Question1:-

RUNNING SIMULATION FOR QUESTION 1:-

We have this main bufferbloat.py file present in the question1 folder.

1) Normal buffer bloat experiment:-
   Just run the command:- Sudo ./run.sh in that folder.
   You will get all graphs.

2) To enable AQM PIE algorithm:-
   Just remove the comment present on lines 160,161,162 to enable this algorithm.
   Content of lines are as follows :-
   ```
   #switch.cmd('tc qdisc add dev s0-eth2 root pie limit 1000 target 20ms')
   #print "AQM pie Algorithm added"
   #switch.cmd('tc -s qdisc show')
   ```

3) To add 10 parallel flow:-
   Comment line number 119 and remove the comment from line number 120
   Content of lines are:-
   Before :-
   ```
   h1.popen("iperf -c %s -t %s > %s/iperf.out" % (h2.IP(), args.time, args.dir), shell=True)
   #h1.popen("iperf -c %s -t %s -P 10> %s/iperf.out" % (h2.IP(), args.time, args.dir),
   shell=True)
   ```
   After :-
   ```
   #h1.popen("iperf -c %s -t %s > %s/iperf.out" % (h2.IP(), args.time, args.dir),
   shell=True)
   h1.popen("iperf -c %s -t %s -P 10> %s/iperf.out" % (h2.IP(), args.time, args.dir),
   shell=True)
   ```

QUESTION 2:-

RUNNING SIMULATION FOR QUESTION 2:-
We have this main bufferbloat.py file present in the question2 folder.

1) We just have to add a loss parameter in the addLink function.
   self.addLink(hosts[i], switch, bw=args.bw_host,loss=2, delay=args.delay,
                          max_queue_size=args.maxq)
   You change this loss parameter to get different value.(Make sure that loss value is an integer value(percentage))

2) Normal buffer bloat experiment:-
   Just run the command:- Sudo ./run.sh in that folder.
   You will get all graphs.

3) To enable AQM PIE algorithm:-
   Just remove the comment present on lines 160,161,162 to enable this algorithm.
   Content of lines are as follows :-
   #switch.cmd('tc qdisc add dev s0-eth2 root pie limit 1000 target 20ms')
   #print "AQM pie Algorithm added"
   #switch.cmd('tc -s qdisc show')

4) To add 10 parallel flow:-
   Comment line number 119 and remove the comment from line number 120
   Content of lines are:-
   Before :-
   h1.popen("iperf -c %s -t %s > %s/iperf.out" % (h2.IP(), args.time, args.dir), shell=True)
   #h1.popen("iperf -c %s -t %s -P 10> %s/iperf.out" % (h2.IP(), args.time, args.dir), shell=True)
   After :-
   #h1.popen("iperf -c %s -t %s > %s/iperf.out" % (h2.IP(), args.time, args.dir), shell=True)
   h1.popen("iperf -c %s -t %s -P 10> %s/iperf.out" % (h2.IP(), args.time, args.dir), shell=True)

QUESTION 3:-

We have two servers:-
TCP Server:-
```
server = h2.popen("iperf -s -w 16m")

# TODO: Start the iperf client on h1.  Ensure that you create a
# long lived TCP flow. You may need to redirect iperf's stdout to avoid blocking.
h1 = net.get('h1')
h1.popen("iperf -c %s -t %s > %s/iperf.out" % (h2.IP(), args.time, args.dir), shell=True)
```

UDP Server:-
```
#starting UDP server.
   h2.popen("iperf -s -u -p 5566 -i 1 > %s/server.out"%(args.dir),shell =True)
   h1.popen("iperf -c %s -u -b 10M -t 25 -p 5566  > %s/iperf1.out" % (h2.IP(), args.dir),
shell=True)
```

RUNNING SIMULATION FOR QUESTION 2:-
We have this main bufferbloat.py file present in the question3 folder.

1) We just have to add a loss parameter in the addLink function.
   self.addLink(hosts[i], switch, bw=args.bw_host,loss=2, delay=args.delay,
   max_queue_size=args.maxq)
   You change this loss parameter to get different value.(Make sure that loss value is an integer value(percentage))

2) Normal buffer bloat experiment:-
   Just run the command:- Sudo ./run.sh in that folder.
   You will get all graphs.

3) To enable AQM PIE algorithm:-
   Just remove the comment present on lines 168,169,170 to enable this algorithm.
   Content of lines are as follows :-
   ```
   #switch.cmd('tc qdisc add dev s0-eth2 root pie limit 1000 target 20ms')
   #print "AQM pie Algorithm added"
   #switch.cmd('tc -s qdisc show')
   ```

4) To add 10 parallel flow:-
   Comment line number 121 and remove the comment from line number 122
   Content of lines are:-
   Before :-
   ```
   h1.popen("iperf -c %s -t %s > %s/iperf.out" % (h2.IP(), args.time, args.dir), shell=True)
   ```

#h1.popen("iperf -c %s -t %s -P 10> %s/iperf.out" % (h2.IP(), args.time, args.dir), shell=True)
After :-
#h1.popen("iperf -c %s -t %s > %s/iperf.out" % (h2.IP(), args.time, args.dir), shell=True)
h1.popen("iperf -c %s -t %s -P 10> %s/iperf.out" % (h2.IP(), args.time, args.dir), shell=True)