Network Term Project Part:2 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 2. It includes three experiments with each having three different cases, to see how the file transfer time changes between source and destination. The report also explains the methodologies used to provide a reliable data transfer over a UDP socket.**

I. Introduction

In the report We begin with initial configutarions that should be set. Then we mentioned about the implementation details and how we created a reliable protocol over a unreliable protocol.

II. Initial Configurations

We reserved resources from **NYU InstaGENI**. After connecting to each server, We needed to synchronize time of all nodes in order to get a correct time transfer value between source and destination. For that purpose; similar to first part of the term project, we executed the following command on each of the node. (This could also be done at source and destination only, but it doesn't take any time to do that, so we executed the command on each terminal.)

- **From the terminal:**
  sudo service ntp stop && sudo ntpdate -s time.nist.gov && sudo service ntp start

The second step was configuring Router 1 and Router 2 nodes, so that without any script they can carry packets from broker to destination and from destination to broker. For that purpose we executed the following commands on corresponding Router 1 and Router 2 nodes:

- **On Router 1's Terminal**
  - *sudo route add -net 10.10.3.2 netmask 255.255.255.255 gw 10.10.3.2*
    Here the 10.10.3.2 represents the interface of the destination on the link between Router 1 and Destination.
  - *sudo route add -net 10.10.2.1 netmask 255.255.255.255 gw 10.10.2.1*
    Here the 10.10.2.1 represents the interface of the Broker on the link between Router 1 and Broker.
- **On Router 2's Terminal**
  - *sudo route add -net 10.10.5.2 netmask 255.255.255.255 gw 10.10.5.2*
    Here the 10.10.5.2 represents the interface of the destination on the link between Router 2 and Destination.
  - *sudo route add -net 10.10.4.1 netmask 255.255.255.255 gw 10.10.4.1*
    Here the 10.10.4.1 represents the interface of the Broker on the link between Router 2 and Broker.

In all the commands 255.255.255.255 is used because it indicates the exact match with the given IP address.

III. Implementation and Provision of Reliability over a Unreliable Connection

In this section we will explain how we change an unreliable connection to a reliable connection by also mentioning about some parts of our code.

A. Source Node

In the Source Node we simply created a TCP Connection with the Broker Node and start to transfer file with packets of 500 bytes sizes. To create the TCP connection we used *IP: "10.10.1.2" and Port: "8000"*. "10.10.1.2 is the interface of broker on the link between source and broker. TCP is a reliable protocol therefore nothing has to be done to provide reliability at this node.

B. Broker Node

In the Broker Node we needed to create a TCP Connection with the Source Node. For that purpose we listen the source from *IP: "10.10.1.2" and Port: "8000"*.

After the connection is established between Source Node and Broker Node. The Source starts to send file's packets with size of 500 bytes.(Totally 10000 packets sent by Source to Broker...) After a packet is arrived to Broker from Source, firstly the Broker calculates a checksum value for that packet and calculates a sequence number. After those calculations packets are put to a packet array to be send to destination.

- **Checksum** For checksum md5 is used. *hashlib.md5(packet).hexdigest()* returns a 32 character

string. The checksum value is used to be sure about the packet's integritiy. At the Destination Node the checksum is calculated again. If the checksum's don't match then the Destination Node do nothing with that packet. The checksum value is appended after the packet's data.

- **Sequence Number** Each packet has a sequence number to provide correct data transfer and receive order. First packet on broker has seq:0, second has seq:1 etc..
  For sequence number we used a string of 5 bytes. To represent each sequence number correctly we put 0's to beginning of the string to complete length of 5. For example, If sequence number is 3, "00003" is appended to the packet. The sequence number is put after the checksum value.

Overall packet structure to be sent to the Destination from Broker:

500 Bytes Data + 32 Bytes Checksum Value + 5 Bytes Sequence Number

Broker has some fixed jobs which are done in 3 different threads. The first thread is responsible for sending packets to Destination Node. The second and third threads are responsible for listening the incoming packets from Router 1 and Router 2 nodes.

- **Listening Router1 and Router2**
  Listening Router 1 and Router 2 Nodes is a simple job. Router 1 is listened from *IP: "10.10.2.1" and Port: "8000"* and Router 2 is listened from *IP: "10.10.4.1" and Port: "8000"*. The packets coming from Router 1 and Router 2 has only Ack Data. These Ack data is used to slide the window at the "Sending Packets to Destination Node" section. More details are explained in that following section.
- **Sending Packets to Destination Node**
  IP: "10.10.3.2" and Port: "8000" is used to send packets over Router 1 node to Destination.
  IP: "10.10.5.2" and Port: "8000" is used to send packets over Router 2 node to Destination.

  Here **Multihoming** is provided by sending the packets over both Router 1 and Router 2 Nodes.

  While sending the file from the broker to destination a window is used with size 5. Our implementation was not the same as the known methods. In our implementation we used time.sleep(0.05) which works similar to timeout, but a simpler one.
  The Broker Node is sending 5 packets to Destination always, then it falls asleep for 0.05 seconds to wait for incoming ack's from Router's. If the ack value coming from Router 1 and Router 2 is indicating that these 5 packets are received then the window is moved to send the next 5 packets. Otherwise the 5 packets are sent to the Destination Node again. This is one of the main differences of our implementation then the known methods such as Go Back N. In Go Back N If the first 2 packet of the window arrives the window will be moved. But in our implementation a greater ack is waited, then the window is moved by 5 always, never a smaller value. This part provided us **Pipelining** in our implementation. Packets are not send one by one, they are sent 5 by 5 without waiting response for each of them. **Retransmission** is also done here. If an expected ack value doesn't arrive at in 0.05 seconds the 5 packets are retransmitted. They are retransmitted until a greater ack value requests for the next packet. If packet **loss** situations the packets are retransmitted. The **loss** cases are handled in this way.

C. Router1 and Router2 Nodes

After executing the commands on the "Initial Configurations" part. The Router 1 and Router 2 Nodes are ready. No codes are written for them.

D. Destination Node

The Destination Node has two threads. One is used to listen from Router 1 and send through Router 1, the second one is used to listen from Router 2 and send through Router 2.

The Destination Node is listening Router 1 on *IP: "10.10.3.2" and Port: "8000"*
The Destination Node is listening Router 2 on *IP: "10.10.5.2" and Port: "8000"*

The Destination Node is sending to Router 1 on *IP: "10.10.2.1" and Port: "8000"*
The Destination Node is sending to Router 2 on *IP: "10.10.4.1" and Port: "8000"*

Both threads do the same job in parallel. We will explain them together.

When one of the threads gets a packet from a Router Node the packet is send to a function to check if the checksum value on the packet and checksum value of the data part is same. If the checksum values are not the same than nothing done with that packet. This provide us to handle **Corruption** cases. If the checksum value is correct than the sequence number of the packet is taken and incremented by one and put in an array which also includes a window with window size of 5. Our implementation is different in a way that it doesn't send ack value for every incoming packet. It first put the ack values to an array. If the 5 packet's ack is written to the array than an ack is sent.

To demonstrate the implementation:
Destination Window [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1]
-1 indicates no ack is came yet.

Seq 0 Arrived, Ack is Written to the window. [1, -1, -1, -1, -1, -1, -1, -1, -1, -1] No ack sent yet.

Seq 1 Arrived, Ack is Written to the window. [1, 2, -1, -1, -1, -1, -1, -1, -1, -1] No ack sent yet.

Seq 2 Arrived, Ack is Written to the window. [1, 2, 3, -1, -1, -1, -1, -1, -1, -1] No ack sent yet.

Seq 3 Arrived, Ack is Written to the window. [1, 2, 3, 4, -1, -1, -1, -1, -1, -1] No ack sent yet.

Seq 4 Arrived, Ack is Written to the window. [1, 2, 3, 4, 5, -1, -1, -1, -1, -1] Ack can be sent.

The window is filled now. The ack packet will be sent to Broker with value of 5. Then the broker will slide it's window. If the ack packet of Destination lost, it sends it again and again.

The destination also fix **reordering** problems. While getting the packets from Broker Node, The packets are placed in an array with corresponding places.

To demonstrate the situation:

A packet arrives with Seq: 2 The packet's data is placed at arr[2]
A packet arrives with Seq: 0 The packet's data is placed at arr[0]
A packet arrives with Seq: 4 The packet's data is placed at arr[4]
A packet arrives with Seq: 5 The packet's data is placed at arr[5]
A packet arrives with Seq: 3 The packet's data is placed at arr[3]

We will see in the reordering experiment: the transfer time value's are the same for big and small reorder values.

IV. Experiments

In experiments part, three different experiments are examined. Each experiment is performed ten times, then these samples are used to measure mean and margin error of the experiment.

A. Experiment 1

In experiment one, packet loss percentage and file transfer time with 95% confidence intervals measured in three different configuration. The first configuration is 0.5% packet loss percentage, the second configuration is 10% packet loss percentage and the last configuration is 20% packet loss percentage. You can see the details of these three configurations in figure1.

1) Experiment 1.1: In this experiment packet loss percentage is 0.5%. Before starting the experiment, we should give the initial configuration by using following commands:

- For Broker:
  sudo tc qdisc add dev eth1 root netem loss 0.5% corrupt 0% duplicate 0% delay 3ms reorder 0% 0%
  sudo tc qdisc add dev eth3 root netem loss 0.5% corrupt 0% duplicate 0% delay 3ms reorder 0% 0%
- For Router1:
  sudo tc qdisc add dev eth1 root netem loss 0.5% corrupt 0% duplicate 0% delay 3ms reorder 0% 0%
- For Router2:
  sudo tc qdisc add dev eth2 root netem loss 0.5% corrupt 0% duplicate 0% delay 3ms reorder 0% 0%

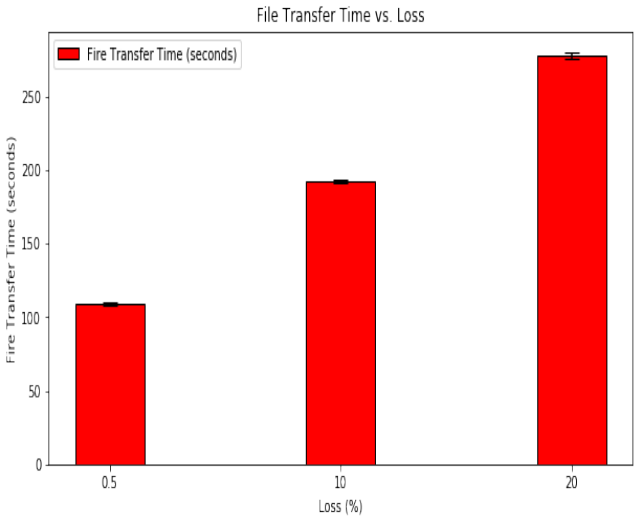

Fig. 1. Experiment One

In this experiment, sample size is ten, so we send the file ten times with same configuration and record the file transfer time of each. Sample mean of file transfer time is 109, the standard deviation of file transfer time is 1.61 and the margin error is 0.998. From these values, 95% confidence interval is 109 ± 0.998 (104 to 106).

2) Experiment 1.2: In this experiment packet loss percentage is 10%. Before starting the experiment, we should give the initial configuration by using the following commands:

- For Broker:
  sudo tc qdisc change dev eth1 root netem loss 10% corrupt 0% duplicate 0% delay 3ms reorder 0% 0%
  sudo tc qdisc change dev eth3 root netem loss 10% corrupt 0% duplicate 0% delay 3ms reorder 0% 0%
- For Router1:
  sudo tc qdisc change dev eth1 root netem loss 10% corrupt 0% duplicate 0% delay 3ms reorder 0% 0%
- For Router2:
  sudo tc qdisc change dev eth2 root netem loss 10% corrupt 0% duplicate 0% delay 3ms reorder 0% 0%

In this experiment, sample size is ten, so we send the file ten times with same configuration and record the file transfer time of each. Sample mean of file transfer time is 192, the

standard deviation of file transfer time is 1.85 and the margin error is 1.15. From these values, 95% confidence interval is 192 ± 1.15 (191 to 193).

3) Experiment 1.3: In this experiment packet loss percentage is 20%. Before starting the experiment, we should give the initial configuration by using the following commands:

- For Broker:
  sudo tc qdisc change dev eth1 root netem loss 20% corrupt 0% duplicate 0% delay 3ms reorder 0% 0%
  sudo tc qdisc change dev eth3 root netem loss 20% corrupt 0% duplicate 0% delay 3ms reorder 0% 0%
- For Router1:
  sudo tc qdisc change dev eth1 root netem loss 20% corrupt 0% duplicate 0% delay 3ms reorder 0% 0%
- For Router2:
  sudo tc qdisc change dev eth2 root netem loss 20% corrupt 0% duplicate 0% delay 3ms reorder 0% 0%

In this experiment, sample size is ten, so we send the file ten times with same configuration and record the file transfer time of each. Sample mean of file transfer time is 278, the standard deviation of file transfer time is 3.41 and the margin error is 2.11. From these values, 95% confidence interval is 278 ± 2.11 (276 to 280).

B. Experiment 2

In experiment two, corruption percentage and file transfer time with 95% confidence intervals measured in three different configuration. The first configuration is 0.2% corruption percentage, the second configuration is 10% corruption percentage and the last configuration is 20% corruption percentage. You can see the details of these configurations in figure2.

1) Experiment 2.1: In this experiment corruption percentage is 0.2%. Before starting the experiment, we should give the initial configuration by using the following commands:

- For Broker:
  sudo tc qdisc change dev eth1 root netem loss 0% corrupt 0.2% duplicate 0% delay 3ms reorder 0% 0%
  sudo tc qdisc change dev eth3 root netem loss 0% corrupt 0.2% duplicate 0% delay 3ms reorder 0% 0%
- For Router1:
  sudo tc qdisc change dev eth1 root netem loss 0% corrupt 0.2% duplicate 0% delay 3ms reorder 0% 0%
- For Router2:
  sudo tc qdisc change dev eth2 root netem loss 0% corrupt 0.2% duplicate 0% delay 3ms reorder 0% 0%

In this experiment, sample size is ten, so we send the file ten times with same configuration and record the file transfer time of each. Sample mean of file transfer time is 106, the standard deviation of file transfer time is 0.725 and the margin error is 0.449. From these values, 95% confidence interval is 106 ± 0.449 (106 to 106).

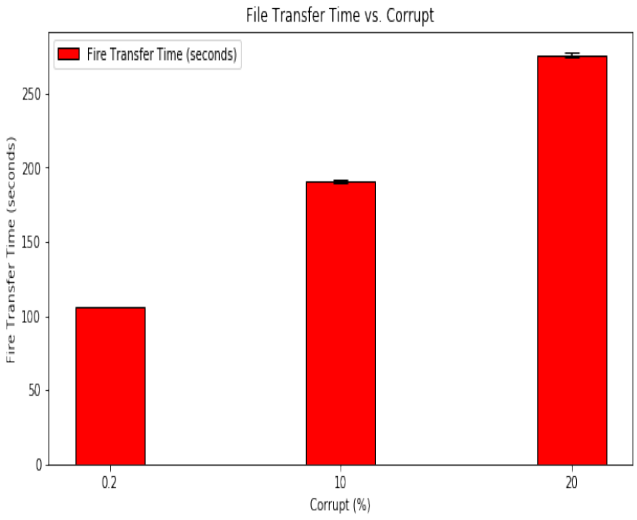

Fig. 2. Experiment Two

2) Experiment 2.2: In this experiment corruption percentage is 10%. Before starting the experiment, we should give the initial configuration by using the following commands:

- For Broker:
  sudo tc qdisc change dev eth1 root netem loss 0% corrupt 10% duplicate 0% delay 3ms reorder 0% 0%
  sudo tc qdisc change dev eth3 root netem loss 0% corrupt 10% duplicate 0% delay 3ms reorder 0% 0%
- For Router1:
  sudo tc qdisc change dev eth1 root netem loss 0% corrupt 10% duplicate 0% delay 3ms reorder 0% 0%
- For Router2:
  sudo tc qdisc change dev eth2 root netem loss 0% corrupt 10% duplicate 0% delay 3ms reorder 0% 0%

In this experiment, sample size is ten, so we send the file ten times with same configuration and record the file transfer time of each. Sample mean of file transfer time is 191, the standard deviation of file transfer time is 1.38 and the margin error is 0.855. From these values, 95% confidence interval is 191 ± 0.855 (190 to 192).

3) Experiment 2.3: In this experiment corruption percentage is 20%. Before starting the experiment, we should give the initial configuration by using the following commands:

- For Broker:
  sudo tc qdisc change dev eth1 root netem loss 0% corrupt 20% duplicate 0% delay 3ms reorder 0% 0%
  sudo tc qdisc change dev eth3 root netem loss 0% corrupt 20% duplicate 0% delay 3ms reorder 0% 0%
- For Router1:
  sudo tc qdisc change dev eth1 root netem loss 0% corrupt

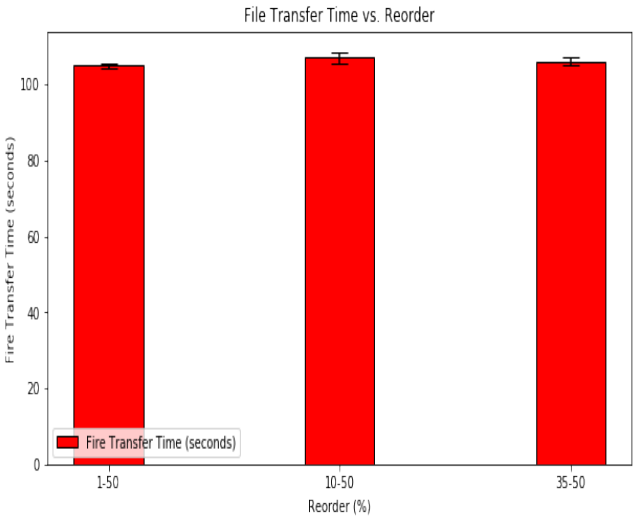

Fig. 3. Experiment Three

20% duplicate 0% delay 3ms reorder 0% 0%

- For Router2:
  sudo tc qdisc change dev eth2 root netem loss 0% corrupt 20% duplicate 0% delay 3ms reorder 0% 0%

In this experiment, sample size is ten, so we send the file ten times with same configuration and record the file transfer time of each. Sample mean of file transfer time is 276, the standard deviation of file transfer time is 2.88 and the margin error is 1.79. From these values, 95% confidence interval is 276 ± 1.79 (274 to 278).

C. Experiment 3

In experiment three, reordering percentage and file transfer time with 95% confidence intervals measured in three different configuration. The first configuration is 1% reordering percentage, the second configuration is 10% reordering percentage and the last configuration is 35% reordering percentage. You can see the details of these configurations in figure3.

1) Experiment 3.1: In this experiment reordering percentage is 1%. Before starting the experiment, we should give the initial configuration by using the following commands:

- For Broker:
  sudo tc qdisc change dev eth1 root netem loss 0% corrupt 0% duplicate 0% delay 3ms reorder 1% 50%
  sudo tc qdisc change dev eth3 root netem loss 0% corrupt 0% duplicate 0% delay 3ms reorder 1% 50%
- For Router1:
  sudo tc qdisc change dev eth1 root netem loss 0% corrupt 0% duplicate 0% delay 3ms reorder 1% 50%
- For Router2:
  sudo tc qdisc change dev eth2 root netem loss 0% corrupt 0% duplicate 0% delay 3ms reorder 1% 50%

In this experiment, sample size is ten, so we send the file ten times with same configuration and record the file transfer time of each. Sample mean of file transfer time is 105, the standard deviation of file transfer time is 0.934 and the margin error is 0.579. From these values, 95% confidence interval is 105 ± 0.579 (104 to 106).

2) Experiment 3.2: In this experiment reordering percentage is 10%. Before starting the experiment, we should give the initial configuration by using the following commands:

- For Broker:
  sudo tc qdisc change dev eth1 root netem loss 0% corrupt 0% duplicate 0% delay 3ms reorder 10% 50%
  sudo tc qdisc change dev eth3 root netem loss 0% corrupt 0% duplicate 0% delay 3ms reorder 10% 50%
- For Router1:
  sudo tc qdisc change dev eth1 root netem loss 0% corrupt 0% duplicate 0% delay 3ms reorder 10% 50%
- For Router2:
  sudo tc qdisc change dev eth2 root netem loss 0% corrupt 0% duplicate 0% delay 3ms reorder 10% 50%

In this experiment, sample size is ten, so we send the file ten times with same configuration and record the file transfer time of each. Sample mean of file transfer time is 107, the standard deviation of file transfer time is 2.38 and the margin error is 1.48. From these values, 95% confidence interval is 107 ± 1.48 (106 to 108).

3) Experiment 3.3: In this experiment reordering percentage is 35%. Before starting the experiment, we should give the initial configuration by using the following commands:

- For Broker:
  sudo tc qdisc change dev eth1 root netem loss 0% corrupt 0% duplicate 0% delay 3ms reorder 35% 50%
  sudo tc qdisc change dev eth3 root netem loss 0% corrupt 0% duplicate 0% delay 3ms reorder 35% 50%
- For Router1:
  sudo tc qdisc change dev eth1 root netem loss 0% corrupt 0% duplicate 0% delay 3ms reorder 35% 50%
- For Router2:
  sudo tc qdisc change dev eth2 root netem loss 0% corrupt 0% duplicate 0% delay 3ms reorder 35% 50%

In this experiment, sample size is ten, so we send the file ten times with same configuration and record the file transfer time of each. Sample mean of file transfer time is 106, the standard deviation of file transfer time is 1.64 and the margin error is 1.02. From these values, 95% confidence interval is 106 ± 1.02 (105 to 107).

V. Observations

A. Observations on Experiment One

In the experiment one, there are three configurations and while packet loss percentage increase, file transfer time also

increase depends on packet loss percentage. When packet loss percentage is 0.1%, the file transfer time is 109 ± 0.998. Then packet loss percentage increase to 10%, the file transfer time becomes 192 ± 1.15. At the end, packet loss percentage becomes 20% and file transfer time becomes 278 ± 2.11. File transfer time increase linearly depending on the packet loss percentage.

B. Observations on Experiment Two

In the experiment two, there are three configurations and while corruption percentage increase, file transfer time also increase depending on corruption percentage. When corruption percentage is 0.2%, the file transfer time is 106 ± 0.449. Then corruption percentage increase to 10%, the file transfer time becomes 191 ± 0.855. At the end, corruption percentage becomes 20% and file transfer time becomes 276 ± 1.79. File transfer time increase linearly depending on the corruption percentage.

1) Observations on Experiment Three: In the experiment three, there are three different configurations and changing on reordering percentage does not affect file transfer time because as mentioned in Destination Node section the packets are placed at the correct places when they arrive.

A packet arrives with Seq: 2 The packet's data is placed at arr[2]
A packet arrives with Seq: 0 The packet's data is placed at arr[0]
A packet arrives with Seq: 4 The packet's data is placed at arr[4]
A packet arrives with Seq: 5 The packet's data is placed at arr[5]
A packet arrives with Seq: 3 The packet's data is placed at arr[3]

VI. Conclusion

We designed a reliable data transfer protocol over a UDP connection. We did three experiments to observe packet loss, packet corruption and packet reordering cases. While first two cases slows our protocols down, the third case didn't impact on our file transfer time because we handled that case at Destination node.