CSC/ECE 573 Internet Protocols Project 2 Report:

Team Members:

1. Vivek Nadimpalli (vvnadimp)
2. Sarath Chandra Chaparla (schapar)

Packet Format:

Packet Header Size is fixed to 8 bytes. Packet body follows the header.

Header consists of the following fields:

32 bit sequence number

16 bit checksum

16 bit type field

Implementation:

P2MP FTP Client:

The P2MP FTP client spawns a separate thread for each server to which data is to be sent. The threads are signaled to transmit the packet once it is composed. Each packet of size of MSS is sent to each receiver by their respective threads concurrently. Threads are synchronized before picking up the next segment to transfer. For every file transmission the sequence number is started with zero. This helps the server identify the end of file and beginning of the new file. This is one mechanism to signal the end of the file. Multiple other mechanism can be implemented to signal the end of the file.

The transfer delay is computed by measuring the time taken to send the entire file from the client to all the receivers. This includes time taken to build segments of size MSS, compute checksum, add the header, transmit and get ACKS from all the receivers, and all the retransmission times.

The timeout for receiving ACK is set to 0.05 seconds. This needs to fine-tuned if there are timeouts even with loss probability being zero.

P2MP FTP Server:

The P2MP FTP Server listens for packets on the port 65400. The loss probability is specified as a command line argument when the server program is being run. Once a packet is received, a random number is generated and then checked to see if it is less than loss probability. If so, the packet is dropped, otherwise, the sequence number and checksum are extracted. If the sequence number received equals the expected sequence number, the server then calculates the checksum and if it matches with the received checksum, the data is written to the file and an ACK is sent back to the client.

If a duplicate packet is received for which an ACK was already sent, the packet is discarded and an ACK is resent. If the sequence number received is zero, it is treated as the end of the file and start of the new file transfer.

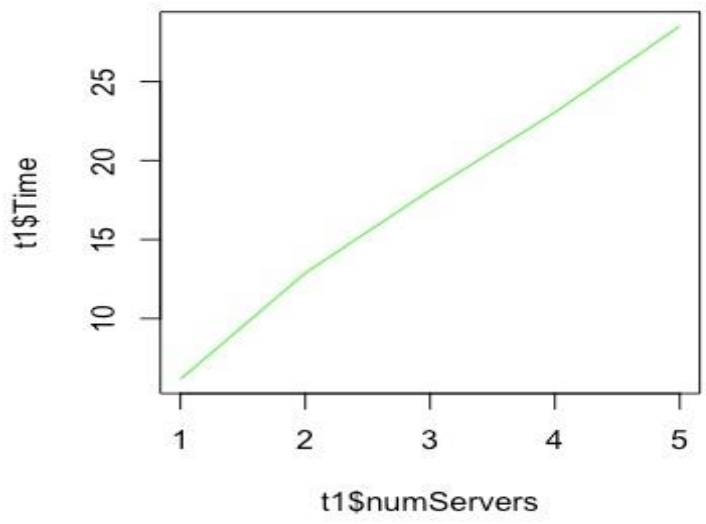
When the client is resending the file five times to compute the average delay, every time the file is being overwritten so that the contents of the file are not accumulated into the file five times.

RTT between Client and Servers:

The average RTT for the client to servers were 0.643, 0.263, 0.764, 0.582, 0.629 ms. Since the RTT values are much smaller, these are dwarfed compared to the retransmission timeouts, packet build and processing time and won't make significant perceptible difference to the total transfer delay.

Test Results:

Task1 (Effect of the Receiver Set Size n):

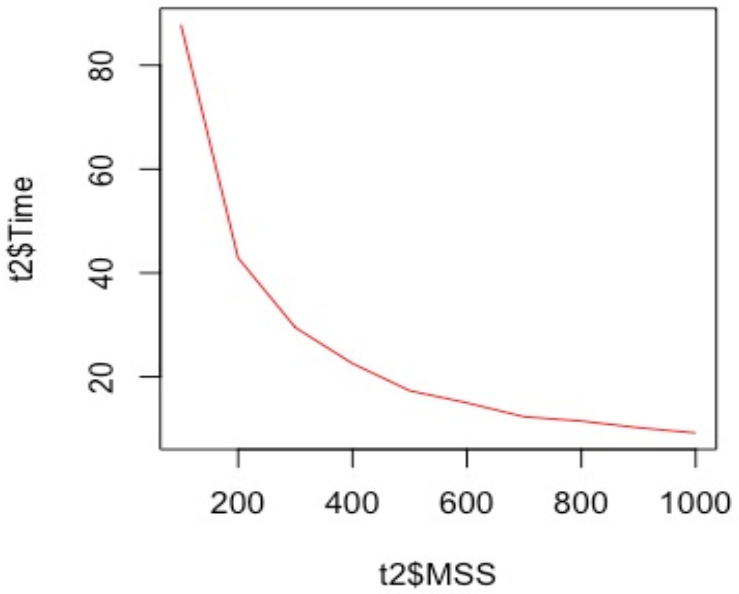


As the number of servers increase, the total transfer delay increases almost linearly. This is due to the fact that all the servers have the same loss probability. The number of retransmissions linearly increase with the number of servers. That being a significant component of the file transfer delay, the file transfer delay thus increases linearly with the number of servers for a given loss probability.

In the real world, not all servers will have the same loss probability, therefore if most of the servers incur no loss of packets, then the file transfer delay would increase only marginally with the number of servers, accounting for the general network latency and the packet segmentation and transmission.

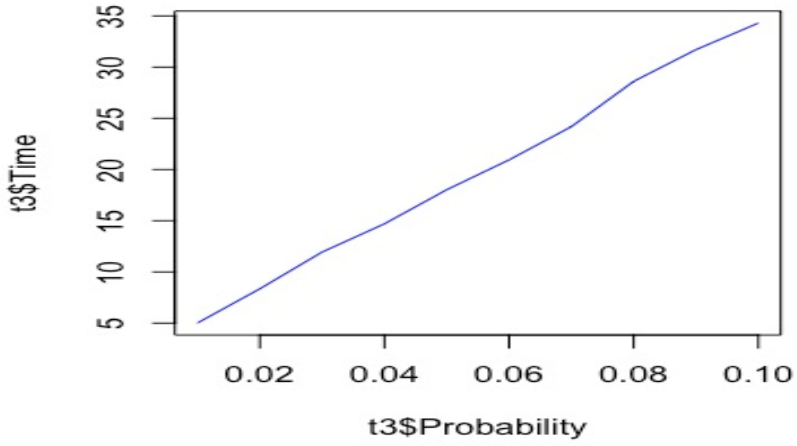
We have tried an experiment where only one server has a loss probability of 0.05 and the other two at 0. The result is that the transfer delay for three servers was around 8 seconds and when run using just one server with loss probability 0.05 was around 6 seconds. This illustrates the point mentioned above.

Task2(Effect of MSS):



As the MSS increases, the total number of packets to be transmitted (file size/MSS) will decrease exponentially. Therefore, the number of retransmission would also decrease accordingly, resulting in an exponentially decreasing file transfer delay with increase in MSS.

Task3(Effect of Loss Probability P):



As the loss probability increases, the number of packet retransmissions also increase linearly. Since the number of servers is fixed to be three, with a linear increase in the number of retransmissions, the file transfer delay also increases linearly.

Conclusion:

This protocol provides a reliable data transfer over UDP albeit not so efficiently. The disadvantage of the Stop and Wait ARQ protocol, where the client waits for ACKs from all the receivers before sending the next segment, results in very poor scalability.

As the number of receiver's increase, and the loss probabilities are non-zero, the data transfer delay would be significantly high making it poorly scalable.

Transferring a file independently to each receiver without synchronizing the transmission of each segment across all receivers, would result in a significant improvement in the transfer delay and scales out better.