A Project Report

On

Reliable UDP implementation using Selective Repeat Protocol

BY

KESHAV BERIWAL	2017B4A71301H
AMAN BADJATE	2017B3A70559H
PRAKHAR SURYAVANSH	2017B4A71017H
GARVIT SONI	2017B3A70458H
KSHITIJ VERMA	2017B1A71145H

Under the supervision of

DR. DIPANJAN CHAKRABORTY

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS OF

CS F303: COMPUTER NETWORKS

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI (RAJASTHAN)
HYDERABAD CAMPUS

Contents

Introduction

The User Datagram Protocol (UDP) is a communication protocol where data transfer can take place before an initial setup. A few problems like package loss, duplication, etc. arise due to its connection less nature. However, such a connection is generally much faster and less complex than TCP. We aim to design a Reliable layer upon the UDP to tackle these issues without adding much overhead or complexity.

Selective repeat protocol, also called Selective Repeat ARQ (Automatic Repeat request), is a data link layer protocol that uses sliding window method for reliable delivery of data frames. Here, just the incorrect or lost frames are re transmitted, while the good frames are received and buffered. It uses two windows of equal size: a sending window that stores the frames to be sent and a receiving window that stores the frames receive by the receiver [1].

Selective Repeat protocol provides for sending multiple frames depending upon the availability of frames in the sending window, even if it does not receive acknowledgement for any frame in the interim. The maximum number of frames that can be sent depends upon the size of the sending window [2].

The receiver records the sequence number of the earliest incorrect or un-received frame. It then fills the receiving window with the subsequent frames that it has received. It sends the sequence number of the missing frame along with every acknowledgement frame.

The sender continues to send frames that are in its sending window. Once, it has sent all the frames in the window, it re transmits the frame whose sequence number is given by the acknowledgements. It then continues sending the other frames.

Code Walk-through

Client Side:

A client connects to the server using the IP address and port number of the server. To establish a connection, the client first sends a packet that has frametype as FT REQ. As this is the first step, no data is sent to the server at this stage. The DATA section of the packet contains the name of the file by which the file will be saved at the server's end.

Once the client receives the acknowledgement for the first packet, the connection between client and server is established and now client can begin sending packets to the server. To simultaneously send packets and receive acknowledgements, the concept of multi-threading has beenused. One thread calls the send file function for sending the files and another function calls theack recv to receive the acknowledgements.

Sending packets: - The packages to be sent to the server are created in the send_file function. Each packet is initialized with a unique number called the sequence number that varies between 0-7. The window size for the client and server is kept as 4. The DATA section of the packet contains the DATA to be sent and frametype of the packet indicated the type of packet (DATA, FT REQ etc.). As the window size is kept as 4, the maximum number of files the client can

send to the server without receiving acknowledgement is 4. Before sending the packet, the client calls the seqnum ok function to check if the packet that is to be sent is within the specified window size. If not, it waits to receive acknowledgement before sending the packet.

Handling timeout: - Once the packets are sent, the client calls the timeout function using another thread. The timeout function keeps track of the time before re-transmitting the lost/corrupt packages. If the client receives the acknowledgement for the packet it sent, the client stops the re-transmission of the packets by cancelling the thread.

Receiving Acknowledgement: - Another thread calls the ack recv function to receive acknowledgements from the server. Once the client receives the acknowledgement for the packets it sent, the client stops the re-transmission of the packets by cancelling the thread.

Server Side:

The server is first setup to receive connection request from the client. As the server is run on a local host, the IP is kept as 127.0.0.1 and the post can be specified by the user. If a client wishes to establish a connection with the server, it first sends a packet with frametype as FT REQ and DATA containing the name of the file that indicates that client wants to establish connection with the server. Once the server receives the packet with frametype as FT REQ, it sends acknowledge and confirms the connection with the client.

Once the connection is established, the server receives packages with frametype as DATA and makes a note of the sequence number. It then sends an acknowledgement of the frame it received back to the client.

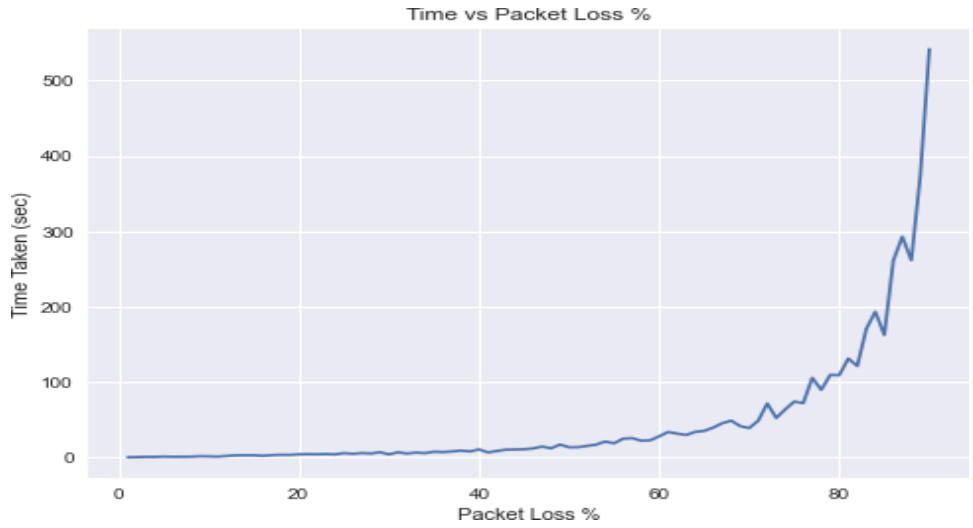
Plots

We have made a script which sends a file from client to server under various network conditions and stores the time taken to transfer the file.

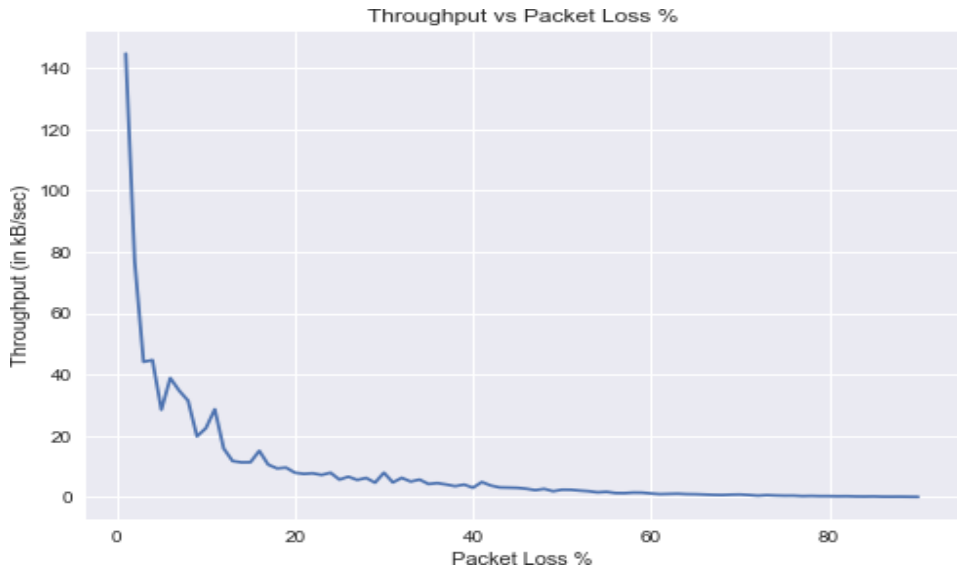
1.1 Throughput vs Packet Loss %

We have noted the time taken to fully transfer file in a condition where packet loss ranges from 0% to 90% keeping all other factors constant. Using this data, we have plotted.

a) Time taken to send file Vs Packet Loss



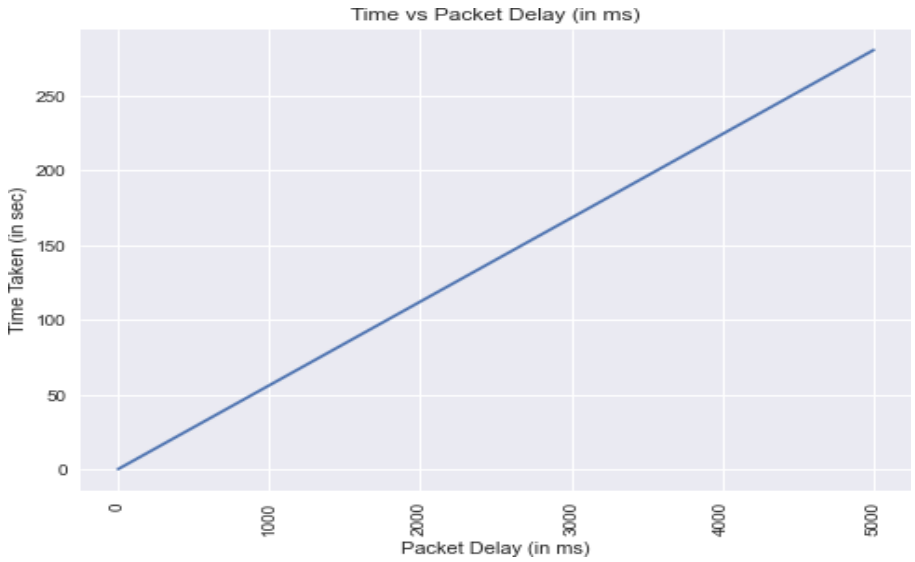
b) Throughput vs Packet Loss



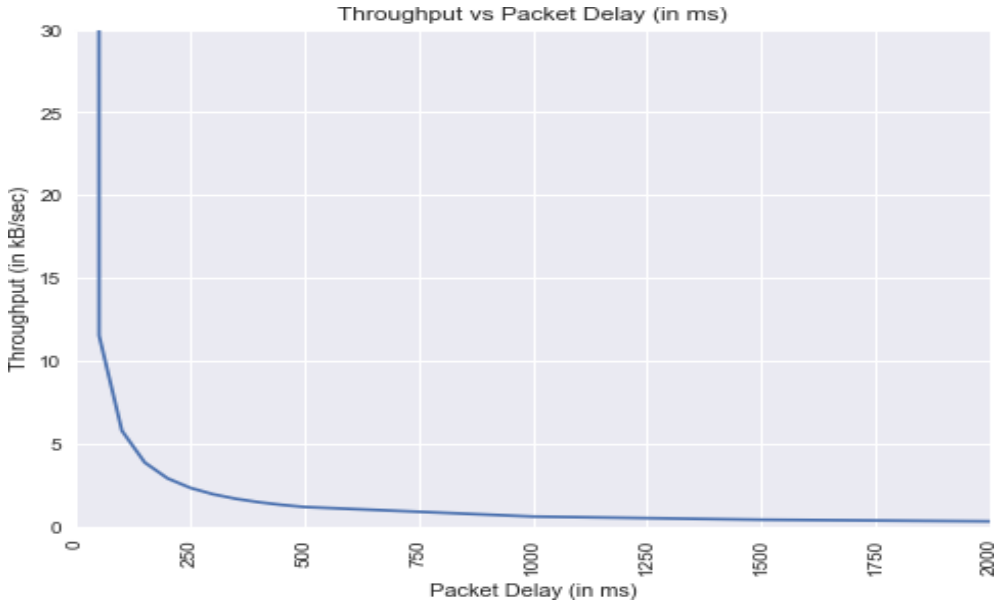
1.2 Throughput vs Packet Delay

We have noted the time taken to fully transfer file in a condition where packet delay ranges from 0ms to 90ms keeping all other factors constant. Using this data, we have plotted

a) Time taken to send file vs Packet Delay



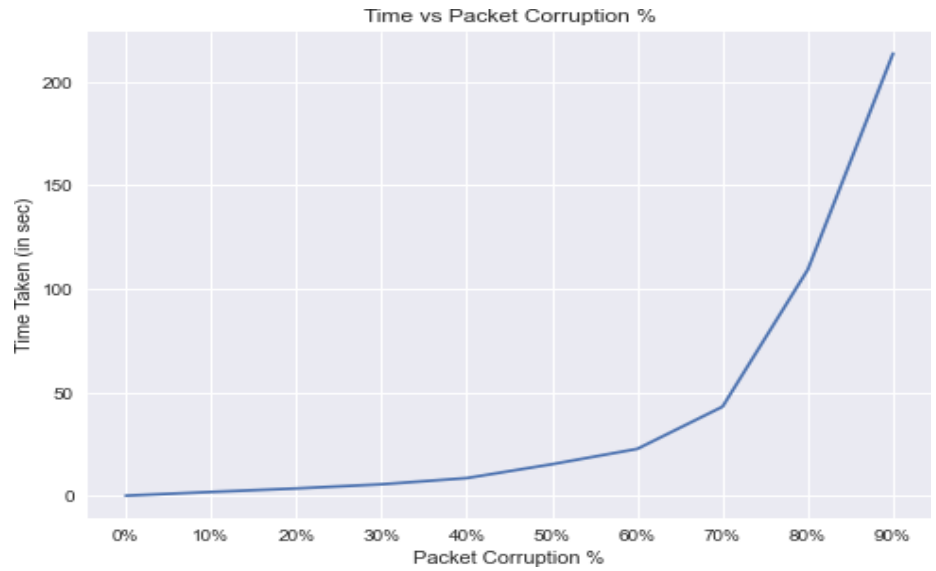
b) Throughput vs Packet Delay



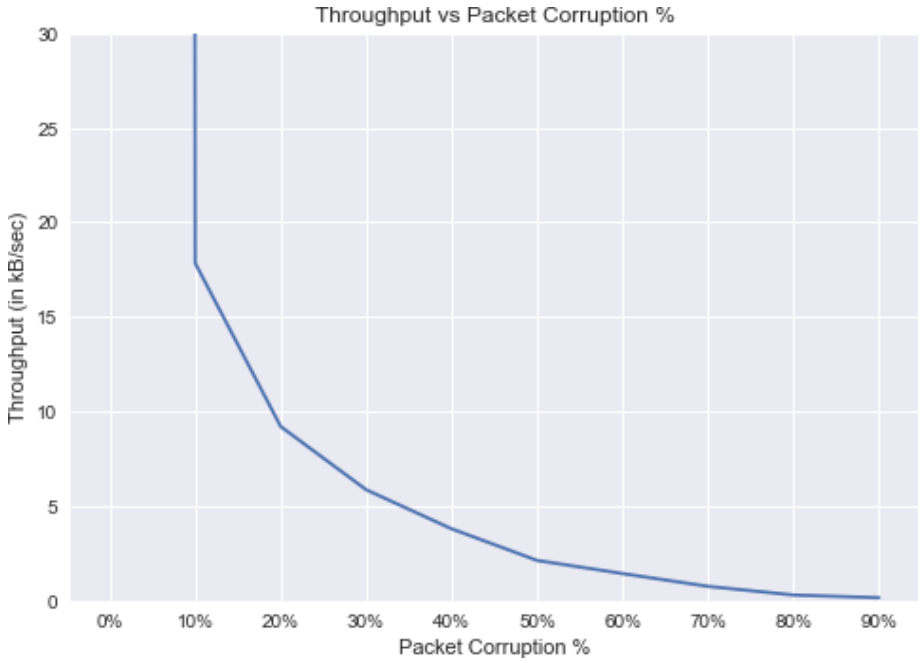
1.3 Throughput vs Packet Corruption %

We have noted the time taken to fully transfer file in a condition where packet corruption ranges from 0 % to 90 % keeping all other factors constant. Using this data, we have plotted

a) Time taken to send file vs Packet Corruption



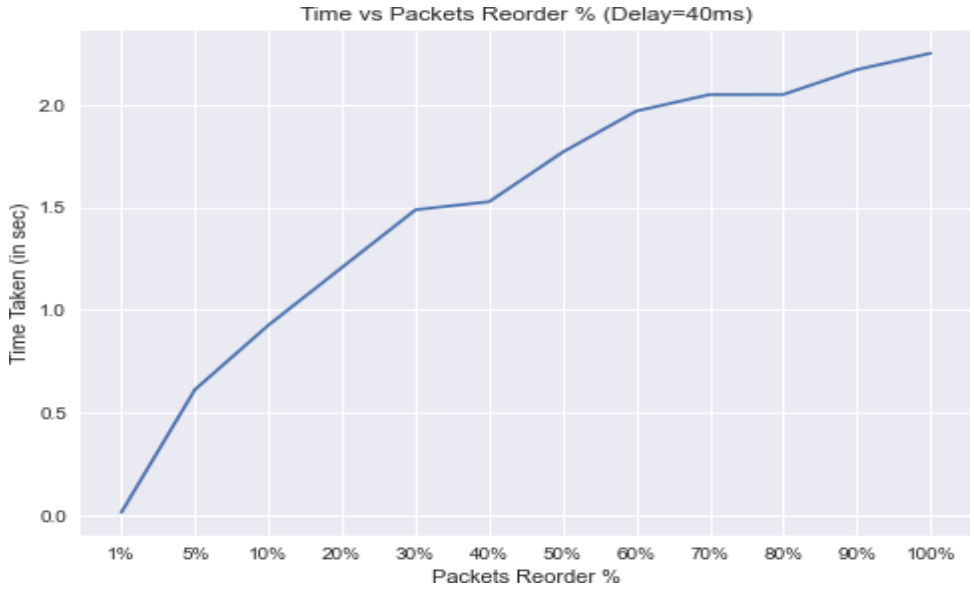
b) Throughput vs Packet Corruption



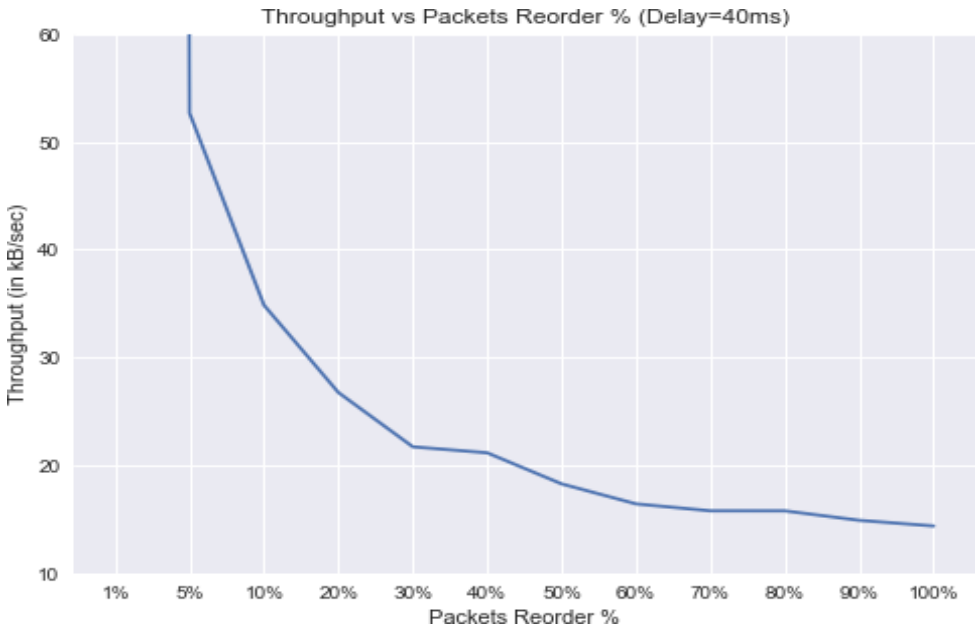
1.4 Throughput vs Packet Reorder %

We have noted the time taken to fully transfer file in a condition where packet reorder ranges from 0 % to 90 % keeping all other factors constant. Using this data, we have plotted-:

a) Time taken to send file vs Packet Reorder



b) Throughput Vs Packet Reorder



References

[1] C Ladas et al. “Class based selective-ARQ scheme for high performance TCP and UDP over wireless links”. In: 4th International Workshop on Mobile and Wireless Communications Network. IEEE. 2002, pp. 311–315.

[2] Abhilash Thammadi. “Reliable user datagram protocol (RUDP).” In: (2011).