Assignment on Mininet/OpenFlow and Routing Algorithm

Module: *Telecommunications Systems and Quality of Services (EE4029)*

Student: Akhmadjon Rajabov

Student ID: 190200705

Professor: Xiaohong Peng

Academic year: 2019-2020

Content

1. Introduction………………………………………………………………….3
2. Methodology………………………………………………………………...3
3. Results and Discussions……………………………………………………..4
   1. Task One……………………………………………………………...4
   2. Task Two……………………………………………………………..8
   3. Task Three…………………………………………………………...12
4. Conclusion………………………………………………………………….17
5. Bibliography………………………………………………………………..18
6. Introduction

This laboratory work is based on virtual networks on a PC. Which uses the power of modern PCs to model and evaluate various systems without the need for real system implementation. This allows us to easily configure a virtual network and test its various functions. Furthermore, it helps to understand the behavior of various components, network parameters and how they are related to each other. Two tasks are based on Mininet. Mininet is a network emulator that creates a network of virtual hosts, switches, controllers and communication channels. MININET hosts are running Linux, and the switches support OpenFlow. OpenFlow is a communications protocol that provides access to forwarding from a switch or router to a controller. This protocol can run a software-defined network SDN.

In the first task, we created a network in a Mininet and analyzed it. We also became familiar with the basic commands for working with the Minnet. In the second task, we analyzed network packets and modified the OpenFlow switch tables. In the last task, we implemented a routing algorithm in a virtual network by developing software.

1. Methodology

*Resources used*

1. University PC and my private laptop
2. Ubuntu 14.04.4
3. Mininet 2.2.2-170321
4. Putty and Xming for Mininet and OpenFlow
5. Laboratory handouts for 1 and 2 tasks

3. Results and Discussions

**3.1. Task One:**

1. Create a Mininet network with an Open vSwitch connecting to 10 host nodes and the OpenFlow/Stanford reference controller.

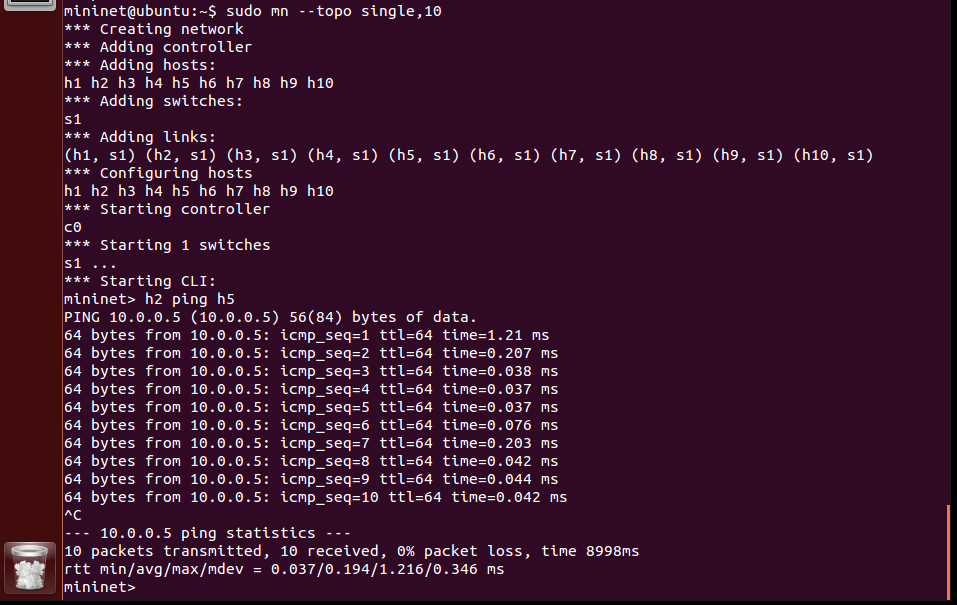


Figure 1.1. Creating a Mininet network with an Open vSwitch

In figure 1.1 we can see the command “sudo mn –topo single,10”. Using this command we created our network which consist of 10 hosts. Here “sudo” means that we run our command with root privilege, “mn” runs Mininet, “—topo single,10” signify type of topology single switch which is connected to all 10 hosts.

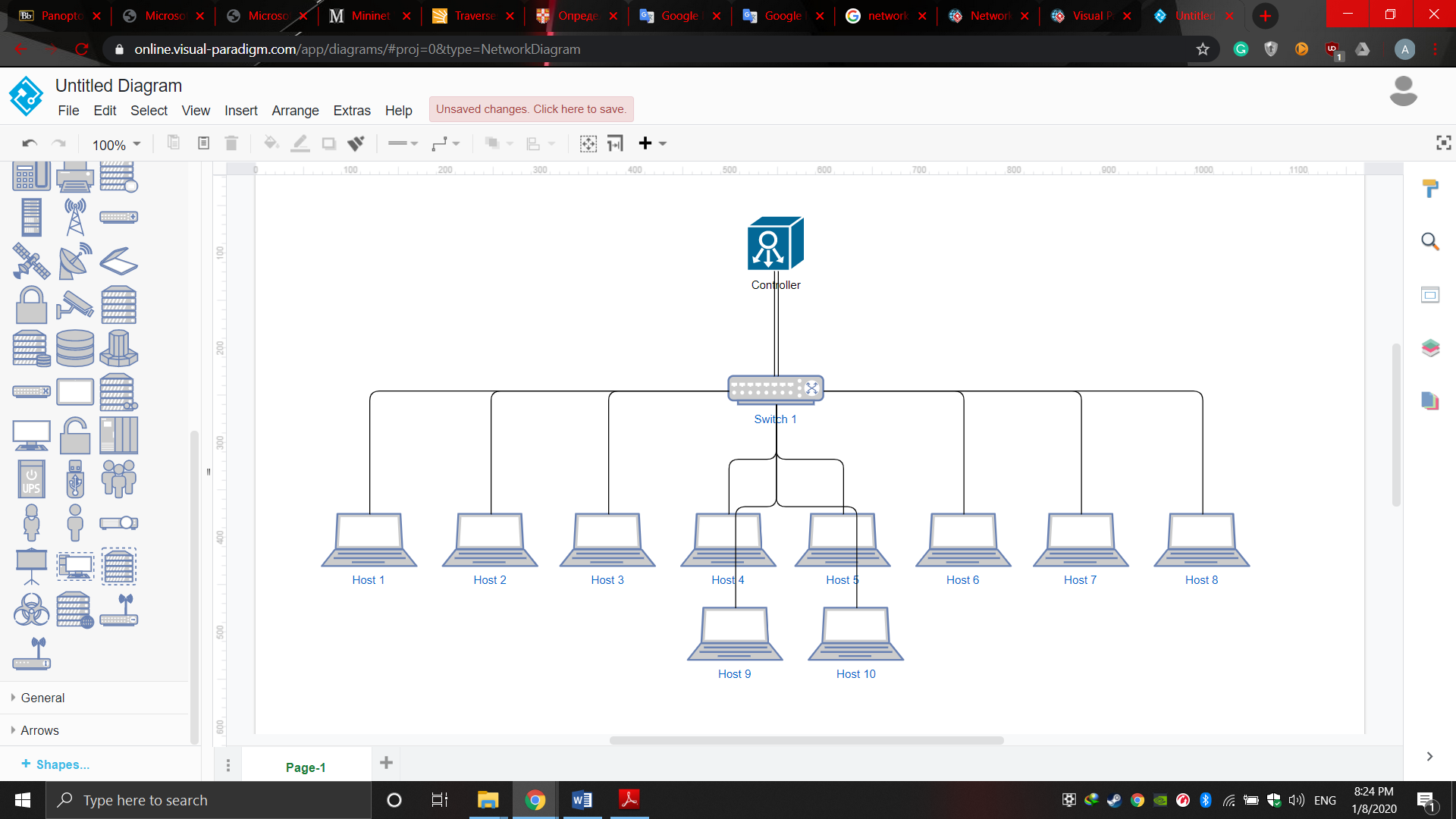


Figure 1.2. Network Diagram

Figure 1.2 illustrates our topology. This network was deployed after the first command.

1. Record the topology related information of the network created by a screenshot, and show the hosts, switches and the controller and their connections by sketching a network diagram.

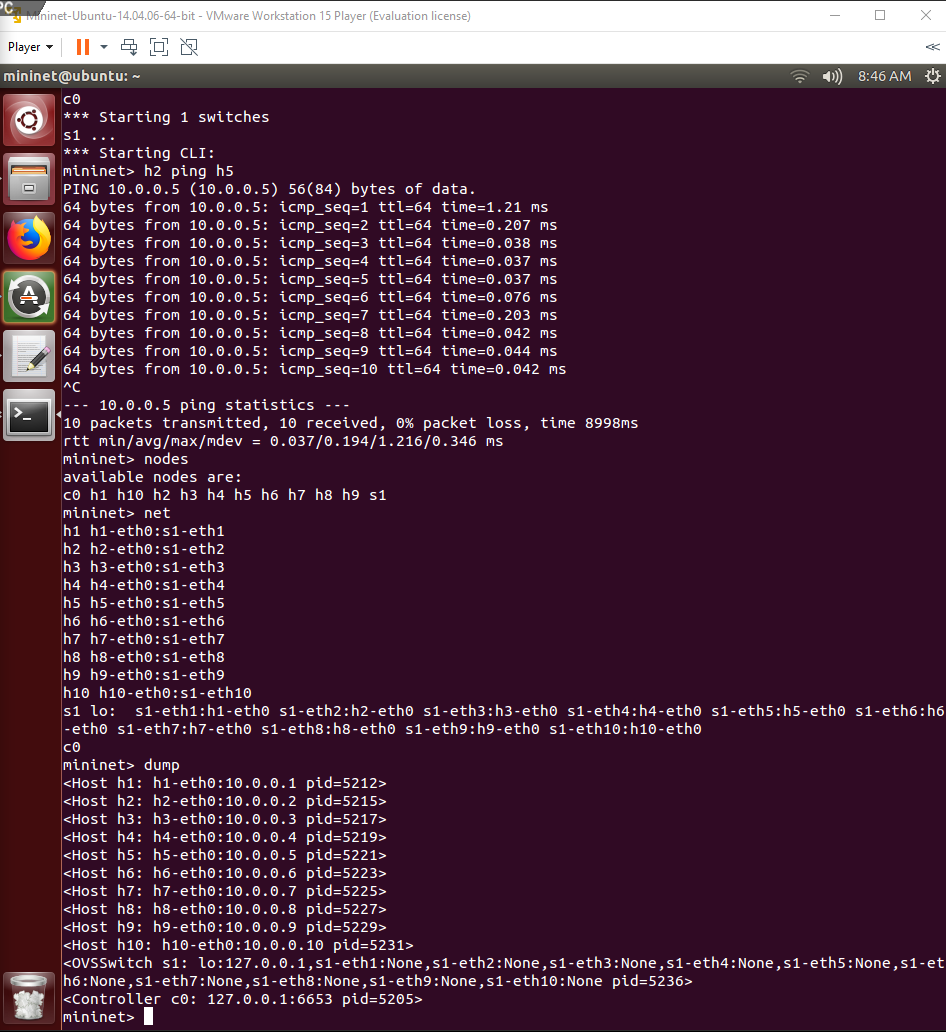


Figure 1.3. Topology related information

In the 1.3. picture we used commands such “nodes” – shows all devices in network, “net” – shows all connections between devices in network, “dump” – shows connected ports and IP addresses of devices.

1. Test the connectivity of host nodes h2 and h5. Record the measured transmission quality in terms of the packet loss rate and overall latency by a screenshot.

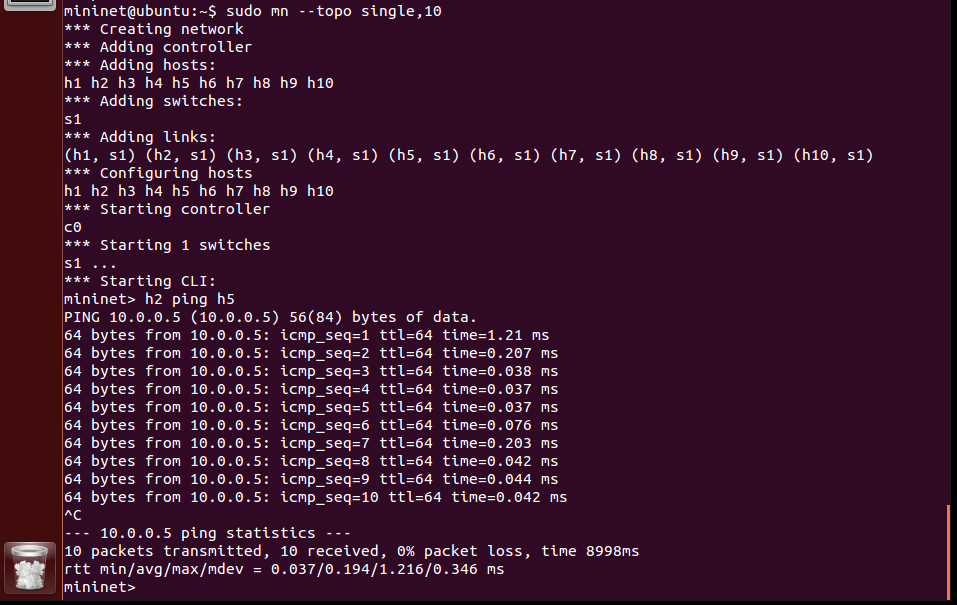


Figure 1.4. Checking the connection of nodes of node h2 and h5

In figure 1.4. represented test connection between h2 and h5 using command “h2 ping h5”. The result of ping was successful and packet loss 0% and average latency was 0.194 ms. However, at the beginning the latency was much higher because of switch doesn’t send package at directly, switch send request to controller require instructions about what to do next with package, After switch gets instructions switch will add new flow for this package and start to follow it. That’s why next packages send faster than first one.

1. Repeat 2 steps for hosts h1 and h10. Discuss the measured results in comparison with those recorded in 2 steps.

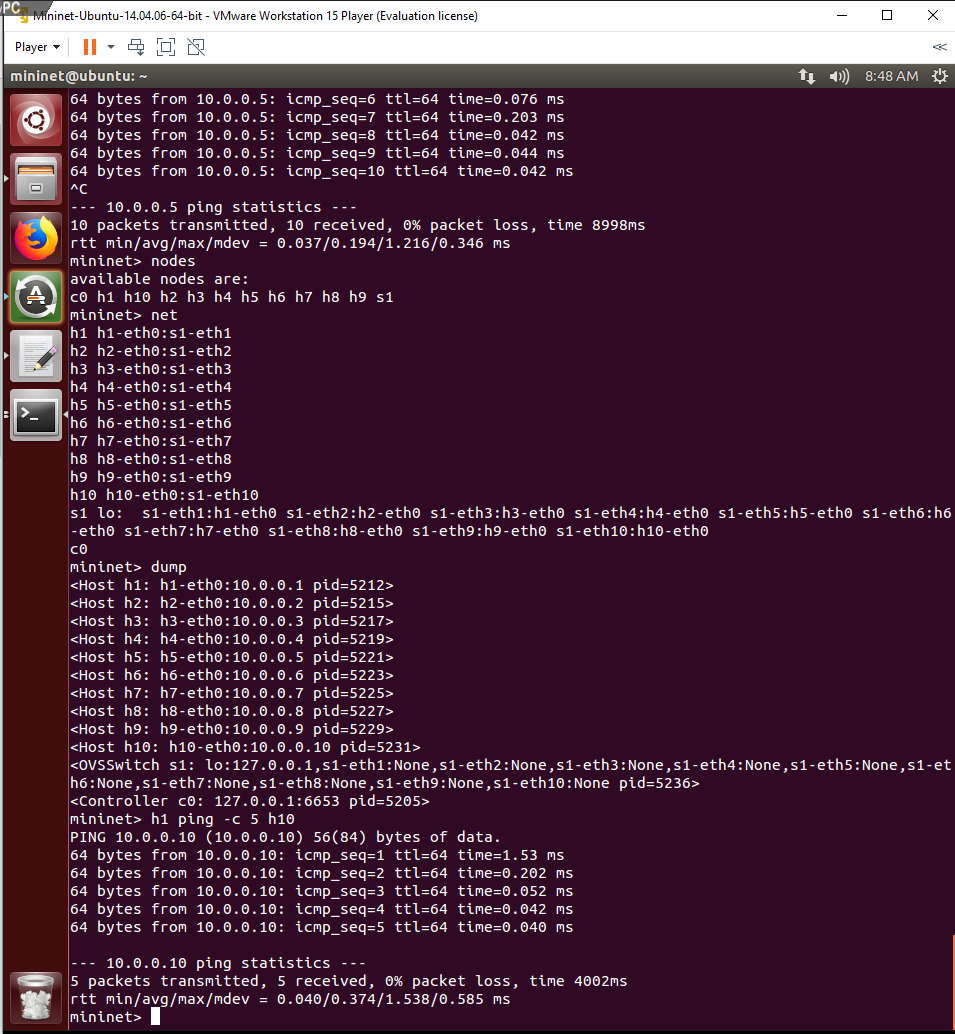


Figure 1.5. Checking the connection of nodes of node h1 and h10

In this figure we checked the connection between other h1 and h10 hosts very similar to previous action. As a result all packages received correctly without any mistakes with the same rules in latency.

1. Measure the bandwidth (data rate) between host nodes h2 and h5 and the measured bandwidth is recorded by a screenshot.

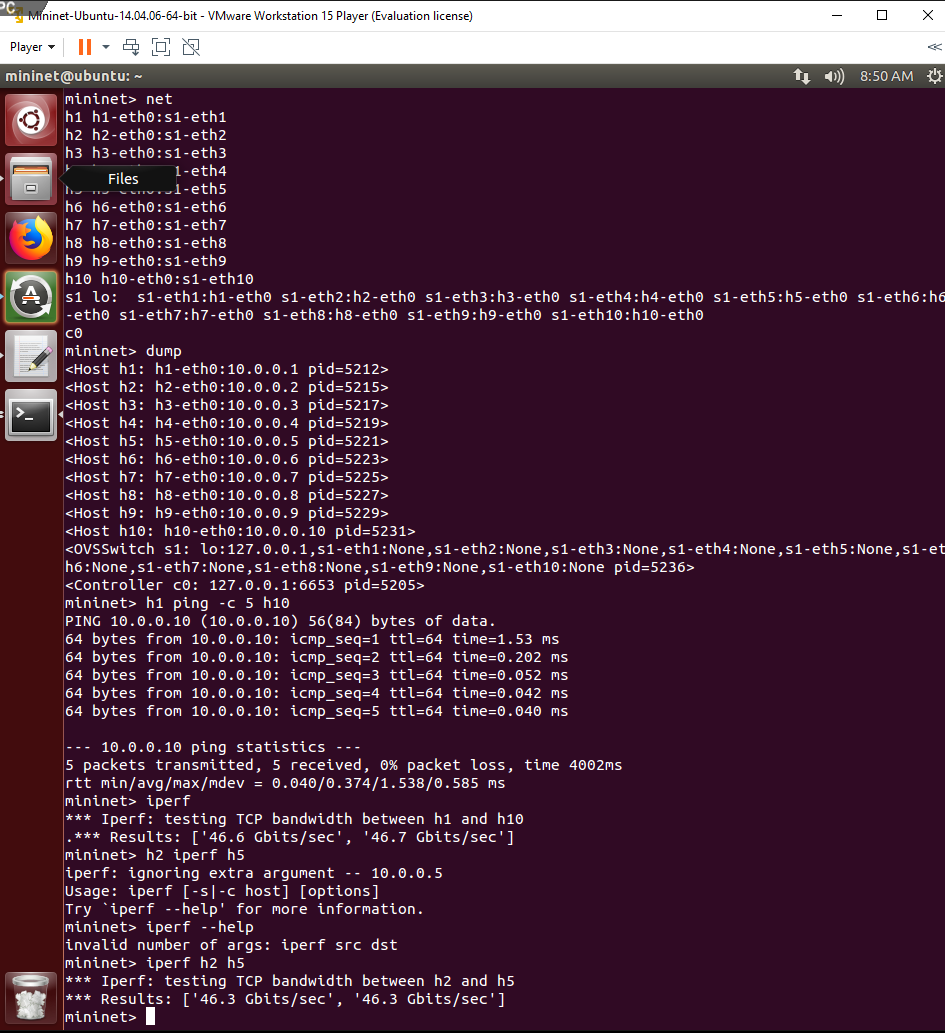


Figure 1.6. Measuring the bandwidth (data rate) between host nodes h2 and h5

We used command “iperf” – to receive information about bandwidth between h2 and h5 hosts which is 46.3 Gbits/s.

1. Change the link bandwidth and delay of the links between hosts and the switch to 30 Mbps and 15 ms, respectively. Repeat 2 steps and 4 steps

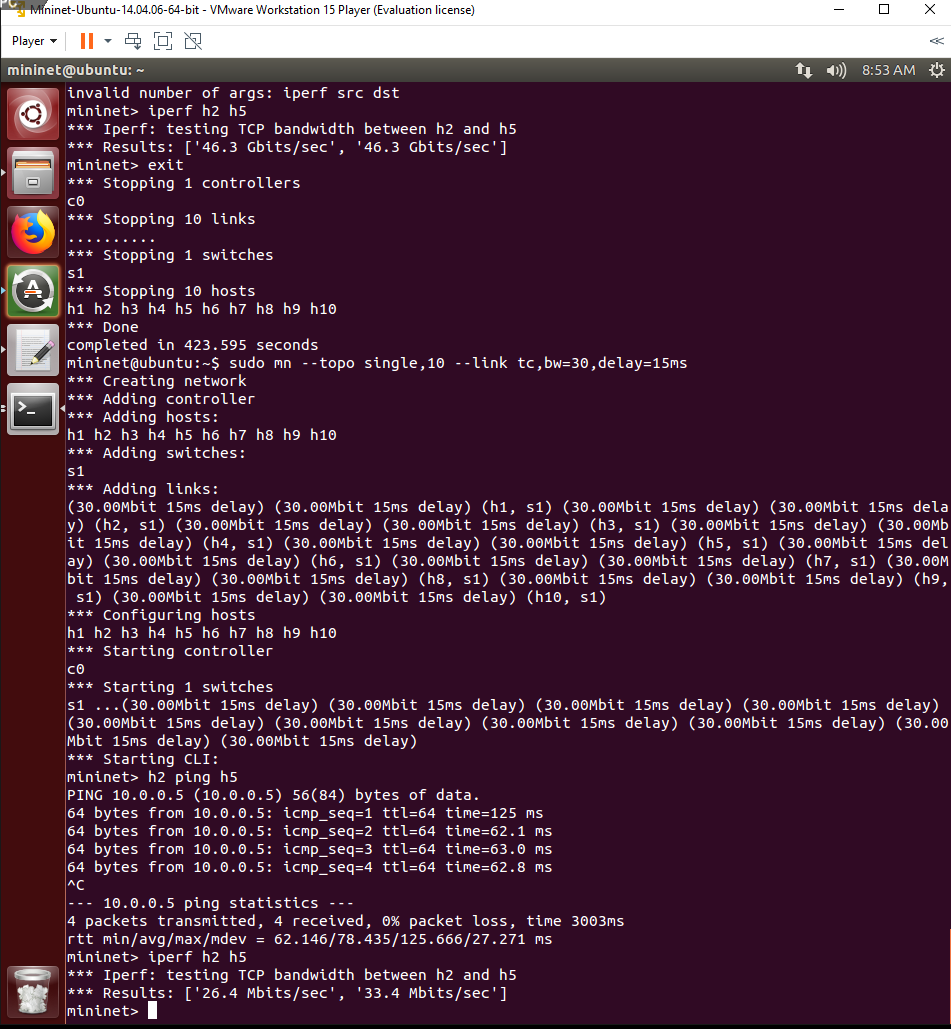


Figure 1.7. Changing the link bandwidth and delay of the links between hosts and the switch to 30 Mbps and 15 ms

In figure 1.7. we recreated our network changing default bandwidth and delay of the links between hosts using extra command “—link tc,bw=30,delay=15ms” in first line. After this we checked connection again between h2 and h5 host and we can see how delay increased. As shows figure latency grow up more than 15 ms due to the request traverses two links, one to the switch, and one to the destination and also reply traverses two links coming back. That’s why our latency is around 60 ms, one link going forwad and back 30 ms, we have two links therefore it is 60 ms. Aslo last command was checking bandwidth again and it was 33.4 Mbits/s.

**3.2. Task Two:**

1. Create a Mininet network with an Open vSwitch and 5 host nodes by the following command: $sudo mn --topo single,5 --mac --switch ovsk --controller remote

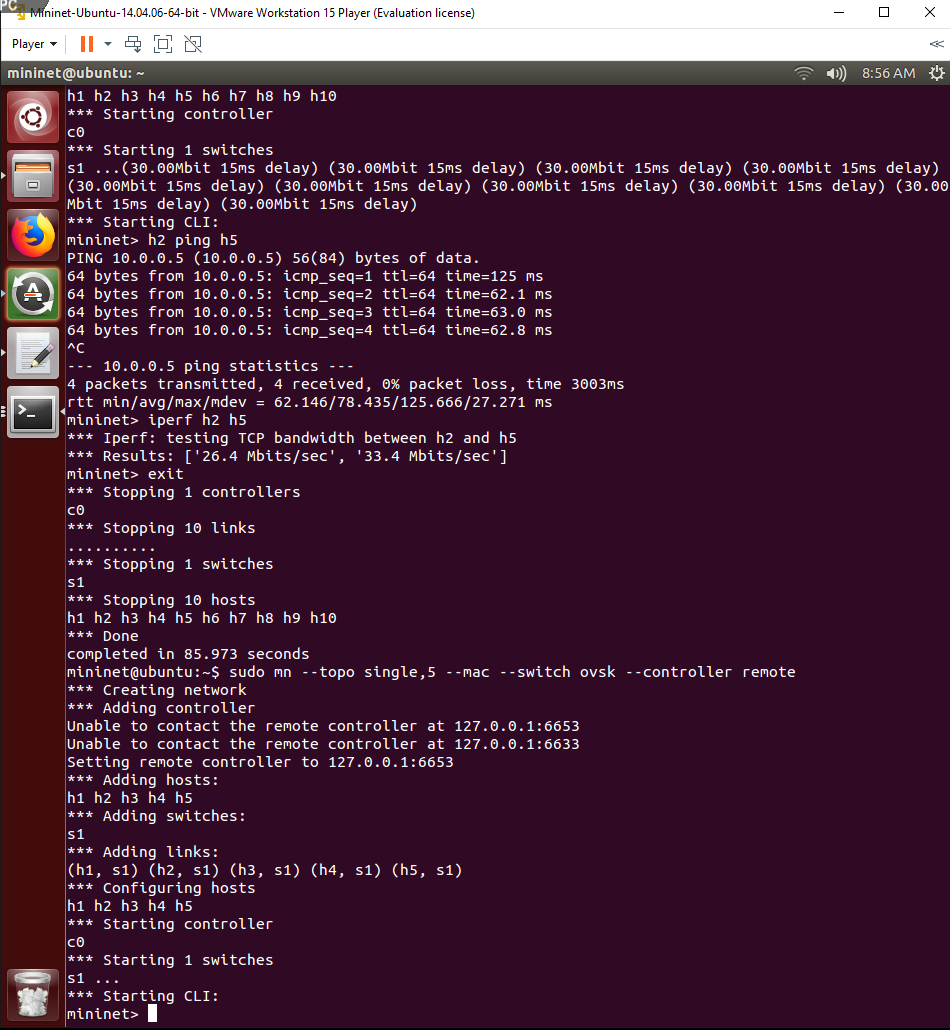


Figure 2.1. Creating a Mininet network with an Open vSwitch and 5 host nodes with remote controller mode

In figure 2.1 we established controller to remote mode, give us option to connect to controller which could be in the VM or outside VM. Moreover, it can be customized controller based on our requirements.

1. Use ovs-ofctl command to install necessary flow entries with the match field on in\_port to set up connectivity between host nodes h2 and h5. Record the flow entry information and the connectivity results by two separate screenshots.

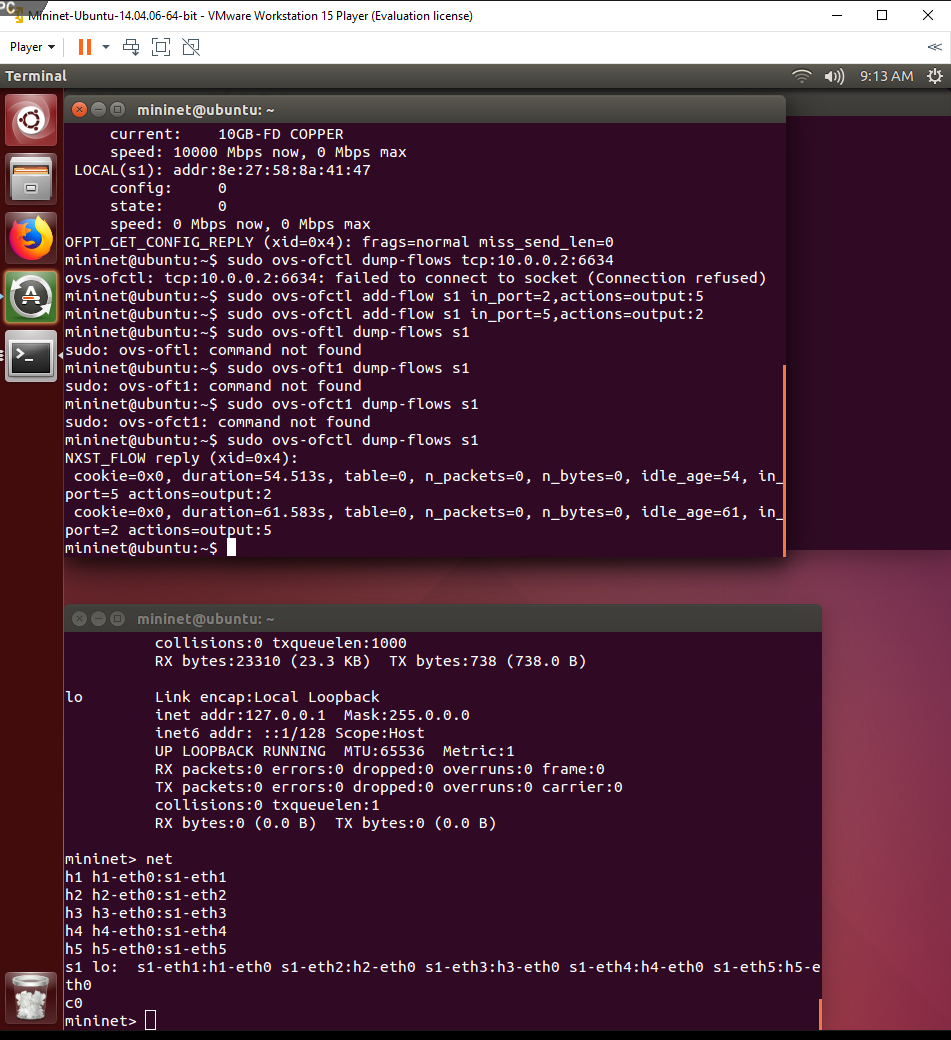


Figure 2.2. Creating flow entry between h2 and h5 hosts

For creating and adding flow entry to our network we used command “ovs-ofctl”. This command line tool for monitoring and administering OpenFlow switches. As figure showed us we added flow to switch 1 – “s1”, with showing ports from 2 to 5 and vis versa.

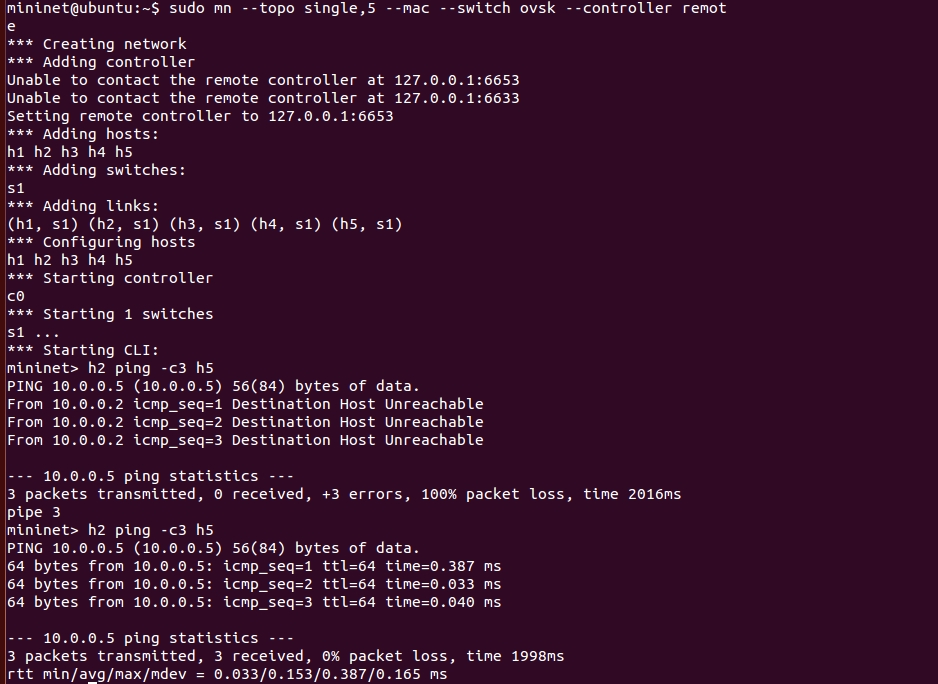


Figure 2.3. Checking connectivity between h2 and h5 hosts

First time when we tried to ping, h2 host couldn’t reach h5 host. However, after adding flows ping was successful.

1. Startup Wireshark dissector to capture network traffic (show this by a screenshot). Start the OpenFlow reference controller, which acts as a learning switch without installing any flow-entries, with the following command:

$ controller ptcp:

Show the types of the OpenFlow Protocol messages observed in the WireShark window by a screenshot.

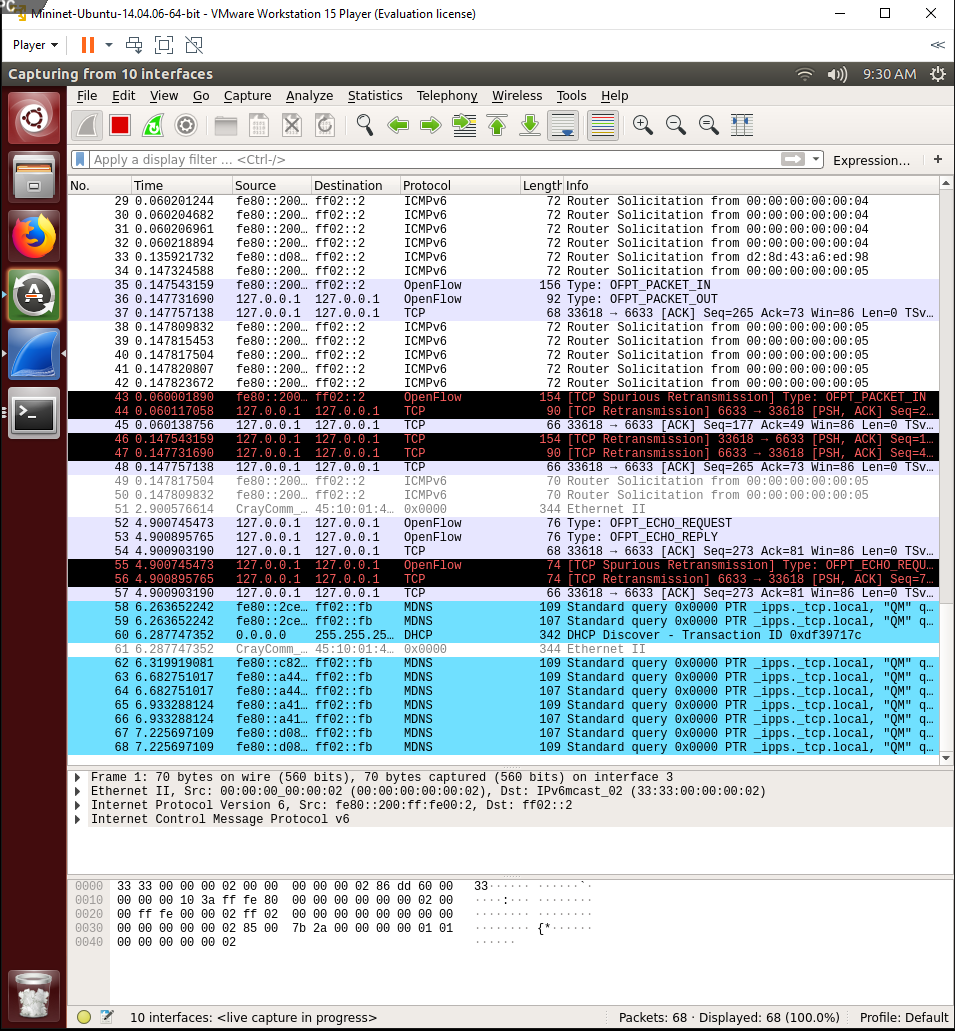


Figure 2.4. Wireshark capture

The types of Open Flow packets in this example are:

* Packet\_in: Packet resent from the switch to the controller, since there are no instructions in the switch flow table about how to handle it.
* Packet\_out: Packet sent from the controller to the switch. It is usually the same packet received in the packet\_in, but it adds the instructions to the switch about what to do with it.
* Flow\_mod: Instructions from the controller to the switch to modify the switch flow table.

1. It should be noted, a host node can only ping at most one host node with flow entries matching only the in\_port field. Now stop the OpenFlow reference controller which is started in 2). Manually install flow entries with ovs-ofctl commands and consider necessary changes on the matching fields and actions for the flow entries (e.g. you: may consider matching fields on MAC address or IP address) to enable simultaneous connectivity between all the nodes. Show the simultaneous connectivity by successfully running the pingall command and taking a screenshot.

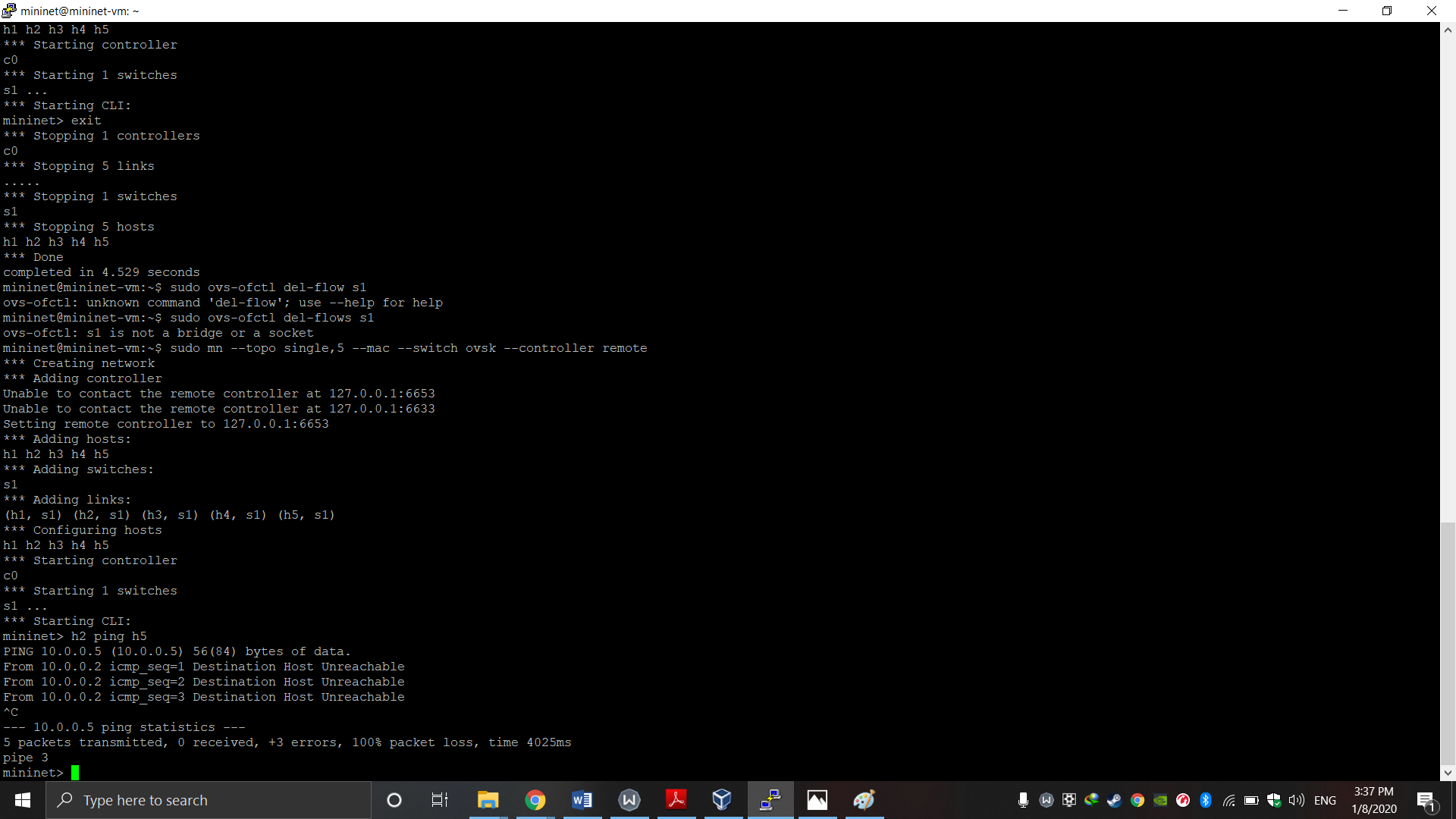


Figure 2.5. Network without controller

In this figure, we are creating a new network with a remote controller. At the same time, the switch does not have any stream entries and it cannot add new stream entries to its table. That’s why ping between h2 and h5 hosts was unsuccessful.

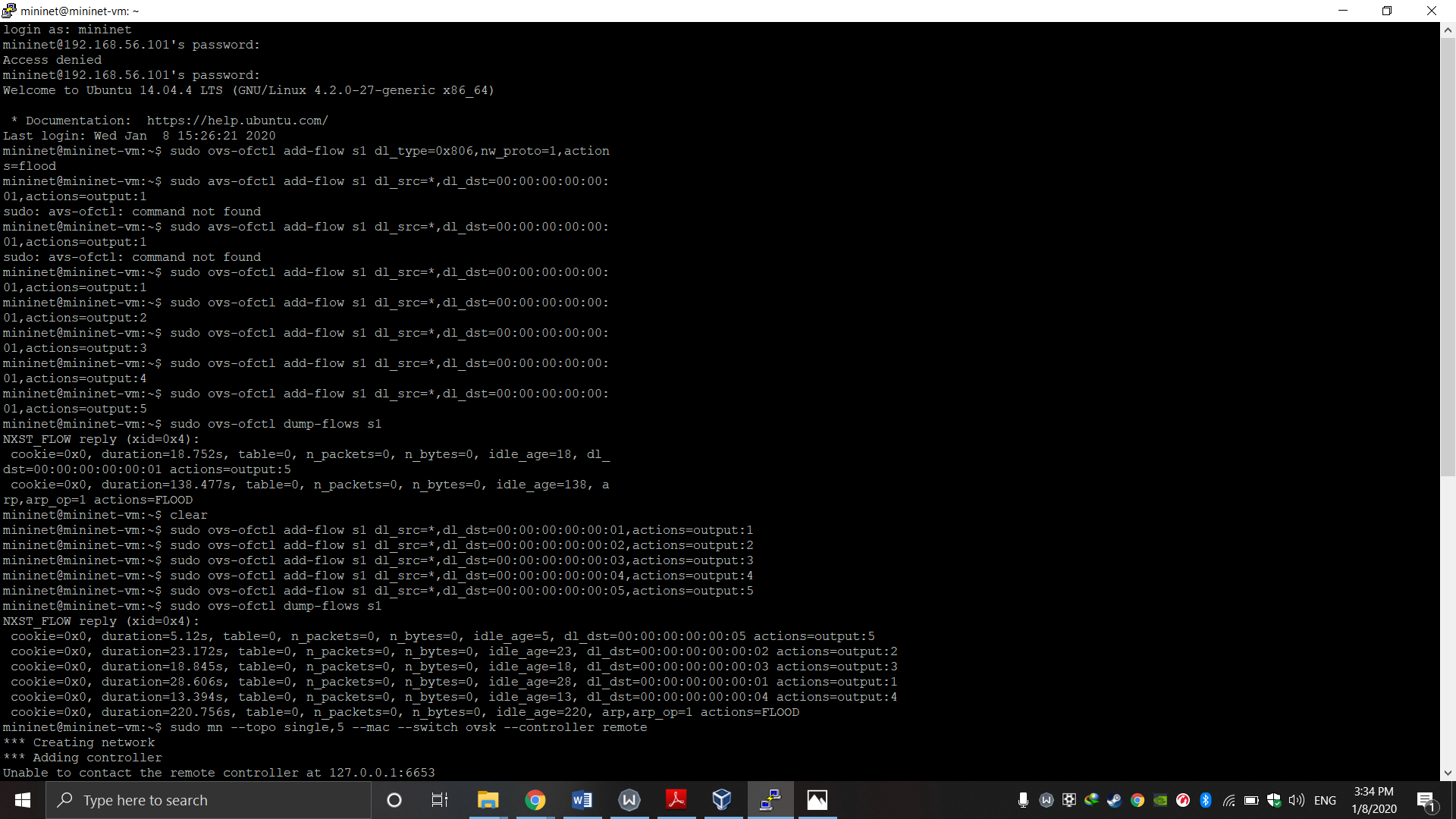


Figure 2.6. MAC flow entry

Figure 2.6 demonstrates a way to enable connectivity between the hosts adding MAC flow entry to switch. The command: *sudo ovs-ofctl add-flow s1 dl\_src=\*,dl\_dst=(MAC),actions=output(Port)* sends all packets to the MAC address through specified port.

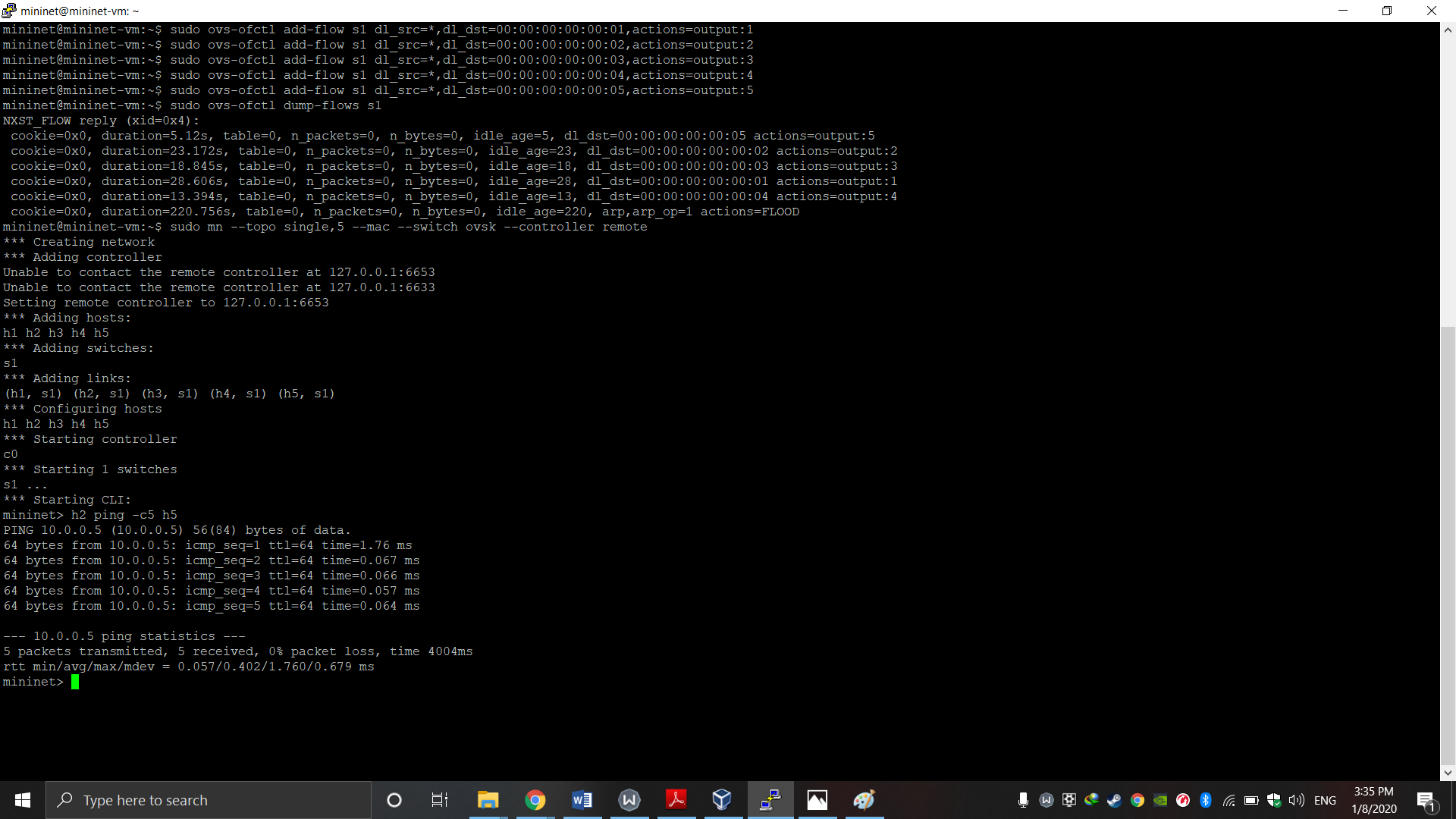


Figure 2.7. Checking connectivity between h2 and h5 hosts

After setting up MAC flow entry switch can send packets between all hosts and ping was successful.

**3.3. Task Three:**

Develop a software program to find the paths from one router to all the other routers in a connected network of at least 6 nodes using the Link State or Distance Vector routing algorithm. The link costs are known in advance. The input is a network topology with given link lengths or costs and the source node, and the output gives the shortest or least-cost paths from a source node to all the other network nodes. A detailed solution with a properly labeled network diagram, program code and the information of all paths generated is required. Students in the same team should differentiate their results on this task by choosing dissimilar network topologies and assigning different sets of link distance or link cost values for their own approach, although discussions between team members on routing algorithms and software applied are encouraged.

**Code for Distance Vector algorithm**

*Distance Vector Algorithm*

Each node uses the distance information (distance vector or cost) with its directly connected neighbors to determine the leas-cost link or path from the node to all neighbors. This process continues until all the leas-cost paths from the node to all the other nodes in the network are found

*Bellman-Ford Algorithm:*

* Find all neighbors in 1 hop – initial DV or least-cost paths.
* Find all neighbors in 2 hops – updated least-cost paths.
* Find all neighbors in n hops – final least-cost paths to all other nodes in the network.

Main formula for finding node j for which the (h+1)-hop cost is minimum is:

*h* - Number of hops being propagated.

- Cost of link i-j.

- Cost of the h-hop path from s to v.

*Code in Java*

import java.util.Scanner;

public class Main

{

private int distances[];

private int numberofvertices;

public static final int MAX\_VALUE = 999;

public Main(int numberofvertices)

{

this.numberofvertices = numberofvertices;

distances = new int[numberofvertices + 1];

}

public void BellmanFordEvaluation(int source, int adjacencymatrix[][])

{

for (int node = 1; node <= numberofvertices; node++)

{

distances[node] = MAX\_VALUE;

}

distances[source] = 0;

for (int node = 1; node <= numberofvertices - 1; node++)

{

for (int sourcenode = 1; sourcenode <= numberofvertices; sourcenode++)

{

for (int destinationnode = 1; destinationnode <= numberofvertices; destinationnode++)

{

if (adjacencymatrix[sourcenode][destinationnode] != MAX\_VALUE)

{

if (distances[destinationnode] > distances[sourcenode]

+ adjacencymatrix[sourcenode][destinationnode])

distances[destinationnode] = distances[sourcenode]

+ adjacencymatrix[sourcenode][destinationnode];

}

}

}

}

for (int sourcenode = 1; sourcenode <= numberofvertices; sourcenode++)

{

for (int destinationnode = 1; destinationnode <= numberofvertices; destinationnode++)

{

if (adjacencymatrix[sourcenode][destinationnode] != MAX\_VALUE)

{

if (distances[destinationnode] > distances[sourcenode]

+ adjacencymatrix[sourcenode][destinationnode])

System.out.println("The Cost contains negative number");

}

}

}

for (int vertex = 1; vertex <= numberofvertices; vertex++)

{

System.out.println("Distance of source " + source + " to "

+ vertex + " is " + distances[vertex]);

}

}

public static void main(String... arg)

{

int numberofvertices = 0;

int source;

Scanner scanner = new Scanner(System.in);

System.out.println("Enter the number of nodes");

numberofvertices = scanner.nextInt();

int adjacencymatrix[][] = new int[numberofvertices + 1][numberofvertices + 1];

System.out.println("Enter the adjacency matrix");

for (int sourcenode = 1; sourcenode <= numberofvertices; sourcenode++)

{

for (int destinationnode = 1; destinationnode <= numberofvertices; destinationnode++)

{

adjacencymatrix[sourcenode][destinationnode] = scanner.nextInt();

if (sourcenode == destinationnode)

{

adjacencymatrix[sourcenode][destinationnode] = 0;

continue;

}

if (adjacencymatrix[sourcenode][destinationnode] == 0)

{

adjacencymatrix[sourcenode][destinationnode] = MAX\_VALUE;

}

}

}

System.out.println("Enter the source node");

source = scanner.nextInt();

Main bellmanford = new Main(numberofvertices);

bellmanford.BellmanFordEvaluation(source, adjacencymatrix);

scanner.close();

}

}

The Code for this algorithm was written in Java. It has main function – *BellmanFordEvaluation()* which calculates the shortest path. Also in code we can see loops for representing graph.

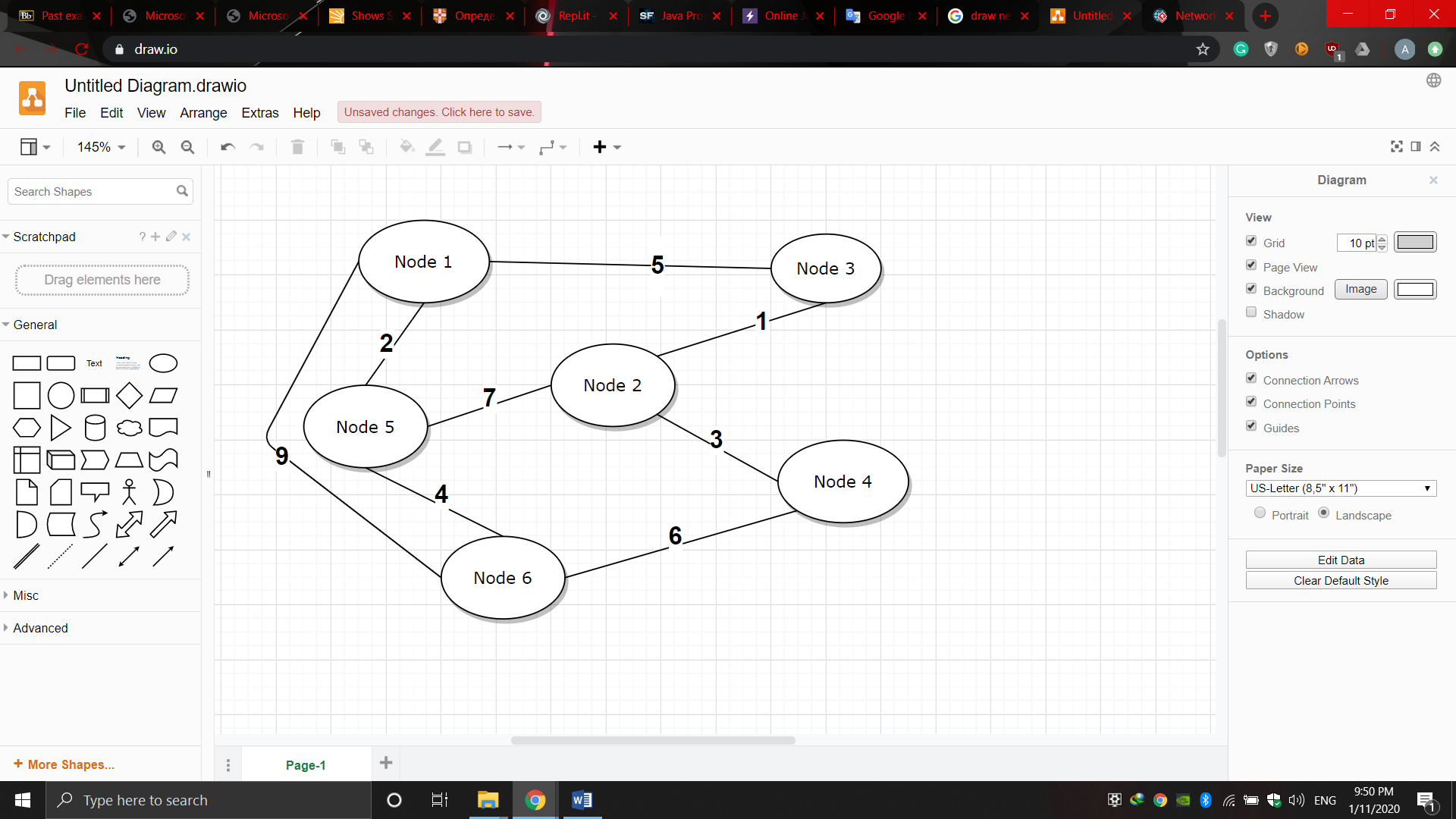


Figure 3.1. Network Diagram

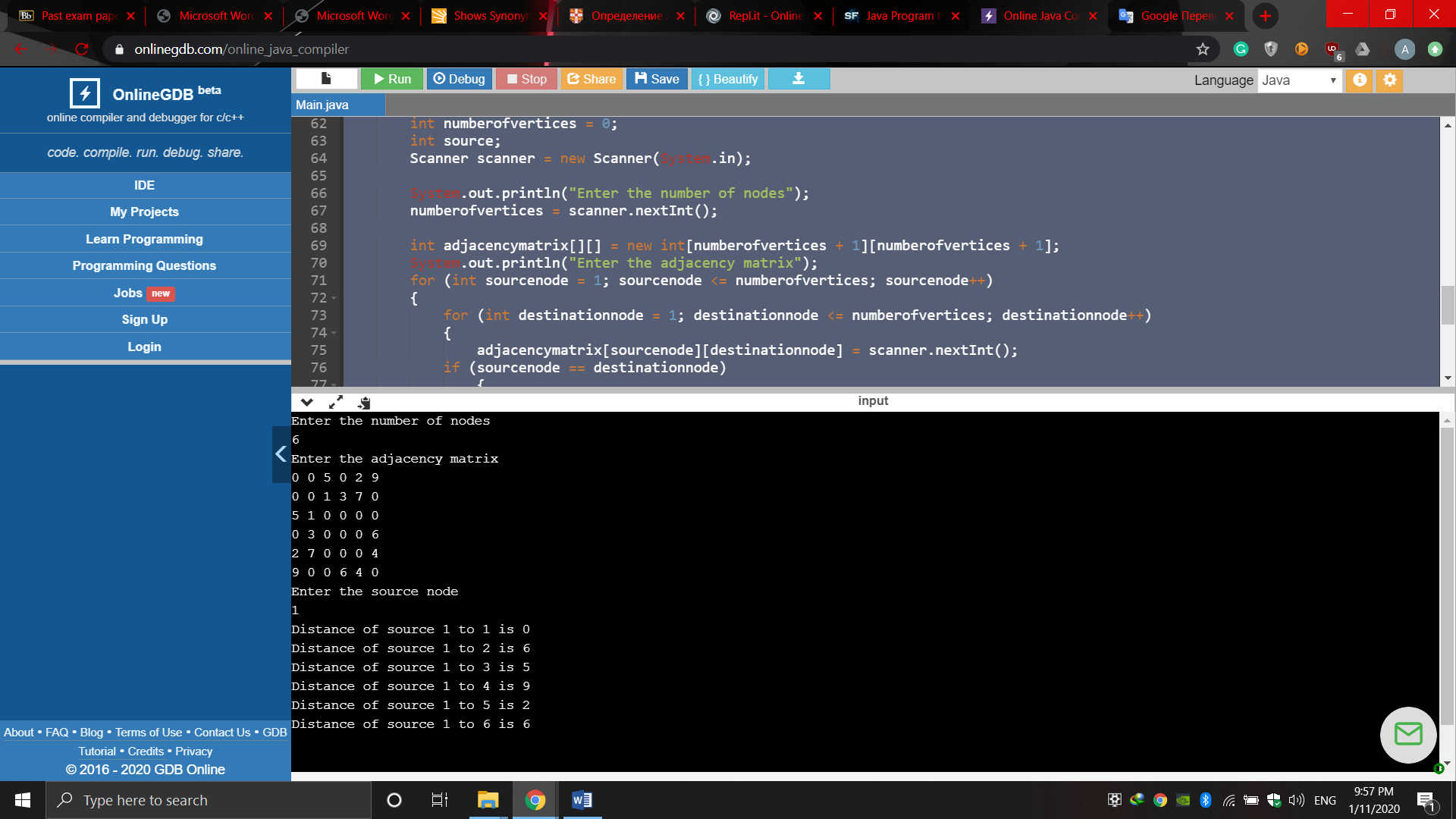


Figure 3.2. Results with node 1 as source node

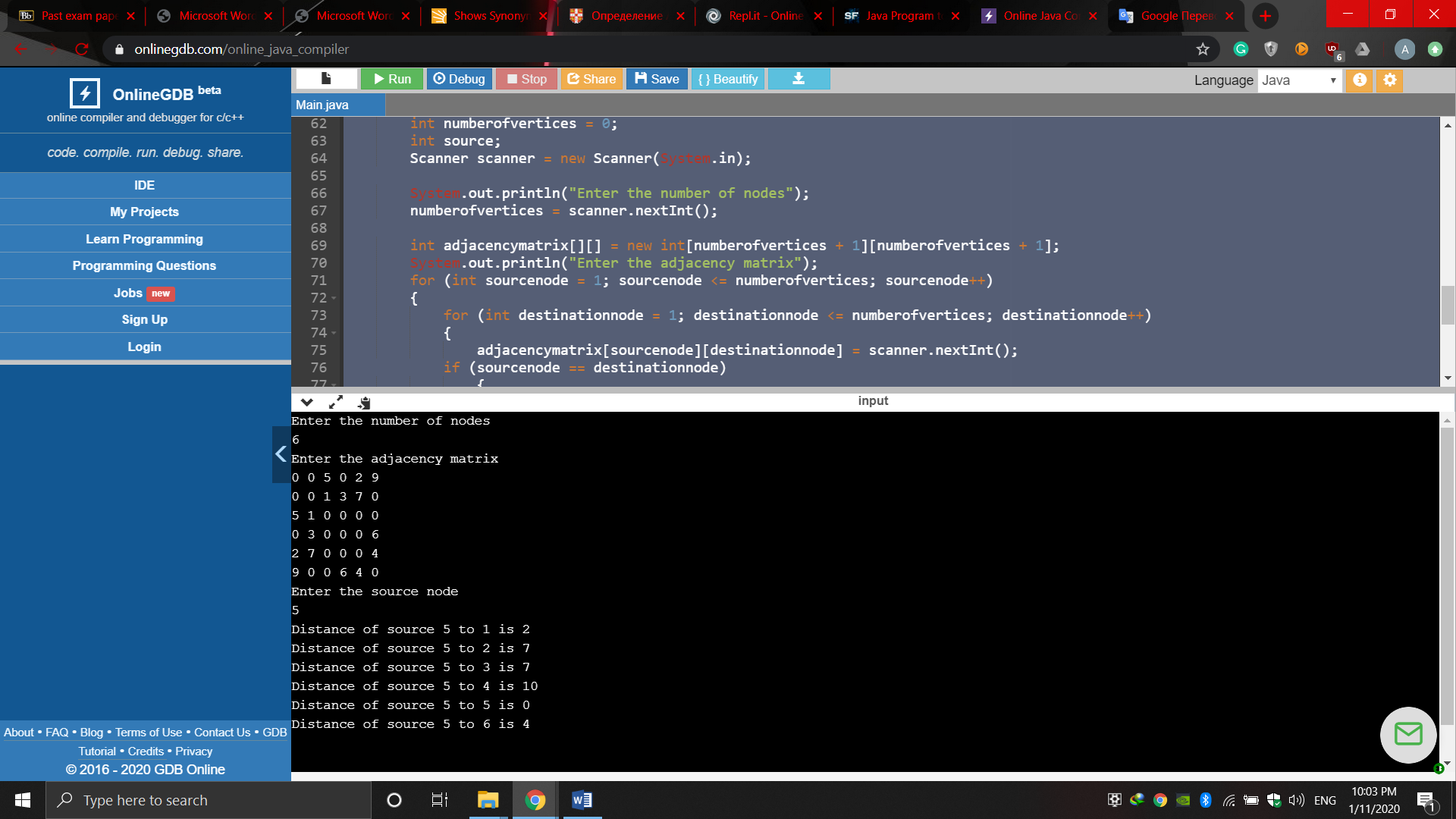


Figure 3.3. Results with node 5 as source node

Adjacency matrix is a square matrix used to represent a finite graph. The elements of the matrix indicate whether pairs of vertices (nodes) are adjacent or not in the graph.

Figure 3.2 shows the results when the source node is the “Node 1”, while figure 3.3 shows the results when the source node is the “Node 5”. The results match the expected distances, therefore, they are correct.

4. Conclusion

In this laboratory work, we examined the development environment of the Mininet and its relationship between the OpenFlow protocols. We became familiar with the basic Mininet commands, as well as with another program like a Wireshark for monitoring traffic on the network. In the last task, we inspected several routing algorithms. As a result, we chose Distance Vector Algorithm more precisely Bellman-Ford Algorithm and developed a program in the Java programming language for calculating the shortest path.

5. Bibliography

* Dr. Peng, 2019, [Assignment Preparation for Open Flow and Mininet](https://learn-eu-central-1-prod-fleet01-xythos.s3-eu-central-1.amazonaws.com/5d2cb9c32e9d7/3194943?response-content-disposition=inline%3B%20filename%2A%3DUTF-8%27%27Asssignment%2520preparation%2520for%2520mininet%2520and%2520openflow%25281%2529.pdf&response-content-type=application%2Fpdf&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Date=20191217T165736Z&X-Amz-SignedHeaders=host&X-Amz-Expires=21600&X-Amz-Credential=AKIAZH6WM4PLYI3L4QWN%2F20191217%2Feu-central-1%2Fs3%2Faws4_request&X-Amz-Signature=d75de3c1393a988a82a0fb33f250be6bf7048eaa842362a8aa755a2db8b6ff86)
* Dr.Peng, 2019, Lecture slides for Telecommunications Networks and Quality of Service.
* Mininet Overview Webpage, viewed 13 Dec. 2019, <http://mininet.org/overview/>
* Bellman-FordAlgorithm in Java webpage, viewed 17 Dec. 2019, available at: <https://www.sanfoundry.com/java-program-implement-bellmanford-algorithm/>