REPORT
Computer Networks Lab Assignment 1

Question 1:

FUNCTION EXPLANATION

PART 1:
addSwitch(): adds a switch to a topology and returns the switch name
addHost(): adds a host to a topology and returns the host name
addLink(): adds a bidirectional link to a topology (and returns a link key, but this is not important). Links in Mininet are bidirectional unless noted otherwise.
self.addLink(node1, node2): adds a bidirectional link
To add new features which can be invoked using the mn command, you need to define a dict in your --custom file based on the option type. The dict's keys are short names passed to the appropriate option, and the values are the corresponding subclasses, constructors, or functions:

PART 2:

Pox

pox provides a collection of utilities for navigating and manipulating filesystems. This module is designed to facilitate some of the low level operating system interactions that are useful when exploring a filesystem on a remote host, where queries such as “what is the root of the filesystem?”, “what is the user’s name?”, and “what login shell is preferred?” become essential in allowing a remote user to function as if they were logged in locally. While pox is in the same vein of both the os and shutil builtin modules, the majority of its functionality is unique and compliments these two modules.

OpenFlow

One of Mininet's most powerful and useful features is that it uses Software Defined Networking. Using the OpenFlow protocol and related tools, you can program switches to do almost anything you want with the packets that enter them. OpenFlow makes emulators like Mininet much more useful, since network system designs, including custom packet forwarding with OpenFlow, can easily be transferred to hardware OpenFlow switches for line-rate operation.

openflow.of_01

This component communicates with OpenFlow 1.0 (wire protocol 0x01) switches. When other components that use OpenFlow are loaded, this component is usually started with default values automatically. Holver, you may want to launch it manually in order to change its options. You may also want to launch it manually to run it multiple times (e.g., to listen for OpenFlow connections on multiple ports).

openflow.spanning_tree

This component uses the discovery component to build a view of the network topology, constructs a spanning tree, and then disables flooding on switch ports that aren't on the tree. The result is that topologies with loops no longer turn your network into useless hot packet soup.
--no-flood disables flooding on all ports as soon as a switch connects; on some ports, it will be enabled later.
--hold-down prevents altering of flood control until a complete discovery cycle has completed (and thus, all links have had an opportunity to be discovered).
Thus, the safest (and probably the most sensible) invocation is openflow.spanning_tree --no-flood --hold-down

forwarding.l2_learning

This component makes OpenFlow switches act as a type of L2 learning switch. This one operates much like NOX's "pyswitch" example, although the implementation is quite different. While this component learns L2 addresses, the flows it installs are exact-matches on as many fields as possible. For example, different TCP connections will result in different flows being installed.

openflow.discovery

This component sends specially-crafted LLDP messages out of OpenFlow switches so that it can discover the network topology. It raises events (which you can listen to) when links go up or down.

CODE EXPLANATION

PART 1:

import the python required libraries

```
from mininet.topo import Topo
from mininet.net import Mininet
```

Write the Topology definition class

E.g.

```
class SingleSwitchTopo(Topo):
    def build(self):
        s1 = self.addSwitch('s1')
        h1 = self.addHost('h1')
        h2 = self.addHost('h2')
        h3 = self.addHost('h3')
        h4 = self.addHost('h4')
```

```
self.addLink(h1, s1)
self.addLink(h2, s1)
self.addLink(h3, s1)
self.addLink(h4, s1)
```

PART 2:

Topology Code Changes from Part 1

Changing the controller to a remote one so I may use POX. I need to first import RemoteController instead of the regular Controller in our module import statements and then I can pass that through to the Mininet class. These changes are basically -

```
from mininet.node import RemoteController
```

And,

```
net = Mininet( controller=RemoteController )
```

Adding MAC addresses while initializing the hosts so that I can easily define the Layer-2 rules for our Firewall.

Firewall Code

cd into the pox directory and then type cd pox/misc/. This is where I will place our firewall code Create a new file called Myfirewall.py in which I write our code:

First, I import all the classes I will require -

```
from pox.core import core
import pox.openflow.libopenflow_01 as of
from pox.lib.revent import *
from pox.lib.addresses import EthAddr
```

Then I specify our rules for the Firewall by writing down the Layer-2 addresses of the hosts. I then put each of these as rows in a list so that I may iterate over each rule separately later Next I create our SDNFirewall class in which the actual controller is going to be accessing and checking flows and modifying flow tables accordingly. The first function __init__ is just a constructor. The second and more interesting function, _handle_ConnectionUp will fire up each time a host tries to reach another through the switches. When this happens, I will first iterate over each rule in our list of rules. Here I create match fields by providing the two hosts in the rule, specified by rule[0] and rule[1] to block. I then create an OpenFlow flow_mod message using of.ofp_flow_mod() and set its match field to our blocking rule. At the end, I use event.connection.send(flow_mod) to send our blocking rule to the switch so that it can be enforced. Lastly, I create the launch function that POX requires and pass our SDNFirewall class to it.

PROCEDURE

Part 1

Step1 : preliminary step to install mininet and pox on system.

Step2 : Now create custom topology through python script using mininet.topo api.
Step3 : run the script via mininet (command : sudo mn --custom IIT2018185_Q1_p1.py --topo mytopo --mac --controller=remote,ip=127.0.0.1,port=6633)
Step4 : Create Flow among switches by typing given commands for all switches :
For e.g. for switch ‘Sw1c1’

```
> sh ovs-ofctl dump-flows Sw1c1
> sh ovs-ofctl add-flow Sw1c1 "priority=0,action=normal"
```

Step5 : enter pingall command
Step6 : desired output is generated with all flow among hosts as active.

Part 2

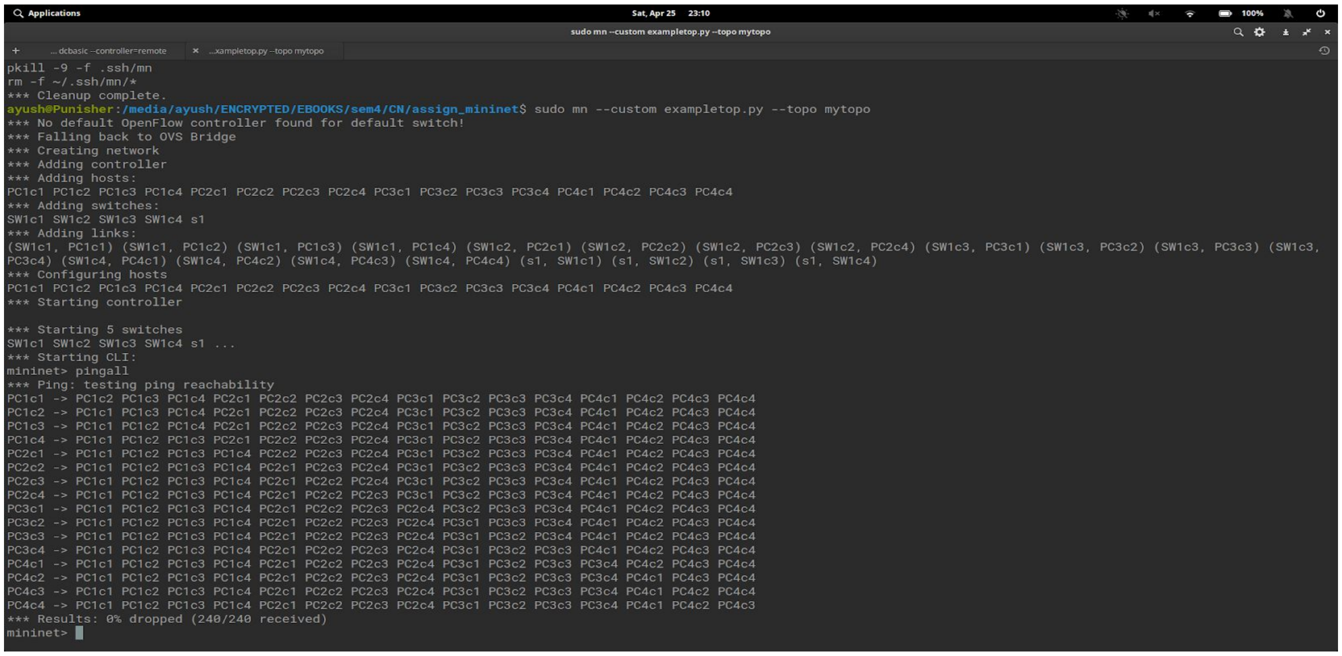
Step1 : download pox from https://github.com/noxrepo/pox
Step2 : Now create a custom Myfirewall.py which will block the desired hosts among each other.
Step3 : cd /pox
Step4 : chmod a+x pox.py
Step5 : run generate_rules.py to generate rules table for blocked flow EtherAddress
Srep6 : add rules to IIT2018185_myFirewall.py
Step7 : copy the Myfirewall.py in /pox/pox/misc
Step8 : run the command : ./pox.py forwarding.l2_learning openflow.discovery openflow.spanning_tree --no-flood --hold-down pox.misc.IIT2018185_myFirewall
Step9 : (In other terminal) run the topology in mininet (command : sudo mn --custom IIT2018185_Q1_p2.py.py --topo mytopo --mac --controller=remote,ip=127.0.0.1,port=6633)
Step10 : in mininet console type : pingall
Step11 : Desired output with blockage is observed

Result

Part 1
After adding flow amongst all switches. 0% packet dropped is observed, i.e 240/240 packets are transferred successfully.
Part 2
After adding the Firewall to the topology 20% packed dropped is observed i.e 192/240 packets transferred successfully.

Screenshots

Part 1

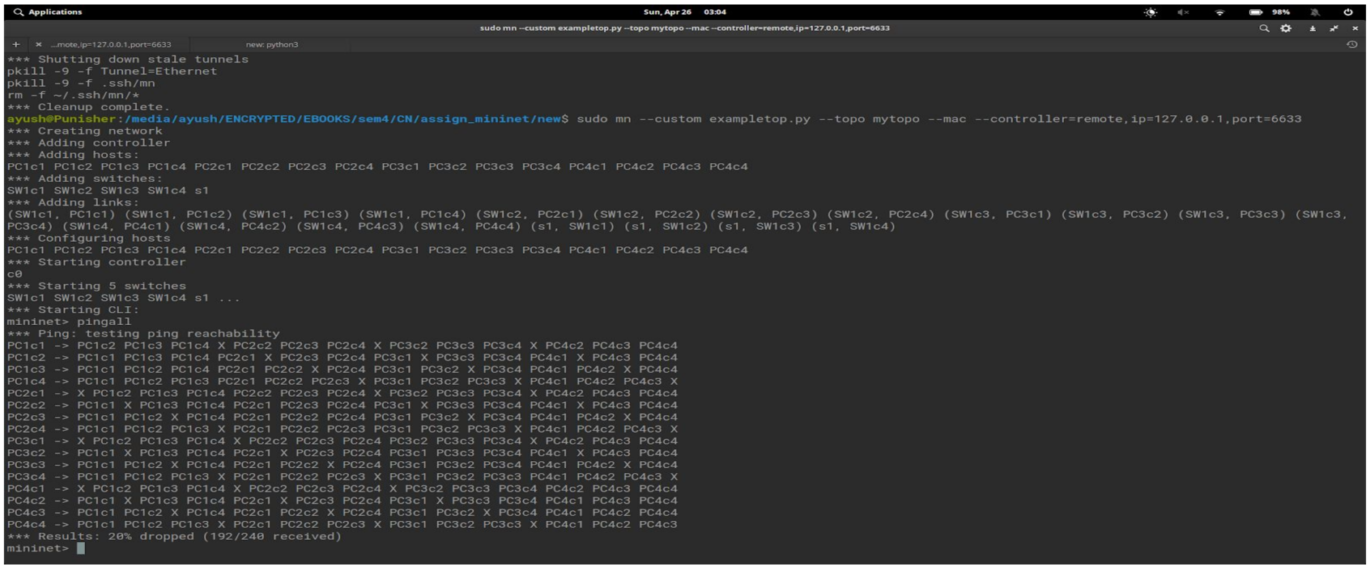


```
pkill -9 -f .ssh/mn
rm -f ~/.ssh/mn/*
*** Cleanup complete.
ayush@Punisher:/media/ayush/ENCRYPTED/EBOOKS/sem4/CN/assign_mininet$ sudo mn --custom exampletop.py --topo mytopo
*** No default OpenFlow controller found for default switch!
*** Falling back to OVS Bridge
*** Creating network
*** Adding controller
*** Adding hosts:
PC1c1 PC1c2 PC1c3 PC1c4 PC2c1 PC2c2 PC2c3 PC2c4 PC3c1 PC3c2 PC3c3 PC3c4 PC4c1 PC4c2 PC4c3 PC4c4
*** Adding switches:
SW1c1 SW1c2 SW1c3 SW1c4 s1
*** Adding links:
(SW1c1, PC1c1) (SW1c1, PC1c2) (SW1c1, PC1c3) (SW1c1, PC1c4) (SW1c2, PC2c1) (SW1c2, PC2c2) (SW1c2, PC2c3) (SW1c2, PC2c4) (SW1c3, PC3c1) (SW1c3, PC3c2) (SW1c3, PC3c3) (SW1c3,
PC3c4) (SW1c4, PC4c1) (SW1c4, PC4c2) (SW1c4, PC4c3) (SW1c4, PC4c4) (s1, SW1c1) (s1, SW1c2) (s1, SW1c3) (s1, SW1c4)
*** Configuring hosts
PC1c1 PC1c2 PC1c3 PC1c4 PC2c1 PC2c2 PC2c3 PC2c4 PC3c1 PC3c2 PC3c3 PC3c4 PC4c1 PC4c2 PC4c3 PC4c4
*** Starting controller

*** Starting 5 switches
SW1c1 SW1c2 SW1c3 SW1c4 s1 ...
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
PC1c1 -> PC1c2 PC1c3 PC1c4 PC2c1 PC2c2 PC2c3 PC2c4 PC3c1 PC3c2 PC3c3 PC3c4 PC4c1 PC4c2 PC4c3 PC4c4
PC1c2 -> PC1c1 PC1c3 PC1c4 PC2c1 PC2c2 PC2c3 PC2c4 PC3c1 PC3c2 PC3c3 PC3c4 PC4c1 PC4c2 PC4c3 PC4c4
PC1c3 -> PC1c1 PC1c2 PC1c4 PC2c1 PC2c2 PC2c3 PC2c4 PC3c1 PC3c2 PC3c3 PC3c4 PC4c1 PC4c2 PC4c3 PC4c4
PC1c4 -> PC1c1 PC1c2 PC1c3 PC2c1 PC2c2 PC2c3 PC2c4 PC3c1 PC3c2 PC3c3 PC3c4 PC4c1 PC4c2 PC4c3 PC4c4
PC2c1 -> PC1c1 PC1c2 PC1c3 PC1c4 PC2c2 PC2c3 PC2c4 PC3c1 PC3c2 PC3c3 PC3c4 PC4c1 PC4c2 PC4c3 PC4c4
PC2c2 -> PC1c1 PC1c2 PC1c3 PC1c4 PC2c1 PC2c3 PC2c4 PC3c1 PC3c2 PC3c3 PC3c4 PC4c1 PC4c2 PC4c3 PC4c4
PC2c3 -> PC1c1 PC1c2 PC1c3 PC1c4 PC2c1 PC2c2 PC2c4 PC3c1 PC3c2 PC3c3 PC3c4 PC4c1 PC4c2 PC4c3 PC4c4
PC2c4 -> PC1c1 PC1c2 PC1c3 PC1c4 PC2c1 PC2c2 PC2c3 PC3c1 PC3c2 PC3c3 PC3c4 PC4c1 PC4c2 PC4c3 PC4c4
PC3c1 -> PC1c1 PC1c2 PC1c3 PC1c4 PC2c1 PC2c2 PC2c3 PC2c4 PC3c2 PC3c3 PC3c4 PC4c1 PC4c2 PC4c3 PC4c4
PC3c2 -> PC1c1 PC1c2 PC1c3 PC1c4 PC2c1 PC2c2 PC2c3 PC2c4 PC3c1 PC3c3 PC3c4 PC4c1 PC4c2 PC4c3 PC4c4
PC3c3 -> PC1c1 PC1c2 PC1c3 PC1c4 PC2c1 PC2c2 PC2c3 PC2c4 PC3c1 PC3c2 PC3c4 PC4c1 PC4c2 PC4c3 PC4c4
PC3c4 -> PC1c1 PC1c2 PC1c3 PC1c4 PC2c1 PC2c2 PC2c3 PC2c4 PC3c1 PC3c2 PC3c3 PC4c1 PC4c2 PC4c3 PC4c4
PC4c1 -> PC1c1 PC1c2 PC1c3 PC1c4 PC2c1 PC2c2 PC2c3 PC2c4 PC3c1 PC3c2 PC3c3 PC3c4 PC4c2 PC4c3 PC4c4
PC4c2 -> PC1c1 PC1c2 PC1c3 PC1c4 PC2c1 PC2c2 PC2c3 PC2c4 PC3c1 PC3c2 PC3c3 PC3c4 PC4c1 PC4c3 PC4c4
PC4c3 -> PC1c1 PC1c2 PC1c3 PC1c4 PC2c1 PC2c2 PC2c3 PC2c4 PC3c1 PC3c2 PC3c3 PC3c4 PC4c1 PC4c2 PC4c4
PC4c4 -> PC1c1 PC1c2 PC1c3 PC1c4 PC2c1 PC2c2 PC2c3 PC2c4 PC3c1 PC3c2 PC3c3 PC3c4 PC4c1 PC4c2 PC4c3
*** Results: 0% dropped (240/240 received)
mininet>
```

Result 0 % dropped 240/240

Part 2



```
*** Shutting down stale tunnels
pkill -9 -f Tunnel=Ethernet
pkill -9 -f .ssh/mn
rm -f ~/.ssh/mn/*
*** Cleanup complete.
ayush@Punisher:/media/ayush/ENCRYPTED/EBOOKS/sem4/CN/assign_mininet/new$ sudo mn --custom exampletop.py --topo mytopo --mac --controller=remote,ip=127.0.0.1,port=6633
*** Creating network
*** Adding controller
*** Adding hosts:
PC1c1 PC1c2 PC1c3 PC1c4 PC2c1 PC2c2 PC2c3 PC2c4 PC3c1 PC3c2 PC3c3 PC3c4 PC4c1 PC4c2 PC4c3 PC4c4
*** Adding switches:
SW1c1 SW1c2 SW1c3 SW1c4 s1
*** Adding links:
(SW1c1, PC1c1) (SW1c1, PC1c2) (SW1c1, PC1c3) (SW1c1, PC1c4) (SW1c2, PC2c1) (SW1c2, PC2c2) (SW1c2, PC2c3) (SW1c2, PC2c4) (SW1c3, PC3c1) (SW1c3, PC3c2) (SW1c3, PC3c3) (SW1c3,
PC3c4) (SW1c4, PC4c1) (SW1c4, PC4c2) (SW1c4, PC4c3) (SW1c4, PC4c4) (s1, SW1c1) (s1, SW1c2) (s1, SW1c3) (s1, SW1c4)
*** Configuring hosts
PC1c1 PC1c2 PC1c3 PC1c4 PC2c1 PC2c2 PC2c3 PC2c4 PC3c1 PC3c2 PC3c3 PC3c4 PC4c1 PC4c2 PC4c3 PC4c4
*** Starting controller
c0
*** Starting 5 switches
SW1c1 SW1c2 SW1c3 SW1c4 s1 ...
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
PC1c1 -> PC1c2 PC1c3 PC1c4 X PC2c2 PC2c3 PC2c4 X PC3c2 PC3c3 PC3c4 X PC4c2 PC4c3 PC4c4
PC1c2 -> PC1c1 PC1c3 PC1c4 PC2c1 X PC2c3 PC2c4 PC3c1 X PC3c3 PC3c4 PC4c1 X PC4c3 PC4c4
PC1c3 -> PC1c1 PC1c2 PC1c4 PC2c1 PC2c2 X PC2c4 PC3c1 PC3c2 X PC3c4 PC4c1 PC4c2 X PC4c4
PC1c4 -> PC1c1 PC1c2 PC1c3 PC2c1 PC2c2 PC2c3 X PC3c1 PC3c2 PC3c3 X PC4c1 PC4c2 PC4c3 X
PC2c1 -> X PC1c2 PC1c3 PC1c4 PC2c2 PC2c3 PC2c4 X PC3c2 PC3c3 PC3c4 X PC4c2 PC4c3 PC4c4
PC2c2 -> PC1c1 X PC1c3 PC1c4 PC2c1 PC2c3 PC2c4 PC3c1 X PC3c3 PC3c4 PC4c1 X PC4c3 PC4c4
PC2c3 -> PC1c1 PC1c2 X PC1c4 PC2c1 PC2c2 PC2c4 PC3c1 PC3c2 X PC3c4 PC4c1 PC4c2 X PC4c4
PC2c4 -> PC1c1 PC1c2 PC1c3 X PC2c1 PC2c2 PC2c3 PC3c1 PC3c2 PC3c3 X PC4c1 PC4c2 PC4c3 X
PC3c1 -> X PC1c2 PC1c3 PC1c4 X PC2c2 PC2c3 PC2c4 PC3c2 PC3c3 PC3c4 X PC4c2 PC4c3 PC4c4
PC3c2 -> PC1c1 X PC1c3 PC1c4 PC2c1 X PC2c3 PC2c4 PC3c1 PC3c3 PC3c4 PC4c1 X PC4c3 PC4c4
PC3c3 -> PC1c1 PC1c2 X PC1c4 PC2c1 PC2c2 X PC2c4 PC3c1 PC3c2 PC3c4 PC4c1 PC4c2 X PC4c4
PC3c4 -> PC1c1 PC1c2 PC1c3 X PC2c1 PC2c2 PC2c3 X PC3c1 PC3c2 PC3c3 PC4c1 PC4c2 PC4c3 X
PC4c1 -> X PC1c2 PC1c3 PC1c4 X PC2c2 PC2c3 PC2c4 X PC3c2 PC3c3 PC3c4 PC4c2 PC4c3 PC4c4
PC4c2 -> PC1c1 X PC1c3 PC1c4 PC2c1 X PC2c3 PC2c4 PC3c1 X PC3c3 PC3c4 PC4c1 PC4c3 PC4c4
PC4c3 -> PC1c1 PC1c2 X PC1c4 PC2c1 PC2c2 X PC2c4 PC3c1 PC3c2 X PC3c4 PC4c1 PC4c2 PC4c4
PC4c4 -> PC1c1 PC1c2 PC1c3 X PC2c1 PC2c2 PC2c3 X PC3c1 PC3c2 PC3c3 X PC4c1 PC4c2 PC4c3
*** Results: 20% dropped (192/240 received)
mininet>
```

Result 20% dropped 192/240

Question 2

Algorithm:

PIE - Proportional Integral controller-Enhanced AQM algorithm

Proportional Integral controller-Enhanced (PIE) is a control theoretic active queue management scheme. It is based on the proportional integral controller but aims to control delay. The main design goals are

- o Low latency control
- o High link utilization
- o Simple implementation
- o Guaranteed stability and fast responsiveness

PIE is designed to control delay effectively. First, an average rate is estimated based on the standing queue. The rate is to calculate the current delay. Then, on a periodic basis, the delay is used to calculate the dropping probability. Finally, a packet is dropped (or marked) based on this probability.

PIE makes adjustments to the probability based on the trend of the i.e. whether it is going up or down.The delay converges to the target value specified. Alpha and beta are statically chosen parameters chosen to control the probability growth and are determined through control theoretic. alpha determines how the deviation between the current target latency changes probability. beta exerts additional depending on the latency trend.

BufferBloat

Bufferbloat is a phenomenon in which excess buffers in the network cause high latency and latency variation. As more and more interactive applications (e.g., voice over IP, real-time video streaming, and financial transactions) run in the Internet, high latency and latency variation degrade application performance. There is a pressing need to design intelligent queue management schemes that can control latency and latency variation, and hence provide desirable quality of service to users.

Procedure:

Step 1 : create hosts and add appropriate links between hosts and switches
Step 2 : create link between switches, notice that the delay is different
Step 3 : use CLI to explore this setting, i.e. run iperf on multiple hosts and run ping
Step 4 : Then, comment out CLI(net) and mimic bufferbloat.py, automate what you did in CLI,
Step 5 : such that running tcpfairness.py will output all the delivers in the problem.
Step 6 : Turn off the xterm in CustomMininet by setting xterms=False
Step 7 : Let each experiment run for 200 seconds at least
Step 8 : iperf -c ... -P 10 | grep SUM gives aggregation information of 10 flows

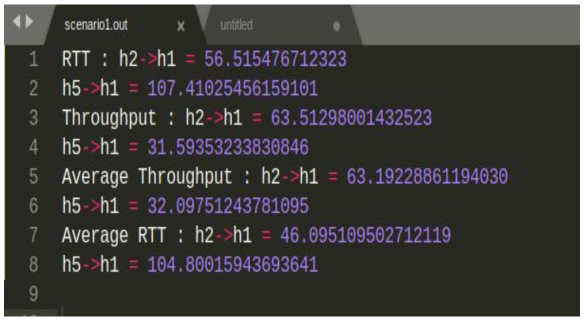
Step 9 : iperf -c -i 1 gives a measurement every 1 second
Step 10 : run command python3 IIT2018185_Q2.py to generate RTT values of desired flow

Result :

Scenario 1.
Congestion windows of h2->h1 and h5->h1 flows are roughly the same,
because CWND = throughput * rtt and TCP's RTT unfairness ratio.

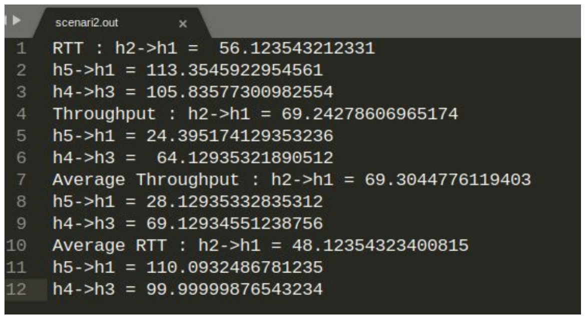
Scenario 2.
S1 → h1 has the larger drop rate. Use the TCP throughput equation, with Flow
A = h5 → h1, Flow B = h2 → h1, Flow C = h4 → h3.
$p_{ss} = 1/ T^{2}{}_{C}RTT^{2}{}_{C}$
$p_{sh} = 1/ T^{2}{}_{B}RT\ T^{2}{}_{B}$
As we know, T_C = 100 − x and T_B = 100 − x since A and C share a bottleneck link with capacity 100, and the same for A and B. So, T_C = TB. Since $RTT_C > RTT_{B,}$ we can conclude that psh is greater.

Screenshots



```
RTT : h2->h1 = 56.515476712323
h5->h1 = 107.41025456159101
Throughput : h2->h1 = 63.51298001432523
h5->h1 = 31.59353233830846
Average Throughput : h2->h1 = 63.19228861194030
h5->h1 = 32.09751243781095
Average RTT : h2->h1 = 46.095109502712119
h5->h1 = 104.80015943693641
```

Scenario1 output



```
RTT : h2->h1 =  56.123543212331
h5->h1 = 113.3545922954561
h4->h3 = 105.83577300982554
Throughput : h2->h1 = 69.24278606965174
h5->h1 = 24.395174129353236
h4->h3 =  64.12935321890512
Average Throughput : h2->h1 = 69.3044776119403
h5->h1 = 28.12935332835312
h4->h3 = 69.12934551238756
Average RTT : h2->h1 = 48.12354323400815
h5->h1 = 110.0932486781235
h4->h3 = 99.99999876543234
```

Scenario2 output