# Network Term Project Part:1 Report
Group 19

Onur Ozdemir
*2036473*
*ODTU Bilgisayar Muhendisligi*
Ankara, Turkey
e2036473@ceng.metu.edu.tr

Turgut Turkmen
*2036226*
*ODTU Bilgisayar Muhendisligi*
Ankara, Turkey
e2036226@ceng.metu.edu.tr

***Abstract*—This report explains the methods used while doing the Network Term Project Part 1. It includes three experiments to see how end to end delay from source to destination will change in our network.**

## I. Introduction

In the report We begin with mentioning the problems we faced on a slice and the solution we found. Later on, We continue with explaining the coding methods and then lastly, We finish the report with the three experiments and observations about them.

## II. Reserving the Resources

When we first begin to our term project, after some struggling we accomplish to get some resources from **Clemson InstaGENI**. After completing our code, we began to test the end to end delays with the following time synchronization methods:

- From the terminal:
  *sudo service ntp stop && sudo ntpdate -s time.nist.gov && sudo service ntp start*
  We executed the command at source and destination and then run our services.
- Inside our code:
  *os.system("sudo service ntp stop")*
  *os.system("sudo ntpdate -s time.nist.gov")*
  *os.system("sudo service ntp start")*
  We tried to accomplish the time synchronization by adding those lines to source and destination nodes.

But both of the methods didn't work and we got negative end to end delay values. We tried really hard on this problem by modifying our code and playing with time synchronization part but we couldn't get reasonable values at any point.

After some search we learn that some other groups also had this problem and the problem may be the resource itself(Clemson InstaGENI). Then we decided to get a new slice and we reserved **Rutgers InstaGENI**. Then the problem is disappeared when we run the first command mentioned on the source and destination nodes, which is:

*sudo service ntp stop && sudo ntpdate -s time.nist.gov && sudo service ntp start*

The second method still wasn't working. So we decided to move on with the terminal codes, whenever we need the time synchronization.

## III. Implementation

### A. Source Node

In the Source Node we needed to create a TCP Connection with the Broker Node. To accomplish this the source node simply creates a TCP connection on *IP: "10.10.1.2" and Port: "8000"* with Broker Node.

Source Node also creates some number of packets in a for loop, sends a packet, then waits for the ack message to come and then continue with sending the second packet etc. This approach prevents packets to be lost or dropped on their way to the destination. That's why we have 0 packet loss.

We also observe the time by:

- *time.time()*

We observe it just before sending the packet and just after the packet is received to calculate the end to end delay between source and the destination.

### B. Broker Node

In the Broker Node we needed to create a TCP Connection with the Source Node, get the corresponding packet and transfer these packets to Router1 and Router2 nodes with a UDP Connection. To accomplish this we created a TCP Connection with source node on link0 *IP: "10.10.1.2" and Port: "8000"*. Then we transfer the incoming packets by a simple routing dictionary over a UDP Connection:

- *1: [("10.10.2.2", 8000), ("10.10.2.1", 8001)],*
  *2: [("10.10.4.2", 8000), ("10.10.4.1", 8002)]*
  First items of lines to send Router1 and Router2 , second items of lines to listen from Router1 and Router2

We also listen the incoming packets from Router1 and Router2 Nodes on UDP Connection then transfer the incoming packets to the Source Node on the same TCP Connection created between Source Node and the Broker Node.

Identify applicable funding agency here. If none, delete this.

The routing from Source Node to Router1 and Router2 node is totally random. We have chosen this design choice to get closer to the real case scenarios.

### C. Router1 and Router2 Nodes

The Router1 and Router2 nodes don't have an important role on our deign.

- Router1 Node

  Router1 Node listens the Broker Node from
  *IP: "10.10.2.2" and Port: "8000"*
  over a UDP connection
  then redirect the packet to the Destination Node at
  *IP: "10.10.2.1" and Port: "8001".*

  It also listens the Destination Node from
  *IP: "10.10.3.2" and Port: "8000"*
  then redirect the packets to the Broker node on
  *IP: "10.10.3.1" and Port: "8001"*

- Router2 Node

  Router2 Node listens the Broker Node from
  *IP: "10.10.4.2" and Port: "8000"*
  over a UDP connection then redirect the packet to the Destination Node at
  *IP: "10.10.4.1" and Port: "8002".*

  It also listens the Destination Node from
  *IP: "10.10.5.2" and Port:"8000"*
  then redirect the packets to the Broker node on
  *IP: "10.10.5.1" and Port: "8001"*

### D. Destination Node

In the destination node we created two threads to listen Router1 and Router2.

- The First thread listens Router1 Node at
  *IP: "10.10.3.2" and Port: "8000"*

- While the second thread listens Router2 Node at
  *IP: "10.10.5.2" and Port: "8000"*

When the Destination Node gets a packet from Router1 or Router2, We immediately observe the time on the server. We then append this observed time value to the packet's data and redirect the packet to the Router1 Node if the packet is arrived from Router1 or to the Router2 Node if the packet is arrived from Router2 Node.
This time value is used to calculate end to end delay from Source Node to Destination Node. The calculation is done at the Source Node. It is mentioned in the Source Node section.

We also add an "ACK" string at the beginning of the packet's data. This way the Source Node can observe if the packet is arrived to the Destination Node, then It can send the next packet.

- Destination Node sends packets to the Router1 Node over a UDP Connection on
  *IP: "10.10.3.1" and Port: "8001"*

- While It sends packets to the Router2 Node on
  *IP: "10.10.5.1" and Port: "8001"*

## IV. Experiments

In the experiments we gave the delays by netem command.

### A. Experiment One

In experiment one we used the following commands:

- For Broker:
  *sudo tc qdisc change dev eth1 root netem delay 1ms 5ms distribution normal*
  *sudo tc qdisc change dev eth3 root netem delay 1ms 5ms distribution normal*
- For Router1:
  *sudo tc qdisc change dev eth2 root netem delay 1ms 5ms distribution normal*
- For Router2:
  *sudo tc qdisc change dev eth1 root netem delay 1ms 5ms distribution normal*

### B. Experiment Two

In experiment two we used the following commands:

- For Broker:
  *sudo tc qdisc change dev eth1 root netem delay 20ms 5ms distribution normal*
  *sudo tc qdisc change dev eth3 root netem delay 20ms 5ms distribution normal*
- For Router1:
  *sudo tc qdisc change dev eth2 root netem delay 20ms 5ms distribution normal*
- For Router2:
  *sudo tc qdisc change dev eth1 root netem delay 20ms 5ms distribution normal*

### C. Experiment Three

In experiment three we used the following commands:

- For Broker:
  *sudo tc qdisc change dev eth1 root netem delay 60ms 5ms distribution normal*
  *sudo tc qdisc change dev eth3 root netem delay 60ms 5ms distribution normal*
- For Router1:
  *sudo tc qdisc change dev eth2 root netem delay 60ms 5ms distribution normal*
- For Router2:
  *sudo tc qdisc change dev eth1 root netem delay 60ms 5ms distribution normal*

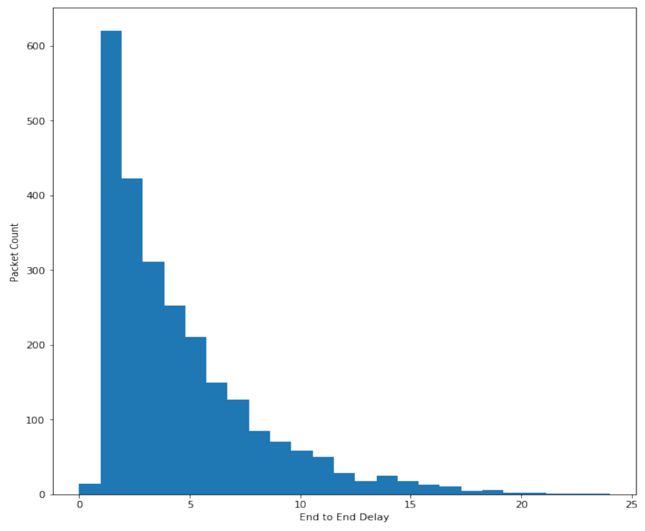

Fig. 1. Experiment One End to End Normal Distribution

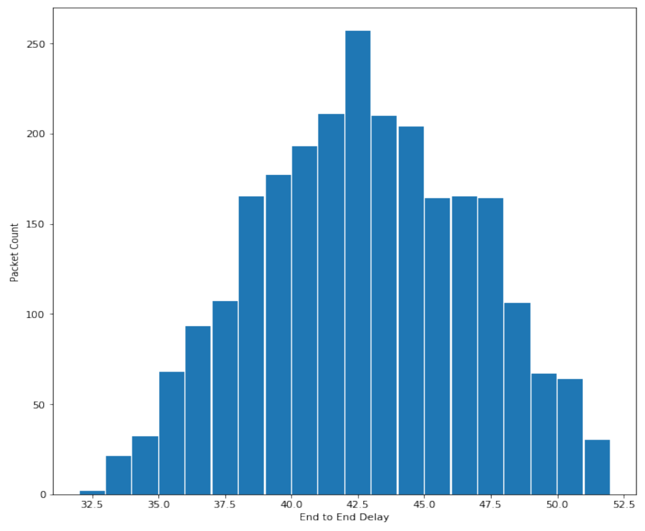

Fig. 2. Experiment Two End to End Normal Distribution

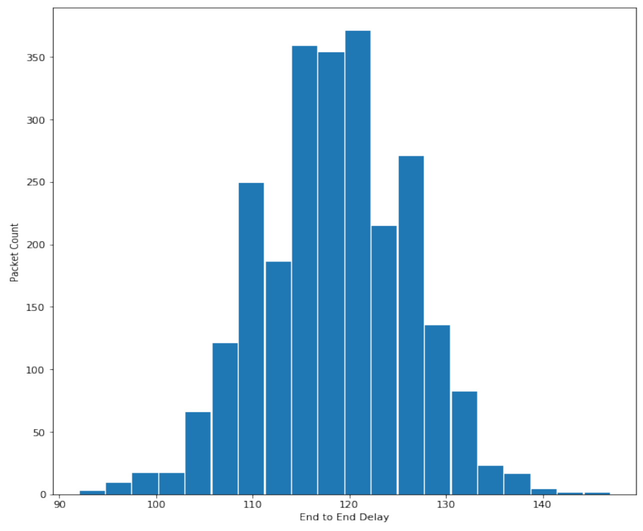

Fig. 3. Experiment Three End to End Normal Distribution

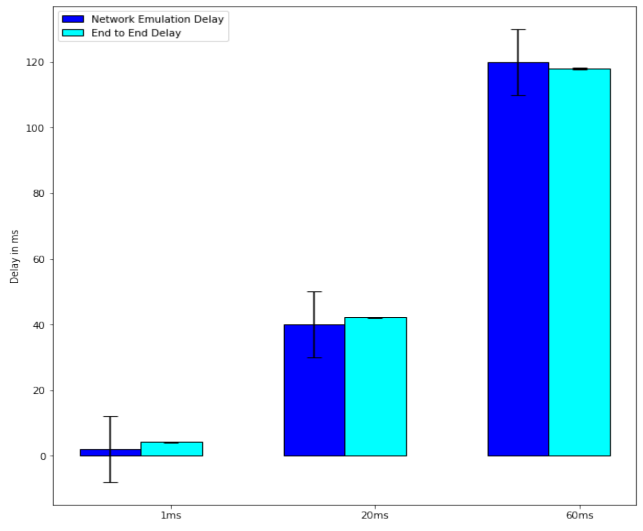

Fig. 4. Network Emulation Delay vs. End-to-End Delay

## V. Observations

### A. Experiment One

In the experiment one, the network emulation delay was 1 ± 5ms. We tried to send 2500 packets from s to d. Expected result was around 2ms, however the average delay was 4.24 ± 0.143ms. The bottleneck links (link between r1 and d, and the link between r2 and d) may cause this problem. You can see the end to end delay distribution in figure 1.

### B. Experiment Two

In the experiment two, the network emulation delay was 20 ± 5ms. We tried to send 2500 packets from s to d. Expected result was around 40ms. Experiment results satisfied this condition, and the results was 42.2652 ± 0.159ms. For 2500 packets, mean of the end to end delay was 42.2652, and standard deviation was 4.067. These values were used to find margin error. You can see the normal distribution of the experiment 2 in the figure 2.

### C. Experiment Three

In the experiment three, the network emulation delay was 60 ± 5ms. We tried to send 2500 packets from s to d. Expected result was around 120ms. Experiment results satisfied this condition, and the results was 118 ± 0.297ms. For 2500 packets, mean of the end to end delay was 118, and the standard deviation was 7.57. These values were used to find margin error. You can see the normal distribution of the experiment 3 in the figure 3.

## VI. Conclusion

After done all the experiments, we need to calculate %95 confidence interval. To accomplish that task, we used this formula:

$\overline{X} \pm Z * \frac{s}{\sqrt{n}}$

- $\overline{X}$ is the mean value. The average of all end to end delays.
- Z is the confidence interval value of %95 which is 1.960.
- s is the standard deviation of all samples.
- n is the sample count which is 2500 in our experiments.

The second part of the formula is margin error. You can see the error bar in the figure 4. We calculate the error bars for three experiments. We expect the resultant margin errors should be less than the actual margin error which is 5 ms. So, calculated margin errors are actually less than network emulation delay's margin error. Also, we expected the mean of 2500 samples are around the network emulation delay, and the results satisfied this condition.