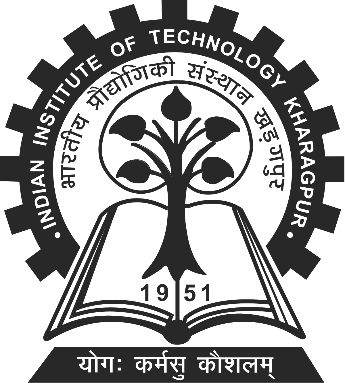
**Design Laboratory Report (CS59001)**

Indian Institute of Technology, Kharagpur

Department of Computer Science and Engineering



**Name**: Harsh Pritam Sanapala

**Roll number:** 17CS30016

**Mentor:** Prof. Arobinda Gupta

**INTRODUCTION**

**Software Defined Networking**

Software-defined networking enables dynamic network configuration to improve network performance. It also facilitates better monitoring of networks with minimal manual interference. Typical or traditional network architectures are decentralized and are very complicated. Software-Defined Networking provides flexibility and makes the process of troubleshooting simpler. It utilizes a central controller (or a set of distributed controllers to handle fault tolerance). This controller has the intelligence to handle many intricate details without needing manual interference all the time.

Software-Defined Networking was commonly associated with the OpenFlow protocol (for remote communication with network plane elements for the purpose of determining the path of network packets across network switches). OpenFlow enables control over the forwarding plane of switches and routers on the network. OpenFlow enables network controllers to determine the path of network packets across a network of switches. The controllers are distinct from the network switches. This separation of the control from the forwarding allows for more sophisticated traffic management when compared to using access control lists (ACLs) and routing protocols.

The Open Networking Foundation manages the OpenFlow standard. It is a user-led organization dedicated to the promotion and adoption of software-defined networking. Open Networking Foundation regards OpenFlow as "the first standard communications interface defined between the control and forwarding layers of an SDN architecture". OpenFlow allows direct access to and manipulation of the forwarding plane of network devices such as switches and routers, both physical and virtual (hypervisor-based). There is an absence of an open interface to the forwarding plane. Thus it led to the characterization of today's networking devices as monolithic, closed, and mainframe-like. A protocol like OpenFlow is needed to move network control out of proprietary network switches and into control software that's open-source and locally managed

Some of the disadvantages of Software-Defined Networking are that it relies on a centralized system which makes the whole system vulnerable due to security, scalability, and elasticity concerns. Software-Defined Networking is also quite expensive because switches need to be Software-Defined Networking enabled in order to run the protocol or policies in a centralized manner. Many companies use OpenFlow to develop their own proprietary techniques to handle specific needs.

**Mininet**

Mininet is a network emulator designed to develop and experiment with Software-Defined Networking systems using OpenFlow. Mininet creates a realistic virtual network, running a kernel, switch and application code on a single machine (VM, cloud or native). Mininet has easy to interact CLI tools and API provided to create custom topologies. There are some issues to be addressed before working on projects using Mininet. But once the setup is complete, it would be very effective and easy to use.

**AIM**

In this work, an attempt to understand Mininet and Software-Defined Networking has been made. Mininet has numerous CLI tools built-in, and most of them have been used in one way or the other to understand the basic idea of how to design networks.

**STEP BY STEP WORKFLOW**

**Setup**

Setting up the environment to experiment with Mininet was the most challenging part. Firstly, it comes in a pre-packaged Mininet/Ubuntu VM for VirtualBox. This VM includes Mininet itself, all OpenFlow binaries and tools pre-installed, and tweaks to the kernel configuration to support large Mininet networks. Documentation is present in detail for installing VM and starting a Mininet session. The main issue faced here was regarding X11 forwarding. To utilize various GUI features of the VM itself, we have to enable X11 forwarding and install a compatible X11 server. In the documentation, recommended X11 server was Xming, but it was not working on windows 10. I found out about VcXsrv, which worked as an alternative to Xming. Newer Linux distros "interfaces" file is replaced by netplan config file. So had to edit it to add interfaces(for enabling DHCP instead of static IP for an eth interface, add eth and set DHCP to yes). While using PuTTY to ssh into the VM, xterm windows were not working properly because of X11 forwarding being a bit clumsy within PuTTY. We have to set up everything in place so that the display on the host system is able to project the VM and PuTTY session's GUI load. Once this obstacle is cleared, I was able to interact with Mininet CLI tools.

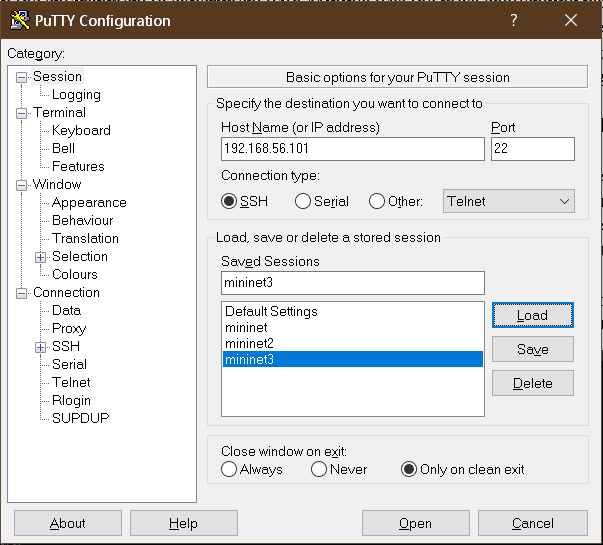


Figure PuTTY Config

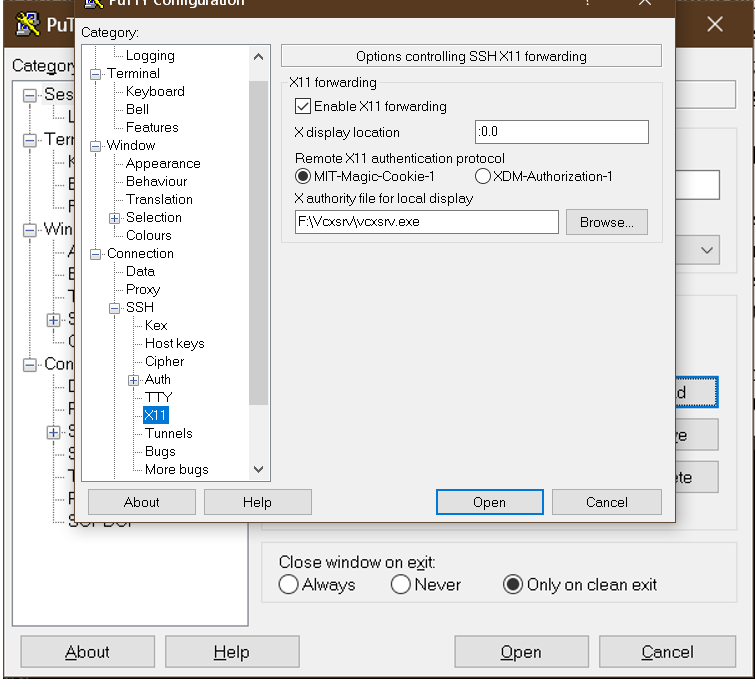


Figure PuTTY X11 Configuration

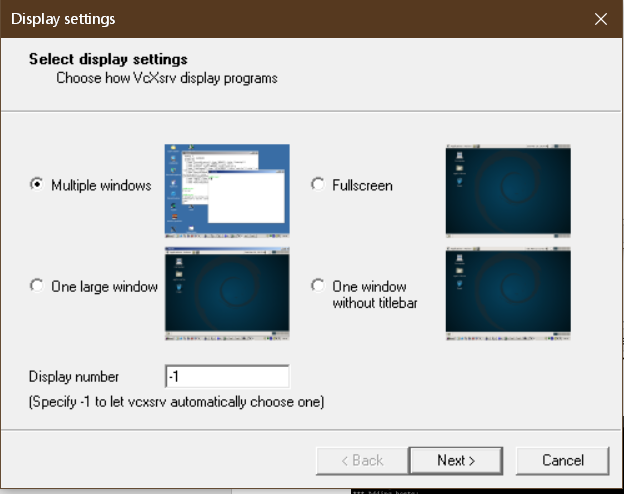


Figure X11 Server Config- VcXsrv

**Mininet Topologies**

Once we are done with the setup, I was able to use Mininet CLI to start a basic network topology which consisted of 2 hosts and one switch and a centralized controller. I used the topology parameter of the Mininet start up command to play with some of the topologies provided to us. Generated the default topology with one switch and two hosts and ran a HTTP Server on one and communicated with the other. Generated single, linear, and tree type topologies and used Wireshark to understand OpenFlow and flow modification messages while trying to communicate between two hosts. The following commands have been used to run a HTTP server on one host and use the other host to interact with it.

mininet> h1 python -m http.server 80 &

mininet> h2 wget -O - h1

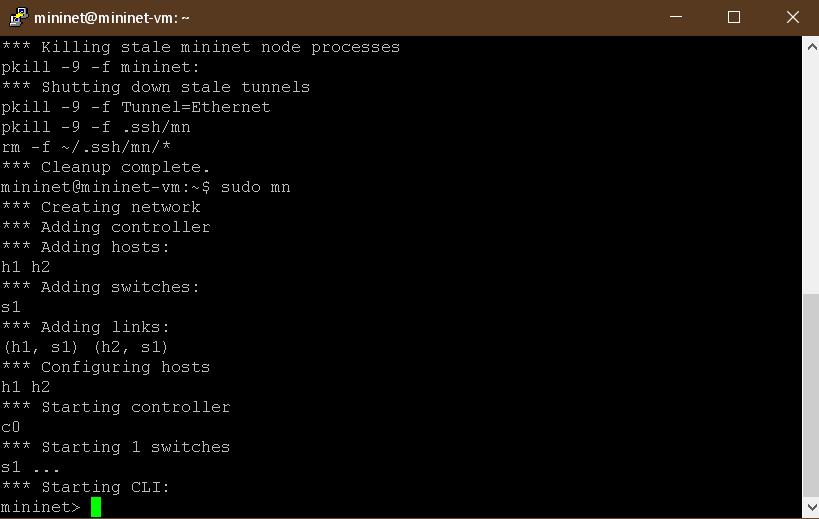


Figure Default Topology

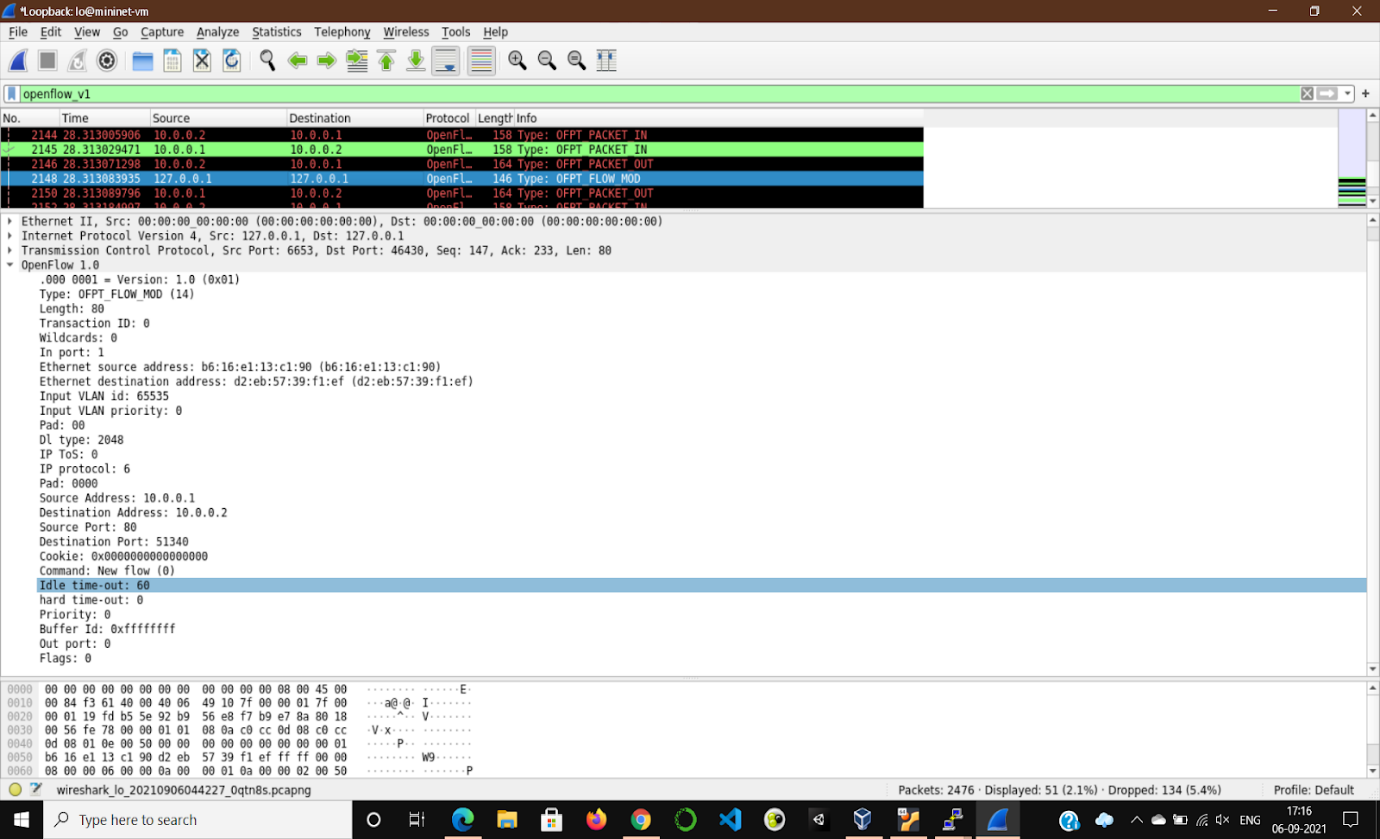


Figure Wireshark to analyse packet flow

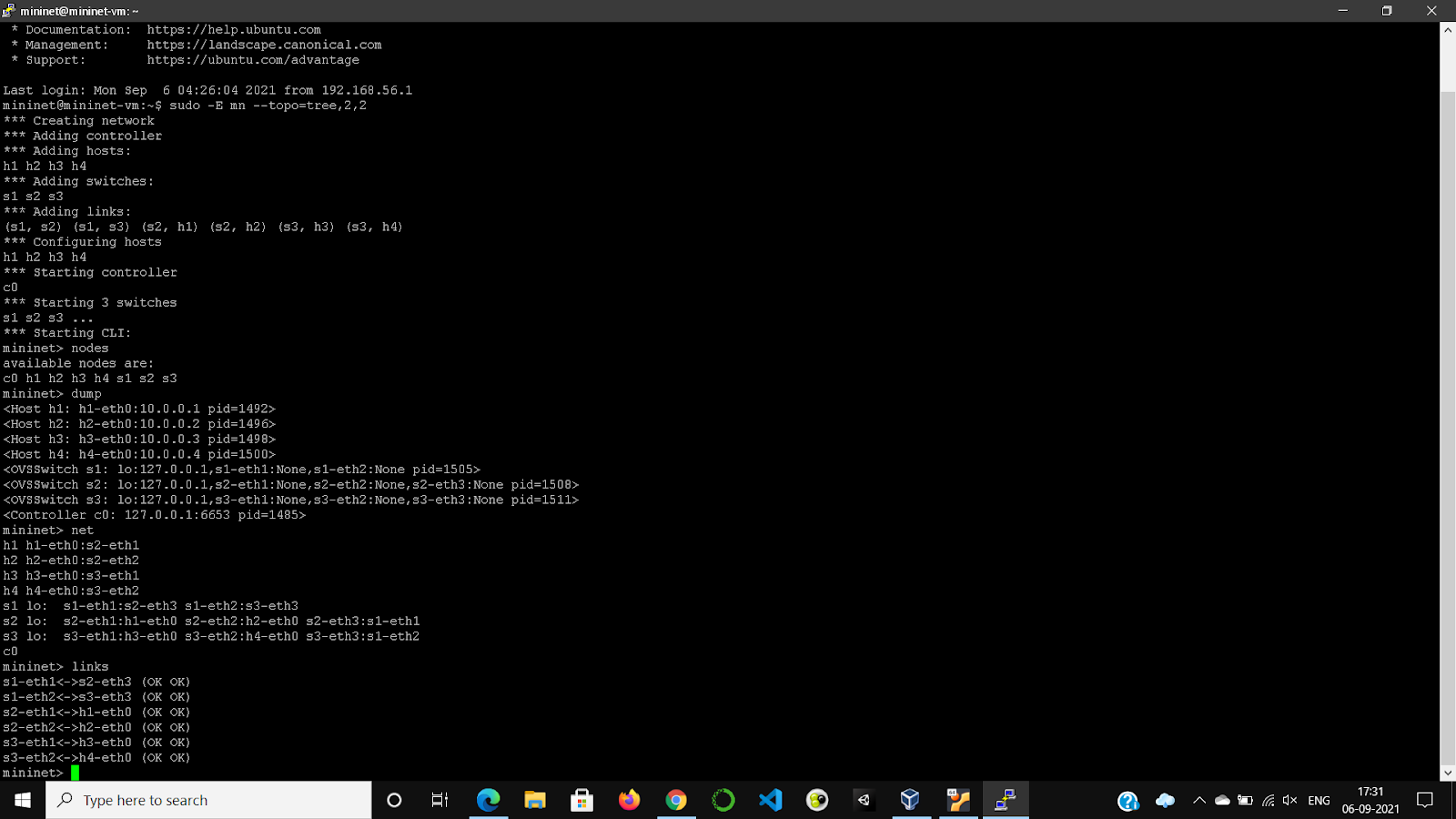


Figure Tree type Topology

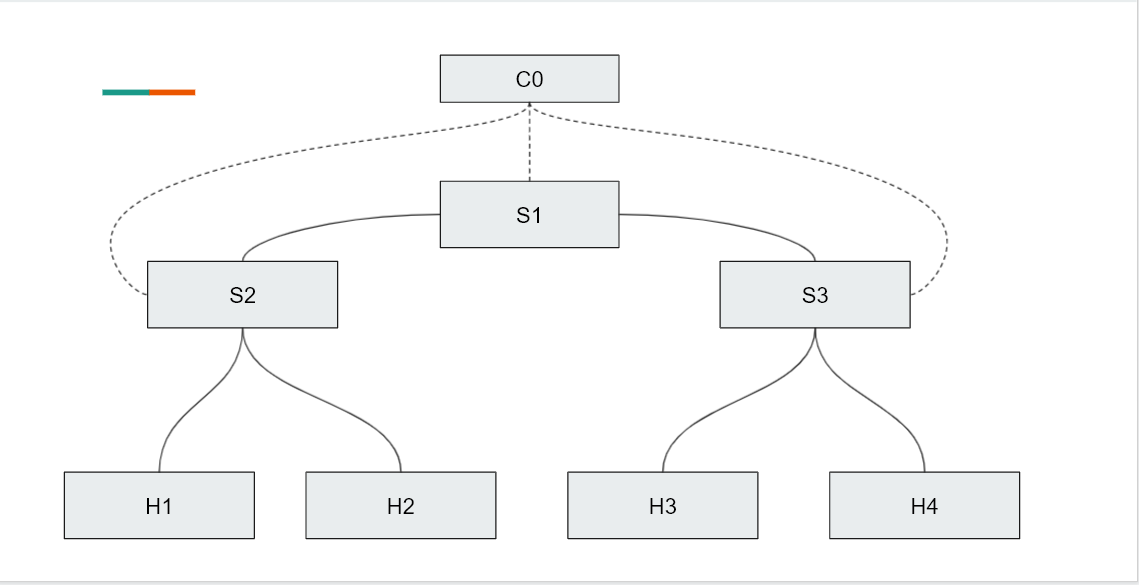


Figure The tree type topology mentioned above

**Looked into Hub and Switch Behaviours**

After getting acquainted with the basics of Mininet, I looked into POX based controller. POX is a Python-based SDN controller platform. Using the pseudo code given in the of\_tutorial controller code, and an algorithm with a tutorial about additional steps, I understood the difference between a hub and a switch and how to implement a switch from a hub. A hub floods all ports whenever it receives a packet to be forwarded. But a learning switch must not do that unless it is the first time it is seeing a certain port. Basic idea was to create a routing table at the controller to remember the ports of hosts using the packet data they are sending. Although most of the code was already available, it has been a good learning experience to implement it again and to understand the basic flow used for a learning switch. In fact, I was able to understand the difference between a hub and a switch in terms of network parameters by running iperf from Mininet CLI.

**Given Algorithmic Idea:**

* The learning switch "brain" is associated with a single OpenFlow switch. When we see a packet, we'd like to output it on a port which will eventually lead to the destination.
* To accomplish this, we build a table that maps addresses to ports.
* We populate the table by observing traffic. When we see a packet from some source coming from some port, we know that source is out that port.
* When we want to forward traffic, we look up the destination in our table. If we don't know the port, we simply send the message out to all ports except the one it came in on

For each packet from the switch:

1) Use source address and switch port to update address/port table

2) Is transparent = False and either Ethertype is LLDP or the packet's

the destination address is a Bridge Filtered address?

Yes:

2a) Drop packet -- don't forward link-local traffic (LLDP, 802.1x)

DONE

3) Is destination multicast?

Yes: 3a) Flood the packet

DONE

4) Port for the destination address in our address/port table?

No: 4a) Flood the packet

DONE

5) Is the output port the same as the input port?

Yes: 5a) Drop packet and similar ones for a while

6) Install flow table entry in the switch so that this the flow goes out the appropriate port

6a) Send the packet out appropriate port

**Related Code:**

from pox.core import core

import pox.openflow.libopenflow\_01 as of

from pox.lib.util import dpid\_to\_str, str\_to\_dpid

from pox.lib.util import str\_to\_bool

import time

log = core.getLogger()

# We don't want to flood immediately when a switch connects.

# Can be overridden on the command line.

\_flood\_delay = 0

class LearningSwitch (object):

def \_\_init\_\_ (self, connection, transparent):

# Switch we'll be adding L2 learning switch capabilities to

self.connection = connection

self.transparent = transparent

# Our table

self.macToPort = {}

# We want to hear PacketIn messages, so we listen

# to the connection

connection.addListeners(self)

# We just use this to know when to log a helpful message

self.hold\_down\_expired = \_flood\_delay == 0

#log.debug("Initializing LearningSwitch, transparent=%s",

# str(self.transparent))

def \_handle\_PacketIn (self, event):

"""

Handle packets in messages from the switch to implement the above algorithm.

"""

packet = event.parsed

def flood (message = None):

""" Floods the packet """

msg = of.ofp\_packet\_out()

if time.time() - self.connection.connect\_time >= \_flood\_delay:

# Only flood if we've been connected for a little while...

if self.hold\_down\_expired is False:

# Oh yes, it is!

self.hold\_down\_expired = True

log.info("%s: Flood hold-down expired -- flooding",

dpid\_to\_str(event.dpid))

if message is not None: log.debug(message)

#log.debug("%i: flood %s -> %s", event.dpid,packet.src,packet.dst)

# OFPP\_FLOOD is optional; on some switches you may need to change

# this to OFPP\_ALL.

msg.actions.append(of.ofp\_action\_output(port = of.OFPP\_FLOOD))

else:

pass

#log.info("Holding down flood for %s", dpid\_to\_str(event.dpid))

msg.data = event.ofp

msg.in\_port = event.port

self.connection.send(msg)

def drop (duration = None):

"""

Drops this packet and optionally installs a flow to continue

dropping similar ones for a while

"""

if duration is not None:

if not isinstance(duration, tuple):

duration = (duration,duration)

msg = of.ofp\_flow\_mod()

msg.match = of.ofp\_match.from\_packet(packet)

msg.idle\_timeout = duration[0]

msg.hard\_timeout = duration[1]

msg.buffer\_id = event.ofp.buffer\_id

self.connection.send(msg)

elif event.ofp.buffer\_id is not None:

msg = of.ofp\_packet\_out()

msg.buffer\_id = event.ofp.buffer\_id

msg.in\_port = event.port

self.connection.send(msg)

self.macToPort[packet.src] = event.port # 1

if not self.transparent: # 2

if packet.type == packet.LLDP\_TYPE or packet.dst.isBridgeFiltered():

drop() # 2a

return

if packet.dst.is\_multicast:

flood() # 3a

else:

if packet.dst not in self.macToPort: # 4

flood("Port for %s unknown -- flooding" % (packet.dst,)) # 4a

else:

port = self.macToPort[packet.dst]

if port == event.port: # 5

# 5a

log.warning("Same port for packet from %s -> %s on %s.%s. Drop."

% (packet.src, packet.dst, dpid\_to\_str(event.dpid), port))

drop(10)

return

# 6

log.debug("installing flow for %s.%i -> %s.%i" %

(packet.src, event.port, packet.dst, port))

msg = of.ofp\_flow\_mod()

msg.match = of.ofp\_match.from\_packet(packet, event.port)

msg.idle\_timeout = 10

msg.hard\_timeout = 30

msg.actions.append(of.ofp\_action\_output(port = port))

msg.data = event.ofp # 6a

self.connection.send(msg)

class l2\_learning (object):

"""

Waits for OpenFlow switches to connect and makes them learning switches.

"""

def \_\_init\_\_ (self, transparent, ignore = None):

"""

Initialize

See LearningSwitch for meaning of 'transparent'

'ignore' is an optional list/set of DPIDs to ignore

"""

core.openflow.addListeners(self)

self.transparent = transparent

self.ignore = set(ignore) if ignore else ()

def \_handle\_ConnectionUp (self, event):

if event.dpid in self.ignore:

log.debug("Ignoring connection %s" % (event.connection,))

return

log.debug("Connection %s" % (event.connection,))

LearningSwitch(event.connection, self.transparent)

def launch (transparent=False, hold\_down=\_flood\_delay, ignore = None):

"""

Starts an L2 learning switch.

"""

try:

global \_flood\_delay

\_flood\_delay = int(str(hold\_down), 10)

assert \_flood\_delay >= 0

except:

raise RuntimeError("Expected hold-down to be a number")

if ignore:

ignore = ignore.replace(',', ' ').split()

ignore = set(str\_to\_dpid(dpid) for dpid in ignore)

core.registerNew(l2\_learning, str\_to\_bool(transparent), ignore)

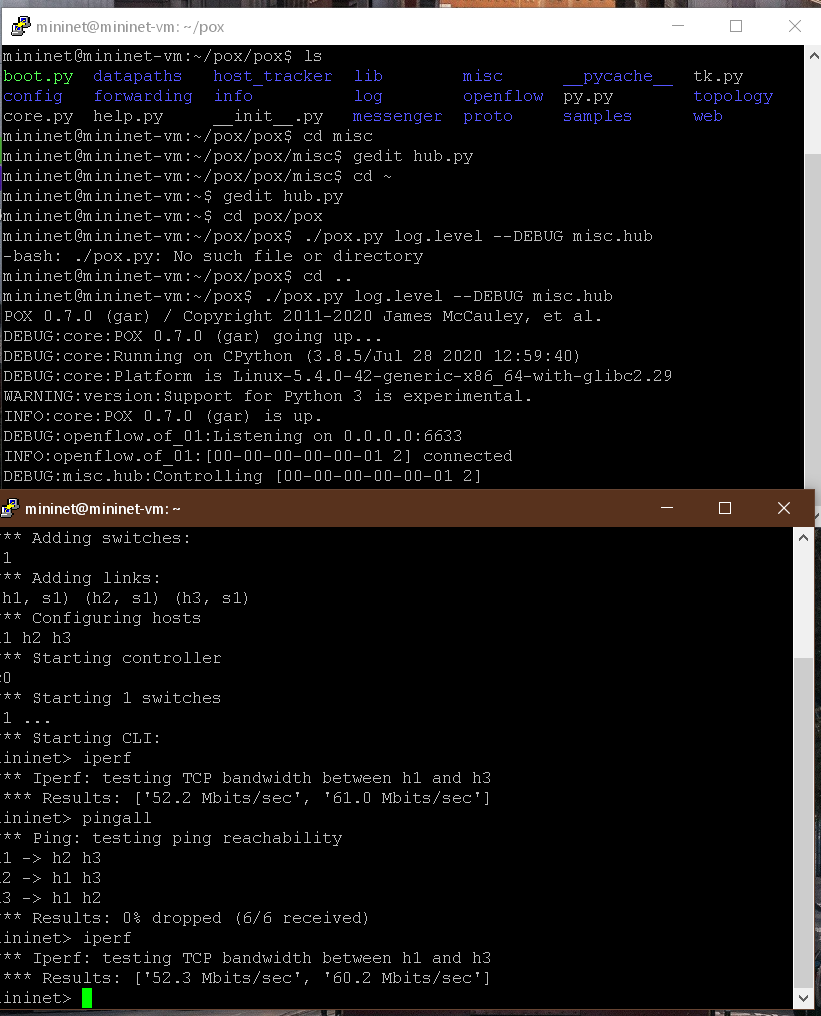


Figure Hub TCP Bandwidth

**Hub vs Learning Switch TCP Bandwidths**

We can see that for the learning switch, we are getting better TCP bandwidth values (in Gbits/sec) when compared to a hub (in Mbits/sec).

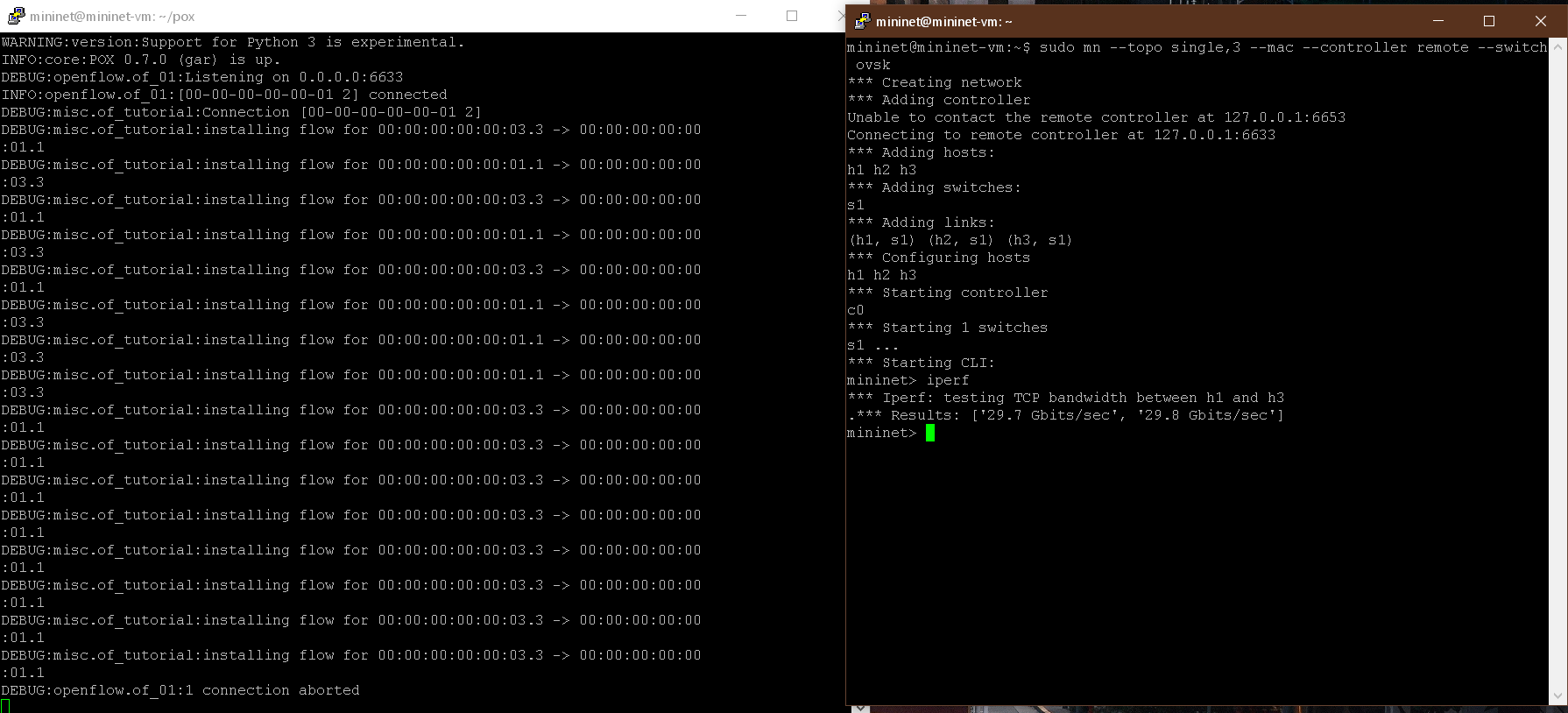


Figure Learning Switch TCP Bandwidth

**Mininet Custom Topologies**

After that, I utilised Mininet Python API to design custom topologies according to our needs. The Mininet.topo class allows us to build custom topologies and the Mininet.link.TCLink class allows us to set custom link parameters for links in the network. Implemented topologies varying from basic linear, tree type to data centre type topologies.

Shown below is the implementation of a scaled University network involving hostels, residential areas, academic areas, etc. Made use of the Topo and the TCLink classes to build the topology and to set link parameters specific to each link.

**Related Code:**

"""IIT KGP sample topology

"""

from mininet.topo import Topo

from mininet.link import TCLink

class MyTopo( Topo ):

def build( self ):

# Add hosts and switches

simHost = self.addHost( 'H0' )

hostel1H1 = self.addHost( 'H1' )

hostel1H2 = self.addHost( 'H2' )

hostelSwitch1 = self.addSwitch( 'S1' )

hostel2H3 = self.addHost( 'H3' )

hostel2H4 = self.addHost( 'H4' )

hostelSwitch2 = self.addSwitch( 'S2' )

hostel3H5 = self.addHost( 'H6' )

hostel3H6 = self.addHost( 'H5' )

hostelSwitch3 = self.addSwitch( 'S3' )

dpid = 256

aggSwitch1 = self.addSwitch( 'AS1', dpid='%x' % dpid )

acad1H7 = self.addHost( 'H7' )

acad1H8 = self.addHost( 'H8' )

acadSwitch1 = self.addSwitch( 'S4' )

acad2H9 = self.addHost( 'H7' )

acad2H10 = self.addHost( 'H8' )

acadSwitch2 = self.addSwitch( 'S5' )

dpid = dpid + 16

aggSwitch2 = self.addSwitch( 'AS2', dpid='%x' % dpid )

resH11 = self.addHost( 'H11' )

resH12 = self.addHost( 'H12' )

resSwitch = self.addSwitch( 'S6' )

dpid = dpid + 16

cicCoreSwitch = self.addSwitch( 'CIC', dpid='%x' % dpid )

# Add links in hostel 1

self.addLink( hostel1H1, hostelSwitch1, cls=TCLink, bw=10, delay='2ms' )

self.addLink( hostel1H2, hostelSwitch1, cls=TCLink, bw=10, delay='2ms' )

self.addLink( hostelSwitch1, aggSwitch1, cls=TCLink, bw=10, delay='2ms' )

# Add links in hostel 2

self.addLink( hostel2H3, hostelSwitch2, cls=TCLink, bw=10, delay='2ms' )

self.addLink( hostel2H4, hostelSwitch2, cls=TCLink, bw=10, delay='2ms' )

self.addLink( hostelSwitch2, aggSwitch1, cls=TCLink, bw=10, delay='2ms' )

# Add links in hostel 3

self.addLink( hostel3H5, hostelSwitch3, cls=TCLink, bw=10, delay='2ms' )

self.addLink( hostel3H6, hostelSwitch3, cls=TCLink, bw=10, delay='2ms' )

self.addLink( hostelSwitch3, aggSwitch1, cls=TCLink, bw=10, delay='2ms' )

# Add links in academic area 1

self.addLink( acad1H7, acadSwitch1, cls=TCLink, bw=10, delay='2ms' )

self.addLink( acad1H8, acadSwitch1, cls=TCLink, bw=10, delay='2ms' )

self.addLink( acadSwitch1, aggSwitch2, cls=TCLink, bw=20, delay='2ms' )

# Add links in academic area 2

self.addLink( acad2H9, acadSwitch2, cls=TCLink, bw=10, delay='2ms' )

self.addLink( acad2H10, acadSwitch2, cls=TCLink, bw=10, delay='2ms' )

self.addLink( acadSwitch2, aggSwitch2, cls=TCLink, bw=20, delay='2ms' )

# Add links in residential area

self.addLink( resH11, resSwitch, cls=TCLink, bw=10, delay='2ms' )

self.addLink( resH12, resSwitch, cls=TCLink, bw=10, delay='2ms' )

# Add links to CIC core switch

self.addLink( resSwitch, cicCoreSwitch, cls=TCLink, bw=20, delay='2ms' )

self.addLink( aggSwitch1, cicCoreSwitch, cls=TCLink, bw=20, delay='2ms' )

self.addLink( aggSwitch2, cicCoreSwitch, cls=TCLink, bw=40, delay='2ms' )

self.addLink( simHost, cicCoreSwitch, cls=TCLink, bw=50, delay='2ms' )

topos = { 'mytopo': ( lambda: MyTopo() ) }

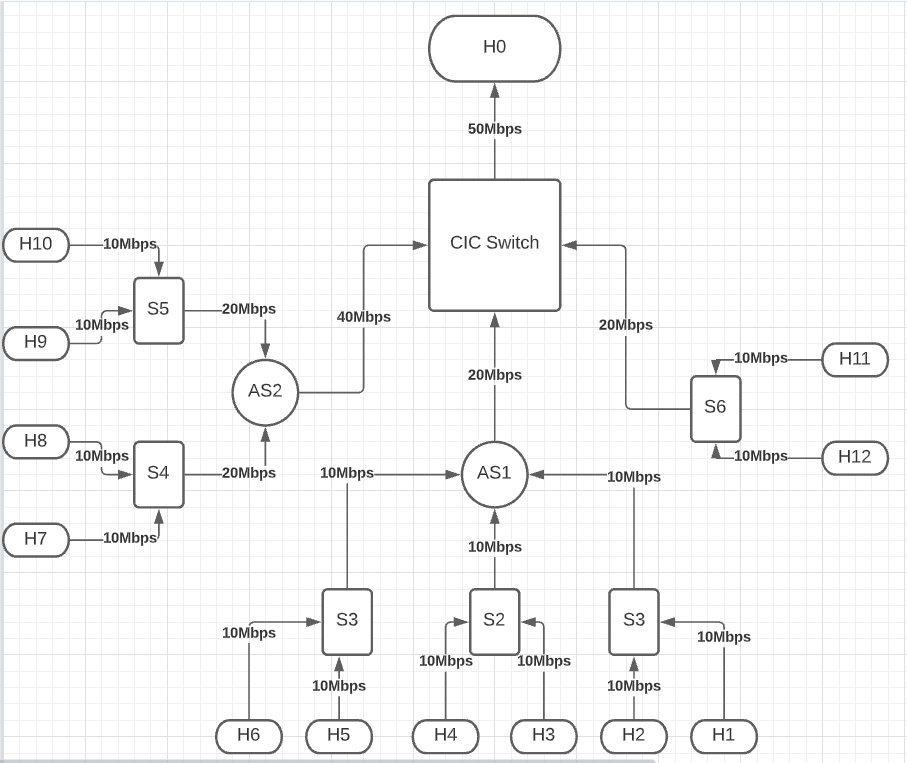


Figure Sketch of the above topology

**Explanation**

H0 represents a host outside the network

Hi 🡪 Hosts on the network, i ɛ {1,2,3,4,5,6,7,8,9,10,11,12}

Si 🡪 Switches on the network, I ɛ {1,2,3,4,5,6}

AS1, AS2 🡪 Aggregator Switches

More specifically, all links incoming into AS1 represent the hostel network. All links incoming into AS2 represent the Academic Area network. And the links incoming to Switch S6 represent the Residential area.

Every link has a set bandwidth value and it is written on the link.

**Generating Traffic**

Using the above topology, I generated traffic at some nodes to check the behaviour of the links and the switches. For this, D-ITG was used. D-ITG (Distributed Internet Traffic Generator) is a platform capable to produce traffic at packet level accurately replicating appropriate stochastic processes for both IDT (Inter Departure Time) and PS (Packet Size) random variables. D-ITG supports both IPv4 and IPv6 traffic generation and it is capable to generate traffic at network, transport, and application layer.

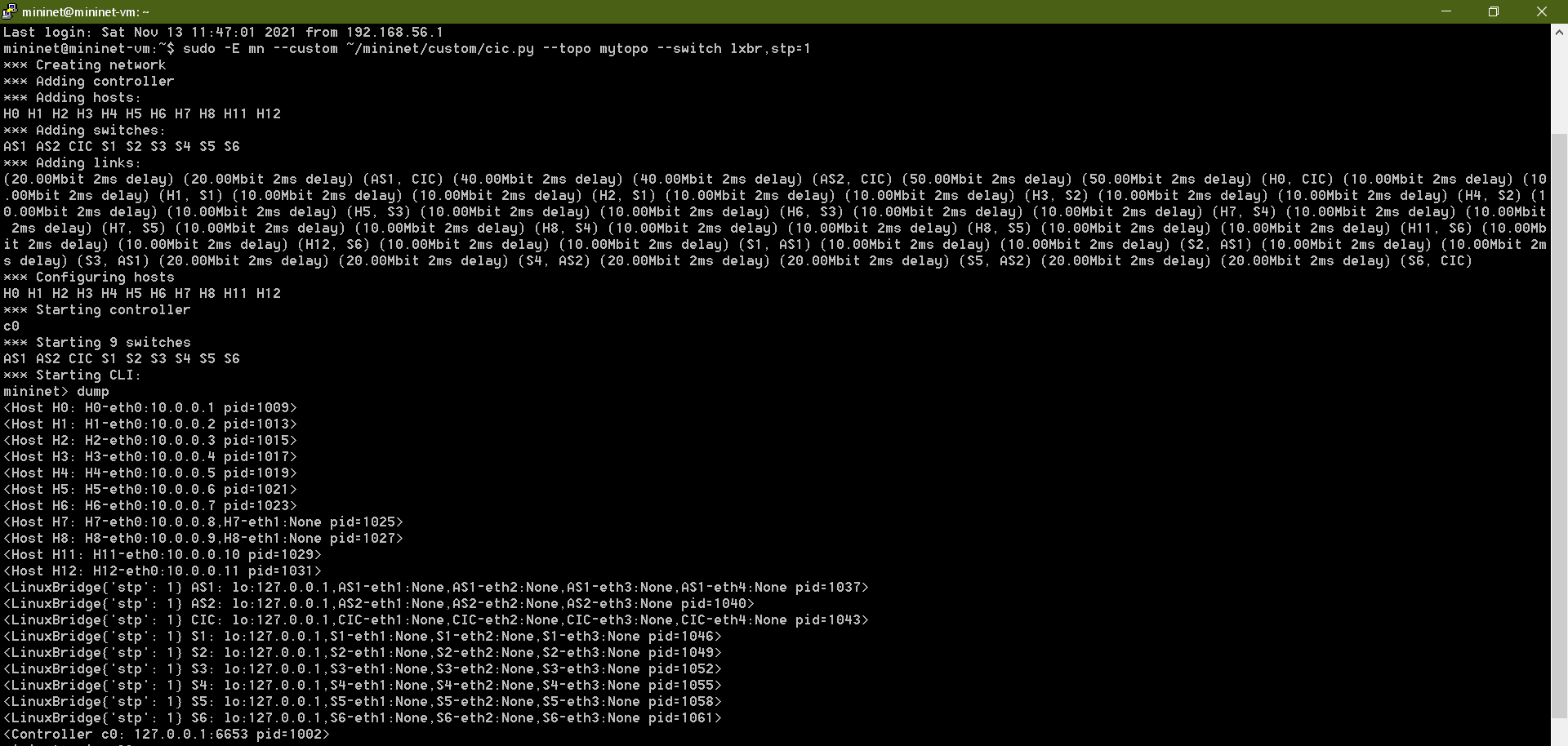


Figure Network Details of Custom Topology

By opening the xterm windows of hosts, we can generate traffic by using the following command. The ‘c’ option denotes how many bytes of data per packet. The ‘C’ option denotes how many packets of data per second. The ‘a’ option denotes what node is acting as the receiver. In our case, H0 is listening to all the traffic. The ‘t’ option denotes total time duration in milliseconds.

**./ITGSend –T UDP  –a 10.0.0.1 –c 1000 –C 1000 –t 15000 -l sender.log –x receiver.log**

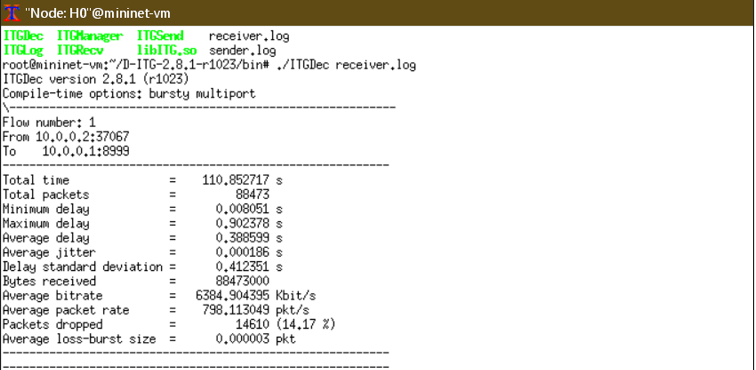


Figure Receiver log after completion of total duration-1

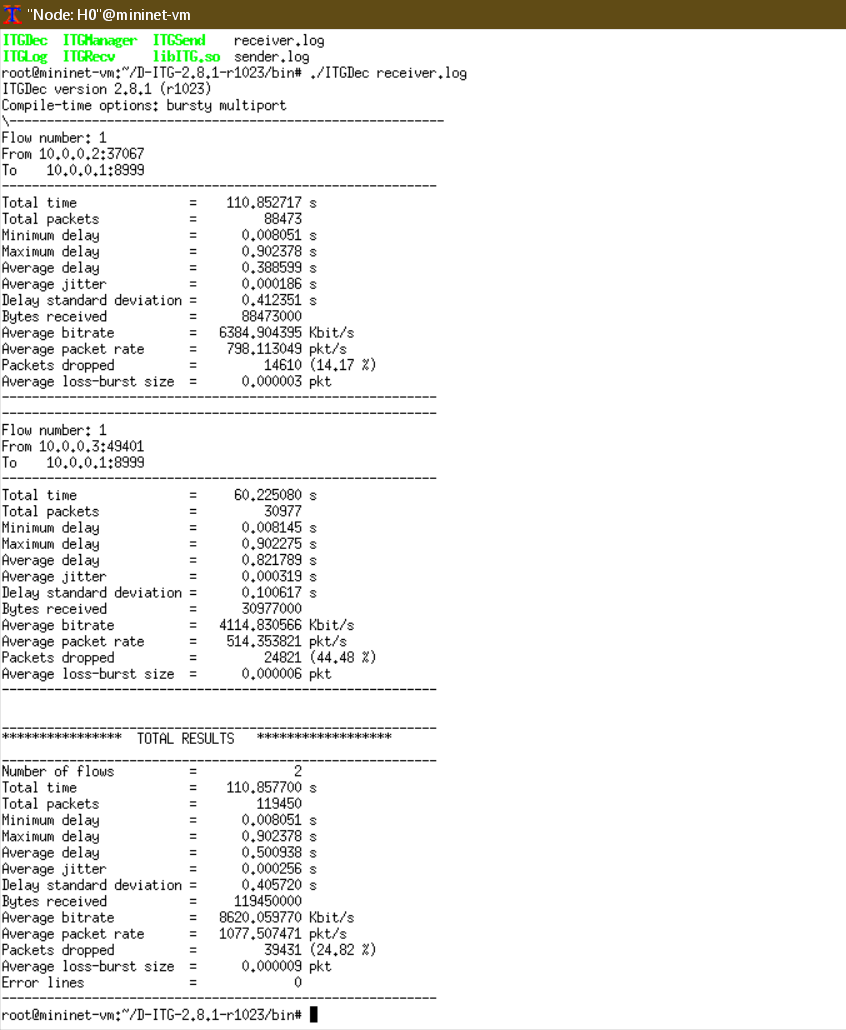


Figure Receiver log after completion of total duration-2

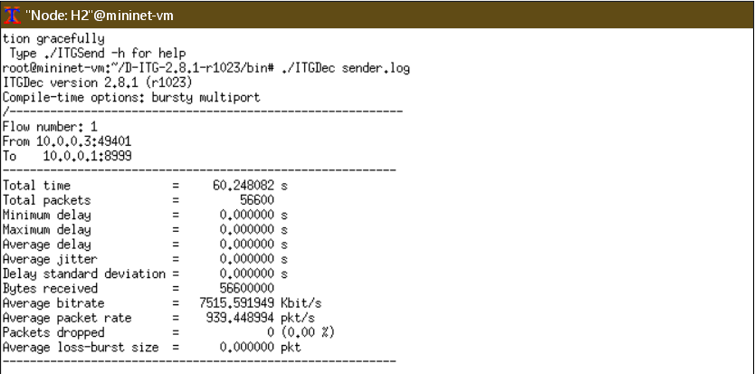


Figure Sender Log (Part1)

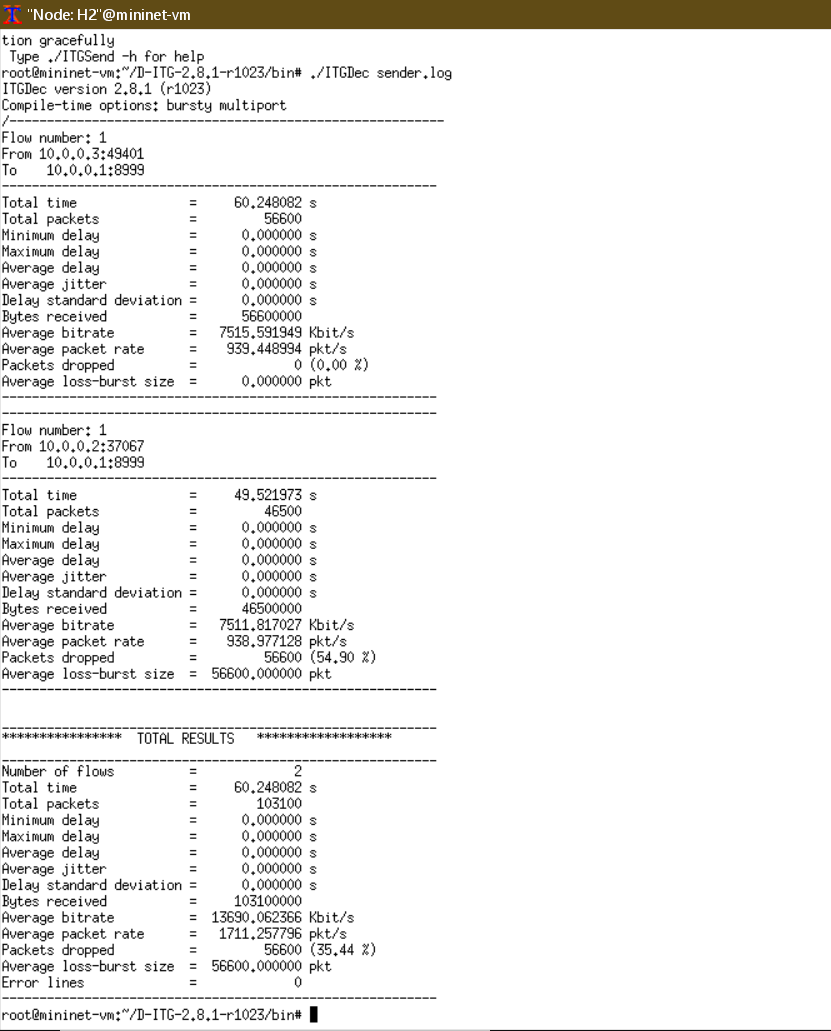


Figure Sender Log (Part2)

From the above logs, we can see that by using (1000 bytes) x (1000 packets per second) we are able to generate approximately 8Mbps traffic per second. But, because of the links being only 10Mbps of bandwidth, we can see packet drop details as well which are sizeable amount and quite random since we have not installed any policy yet to determine flows when there is a choke or bottleneck in the network links.

**CONCLUSION**

Mininet is a very useful tool to emulate networks and to observe how certain things work. Its CLI tools are very useful and understandable to make our work easier. It even provides us with easier topology generation tools like Miniedit (not used here). Overall, I was able to understand a lot of things about SDN, Mininet, OpenFlow, D-ITG as to how they work and how they could be used. All related files can be found at the github link provided at the end.

**REFERENCES**

1. <http://mininet.org/download/>
2. <http://mininet.org/walkthrough/>
3. <http://mininet.org/vm-setup-notes/>
4. <https://github.com/mininet/mininet/wiki/FAQ#x11-forwarding>
5. <https://github.com/mininet/openflow-tutorial/wiki>
6. <https://github.com/mininet/mininet/wiki/Documentation>
7. <https://stackoverflow.com/questions/34932495/forward-x11-failed-network-error-connection-refused>
8. <https://forum.qt.io/topic/102617/wireshark-failed-to-get-the-current-screen-resources-using-xming-ssh>
9. <https://github.com/mininet/mininet/wiki/Introduction-to-Mininet>
10. <https://noxrepo.github.io/pox-doc/html/>
11. <https://tech.ginkos.in/2019/07/is-sdn-really-dead.html>
12. <https://docs.openvswitch.org/en/latest/tutorials/faucet/>
13. <http://rlenglet.github.io/openfaucet/index.html>
14. <https://www.brianlinkletter.com/2015/04/how-to-use-miniedit-mininets-graphical-user-interface/>
15. <http://mininet.org/api/classmininet_1_1topo_1_1Topo.html>
16. <https://www.cse.wustl.edu/~jain/cse570-13/ftp/m_03dct.pdf>
17. <https://www.techtarget.com/searchnetworking/definition/spanning-tree-protocol>
18. <https://www.inap.com/blog/spanning-tree-protocol-explained/>
19. <https://www.cisco.com/c/en/us/td/docs/switches/lan/catalyst6500/ios/12-2SX/configuration/guide/book/spantree.html>
20. <http://mininet.org/api/classmininet_1_1link_1_1TCLink.html>
21. <https://github.com/nsg-ethz/minigenerator/blob/master/minigenerator/examples/simple_topo_test.py>
22. <http://traffic.comics.unina.it/software/ITG/manual/index.html#SECTION00040000000000000000>
23. <http://sdnopenflow.blogspot.com/2015/05/using-of-d-itg-traffic-generator-in.html>
24. <https://www.wikipedia.org/>
25. <https://github.com/harshpritam/Design-Lab>