CSC/ECE 573 – Internet Protocols

Project #2
Due Date: November 27, 2017

Project Objectives

In this project, you will implement point-to-multipoint reliable data transfer protocol using the Stop-and-Wait automatic repeat request (ARQ) scheme, and you will carry out a number of experiments to evaluate its performance. In the process I expect that you will develop a good understanding of ARQ schemes and reliable data transfer protocols and build a number of fundamental skills related to writing transport layer services, including:

- encapsulating application data into transport layer segments by including transport headers,

- buffering and managing data received from, or to be delivered to, multiple destinations,

- managing the window size at the sender,

- computing checksums, and

- using the UDP socket interface.

Point-to-Multipoint File Transfer Protocol (P2MP-FTP)

The FTP protocol provides a sophisticated file transfer service, but since it uses TCP to ensure reliable data transmission it only supports the transfer of files from one sender to one receiver. In many applications (e.g., software updates, stock quote updates, document sharing, etc) it is important to transfer data reliably from one sender to multiple receivers. You will implement P2MP-FTP, a protocol that provides a simple service: transferring a file from one host to multiple destinations. P2MP-FTP will use UDP to send packets from the sending host to each of the destinations, hence it has to implement a reliable data transfer service using some ARQ scheme; for this project, you will implement the Stop-and-Wait ARQ. Using the unreliable UDP protocol allows us to implement a "transport layer" service such as reliable data transfer in user space.

Client-Server Architecture of P2MP-FTP

To keep things simple, you will implement P2MP-FTP in a client-server architecture and omit the steps of opening up and terminating a connection. The P2MP-FTP client will play the role of the *sender* that connects to a set of of P2MP-FTP servers that play the role of the *receivers* in the reliable data transfer. All data transfer is from sender (client) to receivers (servers) only; only ACK packets travel from receivers to sender.

The P2MP-FTP Client (Sender)

The P2MP-FTP client implements the sender in the reliable data transfer. When the client starts, it reads data from a file specified in the command line, and calls *rdt_send()* to transfer the data to the P2MP-FTP servers. For this project, we will assume that *rdt_send()* provides data from the file on a byte basis. The client also implements the sending side of the reliable Stop-and-Wait protocol, receiving data from *rdt_send()*, buffering the data locally, and ensuring that the data is received correctly at the server.

The client also reads the value of the maximum segment size (MSS) from the command line. The Stop-and-Wait protocol buffers the data it receives from *rdt_send()* until it has at least one MSS worth of bytes. At that time it forms a segment that includes a header and MSS bytes of data; as a result, all segments sent, except possibly for the very last one, will have exactly MSS bytes of data.

The client transmits each segment separately to each of the receivers, and waits until it has received ACKs from every receiver before it can transmit the next segment. Every time a segment is transmitted, the sender sets a timeout counter. If the counter expires before ACKs from all receivers have been received, then the sender re-transmits the segment, *but only to those receivers from which it has not received an ACK yet.* This process repeats until all ACKs have been received (i.e., if there are n receivers, n ACKS, one from each receiver have arrived at the sender), at which time the sender proceeds to transmit the next segment.

The header of the segment contains three fields:

- a 32-bit sequence number,

- a 16-bit checksum of the data part, computed in the same way as the UDP checksum, and

- a 16-bit field that has the value `0101010101010101`, indicating that this is a data packet.

For this project, you may have the sequence numbers start at 0.

The client implements the sending side of the Stop-and-Wait protocol as described in the book, including setting the timeout counter, processing ACK packets (discussed shortly), and retransmitting packets as necessary

The P2MP-FTP Server (Receiver)

The server listens on the well-known port `7735`. It implements the receive side of the Stop-and-Wait protocol, as described in the book. Specifically, when it receives a data packet, it computes the checksum and checks whether it is in-sequence, and if so, it sends an ACK segment (using UDP) to the client; it then writes the received data into a file whose name is provided in the command line. If the packet received is out-of-sequence, an ACK for the last received in-sequence packet is sent;, if the checksum is incorrect, the receiver does nothing.

The ACK segment consists of three fields and no data:

- the 32-bit sequence number that is being ACKed,

- a 16-bit field that is all zeroes, and

- a 16-bit field that has the value `1010101010101010`, indicating that this is an ACK packet.

Generating Errors

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).

To this end, you will implement a *probabilistic loss service* at the server (receiver). Specifically, the server will 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.

Offline Experiments

You will carry out a number of experiments to evaluate the effect of the number n of receivers, MSS, and packet loss probability p on the total delay for transferring a file using the P2MP-FTP. To this end, select a file that is approximately 10MB in size, and run the client and n servers on different hosts, such that the client is separated from the servers by several router hops. For instance, run the client on your laptop/desktop connected at home and the servers on EOS machines on campus; or the client machine (e.g., your laptop)

could be located in the same room as the servers, but have the client connected to a wireless LAN while the servers are connected to the wired network. Record the size of the file transferred and the round-trip time (RTT) between client and servers (e.g., as reported by `traceroute`), and include these in your report.

Task 1: Effect of the Receiver Set Size n

For this first task, set the MSS to 500 bytes and the loss probability $p = 0.05$. Run the P2MP-FTP protocol to transfer the file you selected, and vary the number of receivers $n = 1, 2, 3, 4, 5$. For each value of n, transmit the file 5 times, time the data transfer (i.e., delay), and compute the average delay over the five transmissions. Plot the average delay against n and submit the plot with your report. Explain how the value of n affects the delay and the shape of the curve.

Task 2: Effect of MSS

In this experiment, let the number of receivers $n = 3$ and the loss probability $p = 0.05$. Run the P2MP-FTP protocol to transfer the same file, and vary the MSS from 100 bytes to 1000 bytes in increments of 100 bytes. For each value of MSS, transmit the file 5 times, and compute the average delay over the five transmissions. Plot the average delay against the MSS value, and submit the plot with your report. Discuss the shape of the curve; are the results expected?

Task 3: Effect of Loss Probability p

For this task, set the MSS to 500 bytes and the number of receivers $n = 3$. Run the P2MP-FTP protocol to transfer the same file, and vary the loss probability from $p = 0.01$ to $p = 0.10$ in increments of 0.01. For each value of p transmit the file 5 times, and compute the average delay over the five transfers. Plot the average delay against p, and submit the plot with your report. Discuss and explain the results and shape of the curve.

Submission and Deliverables

You must submit your report and source code, as explained below, using the submit facility by 11:59pm on the day due. There are several weeks until the due date for you to work on this project, therefore no late submissions will be accepted.

You will carry out Tasks 1-3 offline, and submit a report (PDF or Word file) with the results and your explanation/discussion of the findings. In particular, your report should include the file transfer delay curves for Tasks 1-3, your explanation of the behavior of the curves, and conclusions you may draw regarding the scalability of the P2MP-FTP protocol.

In order for us to test your program, you will also submit the source code (*no object files!*) separately. Name the file containing the source code `proj1`, with the appropriate extension (e.g., `proj1.c` if you code in C). We would like to ensure that the TA does not spend an inordinate amount of time compiling and running your programs. Therefore, make sure to include a `makefile` with your submission, or a file with instructions on how to compile and run your code. Therefore, if you fail to include such a file, we will **subtract 5 points** from your project grade.

The code you submit must follow these specifications.

Command Line Arguments

The P2MP-FTP server must be 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 discussed above.

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.

Output
The code you submit must print the following to the standard output:

- P2MP-FTP server: whenever a packet with sequence number X is discarded by the probabilistic loss service, the server should print the following line:

  `Packet loss, sequence number = `X

- P2MP-FTP client: whenever a timeout occurs for a packet with sequence number Y, the client should print the following line:

  `Timeout, sequence number = `Y

Grading

P2MP-FTP server and client code (compiles and runs correctly)	70 Points
Task 1	10 Points
Task 2	10 Points
Task 3	10 Points
	—————
	100 Points