**Point to Multipoint File Transfer Protocol [P2MP-FTP]**

**Introduction:**

In this project we have implemented a point-to-multipoint reliable data transfer protocol. The main features are encapsulation of data into transport layer headers, buffering and managing data receivers. UDP has been used as the transport protocol and the Automatic repeat request scheme using Stop-and-Wait protocol has been implemented over it. Thus a reliable data transfer protocol is made to run over an unreliable data transfer protocol [udp].

**Implementation Details:-**

The Protocol has a single client which would be simultaneously sending files to many servers.

The functions of the client include:-

--Send data to the servers according to the specified MSS.

--Along with the data there should be a header attached, the header will consist of the checksum, sequence no and a field indicating that it is a data packet.

--The client would simultaneously send data to all the servers and when it receives a ACK from all the servers, it would increment the sequence no and send the next data packet.

--Since, stop-and-wait protocol has been employed, hence the client would wait until it receives ACK from all the receivers, in case an ACK is not received and the timer times out then it would resend the data to that server.

The functions of the server include:

--Receiving data from the client and first verifying the sequence no and the checksum.

--It would send an ACK for the data packet unless the checksum computed is incorrect or there is a sequence no mismatch, in which case it would not send anything.

For experimental purpose, there is a mechanism to introduce packet loss inorder to check the efficiency of the protocol.

**Messages for invoking the server and client**

**Command Line Arguments**

Both the server and client are invoked through the command line arguments

The P2MP-FTP server is invoked as follows:

p2mpserver port# file-name p

where port# is the port number to which the server is listening (for this project, this port number is always 7735), file-name is the name of the file where the data will be written, and p is the packet loss probability.

The P2MP-FTP client must be invoked as follows:

p2mpclient server-1 server-2 server-3 server-port# file-name MSS

where server-i is the host name where the i-th server (receiver) runs, i = 1, 2, 3, server-port# is the port number of the server (i.e., 7735), file-name is the name of the file to be transferred, and MSS is the maximum segment size.

**Tasks**

For the following tasks we used vcl machines as servers and the client was a PC located at home. A file of size 5Mb was used for transfer, for the initial check we sent a ping of 1000 bytes from the client to the server and the round trip time was recorded as 30 milliseconds, thus the timer value was set to about 150 milliseconds which is about 5 times the RTT value.

**Task 1:**

As can be seen from the curves, as the no of servers increases the total delay in sending the file also increases. This is because of the underlying protocol to send the file which is stop-and-wait, since all the sender has to wait till it receives ack’s from all the servers to send the next data segment hence the delay increases as n increases, the shape of curve is more or less linear indicating that there is a direct relation between the no of servers and the delay.

**Task 2:**

The above graph is a plot of the MSS value against the average delay for a probability of 0.05. From the graph, we observe that as the MSS size increases the average delay in transferring the file decreases. This is expected since as MSS increases more no of bits would be sent simultaneously hence requiring fewer RTTs to send the complete file across. However, there also has to be an upper limit on the MSS, if the MSS is increased beyond a particular value it would mean that the processing time would increase and also in case of retransmission more no of bytes would have to be sent increasing the overall delay. However, we do not notice this behavior in the graphs indicating that it has not yet reached its threshold value.

**Task 3:**

The graph shown above is a plot of the loss probability against time taken for n=3 servers and MSS=500. From the delay values we notice that as the loss probability increases the time taken to transfer the file also increases which is expected, since increase in loss probability would mean more no of packets are dropped hence there would more retransmissions and overall delay would increase. As evident by the shape of the curve there is a direct relation between the loss probability and the average delay, which indicates that the shape of the curve is more or less linear.

**Conclusion of the protocol for scalability**

From the above experiments we conclude that, though there is reliability in the point to multipoint protocol, however the total delay as the no of servers increase is beyond acceptable limits. Thus, it does not scale efficiently, instead of stop-and wait even if we were using go-back n with a slightly larger window size then it would still hamper the performance with addition of servers. This isn’t a viable protocol when using over internet and practically can only be used under extenuating circumstances. The stop and wait protocol is not an efficient protocol for scalability of receivers. If the number of receivers are increased as seen in graph 1, the protocol’s performance decreases and the delay required to transfer a complete file would increase to unacceptable values. Thus, this is not an efficient protocol for scalability.