Term Project

1.Steps

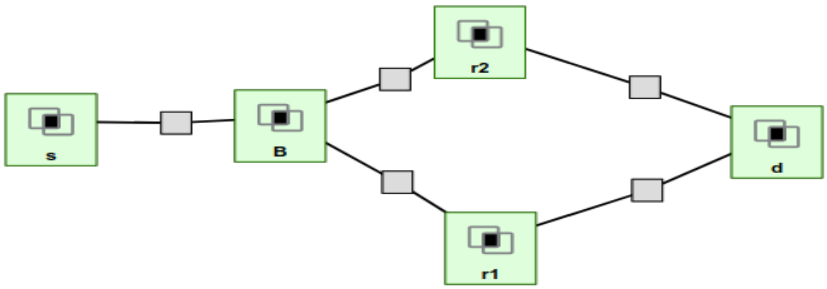
We followed steps below in this assignment. Some of these processes come from our First Term Project.

1. Using our Geni account which we register before for Project 1, we create a new slice using new XML file with button (+New Slice and +Add Resources)
2. We try to understand our Topology which is given in ODTUCLASS. 3. After our resources in geni being ready, we try to decide how we can implement our Reliable Data Transfer
4. By using Linux route add/del commands, we create our Routing Table
5. We try to implement Socket Programming Codes in Python that we write to our RDT
6. We transfer the code files into Virtual Machine.Then, we try to test RDT and default(starting) of transfer time
7. We used TC Netem Commands
8. We get statistics from experiments and creat figures.

2.GENI Platform

Although we were already familiar with the Geni platform due to the 1st Project, we had some problems connecting to resources for the first time after creating the new slice.However, we did not experience any problems in our later trials.In addition to this, while we were creating slice with xml file without changing, we had trouble in routing table. However, when we upload a new xml, we overcome this problem.

3.About Our topology



There are four nodes, which are source ,broker, router, destination respectively.

s : UDP source that sends packets also receives responses

b : sends the data received from source (s) to the destination (d) via the routers (r1 and r2) in a random way.

r : transmits the data in the direction it should go(from s to d or vice versa). Also it has routing tables.

d : is used for listening routers and receiving packets then send ack

In our project , For the given topology we create a new RDT. The designed RDT has pipeling feature (the second packet don't have to wait for first packet to be transmitted) onto the UDP data transfer. Steps of our design is below;

1.Data is divided into frames ,which is consist of 16 bytes pieces.

2.The window ,which consists of 100 frames,is sent.

3.Each of packet consists of frame, frame number,and frame's checksum.Before the frame is sent,we pack it.

4.The current window has a checklist which initializes elementa as 'FALSE'.

5.The window's framea are sent individually, the frames' ACKs are waited.

6.When the ACK data of a frame is 1 and accepted in the specific time limit , which is 1 ms ,mark the element of checklist as TRUE related to the frame no.

7.When the ACK data of a frame is 0 or it did not accept the time limit, the checklist element's stays as FALSE related to the frame number.

8.The respond corresponding frame of FALSE element when there is no FALSE in the checklist.

9.The checklist is TRUE , this process is being repeated as far as sending all cuurent window frames' and receiving correct every frames' ACK.

10.If there is any FALSE element in the checklist , it sends the next window.

4.Routing Table

RT for Source

```
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         172.16.0.1      0.0.0.0         UG    0      0        0 eth0
10.10.1.0       0.0.0.0         255.255.255.0   U     0      0        0 eth1
10.10.2.0       10.10.1.2       255.255.255.0   UG    0      0        0 eth1
10.10.3.0       10.10.1.2       255.255.255.0   UG    0      0        0 eth1
10.10.4.0       10.10.1.2       255.255.255.0   UG    0      0        0 eth1
10.10.5.0       10.10.1.2       255.255.255.0   UG    0      0        0 eth1
172.16.0.0      0.0.0.0         255.240.0.0     U     0      0        0 eth0
```

RT for Broker

```
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         172.16.0.1      0.0.0.0         UG    0      0        0 eth0
10.10.1.0       0.0.0.0         255.255.255.0   U     0      0        0 eth3
10.10.2.0       0.0.0.0         255.255.255.0   U     0      0        0 eth1
10.10.3.0       10.10.2.2       255.255.255.0   UG    0      0        0 eth1
10.10.4.0       0.0.0.0         255.255.255.0   U     0      0        0 eth2
10.10.5.0       10.10.4.2       255.255.255.0   UG    0      0        0 eth2
172.16.0.0      0.0.0.0         255.240.0.0     U     0      0        0 eth0
```

RT for Router1

```
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         172.16.0.1      0.0.0.0         UG    0      0        0 eth0
10.10.1.0       10.10.2.1       255.255.255.0   UG    0      0        0 eth2
10.10.2.0       0.0.0.0         255.255.255.0   U     0      0        0 eth2
10.10.3.0       0.0.0.0         255.255.255.0   U     0      0        0 eth1
172.16.0.0      0.0.0.0         255.240.0.0     U     0      0        0 eth0
```

RT for Router2

```
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         172.16.0.1      0.0.0.0         UG    0      0        0 eth0
10.10.1.0       10.10.4.1       255.255.255.0   UG    0      0        0 eth2
10.10.4.0       0.0.0.0         255.255.255.0   U     0      0        0 eth2
10.10.5.0       0.0.0.0         255.255.255.0   U     0      0        0 eth1
172.16.0.0      0.0.0.0         255.240.0.0     U     0      0        0 eth0
```

RT for Destination

```
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         172.16.0.1      0.0.0.0         UG    0      0        0 eth0
10.10.1.0       10.10.3.1       255.255.255.0   UG    0      0        0 eth2
10.10.1.0       10.10.5.1       255.255.255.0   UG    0      0        0 eth1
10.10.2.0       10.10.3.1       255.255.255.0   UG    0      0        0 eth2
10.10.3.0       0.0.0.0         255.255.255.0   U     0      0        0 eth2
10.10.4.0       10.10.5.1       255.255.255.0   UG    0      0        0 eth1
10.10.5.0       0.0.0.0         255.255.255.0   U     0      0        0 eth1
172.16.0.0      0.0.0.0         255.240.0.0     U     0      0        0 eth0
```

4.Programming Part

We used main two modules for socket programming.
Because of any limitation, we use Python as a programming language instead of C++ because it is easier on Python to encode socket programming for us.

We edit our Python Codes in vim editor instead of nano because of easy to use.

We used 'from socket import socket, AFINET, SOCKDGRAM, timeoutfrom sys import argv from common import ipchecksum' for source.py

We used 'from socket import import random import time' for broker.py

We used 'from socket import socket, AFINET, SOCKDGRAM from sys import argv, stdout from common import ipchecksum' for dest.py

1.Time : While sending data

2.Socket: While sending message and creating socket, we used this module.

3.Random: To generate random number to sent to routers randomly

4.Threading: Since we need simultaneous access for broker,we had to use concurrency. That's why while broker listens two socket which one receive packet, to handle this task we used thread function in this module.

5.Uploading codes into virtual machines

We connected our nodes on GENI by using SSH. We write our codes on Local and then we implement them on nodes(etc. s..) via Vim Editor instead of nano

6 Test communication

We cannot synchronize time so we test our time data on local platform. Therefore we send 1000 data and divide total delay by 1000.

7 Using Netem Commands

We used TC Netem Commands while we create routing table like;

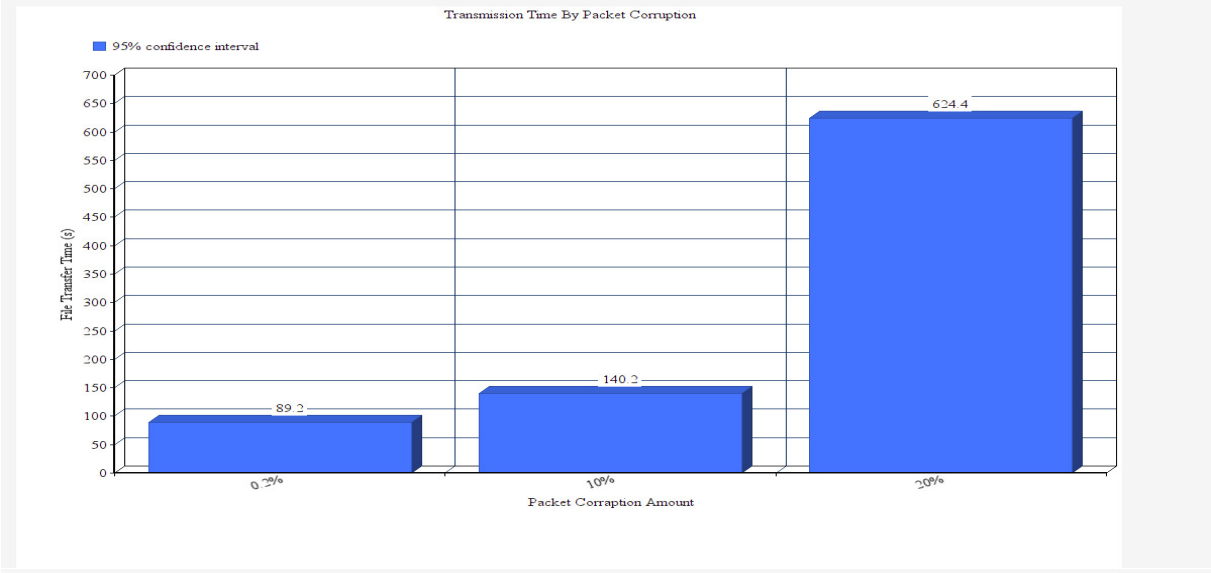
in order to add we use;

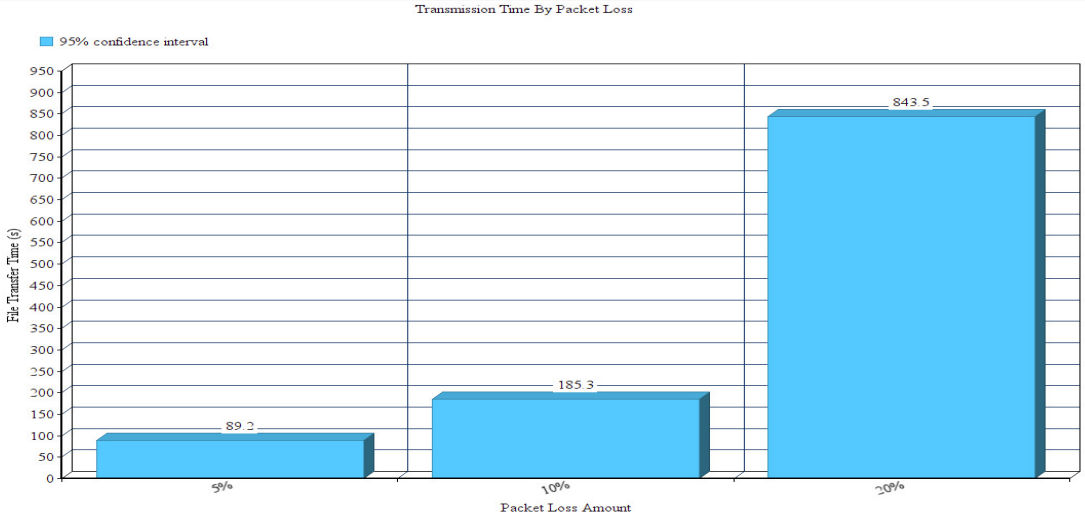
sudo route add -net 10.10.2.0 netmask 255.255.255.0 gw 10.10.1.2 dev eth1

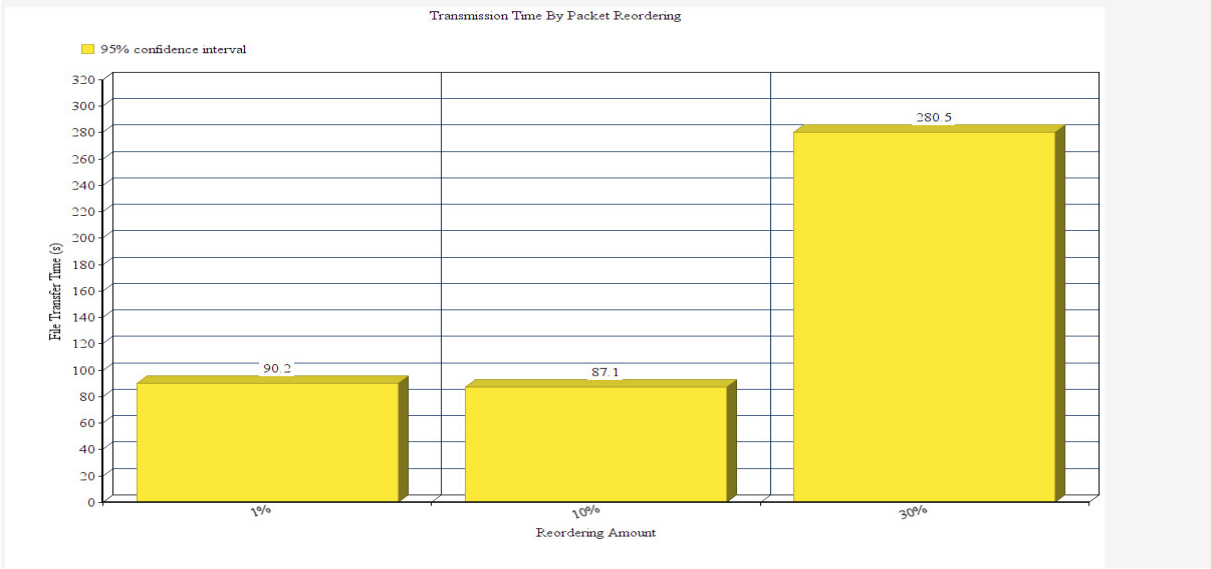
and to del;

sudo route dell -net 10.10.2.0 netmask 255.255.255.0 gw 10.10.1.2 dev eth1..

1 **7 Graphs for Experiment**


Transmission Time By Packet Corruption
95% confidence interval
File Transfer Time (s)
700
650
600
550
500
450
400
350
300
250
200
150
100
50
0
89.2
140.2
624.4
0 2%
10%
20%
Packet Corruption Amount


Transmission Time By Packet Loss
95% confidence interval
File Transfer Time (s)
950
900
850
800
750
700
650
600
550
500
450
400
350
300
250
200
150
100
50
0
89.2
185.3
843.5
5%
10%
20%
Packet Loss Amount


Transmission Time By Packet Reordering
95% confidence interval
File Transfer Time (s)
320
300
280
260
240
220
200
180
160
140
120
100
80
60
40
20
0
90.2
87.1
280.5
1%
10%
30%
Reordering Amount