**Homework 3 - Mininet & OpenFlow**

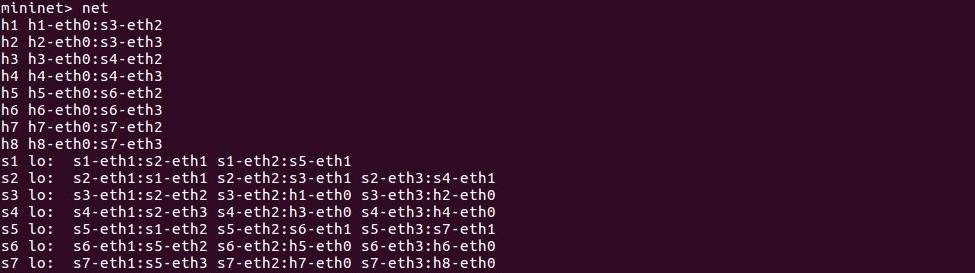
**Task 1: Defining custom topologies**

1. **What is the output of "nodes" and "net"**

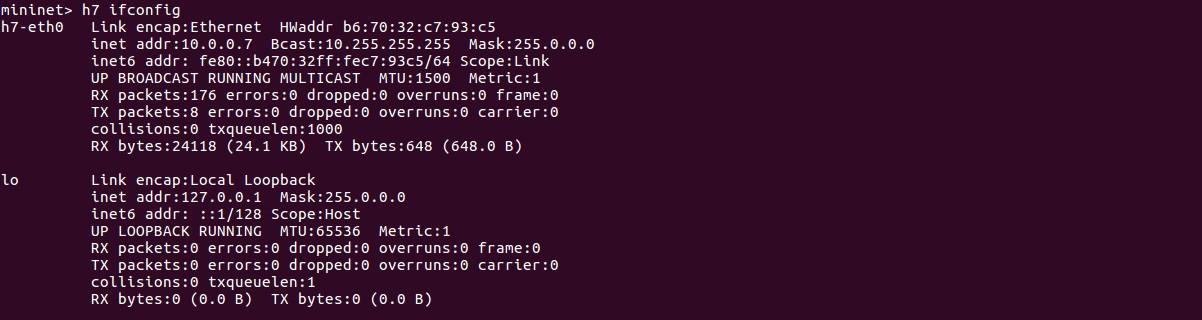
Output of "nodes" command:



Output of "net" command:



2. **What is the output of "h7 ifconfig"**

****

**Task 2: Analyze the "of\_tutorial" controller**

1. **Draw the function call graph of this controller. For example, once a packet comes to the controller, which function is the first to be called, which one is the second, and so forth?**

We first start the POX listener using the command: **./pox.py log.level –DEBUG misc.of\_tutorial**

The process is initiated with the 'start switch' command. This command invokes the '\_handle\_PacketIn()' method, which is designed to manage 'packet in' messages received from the switch. Next, '\_handle\_PacketIn()' calls upon the 'act\_like\_hub()' method. The role of 'act\_like\_hub()' is to dispatch the packets to all ports, excluding the port where the packet originated, thus replicating the function of a hub. Subsequently, the 'resend\_packet()' method is employed. This method is tasked with appending a packet to the message's data and executing the corresponding action. Finally, a command is relayed to the switch, instructing it to 'resend' the packet to a port that is specified by this command.

|  |
| --- |
| **Function call graph:**  start switch : \_handle\_PacketIn() -> act\_like\_hub() -> resend\_packet() -> send(msg)  \_handle\_PacketIn()  act\_like\_hub()  resend\_packet()  Packet comes in  Forwards message to the port |

2. **Have h1 ping h2, and h1 ping h8 for 100 times (e.g., h1 ping -c100 p2).**

**h1 ping -c100 h2:**





**h1 ping -c100 h8:**

****

****

2.a **How long does it take (on average) to ping for each case?**

|  |  |
| --- | --- |
| Average ping | |
| h1 ping h2 | 1.670 ms |
| h1 ping h8 | 5.486 ms |

2.b **What is the minimum and maximum ping you have observed?**

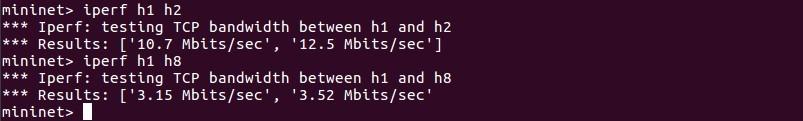
|  |  |
| --- | --- |
| Minimum ping | |
| h1 ping h2 | 1.121 ms |
| h1 ping h8 | 4.572 ms |
| Maximum ping | |
| h1 ping h2 | 3.612 ms |
| h1 ping h8 | 12.212 ms |

2.c **What is the difference, and why?**

The time required to ping is longer for h1 to h8 as compared to h1 to h2. The reason is that only one switch is present between h1 and h2 which is s3 whereas there are several hops between h1 and h8 i.e. s3, s2, s1, s5, s7.

3. **Run "iperf h1 h2" and "iperf h1 h8"**

Output for "iperf h1 h2" and "iperf h1 h8" commands:



3.a **What is "iperf" used for?**

iperf is a widely-utilized, open-source utility that aids system administrators in gauging network performance and bandwidth capabilities. This tool operates between two hosts that establish the endpoints of the network connection under test. It is employed to assess the data transfer rate, or throughput, between these two nodes across the network link.

3.b **What is the throughput for each case?**

|  |
| --- |
| mininet> iperf h1 h2  \*\*\* Iperf: testing TCP bandwidth between h1 and h2  \*\*\* Results: ['10.7 Mbits/sec', '12.5 Mbits/sec']  mininet> iperf h1 h8  \*\*\* Iperf: testing TCP bandwidth between h1 and h8  \*\*\* Results: ['3.15 Mbits/sec', '3.52 Mbits/sec' |

3.c **What is the difference, and explain the reasons for the difference**.   
Data moves more quickly from h1 to h2 than from h1 to h8 due to fewer network interruptions and shorter travel time. The direct route between h1 and h2 allows for a larger amount of data to be sent in less time. On the other hand, the connection from h1 to h8 has more points it must pass through, which slows down the transfer, allowing less data to be sent over the same period.

4. **Which of the switches observe traffic? Please describe your way for observing such traffic on switches (e.g., adding some functions in the “of\_tutorial” controller).**

Inserting the line **log.info("Switch observing traffic: %s" % (self.connection))** into line number 107 in the "of\_tutorial" controller script allows us to monitor the network traffic through the console output. This information reveals that every switch is tracking the traffic, especially noticeable when the network is busy with a flood of packets. The function **\_handle\_PacketIn** is set up to respond to events, meaning it gets activated each time a new packet arrives at the switch.

**Task 3: MAC Learning Controller**

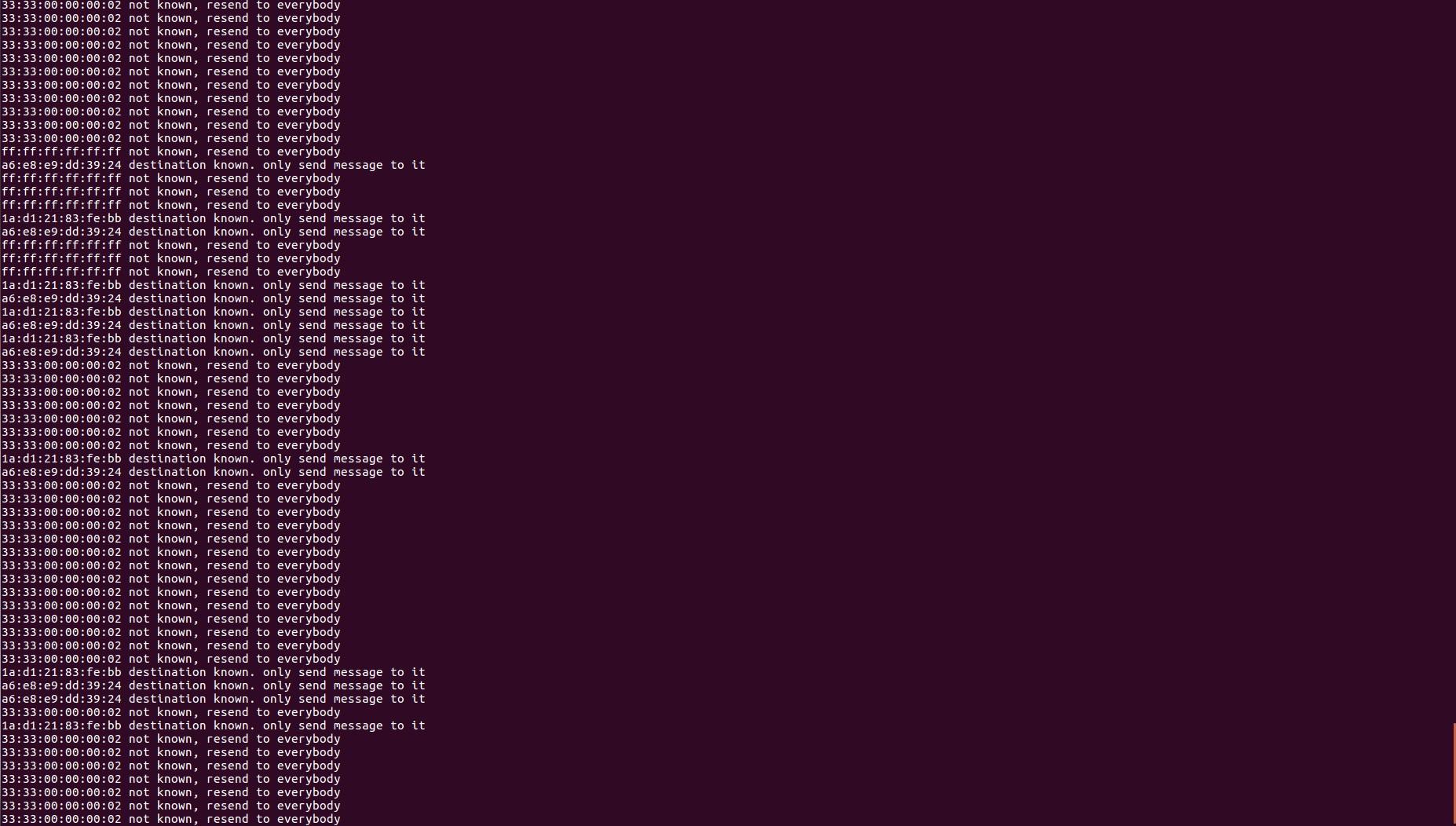
1. **Describe how the above code works, such as how the "MAC to Port" map is established. You could use a ‘ping’ example to describe the establishment process (e.g., h1 ping h2).**

The **act\_like\_switch** function is designed to create an association between MAC addresses and their corresponding ports. Once a MAC address is identified as the recipient of a packet, the controller can simplify the process by assigning that MAC address to a specific port. This not only streamlines the controller’s operation when routing packets to familiar addresses by sending them directly to the associated port but also enhances the network's efficiency. In instances where the recipient’s address is unknown, the function resorts to broadcasting the packet to all ports. Utilizing the MAC Learning Controller also contributes to better network performance, as it reduces the need for such packet floods, which in turn can positively affect ping response times and data transmission rates. The effectiveness of this mapping and performance enhancement becomes evident when observing the results from a single packet ping from h1 to h2:Top of Form

Bottom of Form

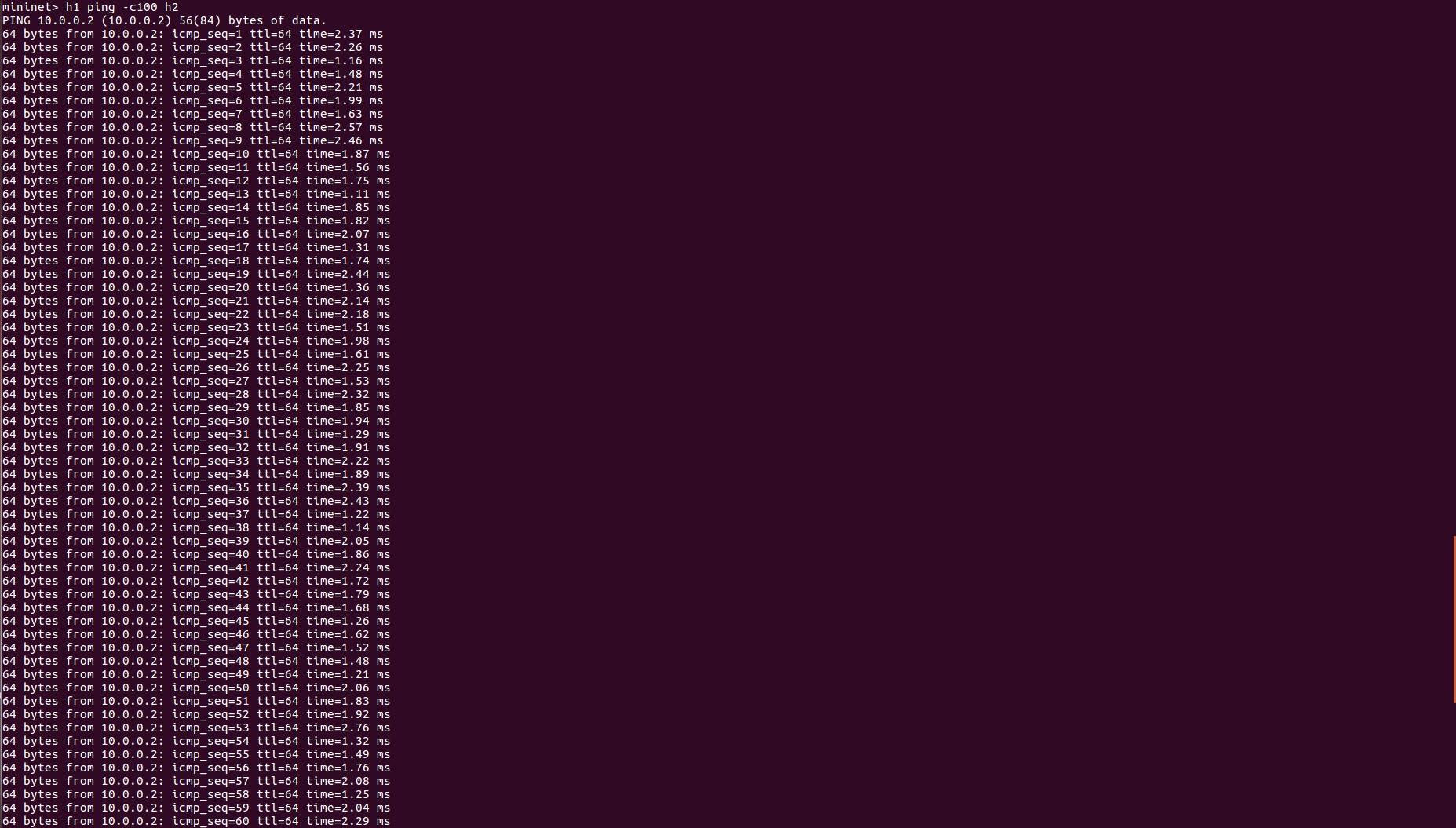
****

Continued..

****

2. **(Comment out all prints before doing this experiment) Have h1 ping h2, and h1 ping h8 for 100 times (e.g., h1 ping -c100 p2).**

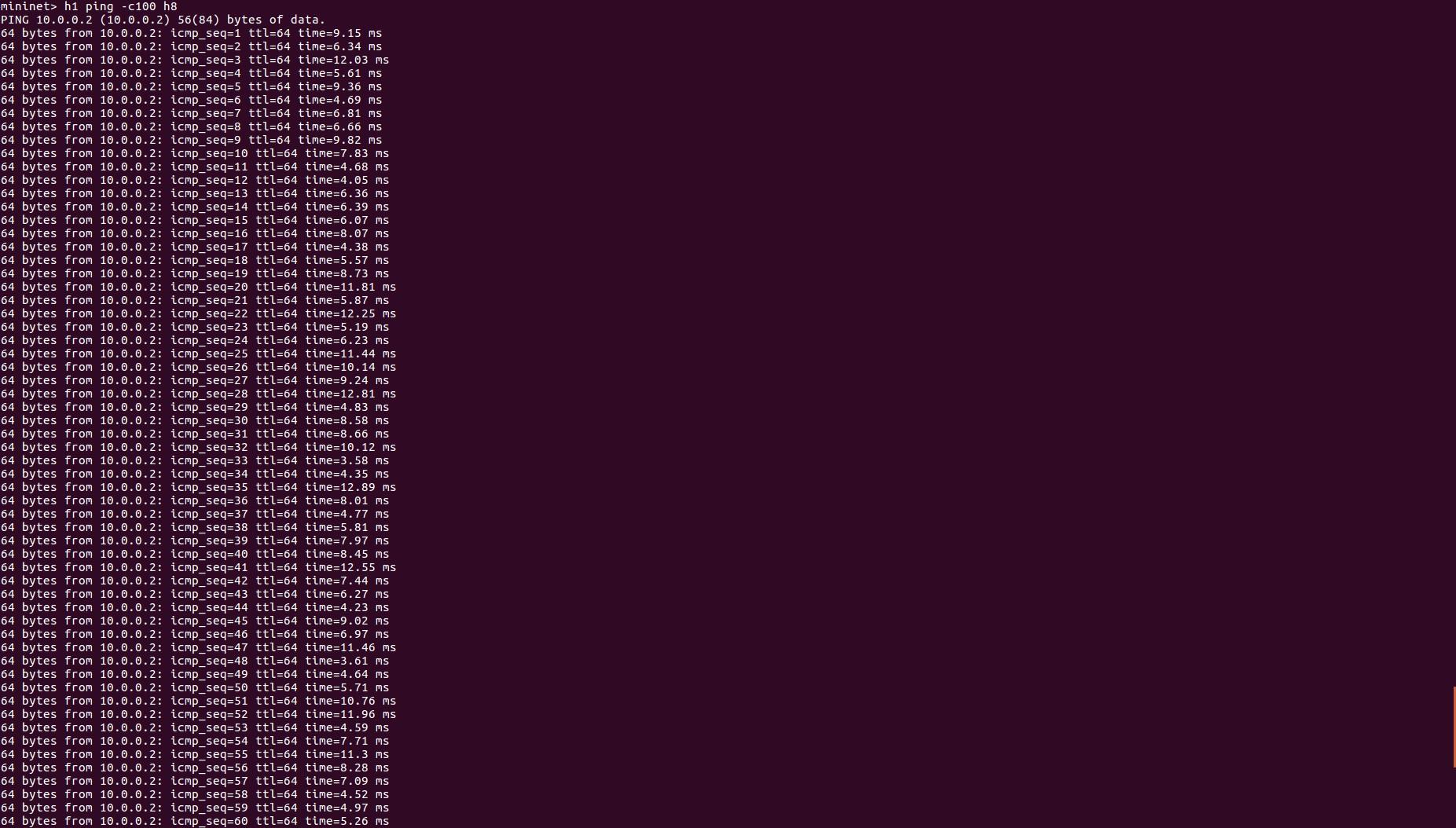
**h1 ping -c100 h2:**



Continued..



h1 ping -c100 h8:





2.a **How long did it take (on average) to ping for each case?**

|  |  |
| --- | --- |
| Average ping | |
| h1 ping h2 | 1.814 ms |
| h1 ping h8 | 4.282 ms |

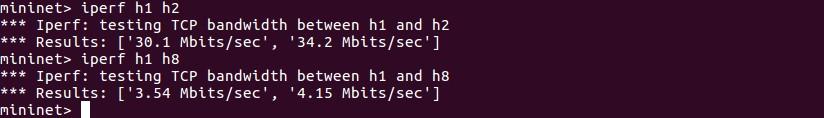
2.b **What is the minimum and maximum ping you have observed?**

|  |  |
| --- | --- |
| Minimum ping | |
| h1 ping h2 | 1.121 ms |
| h1 ping h8 | 3.542 ms |
| Maximum ping | |
| h1 ping h2 | 2.562 is ms |
| h1 ping h8 | 12.901 is ms |

2.c Any difference from Task 2 and why do you think there is a change if there is?

In task 3, we see a marginal improvement in the time it takes for h1 to ping h2 compared to task 2. The reduction in time is relatively small. However, for h1 to h8, the decrease in ping time is more substantial due to the longer path the packets travel through multiple switches. Task 3 exhibits quicker ping responses because it initially floods only a few packets. Once the switches have identified the destination MAC address and recorded it in the "mac\_to\_port" table, they will direct the packets solely to the mapped port. As a result, later pings proceed much quicker with less network traffic congestion.

3**. Run "iperf h1 h2" and "iperf h1 h8".**



3.a **What is the throughput for each case?**

|  |
| --- |
| mininet> iperf h1 h2  \*\*\* Iperf: testing TCP bandwidth between h1 and h2  \*\*\* Results: ['30.1 Mbits/sec', '34.2 Mbits/sec']  mininet> iperf h1 h8  \*\*\* Iperf: testing TCP bandwidth between h1 and h8  \*\*\* Results: ['3.54 Mbits/sec', '4.15 Mbits/sec' |

3.b **What is the difference from Task 2 and why do you think there is a change if there is?**

For task 3, the data throughput is improved compared to task 2 for both scenarios tested. The enhanced performance is attributed to reduced network congestion since task 3 eliminates the need to flood packets after the 'mac\_to\_port' map has learned and recorded all the ports, resulting in less strain on the switches. Specifically, for h1 to h2, the throughput in tasks 1 and 2 shows a roughly threefold increase on average, which can be credited to the more efficiently pre-determined and learned routes as a result of modifications in the controller's behavior. Although the improvement in throughput from h1 to h8 is not as pronounced, there is a slight advancement due to the reduction in the number of hops and packet loss.

**Git Repository Information:**

|  |  |
| --- | --- |
| Link to repository | https://github.com/ASBhullar/cloud-computing |
| Commit ID | 08156acb488fd2b20addea0d665084cd9ee60769 |