CENG435 Data Communications and Networking Term Project Part 1

Misranur YAZGAN
Department of Computer Engineering
Middle East Technical University
Ankara, Turkey
misranur.yazgan@ceng.metu.edu.tr

Elif KAYA
Department of Computer Engineering
Middle East Technical University
Ankara, Turkey
elif.kaya@ceng.metu.edu.tr

***Abstract*—This paper introduces an application of the network socket programming with the purpose of transmitting packets from a source to a destination over a network which has a number of intermediate nodes. The services provided by the transport layer protocols, namely Transmission Control Protocol(TCP) and User Datagram Protocol(UDP), are both analyzed and implemented in the project. In order to understand the effect of network emulation delay on the end-to-end delay over a network, we conducted a number of experiments by manipulating the network emulation delay on the links between the nodes. To serve this purpose, we exploited netem/tc Linux commands which provides the network emulation functionality for experimental purposes.**

***Index Terms*—Network Socket Programming, Client-Server Architecture, Transport Layer Protocols, Network Emulation Delay, End-to-End Delay**

I. Introduction

This paper aims to highlight the importance of the delays introduced to the links in a network over the end-to-end delay along with designing an overall network structure by implementing different transport layer protocols namely TCP and UDP.

The project concentrates on the implementation of a network which allows the transmission of the packets from source to destination over some intermediate nodes. There are mainly five nodes which are listed as the following:

- Source
- Broker
- Router1
- Router2
- Destination

The communication between the Source node and the Broker node is established through TCP whereas the communication between the Broker, routers and the Destination nodes is provided through UDP.

II. Background Information

A. The Differences Between TCP and UDP

TCP employs three-way handshake in order to provide a reliable transport. Furthermore, it also provides flow control in a way that ensuring the Source does not overload the network by sending packets faster than the Broker can handle. It provides congestion control to prevent message traffic from slowing down the network response time for the general welfare of the network.

On the other hand, UDP does not provide a reliable data transmission due to the fact that it does not employ any handshake and directly sends the data to the given address regardless of whether the server is alive or not.

B. Importance of the Broker Node

Broker is the node where the packets coming from the Source are received through TCP sockets and forwarded to the next hop addresses through UDP sockets. Broker has the ability to send packets through multiple links. In our project, we implement this ability in a way that the Broker forwards the packets to two different routers alternatingly. The Broker plays a significant role in the overall network structure since it both acts like a server by listening to the Source node and acts like a router by implementing the store-and-forward logic.

III. Design and Implementation Approach

A. Message Structure

We have designed our messages to consist of 39 bytes, the first 25 bytes represent the original message part, the 26th byte is a delimiter and the rest of the bytes are representing the timestamp to be used later in the Destination node for the purpose of calculating end-to-end delay. We have sent 100 messages in this format. Therefore, our messages has the following form:

```
Current temperature: ** C;**********
```

We have a list consisting of 10 elements and in the s.py we multiply it by 10 to have 100 messages.

B. GENI Platform

We have used the virtual networks on GENI Platform and created a slice using the aggregate University of Texas InstaGENI. We have downloaded and used the topology file in the XML format as it is given in the project requirements. The topology has created the nodes that are mentioned in the Introduction part. Then, we created and downloaded an SSH Key. After these steps, we connected to the University of Texas InstaGENI through ssh(related commands are explained in README.txt file). We have used Secure File Transfer Protocol(SFTP) to get the files from our local environments.

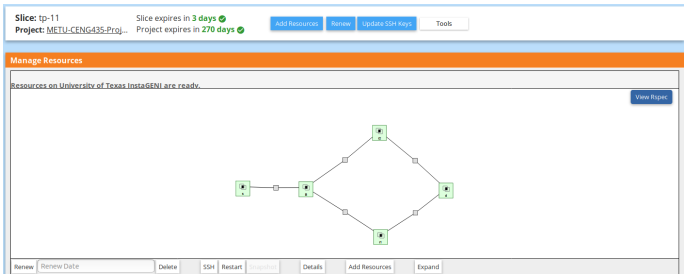

Fig. 1. GENI Platform Resources View

C. Source Node

We have designed s node as a source device which employs Transmission Control Protocol (TCP) based socket application. In s.py we have implemented this node based on our design. In the s.py file we have a list named "measurements" which carries messages.

Definitions of the functions in s.py:

- packetizeMsg(message): it takes the current time from the function datetime.now() and changes its format for the ease of use. In the final step, message + "; " + currenttime is returned.
- connection(): It creates a TCP socket and connects it to the Broker's address. Broker's address was found from the GENI slice. This action starts the three-way-handshake protocol. In addition, socket.connect() method might fail if the Broker is not running at that moment. In a for loop we send each of the messages (which is ten times measurements list = 100 packets) to the socket and increment packet number by one.

D. Broker Node

We have designed b node as a broker device which employs both User Diagram Protocol(UDP) (between destination and broker) and Transmission Control Protocol (TCP)(between source and broker) based socket applications. In b.py we have implemented this node based on our design.

Definitions of the functions in b.py:

- receiveMsg(client, sockbr1, sockbr2, turn): It receives the messages from Source Node and send them to the router1 or router2 according to the variable named as 'turn'. Turn is an integer used for the purpose of sending messages to routers sequentially (If turn is even, send it to the Router2; if turn is odd, send it to Router1).
- connection(): It creates both TCP and UDP sockets with the related node's address; creates TCP socket with Source nodes address which was found from the GENI slice and binds it, creates UDP sockets (namely sockbr1 and sockbr2) with router nodes addresses which were also found from the GENI slice. After creating sockets it accepts the TCP connection request from the Source node (three-way-handshake protocol). In a while loop it calls receiveMsg for receiving each of the messages from the Source node and sends them to the routers and increment turn by one to alternate between the routers.

E. Router Nodes

We have designed r1 and r2 nodes as router devices which employ User Datagram Procotol (UDP) based socket application. In r1.py and in r2.py we have implemented these nodes based on our design.

Definitions of the functions in r1.py and r2.py:

- receiveMsg(client, sockr1d): It receives messages from the Broker Node and sends them to the Destination node.
- connection(): It creates a UDP sockets with Broker node's address and binds it; creates UDP socket with Destination node's address which was found from the GENI slice. After creating the sockets, in a while loop it calls receiveMsg for receiving each of the messages from Broker node and sends them to the Destination node. Calling receiveMsg also handles the forwarding of the messages.

F. Destination Node

We have designed d node as a destination device which employs User Datagram Procotol (UDP) based socket application.In d.py we have implemented the node based on our design.

Definitions of the functions in d.py:

- totalTranmissionTime(received_message, packet_number): It is the part that unpacks the pack according to the message format. It gets the timestamp created in Source Node and calculates the end-to-end delay. Function prints the packet number and delay.
- receiveMsg(client, sockr1d): It receives messages from one of the router nodes by select function(it selects the node with a ready packet), calls totalTransmissionTime to calculate end-to-end delay, and increments packet_number by one.
- connection(): It creates a UDP socket with router node's address and binds them with router node's address which were found from the GENI slice. After creating the sockets, it calls receiveMsg function to perform the related operations.

IV. Experiments and Results

For the experimental part of the project, we aimed to observe how manipulating the network emulation delay on the links affects the end-to-end delay over the network. In order to conduct the experiments, we have utilized a series of netem/tc Linux commands which provides facilities to control the network traffic such as introducing delay, packet loss and other characteristics to the packets.

A. Configuration of the Links

In our project, we needed to manipulate the network emulation delay over all the links between the Broker node and the Destination node. These links are specified as the following:

1) The link between the Broker and the Router1
2) The link between the Broker and the Router2
3) The link between Router1 and the Destination
4) The link between Router2 and the Destination

In order to find out which links to configure in the nodes, we checked the relevant interfaces and IP addresses of the resource nodes that we have reserved on the Geni Portal. After that, we typed the following command on the Broker, Router1 and Router2 nodes:

```
ifconfig
```

In this way, we learned which links to configure on the nodes for changing the network emulation delay. The corresponding names of the links shown above in our resource nodes are given as the following:

1) eth2 in Broker
2) eth3 in Broker
3) eth2 in Router1
4) eth2 in Router2

We performed three different experiments by introducing delays to the links mentioned above with the following configurations:

1) $1ms \pm 5ms$ with normal distribution
2) $20ms \pm 5ms$ with normal distribution
3) $60ms \pm 5ms$ with normal distribution

Having decided the correct links to configure, we used the following command to introduce delays on a link:

```
sudo tc qdisc <add/change> dev <link> root
    netem delay <delay> 5ms distribution
    normal
```

where

- $< add/change >$: If the delay is introduced to the link for the first time add command is used, otherwise $change$ is used.
- $< link >$: The name of the link we learned with the command $ifconfig$, e.g. eth2, eth3.
- $< delay >$: One of $1ms$, $20ms$ or $60ms$ for the related experiment.

B. The correlation between the Network Emulation Delay and the End-to-End Delay

To achieve the proper results, we conducted each of the experiments many times and took the mean of the results to calculate end-to-end delay.

In our packets, we added timestamps to the original messages in order to indicate the time that the message was sent. Then, we calculated the total transmission time of a packet by using these timestamps.

The following graph in Figure 2 shows the correlation between the network emulation delay and the end-to-end delay for each of the packets(100 packets in our case).

The blue, red and yellow lines correspond to the experiments with $1ms$ delay, $20ms$ delay and $60ms$ delay respectively. The horizontal axis shows the number of packets from 1 to 100 while the vertical axis shows the end-to-end delay in milliseconds. We can see that the more the network emulation delay is added to the links, the more the end-to-end delay increases. In addition, we can infer from the graph that as

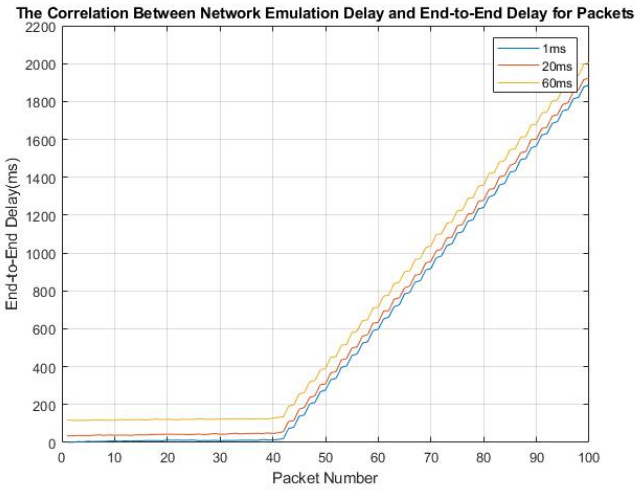

Fig. 2. The correlation between the network emulation delay and the end-to-end delay for each packet

the packet number increases, the end-to-end delay increases after a certain amount of packets(40 packets) for each different configuration for the network emulation delay.

The second graph which is given below in Figure 3 shows the correlation between the network emulation delay and the end-to-end delay for one packet.

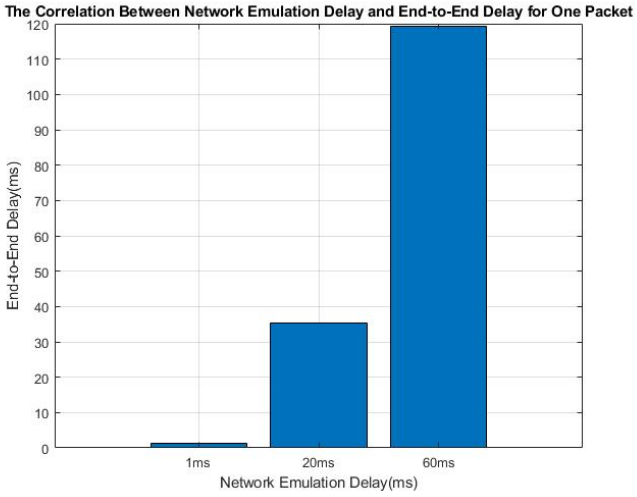

Fig. 3. The correlation between the network emulation delay and the end-to-end delay for one packet

As it is shown in the bar graph, end-to-end delay increases as the network emulation delay on the links increases. For the $1ms$ delay, end-to-end delay is approximately $2ms$; for the $20ms$ delay, end-to-end delay is approximately $40ms$ and for the $60ms$ delay, end-to-end delay is approximately $120ms$ due to the fact that between Broker and Destination there are two links that we have introduced delay.

V. Conclusion

This paper presented an overall network design in order to transfer packets from a source to a destination by making use of both the broker logic and the router logic.

We have conducted several experiments with the purpose of observing the effects of delays in the links on the overall transmission time from one end-point to another. Therefore, as a result of our experiments we concluded that the transmission time of a packet from source to destination depends significantly on the delays of the links. Furthermore, as the delays caused by the links are increased, the end-to-end delay increases proportionally.

References