IP Project 2 Report

Ashlesha Atrey (200203500)

Sai Kiran (200202554)

Introduction:

We implemented a Point-to-multipoint File Transfer Protocol (P2MP-FTP) using the Stop and Wait ARQ scheme. The P2MP-FTP will use UDP to send packets from the sending host to the destination. But we know that UDP is one protocol which is unreliable for data transfer. Hence, an ARQ scheme (Stop and Wait for this project) is used to implement the reliable data transfer. The following explanations of the blocks implemented in this project, explain in detail the protocol and the methodologies applied to implement the protocol.

P2MP Sender(p2mpcient):

The sender obtains the data to be sent to the server from a file that is stored at the sender side, reads it, and then sends it to the receiving host. The name of the file to be read is given in the command line. The sender has the roles such as implementing time-out functions and processing acknowledgments from the server. On detecting correct or incorrect transmission of packets (based on ACK received from server), it decides what further actions to take. These actions are explained in the later part of this report. The sender side has a parameter called the Maximum Segment Size, which indicates the maximum size of the segment that can be sent by the sender.

Sequence numbers:

These are used to detect packet losses and the order in which the segments arrive. Each packet header consists of a 32-bit value which is used as a parameter that denotes the identity of a packet to distinguish it from another. So, the packets will be sent in an ascending order of sequence numbers, which is used to detect improper packet delivery if the packets don't arrive in the expected order.

In the program, the sender side initially stores a variable used to address a sequence number. On receiving the acknowledgement for a packet it sends, it will update this variable such that the next packet that is sent from the sender side, has a sequence number which is greater than the previous sequence number. This sequence number assigned to the new packet is the updated value of the variable used to store the sequence number. By this way we can also keep trace of the order in which the data packets are arriving. This is one way by which the data transfer is made reliable in the implemented protocol.

Checksum

The checksum is obtained by the binary addition of the header data after breaking them into chunks of16 bits. As the data is received, it is parsed in the sender side so as to include only 16 bits of the data for calculation. An iteration variable is used to parse through the next 16 bits. The iteration continues and the variables are added after which the 1's compliment is taken. This computed value before calculating the 1's compliment as well as the 1's compliment value is sent to the receiver. The receiver verifies that the data received is exactly as that of what was sent by the sender. If the sum obtained is "1111111111111111", then it means that data was not modified.

Time-out Mechanism

The time-out mechanism is used in the protocol to ensure that if a ACK is not received within a period (1 round trip time), then that particular segment is re-transmitted. The protocol ensures that the client does not send any new data until it received acknowledgements from all the receiving hosts. In case that any of the receiving hosts don't send the acknowledgement within a roundtrip time, then that segment is re-transmitted to that particular host. Untill that, no new segment is re-transmitted. Time out counter is set to 0.1 sec.

The segment contains five fields in the following order:

- A 32-bit sequence number
- A 16-bit field that has the value 0101010101010101, indicating that this is a data packet.
- A 16-bit field which store the length of MSS
- Actual payload to be sent (along with padding if needed)
- A 16-bit checksum of the data part, computed in the same way as the UDP checksum

Command Line argument for Client:

The P2MP-FTP client must be invoked as follows,

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

p2mpclient.py 192.168.0.4 192.168.0.17 7735 rfc3435.txt 500

Timeout Received, Sequence Number: 1

Timeout Received, Sequence Number: 16

here,

There are two servers with IP 192.168.0.4 and 192.168.0.17 binded to port 7735.

We are transferring file rfc3435.txt with MSS=500

P2MP Receiver(p2mpserver):

The servers listen to the client on a well-known port number 7735. On receiving the data from the sender, the receiver computes the checksum of the received data and sends an acknowledgement for the segment which it received.

Probability function to check reliability of UDP:

Despite the fact that UDP is unreliable, the Internet does not in general lose packets. Therefore, we need a systematic way of generating lost packets so as to test that the Stop-and-Wait protocol works correctly (and to obtain performance measurements, as will be explained shortly).

We implemented a probabilistic loss service at the server (receiver). Specifically, the server read the probability value p,$0 < p < 1$ from the command line, representing the probability that a packet is lost. Upon receiving a data packet, and before executing the Stop-and-Wait protocol, the server will generate a random number r in (0,1). If $r \leq p$, then this received packet is discarded and no other action is taken; otherwise, the packet is accepted and processed according to the Stop-and-Wait rules.

The action taken for reception of correct segment and error detection (if error present) are as follows:

Actions taken for correct reception of the segment

Now, when a packet is received, an ACK has to be sent to the sender. Now, this ACK packet that has to be sent consists of a 32 bit sequence number, a 16 bit field with all zeroes and a 16-bit stream of given data in the binary format. The sequence number to be allocated is obtained from the program in the form of an integer value which is converted into binary form in order to proceed with the process of filling the header field and thereby filling the packet which is used to send the acknowledgment.

Actions taken on error

If a packet at the receiver is received such that the sequence number is not as expected (out of order packet), then the acknowledgement for the previous packet is sent. For example, if packets of the order 1,2,3,4,5 are expected, but the transmission of packets is proper till the 3rd packet and then the 5 th packet is received before the 4th, then the acknowledgement for the previous packet (3rd) is sent. So, on receiving the duplicate acknowledgement, the sender knows that the packet after the 3rd one has not yet been received properly. Therefore, it re-transmits the 4th packet. However, in the case if checksum criterion described previously is not satisfied, then no action is taken. However, this scenario which is explained is used to indicate the action taken in the case of a checksum error in this protocol. It is not for any specific test case.

Command line arguments for Server:

p2mpserver port# file-name p

for example:

python p2mpserver.py 7735 2 0.1

Packet loss, sequence number =1

Packet loss, sequence number =16

Packet loss, sequence number =16

here,

port# is 7735, filename is test.txt where the received file will be saved. P=0.01 which indicated the probability value.

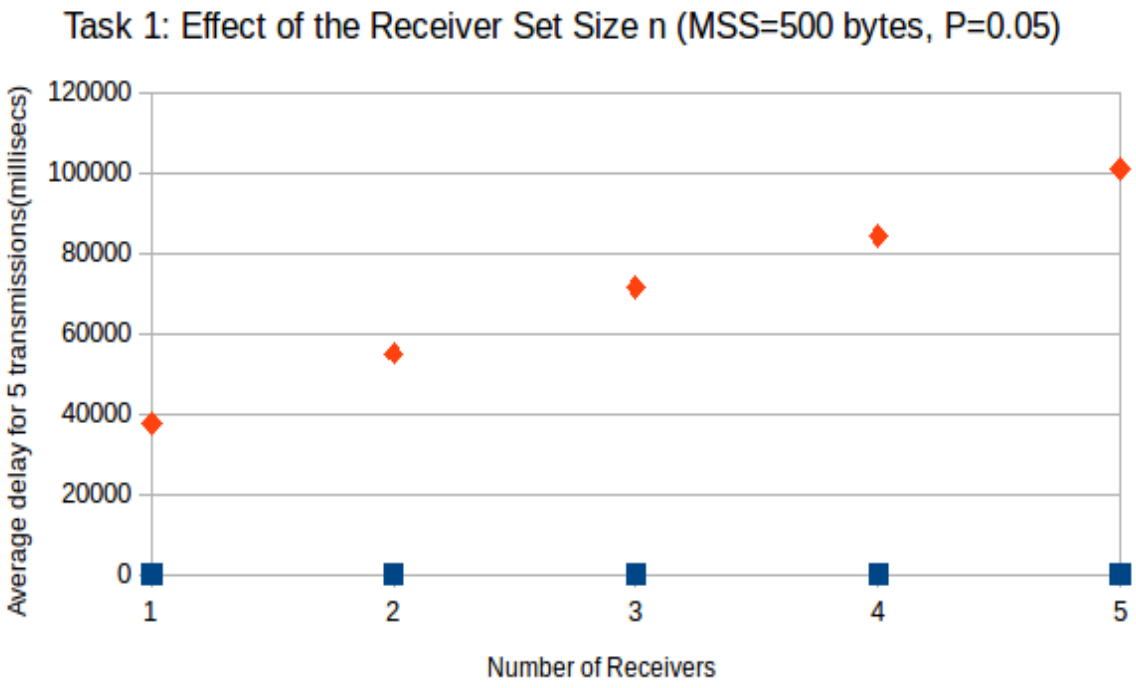
Testing Scenarios:

We had 5 servers running in different virtual machines (Ubuntu 16) binded to localhost and running on port 7735.

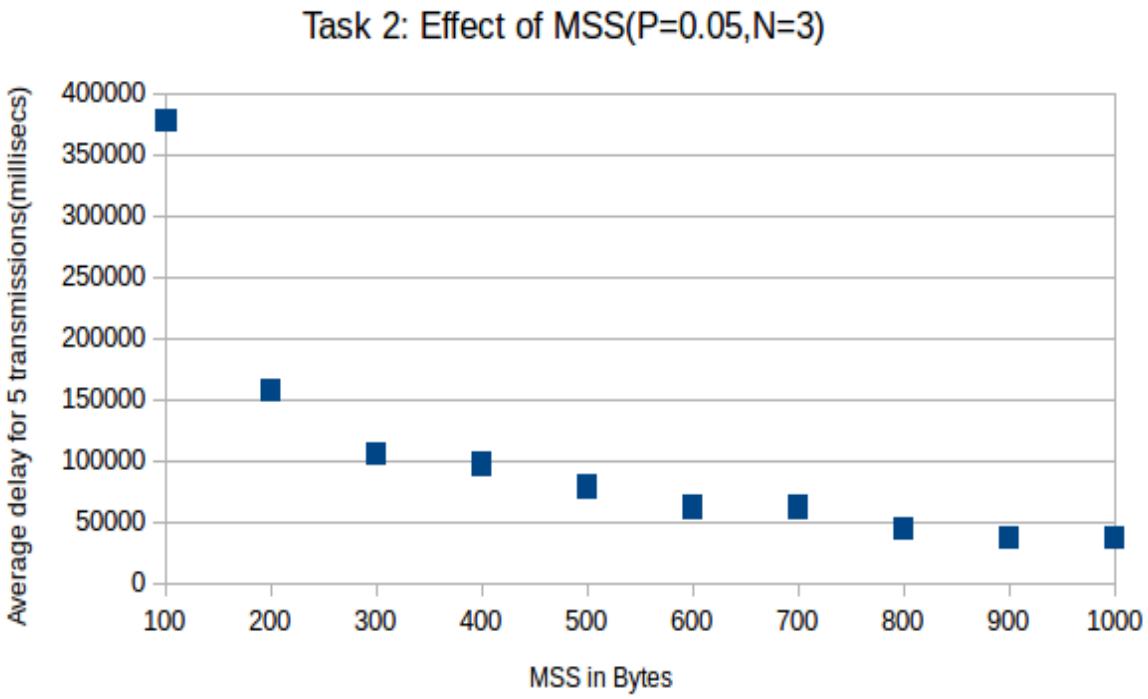
Client was running in one the above virtual machines.

Our test file which was sent to all the servers was 1.05 MB.

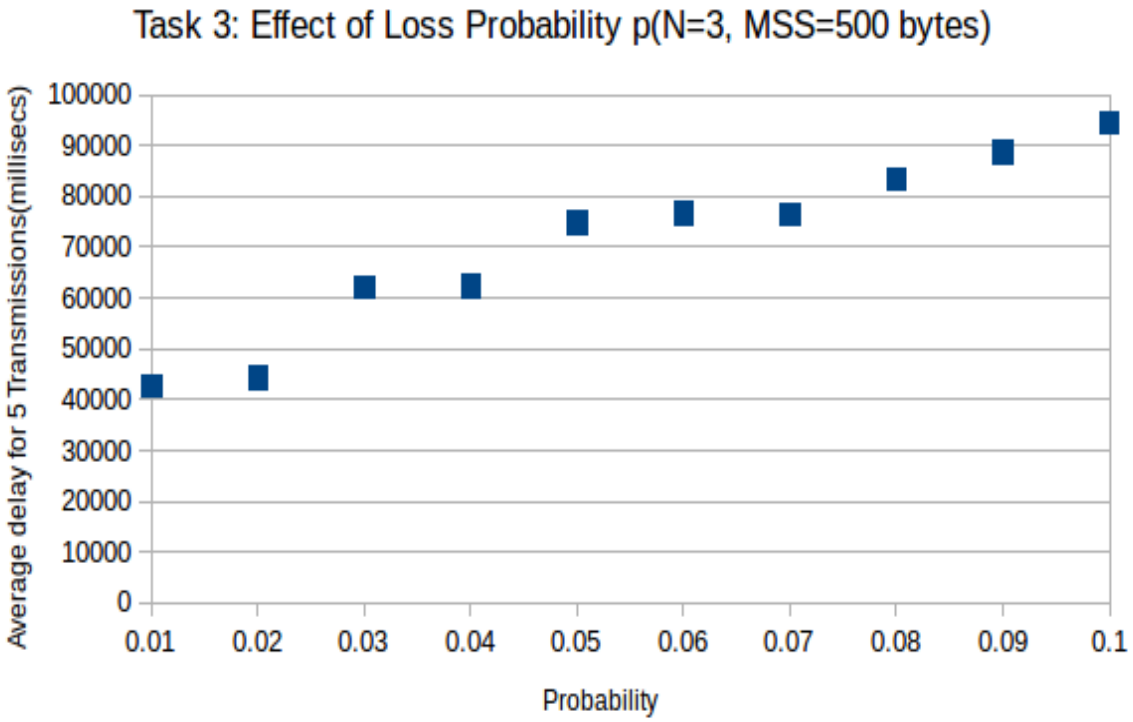
The offline tasks were performed and the results for them are as follows:



The above graph shows the plot of number of receivers (n) vs average delay (d). We see that as the number of receivers increase, the delay for transmission increases. The increase in delay is linear. So, if the number of receivers increase by 10 times, the delay goes up by approximately 10 times as well. This is evident from the fact that, as there will be more number of hosts that want to receive data, the sender has to send the data to more number of receivers. So, the delay keeps increasing as the more number of receivers are being added.



From the above graph, we can infer that, as the size of the MSS increases, the delay decreases exponentially. When the maximum segment size (MSS) increases, the amount of data that can be received in a segment will be more. This means that the number of packets required to transmit the same amount of data is lower. So, per-packet processing time is less. However, since packets are now bigger, per flow delays between packets are now also higher, so the minimum delay increases. So, as we are required to calculate the per-process delay in our case, the exponential decrease is as expected.



From the graph, we can observe an increase (excluding slight variations) in the average delay as probability of loss increases. Our random value is calculated between 0 to 1. This random value is compared with the 'p' which we are varying between 0.01 to 0.1. Our condition is when the random value is less than the probability value indicated by the server, the packet is considered to be lost and should be transmitted again by the client.

Hence when p=0.01, the probability of success is more as there is more chance of r being greater than 0.01. As p is kept increasing till 0.1 the probability of success decreases but not linearly as the random variable is Pseudo random. So, there has to be increased amount of re-transmissions. This leads to more consumption of time, thereby increasing the delay as the loss probability is varied in increments of 0.01.