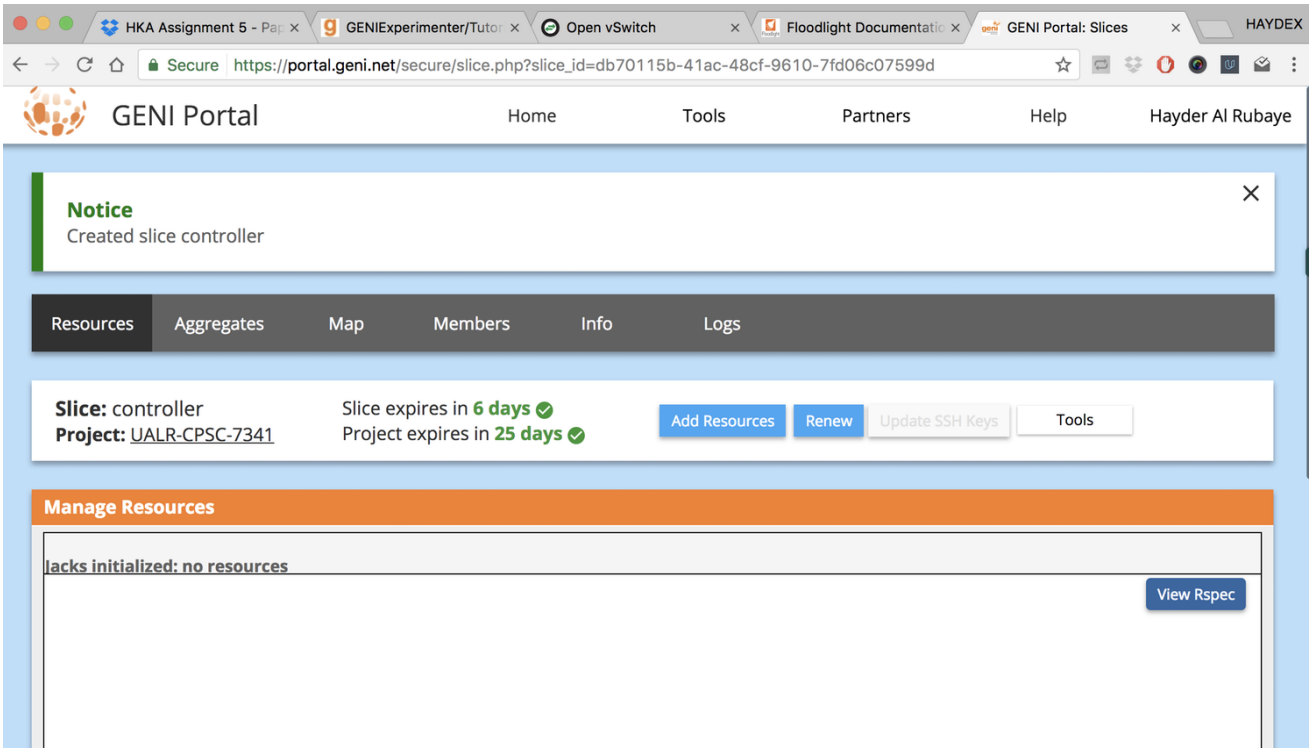

Figure 5

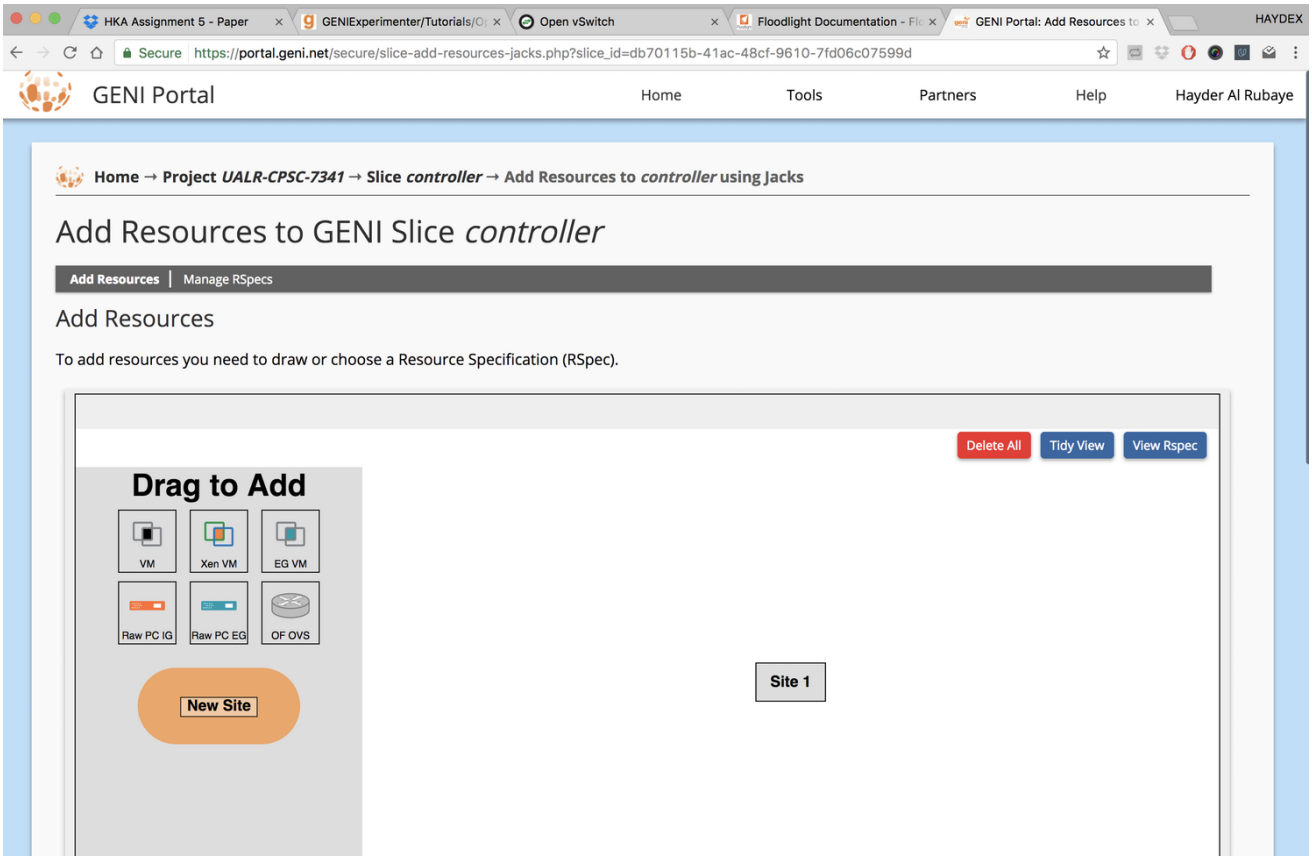

Figure 6

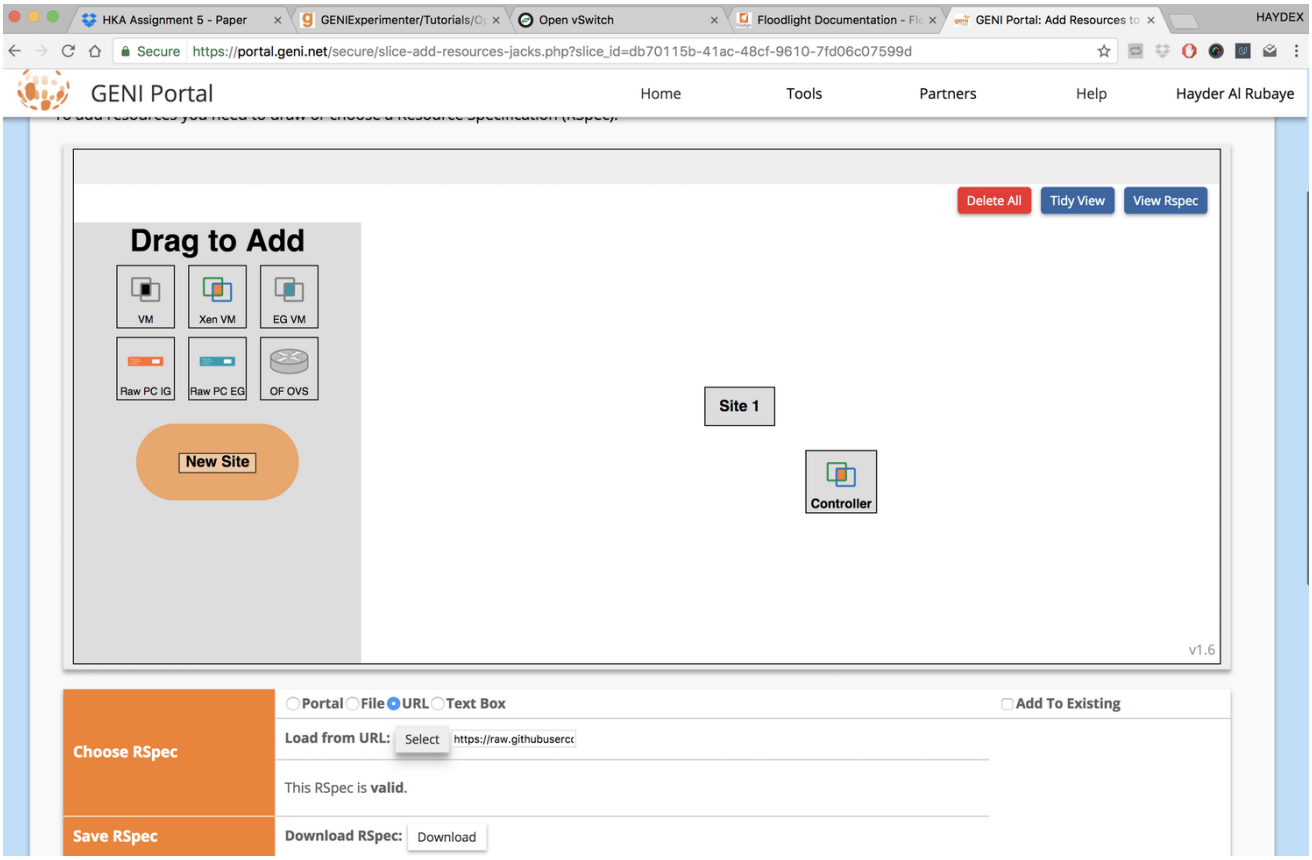

Figure 7 RSpec loaded

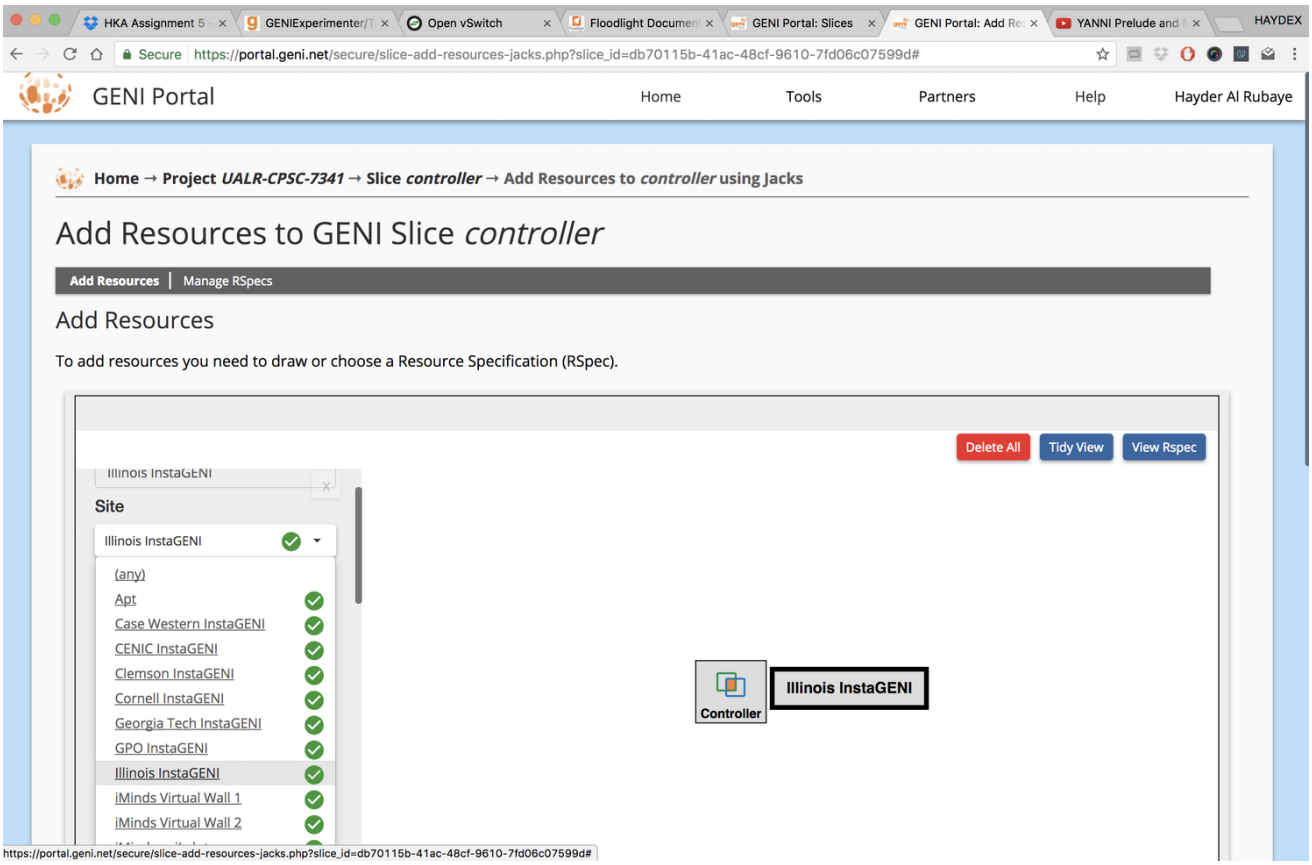

Figure 8

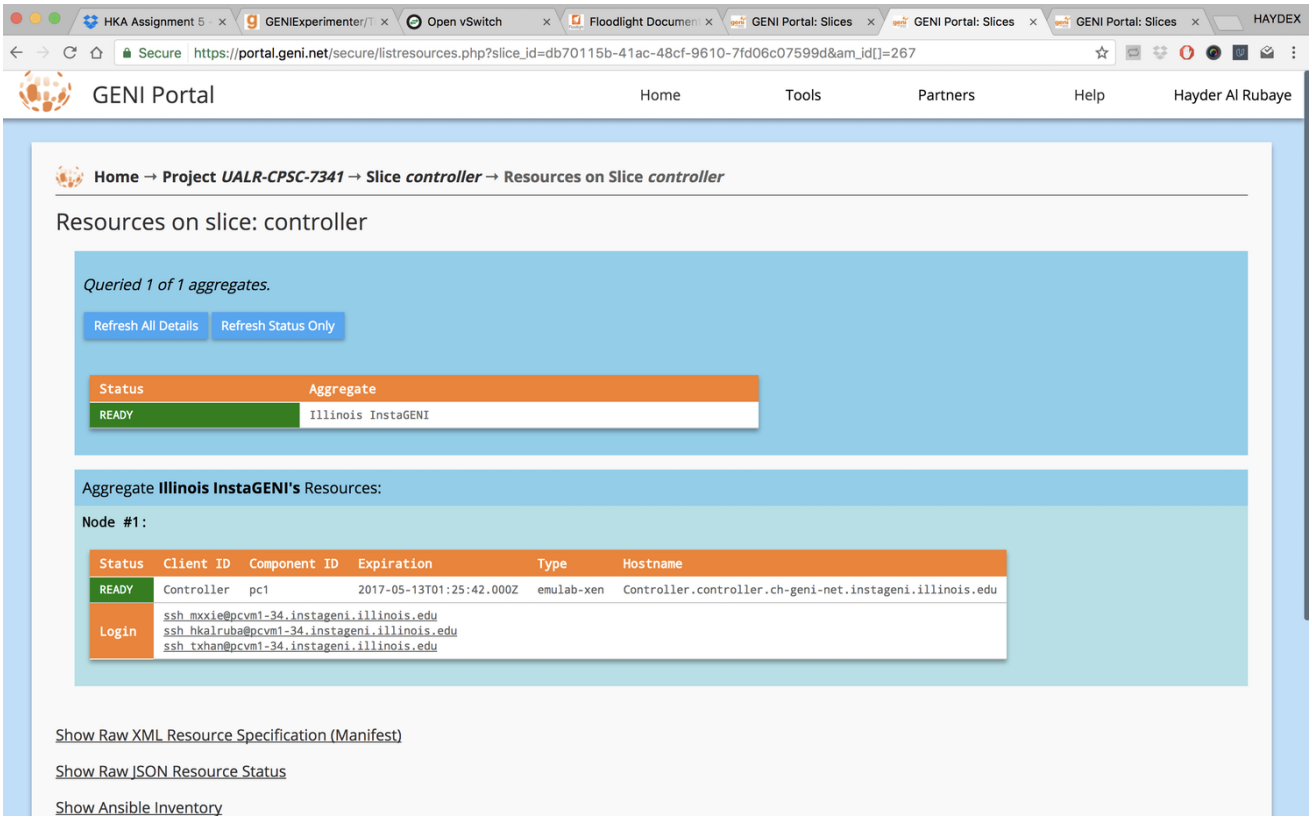

Figure 8 Continued Resources allocated successfully

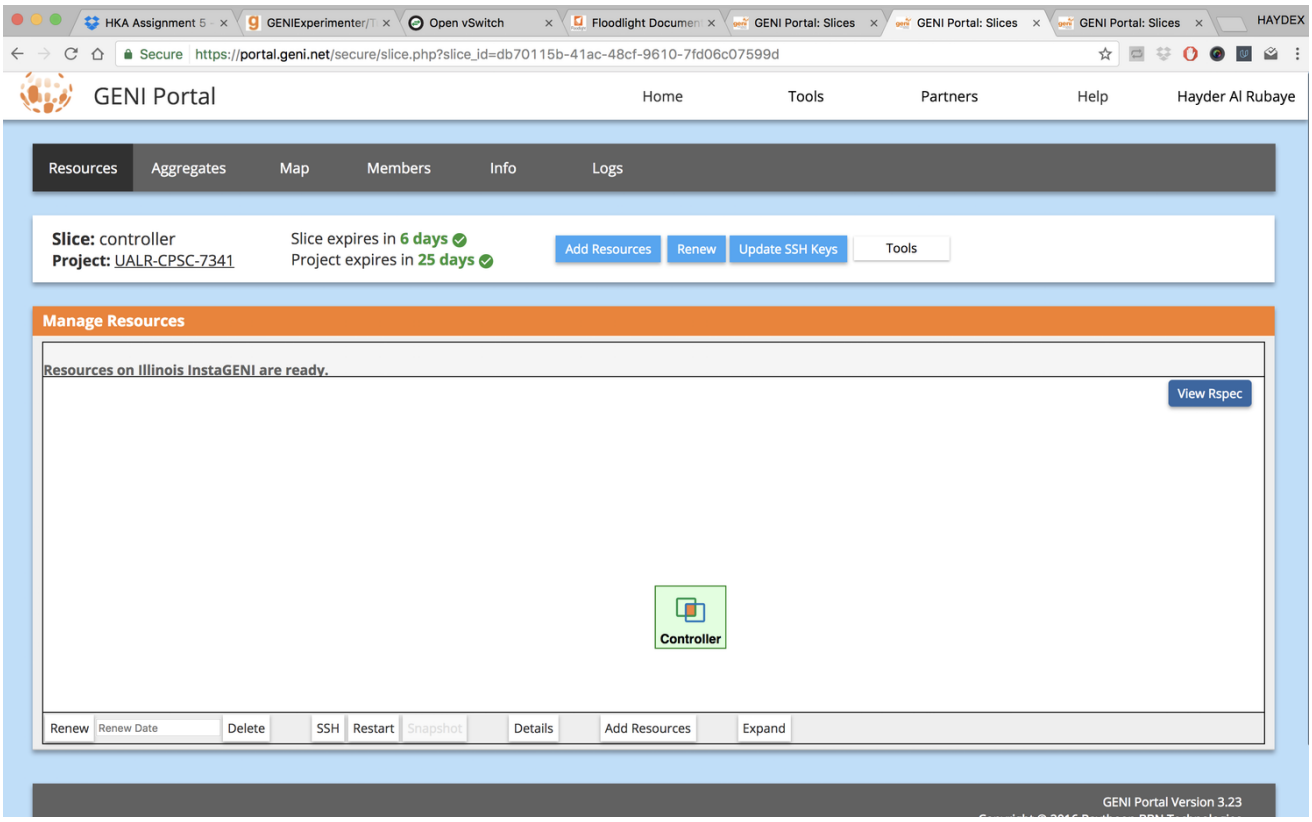

Figure 8 Continued

b) We *create the network topology slice* which includes a VM with OVS installed and the endpoint hosts *by loading its RSpec file from the given link. (Figures 9-13)*

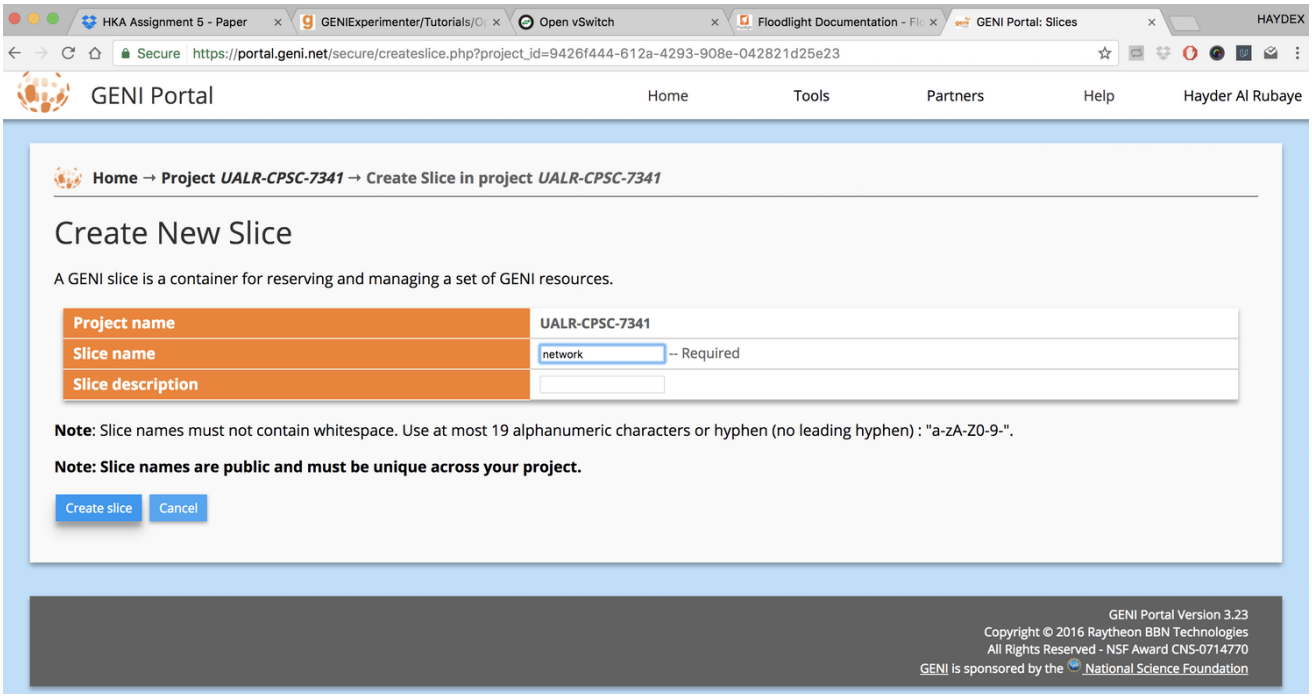

Figure 9

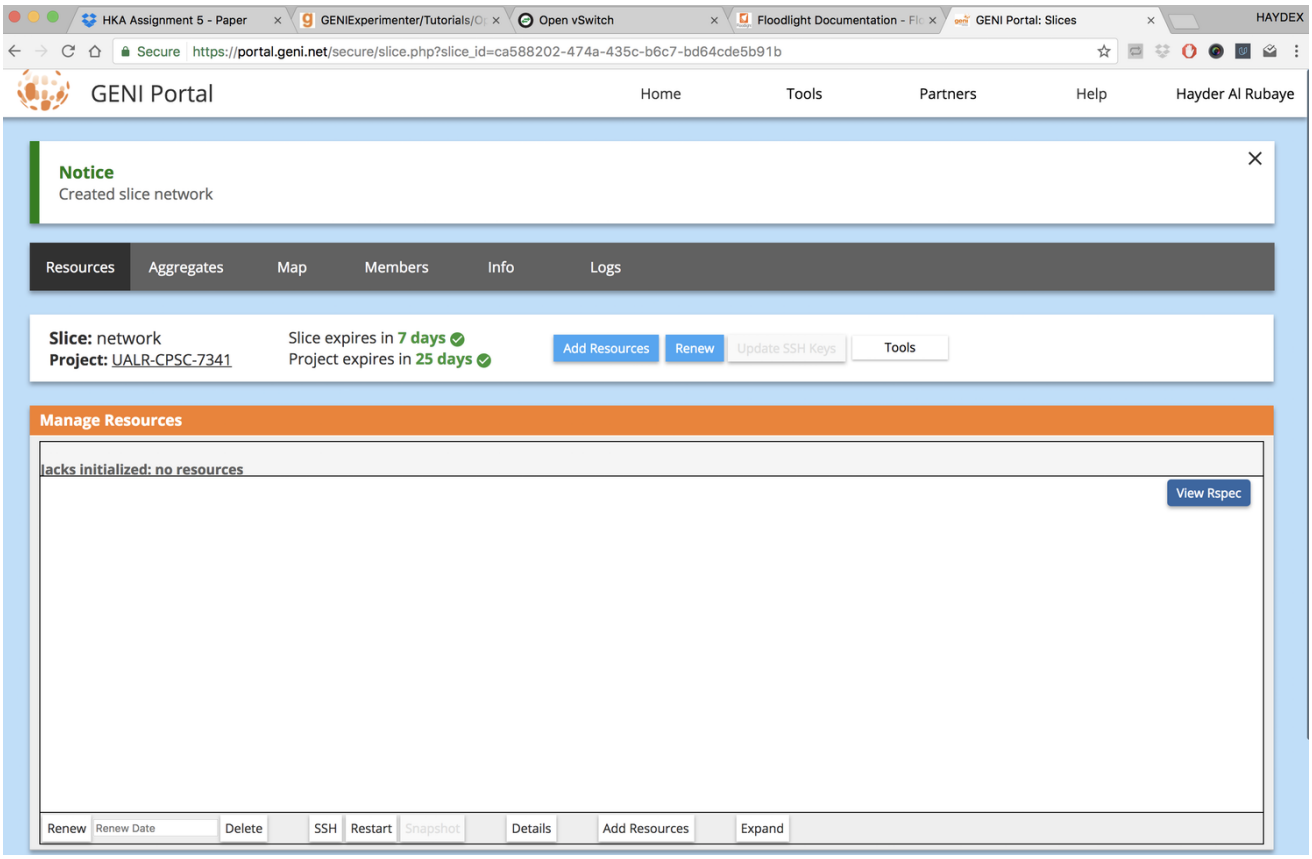

Figure 10

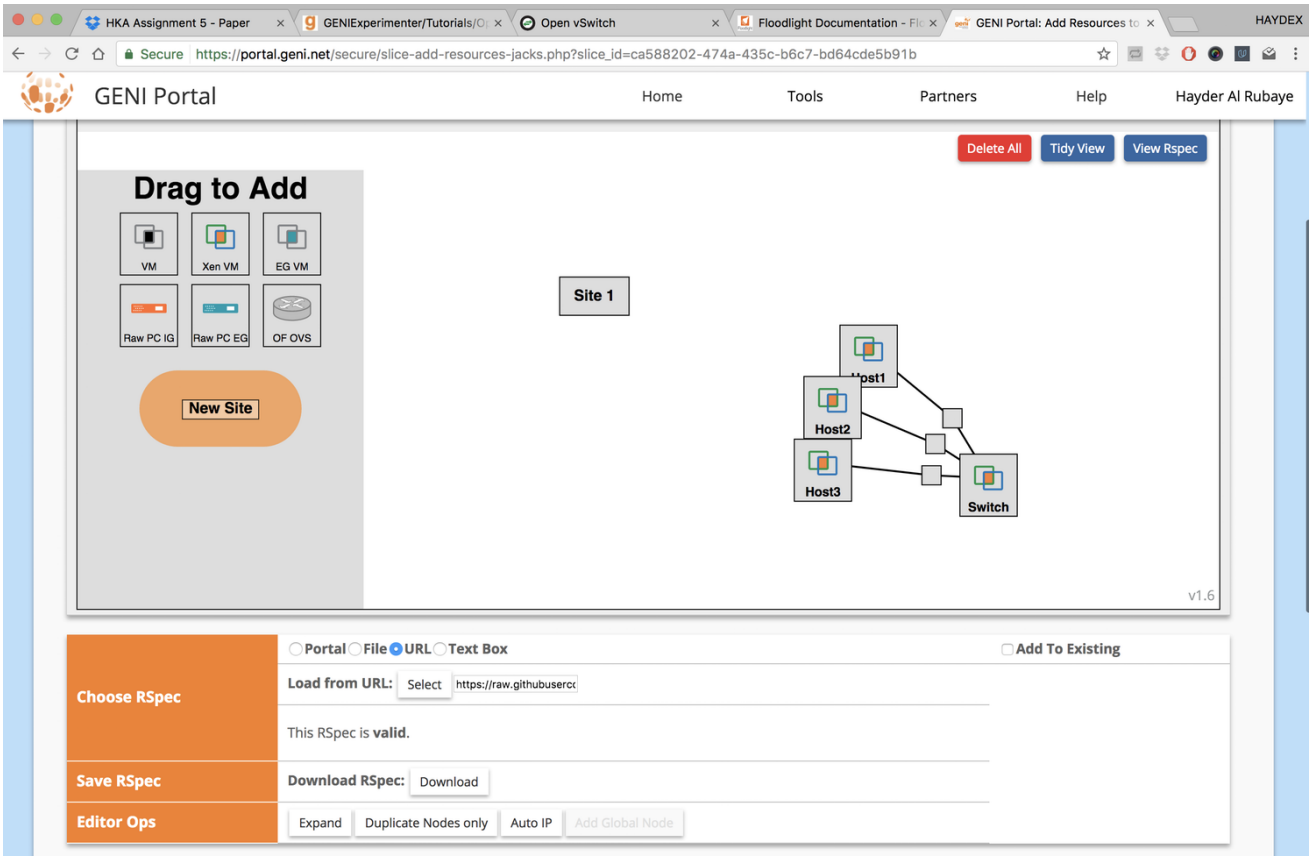

Figure 11 Network RSpec is loaded

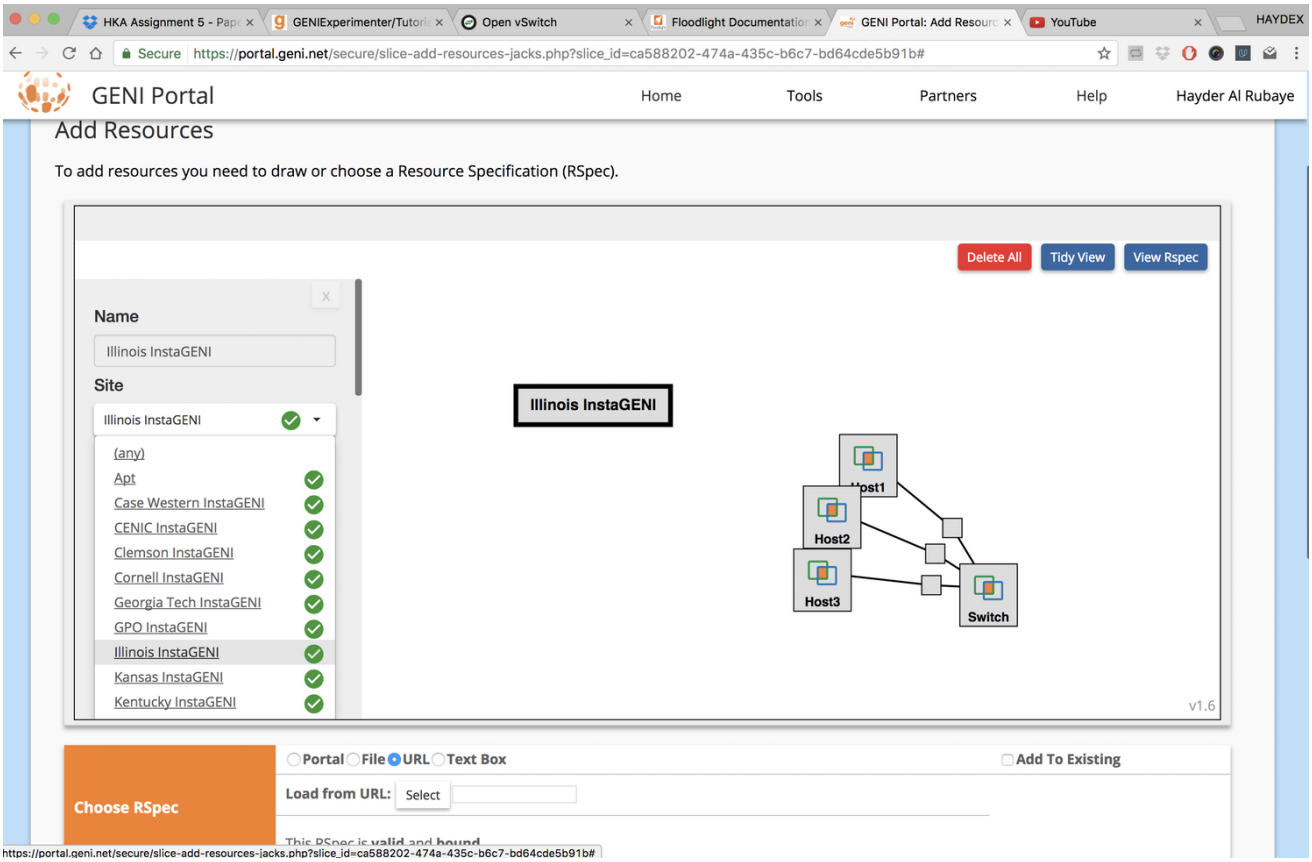

Figure 12 Reserving the resources from Illinois InstaGENI

GENI Portal Home Tools Partners Help Hayder Al Rubaye

Home → Project *UALR-CPSC-7341* → Slice *network* → Resources on Slice *network*

Resources on slice: network

Queried 1 of 1 aggregates.

Refresh All Details | Refresh Status Only

Status	Aggregate
READY	Illinois InstaGENI

Aggregate **Illinois InstaGENI's** Resources:

Node #1:

Status	Client ID	Component ID	Expiration	Type	Hostname
READY	Host1	pc1	2017-05-13T01:34:16.000Z	emulab-xen	Host1.network.ch-geni-net.instageni.illinois.edu
Login	ssh mxxie@pc1.instageni.illinois.edu -p 25074 ssh hkalruba@pc1.instageni.illinois.edu -p 25074 ssh txhan@pc1.instageni.illinois.edu -p 25074				

Interfaces		MAC	Layer 3
interface-6	pc1:lo0	0265d7e95af5	ipv4: 10.0.0.1

Node #2:

Status	Client ID	Component ID	Expiration	Type	Hostname
READY	Host2	pc1	2017-05-13T01:34:16.000Z	emulab-xen	Host2.network.ch-geni-net.instageni.illinois.edu
Login	ssh mxxie@pc1.instageni.illinois.edu -p 25075 ssh hkalruba@pc1.instageni.illinois.edu -p 25075 ssh txhan@pc1.instageni.illinois.edu -p 25075				

Interfaces		MAC	Layer 3
interface-8	pc1:lo0	02c33308c094	ipv4: 10.0.0.2

Node #3:

Status	Client ID	Component ID	Expiration	Type	Hostname
READY	Host3	pc1	2017-05-13T01:34:16.000Z	emulab-xen	Host3.network.ch-geni-net.instageni.illinois.edu
Login	ssh mxxie@pc1.instageni.illinois.edu -p 25076 ssh hkalruba@pc1.instageni.illinois.edu -p 25076 ssh txhan@pc1.instageni.illinois.edu -p 25076				

Interfaces		MAC	Layer 3
interface-10	pc1:lo0	02939a4e7789	ipv4: 10.0.0.3

Node #4:

Status	Client ID	Component ID	Expiration	Type	Hostname
READY	Switch	pc1	2017-05-13T01:34:16.000Z	emulab-xen	Switch.network.ch-geni-net.instageni.illinois.edu
Login	ssh mxxie@pc1.instageni.illinois.edu -p 25077 ssh hkalruba@pc1.instageni.illinois.edu -p 25077 ssh txhan@pc1.instageni.illinois.edu -p 25077				

Interfaces		MAC	Layer 3
interface-7	pc1:lo0	025f8c2737e6	ipv4: 10.10.1.2
interface-9	pc1:lo0	02731746ece7	ipv4: 10.10.2.2
interface-11	pc1:lo0	02ab11b7eeec	ipv4: 10.10.3.2

Link #1:

Client ID	Endpoint #0	Endpoint #1
link-3	interface-6	interface-7

Link #2:

Client ID	Endpoint #0	Endpoint #1
link-4	interface-8	interface-9

Link #3:

Client ID	Endpoint #0	Endpoint #1
link-5	interface-10	interface-11

Show Raw XML Resource Specification (Manifest)

Show Raw JSON Resource Status

Show Ansible Inventory

Back to All slices
Back to Slice *network*

Figure 13 Resources allocated successfully

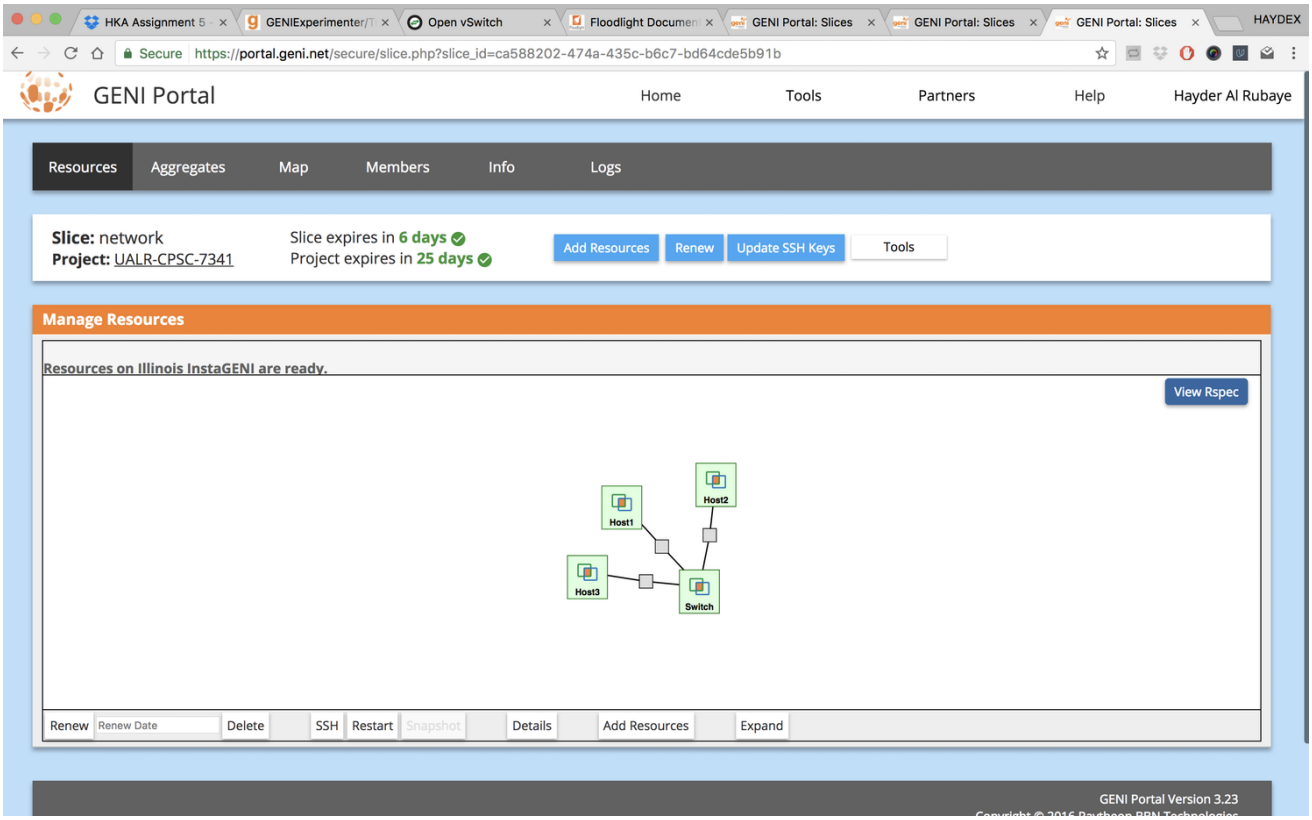

Figure 13 Continued

Step 2: Configuring the Open vSwitch

Although OVS is installed and initialized on the host that acts as a software switch, it has not been configured yet. There are two configuration operations that need to be executed: 1) configure the software switch with the interfaces as ports and 2) point the OVS switch to an OpenFlow controller.

2a. Configure the Software Switch (OVS Window)

i. We login to the OVS host by using the GENI Desktop, then we connect through SSH in the browser (Figures 14-16)

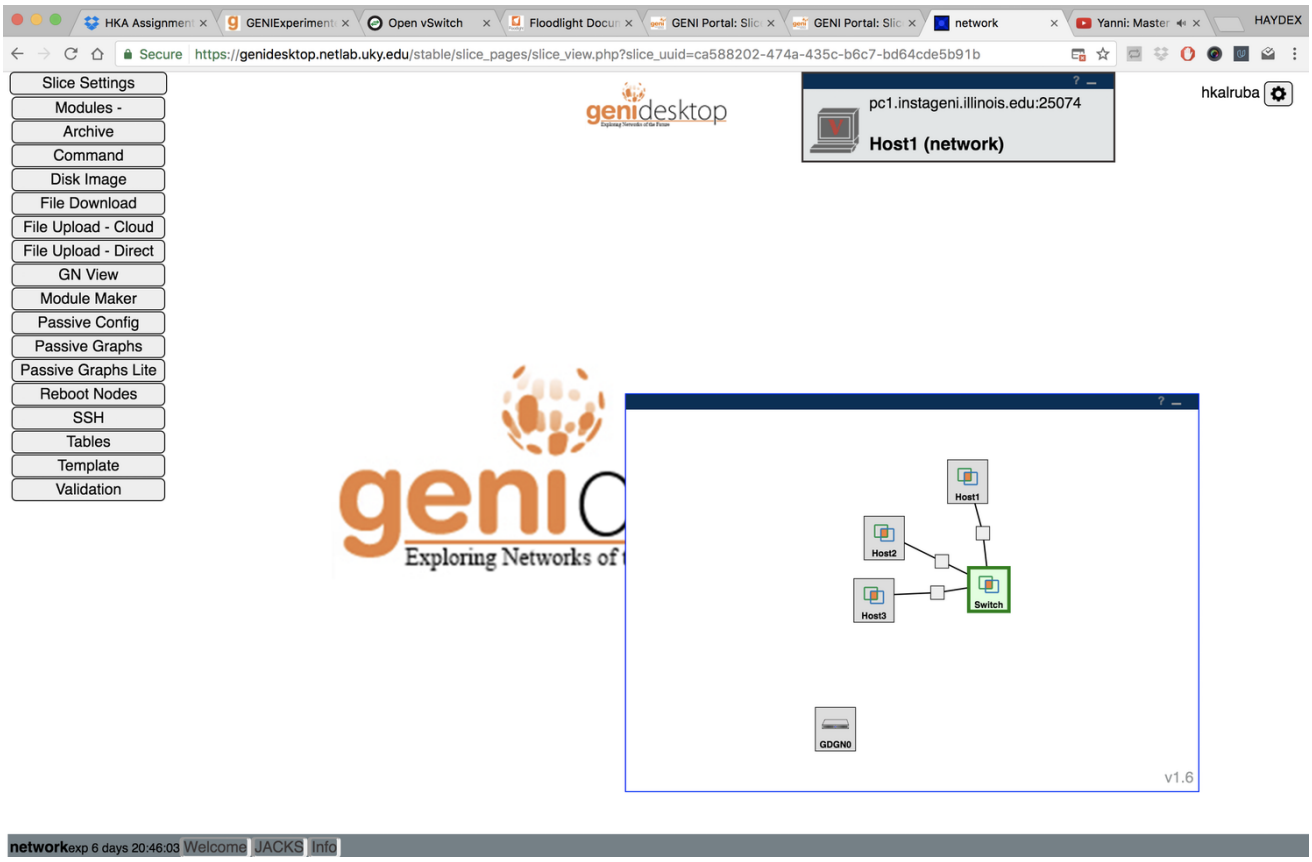

Figure 14

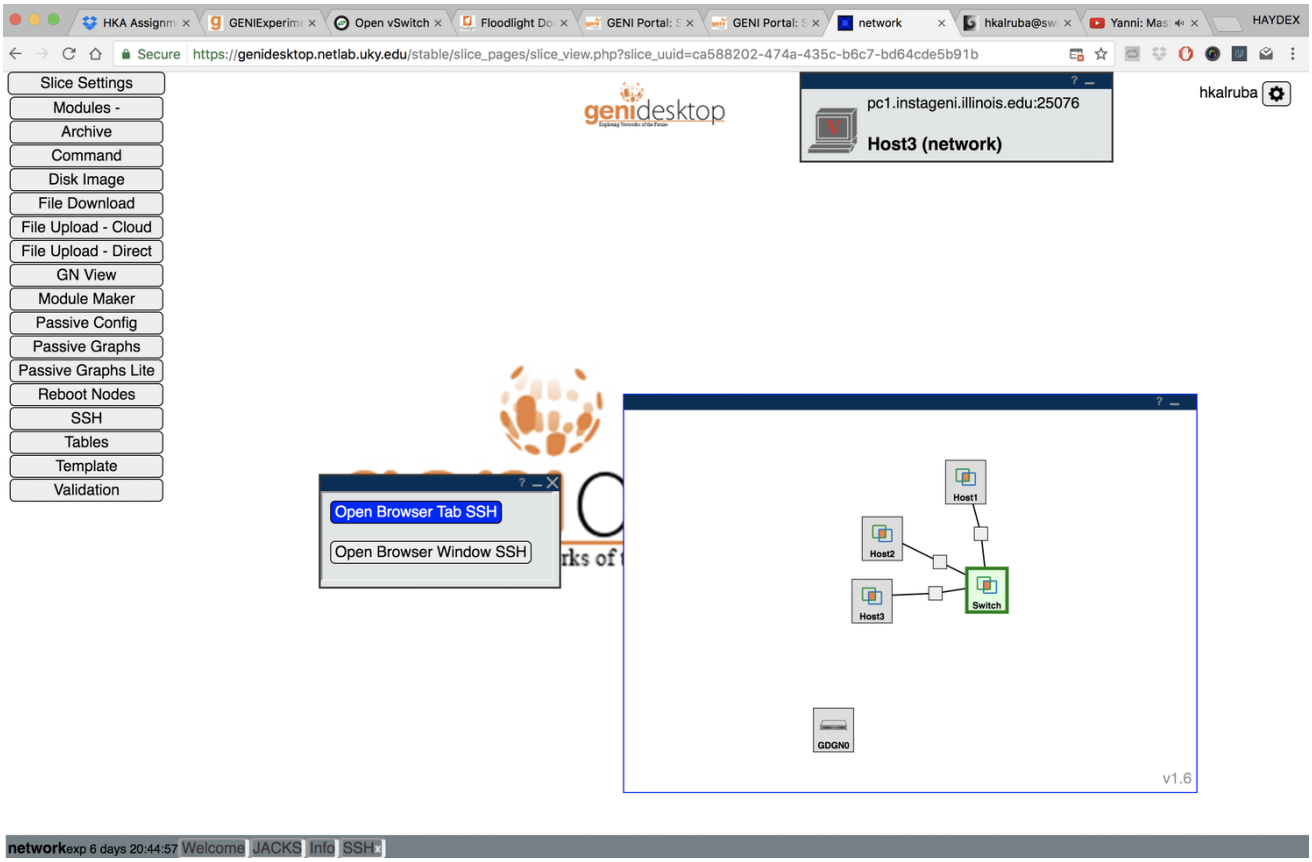

Figure 15

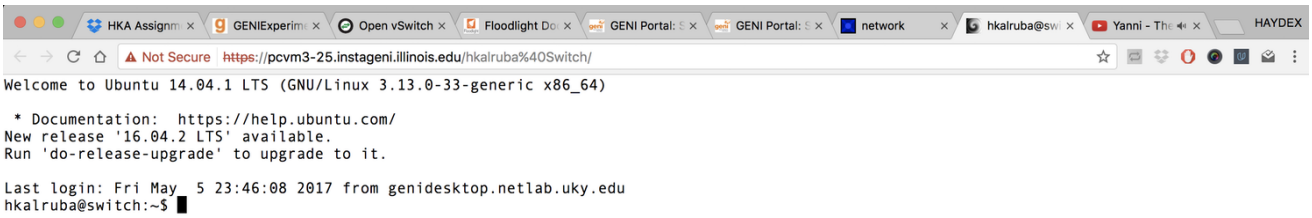

Figure 16 OVS host SSH connection in the browser

ii. We create an Ethernet bridge that will act as our software switch, Figure 17

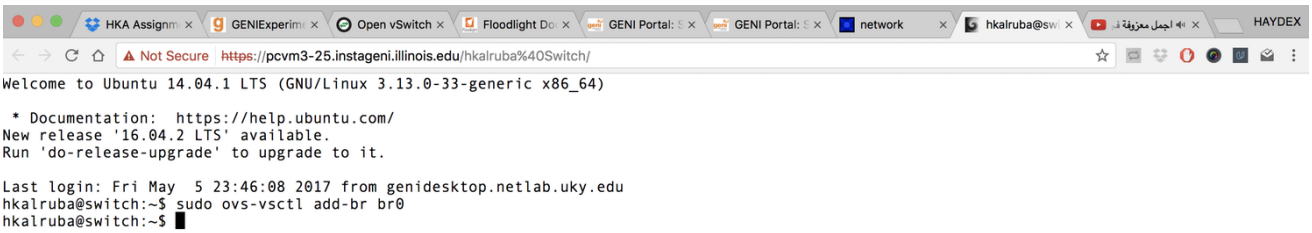

Figure 17

iii. Prepare the interfaces to be added as ports to the OVS switch

The OVS bridge will be a Layer 2 switch and the ports will not need IP addresses. Before we remove them let's keep track of some information

- We Run ifconfig

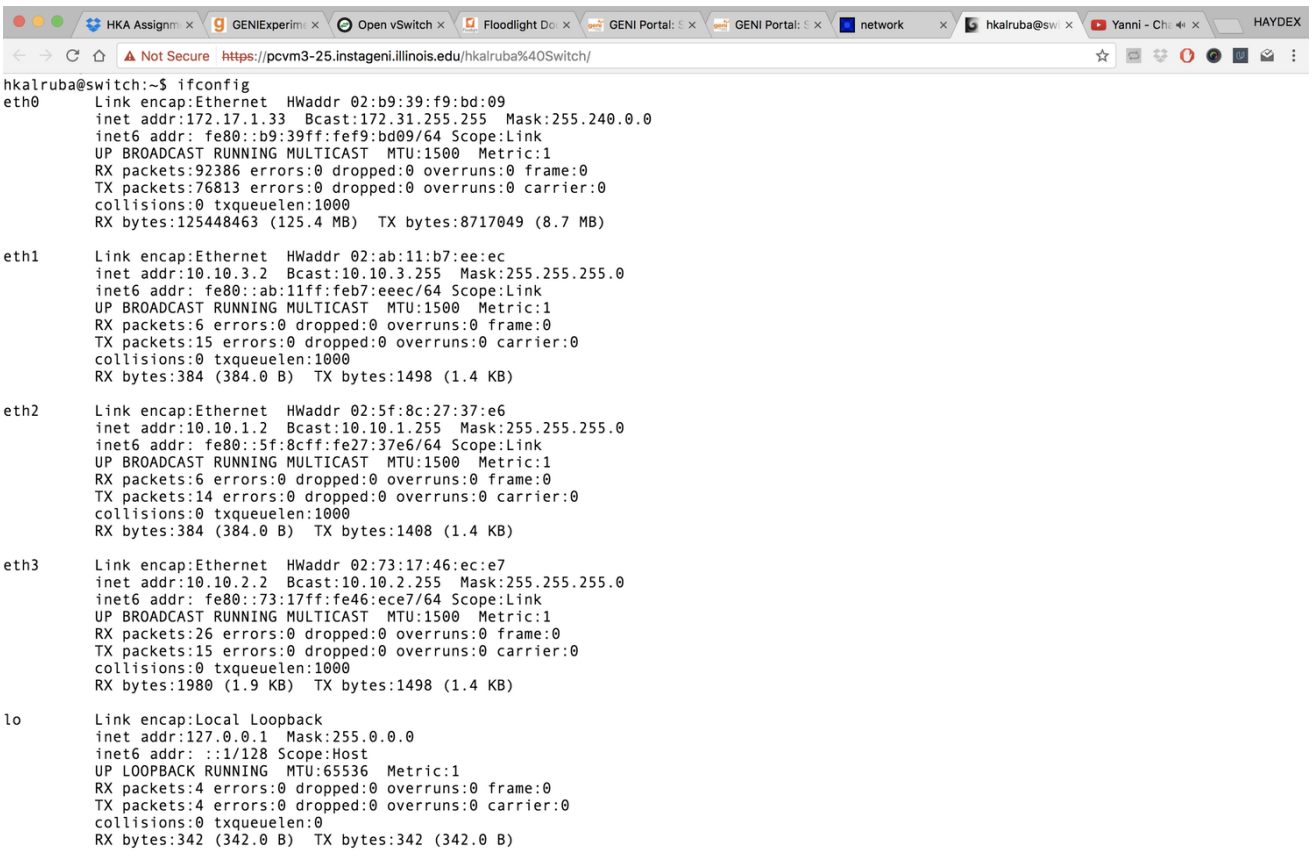

Figure 18

We note down the interfaces that connect to the three hosts we have and remove their IPs. These interfaces are eth1, eth2 and eth3. (Figures 18-19)

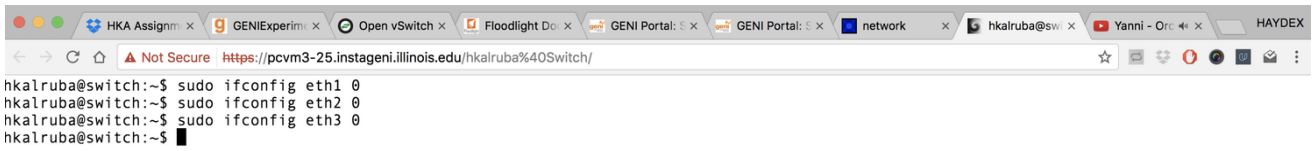

Figure 19

iv. Then we add all the data interfaces to the switch (bridge). (Figure 20)

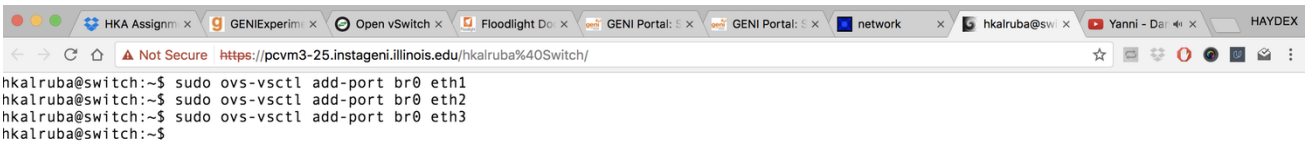

Figure 20

v. Now we verify if the three ports are configured correctly. (Figure 21)

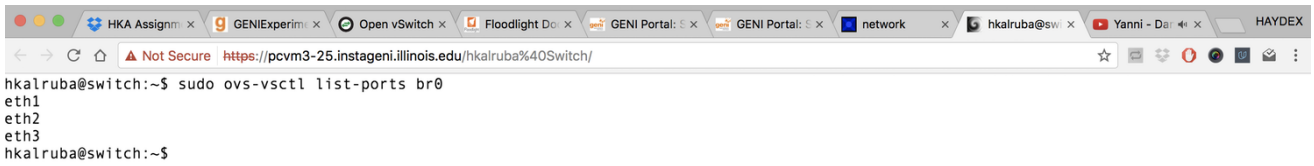

Figure 21

2b. Now we should point the switch to a controller

i. We login to the controller. (Figures 22-25)

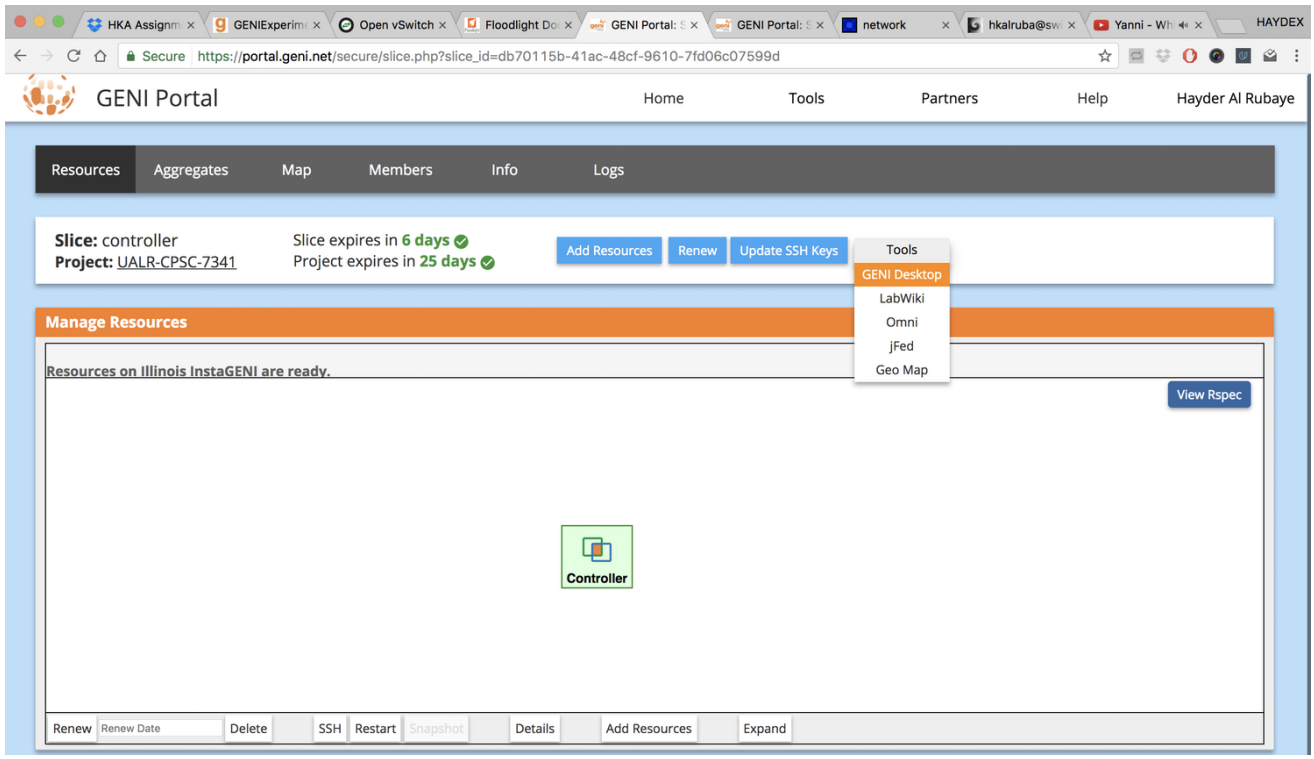

Figure 22

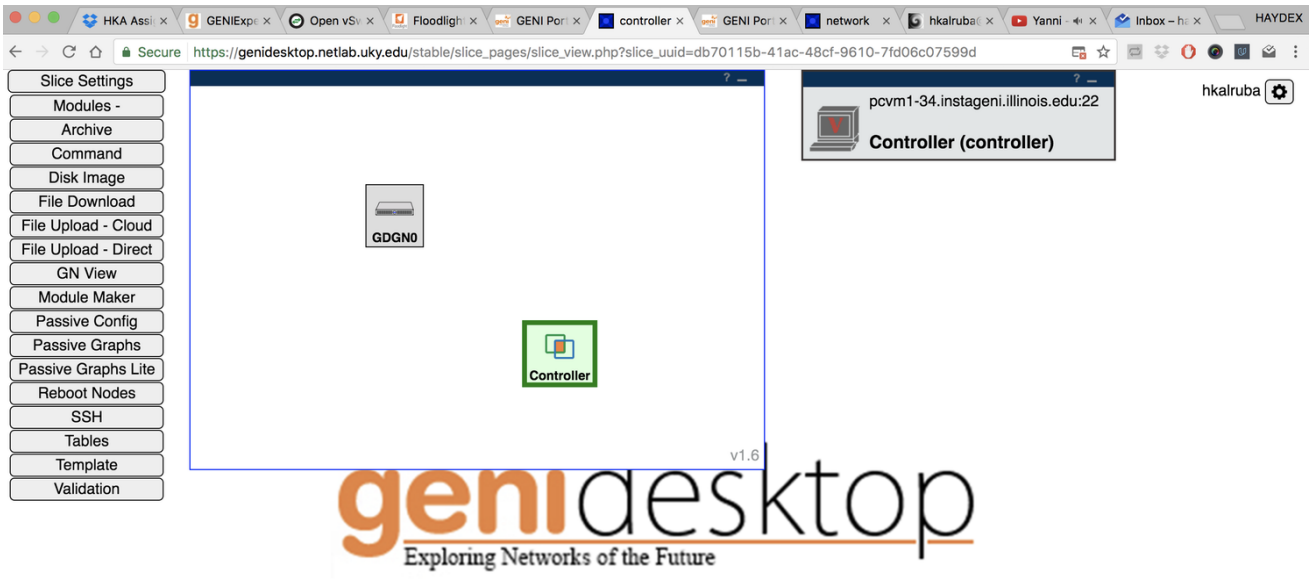

Figure 23

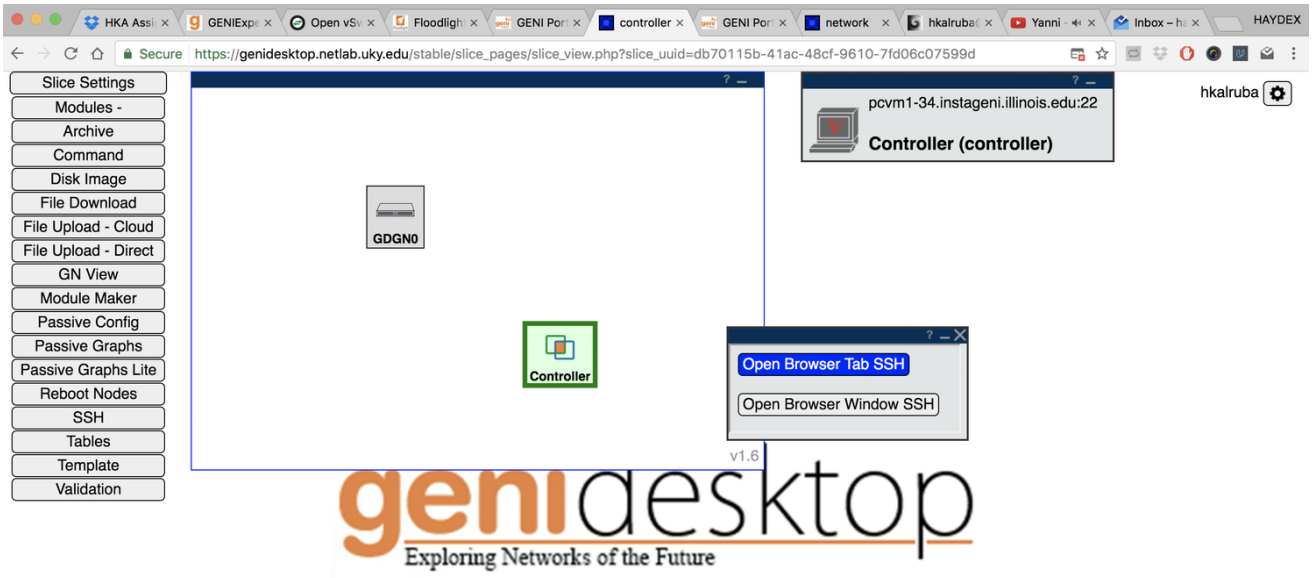

Figure 24

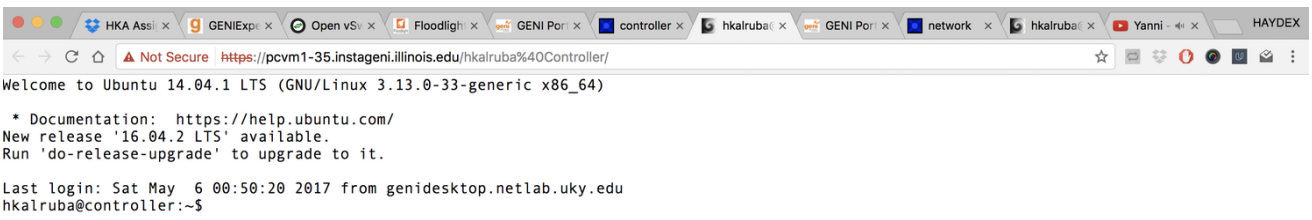

Figure 25

ii. We find the control interface IP of the controller, we use *ifconfig* and note down the IP address of eth0, which is 72.36.65.65 as seen in (Figure 26)

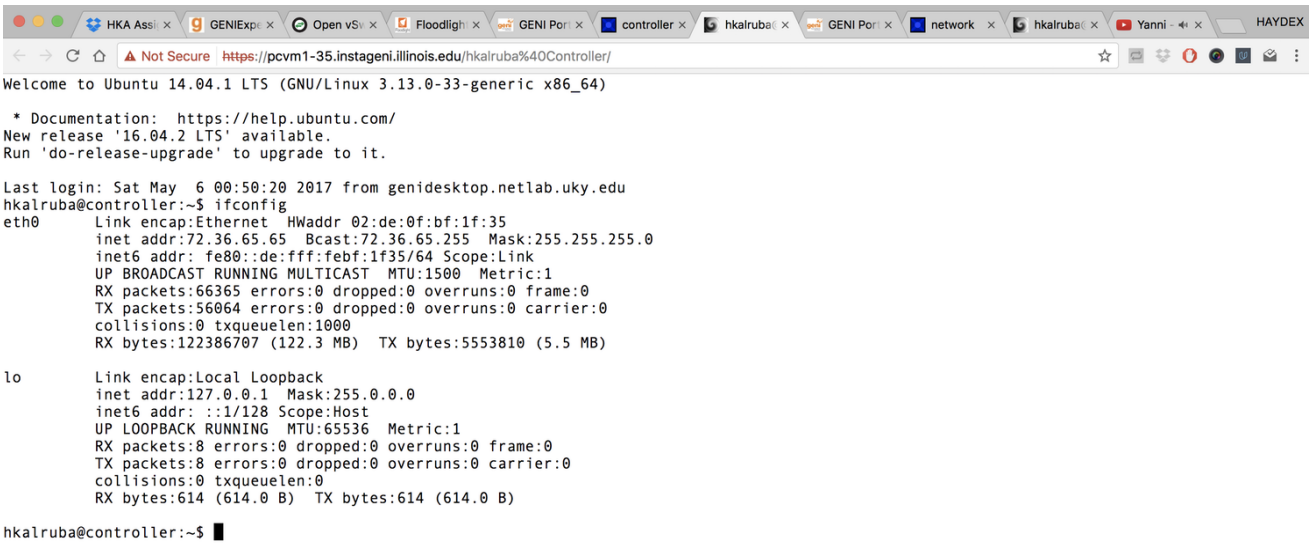

Figure 26

iii. In order to point our software OpenFlow switch to the controller, in the *ovs switch* window, we run the following command (Figure 27)

```
sudo ovs-vsctl set-controller br0 tcp:<controller_ip>:6633
```

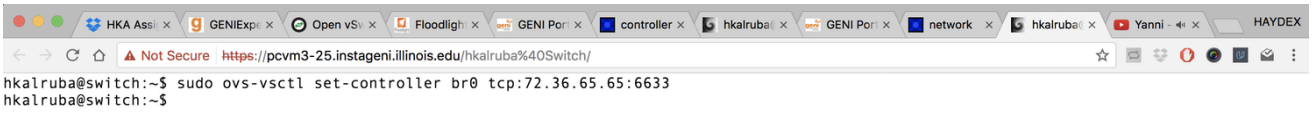

Figure 27

iv. We set the switch to fail-safe-mode. (Figure 28)

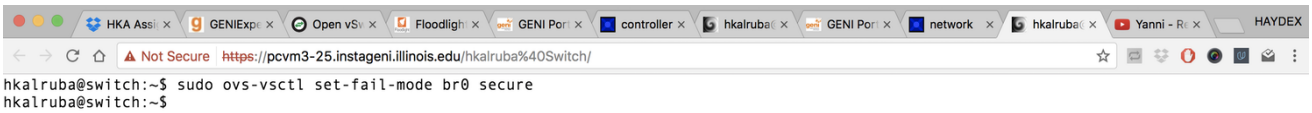

Figure 28

v. We verify the OVS settings. (Figure 29)

Figure 29

Step 4. Experiment Execution

4a. Login to the hosts 10.0.0.1, 10.0.0.2, 10.0.0.3 (Figures 30-33)

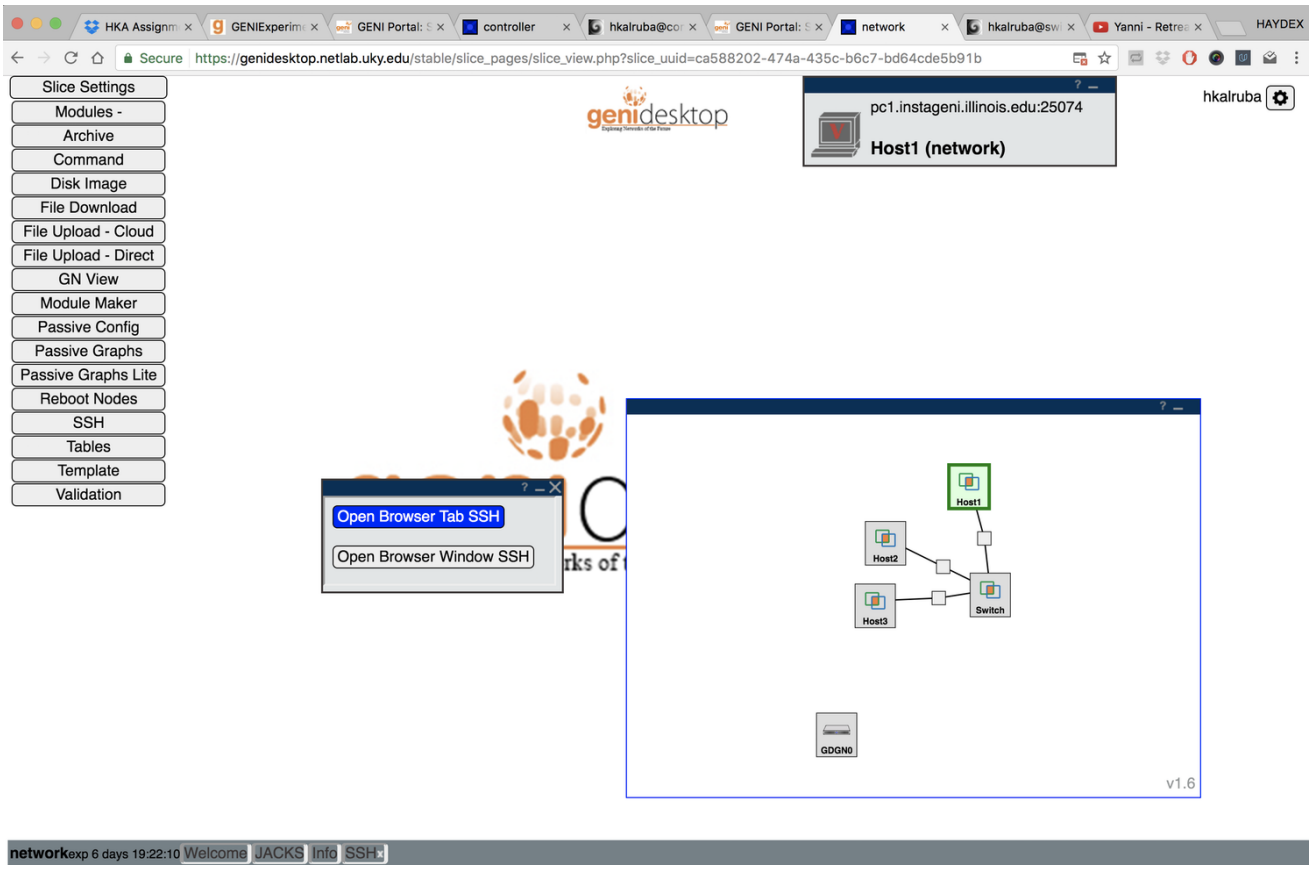

Figure 30

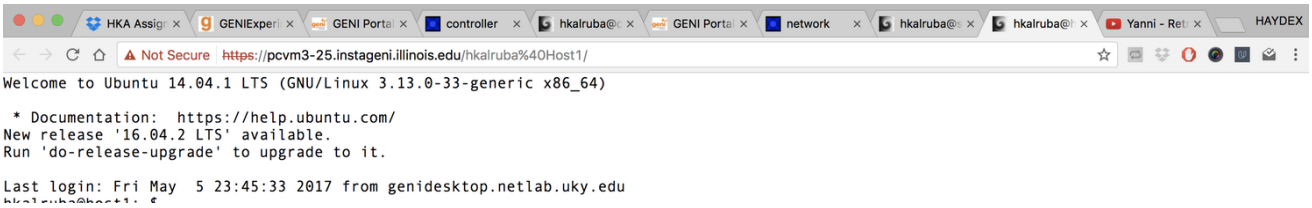

Figure 31 Logged to Host1

Figure 32 Logged to Host2

Figure 33 Logged to Host3

4b. We use a Learning Switch Controller

1. First we start a ping from host1(10.0.0.1) to host2(10.0.0.2) , which should timeout, since there is no controller running. (Figure 34)

Figure 34

2. Then we start the Floodlight Controller by running the following commands: (Figure 35)

```
cd /local/floodlight
java -jar target/floodlight.jar
```

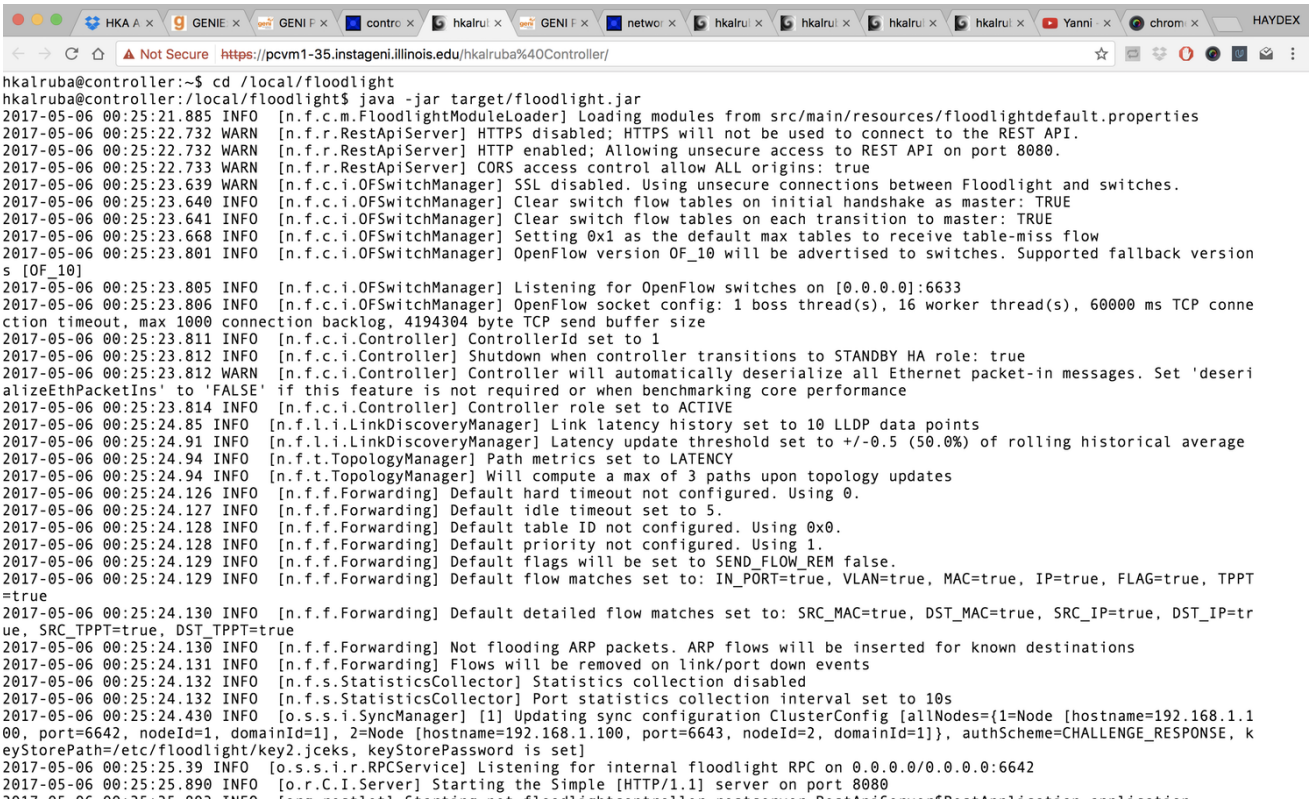

Figure 35

```
2017-05-06 00:25:24.129 INFO  [n.f.f.Forwarding] Default flags will be set to SEND_FLOW_REM false.
2017-05-06 00:25:24.129 INFO  [n.f.f.Forwarding] Default flow matches set to: IN_PORT=true, VLAN=true, MAC=true, IP=true, FLAG=true, TPPT
=true
2017-05-06 00:25:24.130 INFO  [n.f.f.Forwarding] Default detailed flow matches set to: SRC_MAC=true, DST_MAC=true, SRC_IP=true, DST_IP=tr
ue, SRC_TPPT=true, DST_TPPT=true
2017-05-06 00:25:24.130 INFO  [n.f.f.Forwarding] Not flooding ARP packets. ARP flows will be inserted for known destinations
2017-05-06 00:25:24.131 INFO  [n.f.f.Forwarding] Flows will be removed on link/port down events
2017-05-06 00:25:24.132 INFO  [n.f.s.StatisticsCollector] Statistics collection disabled
2017-05-06 00:25:24.132 INFO  [n.f.s.StatisticsCollector] Port statistics collection interval set to 10s
2017-05-06 00:25:24.430 INFO  [o.s.s.i.SyncManager] [1] Updating sync configuration ClusterConfig [allNodes={1=Node [hostname=192.168.1.1
00, port=6642, nodeId=1, domainId=1], 2=Node [hostname=192.168.1.100, port=6643, nodeId=2, domainId=1]}, authScheme=CHALLENGE_RESPONSE, k
eyStorePath=/etc/floodlight/key2.jceks, keyStorePassword is set]
2017-05-06 00:25:25.39 INFO  [o.s.s.i.r.RPCService] Listening for internal floodlight RPC on 0.0.0.0/0.0.0.0:6642
2017-05-06 00:25:25.890 INFO  [o.r.C.I.Server] Starting the Simple [HTTP/1.1] server on port 8080
2017-05-06 00:25:25.892 INFO  [org.restlet] Starting net.floodlightcontroller.restserver.RestApiServer$RestApplication application
2017-05-06 00:25:27.910 INFO  [n.f.c.i.OFChannelHandler] New switch connection from /172.17.1.33:33723
2017-05-06 00:25:28.1 INFO  [n.f.c.i.OFChannelHandler] Negotiated down to controller OpenFlow version of OF_10 for /172.17.1.33:33723 usi
ng lesser hello header algorithm.
2017-05-06 00:25:28.435 INFO  [n.f.c.i.OFSwitchHandshakeHandler] Switch OFSwitch DPID[00:00:06:14:8d:84:32:4b] bound to class class net.f
loodlightcontroller.core.internal.OFSwitch, description SwitchDescription [manufacturerDescription=Nicira, Inc., hardwareDescription=Open
 vSwitch, softwareDescription=2.3.1, serialNumber=None, datapathDescription=None]
2017-05-06 00:25:28.524 INFO  [n.f.c.i.OFSwitchHandshakeHandler] Clearing flow tables of 00:00:06:14:8d:84:32:4b on upcoming transition t
o MASTER.
2017-05-06 00:25:29.142 INFO  [n.f.t.TopologyManager] Recomputing topology due to: link-discovery-updates
2017-05-06 00:25:36.576 INFO  [n.f.j.JythonServer] Starting DebugServer on :6655
2017-05-06 00:25:40.129 INFO  [n.f.l.i.LinkDiscoveryManager] Sending LLDP packets out of all the enabled ports
2017-05-06 00:25:55.141 INFO  [n.f.l.i.LinkDiscoveryManager] Sending LLDP packets out of all the enabled ports
2017-05-06 00:26:10.146 INFO  [n.f.l.i.LinkDiscoveryManager] Sending LLDP packets out of all the enabled ports
2017-05-06 00:26:25.154 INFO  [n.f.l.i.LinkDiscoveryManager] Sending LLDP packets out of all the enabled ports
2017-05-06 00:26:40.167 INFO  [n.f.l.i.LinkDiscoveryManager] Sending LLDP packets out of all the enabled ports
2017-05-06 00:26:55.177 INFO  [n.f.l.i.LinkDiscoveryManager] Sending LLDP packets out of all the enabled ports
2017-05-06 00:27:10.185 INFO  [n.f.l.i.LinkDiscoveryManager] Sending LLDP packets out of all the enabled ports
2017-05-06 00:27:25.197 INFO  [n.f.l.i.LinkDiscoveryManager] Sending LLDP packets out of all the enabled ports
2017-05-06 00:27:40.206 INFO  [n.f.l.i.LinkDiscoveryManager] Sending LLDP packets out of all the enabled ports
2017-05-06 00:27:55.211 INFO  [n.f.l.i.LinkDiscoveryManager] Sending LLDP packets out of all the enabled ports
```

Figure 35 Continued

3. In the terminal window of host1, we re-run the ping to host2 i.e. 10.0.0.2 and host3 10.0.0.3 (Figures 36-37)

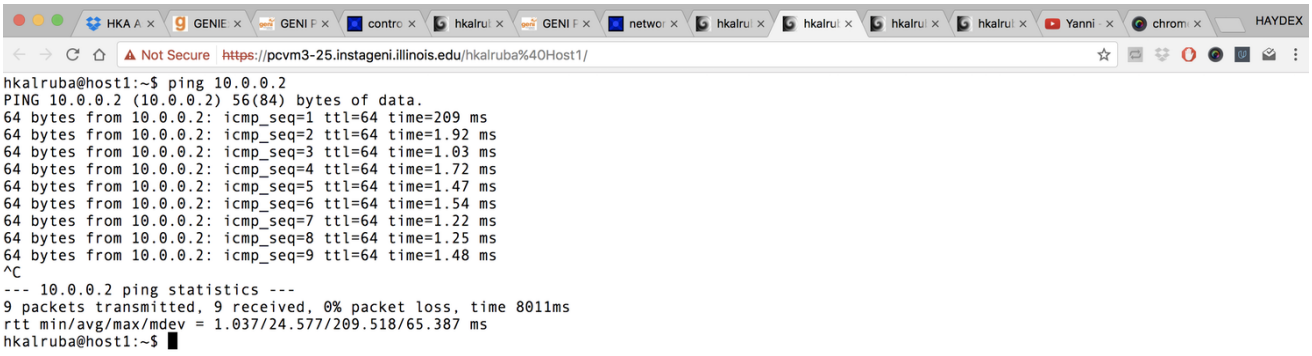

Figure 36

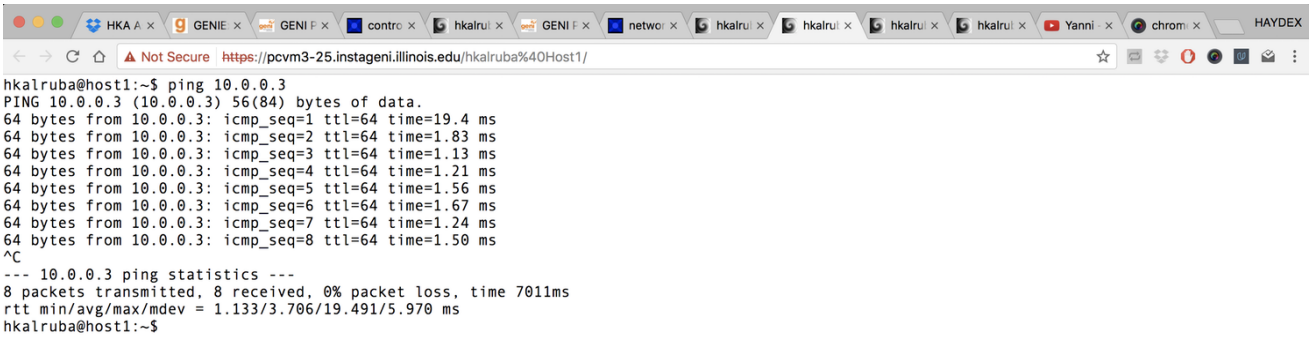

Figure 37

4. We now go to the Open vSwitch host and take a look at the flows. We should see that the controller installed flows based on the mac addresses of the packets. (Figure 38)

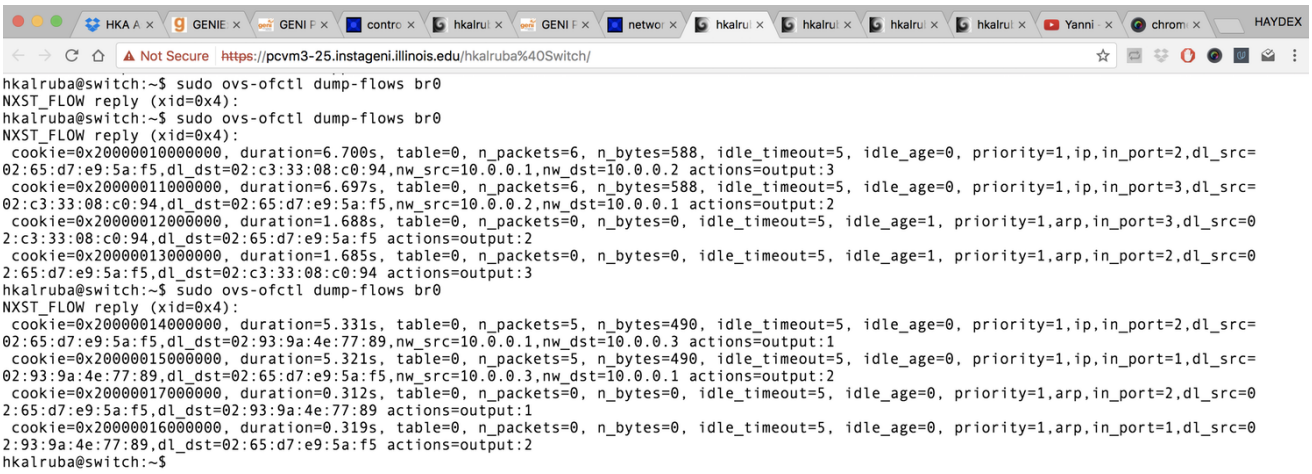

Figure 38

4c. We take a look around the OVS switch

1. To see messages go between the switch and the controller, we open a new ssh window to the controller node and run tcpdump on the eth1 interface and on the tcp port that your controller is listening on usually 6633. (Figure 39)

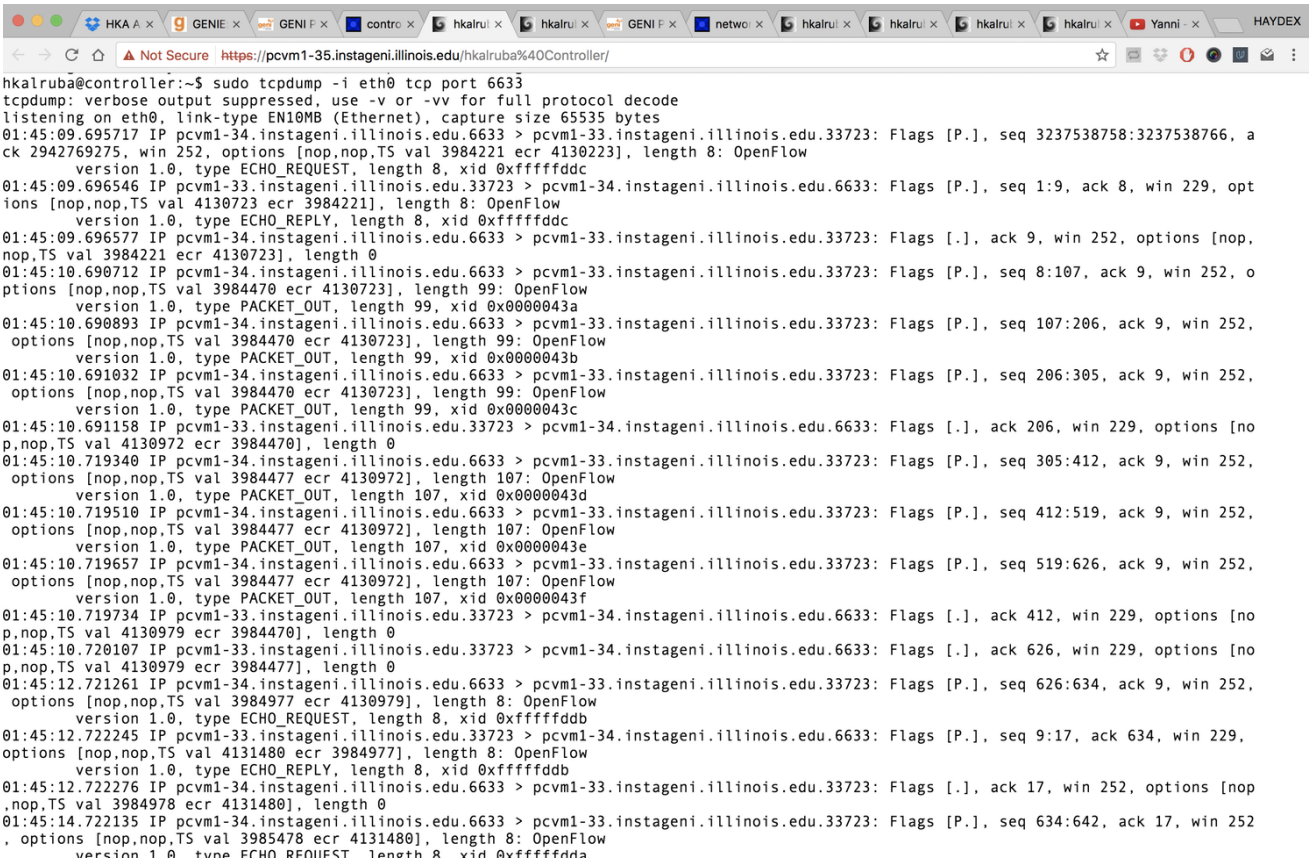

Figure 39

3. we now Kill the Floodlight controller by pressing Ctrl-C. (Figure 40)

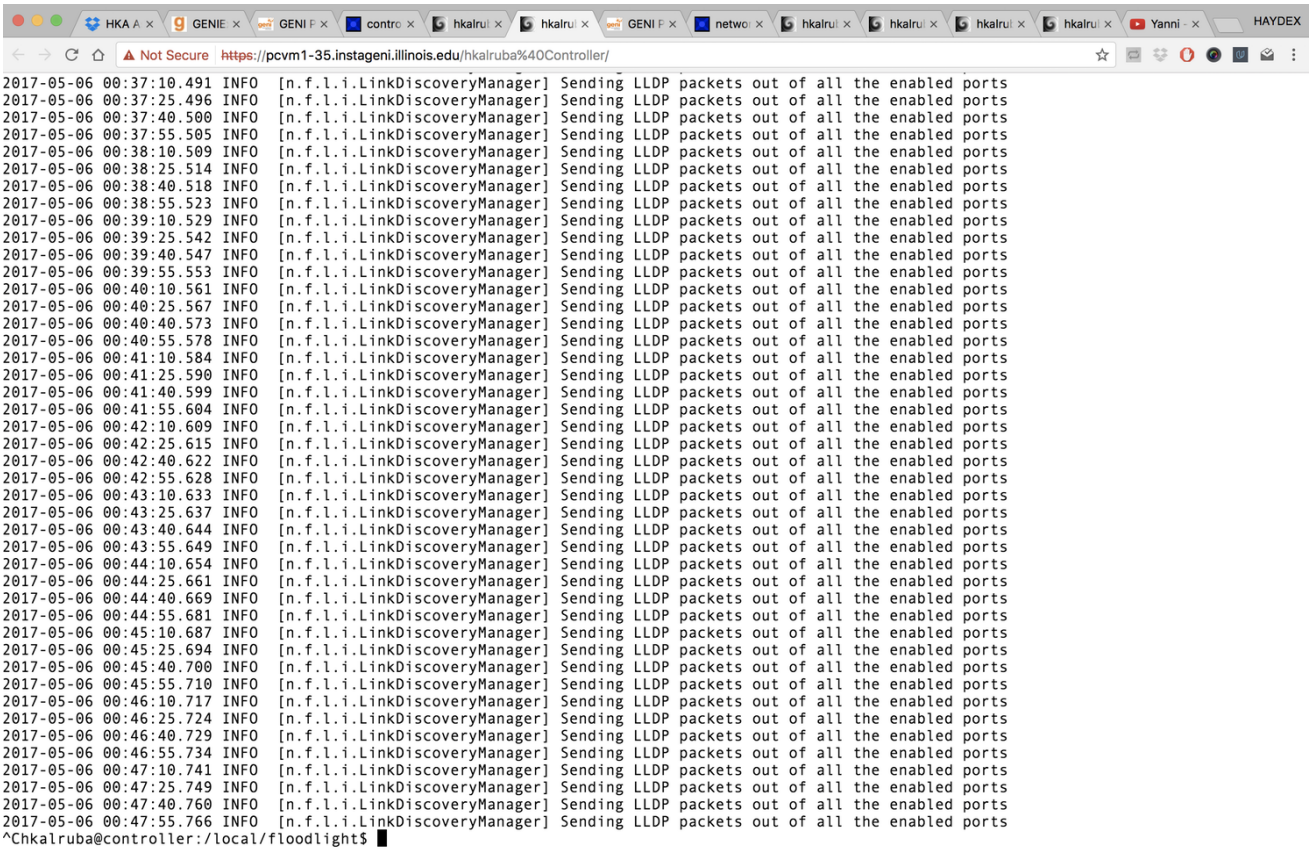

Figure 40

4. The ping on host1 has stopped and Host2 and Host3 became unreachable. (Figure 41)

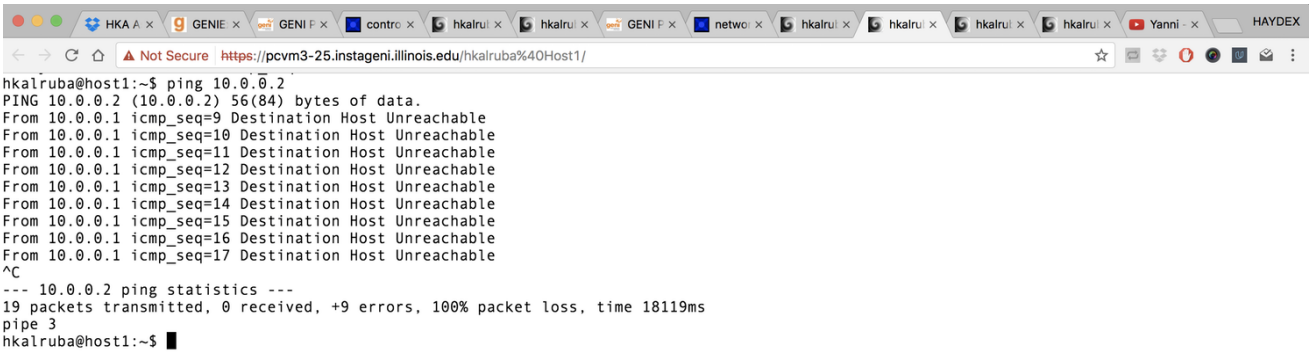

Figure 41

5. We check the flow table entries on the OVS switch, they are empty. (Figure 42)

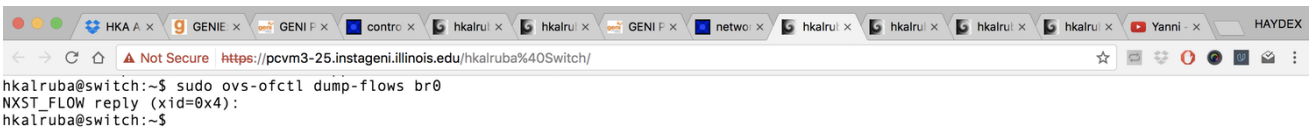

Figure 42

Since we set the switch to "secure" mode, i.e. don't forward packets if the controller fails, we will not see flow table entries. (Figure 42)

4d. Debugging the Controller

i. Print messages

ii. Checking the status in the switch (Figure 43)

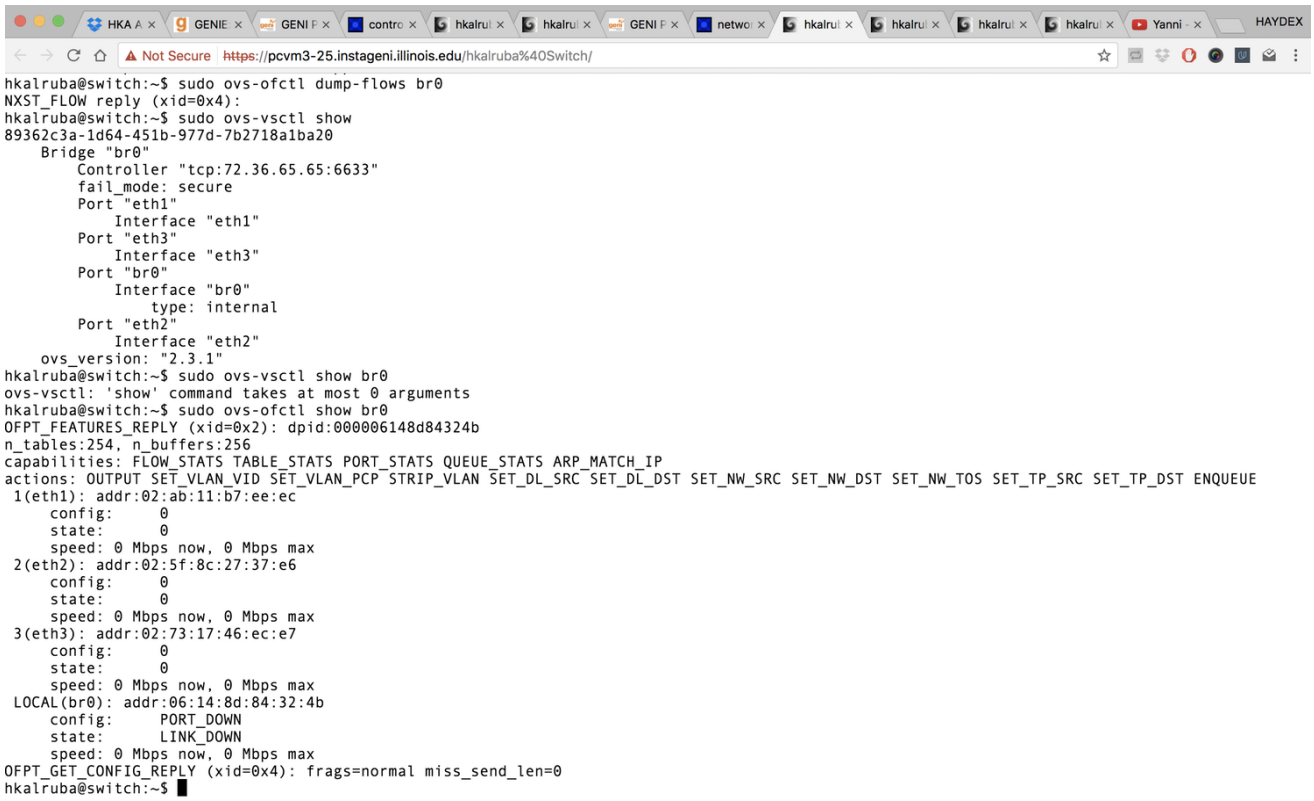

Figure 43

iii. Using Wireshark to see the OpenFlow messages

Using the OpenFlow dissector for Wireshark on the reserved controller host. (Figures 44-46)

We first install the XQUARTZ server (Figure 43.1) to get the wireshark to work on the localhost.

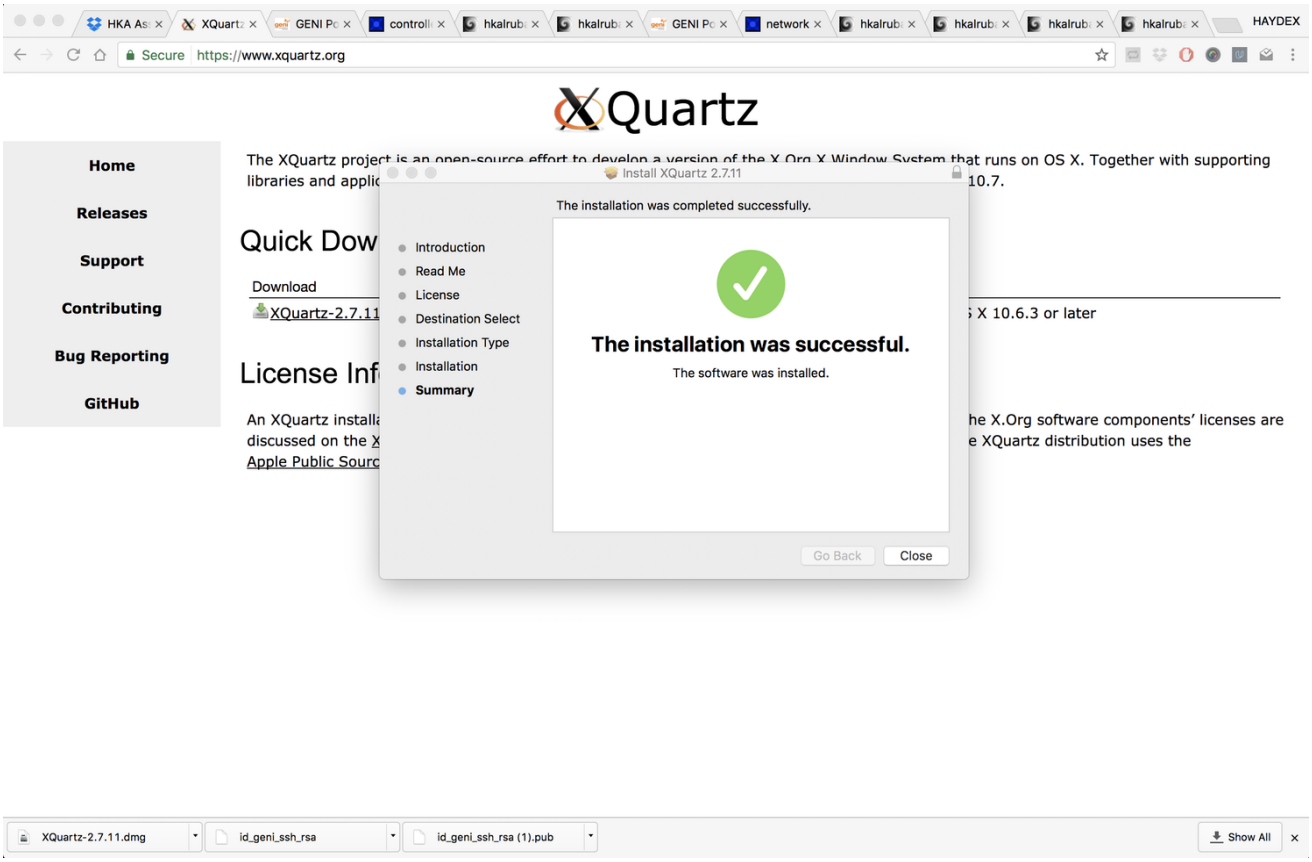

Figure 43.1

To use Wireshark we would use mac's Terminal this time to connect to the controller and the OVS. (Figure 44-46)

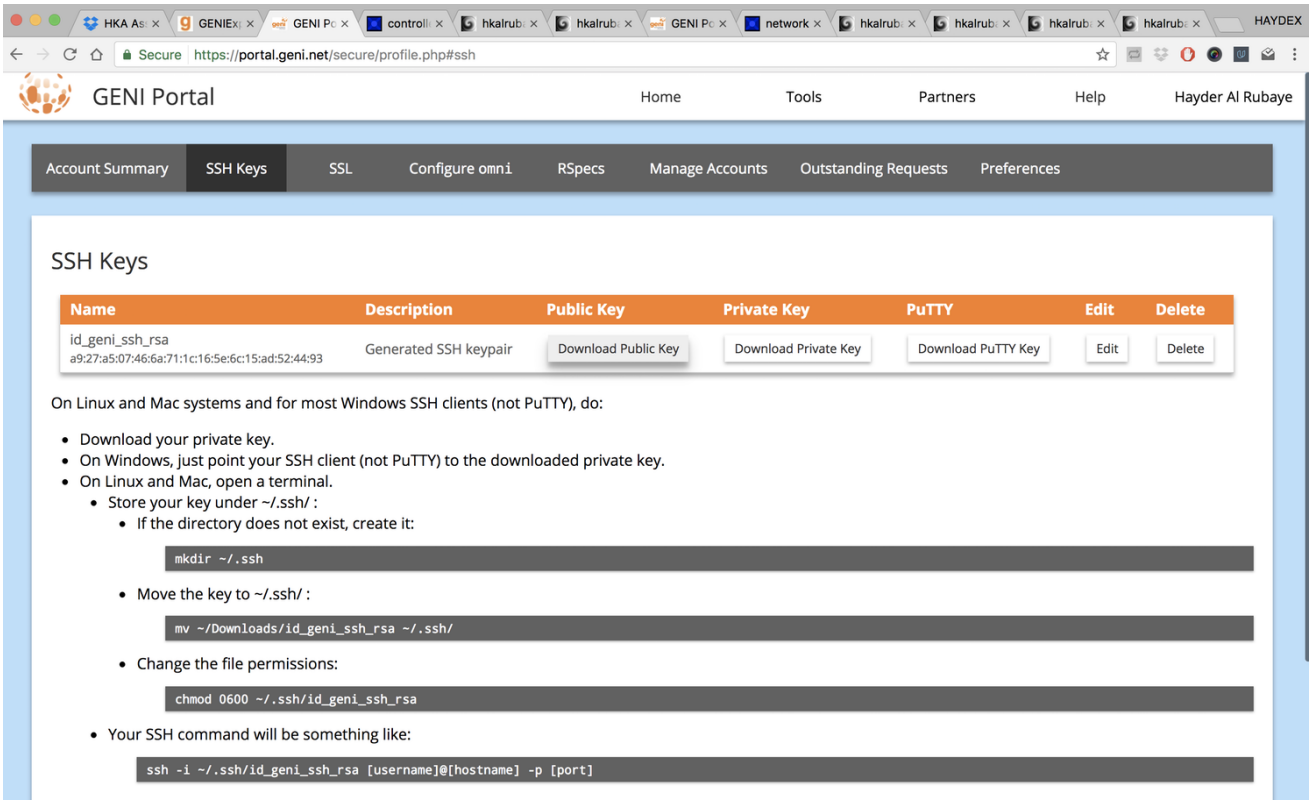

Figure 44 We download the public and private keys

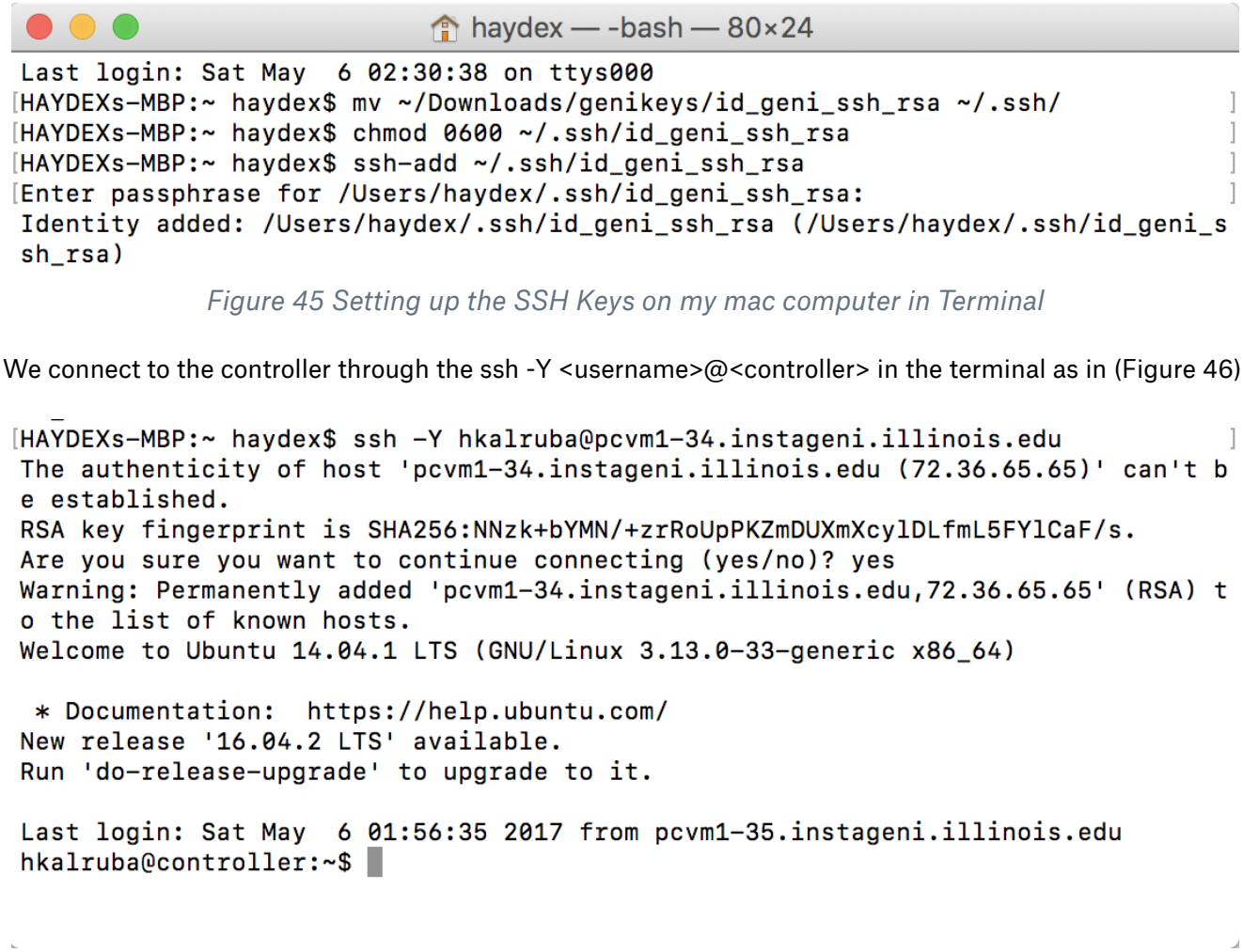

Figure 45 Setting up the SSH Keys on my mac computer in Terminal

We connect to the controller through the ssh -Y <username>@<controller> in the terminal as in (Figure 46)

Figure 46 Now I am connected to the controller through the Terminal

Assuming that the public IP address on the controller is on eth0, we run Wireshark by typing: (Figure 47)

```
sudo wireshark -i eth0&
```

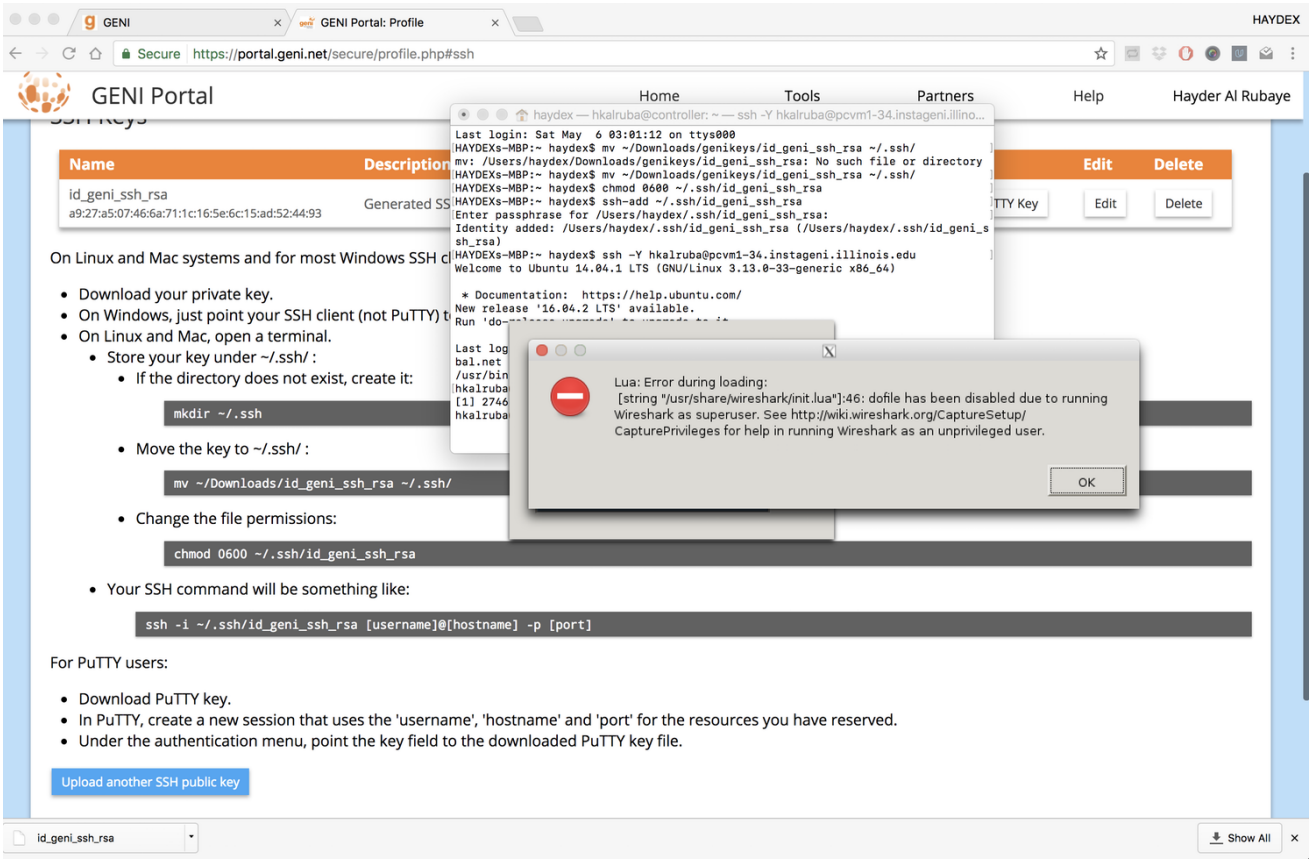

Figure 47 Wireshark in controller

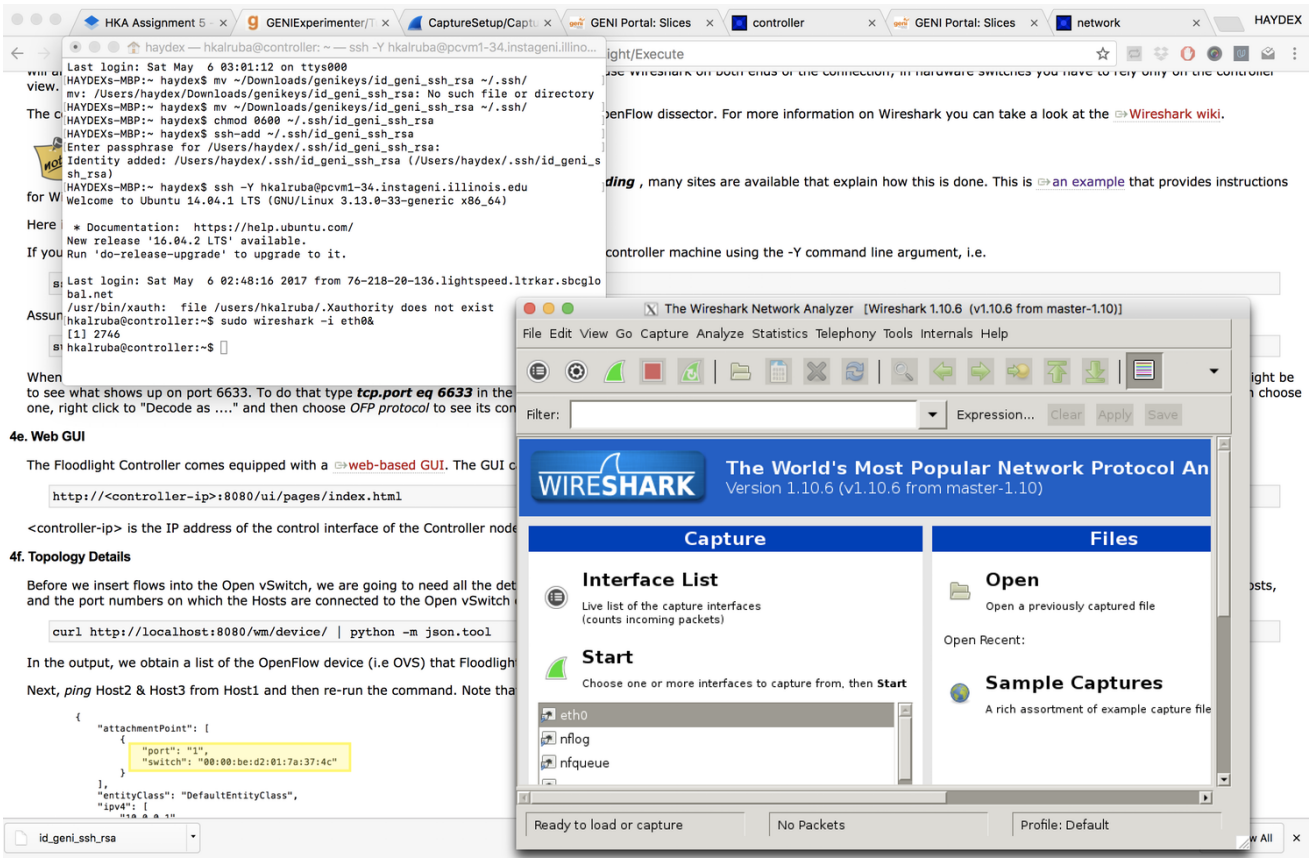

Figure 48

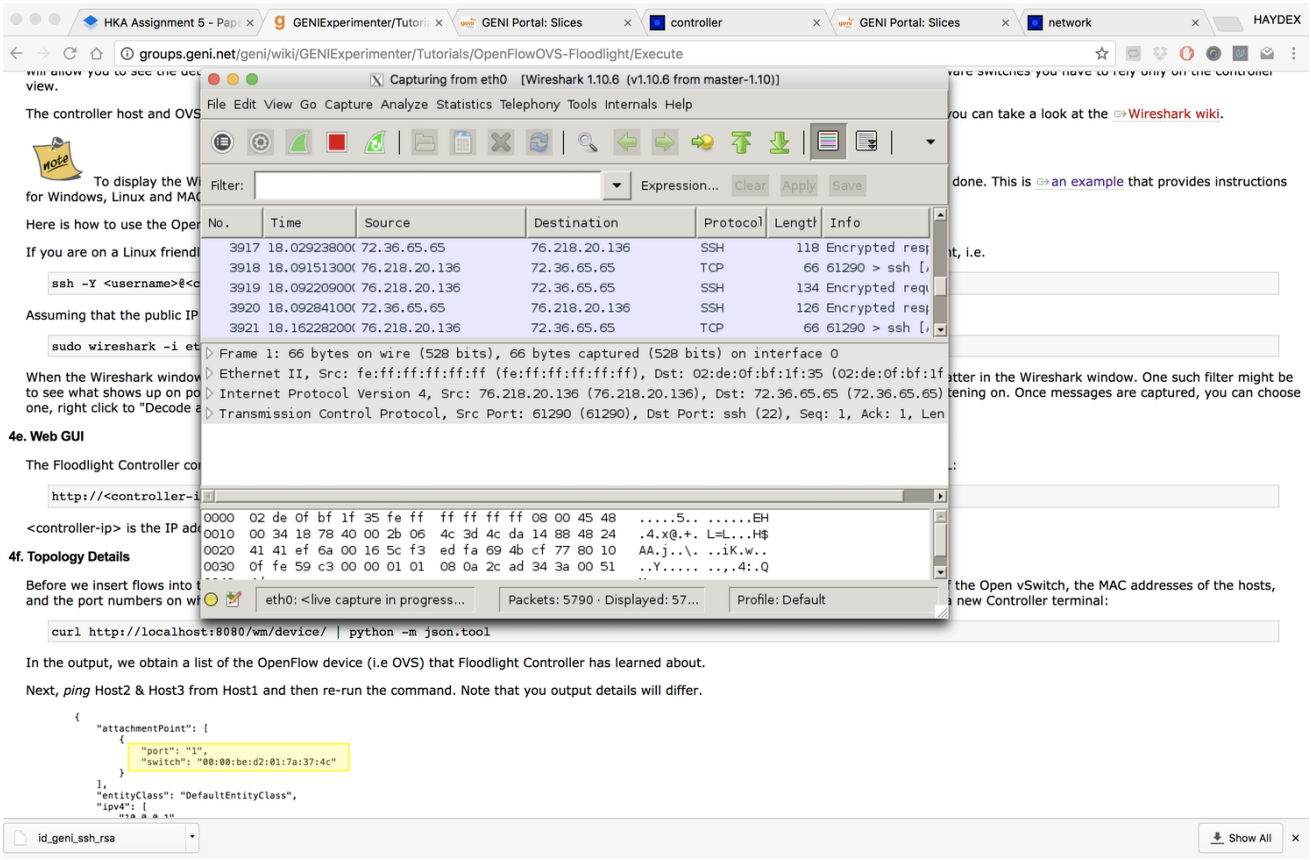

Figure 49 Wireshark started capturing eth0 traffic

When the Wireshark window pops up, we might still have to choose eth0 for a live capture. (Figure 49). And we will want to use a filter to cut down on the chatter in the Wireshark window. One such filter might be to see what shows up on port 6633. To do that we type ***tcp.port eq 6633*** in the filter window, assuming that 6633 is the port that the controller is listening on. Once messages are captured, we can choose one, right click to "Decode as ...." and then we choose *OFP protocol* to see its content.

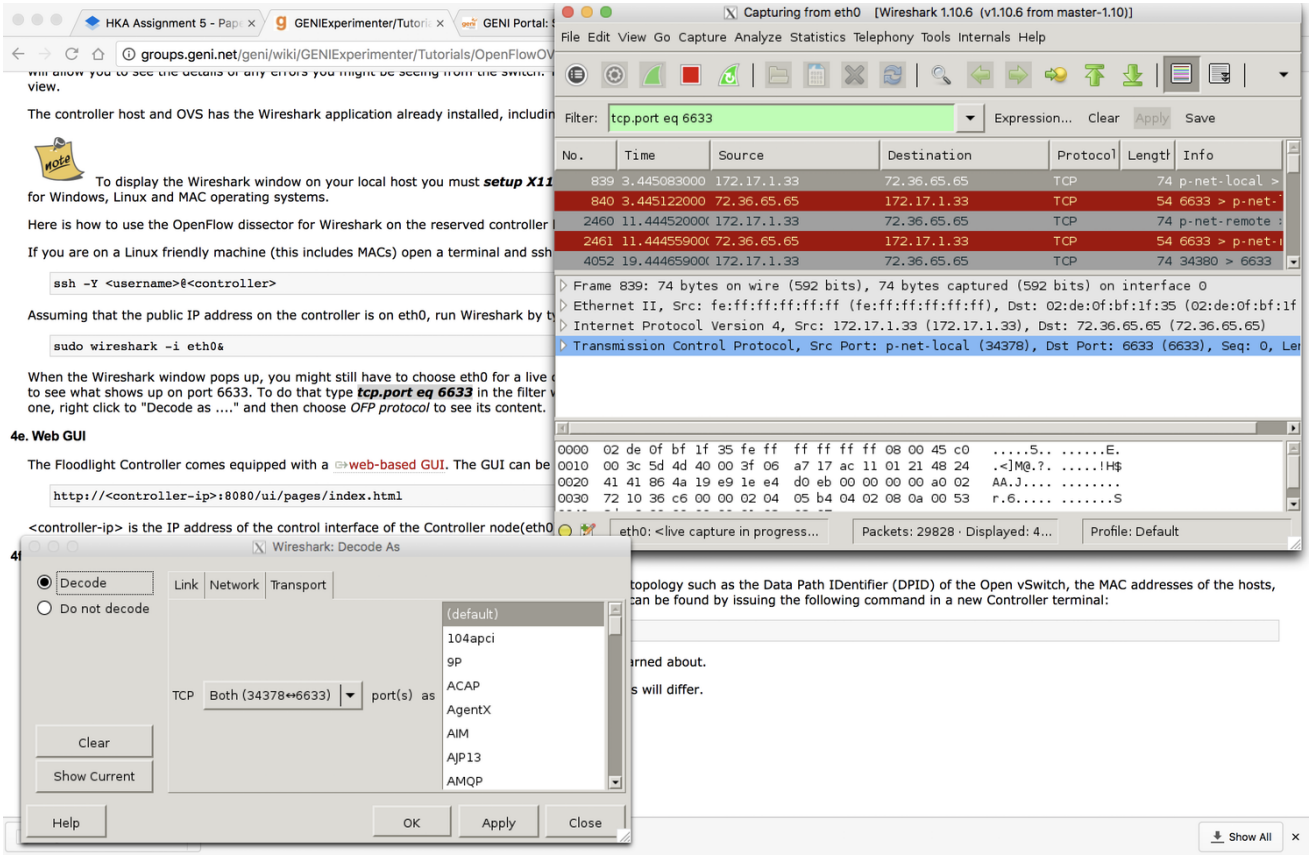

Figure 50 Wireshark applied filter tcp.port eq 6633 and right clicked on one of the packets and chose Decode As

4e. Web GUI

The Floodlight Controller comes equipped with a web-based GUI.

First we start the floodlight controller (Figure 50.1)

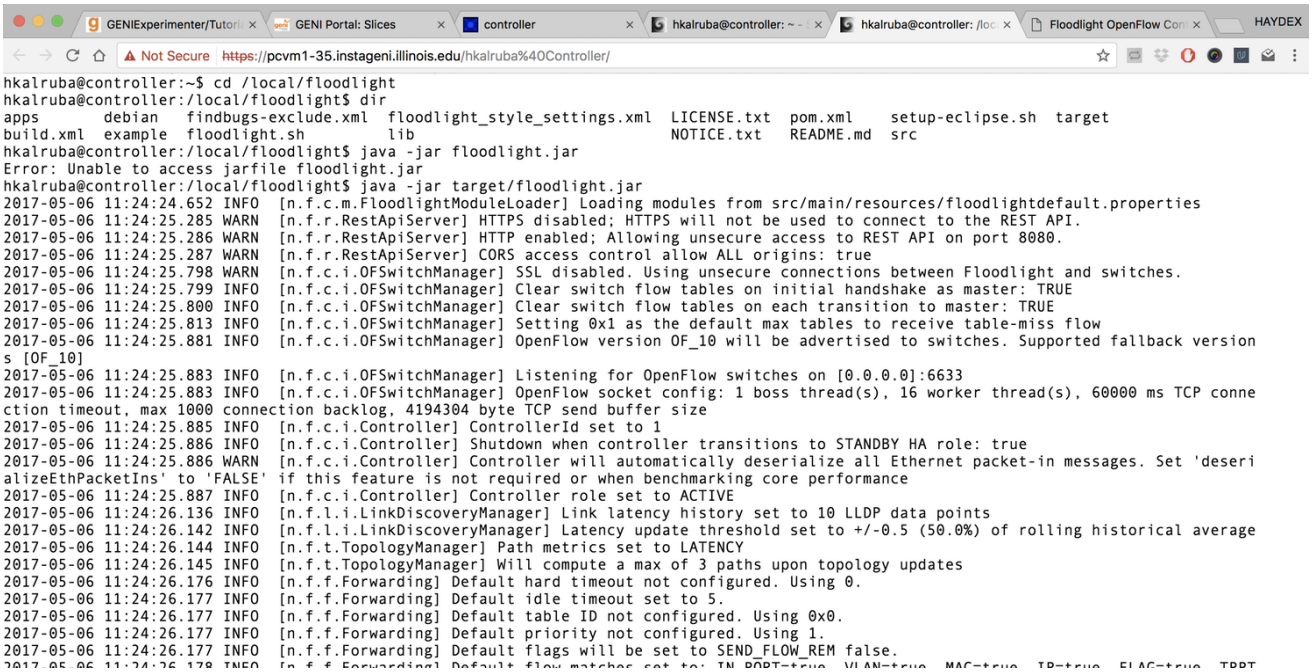

Figure 50.1

Then the GUI can be accessed by pointing the browser to the following URL. (Figure 51)

```
http://72.36.65.65:8080/ui/pages/index.html
```

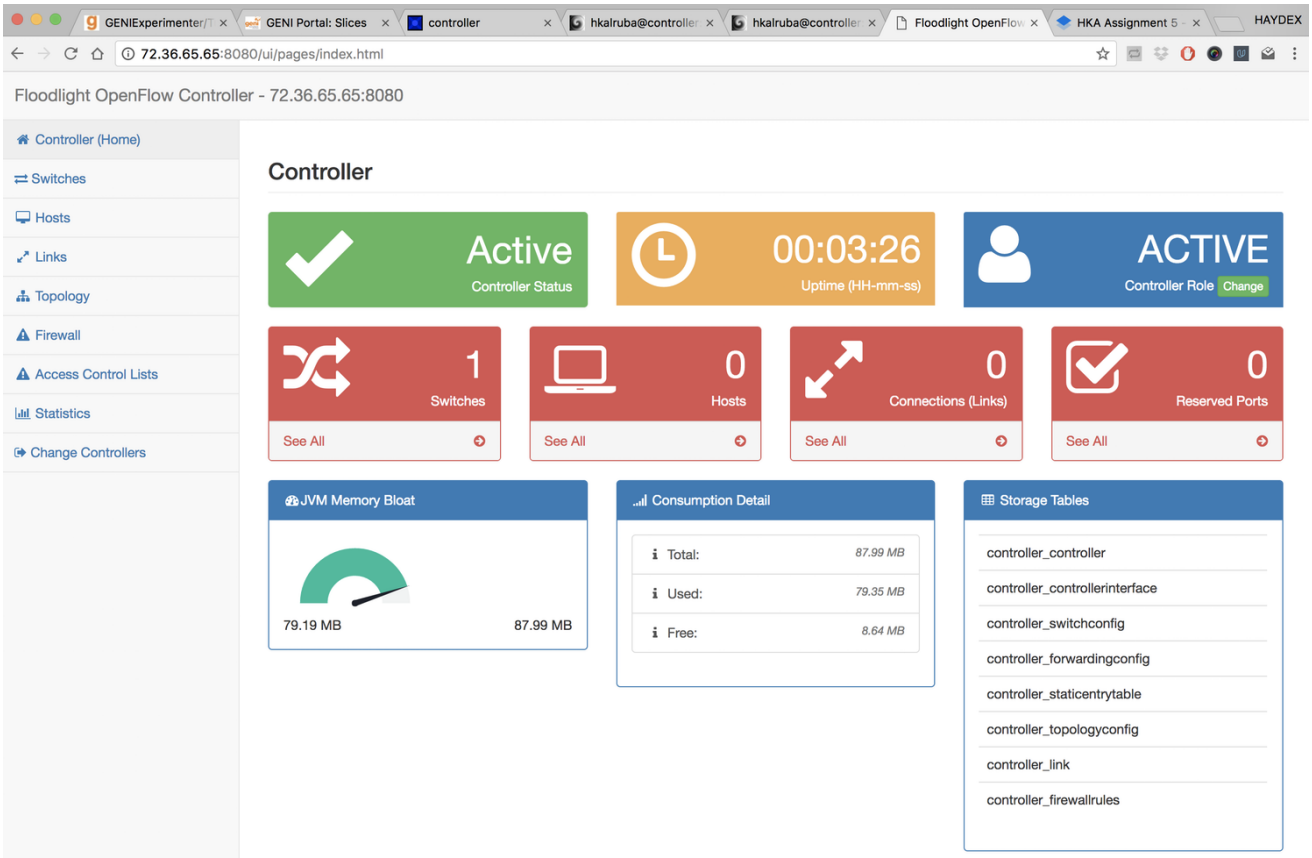

Figure 51

4f. Topology Details

Before we insert flows into the Open vSwitch, we are going to need all the details regarding the topology such as the Data Path IDentifier (DPID) of the Open vSwitch, the MAC addresses of the hosts, and the port numbers on which the Hosts are connected to the Open vSwitch etc. These details can be found by issuing the following command in a new Controller terminal (I have used GENI Desktop in this step): (Figure 52)

curl http://localhost:8080/wm/device/ | python -m json.tool

In the output, we obtain a list of the OpenFlow device (i.e OVS) that Floodlight Controller has learned about.

Next, we *ping* Host2 & Host3 from Host1 and then re-run the command. We will find that the output details will differ.(Figure 52)

We Highlight all of the important details we would need before we start manipulating flows. And we get the DPID by running the following command (Figure 55):

sudo ovs-ofctl show br0

Open vSwitch

DPID = 00:00:06:14:8d:84:32:4b

IP = 10.0.0.1

port = 2

switch = 00:00:06:14:8d:84:32:4b

mac = 02:65:d7:e9:5a:f5

IP = 10.0.0.2
port = 3
switch = 00:00:06:14:8d:84:32:4b
mac = 02:c3:33:08:c0:94

IP = 10.0.0.3
port = 1
switch = 00:00:06:14:8d:84:32:4b
mac = 02:93:9a:4e:77:89

As seen in (Figure 52)

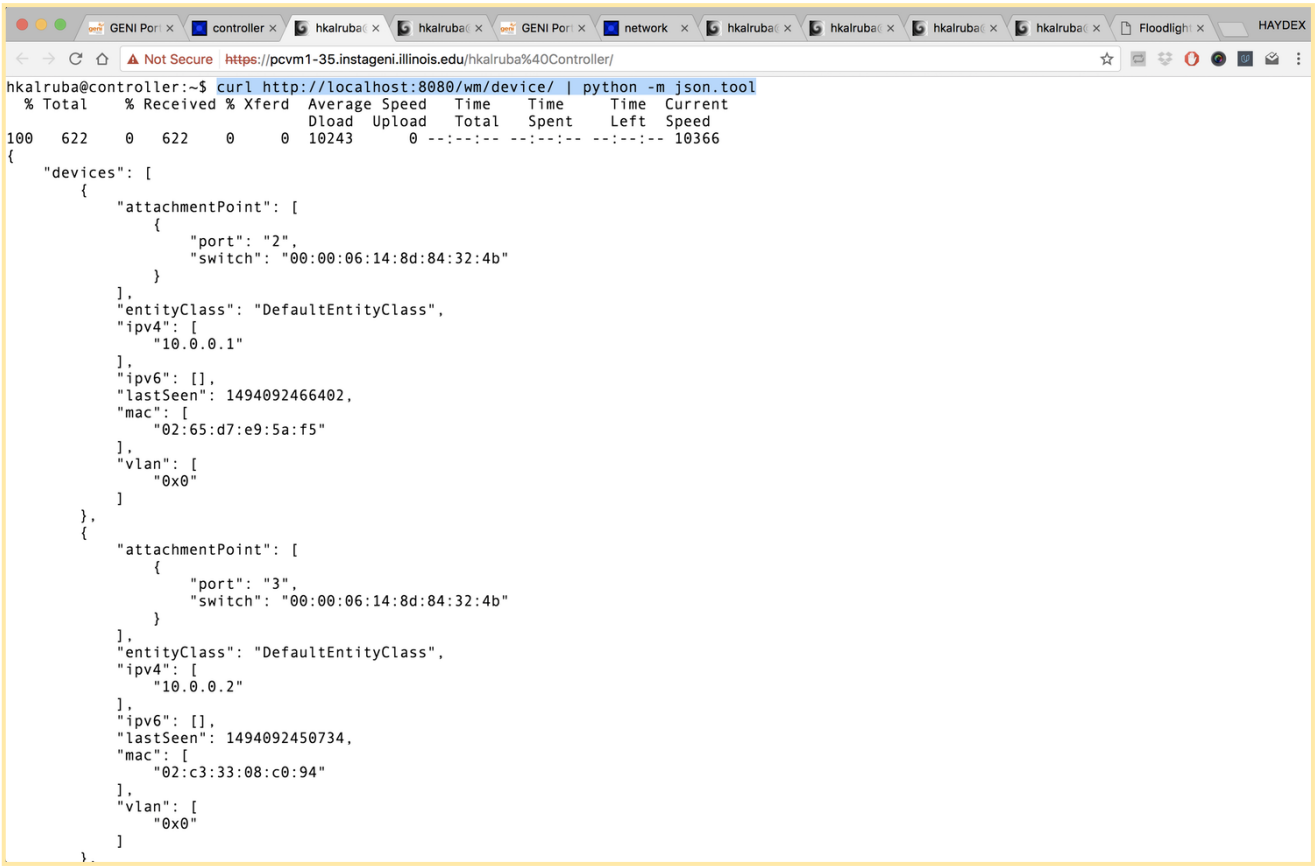

Figure 52

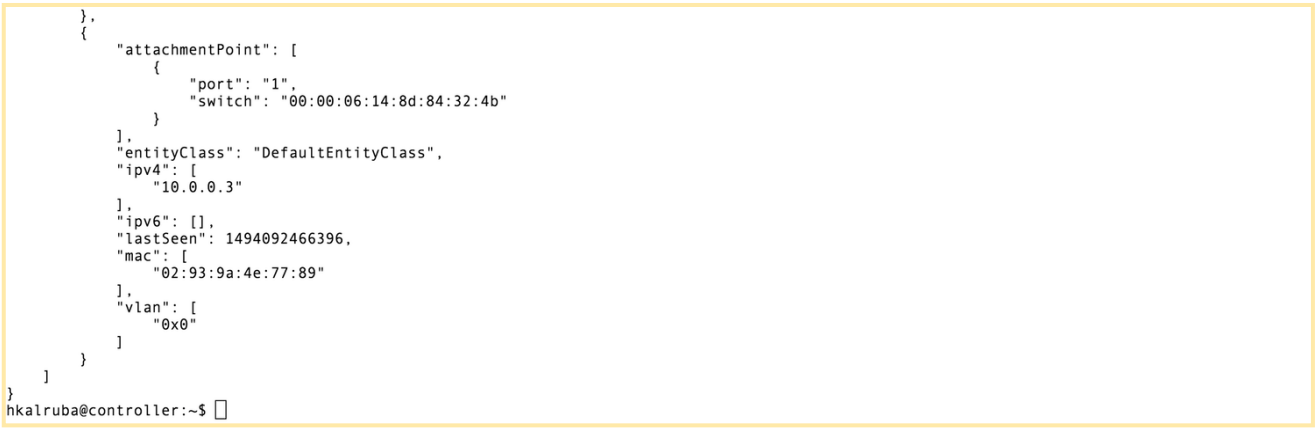

Figure 52 Continued

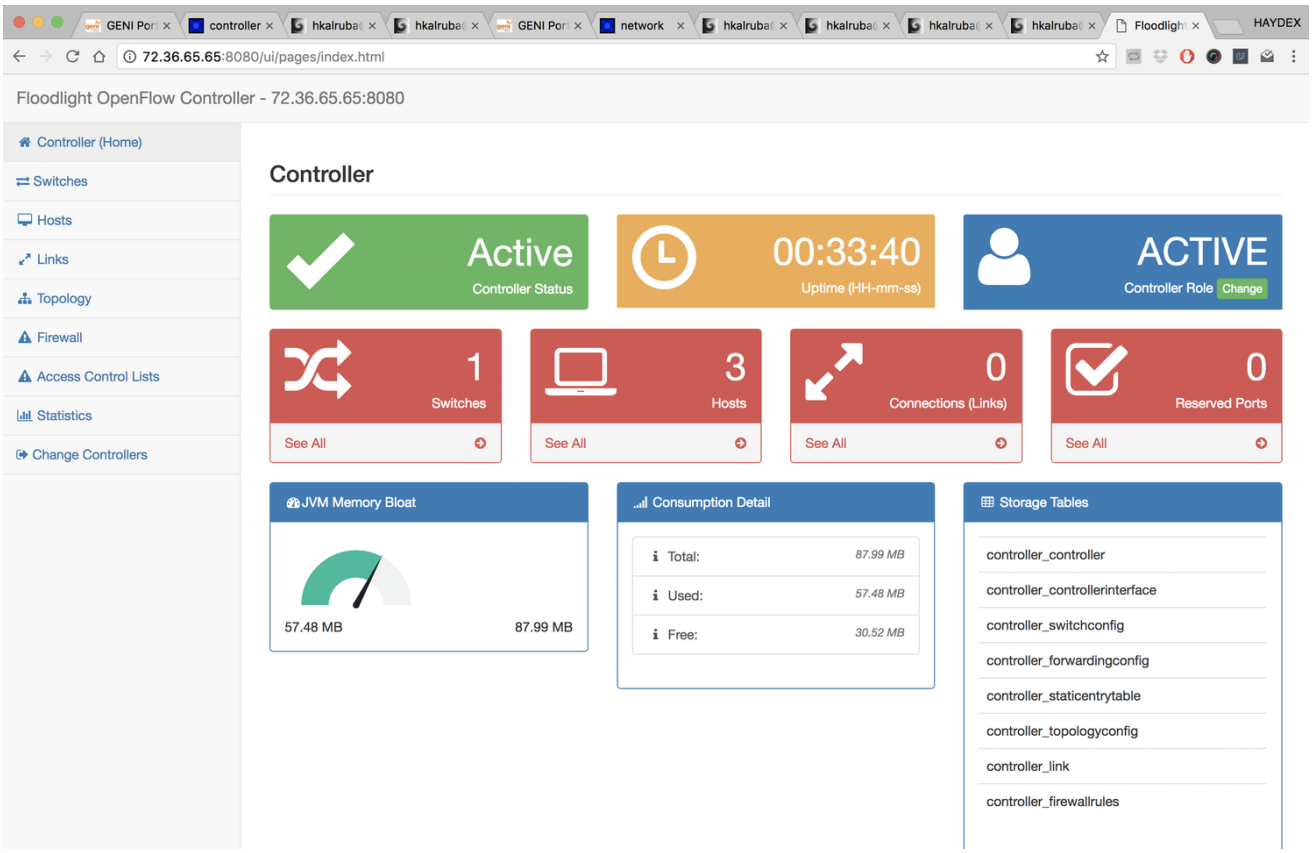

Figure 53 Floodlight detected 3 hosts after pinging them

4f. Now we run the traffic duplication controller

In the above example we ran a very simple learning switch controller. The power of OpenFlow comes from the fact that we can decide to forward the packet any way we want based on the supported OpenFlow actions. A very simple but powerful modification we can do, is to duplicate all the traffic of the switch out a specific port. This is very useful for application and network analysis. At the port where we duplicate traffic we can connect a device that does analysis. For this tutorial we are going to verify the duplication by doing tcpdump on two ports on the OVS switch.

1. We insert the flow to Duplicate Traffic

We are going to duplicate IPv4 traffic from Host1 destined to Host2 on Host3. We open a new *Controller* terminal and type the following flow:

```
curl -X POST -d '{"switch":"<DPID OF OPEN vSWITCH>","name":"flow-1","priorit
y":"32768","in_port":"<PORT OF 10.0.0.1>","active":"true", "eth_type":"0x080
0", "eth_src":"<MAC OF 10.0.0.1>", "eth_dst":"<MAC OF 10.0.0.2>",  "ipv4_sr
c":"10.0.0.1", "ipv4_dst":"10.0.0.2", "actions":"set_eth_dst=<MAC OF 10.0.0.
2>,set_ipv4_dst=10.0.0.2,output=<PORT OF 10.0.0.2>,set_eth_dst=<MAC OF 10.0.
0.3>,output=<PORT OF 10.0.0.3>"}' http://localhost:8080/wm/staticflowpusher/
json
```

```
curl -X POST -d '{"switch":"00:00:06:14:8d:84:32:4b","name":"flow-1","priori
ty":"32768","in_port":"2","active":"true", "eth_type":"0x0800", "eth_src":"0
2:65:d7:e9:5a:f5", "eth_dst":"02:c3:33:08:c0:94",  "ipv4_src":"10.0.0.1", "i
pv4_dst":"10.0.0.2", "actions":"set_eth_dst=02:c3:33:08:c0:94,set_ipv4_dst=1
```

```
0.0.0.2,output=3,set_eth_dst=02:93:9a:4e:77:89,output=1"}' http://localhost:
8080/wm/staticflowpusher/json
```

This time we login to the Controller from the mac Terminal because it is hard for me to write all of the code details in a GENI Desktop Terminal as it does not accept copying and pasting. (Figure 54)

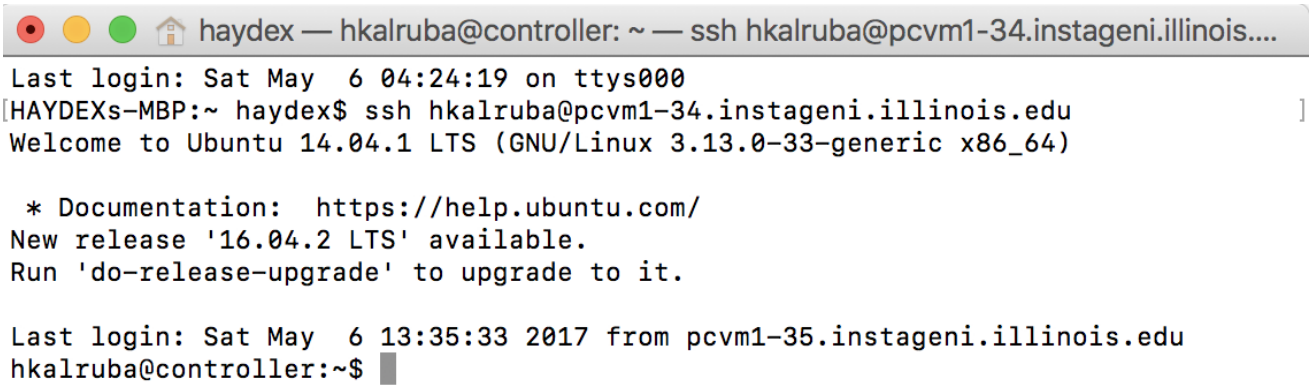

Figure 54

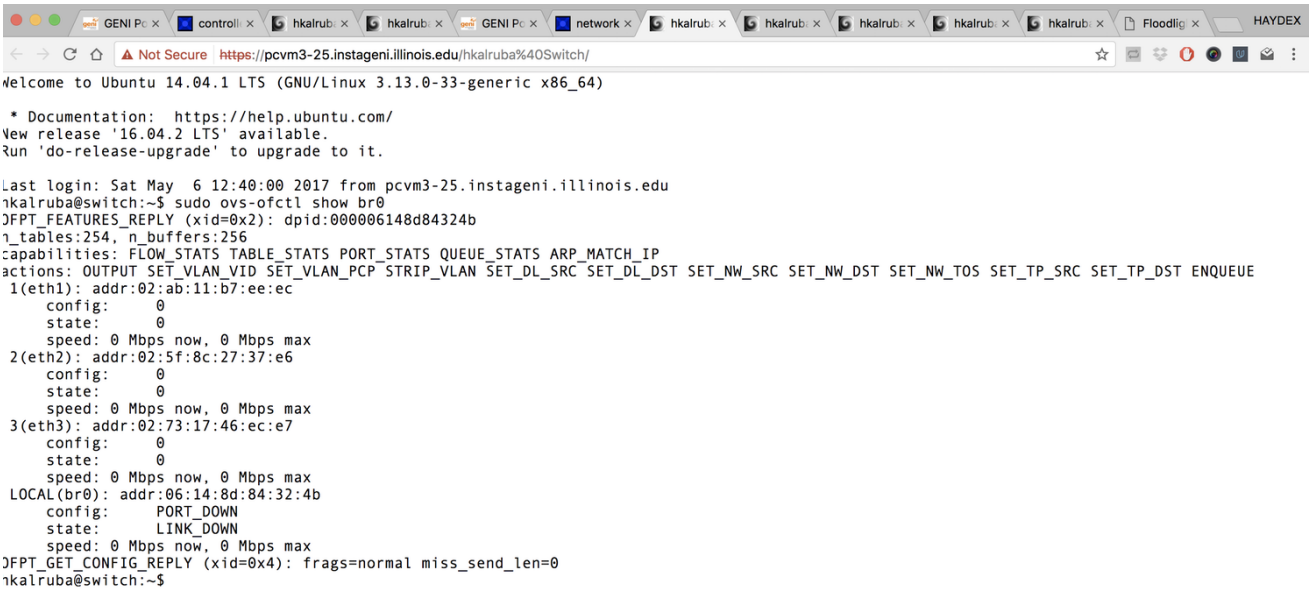

Figure 55 Show DPID of the OVS

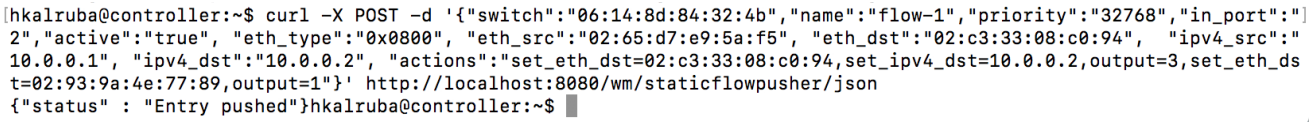

Figure 56 Duplication controller

To see that duplication is happening, on Host2 and Host3 , we run: sudo tcpdump -i eth1 (Figure 57)

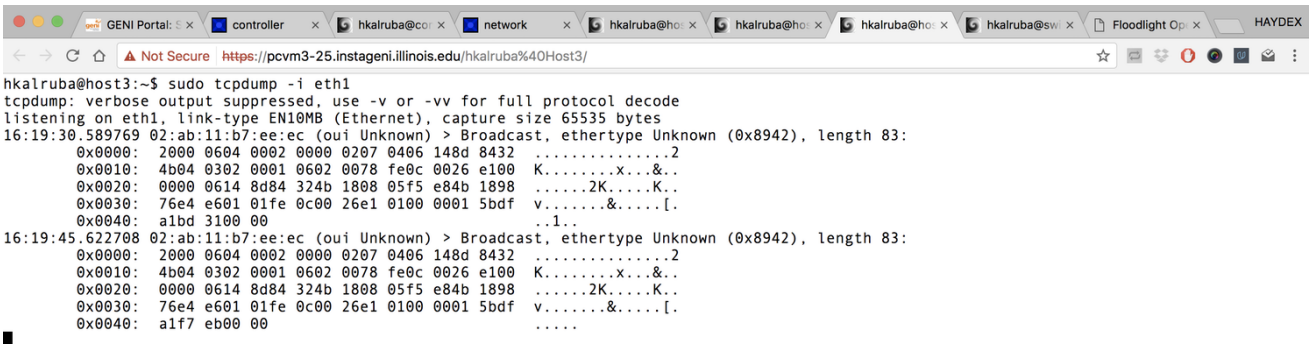

Figure 57 Host3 tcpdump

Now we see traffic from host1 to host2 showing up in the tcpdump window for host3 as shown in (Figure 57)

4g. Running a Port Forwarding Controller

Now let's do a slightly more complicated controller. OpenFlow gives the power to overwrite fields of your packets at the switch, for example the TCP source or destination port and do port forwarding. We can have clients trying to contact a server at port 5000, and the OpenFlow switch can redirect your traffic to a service listening on port 6000.

1. To test the controller we are going to use netcat. We open two terminals window on host2. In one terminal we run: (Figure 58)

```
nc -l 5000
```

and in the other terminal we run

```
nc -l 6000
```

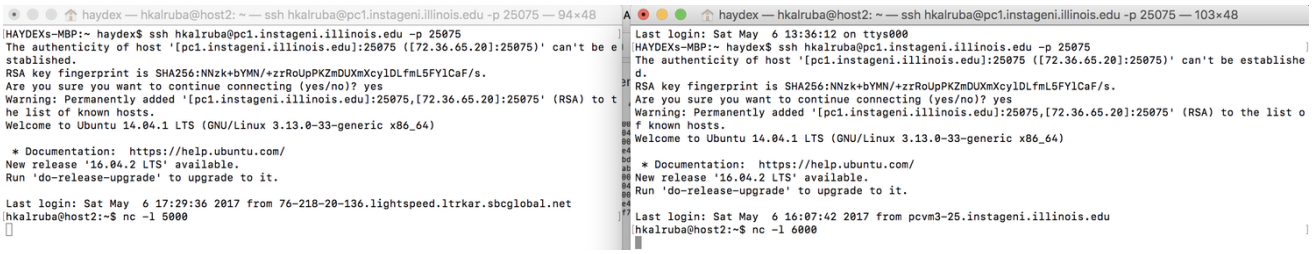

Figure 58

2. We will check the normal functionality before the flow for a Port Forwarding Controller is inserted. We go to the terminal of host1 and connect to host2 at port 5000:

```
nc 10.0.0.2 5000
```

3. If we type something and we should see it in the terminal on host2 at port 5000. (Figure 59)

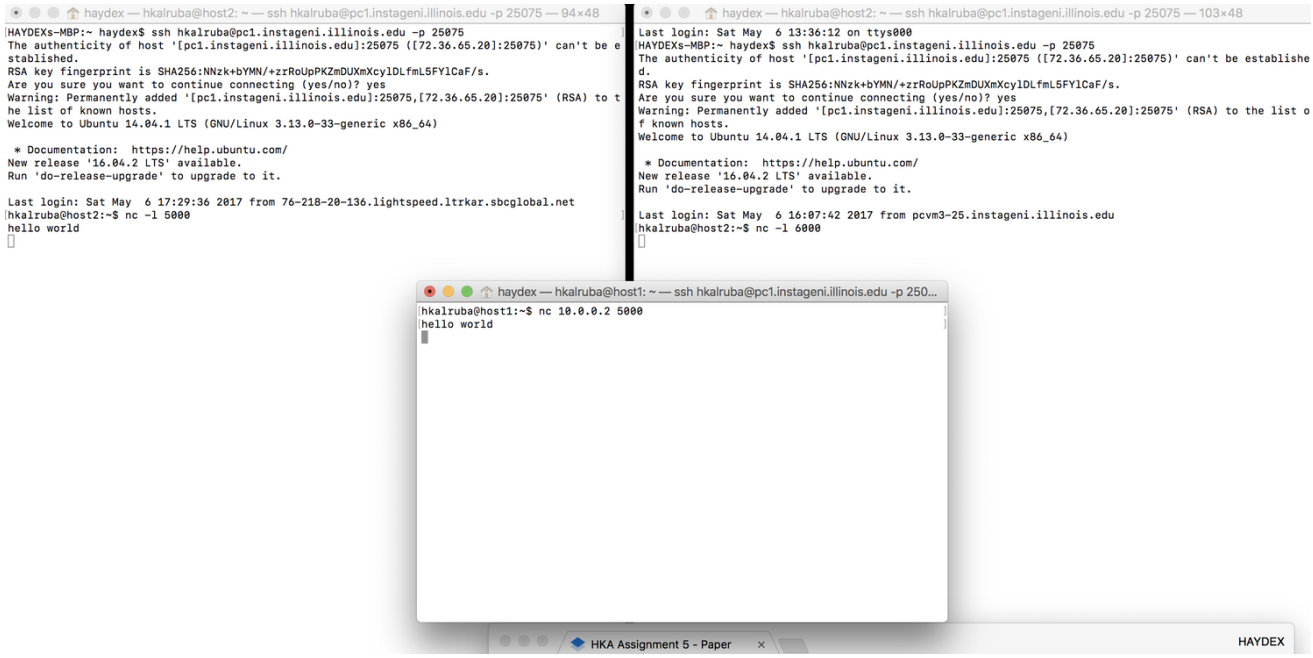

Figure 59

4. Now, we insert the flow for a Port Forwarding Controller: (Figure 60)

```
curl -X POST -d '{"switch":"<DPID OF OPEN vSWITCH>","name":"flow-2","priorit
y":"32768","in_port":"<PORT OF 10.0.0.1>","active":"true", "eth_type":"0x080
0", "ip_proto":"0x06", "eth_src":"<MAC OF 10.0.0.1>", "eth_dst":"<MAC OF 10.
0.0.2>", "tp_dst":"5000", "ipv4_src":"10.0.0.1", "ipv4_dst":"10.0.0.2", "act
ions":"set_tp_dst=6000,output=<PORT OF 10.0.0.2>"}' http://localhost:8080/w
m/staticflowpusher/json
```

```
curl -X POST -d '{"switch":"00:00:06:14:8d:84:32:4b","name":"flow-2","priori
ty":"32768","in_port":"2","active":"true", "eth_type":"0x0800",
"ip_proto":"0x06", "eth_src":"02:65:d7:e9:5a:f5", "eth_dst":"02:c3:33:08:c0:
94", "tp_dst":"5000", "ipv4_src":"10.0.0.1", "ipv4_dst":"10.0.0.2", "action
s":"set_tp_dst=6000,output=3"}' http://localhost:8080/wm/staticflowpusher/js
on
```

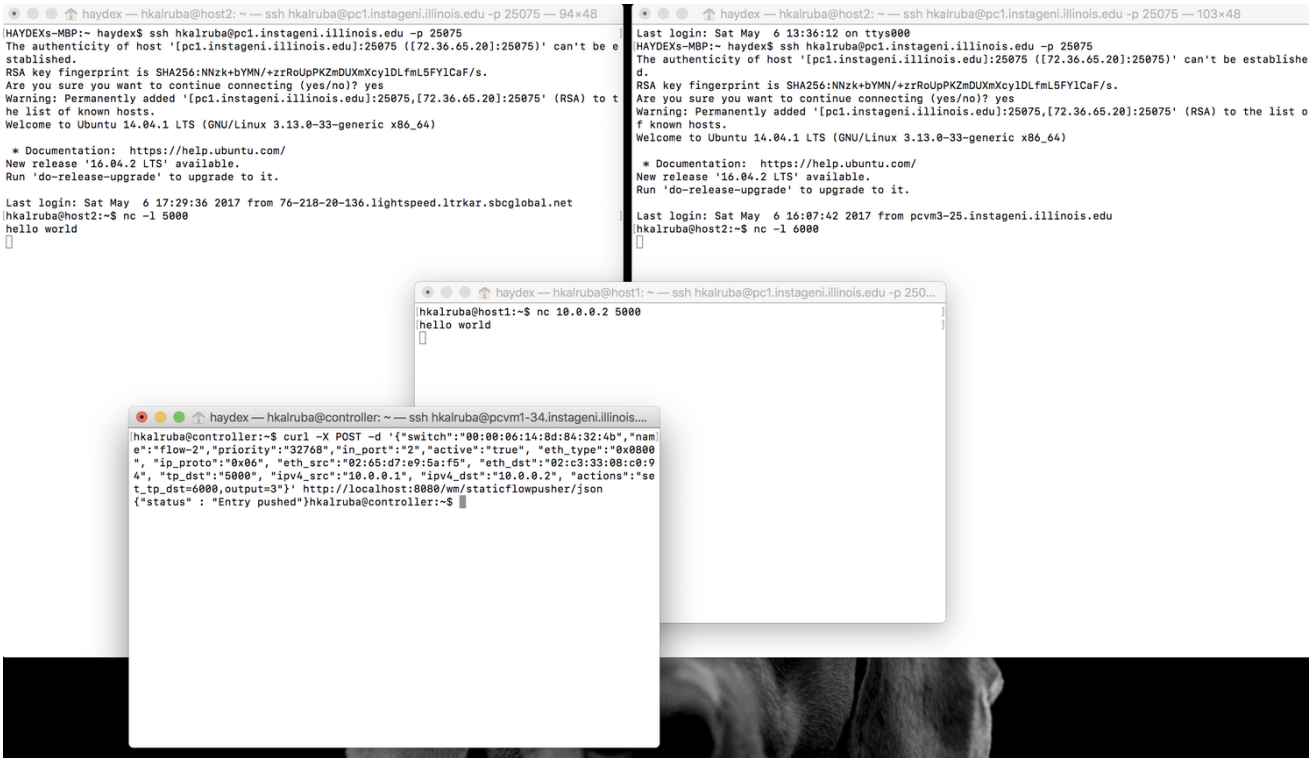

Figure 60

5. In the previous step, we inserted a flow to forward TCP traffic from Host1 destined to Host2 at port 5000 to port 6000. But Host1 still thinks it is speaking to Host2 at port 5000. So we need to insert a flow to handle traffic from Host2 Port 6000 for a seamless transition

```
curl -X POST -d '{"switch":"<DPID OF OPEN vSWITCH>","name":"flow-3","priorit
y":"32768","in_port":"<PORT OF 10.0.0.2>","active":"true", "eth_type":"0x080
0", "ip_proto":"0x06", "eth_src":"<MAC OF 10.0.0.2>", "eth_dst":"<MAC OF 10.
0.0.1>", "tp_src":"6000", "ipv4_src":"10.0.0.2", "ipv4_dst":"10.0.0.1", "act
ions":"set_tp_src=5000,output=<PORT OF 10.0.0.1>"}' http://localhost:8080/w
m/staticflowpusher/json
```

```
curl -X POST -d '{"switch":"00:00:06:14:8d:84:32:4b","name":"flow-3","priori
ty":"32768","in_port":"3","active":"true", "eth_type":"0x0800",
"ip_proto":"0x06", "eth_src":"02:c3:33:08:c0:94", "eth_dst":"02:65:d7:e9:5a:
f5", "tp_src":"6000", "ipv4_src":"10.0.0.2", "ipv4_dst":"10.0.0.1", "action
s":"set_tp_src=5000,output=2"}' http://localhost:8080/wm/staticflowpusher/js
on
```

4h. Running a Server Proxy Controller

3. We need to nsert the following flow in the Controller terminal to implement a Server Proxy Controller:

```
curl -X POST -d '{"switch":"<DPID OF OPEN vSWITCH>","name":"flow-4","priorit
y":"32768","in_port":"<PORT OF 10.0.0.1>","active":"true", "eth_type":"0x080
0", "ip_proto":"0x06", "eth_src":"<MAC OF 10.0.0.1>", "eth_dst":"<MAC OF 10.
0.0.2>", "tp_dst":"6000", "ipv4_src":"10.0.0.1", "ipv4_dst":"10.0.0.2", "act
ions":"set_eth_dst=<MAC OF 10.0.0.3>,set_tp_dst=7000,set_ipv4_dst=10.0.0.3,o
utput=<PORT OF 10.0.0.3>"}' http://localhost:8080/wm/staticflowpusher/json
```

```
curl -X POST -d '{"switch":"00:00:06:14:8d:84:32:4b","name":"flow-4","priori
ty":"32768","in_port":"2","active":"true", "eth_type":"0x0800",
"ip_proto":"0x06", "eth_src":"02:65:d7:e9:5a:f5", "eth_dst":"02:c3:33:08:c0:
94", "tp_dst":"6000", "ipv4_src":"10.0.0.1", "ipv4_dst":"10.0.0.2", "action
s":"set_eth_dst=02:93:9a:4e:77:89,set_tp_dst=7000,set_ipv4_dst=10.0.0.3,outp
ut=1"}' http://localhost:8080/wm/staticflowpusher/json
```

```
hkalruba@controller:~$ curl -X POST -d '{"switch":"00:00:06:14:8d:84:32:4b","name":"flow-4","priority":"32768","in_port":"2","act
ive":"true", "eth_type":"0x0800", "ip_proto":"0x06", "eth_src":"02:65:d7:e9:5a:f5", "eth_dst":"02:c3:33:08:c0:94", "tp_dst":"6000
", "ipv4_src":"10.0.0.1", "ipv4_dst":"10.0.0.2", "actions":"set_eth_dst=02:93:9a:4e:77:89,set_tp_dst=7000,set_ipv4_dst=10.0.0.3,o
utput=1"}' http://localhost:8080/wm/staticflowpusher/json
```

Figure 61 Flow-4

4. In the previous step, we inserted a flow to forward TCP traffic from Host1 destined to Host2 at port 5000 to Host 3 at port 6000. But Host1 still thinks it is speaking to Host2 at port 5000. So we need to insert a flow to handle traffic from Host3 Port 6000 for a seamless transition.

```
curl -X POST -d '{"switch":"<DPID OF OPEN vSWITCH>","name":"flow-5","priorit
y":"32768","in_port":"<PORT OF 10.0.0.3>","active":"true", "eth_type":"0x080
0", "ip_proto":"0x06", "eth_src":"<MAC OF 10.0.0.3>", "eth_dst":"<MAC OF 10.
0.0.1>", "tp_src":"7000", "ipv4_src":"10.0.0.3", "ipv4_dst":"10.0.0.1", "act
ions":"set_eth_src=<MAC OF 10.0.0.2>,set_ipv4_src=10.0.0.2,set_tp_src=6000,o
utput=<PORT OF 10.0.0.1>"}' http://localhost:8080/wm/staticflowpusher/json
```

```
curl -X POST -d '{"switch":"00:00:06:14:8d:84:32:4b","name":"flow-5","priori
ty":"32768","in_port":"1","active":"true", "eth_type":"0x0800",
"ip_proto":"0x06", "eth_src":"02:93:9a:4e:77:89", "eth_dst":"02:65:d7:e9:5a:
f5", "tp_src":"7000", "ipv4_src":"10.0.0.3", "ipv4_dst":"10.0.0.1", "action
s":"set_eth_src=02:c3:33:08:c0:94,set_ipv4_src=10.0.0.2,set_tp_src=6000,outp
ut=2"}' http://localhost:8080/wm/staticflowpusher/json
```

```
hkalruba@controller:~$ curl -X POST -d '{"switch":"00:00:06:14:8d:84:32:4b","name":"flow-5","priority":"32768","in_port":"1","act
ive":"true", "eth_type":"0x0800", "ip_proto":"0x06", "eth_src":"02:93:9a:4e:77:89", "eth_dst":"02:65:d7:e9:5a:f5", "tp_src":"7000
", "ipv4_src":"10.0.0.3", "ipv4_dst":"10.0.0.1", "actions":"set_eth_src=02:c3:33:08:c0:94,set_ipv4_src=10.0.0.2,set_tp_src=6000,o
utput=2"}' http://localhost:8080/wm/staticflowpusher/json
{"status" : "Entry pushed"}hkalruba@controller:~$
```

Figure 62 Flow-5

5. We go back to the terminal of host1 and try to connect netcat to host2 port 6000 (Figure 63)

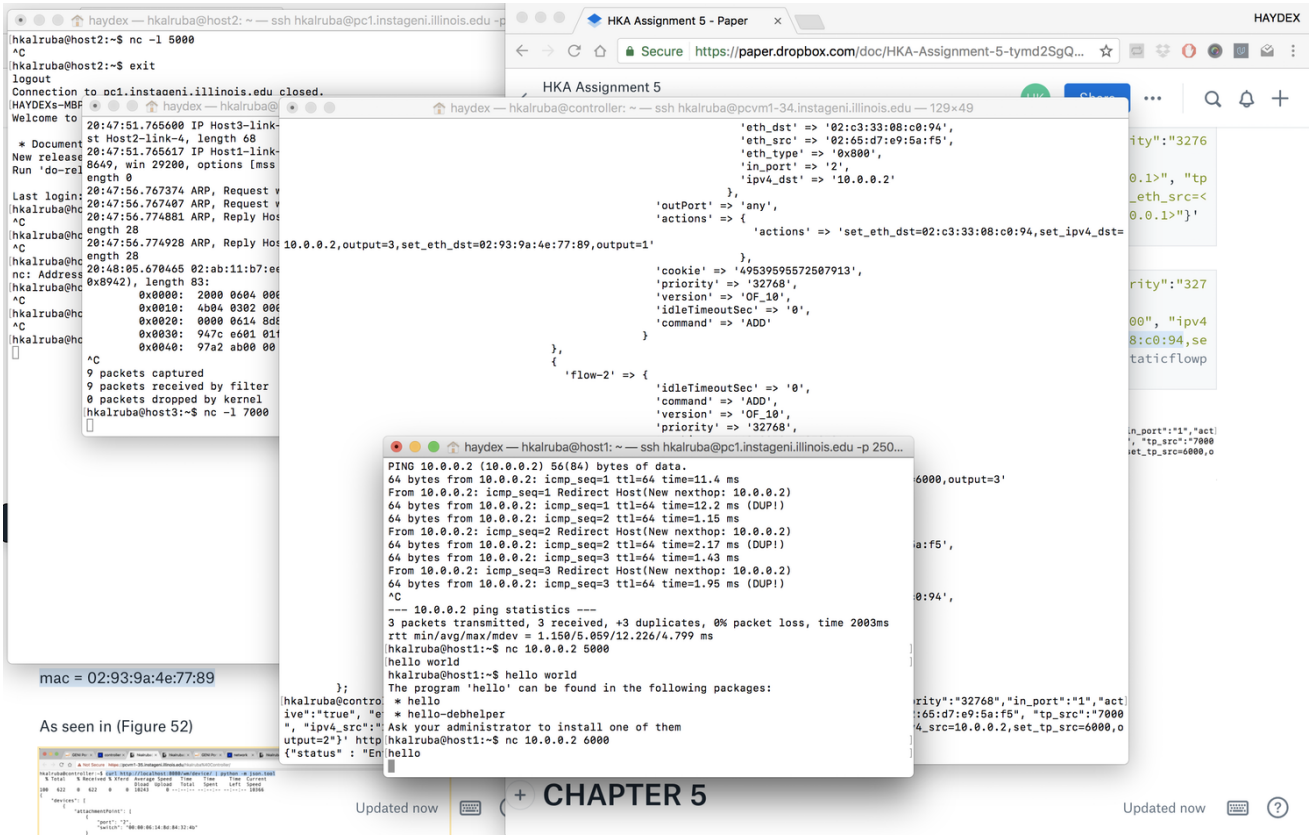

Figure 63 Testing the flow from host1 to host3

6. If the controller works correctly, we should have seen text appearing on the terminal window of host3. (Unsuccessful! there was no text appearing from host1 to host3)

4i. Deleting the bridge

haydex — hkalruba@switch: ~ — ssh hkalruba@pc1.instageni.illinois.edu -p 25...

```
     speed: 0 Mbps now, 0 Mbps max
 LOCAL(br0): addr:06:14:8d:84:32:4b
     config:     PORT_DOWN
     state:      LINK_DOWN
     speed: 0 Mbps now, 0 Mbps max
OFPT_GET_CONFIG_REPLY (xid=0x4): frags=normal miss_send_len=0
hkalruba@switch:~$ sudo ovs-vsctl show
89362c3a-1d64-451b-977d-7b2718a1ba20
    Bridge "br0"
        Controller "tcp:72.36.65.65:6633"
            is_connected: true
        fail_mode: secure
        Port "eth1"
            Interface "eth1"
        Port "eth3"
            Interface "eth3"
        Port "br0"
            Interface "br0"
                type: internal
        Port "eth2"
            Interface "eth2"
    ovs_version: "2.3.1"
hkalruba@switch:~$ sudo ovs-vsctl del-br br0
hkalruba@switch:~$
```

Figure 64 Deleting the bridge in the OVS

5. FINISH : Deleting the resources in the slices.

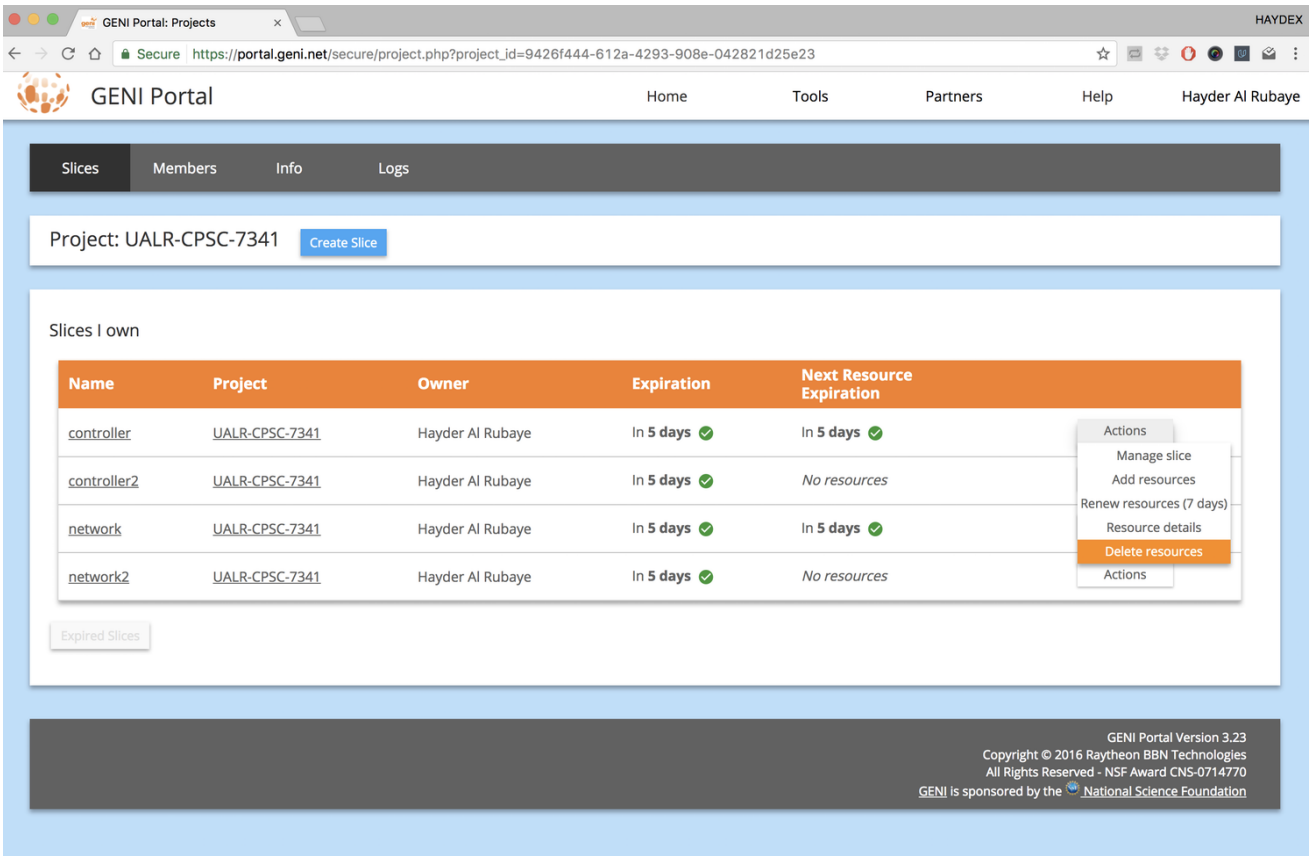

Figure 65

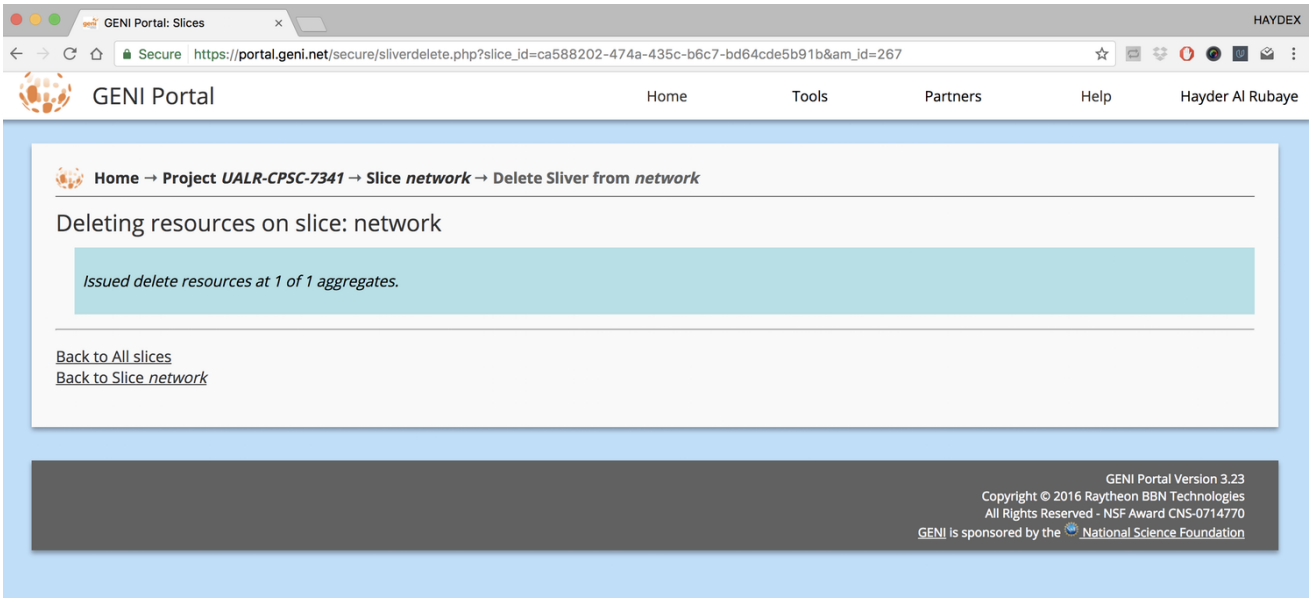

Figure 66

PROBLEMS

CHAPTER 5

Problem 3:
Solution:
The ASCII code for the word “Networking”
N e t w o r k i n g
078 101 116 119 111 114 107 105 110 103
The binary representation of these numbers is
01001110 01100101 01110100 01110111 01101111 01110010 01101011 01101001 01101110 0110 0111

Checksum Calculation:

0100 1110 0110 0101 + 0111 0100 0111 0111 = 1100 0010 1101 1100
1100 0010 1101 1100 + 0110 1111 0111 0010 =1 0011 0010 0100 1110
(overflow, wrap around) => 0011 0010 0100 1111
0011 0010 0100 1111 + 0110 1011 0110 1001 = 1001 1101 1011 1000 1001 1101 1011
1000+0110 1110 0110 0111 = 1 0000 1100 0001 1111 =>0000 1100 0001 1111
(overflow)
1’s complement of 0000 1100 0001 1111 = 1111 0011 1110 0000.

Problem 5:
Solution:

G=10011
then r = 4
D + r 0s= 1010101010 0000
divide 10011 into 1010101010 0000,
Remainder of R=0100

Problem 10:
Solution:
a) The formula of A's average throughput is pA(1-pB).
Total efficiency is pA(1-pB) + pB(1-pA)

b)A's throughput = pA(1-pB) = 2pB(1-pB) = 2pB- 2(pB)2.
B's throughput = pB(1-pA) = pB(1-2pB) = pB- 2(pB)2.
A's throughput is not twice as large as B's.
To make pA(1-pB) = 2 pB(1-pA). pA= 2 – (pA / pB).

c) A's throughput = 2p(1-p)N-1.
Any other node's throughput = p(1-p)N- 2(1-2p)

Problem 17:
Solution:
for a 10Mbps = K*512*0.0001= 5.12 ms
for a 100Mbps = K*512*0.00001= 0.512ms

Problem 23:
Solution:
If all the 11= 9+2 nodes send out data at the maximum possible rate of 100 Mbps, a total aggregate throughput of 11*100 = 1100 Mbps is possible.