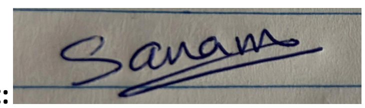
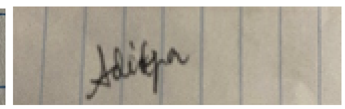
Name: Suresh, Aditya & Sanam Palsule Date: 5/14/2024

NYU ID: as17339 & sp7940@nyu.edu

Course Section Number: CSCI-GA.2662-001

Affirmation of my independent effort: 



Total in Points (100 points total):

Professor’s Comments:

Introduction

Software-Defined Networking (SDN) is an approach to networking that uses software-based controllers or application programming interfaces (APIs) to communicate with underlying hardware infrastructure and direct traffic on a network.

This project aims to present the design and implementation of a software-defined networking (SDN) application that combines layer-3 shortest path routing with distributed load balancing.

Two primary applications :

1. Shortest Path Routing
2. LoadBalancing

The ShortestPathSwitching module computes and installs shortest path routes between hosts using Dijkstra's algorithm, while the LoadBalancer module distributes incoming TCP connections across a set of backend hosts. The two modules work together to provide efficient routing and load balancing in an SDN environment.

Motivation

The goal of this project is to use the centralized control mechanism of SDN to improve network management in particular areas, like load allocation and rigid routing. The objective of this project is to illustrate the useful advantages of SDN in practical applications through implementing a distributed load balancing system and a shortest-path routing application within an SDN framework.

System Architecture

System contains 3 main components:

1. SDN controller i.e. floodlight
2. SDN capable switches
3. Host machines

All this is simulated within the Mininet.

PART 1: Referenced Documents

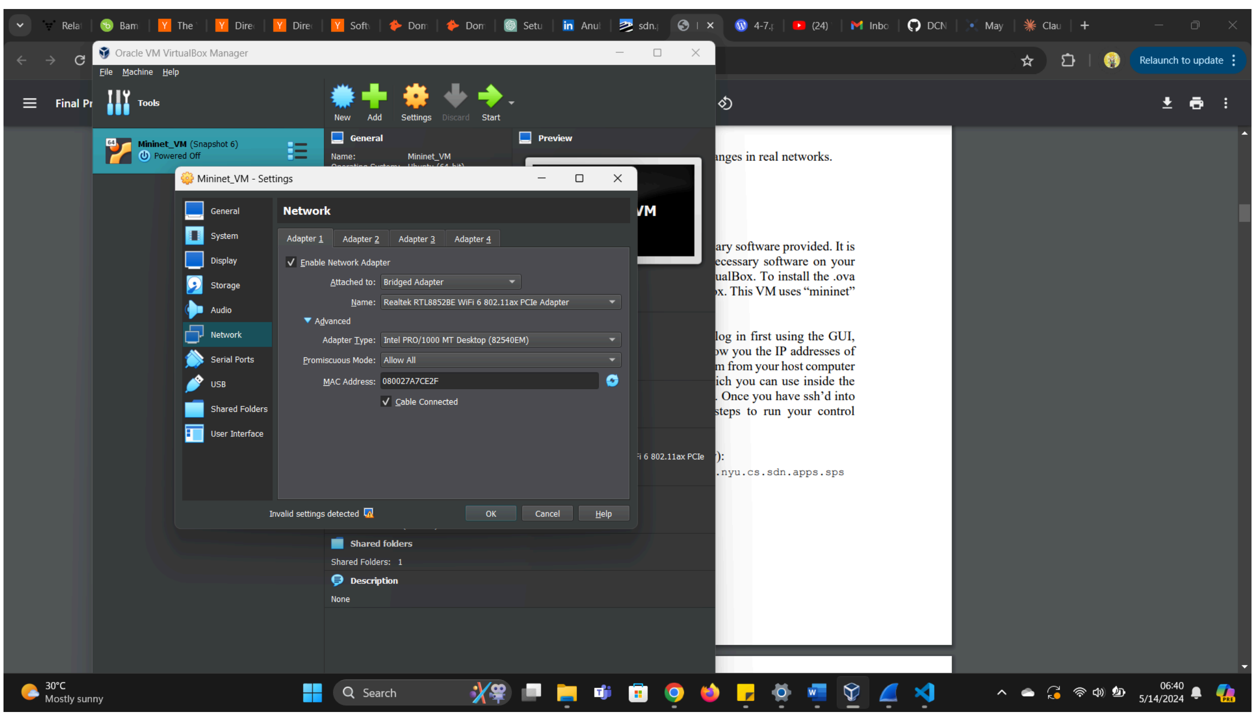
We thoroughly reviewed all the documentation and referenced materials provided to gain a comprehensive understanding of the project's background, as well as the associated concepts and technologies.

PART 2: Oracle VM Ubuntu

a. *Installed the Oracle Virtual box*

b. *Downloaded the virtual box image and set it up :*

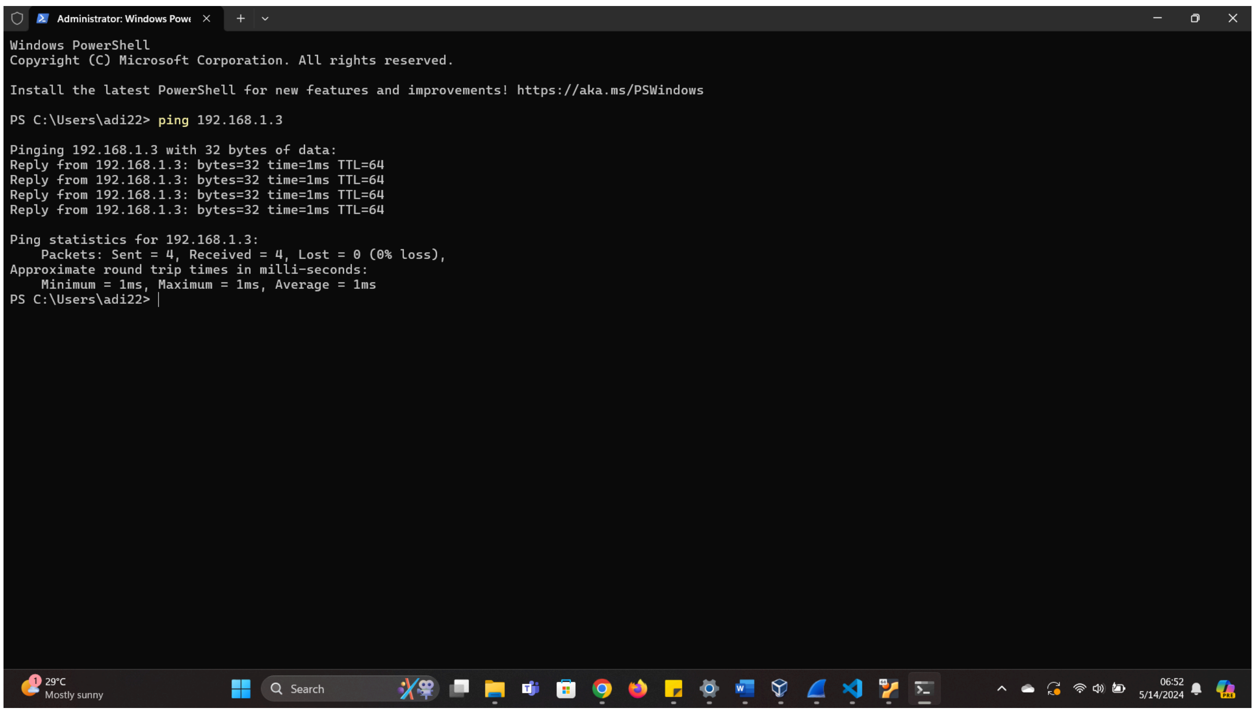
Initial Setup:



Adjust the network settings within VirtualBox by transitioning Network Adapter 1 from NAT to Bridged Adapter. Additionally, modify the Promiscuous Mode from Deny to Allow all. This configuration alteration facilitates SSH access into the virtual machine from the local host machine.

c. SSH:

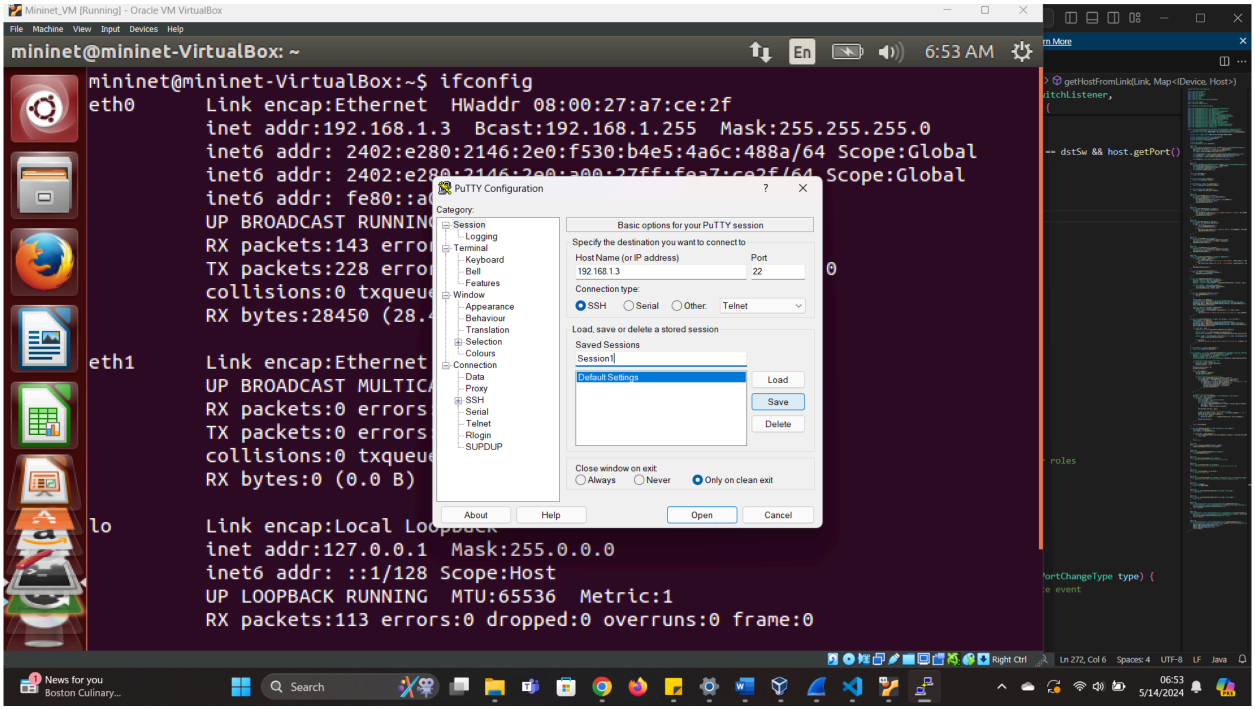
- First, start the virtual machine.
- Go into the terminal and type ifconfig.
- Get the inet address (IP) and save it.
- In our case it is 192.168.1.3



- To see if packets can be transferred from our local machine to the virtual machine, we used the terminal in windows, and typed ping <IP Address that we noted down>.
- Since the packets were transferred successfully, we can now ssh into the virtual machine using the same IP address.

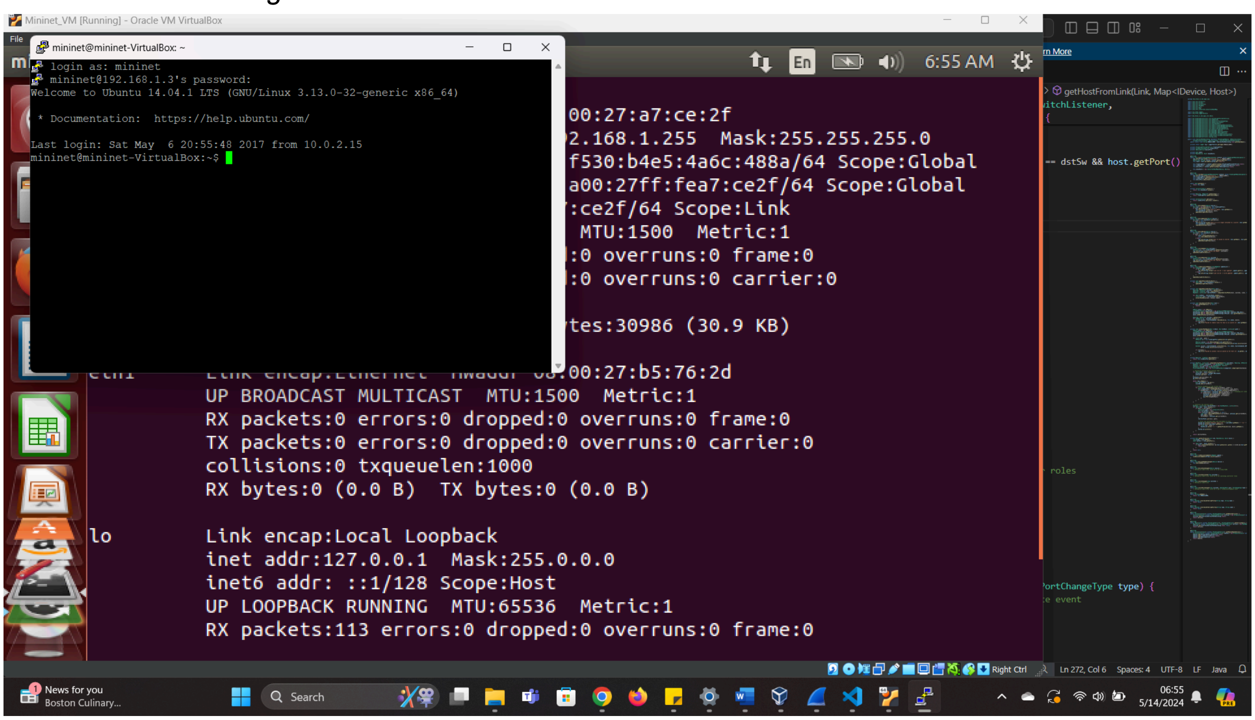
Using Putty:

To ssh into the virtual machine, we downloaded an application called putty. The image below shows its use:



- We entered the IP address under Host Name.
- Leave the port as port 22.
- The connection type was ssh.
- We saved the session so that we don’t have to do this every time we restart.

Successful SSH Image:

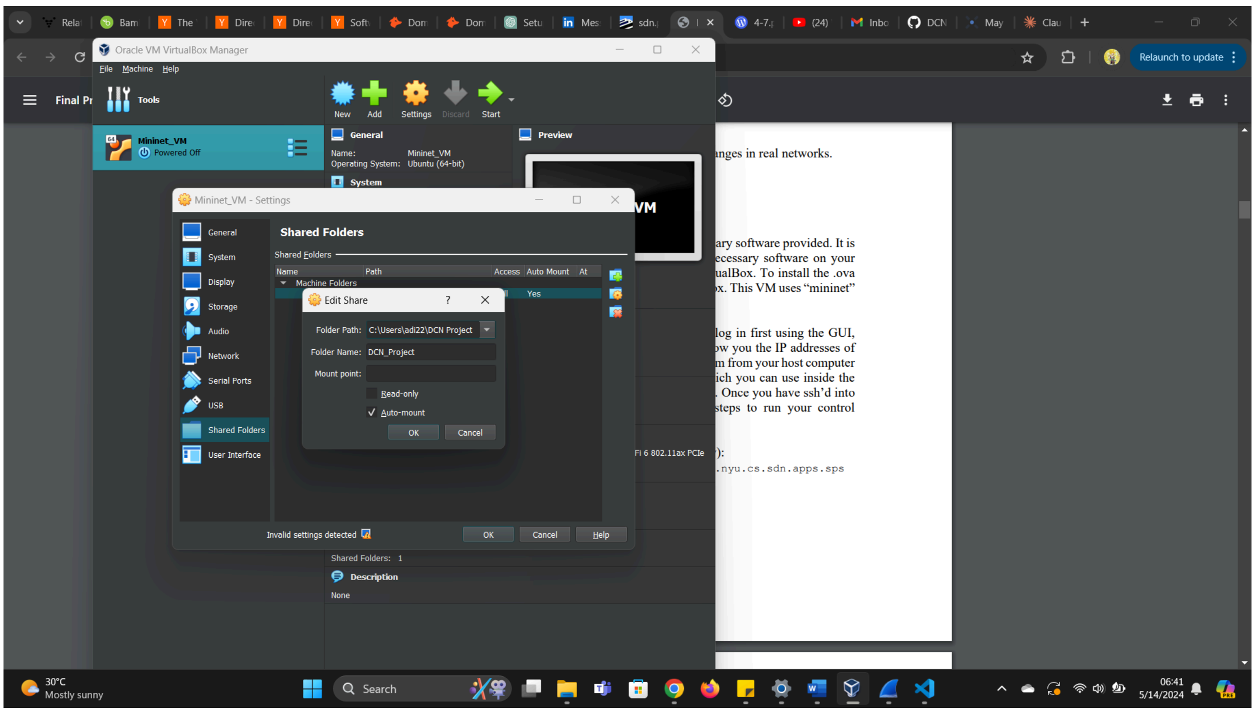


- Once we login using mininet, we can see that the display says Welcome to Ubuntu, meaning ssh was successful.

Shared Folder for easier coding access:
Since coding in windows is faster than the virtual machine, we needed to find a way to transfer the java files. We could use ssh, but for dynamic changes, it would be easier to set up a shared folder. For this we did the following:

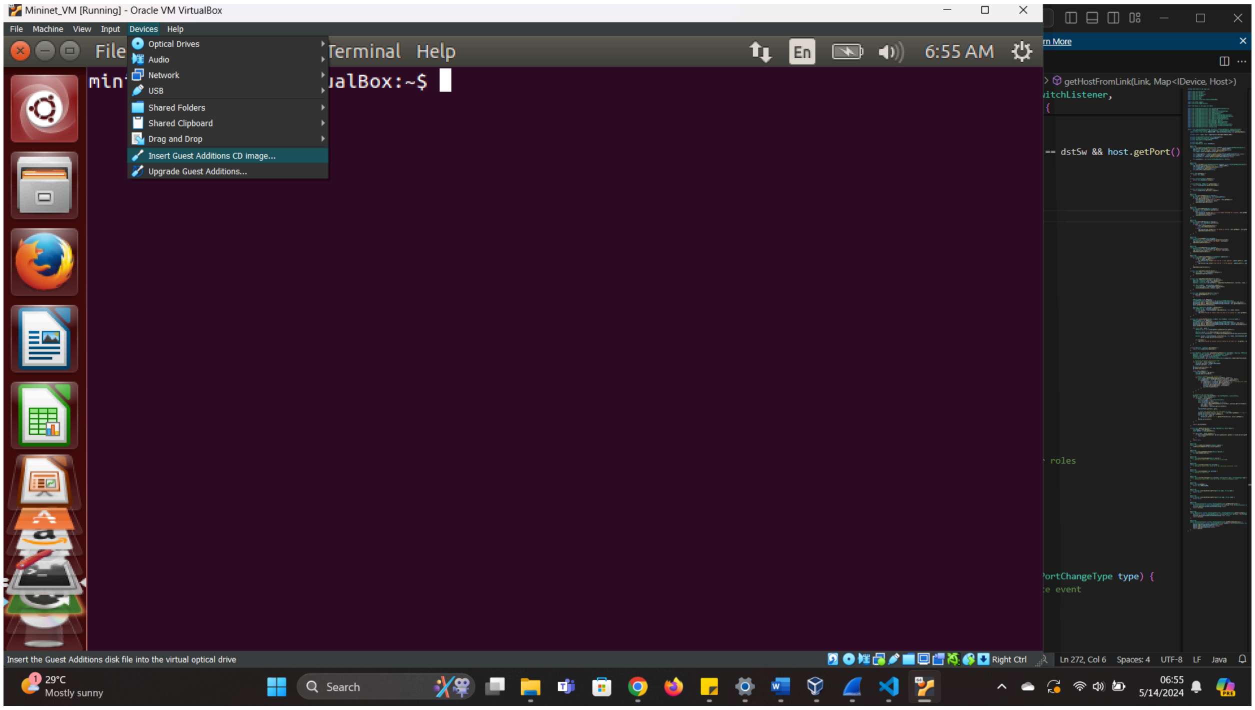
Step 1:

- Shut down the mininet and go back to the virtual image settings.
- Go to the shared folder option, click on the ‘+’ icon to add a local path folder.
- In our case, we have used “C:\Users\adi22\DCN_Project” as our path, where the changes to the code will be made.
- Check the auto mount option so that it mounts automatically on restart.



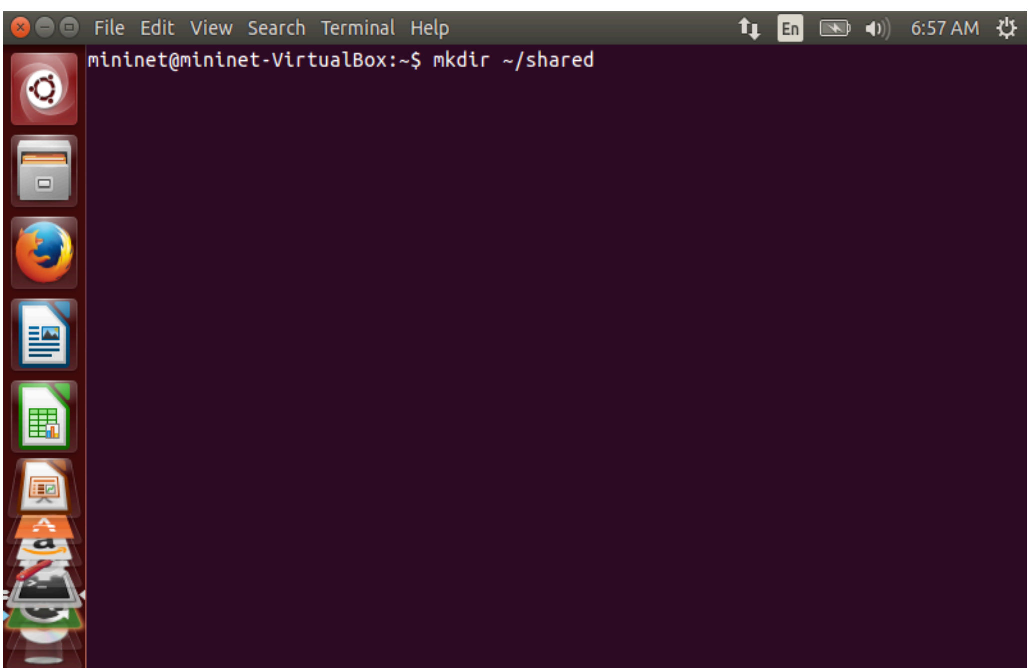
Step 2:

- Installing guest additions and utilities:
- Start the virtual machine.
- Go to the devices tab on the virtual box and click on install guest additions cd image.
- This installs the guest utilities needed to share the folder from virtual to local, or guest to host.
- It will open a terminal window where various utilities are installed.

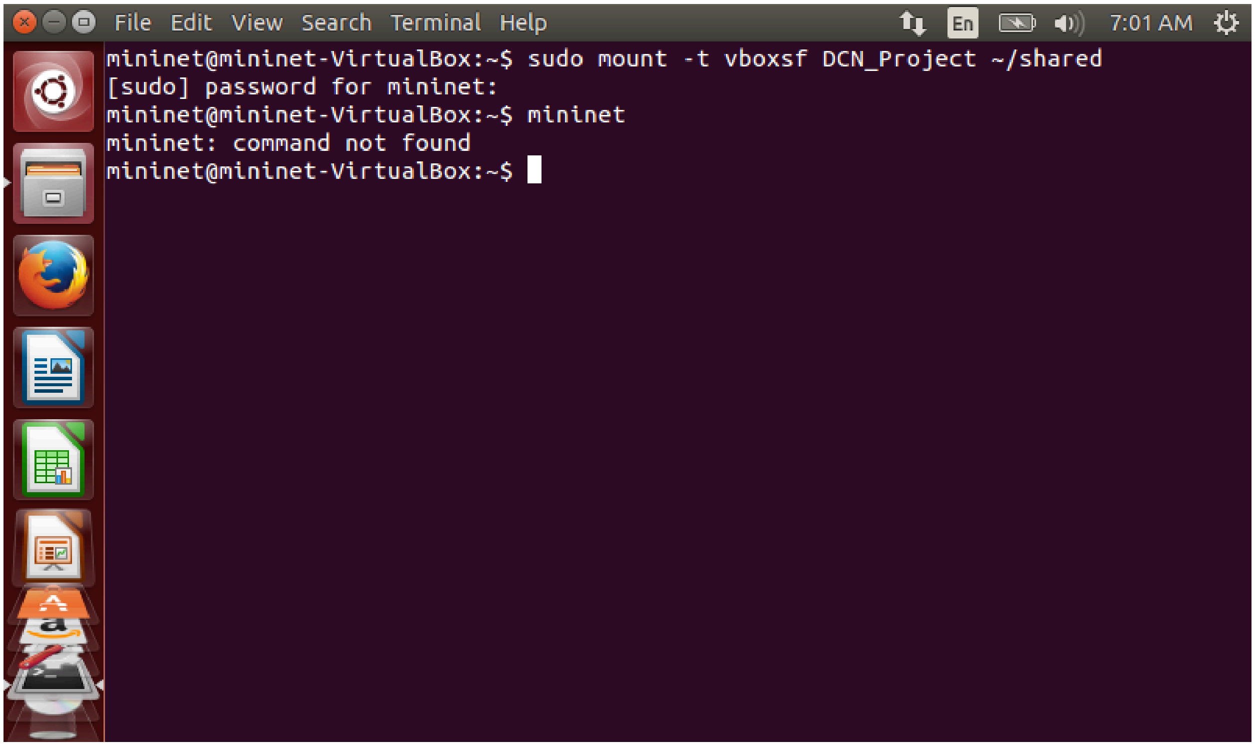


Step 3:

Make a new directory “shared” which would eventually contain the shortestPathSwitching.java file. The terminal command for that is shown below:

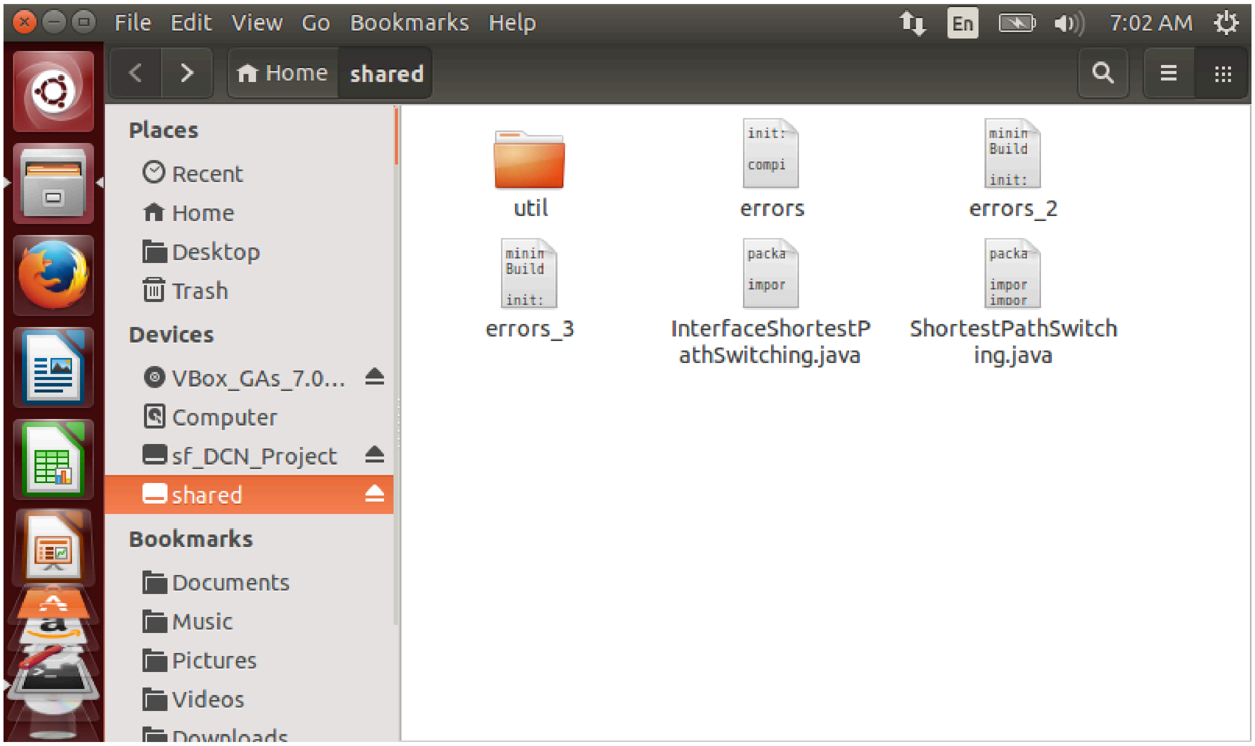


Step 4:
Mount this shared file on to the DCN_Project folder that we had earlier specified in the shared folders. Terminal command below:

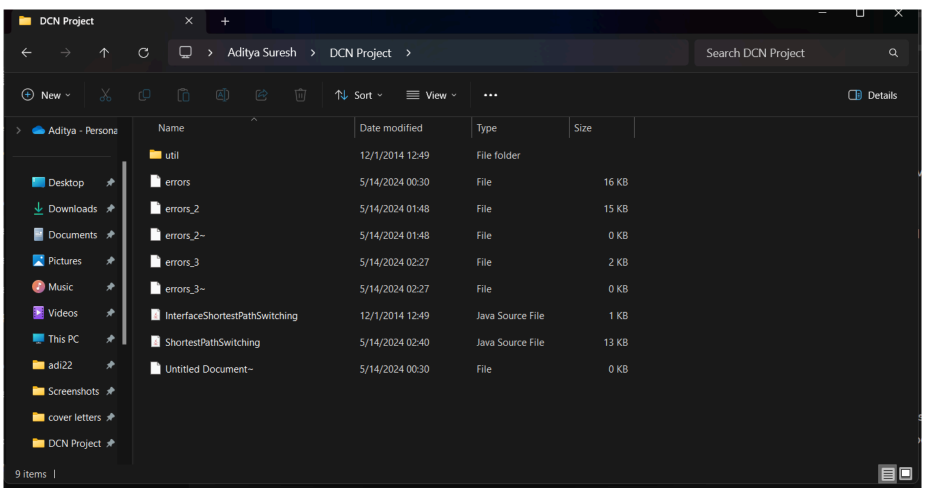


Step 5:

- Once it is mounted, go to the openflow folder, copy the ShortesPathSwitching.java file from edu.brown.cs.sdn.apps.sps
- We have also pasted other files, but it is not needed. The shared folder is shown below:



The same folder in windows:



Changes made in the ShortestPathSwitching.java file in the local machine will now be reflected on the shared folder in the virtual machine.

PART 3: Shortest Path Switching

ShortestPathSwitching application implementation:

Logic and implementation of the code :
This part mainly focuses on computing shortest path routes between the hosts. We have utilized Dijkstra's algorithm to find the shortest paths,

Dijkstra's Algorithm:

The algorithm maintains:
- a distances map to store the distances
- a previous map to keep track of the previous hop in the shortest path
- a priority queue (PQ) to efficiently select the unvisited host with the smallest distance in each iteration

The algorithm :
- Initialize the distances and previous maps
- Add the source host to the PQ.
- Until all reachable hosts are visited:
  - ➢ Remove the host with the smallest distance from the PQ
  - ➢ Update the distances and previous hops of its unvisited neighbors
  - ➢ Add those neighbors to the PQ.
- Finally, the shortest paths are constructed by tracing back the previous hops from each destination host to the source.

We have also added the code with comments below:

```
private Map<Host, List<Link>> computeShortestPaths(Host sourceHost,
Map<Long, IOFSwitch> switches, Map<Link, LinkInfo> links, Map<IDevice,
Host> hosts) {
    // Initialize distances and previous maps
    Map<Host, Integer> distances = new HashMap<Host, Integer>();
    Map<Host, Host> previous = new HashMap<Host, Host>();

    // Set the distance of the source host to 0 and all other hosts to
infinity
    for (Host host : hosts.values()) {
        distances.put(host, Integer.MAX_VALUE);
        previous.put(host, null);
    }
    distances.put(sourceHost, 0);
```

```
    // Create a priority queue to store hosts based on their distances
    PriorityQueue<Host> pq = new
PriorityQueue<Host>(Comparator.comparingInt(distances::get));
    pq.offer(sourceHost);

    // Set to keep track of visited hosts
    Set<Host> visited = new HashSet<Host>();

    // Dijkstra's algorithm main loop
    while (!pq.isEmpty()) {
        // Remove the host with the smallest distance from the priority
queue
        Host currentHost = pq.poll();
        visited.add(currentHost);

        // Explore neighbors of the current host
        for (Link link : getNeighborLinks(currentHost, links)) {
            Host neighborHost = getNeighborHost(currentHost, link, hosts);
            if (neighborHost != null && !visited.contains(neighborHost)) {
                // Calculate the new distance to the neighbor via the
current host
                int newDistance = distances.get(currentHost) + 1; //
Assuming all links have equal weight of 1

                // If the new distance is smaller than the neighbor's
current distance, update it
                if (newDistance < distances.get(neighborHost)) {
                    distances.put(neighborHost, newDistance);
                    previous.put(neighborHost, currentHost);
                    pq.offer(neighborHost);
                }
            }
        }
    }

    // Construct the shortest paths using the previous map
    Map<Host, List<Link>> shortestPaths = new HashMap<Host, List<Link>>();
    for (Host host : hosts.values()) {
        if (host != sourceHost) {
            List<Link> path = new ArrayList<Link>();
            Host currentHost = host;
            while (previous.get(currentHost) != null) {
                Link link = getLinkBetweenHosts(currentHost,
```

```
previous.get(currentHost), links);
                path.add(0, link);
                currentHost = previous.get(currentHost);
            }
            shortestPaths.put(host, path);

            // Print the shortest path from sourceHost to host (for
debugging purposes)
            System.out.println("Shortest path from " + sourceHost.getName()
+ " to " + host.getName() + ":");
            System.out.print(sourceHost.getName());
            for (Link link : path) {
                System.out.print(" -> " + getHostFromLink(link,
hosts).getName());
            }
            System.out.println();
        }
    }

    return shortestPaths;
}
```

Apart from ‘computeShortestPaths’ a few other methods were also added/ modified:

installPathRules(Host srcHost, Host dstHost, List<Link> path):

- We have defined this method for installing rules on switches to route packets from a source host to a destination host based on a given path. Using this method, we set up the forwarding rules for the shortest paths computed by another method 'computeShortestPaths'.
- Working : the method iterates over each link and for each link, source switch is found, creates OFAction to output packet and OFinstruction to create instruction. Then it calls the installRule() command from utils class switchCommands to install the rule for that switch.

removeRulesForHost(Host host):

- When a host is moved/removed, we need to remove all the rules related to that switch.
- Working : Gets the map of all switches and removes rules using removeRules from util class SwitchCommand.

updateRoutingForHost(Host host) :

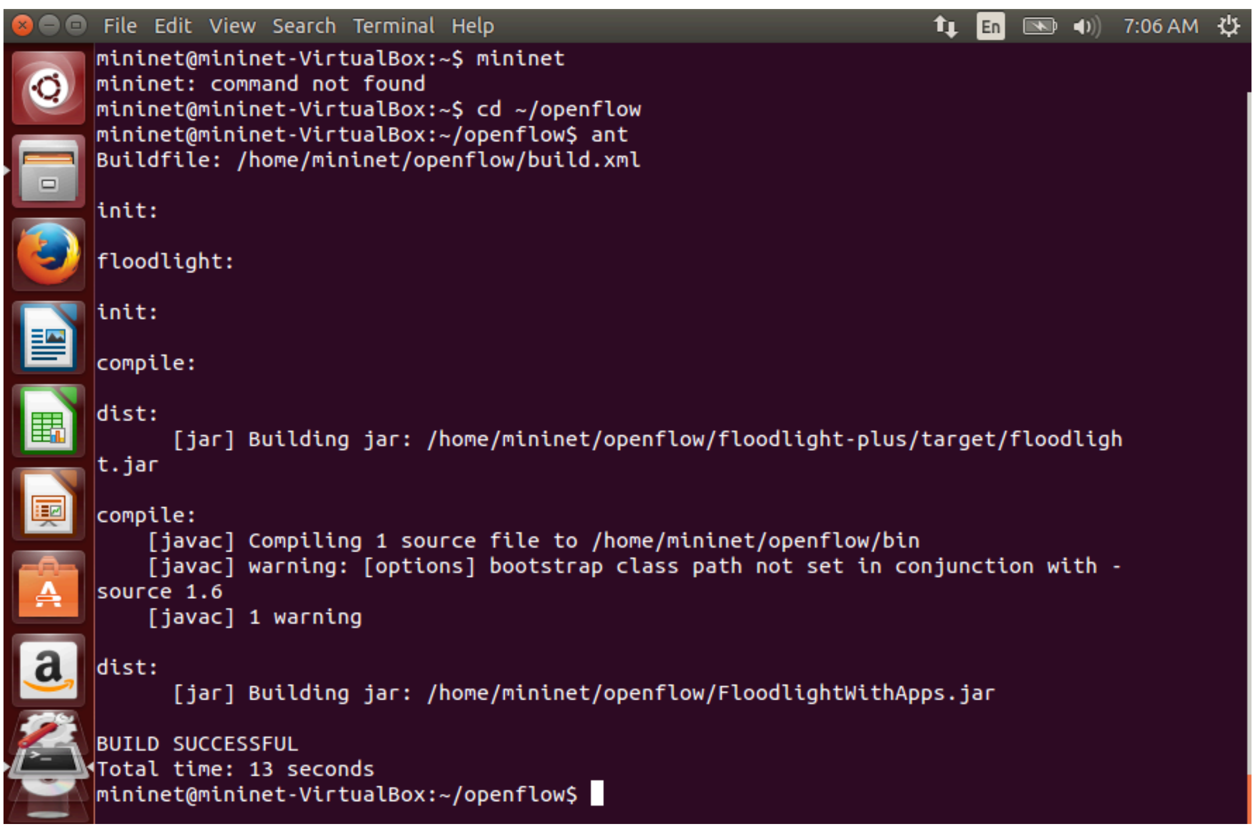
- This method updates routing rules for all hosts in the network which further calls shortestPath method
- Use of this method : This is called when there is a change in the network topology, ex: switch/host are added /removed

updateRoutingForAllHosts():

- Responsible for updating the routing rules for all known hosts in the network.
- It iterates over each host and for each host it calls updateRoutingForHost(host)

Once relevant changes are made to the ShortestPathSwitching.java code, copy this file from the shared folder and replace it in the openflow folder path edu.brown.cs.sdn.apps.sps.

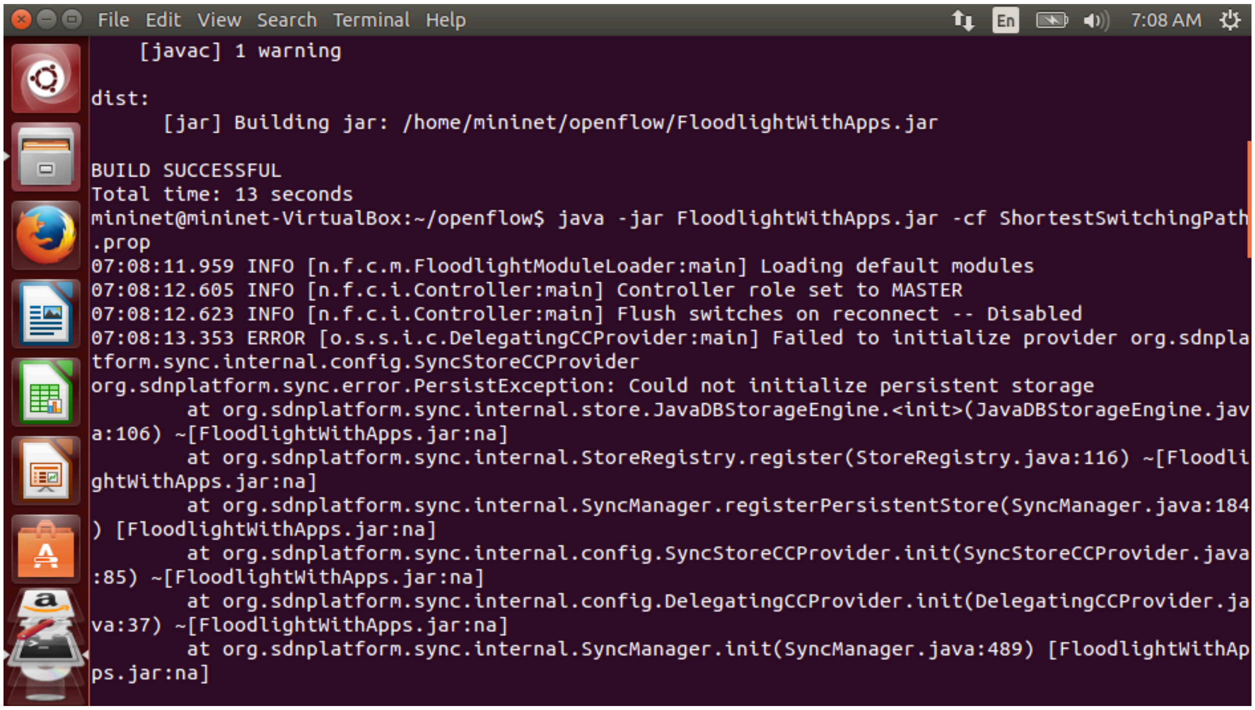
Jar Build:
After the code is ready, compile and build the jar file using the following command:



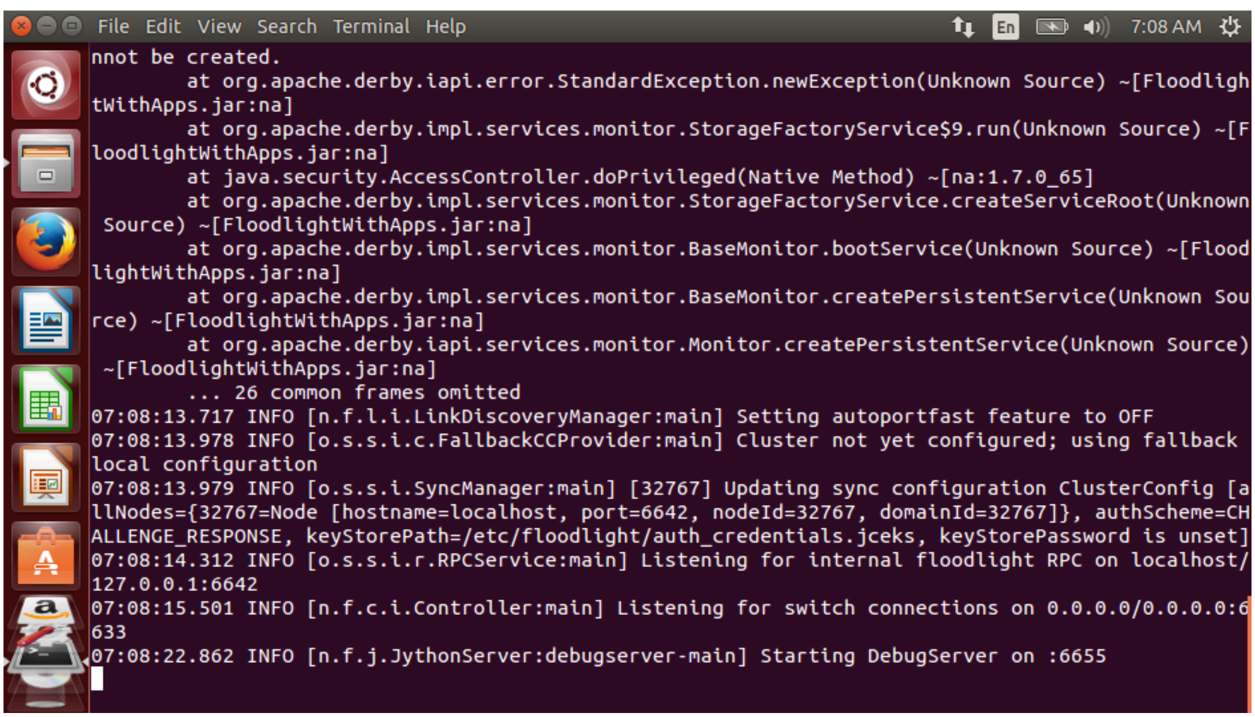
If the code is working, it should say “build successful”

Configuring ShortestPathSwitching:

After the successful build, run the following command in the terminal:

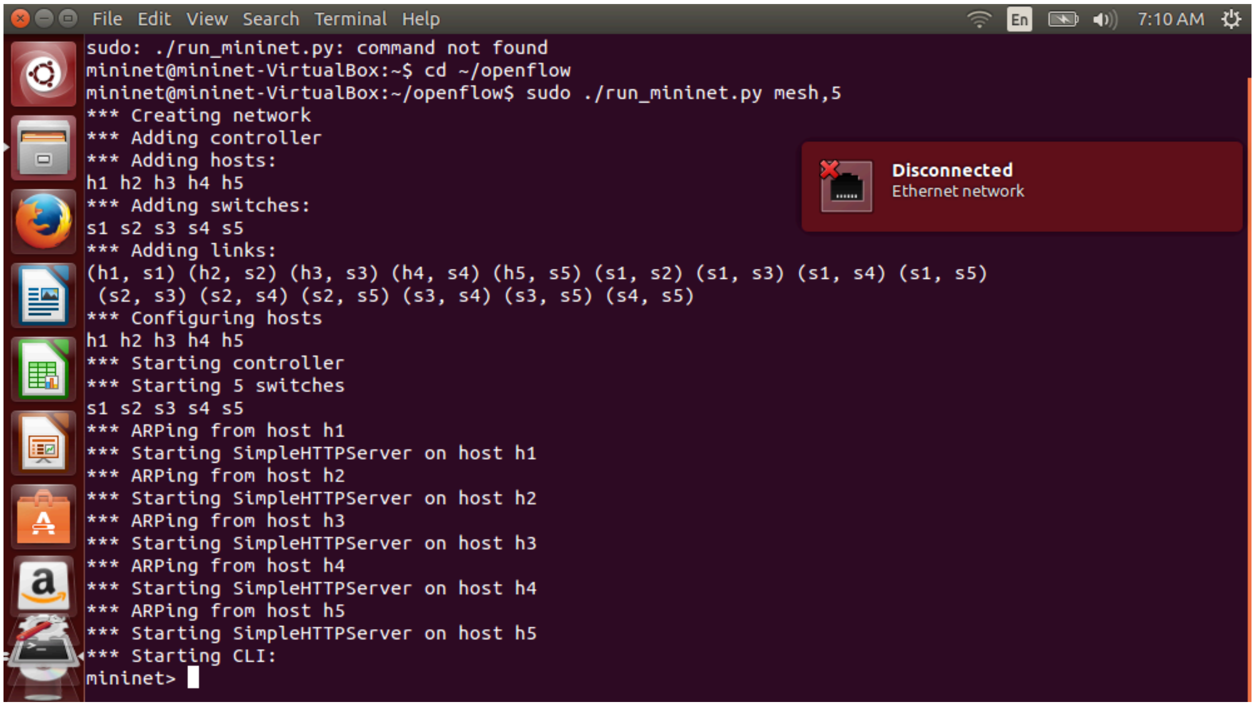


- Using the command java -jar FloodlightWithApps.jar -cf ShortestPathSwitching.prop configures the shortespathswitching.java file.
- This terminal screen is where the output will be displayed.
- The O/P will be displayed after the following “listening” string that is visible on the output screen:



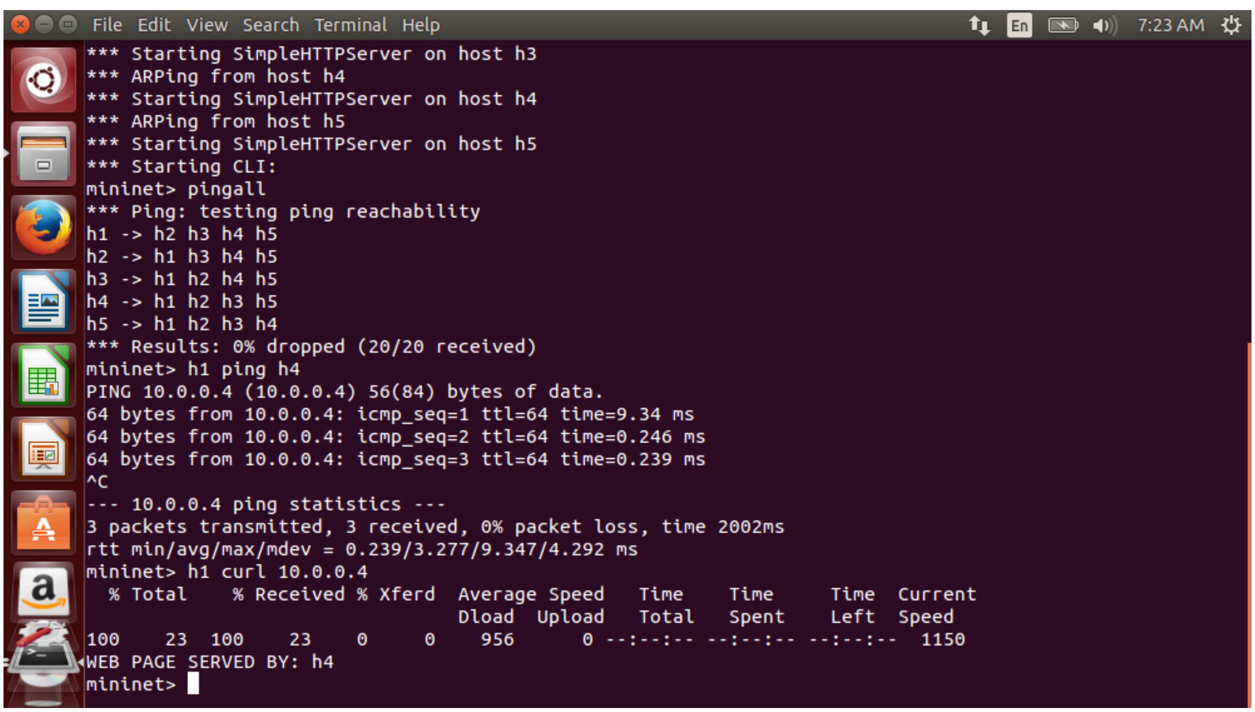
Running Mininet:

Run mininet using the following command:



- Here, we have used a mesh network with 5 hosts and switches.
- It is seen that every host has started a HTTP Server.

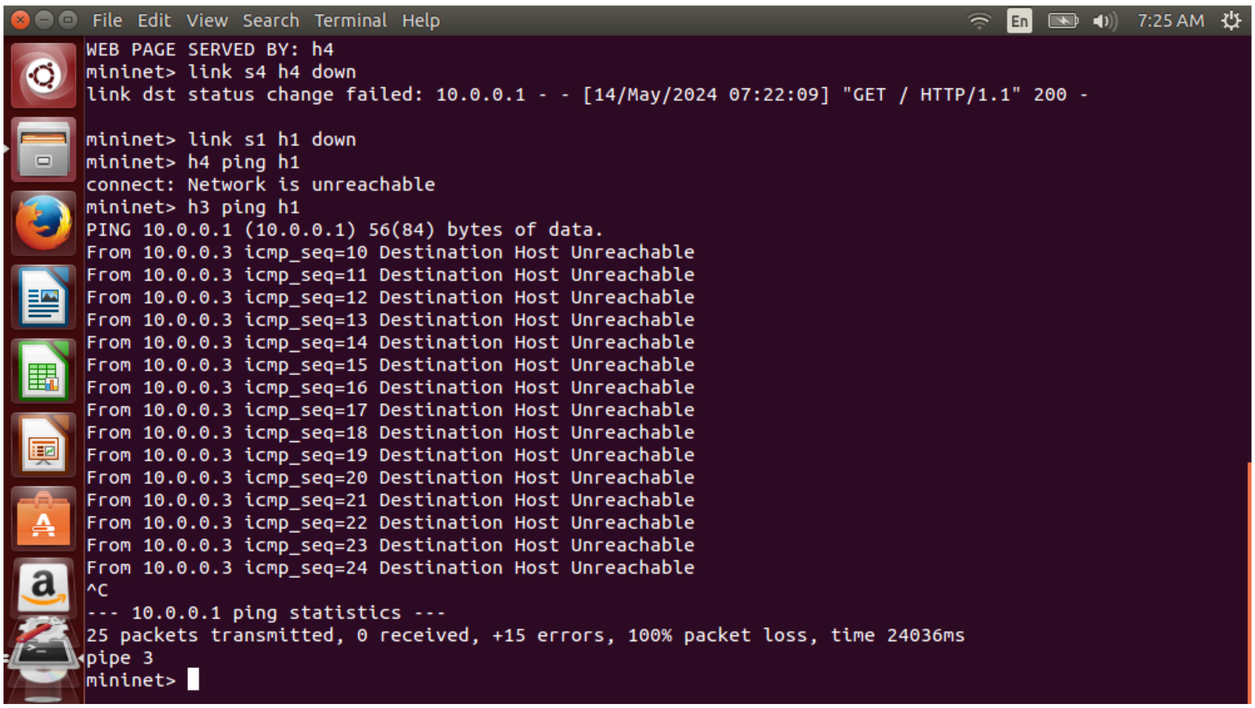
Mininet Commands:



- To check the reachability of all the hosts, we use the pingall command.

- If the packet transfer is lossless, it means that our code is running properly.
- To send packets from one host to another, we use the command- host_name ping another_host_name.
- In our case we have used 2 hosts h1 and h4 and sent packets from h1 to h4.
- As seen, the packet transfer from h1 to h4 is lossless and follows the shortest path.
- We have also used the curl command to see the ping statistics.

Testing:



- We use the command link switch_name host_name down to cut a link for the topology of the network.
- Here we have used an example of cutting the link between h1 and s1.
- In a mesh network this would mean that host 1 should be unreachable from all other hosts.
- To test this, we sent packets from different hosts to h1.
- As seen, it clearly says that host 1 is unreachable, confirming that the topology of our network is working correctly.

Testing shortestpathswitching.java:

- In the initial phase of our project, we deployed the shortest path algorithm and encapsulated it into a jar file for implementation. Operating within a mesh network architecture consisting of five hosts, we facilitated packet transmission from one host to another through this layer 3 network infrastructure.
- To monitor network traffic and assess the efficacy of our setup, we employed the 'sudo ovs-ofctl dump-flows' command to examine switch statistics in real-time.

- For validation purposes, we conducted a specific test scenario wherein packets were transmitted from host h1 to h4. By scrutinizing the table generated from the aforementioned command, we confirmed that the packets indeed traversed the shortest available path between the designated source and destination hosts.

PART 4: Distributed Load Balance Routing

LoadBalancer:

The LoadBalancer module is responsible for distributing incoming TCP connections across a set of backend hosts.

This is done by

- Installing openFlow rules in the switches to redirect TCP SYN packets for VIPs to the controller
- Selecting backend Host for each new connection
- Installing connection - specific rules to rewrite packet header

LOGIC AND IMPLEMENTATION:

A VIP related rule is installed in table 0 when a switch connects to the controller. These rules match the destination IP and MAC address of the VIPs and then does one if the following :

- Send a packet to the controller
- Forward it to table 1 for routing

If the controller receives TCP SYN packet destined for VIP, it does the following :

- Selects a host for new connection using ‘RoundRobin’
- Installs connection- specific rule in table 0 to rewrite addresses of subsequent packets.

If the controller receives a ARP request for VIP:

- Constructs an ARP reply containing virtual MAC address linked to the VIP
- Send it back to the requesting host.

Methods modified in the existing code:

switchAdded(long switchId):

- Installs a default rule to send all packets to table 1 for L3
- Installs rules to send TCP SYN packets and ARP requests destined to VIPs

receive(IOFSwitch sw, OFMessage msg, FloodlightContext cntx):

- For TCP-SYN packets, selects a host and installs host specific rules
- For TCP non-SYN, sends a TCP reset (RST)
- For ARP requests : sends ARP reply to client with VIP MAC address
- Unhandled packets are pushed to next listener

Methods added (helper methods):

sendTcpReset(IOFSwitch sw, OFPacketIn pktIn, IPv4 ipv4, TCP tcp):

This method constructs and sends RST back to client,

- creates an ethernet frame with src and dest MAC address
- an IPv4 packet with the source and destination IP addresses swapped
- TCP packet with the source and destination ports swapped and the RST flag set.
- It then serializes the packet and sends it to the switch.

sendArpReply(IOFSwitch sw, OFPacketIn pktIn, ARP arp, LoadBalancerInstance instance):

- Constructs and send an ARP reply with Virtual MAC and IP addresses to the client.

getHostMACAddress(int hostIPAddress):

- Receives MAC Address associated with a given host IP address from device manager.

GitHub Link:

Github link for the code : https://github.com/sanampalsule/SDN-app

Deliverables:

The zip file is attached to this. Please go into the deliverables folder in the DCN Project, which contains the source code of part 3 and 4, as well as the executable jar file.

Conclusion

- This project report introduces an SDN (Software-Defined Networking) application designed to enhance network performance by integrating both shortest path routing and load balancing mechanisms.
- Within our application architecture, the ShortestPathSwitching module plays a pivotal role, leveraging Dijkstra's algorithm to compute and establish optimal routing paths between hosts. By dynamically analyzing network topology and traffic conditions, this module efficiently directs data packets along the shortest available paths, thereby minimizing latency and enhancing overall network throughput.
- Complementing the routing capabilities, the LoadBalancer module seamlessly manages incoming connections by transparently redirecting them to selected backend hosts. Through intelligent packet header rewriting techniques, this module optimizes resource utilization and ensures equitable distribution of traffic across the network, thereby preventing congestion and improving service delivery.
- Together, these components form a robust SDN solution that not only enhances network efficiency but also provides scalability and adaptability to evolving network requirements.

Other References

- Brown University Project Guide: https://cs.brown.edu/courses/cs168/f14/content/projects/sdn.pdf
- Accessing oracle VM Ubuntu using Putty: https://www.youtube.com/watch?v=9EuvlQ5XHeI
- GitHub Link: https://github.com/mattboran/DCN---Simple-OpenFlow-SDN-Controller