Point to Multipoint File Transfer Protocol (P2MP-FTP)

-Ragavi Swarnalatha Raman – 200203611

Sampreetha Nattamai Sureshbabu - 200205625

Introduction:

A File Transfer Protocol is generally used to transfer files between a client and a server over a network. It uses TCP to provide reliable transfer of files from one client to one server. We implement a variation of it: transferring a file from one host to multiple receivers using UDP. To add reliable communication, we use a stop and wait protocol (ARQ). The client-server architecture is as follows:

The P2MP-Client is the sender that connects to multiple P2MP-Servers that acts as receivers in the reliable data transfer. Data transfer is done from client to servers and the servers return an ACK for every data segment received.

Implementation:

P2MP-Client:

The client uses a function called rdt_send() that reads the file to be sent on a byte basis. A local buffer accumulates the data read until it reaches the size of MSS. This data is added along with the following header data:

i) Sequence number of size 32 bits
ii) Checksum of the data part of size 16 bits
iii) A 16-bit data sequence that indicates that it is a data packet

Once the segment is ready to be sent, it is sent to all the receivers simultaneously, starts a timer and waits for all the receivers to send a ACK back before proceeding to the next segment. If the timer goes out before receiving all the acknowledgements, a retransmission of the segment happens only to the receivers from which it has not received an ACK. The stop and wait protocol uses a window size of 1.

P2MP-Server:

- The server implements the receiver side of the ARQ protocol.
- As soon as it receives segment from the sender it checks for the correct sequence number and discards if it does not match.
- If the sequence number matches with the expected one, then the data is extracted from the segment.

- Since this experiment is carried out in local network, errors are induced by generating a random number in the range (0,1). If the random number generated is greater than the probability value, then the following steps are carried out. If it is less, then the server does not send an ACK.
- Checksum is calculated for the data extracted and discarded if the checksum is wrong and waits for retransmission of data from the client, else appends the data to the file.
- Once the data received is appended to the file, an ACK is sent back to the client. The ACK consists of the following fields:
  i) The sequence number of the data that is being acknowledged of size 32bit
  ii) A 16-bit field of all zeros
  iii) A 16-bit value indicating that it is a ACK packet

Retransmission:

Retransmission of data occurs in the following cases:

i) When a server does not acknowledge a data segment (due to error in receiving data/wrong checksum) and timeout occurs at the client side
ii) When the server sends an acknowledgement, but it gets lost somewhere along the transit and timeout occurs at the client side

Offline Experiments:

File Size: 1Mb

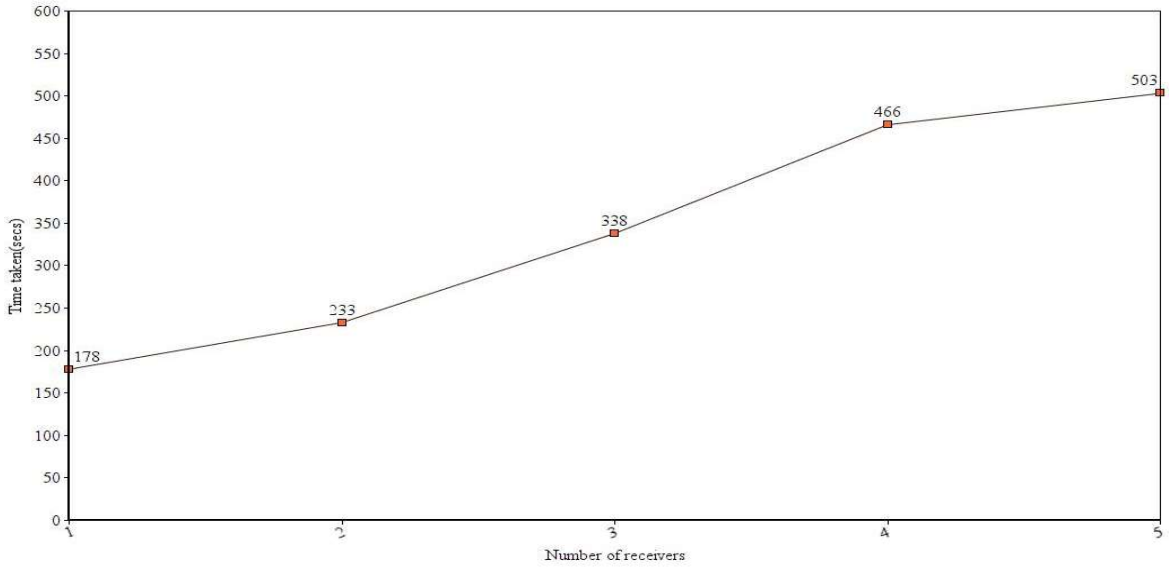
Timeout value:

- The client was connected to a wired network and all the servers were connected to a wireless network. This ensures that there are a few hops in between.
- The RTT value was determined by performing traceroute to one of the servers and obtained an average value of 10ms.
- Hence, for all the following offline experiments, the timeout value was set to be 5*RTT = 0.05 seconds.

Task1 - Effect of the Receiver Set Size n:

MSS = 500 bytes, loss probability = 0.05, no of receivers = 1 to 5, Timeout = 0.05 s

Average Delay Graph:



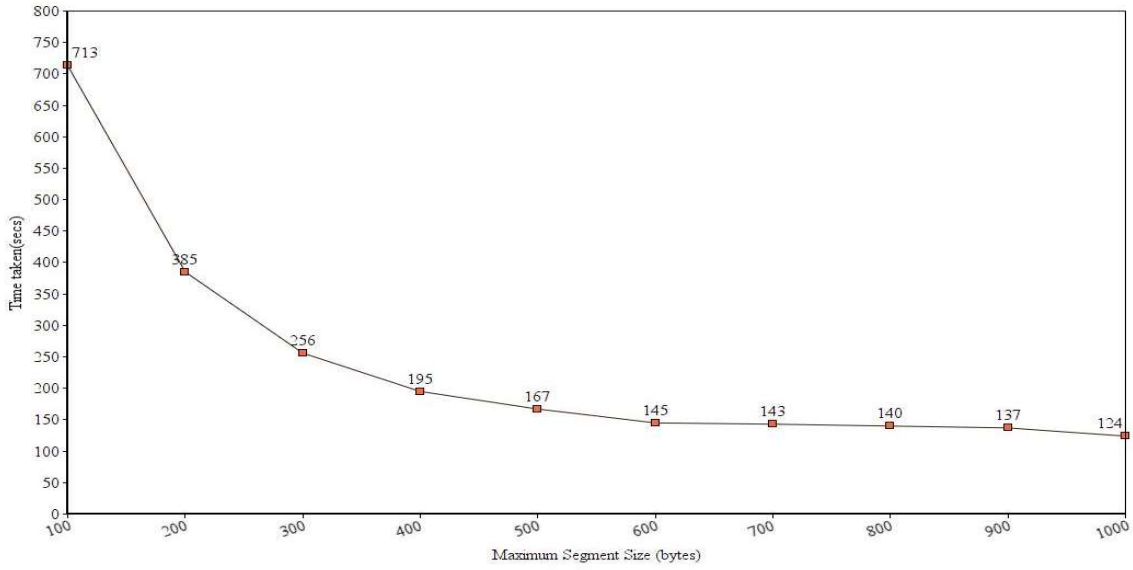
Inference:

For a constant value of the MSS and the loss probability, an increase in the number of receivers results in an increase in the average delay taken to transfer the file. This is in line with the expectation as the client has to wait for ACKs from all the servers in order to send the next segment. Thus the shape of the graph is almost linear.

Task2 – Effect of MSS:

No of receivers = 3, loss probability = 0.05, MSS = 100 to 1000 bytes, Timeout = 0.05 s

Average Delay Graph:



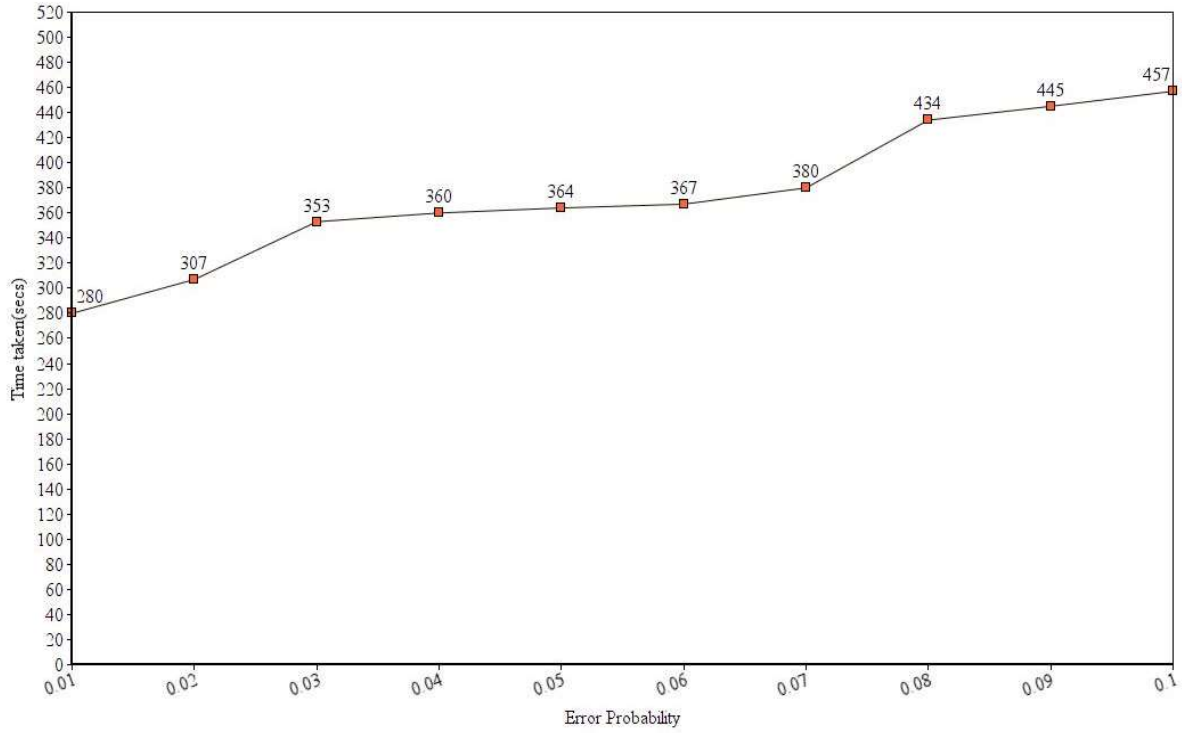
Inference:

For a constant value of 0.05 loss probability and 3 receivers, the average delay seems to decrease as the size of the segment increases. But after a certain MSS the delay is observed to stay almost constant irrespective of the increase in the segment size.

Task3 – Effect of Loss Probability:

MSS = 500 bytes, no of receivers = 3, loss probability = 0.01 to 0.10, Timeout = 0.05 s

Average Delay Graph:



Inference:

For a constant segment size of 500 bytes and 3 receivers, the average delay increases as the loss probability is increased. This is a totally expected behavior as a greater loss probability infers greater number of segment loss and increased retransmissions. Thus, the shape of the graph is almost linear.

Conclusion:

Based on the offline experiments conducted, the following conclusions can be made:

i) Since a stop and wait protocol is used here, the link/channel is underutilized as only one segment is sent to the receiver at a time. A Go Back N or Selective Repeat protocol would be more efficient and be of use in practical real time scenarios as multiple segments can be sent at a time but still there is a question of scalability.

ii) We also observe that the protocol does not support scalability. From experiment1 we observe that delay increases as the number of receivers increase indicating that the protocol experiences huge inefficiency when larger number of receivers are introduced.