CENG435 Term Project Part-2 Report

Simge Nur Çankaya
Computer Engineering
Middle East Technical University
Ankara, Turkey
Student ID : 2099554
Group ID : 36
simgenurcankaya@gmail.com

Muhammed Süha Demirel
Computer Engineering
Middle East Technical University
Ankara, Turkey
Student ID : 2098911
Group ID : 36
msuhademirel@gmail.com

***Abstract*—In this project, we are expected to use the topology of part-1 and that topology consist one source, one destination and three router nodes. The topology will be discussed in the following chapters more detailed. Moreover, a UDP-based "Reliable Data Transfer" (RDT) protocol of our own that supports pipelining and multi-homing. We are also expected to modify your routing implementations based on the requirements of the file transmission. Our protocol should be implemented by considering our own approach and design.**

***Index Terms*—UDP, RDT, multi-homing, pipelining, Node, IP, Port, Socket, Request, Routing Logic, End-to-End Delay**

I. Specifications

- Implementing a UDP-based "Reliable Data Transfer" socket application with pipelining and multihoming.
- All nodes should handle the links in a reliable fashion.
- Assuming that a sender and receiver as an application layer implementation on the node "s" and the node "d" are running.
- Our source node "s" will send this large file (exactly 5 MBytes) to the destination node "d".

II. Introduction

The second part of the project, the slice has taken from the Global Environment for Network Innovations(GENI) platform as we have done in the part one. The hosts are :

- 1 source node
- 3 router nodes
- 1 destination node

UDP-based "Reliable Data Transfer" (RDT) protocol of our own that supports pipelining and multi-homing implemented between five nodes. UDP connection is implemented between these five nodes.

III. Design & Implementation

The topology of a network is already given and its visual examination is shown Figure 1.

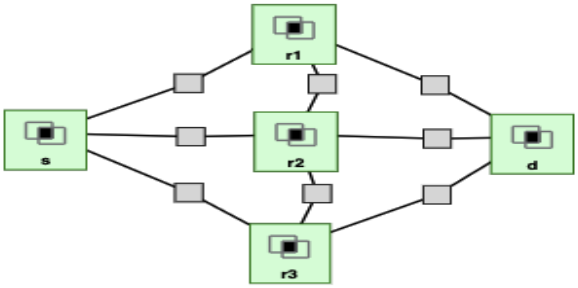

Fig. 1. The Structure of the Topology

We used Colorado Instageni Site to take slice and connect VMs. In order to connect VMs via ssh, we need to generate ssh keys(public and private keys). After the generation of keys, we need to add this key to our .ssh directory. Connection for a VM via ssh is established with the following code : `ssh -i [SSH KEY_DIRECTORY] e2099554@pc1.instageni.cs.colorado.edu -p [PORT_NUMBER]`. When the connection is established we are able to use our VM. To copy a file from our local machine to VM, we used the following code: `scp -i [SSH KEY DIRECTORY] -P [PORT_NUMBER] [FILE DIRECTORY] e2099554@pc1.instageni.cs.colorado.edu:[COPY DIRECTORY].`

We used GitHub for team communication and written our scripts with Python 2.

The topology of the network is imported to GENI Platform via .xml file. Also, we were given configuration scripts,which are shown below, to implement into all nodes of the topology via ssh. These scripts add determined delay which is equal to 3ms and three different packet loss percentage (%5,%15,%38) to links to the node that they run on.

The `configuration scripts`:

- tc qdisc change dev [INTERFACE] root netem loss 5% delay 3ms
- tc qdisc change dev [INTERFACE] root netem loss 15% delay 3ms
- tc qdisc change dev [INTERFACE] root netem loss 38% delay 3ms

Before the explanation of all of the developments and implementations, strategies, approaches, their reasons for our own protocol; we will describe briefly some key terms and mechanisms.

A. Reliable Data Transfer

The data will be transferred should :

- have no bit errors
- have no packet loss
- be delivered in the same sequence as they were sent to below layer.

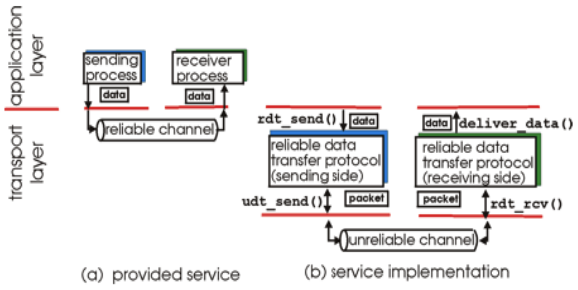

Fig. 2. The RDT Principles

B. Checksum

To detect errors (e.g. flipped bits) in transmitted segment is the goal.

Sender treats segment contents, including header fields, as sequence of 16-bits integers. Addition(one's complement sum) of the segment contents give the checksum and checksum value is put into UDP checksum field by the user.

Receiver computes the checksum of the received segment and check if computed checksum equals checksum field value.

C. ACK & NAK

To recover from errors in packets, receiver explicitly tells sender that packet received:

- is OK with acknowledgements(ACKs)
- had errors with negative acknowledgements(NAKs)

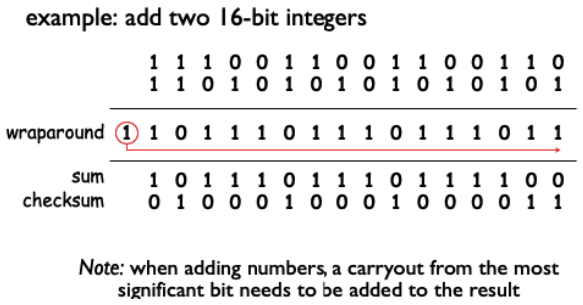

Fig. 3. The Checksum Example

D. Timeout

Receiver can not know if its last ACK/NAK received OK at sender. In order to solve this problem, sender waits "reasonable" amount of time for ACK and then:

- Retransmits if no ACK received in this time
- If packet (or ACK) just delayed (not lost): Retransmission will be duplicate, but sequence number's already handles this receiver must specify sequence number of packet being ACKed.
- requires countdown timer.

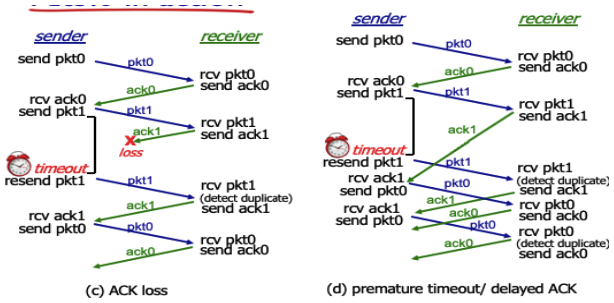

Fig. 4. The Timeout

E. Pipelined Protocols

Sender allows multiple yet-to-be-acknowledged packets. Two generic forms of pipelined protocols:

- Go-Back-N: Sender can have up to N unACKed packets in pipeline. Receiver only sends cumulative ACKs; doesn't send ACK packet if there's a gap. Sender has timer for oldest unACKed packet, when timer expires, retransmits all unACKed packets.
- Selective Repeat: Sender can have up to N unACKed packets in pipeline. Receiver sends individual ACK for each packet. Sender maintains timer for each unACKed packet. When timer expires, retransmit only that unACKed packet.

F. Multihoming

Multihoming is a mechanism used to configure one computer with more than one network interface and multiple IP

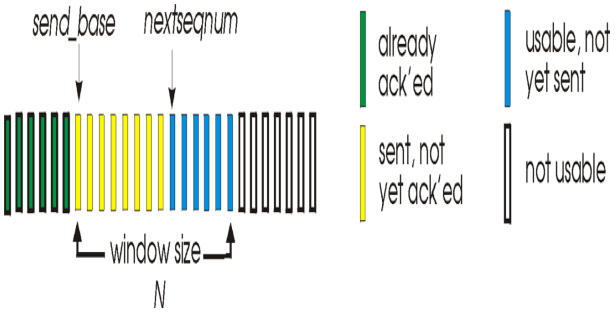

Fig. 5. The Go-Back-N

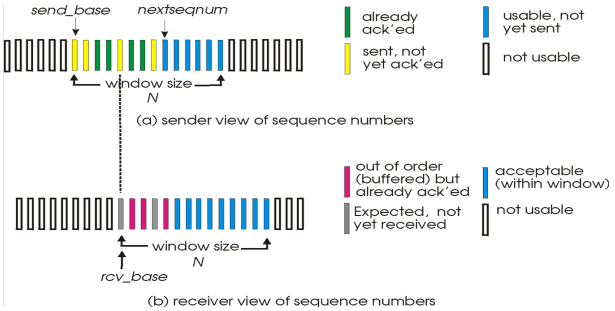

Fig. 6. The Selective Repeat

addresses. It provides enhanced and reliable Internet connectivity without compromising efficient performance. The multihoming computer is known as the host and is directly or indirectly connected to more than one network.

G. Our Design Strategies

IV. Experiment Specifications

- The file transmission is performed for two experiments: The source node will send the specified file to the destination node by using two different paths by conducting two different experiments:
  - The first path will be the shortest one based on the Dijkstra Algorithm that we found in the first part, and the second one will use the other available links "at the same time". These two different transmissions should be handled by using the same script. You can give the number of experiments as a parameter to your scripts and run them for two experiments.
  - For Experiment 1, our hosts (nodes) should route each of the packet based on the shortest path. This means that our file transmission will be over the shortest-path from the source node (s) to the destination node (d). The naming convention for the file can be for this transmission like input1 on the node "s" and output1 on the node "d".
  - In Experiment 2, the source node exploits the remaining available links of (two) for a copy of this input file. The naming convention for the file can be like input2 on the node "s" and output2 on the node "d" for this transmission.
    - Exploit disjoint links between the source and destination. We will use the advantage of two links while transferring the file, exploiting these multiple paths will allow us to transfer the file faster. This part will provide us a multi-homed protocol that is expected to implement.
    - Assume that the shortest path is s-r3-d for experiment 1, and, then the second file transmission (for experiment 2) routing information should be like the following: (Use these paths at the same time) for the first path: s-r1-d and the second path s-r2-d, and for response messages also follow the inverse of these paths. The links between the r1 and r2; r2 and r3 will not be used.
    - The network may be faced with one type of failure which is link failure. One of the links between the source node "s" and two other nodes (r1 and r2) can be down. Our reliable and multi-homed implementation should be aware of the links and update the routing and modifying the reliable transmission over the available link. This part will be applied only for the second experiment (Experiment 2).
- For two experiments:
  - The files will be protected with a checksum in the assessment. In other words, the input file at the source side, and the transmitted file at the destination side should be exactly the same.
  - We will implement and develop our own RDT protocol on top UDP. Therefore, we will use our own reliable protocol to send a large file from s to d. Then, the communications will be based on our RDT protocol.
  - Develop our reliable transport protocol that supports multi-homing by using UDP sockets.
  - Develop a packet-based protocol.
  - Design our packet structure that will go into the UDP payload. The maximum packet sizes of our RDT protocol (header + payload) can be at most 1000 bytes.

– Only one script will run in each host.

V. Experimental Results

We are expected to plot the figure for two experiments by using our own reliable protocols for each experiment provides the relation of packet loss percentage and file transfer time with 95% confidence interval.

The configurations are listed below:

- Configuration 1: Packet Loss is equal to %5 .
- Configuration 2: Packet Loss is equal to %15.
- Configuration 3: Packet Loss is equal to %38.

.

To add emulation delay and packet loss for the experiments to the specified links, we used following code:

```
tc qdisc change dev [INTERFACE] root
netem loss [LOSS] delay [DELAY].
```

- Result for configuration 1 is 0.0499 ± 0.00477:
- Result for configuration 2 is 0.0852425 ± 0.00636:
- Result for configuration 3 is 0.103663 ± 0.00406:

.

Deriving from the graph, we can say that file transfer time increasing while the packet loss percentage is increasing.

VI. Conclusion

With this project, we had a chance to discover how application layer looks like, what is a (partial) network, how UDP configuration can be implement, how end-to-end delay can be measured between nodes and how emulation delay can be added to the links between nodes. Also we experienced that if the the bandwidth increases, the RTT decrease and if emulation delay increases, the end-to-end delay increases.

References

threading - Thread-based parallelism¶. (n.d.). Retrieved from https://docs.python.org/3/library/threading.html.

socket - Low-level networking interface¶. (n.d.). Retrieved from https://docs.python.org/3/library/socket.html.

time - Time access and conversions¶. (n.d.). Retrieved from https://docs.python.org/3/library/time.html#time.time.

Page. (n.d.). Retrieved from https://wiki.python.org/moin/UdpCommunication.

Updated November 02, 2017 / P. J. 24. (n.d.). List of Well-Known TCP Port Numbers. Retrieved from https://www.webopedia.com/quick_ref/portnumbers.asp.

What is Multihoming? - Definition from Techopedia. (n.d.). Retrieved from https://www.techopedia.com/definition/24984/multihoming.