Name_ **Singh, Avantika**_________ Username:________________ Date:12/21/2020____
(last name, first name)
Section: ____________

Assignment 8: Final Project

______Part 3 functionality [Max 70 points]
______Part 4 functionality [Max 25 points]
______Style [Max 5 points]
______Part 3 extra credit questions [Max 20 points (10 each)]

_____Total [Max 100 points + Extra Credit points]

Total in points: ____________________

Total in extra credit points: ____________________

Professor's Comments:

Affirmation of my Independent Effort: _______Avantika Singh_________
(Sign here)

ACKNOWLEDGEMENT

This project couldn't be completed without the advice and support of Professor Franchitti. I would also like to extend my gratitude to the TA Joanna for her guidance during the project and helping us whenever we got stuck on this project.
I am also grateful to University of Wisconsin and at Brown University who developed the part of code and instructions for this project.

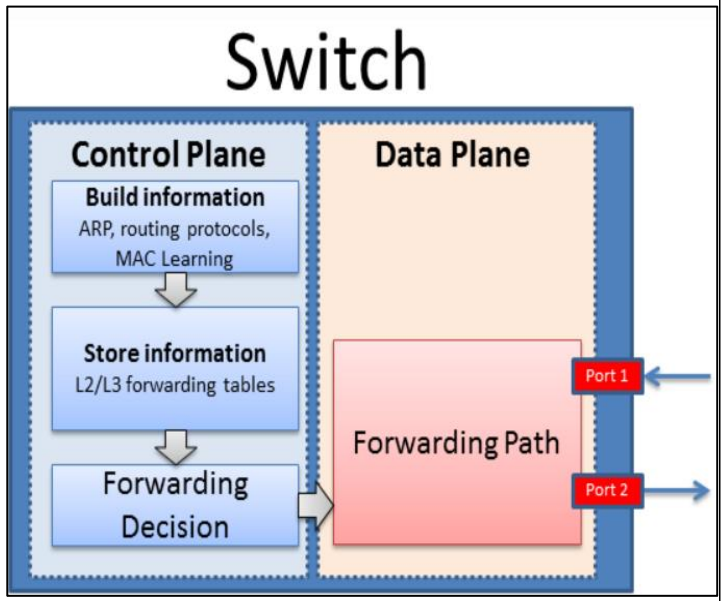
Thanking You,
Avantika Singh

Background:

Traditional networks use both a distributed data plane and a distributed control plane. In other words, each device has a data plane and a control plane, and the network distributes those functions into each individual device. Traditional networking is rooted in fixed-function network devices, such as a switch or router which are somewhat autonomous in their working. These devices each have certain functions that operate well together and support the network. For example : The switches/routers when connected to any topology in a computer network exchange certain messages which enables them to pass the information about their neighbors and the resulting topology to other nodes. Thus, every node has some idea (or complete picture of topology in case of link state algorithms) of the topology in which they are located.

Each router consists of two functioning planes: **Control Plane and Data Plane.**

1. **Control Plane:** Control plane contains the protocols and mechanisms that enable routers to efficiently learn how to forward packets towards their destination. It is used to determine how a packet is routed among routers along the path from source to destination. It is the part of router architecture that is concerned with drawing the network topological information using the information in routing tables that defines what to do with the incoming packets. Routing protocols specify how routers gather information for the routing tables and talk to their neighboring routers, disseminating information that enables them to select any two nodes on a computer network.



2. **Data Plane:** Data plane contains the protocols and mechanisms that allow hosts and routers to exchange packets carrying user data. Data plane is used for forwarding packets from one interface to another via forwarding table and decides what to do with packets arriving on an inbound interface. Mostly, it refers to a table in which the router looks up the destination address of the incoming packet and retrieves the information necessary to determine the path from the receiving element, through the internal forwarding fabric of the router, and to the proper outgoing interface(s).

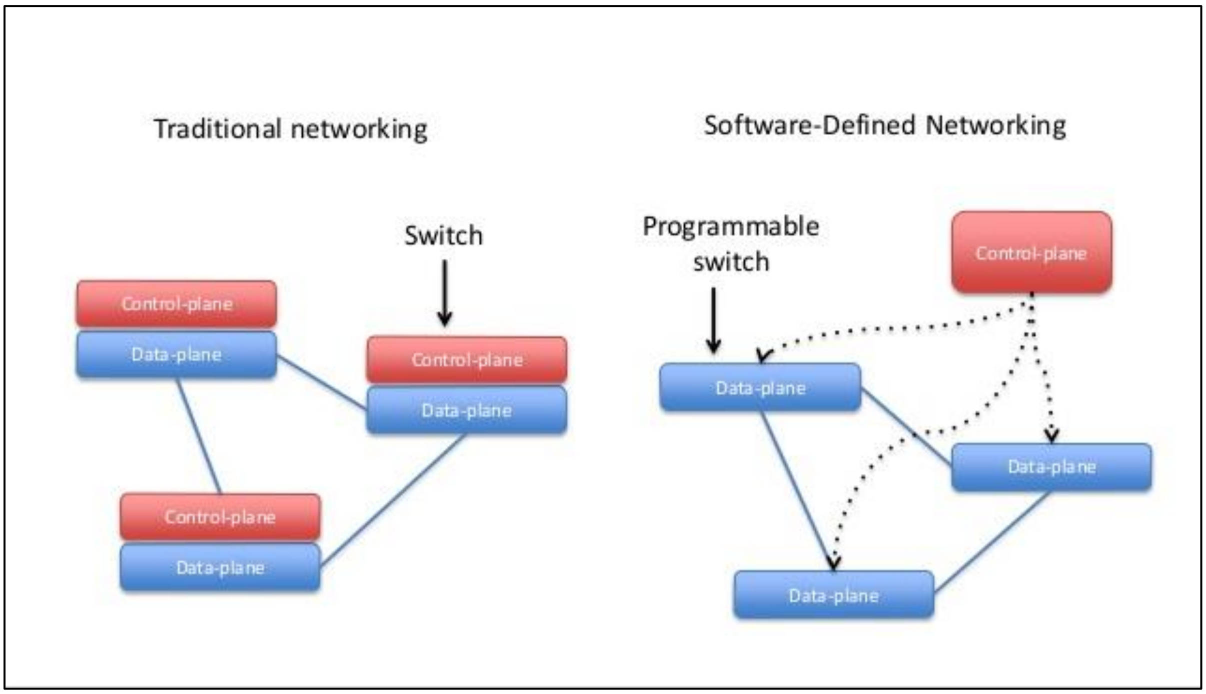
These two parts works in combination to define the complete working of a router.

But with such complexity, this approach draws certain drawbacks which are as follows:

- Flexibility is a recurring hurdle for traditional networks. Few APIs are exposed for provisioning and most switching hardware and software is proprietary. Traditional networks often work well with proprietary provisioning software, but this software can't be quickly modified as needed.

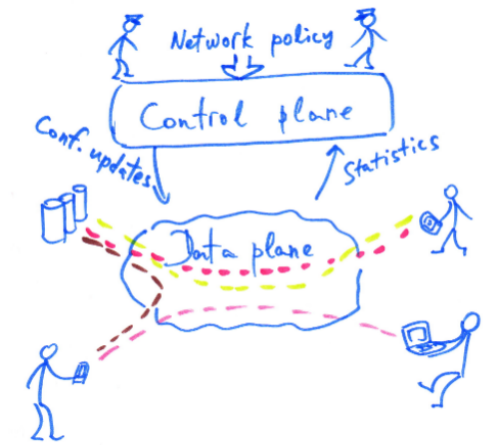
- Since the routers need to setup the topology when they are connected (determine neighboring nodes, spanning tree construction etc.), they need significant processing capability to do this which in return results in expensive routers/switches.
- With traditional networking significant amount of setup time is required to identify loop free topology and the neighboring nodes.
- With a traditional network the physical location of the control plane hinders an IT administrator's ability to control the traffic flow.

It is with these shortcomings; concept of Software Defined Networks was introduced.



Introduction:

Software-Defined Networking(SDN) is a relatively new network architecture in which the control plane is separated from each individual network device and instead implemented in an external software entity. The external entity has complete knowledge of the topology of a network under its control, and programs the forwarding tables of each individual device in the network. This architecture decouples the network control and forwarding functions enabling the network control to become directly programmable and the underlying infrastructure to be abstracted for applications and network services. Software-Defined Networking is an emerging architecture that is manageable, cost-effective, and adaptable, making it ideal for the high-bandwidth, dynamic nature of today's applications.



A view of an SDN system: the *control plane* accepts network policy updates from network operators and install them on the *data plane* so that the concurrent user traffic is not affected during the transition.

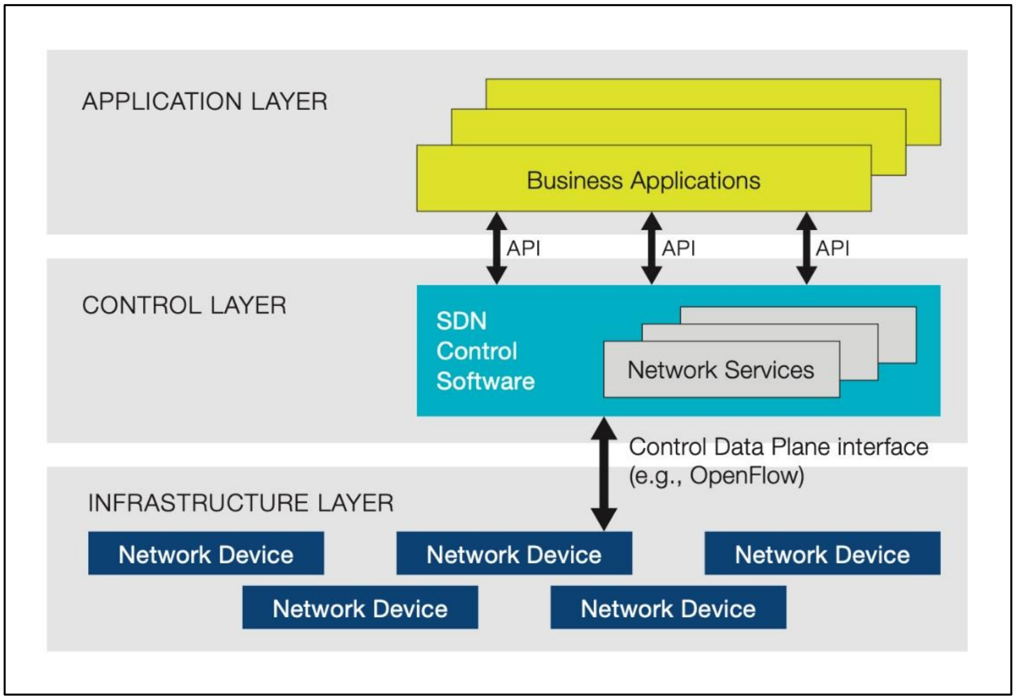
Distinct properties of SDN Architecture:

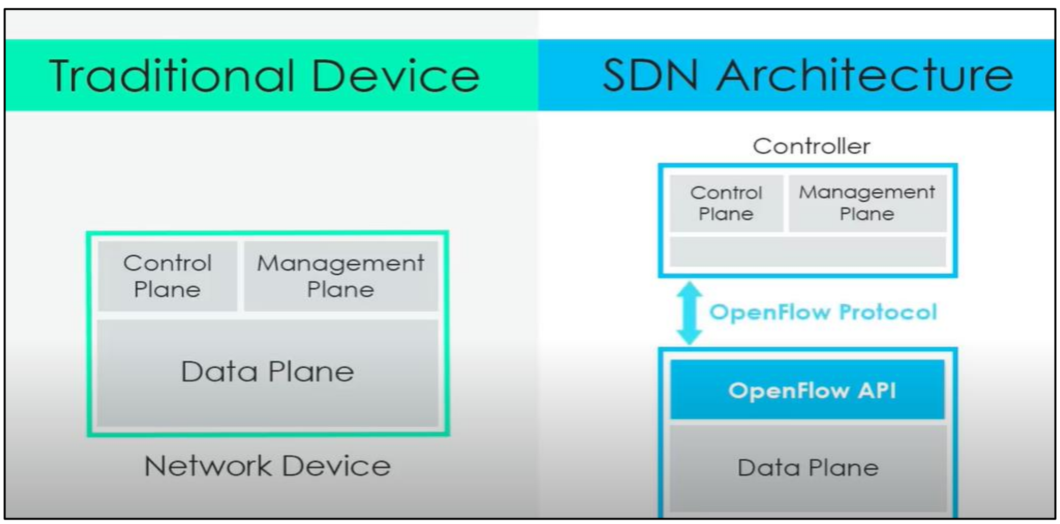
1. **Control and data plane separation:** Removing the control plane from network devices and implementing it in an external SDN controller significantly reduces the complexity of network devices, making them simpler and cheaper than CN devices whose distributed control plane functionality is implemented across millions of lines of code, defined across hundreds of RFCs.
2. **Network programmability**: An SDN controller, with complete knowledge of a network's topology, controls a multitude of network devices within its administrative domain. By providing application programming interfaces (APIs), SDN makes it possible to develop networking applications, e.g., traffic engineering, thus enabling network innovation. In contrast, traditional networking devices are proprietary and closed, making it hard or impossible to develop innovative network applications

Advantages of SDN:

- Control plane and Data plane technologies can evolve individually without one restricting the growth of other.
- It is possible to centralize the control of the network which offers many advantages including better support for traffic engineering, maintaining a consistent network-wide policy implementation, security.
- Virtualization epitomizes the primary difference between SDN and traditional networking. When SDN virtualizes your entire network, it generates an abstract copy of your physical network, and lets you provision resources from a centralized location.
- SDN lets users use software to provision new devices instead of using physical infrastructure, so IT administrators can direct network paths and proactively arrange network services. Unlike traditional switches, SDN can communicate better with devices using the network.

- The use of SDN helps revive older network devices and simplifies the process of optimizing commoditized hardware. By following the instructions from the SDN controller, older hardware can be repurposed as this process allows new devices to become “white box” switches that have intelligence focused at the SDN controller.
- Centralized network provisioning. SDN’s helps centralize enterprise management and provisioning by offering a unified perspective on the whole network. SDN can also speed up service delivery and boost agility in provisioning virtual and physical network devices in a central location.





Project Overview:

SDN switches and routers do not run control plane protocols and mostly only forward packets based on matching of packet predicates to a set of forwarding rules. They export a simple API to configure these rules, as well as some feedback about current and past packets. The currently accepted standard for this API is the **OpenFlow protocol**, which has been implemented by dozens of switch vendors and has fostered a rich software ecosystem. The intelligence of the control plane is (logically) centralized in a **network controller.** The controller decides which rules to install based on its configuration, and on a global view of the network topology and flows.

For this project we will implement two control application for a software defined network (SDN).

- A layer-3 routing application will install rules in SDN switches to forward traffic to hosts using the shortest, valid path through the network. The application logic will manage the efficient switching of packets among hosts in a large LAN with multiple switches and potential loops. We create an SDN controller application that will compute and install shortest path routes among all the hosts in your network.
- A distributed load balancer application will redirect new TCP connections to hosts in a round-robin order.

Implementation:

1. **Layer-3 routing application**

- The code for the layer-3 routing application resides in L3Routing.java in the edu.wisc.cs.sdn.apps.l3routing package.
- Bellman-Ford algorithm was used to compute the shortest paths to reach a host h from every other host h' ∈ H, h ≠ h' (H is the set of all hosts).
- There are two link objects between pairs of switches, one in each direction. Due to the way links are discovered, there may be a short period of time (tens of milliseconds) where the controller has a link object only in one direction.
- When a host joins the network, both the deviceAdded(...) and linkDiscoveryUpdate(...) event handlers will be called. There are no guarantees on which order these event handlers are called. Thus, a host may be added but we may not yet know which switch it is linked to.
- The isAttachedToSwitch() method in the Host class will return true if we know the switch to which a host is connected, otherwise it will return false.
- The following is assumed to hold true in the network:

  The network is a connected graph. In other words, there will always be at least one possible path between every pair of switches.

  There is only one physical link between a pair of switches. Links are undirected.

```
function BellmanFord(list vertices, list edges, vertex source)
   ::distance[],predecessor[]

   // This implementation takes a graph as input
   // A graph is represented as a list of vertices and edges
   // The algorithm fills two arrays
   // (distance and predecessor) with shortest-path
   // (less cost/distance/metric) information

   // Step 1: initialize graph
   for each vertex v in vertices:
       distance[v] := inf            // At the beginning: All vertices have a weight of infinity
       predecessor[v] := null        // And a null predecessor

   distance[source] := 0             // Inital distance for the source is NULL
   // Step 2: Relax edges repeatedly
   for i from 1 to size(vertices)-1:
       for each edge (u, v) with weight w in edges:
           if distance[u] + w < distance[v]:
               distance[v] := distance[u] + w
               predecessor[v] := u
   // Step 3: Check for negative-weight cycles
   for each edge (u, v) with weight w in edges:
       if distance[u] + w < distance[v]:
           error "Graph contains a negative-weight cycle"
   return distance[], predecessor[]
```

The Bellman–Ford algorithm is an algorithm that computes shortest paths from a single source vertex to all of the other vertices in a weighted digraph. Bellman–Ford is based on the principle of relaxation, in which an approximation to the correct distance is gradually replaced by more accurate values until eventually reaching the optimum solution. We implement Bellman-Ford using a queue to define which edges to explore next

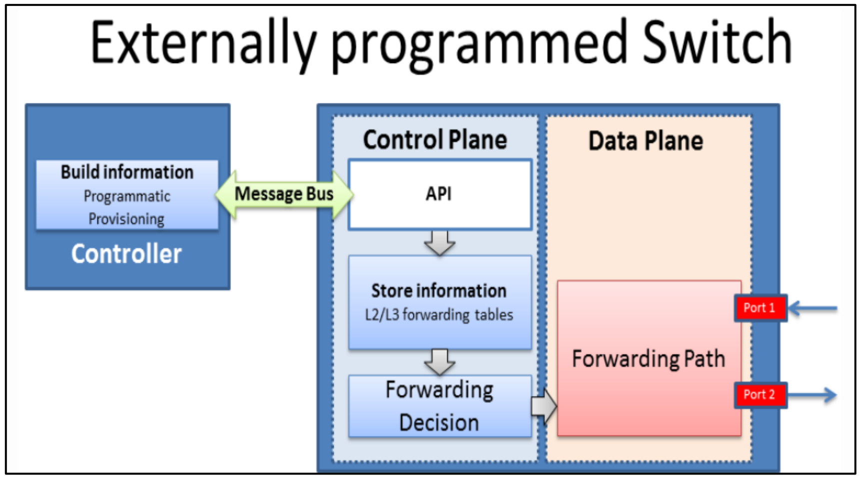
2. Load balancer:

Networks employ load balancing to distribute client requests among a collection of hosts running a specific service (e.g., a web server). A hardware load balancer is placed in the network and configured with an IP address and a set of hosts among which it should distribute requests. Clients wanting to communicate with a service (e.g., a web server) running on those hosts are provided with the IP address of the load balancer, not the IP address of a specific host. Clients initiate a TCP connection to the IP address of the load balancer (10.0.100.1) and the TCP port associated with the service. For each new TCP connection, the load balancer selects one of the specified hosts (usually in round robin order). The load balancer maintains a mapping of active connections—identified by the client's IP and TCP port—to the assigned hosts.

- The code for the load balancer application resides in LoadBalancer.java in the edu.wisc.cs.sdn.apps.loadbalancer package.
- The LoadBalancerInstance class represents a single distributed load balancer.
- Each load balancer instance has a virtual IP address, virtual MAC address, and set of hosts among which TCP connections should be distributed.
- The instances class variable in the LoadBalancer class maps a virtual IP address to a specific load balancer instance.

Execution Environment:

We run the code for the project in an emulated network inside of a single Linux VM. We use the Mininet network emulator, which is designed to emulate arbitrary topologies of emulated OpenFlow switches and Linux hosts. It uses container-based virtualization for very light-weight emulated nodes.

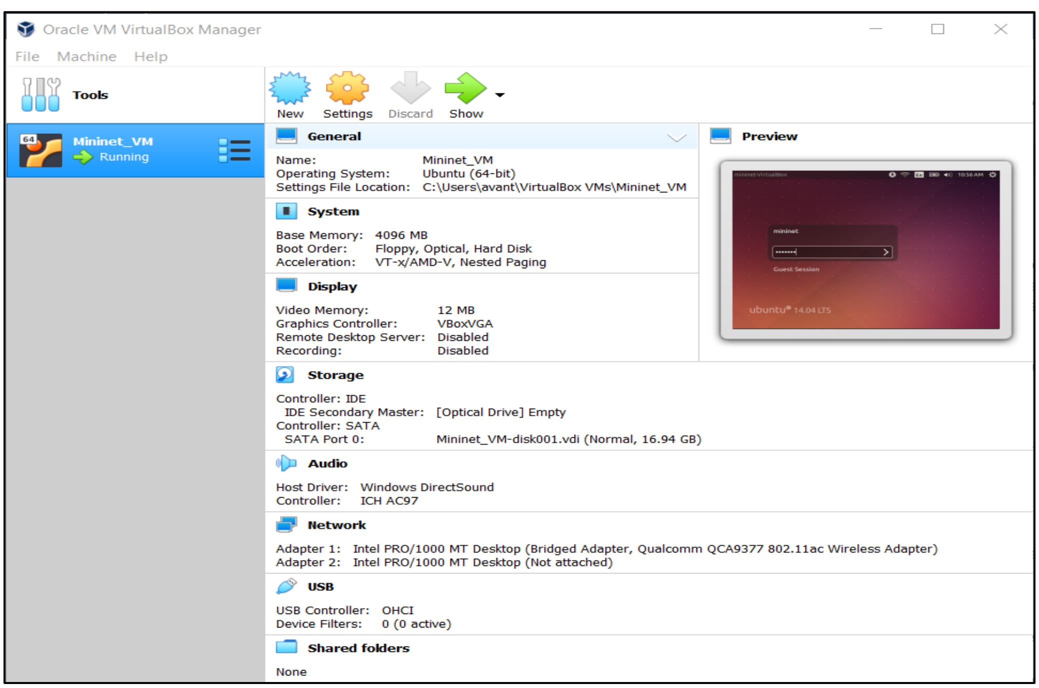


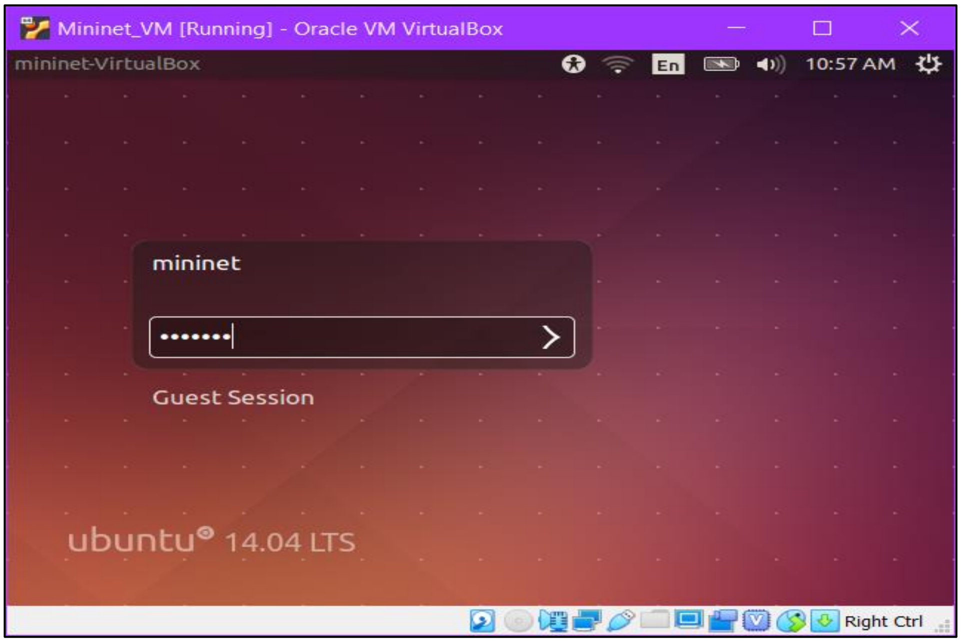
The switches in our network run the open source OpenvSwitch switch software, which implements the Openflow protocol. The switches connect to an Openflow network controller. We will use Floodlight, a relatively mature Java-based controller. We will use OpenFlow version 1.0 for this project. The SDN applications are written in Java and run atop the Floodlight OpenFlow controller.

Environment Setup :

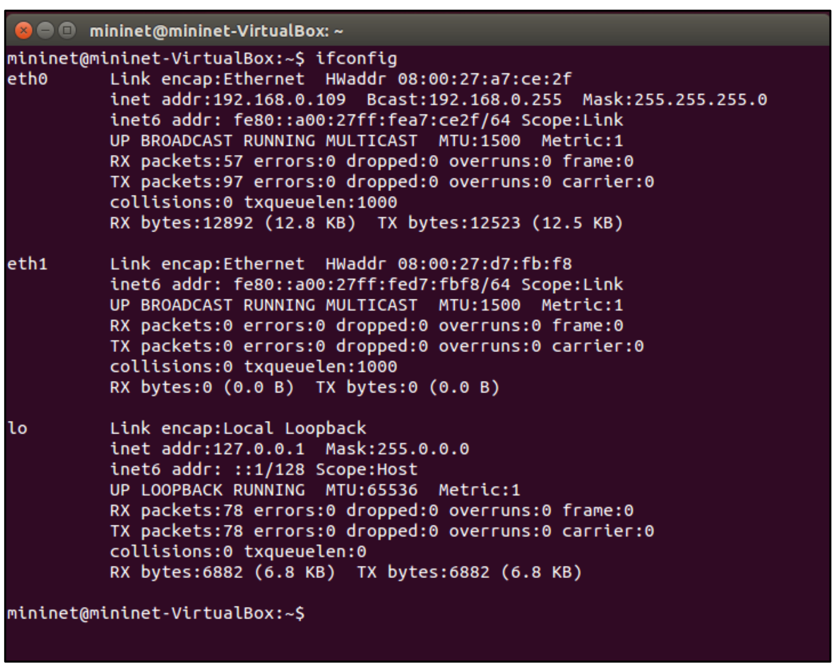
The steps followed have been listed below:

1. Install the Oracle VirtualBox and download the Virtual Box Image with the necessary software required for development. To install the .ova file go to File and Import Appliance on VirtualBox. This VM uses “mininet” as username and password.



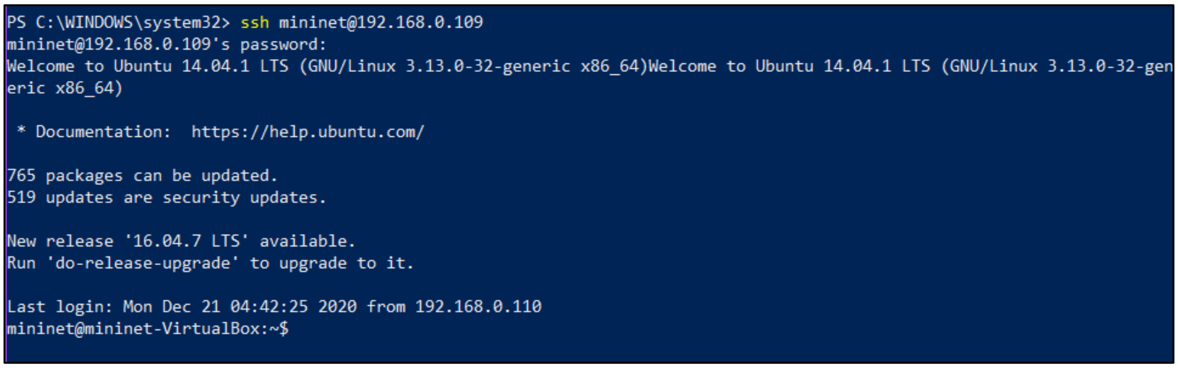


2. To ssh into the VM from your host computer we first log in first using the GUI, open a terminal, and type ifconfig. This shows us the IP addresses of the VM. We can now connect to the virtual machine from our host computer via ssh.



```
mininet@mininet-VirtualBox: ~
mininet@mininet-VirtualBox:~$ ifconfig
eth0      Link encap:Ethernet  HWaddr 08:00:27:a7:ce:2f
          inet addr:192.168.0.109  Bcast:192.168.0.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fea7:ce2f/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:57 errors:0 dropped:0 overruns:0 frame:0
          TX packets:97 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:12892 (12.8 KB)  TX bytes:12523 (12.5 KB)

eth1      Link encap:Ethernet  HWaddr 08:00:27:d7:fb:f8
          inet6 addr: fe80::a00:27ff:fed7:fbf8/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:78 errors:0 dropped:0 overruns:0 frame:0
          TX packets:78 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:6882 (6.8 KB)  TX bytes:6882 (6.8 KB)

mininet@mininet-VirtualBox:~$
```

We can now connect to the virtual machine from our host computer via ssh.
ssh mininet@192.168.0.109



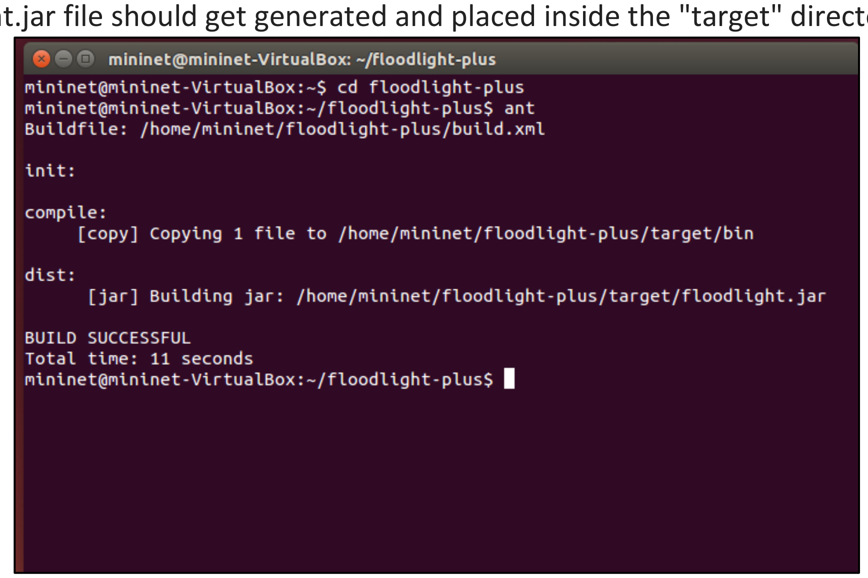
```
PS C:\WINDOWS\system32> ssh mininet@192.168.0.109
mininet@192.168.0.109's password:
Welcome to Ubuntu 14.04.1 LTS (GNU/Linux 3.13.0-32-generic x86_64)Welcome to Ubuntu 14.04.1 LTS (GNU/Linux 3.13.0-32-gen
eric x86_64)

 * Documentation:  https://help.ubuntu.com/

765 packages can be updated.
519 updates are security updates.

New release '16.04.7 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

Last login: Mon Dec 21 04:42:25 2020 from 192.168.0.110
mininet@mininet-VirtualBox:~$
```

3. We will additionally be using Putty and WinSCP to connect to the Virtual machine and transfer files.

4. **Refactor the code to resolve the any module loading errors.**
   In order to resolve any issues during execution we need to refactor the code:
   - Navigate to the openFLow directory and change the directory path from src/brown/cs/sdn/apps/ to /src/wics/cs/sdn/apps/
   - For all files in sub-directories of /src/wics/cs/sdn/apps/ change the package names to edu.wisc.cs.sdn.apps..
   - Navigate to the floodlight-plus directory in /src/main/resources/META-INF/services/ and open the file called net.floodlightcontroller.core.module.IFloodlightModule and change the following from: edu.brown.cs.sdn.apps.sps.ShortestPathSwitching and edu.brown.cs.sdn.apps.util.ArpServer to: edu.wisc.cs.sdn.apps.sps.Shortest PathSwitching and edu.wisc.cs.sdn.apps.util.ArpServer
   - Add edu.wisc.cs.sdn.apps.l3routing.L3Routing to the net.floodlightcontroller.core.module.IFloodlightModule

Steps for Execution:

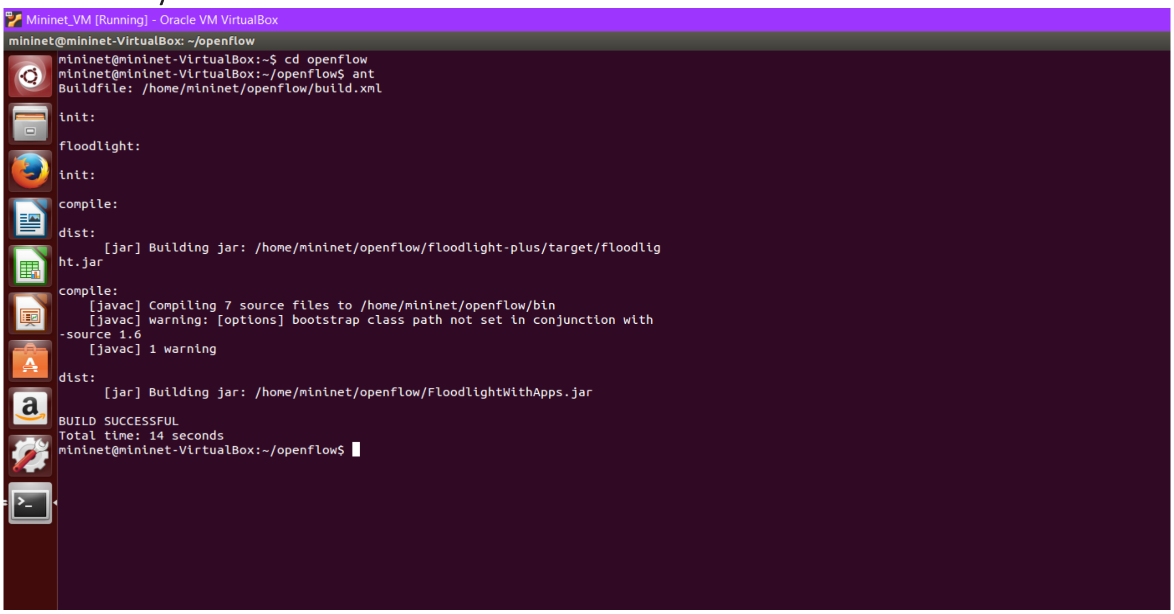
1. Navigate to the floodlight-plus directory and run "ant" to compile the code. The floodlight.jar file should get generated and placed inside the "target" directory.



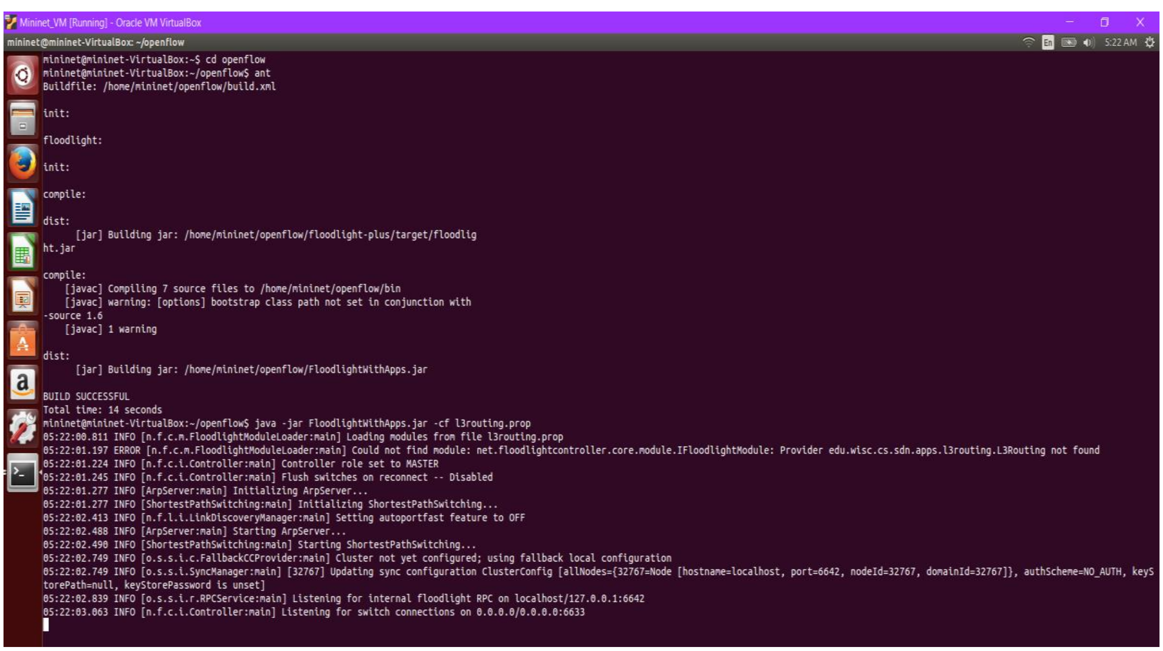
```
mininet@mininet-VirtualBox: ~/floodlight-plus
mininet@mininet-VirtualBox:~$ cd floodlight-plus
mininet@mininet-VirtualBox:~/floodlight-plus$ ant
Buildfile: /home/mininet/floodlight-plus/build.xml

init:

compile:
     [copy] Copying 1 file to /home/mininet/floodlight-plus/target/bin

dist:
      [jar] Building jar: /home/mininet/floodlight-plus/target/floodlight.jar

BUILD SUCCESSFUL
Total time: 11 seconds
mininet@mininet-VirtualBox:~/floodlight-plus$
```

2. Navigate to the openflow directory and open the build.xml file. Additionally, make sure the relative paths to the floodlight-plus/target directory are properly configured to the correct relative paths.

3. Run "ant" in the root level of the openflow directory. In the bin directory, you should see the shortestPathSwitching.class file with the new package and directory structure. The FloodlightWithApps.jar file gets generated in this openflow root directory.

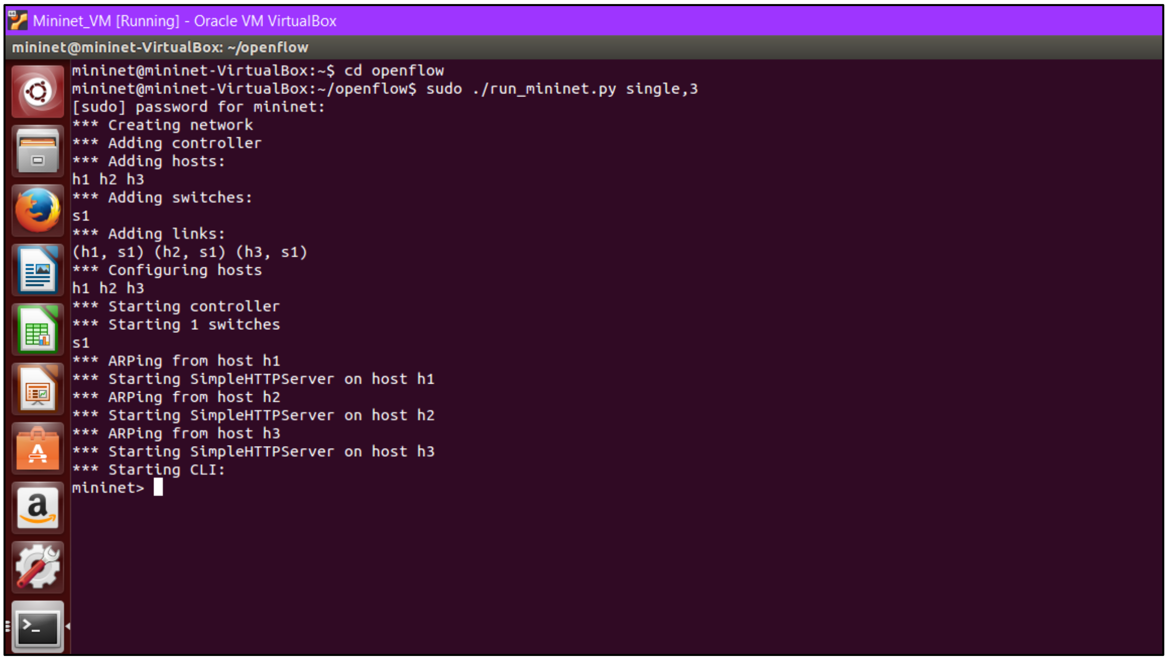


4. Start Floodlight and your SDN applications.Run the program with the command: **java -jar FloodlightWithApps.jar -cf l3routing.prop**
The above command will start Floodlight and only our layer-3 routing application. The .prop file configures our application.
Output we get once Floodlight starts :

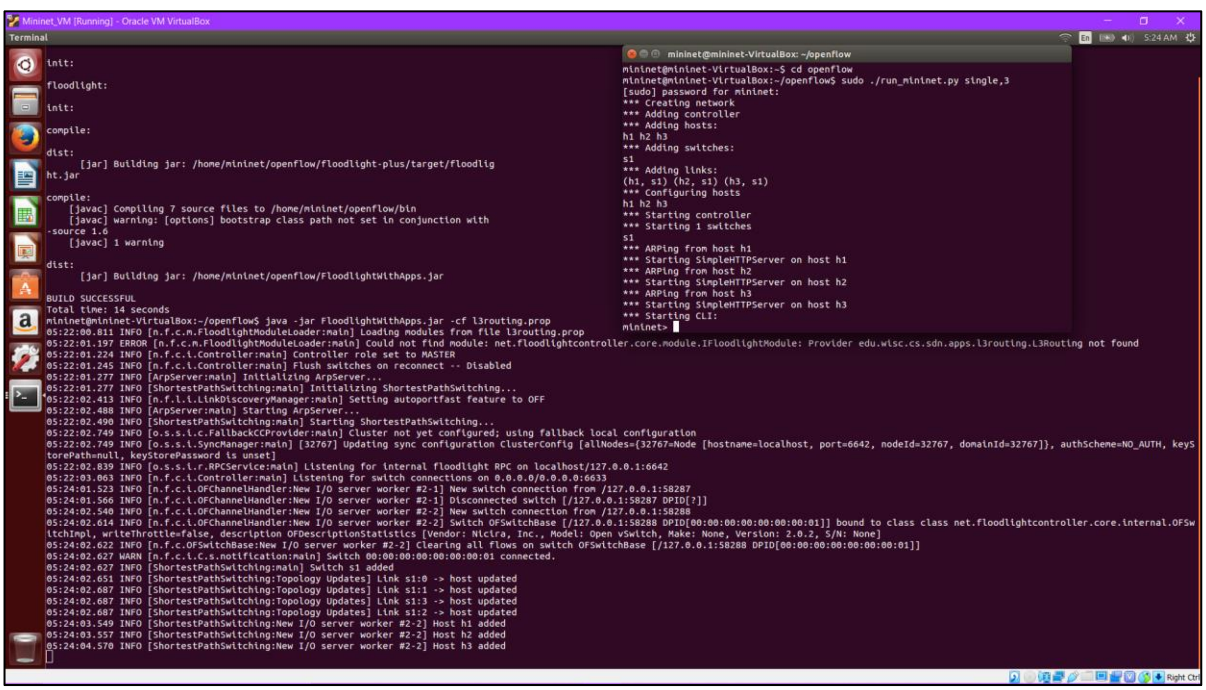


Keep the terminal with Floodlight open, as we will need to see the output for debugging.

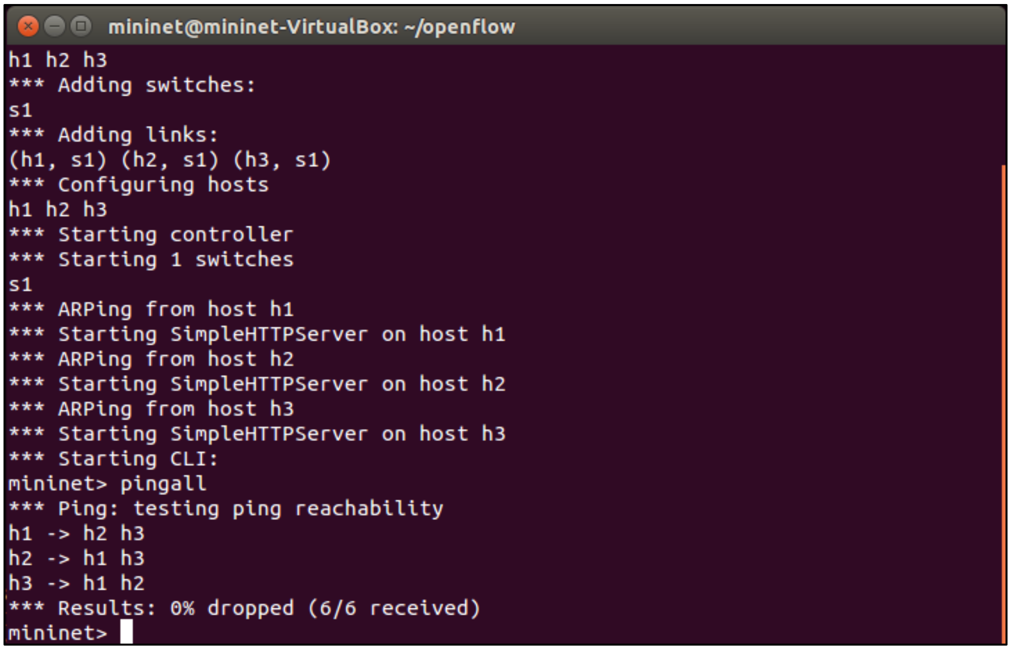
5. Start Mininet: Mininet is a system for rapidly prototyping large networks on the constrained resources of a single laptop. The lightweight approach of using OS-level virtualization features, including processes and network namespaces, allows it to scale to hundreds of nodes.
   $ sudo ./run_mininet.py single,3
   The above command will create a topology with a single SDN switch (s1) and three hosts (h1 - h3) directly connected to the switch.

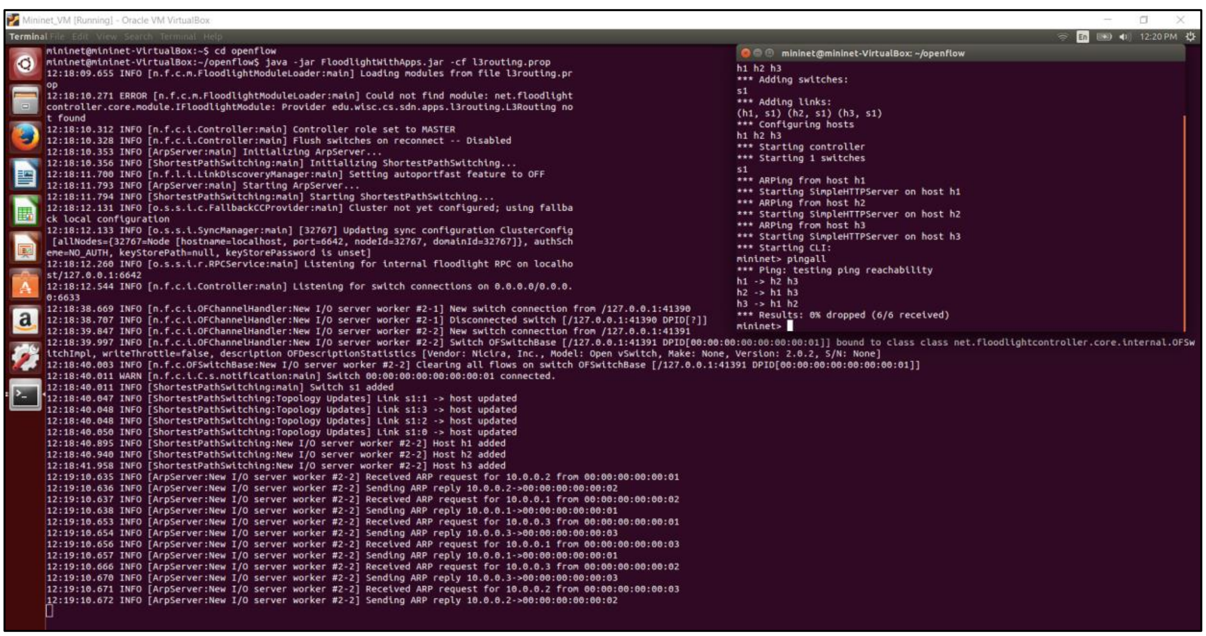


We can see Floodlight produce the following output :

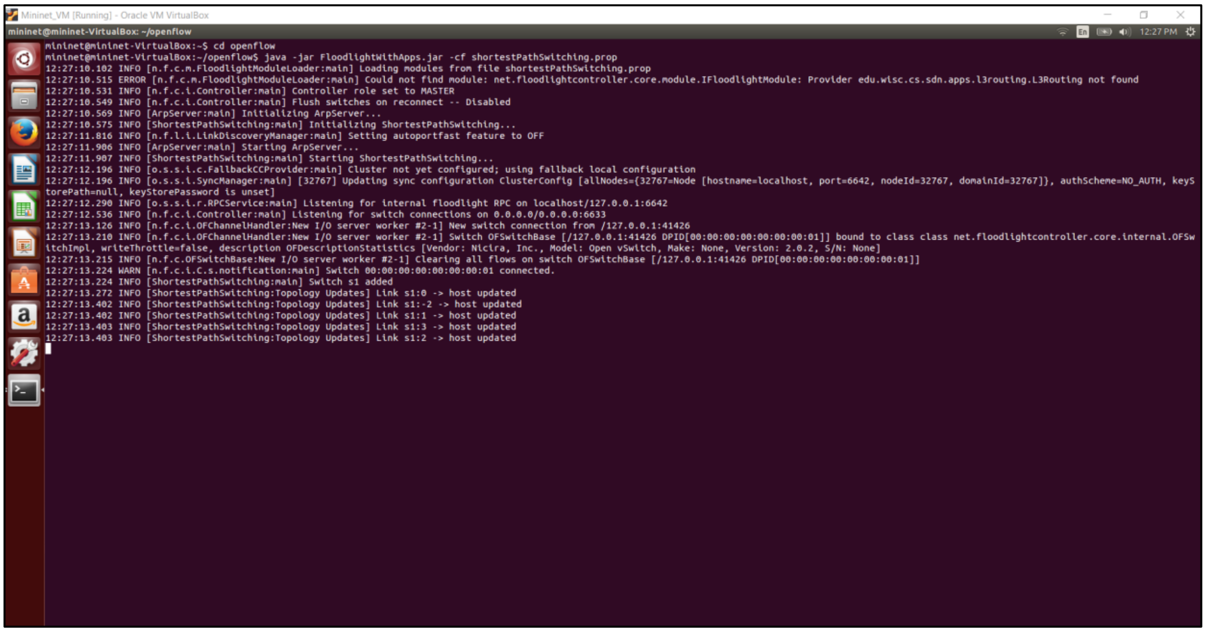


Debugging: Now for debugging we run commands (e.g., ping) in Mininet.

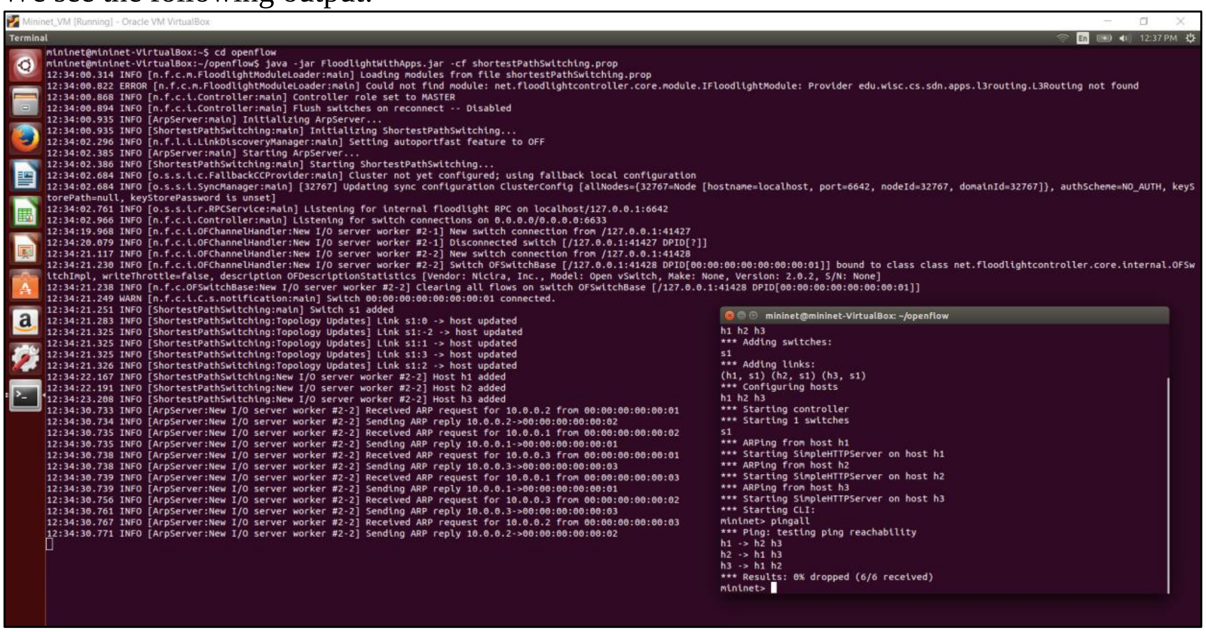




6. Run the command to execute the Layer 3 Routing Application.
   java -jar FloodlightWithApps.jar -cf shortestPathSwitch.prop



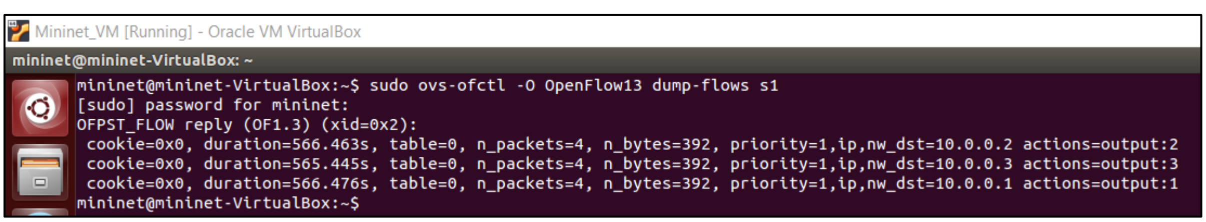
We see the following output:



Debugging :

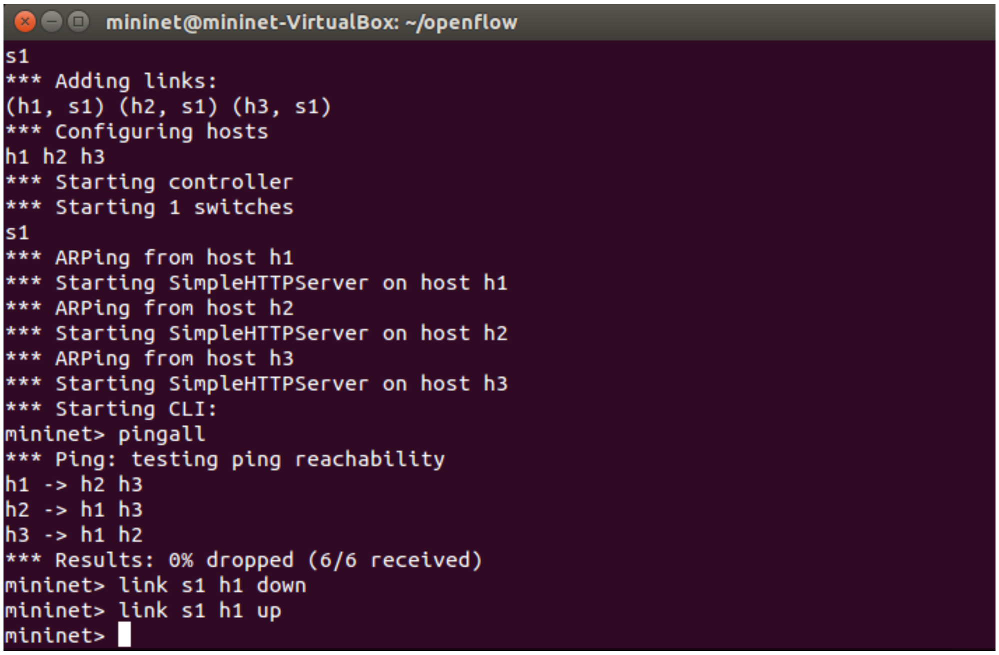
To debug, we view the contents of an SDN switches flow tables by running the following command in your mininet VM.

```
$ sudo ovs-ofctl -O OpenFlow13 dump-flows s1
```

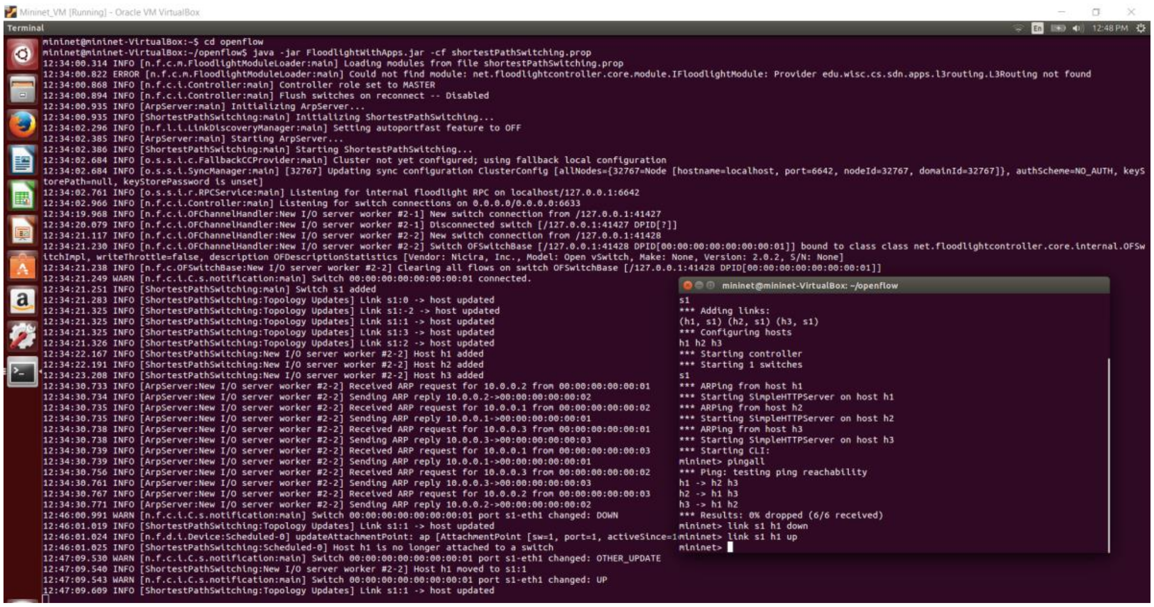


To debug further we trigger Event Handlers: We trigger the `linkDiscoveryUpdate(...)` event handler by running the following commands in Mininet

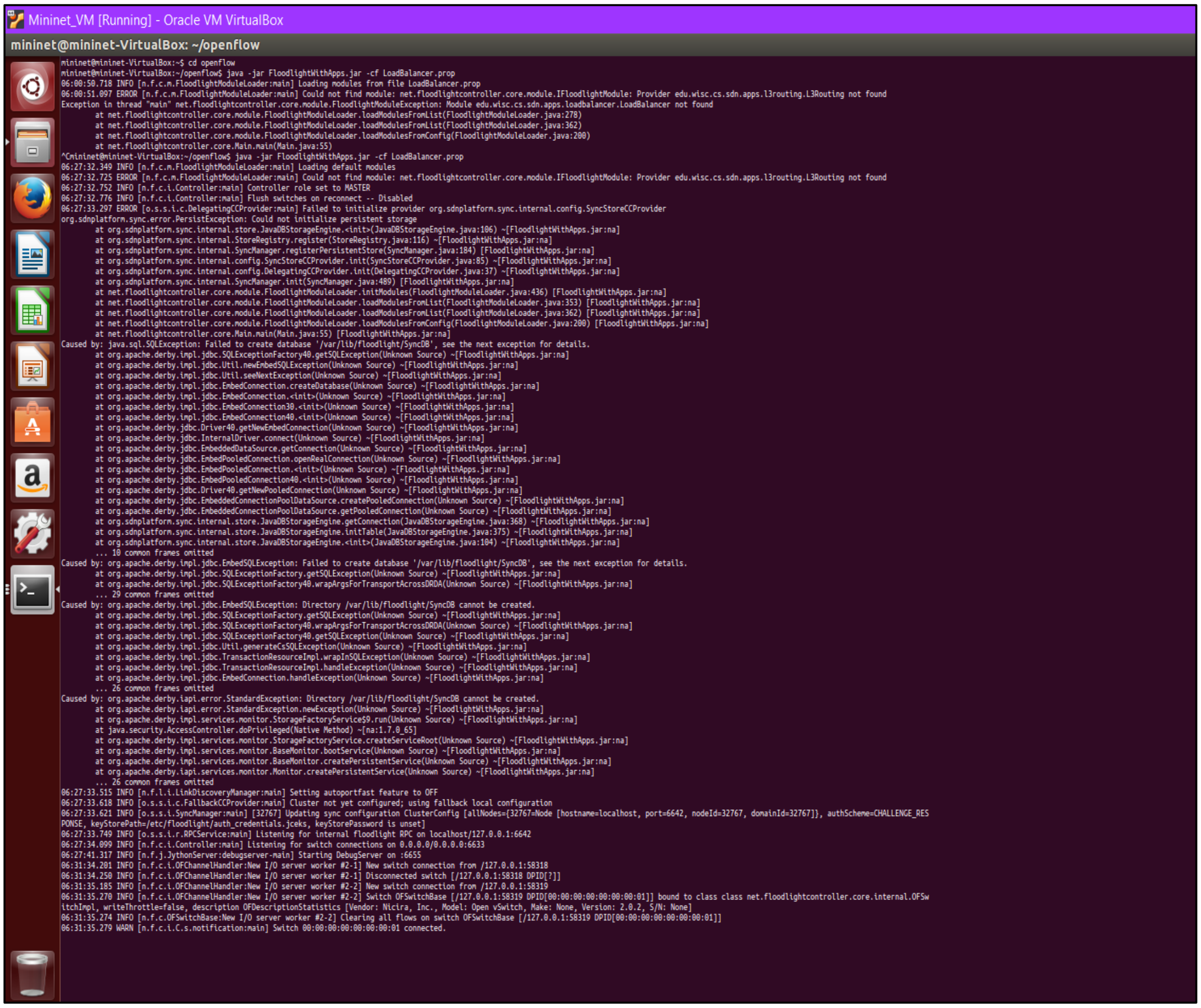
- `link s1 h1 down` — takes down the link between `s1` and `h1`; this will also result in a `deviceRemoved(...)` event and the `isAttachedToSwitch()` method for the Host object for `h1` will now return false
- `link s1 h1 up` — brings up the link between `s1` and `h1`; this will also result in a `deviceMoved(...)` event and the `isAttachedToSwitch()` method for the Host object for `h1` will now return true

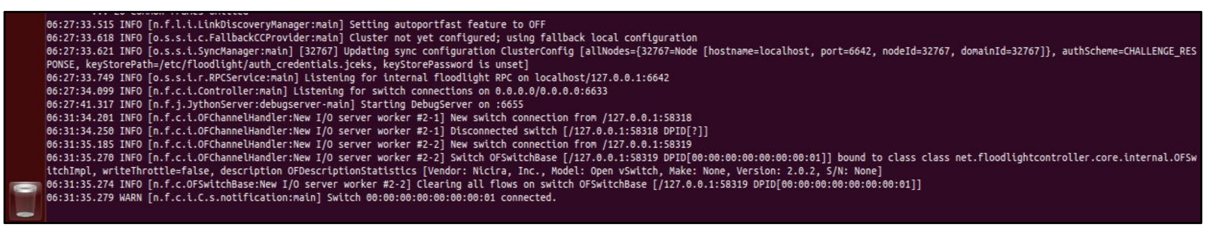


We see the following output :



7. Run the command to execute the loadbalancer application.
   java -jar FloodlightWithApps.jar -cf loadbalancer.prop





Results and Conclusion:

For this project, I developed a Software Defined Networking layer-3 routing application that constructs routing tables based on a global view of the network topology. The routing table is installed in each SDN switch by the application, and each SDN switch forwards packets according to the installed route table. The application installs routing table entries that match packets based on their destination IP address and execute an output action to send the packet out a specific port on the SDN switch. Traffic is forwarded to the intended host using the shortest path. We use the Bellman-Ford algorithm to compute the shortest paths to reach a host from every other host. We implement Bellman-Ford using a queue to define which edges to explore next

We also develop a second SDN application. We implement a load balancer. It is provided with a list of virtual IPs and a set of hosts among which connections to the virtual IPs should be load balanced. When the clients initiate TCP connections with a specific virtual IP, the SDN switches will send the TCP SYN packet to the SDN controller. Our SDN application selects a host from a pre-defined set and install rules in an SDN switch to rewrite the IP and MAC addresses of packets associated with the connection.

The assignment gave us a chance to implement a software defined network(SDN's). Some of the challenges that I faced during the implementation were in setting up the environment and refactoring the source code to resolve module related errors. I had to understand and gather some domain related knowledge(regarding Mininet, Openflow etc.) to work on the problem. After this assignment I feel more comfortable in working in virtualized environments(VM's). The key takeaway from the project is the advantages SDN's bring to the networking domain and the significant improvement they offer over traditional networking.

References and Related works:

1. Mininet network emulator documentation (http://mininet.org)
2. Openflow documentation (https://www.opennetworking.org/sdn-resources/onf-specifications/openflow)
3. https://www.opennetworking.org/sdn-resources/sdndefinition
4. Floodlight Java-based SDN controller documentation (https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/overview)
5. http://pages.cs.wisc.edu/~agember/cs640/s15/assign5/
6. https://www.ibm.com/services/network/sdn-versus-traditional-networking