**CSE434 Lab 4 Report**

**Group 82**

**Group Members:**

**Jeet Thakkar**

**Wei Hng Yeo**

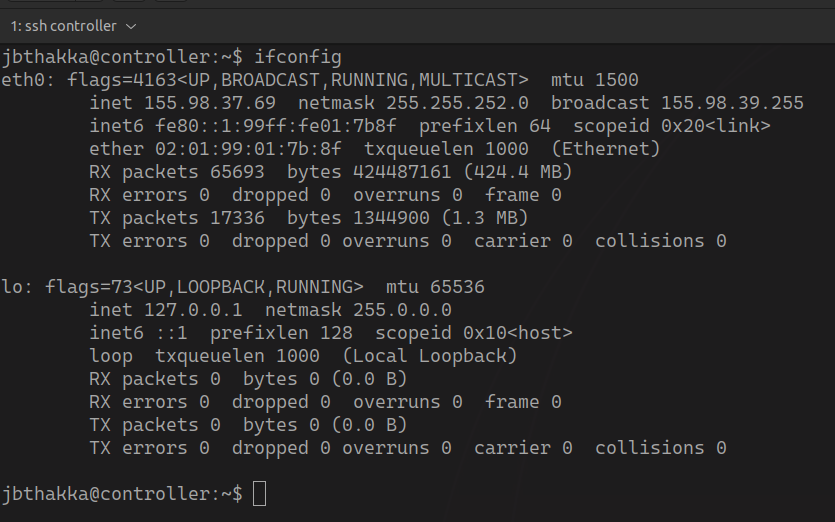
**Exercise 1.1**



Software Switch Configuration

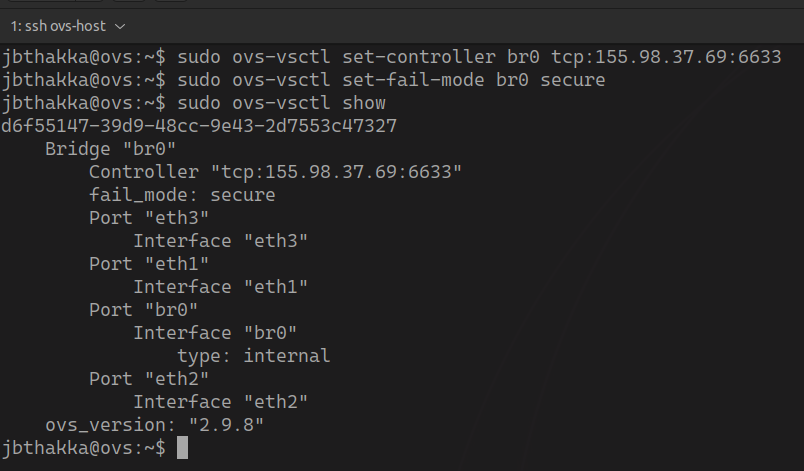
From the above screenshot, we can see the data interfaces for the three hosts are are added to the bridge created on the software switch.

**Exercise 1.2**

****

Ifconfig of Controller

The above screenshot shows the ifconfig of the controller.

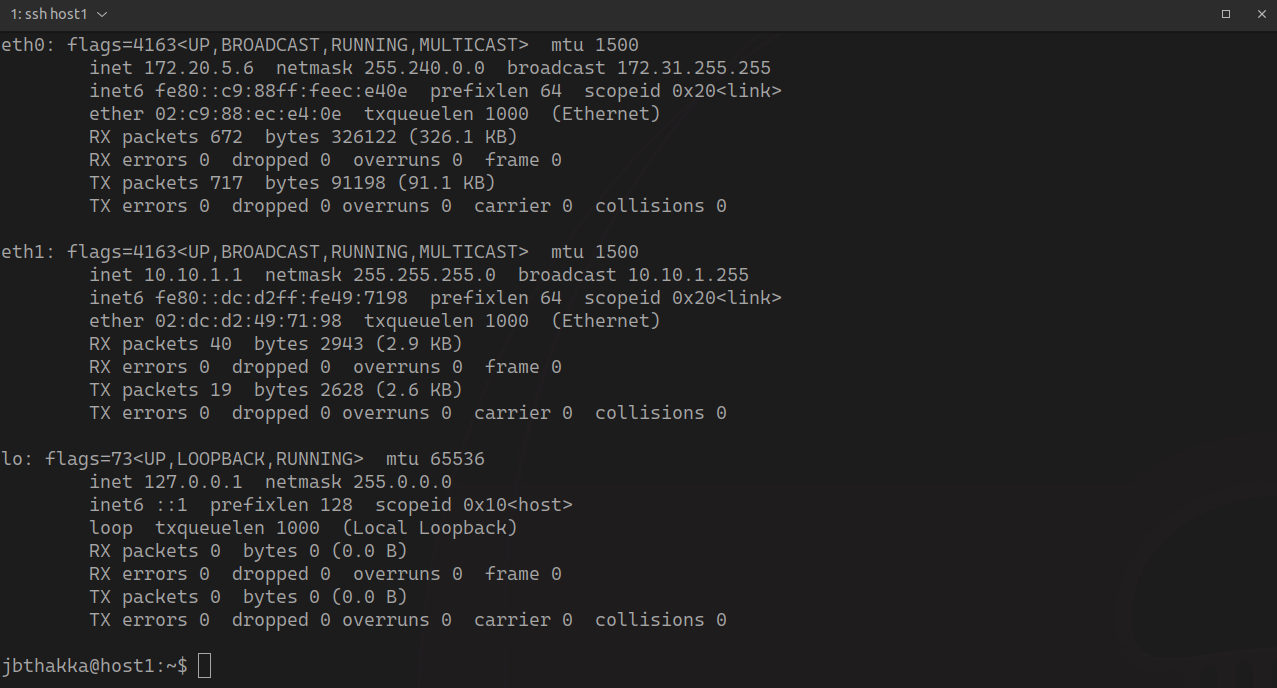
****

OVS settings shown

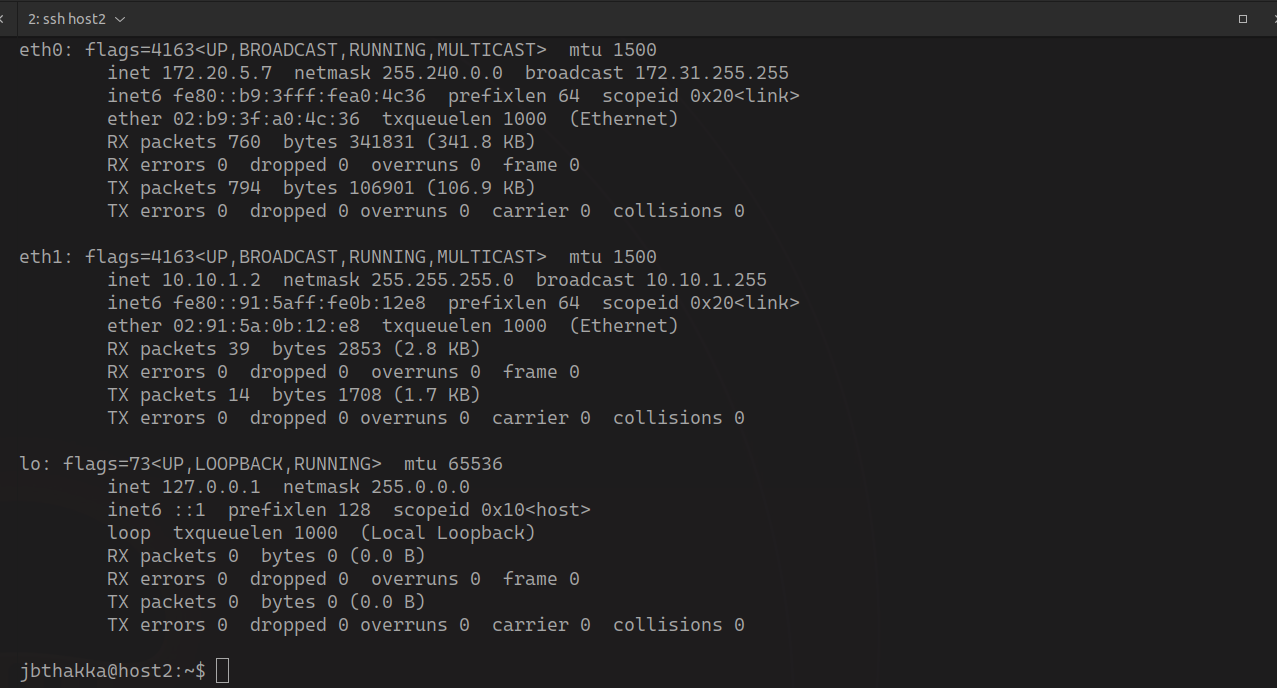
The above screenshot shows that the switch has been successfully pointed to the controller using the IP address, which corresponds to the IP address of the controller we’ve shown in the previous ifconfig screenshot.

**Exercise 1.3**

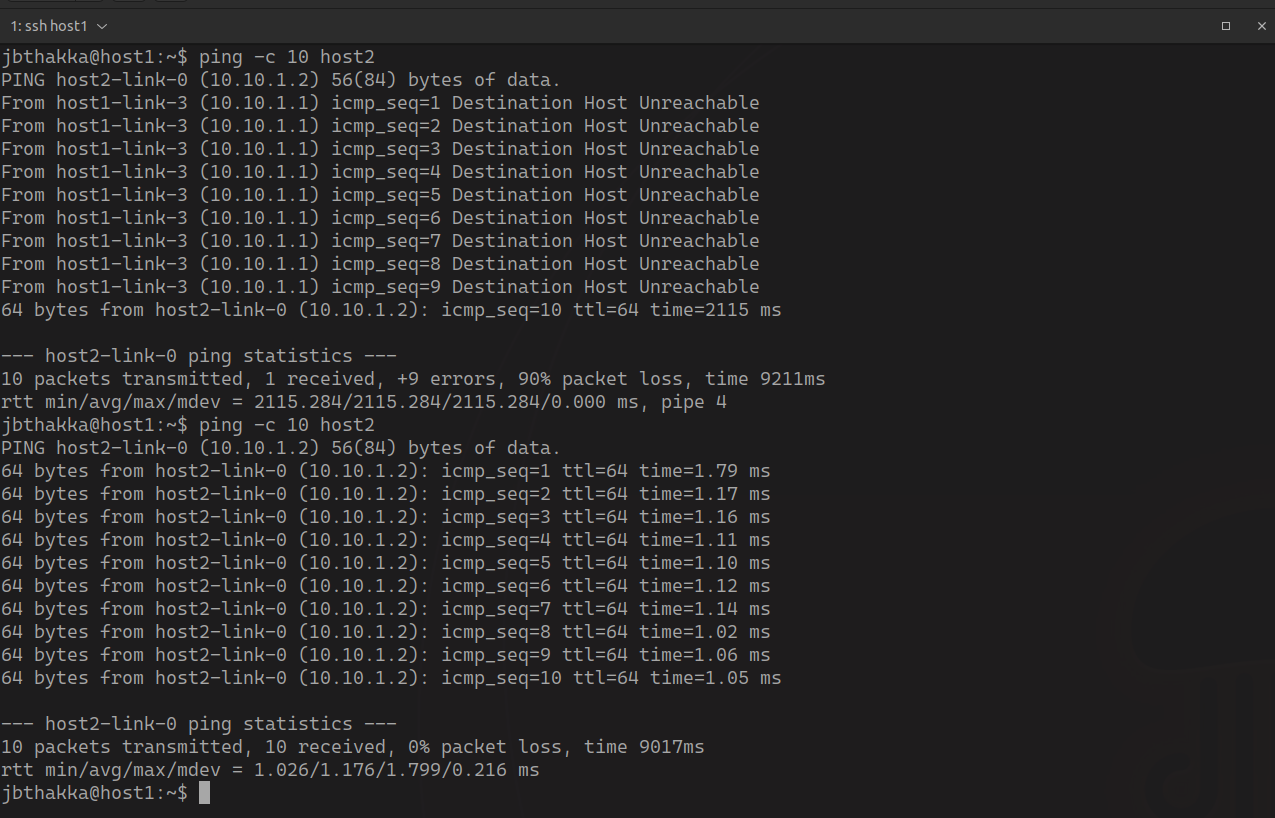
Host 1: IP and MAC address

****

Host 2: IP and MAC address

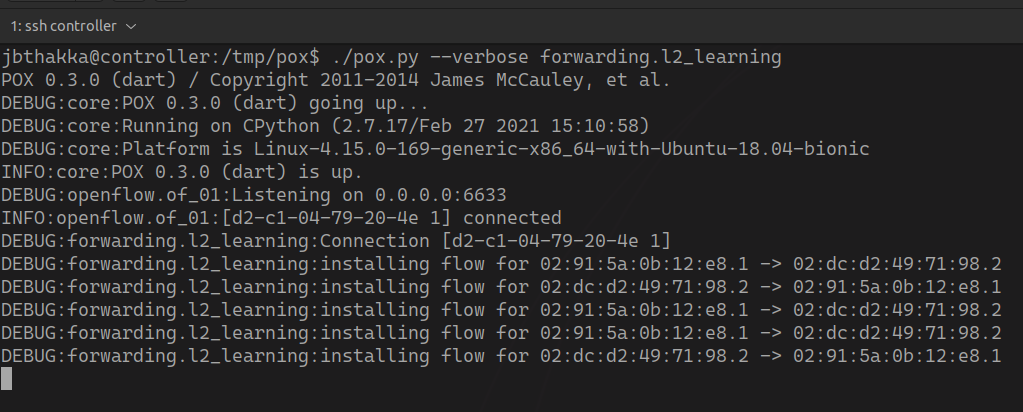
****

The ping from host 1 to host 2 shown below fails initially since the controller hasn’t started running yet. Once the controller starts running, the ping becomes successful because pox is initialized, and it sees that the software switch has received an ARP request from hostA for hostB - then it install the correct flows which make it possible for hostA to communicate with hostB.

****

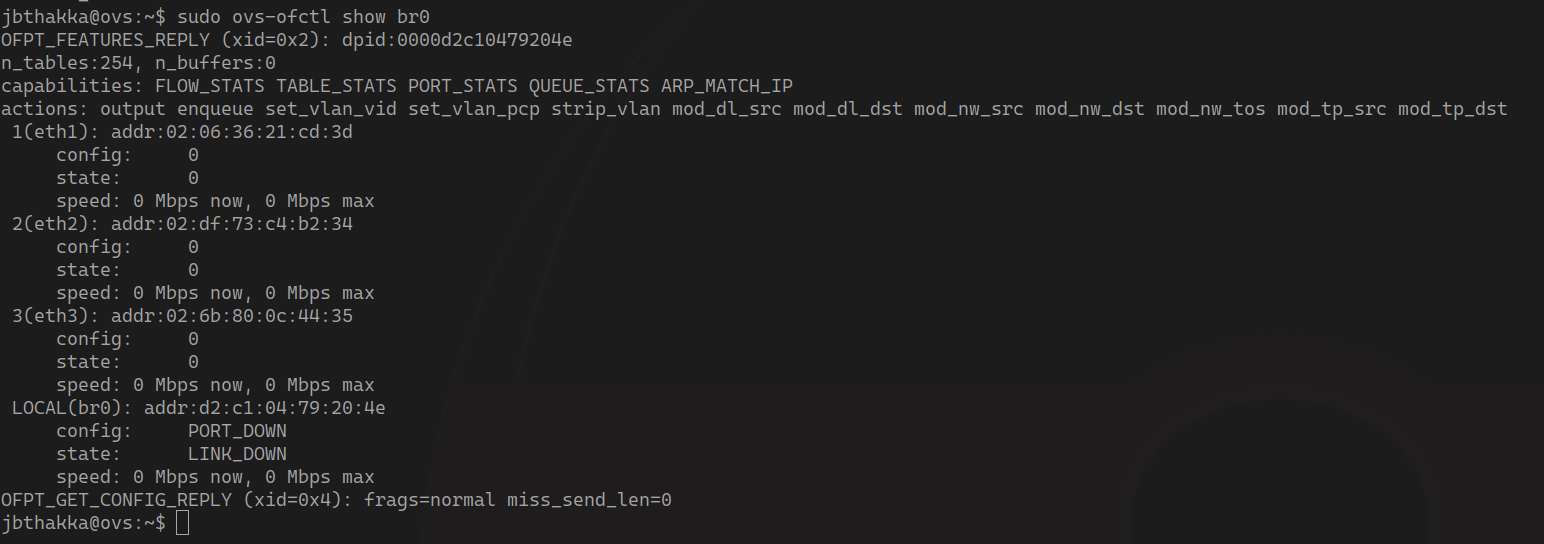
Ping output between host1 and host2

Controller pox initialization and output is shown below. The controller is installing flows on the software switch that allow traffic from from host 1 to host 2 and vice versa.

****

**Exercise 1.4**

Manifest from the openvswitch

****

Manifest

MAC addresses:

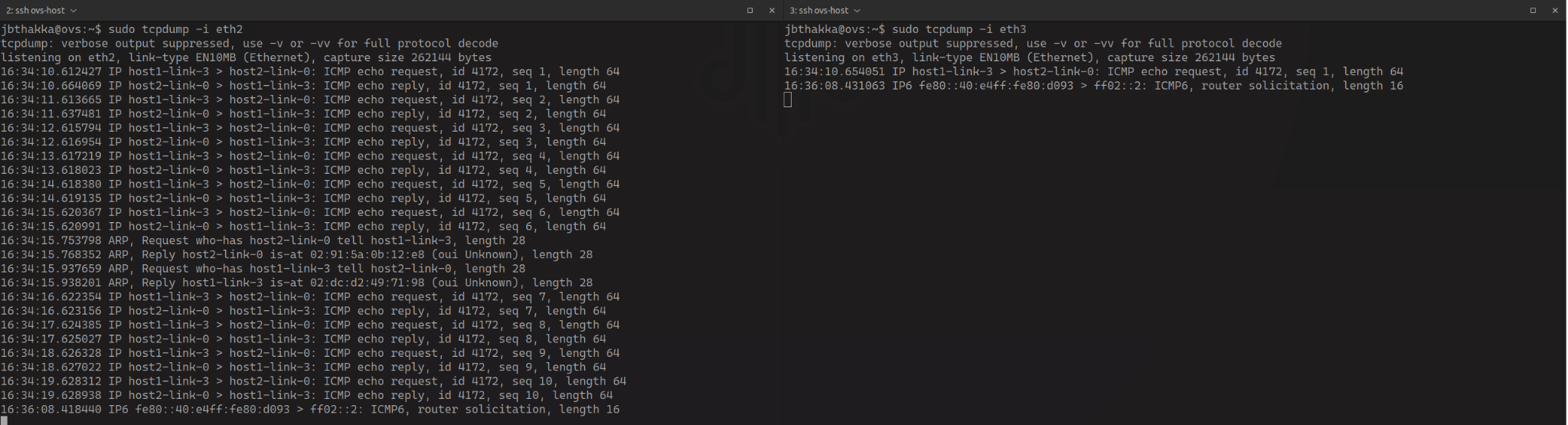
eth1@OVS - 02:06:36:21:cd:3d - 1 (eth1)

eth2@OVS - 02:df:73:c4:b2:34 - 2 (eth2)

eth3@OVS - 02:6b:80:0c:44:35 - 3 (eth3)

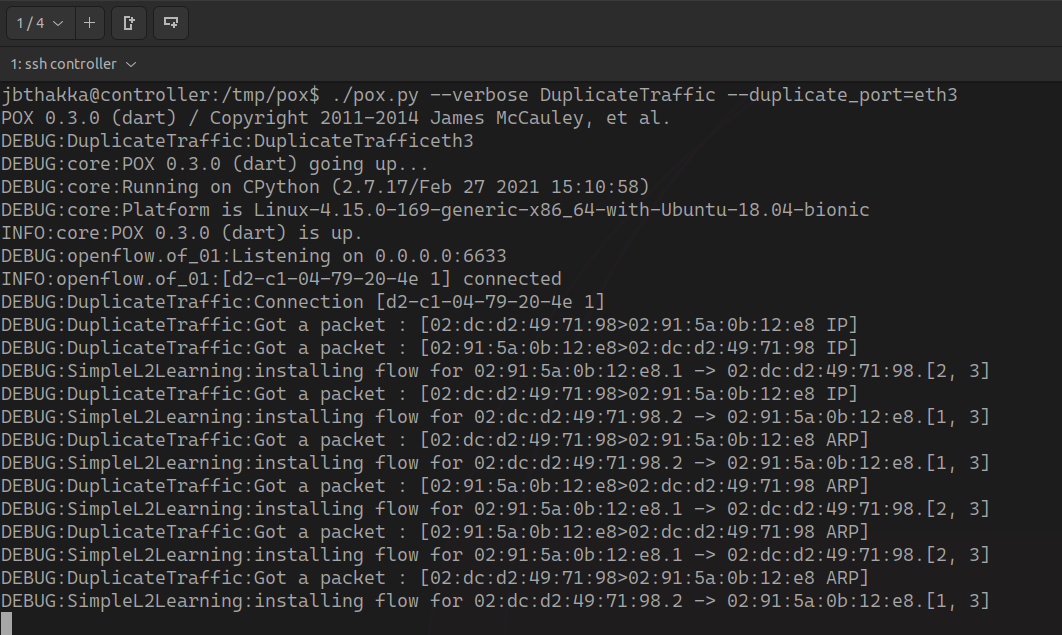
eth1@host1 - 02:dc:d2:49:71:98  
eth1@host2 - 02:91:5a:0b:12:e8

eth3@host3 - 02:b0:83:6c:35:fb

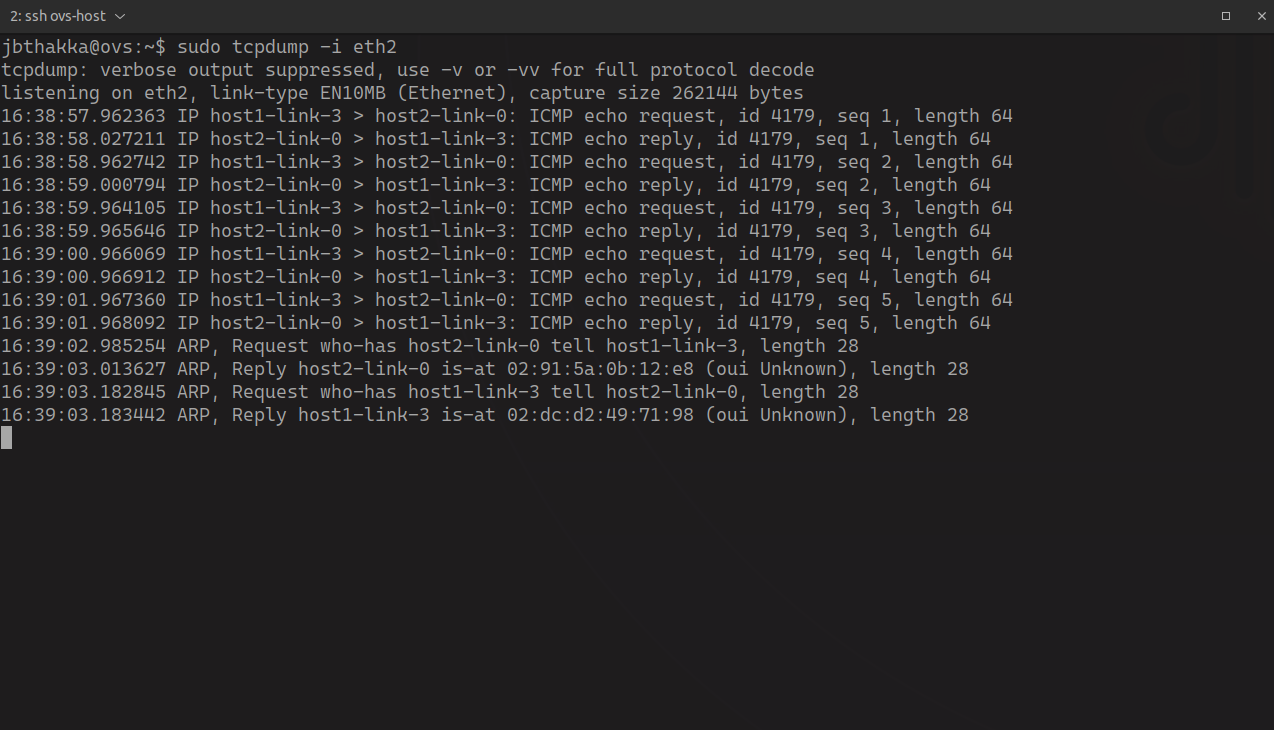
****

There is no duplication of traffic as shown on the above diagram when we simply run the learning switch program.

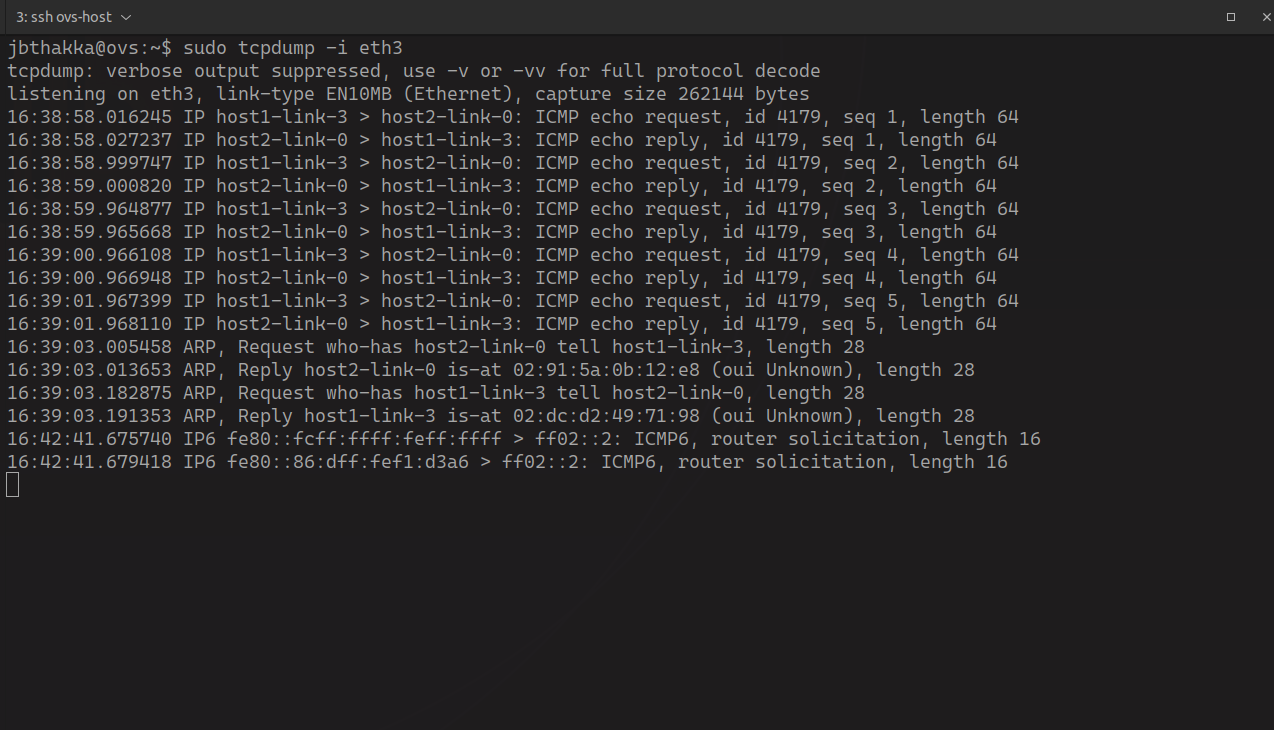
However when we run the DuplicateTraffic program with the eth3 specified as the duplicate interface we see the following output on the controller as shown below:

****

And the following traffic on host2:

****

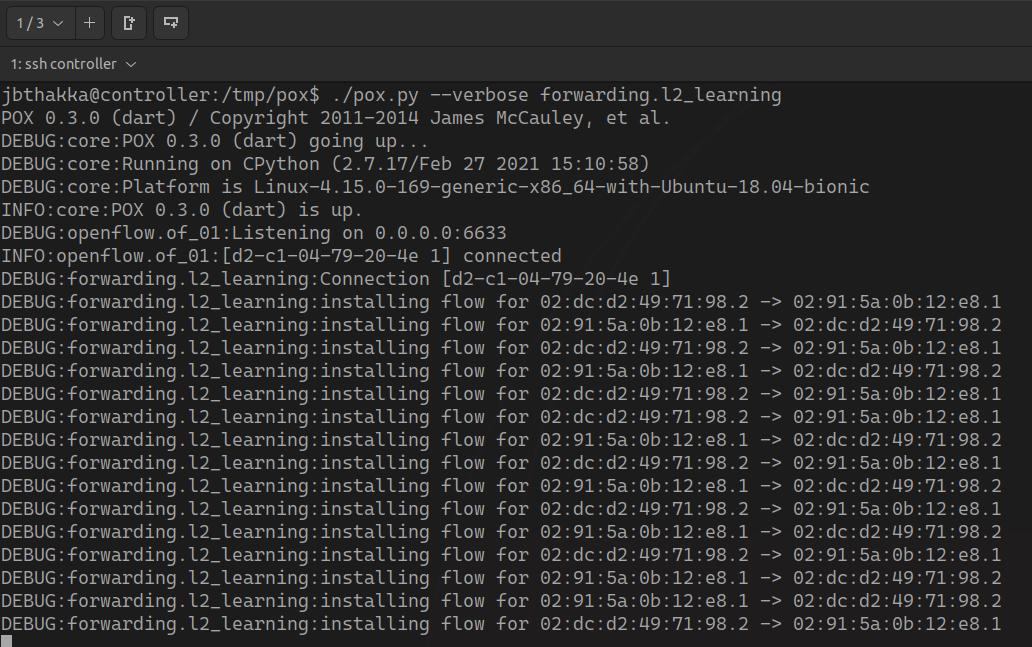
And the duplicated traffic on host3:

****

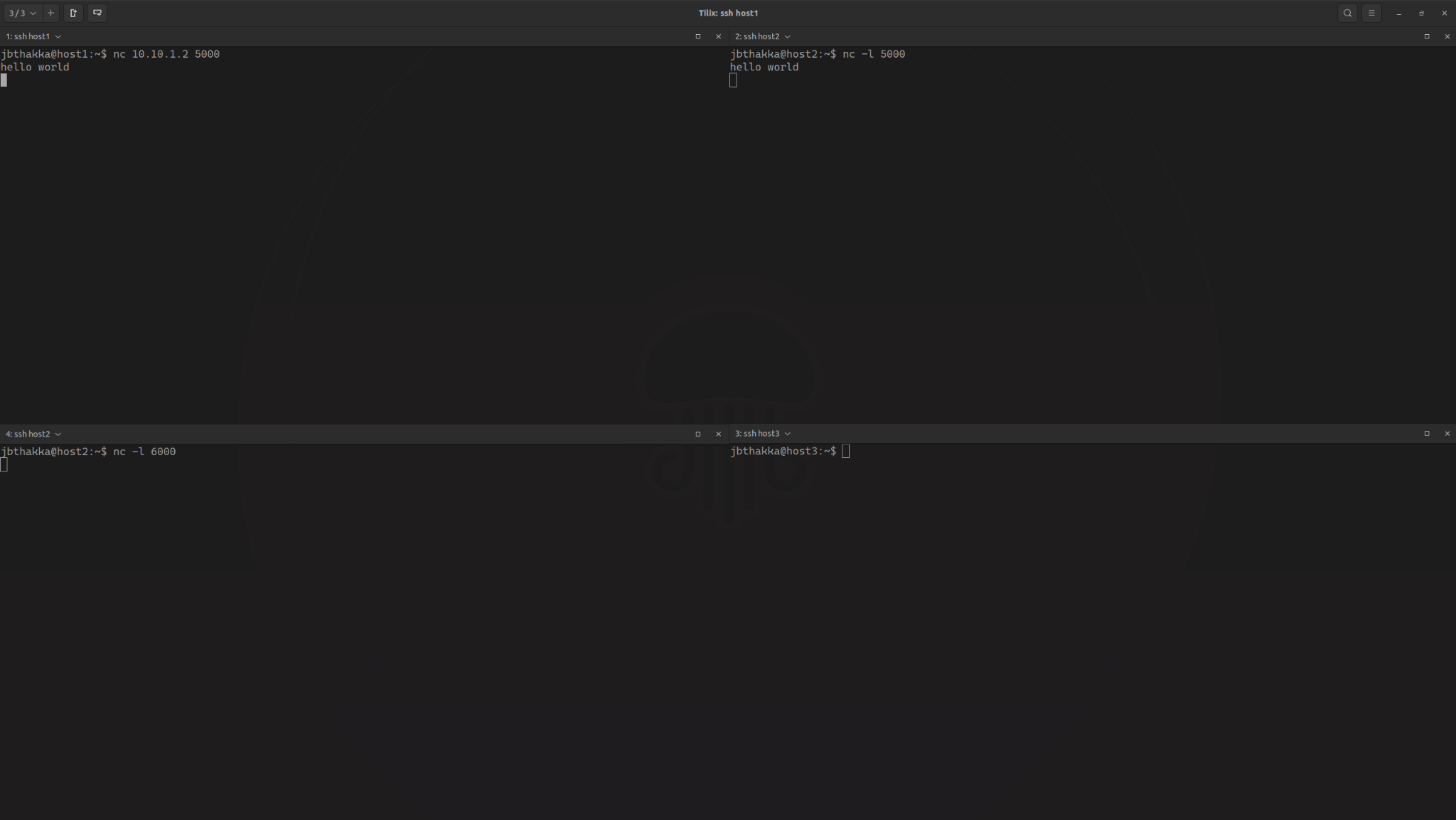
The controller receives a packet for host2, installs the correct flow to allow the packet to be delivered to host 2 and then runs a flow to duplicate the traffic to host 3.

**Exercise 1.5**

Before the rule for the port forwarding is inserted, when we send a message using netcat from host 1 to host 2, host 2 will receive the message on port 5000 - as it should go.

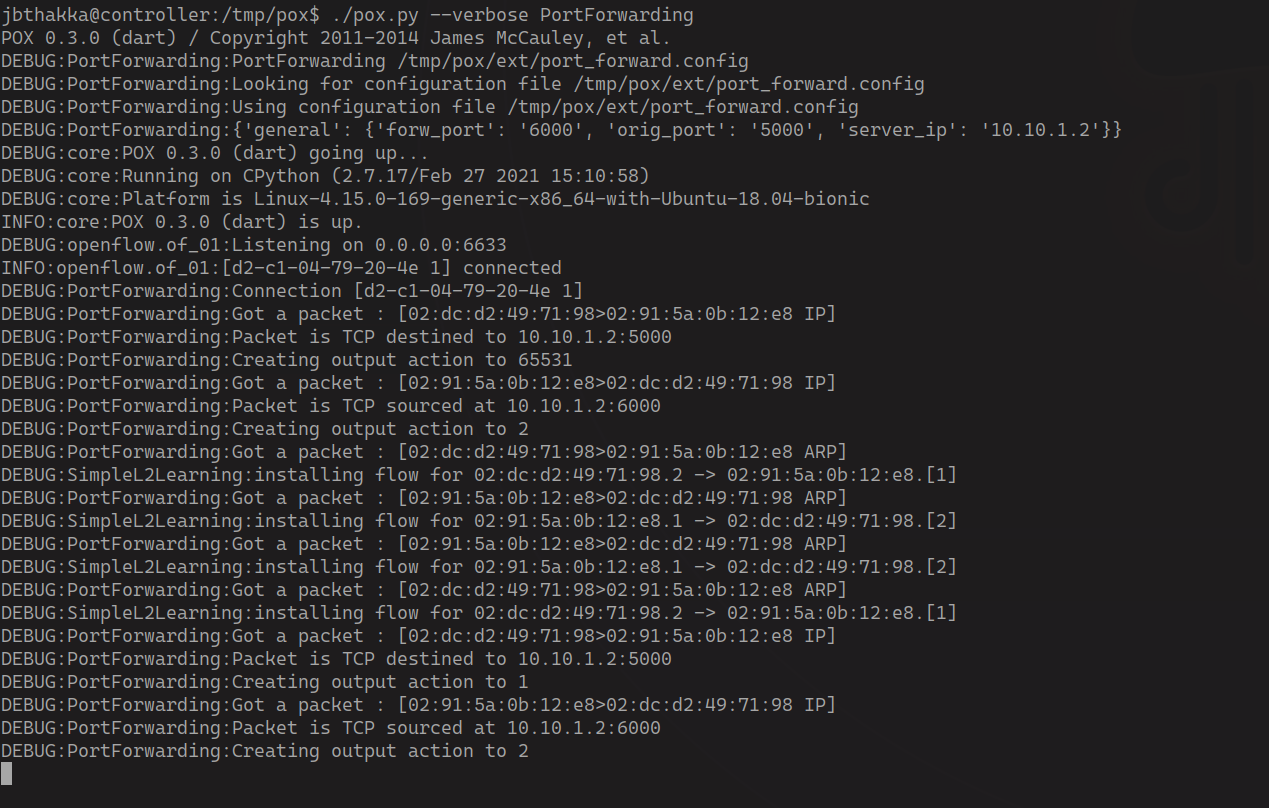


Using l2\_learning

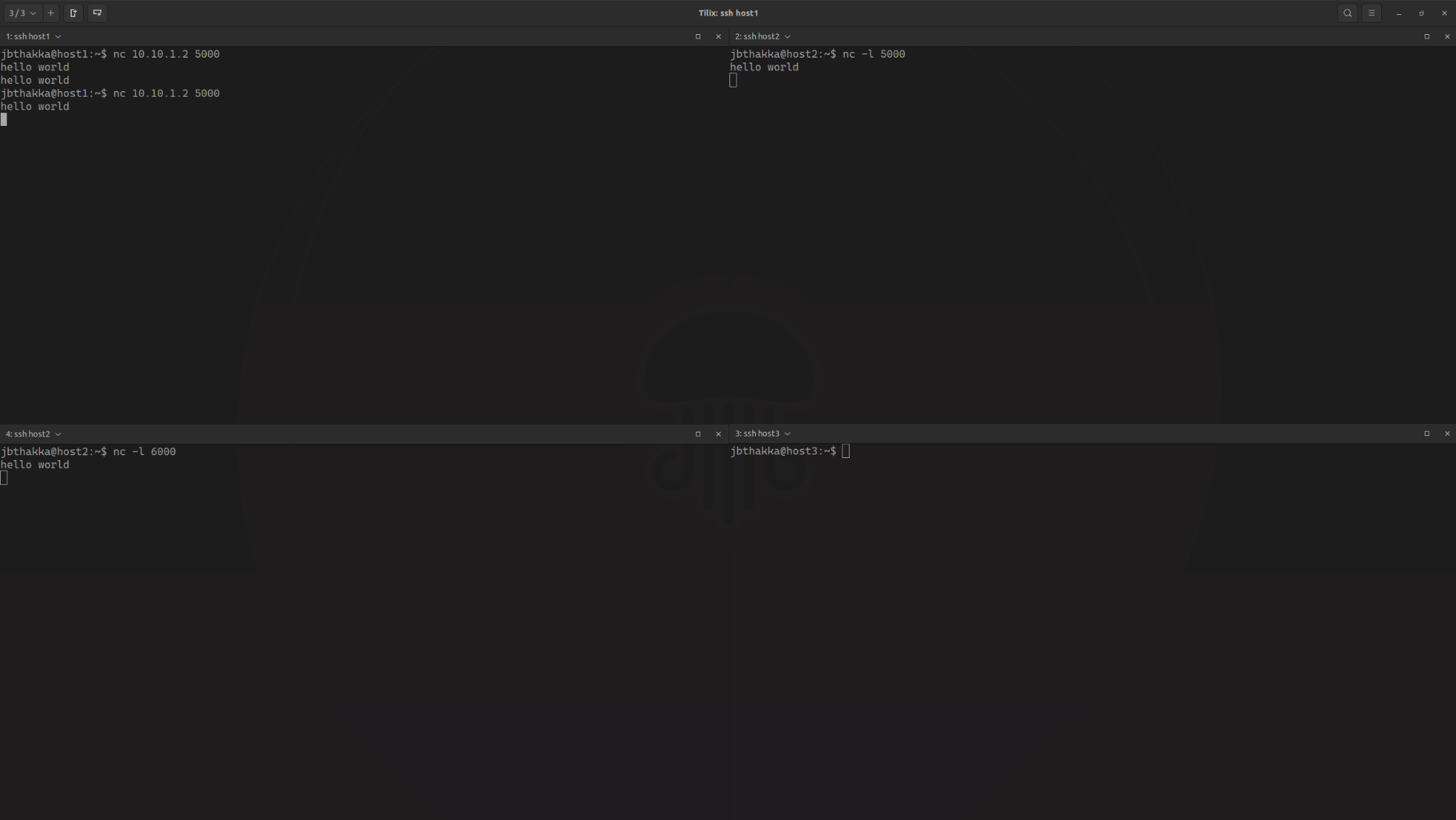


Netcat test

However, with port forwarding, when host 1 sends a message to host 2 on port 5000, the controller will forward the message to port 6000 on host 2, as shown below on the screenshot. The SDN rewrites the destination port on the packet and it gets delivered to the different port.



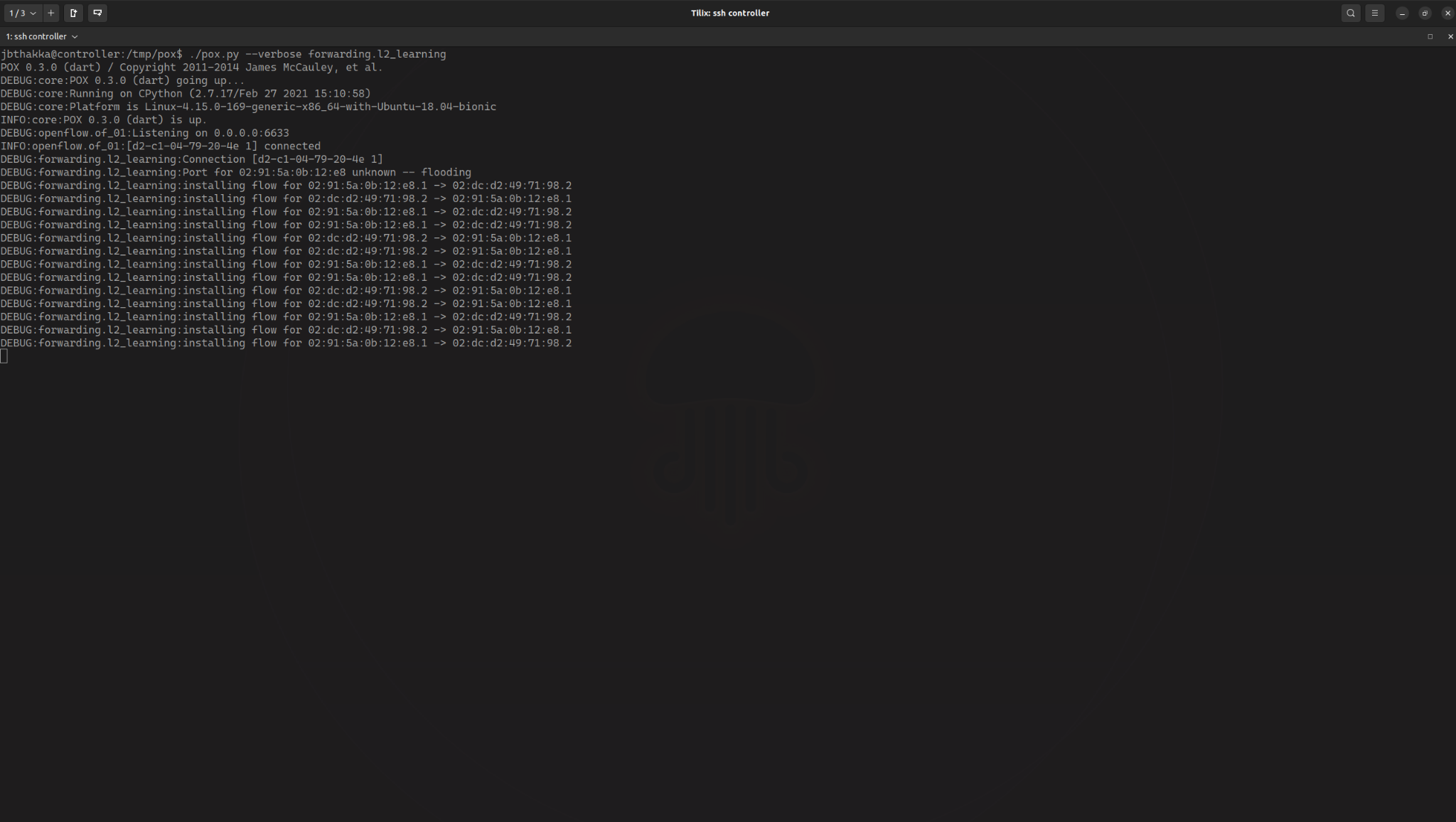
Using PortForwarding

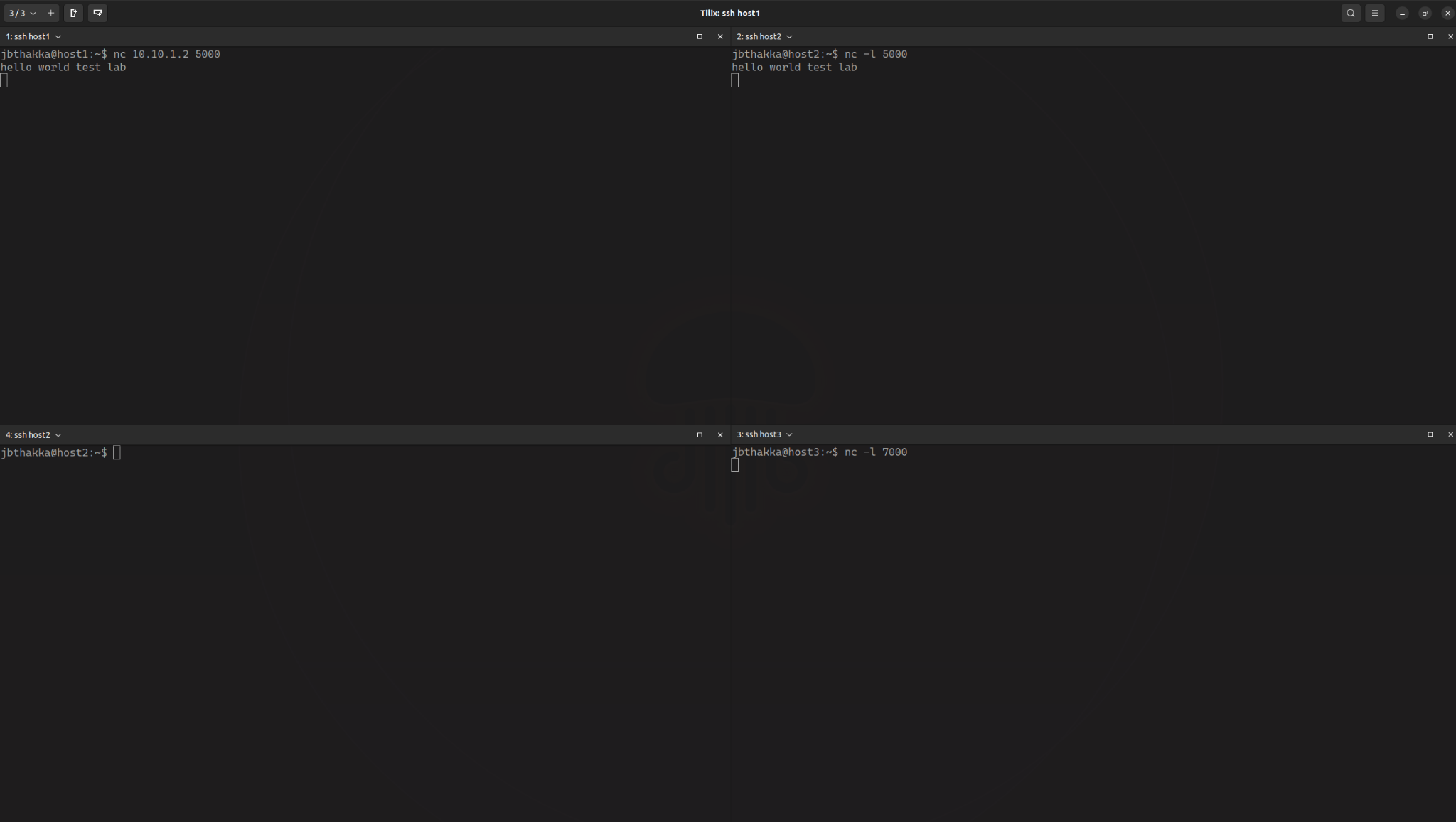


Netcat Test

**Exercise 1.6**

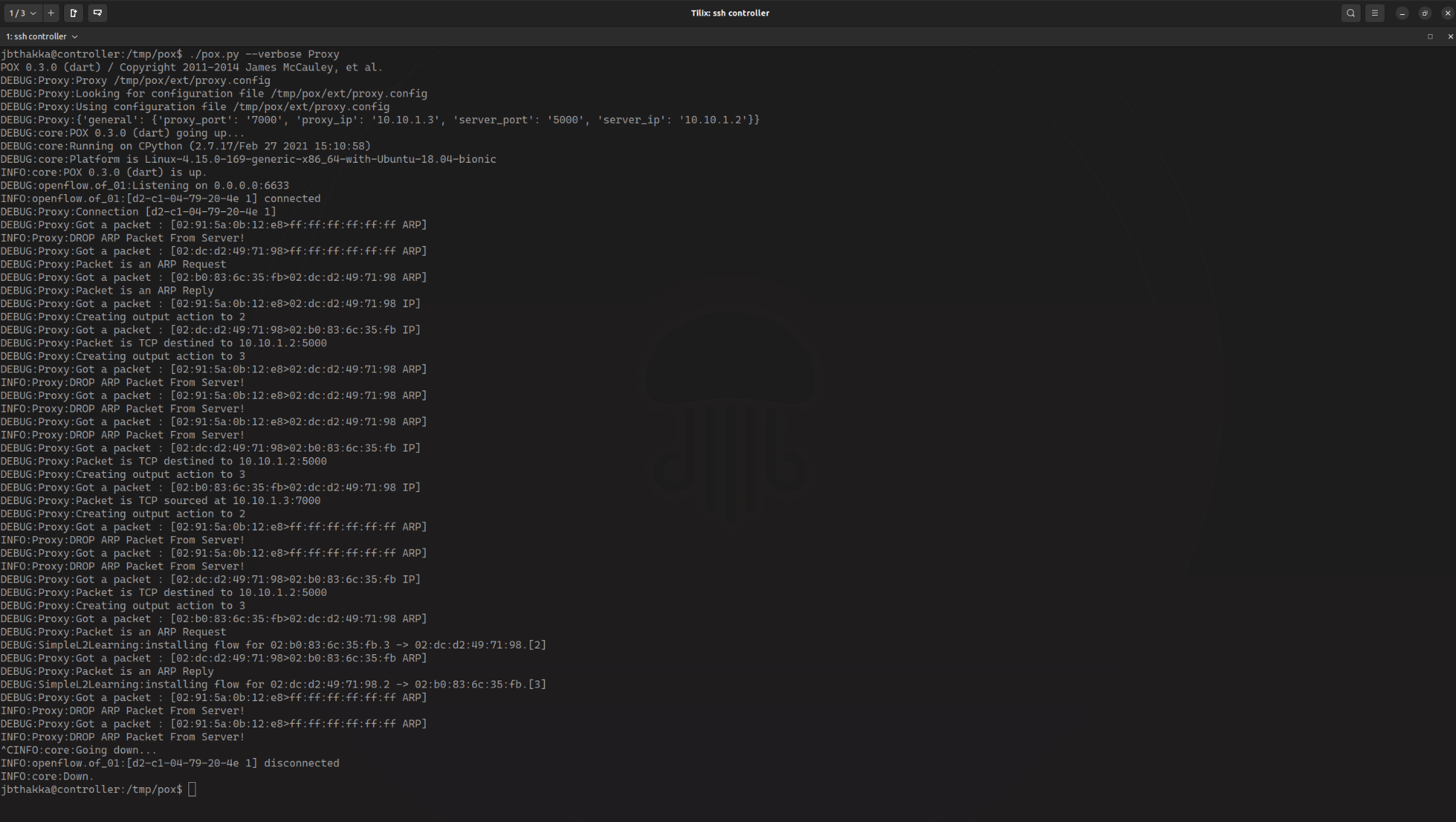
If we do not use the proxy, when host 1 messages host 2, the message will just appear on host 2 as shown below in the screenshot for the hosts. .

****Using l2\_forwarding

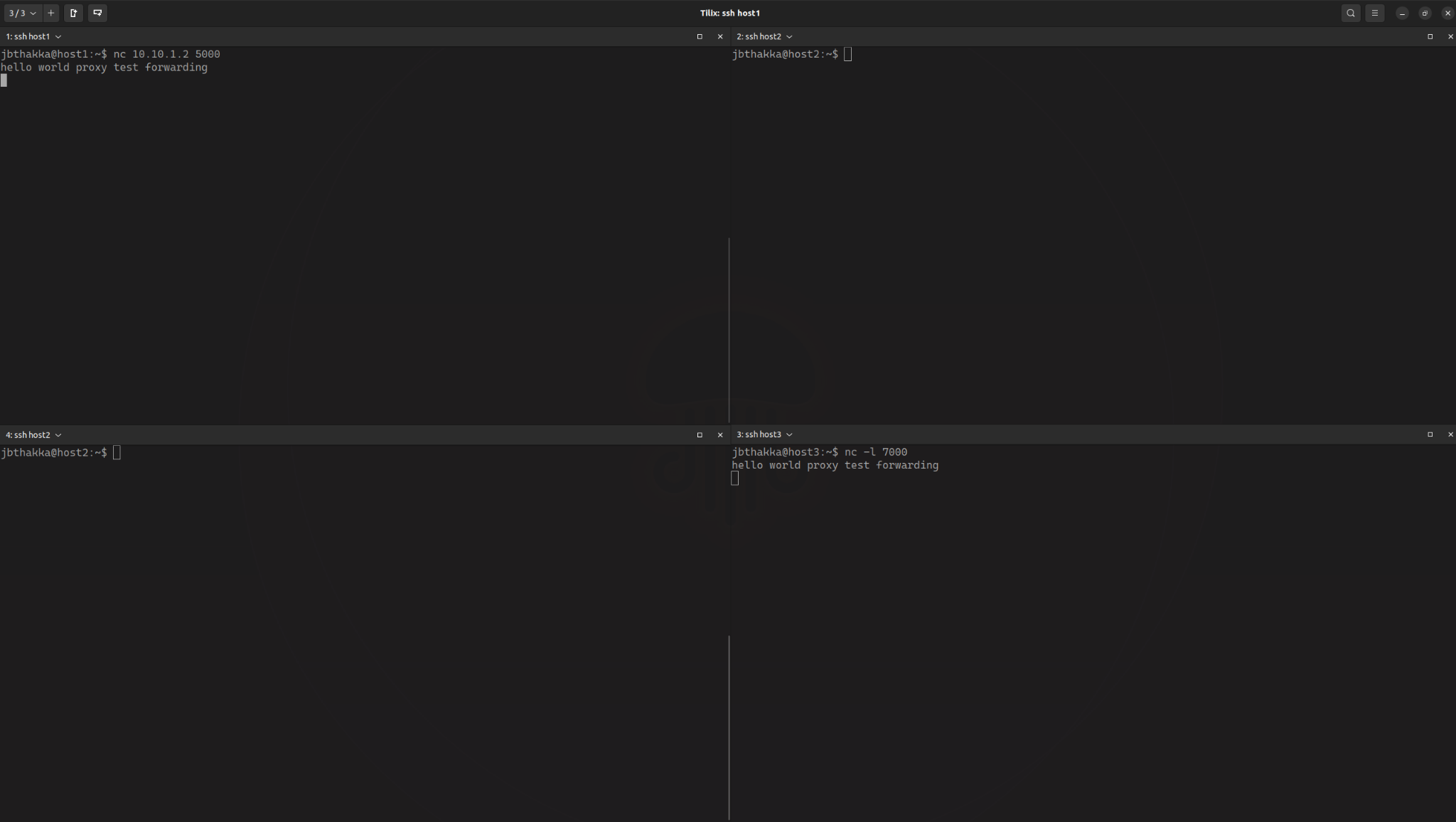
****

Netcat test

If we use proxy, when host 1 messages host 2, the message will appear on host 3 instead - because the pox controller is rewriting the packet destination in flight and routing the message to host3 as demonstrated by the screenshot.

****

Using Proxy

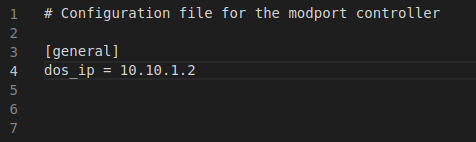
****

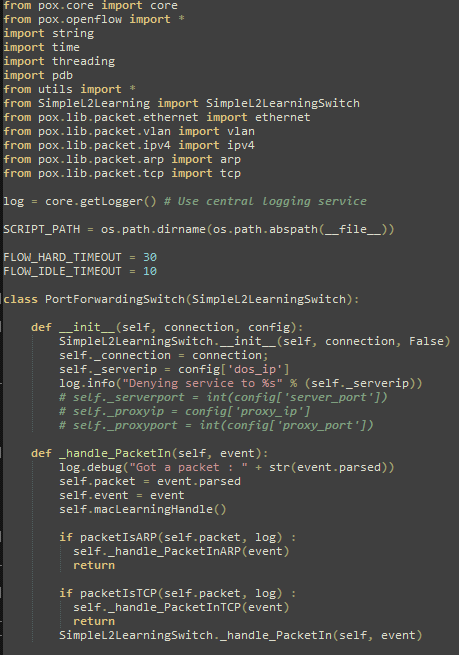
Netcat Test

**Exercise 1.7**

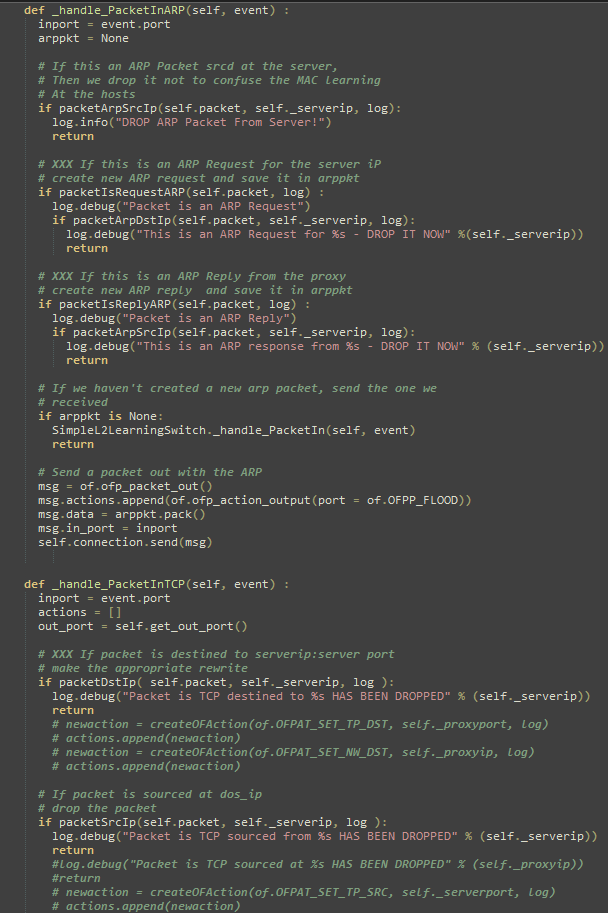
We choose to block traffic for host 2.

What we have done is modify the Proxy.py controller program to check if there is a ARP packet or TCP packet that is received from or intended for host 2 using packetDstIp and packetSrcIP functions inside the controller’s function for the TCP and the ARP requests - basically we do not install any flows when encountering the packets for host 2, simply drop them, and log the message on the controller screen - this effectively prevents any host from connecting to/from host 2. You can DoS any other IP by providing a config file named “dos\_config” with the following format:

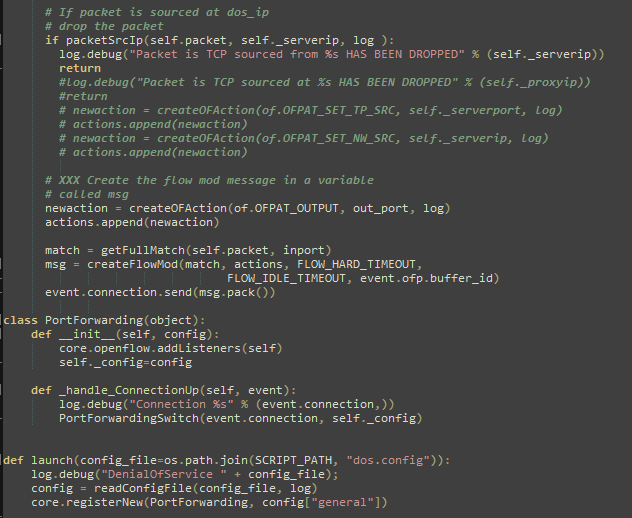
  
The source code is provided below:



Code Part 1

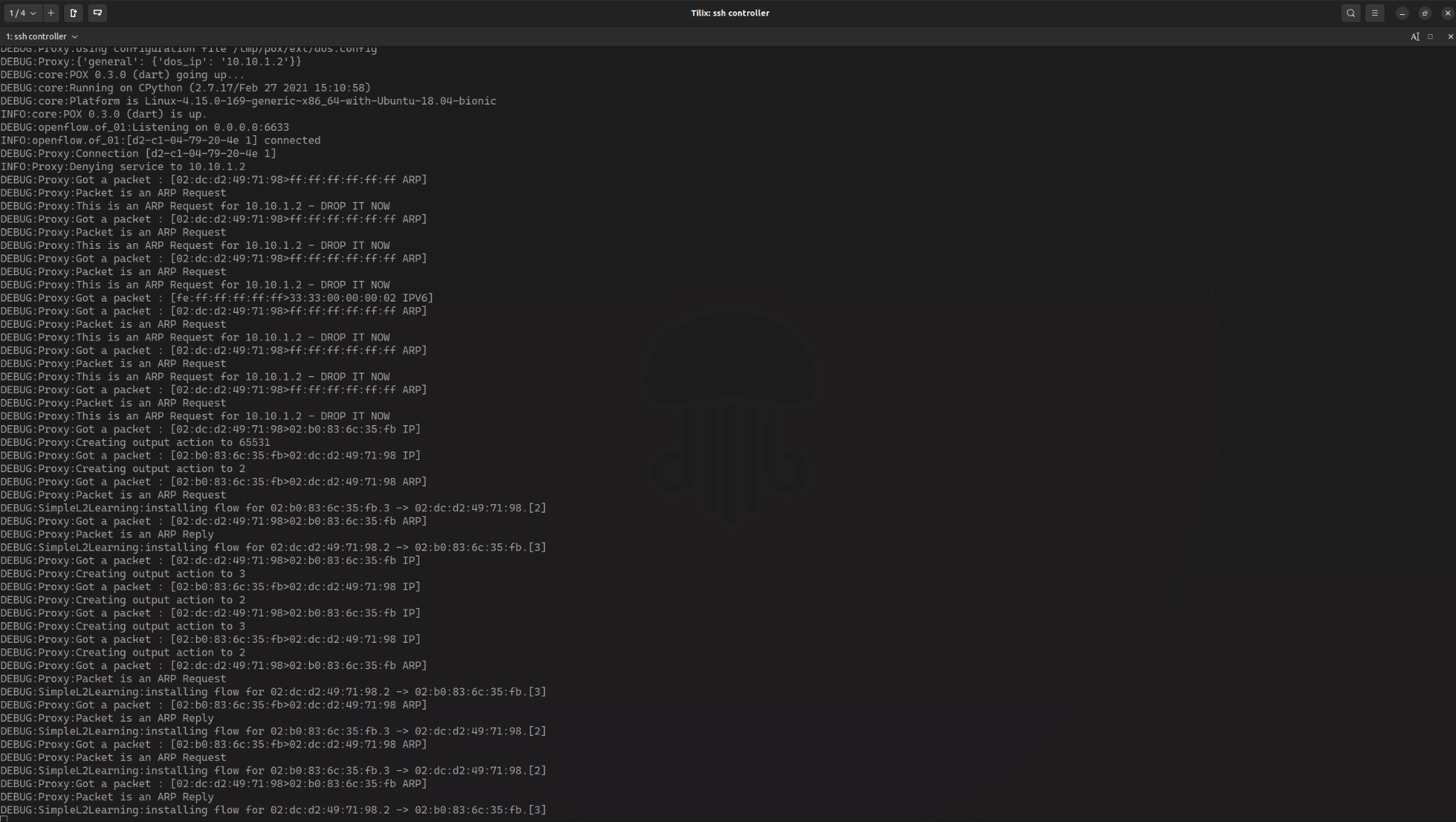


Code Part 2



Code Part 3

Demo - When host 1 tries to message host 3, the message goes through successfully.

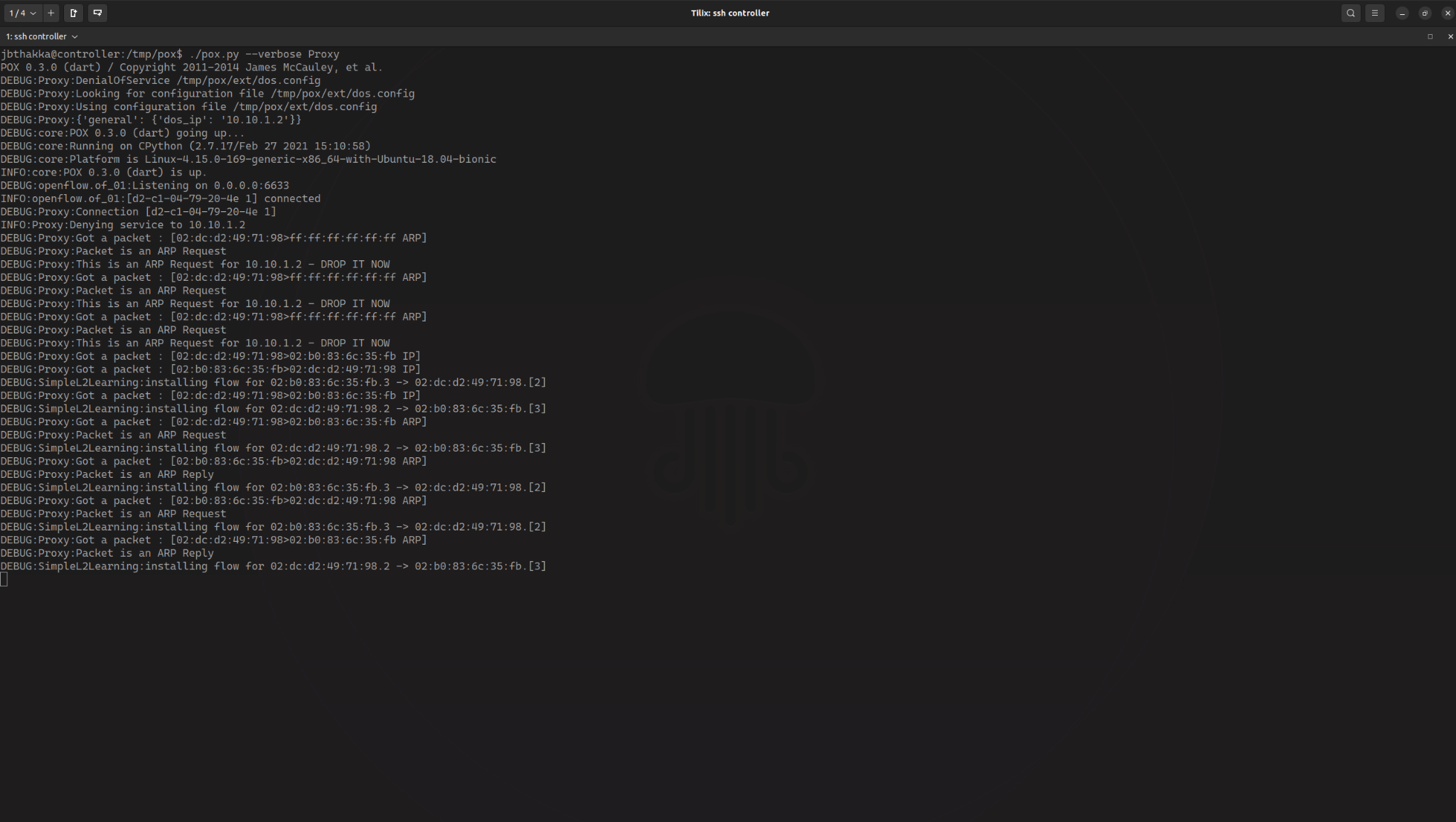


Host 1 message Host 3 (Controller View)

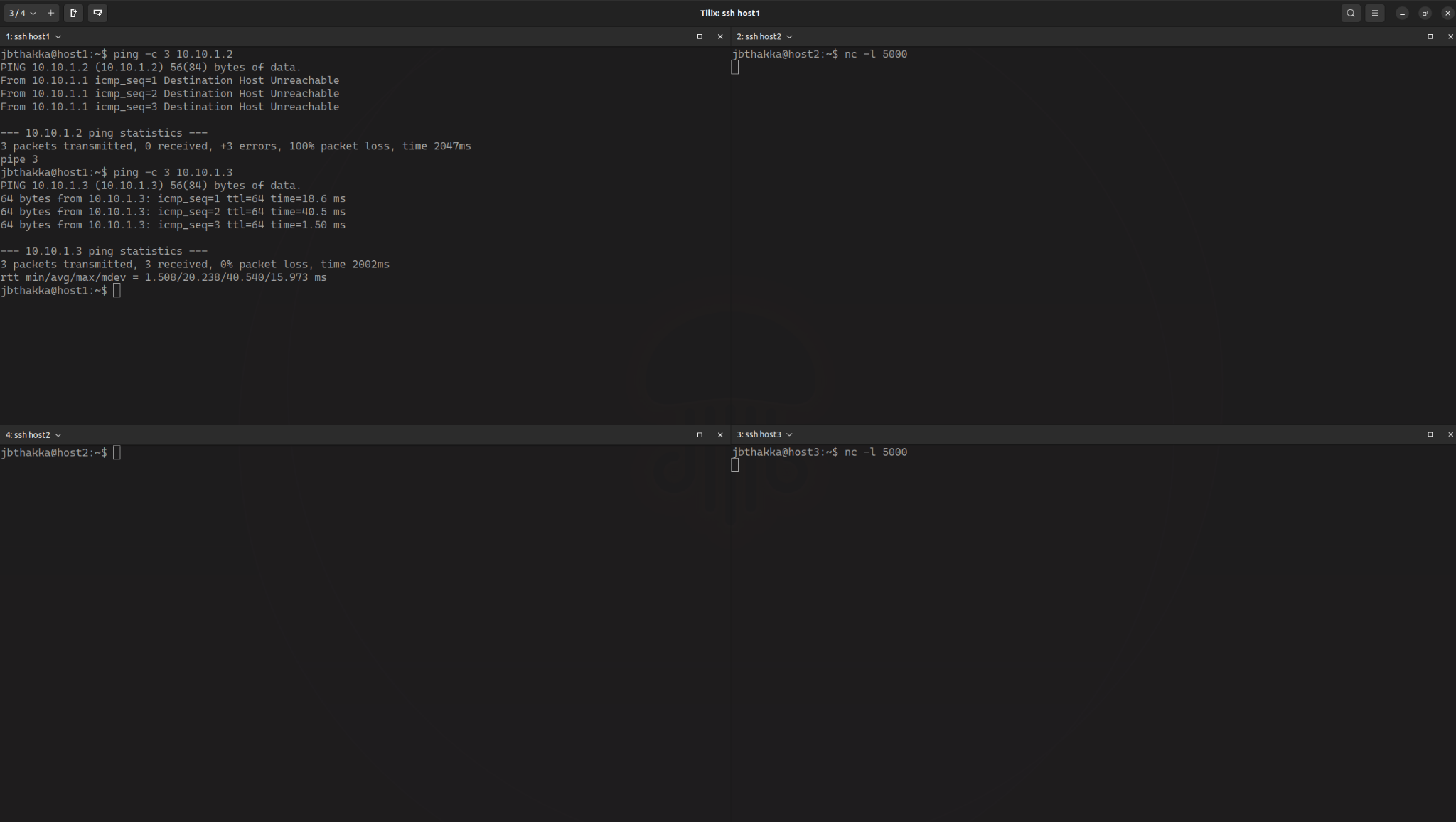


Host 1 message Host 3

When Host 1 tries to ping Host 2, all the packets will be dropped and hence Host 2 is unreachable from Host 1 as shown below.

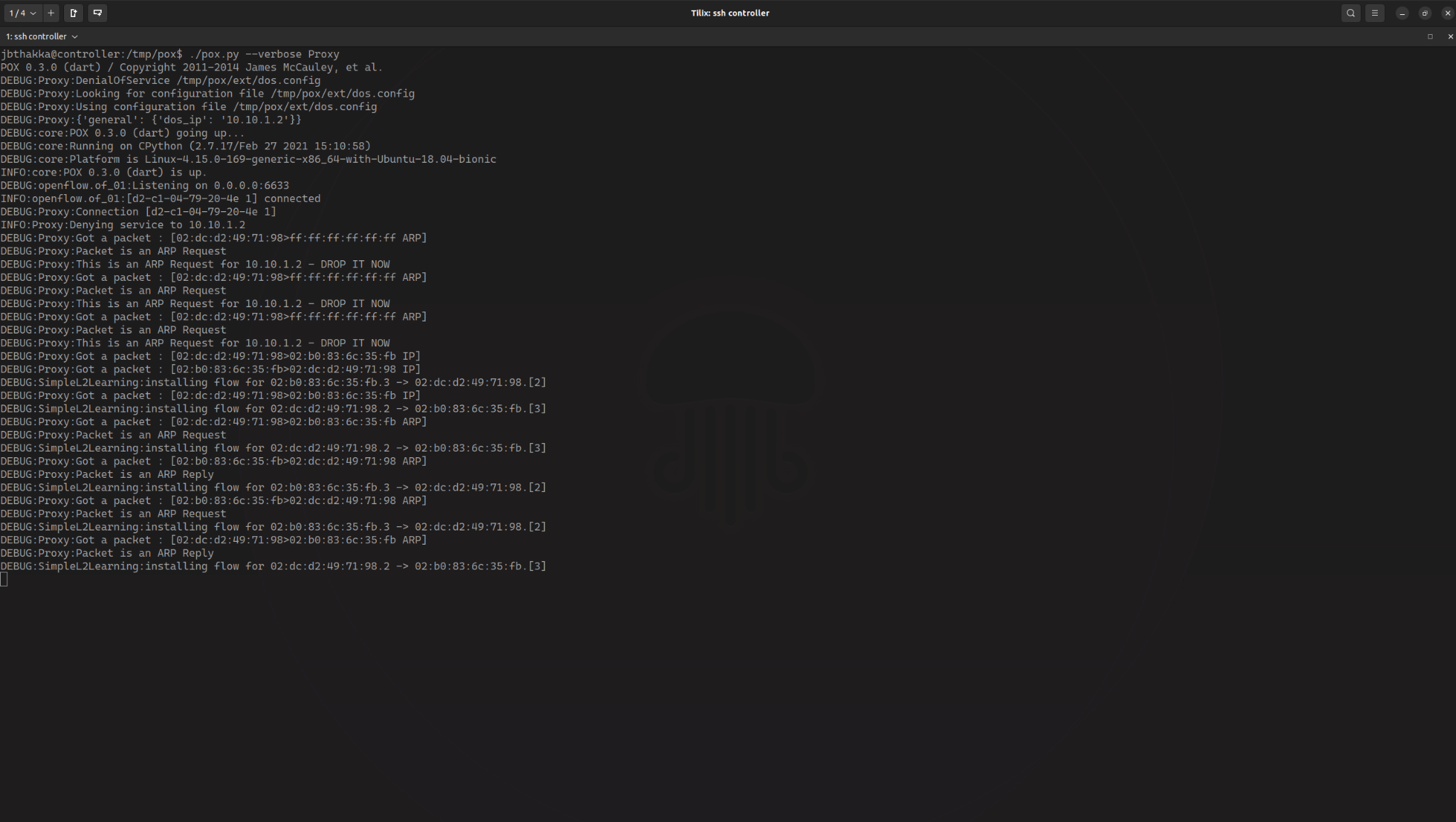


Host 1 tries to ping Host 2 (Fails) (Controller View)

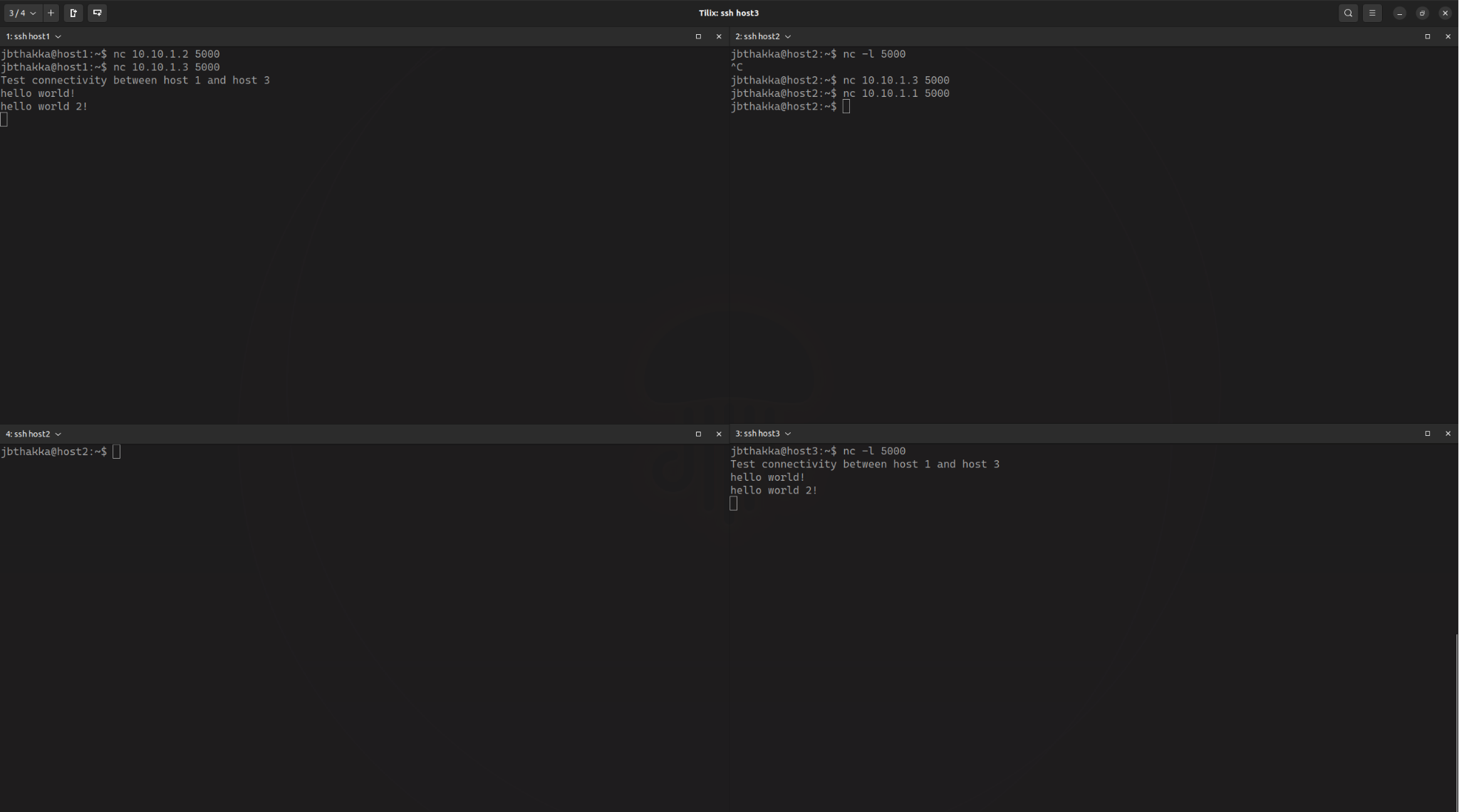


Host 1 tries to ping Host 2 (Fails)

When Host 2 tries to connect to Host 1 or 3, it wouldn’t be able to as its packet is all dropped.



Host 2 tries to connect to host 1 and 3 via Netcat using TCP (fails) (Controller View)



Host 2 tries to connect to host 1 and 3 via Netcat using TCP (fails)