CSE156/L Programming Lab 3: Simple Reliable File Transfer

Serjo Barron
sepbarro
Winter 2024

Program Design (Server):

1. Parse arguments: port number and droppc (drop percentage)
2. Check if port is valid
3. Create a server socket and bind it to the port number
4. Run an infinite loop that calls “process_packet()” which does the following:
    a. Receive bytes from client (when they are sent)
    b. Store received bytes in a buffer (up to 4096)
    c. Log received DATA packet
    d. Case 1: first packet received (contains outfile path)
        i. No droppc is applied
        ii. Send ACK to client to confirm packet was received
        iii. The buffered data (which contains outfile path at this point) is not written to the outfile path
        iv. Log ACK packet
        v. Increment packet sequence number
    e. Case 2: all subsequent packets received
        i. Use srand(time(NULL)) and rand() to generate a random number of range 0-99
        ii. If the random number < droppc:
            1. Log dropped packet
            2. Exit process_packet()
        iii. If packet is not dropped (random number >= droppc):
            1. Open outfile
            2. Write buffered data to outfile path in append bytes mode
            3. Close outfile
            4. Send ACK to client to confirm packet was received
            5. Log ACK packet
            6. Increment sequence packet number
5. The program ends when terminated by the user

Program Design (Client):

1. Parse arguments: server IP, server port number, MTU, winsz, infile path, and outfile path
2. Check if server port is valid
3. Check if MTU is at least 1
4. Check if winsz is at least 1
5. Call a function “sendfile()” which does the following:
    a. Open infile path in reading bytes mode
    b. Truncate the infile to empty it before the server appends data to it
    c. Create a socket, and initialize it according to the server IP and port number

d. Send the outfile path as the first packet
e. Run an infinite while loop:
    i. Run a for loop starting with base = 0 while base < base + winsz:
        1. Read bytes from infile in segments up to MTU
        2. Send the bytes read to server using the socket as a connection point
        3. Create a file descriptor set and add the socket to it
        4. Use select() to monitor activity on socket, timeout is 10 seconds of no activity
        5. If select() == 0, timeout occurred:
            a. Print to stderr “Packet loss detected”
            b. Move pointer back with fseek()
            c. Increment retransmissions
        6. If no timeout occurred treat packet normally:
            a. Receive response (bytes) from server
            b. If the ACK sequence number matches the packet
                i. Increment the base sequence number
        7. If retransmissions count reaches 5:
            a. Print to stderr “Reached max re-transmission limit”
            b. Close infile
            c. Close socket
            d. Exit program (with failure)
    ii. If end of infile is reached via feof(infile) break from the infinite loop
f. Close infile
g. Close socket
6. Exit program (success)

Program Instructions:

1. Go to top directory (where the Makefile is) and use the command “make” to create a binary executable (for both server and client) in the bin directory
2. Change the directory to the bin directory
3. Open two terminal windows, you will need one for the server and one for the client
4. To run the server:
    a. ./myserver <port number> <drop percentage>
    b. The port number cannot be well known: 0-1024, but it may be within the range of 1024-65535, anything outside of that range is not permitted
5. To run the client:
    a. ./myclient <server IP> <port number> <MTU> <window size> <infile path> <outfile path>
    b. The server IP may be “127.0.0.1” localhost
    c. The port number cannot be well known: 0-1024, but it may be within the range of 1024-65535, anything outside of that range is not permitted (it should also be the same port number used to initialize the server)
    d. The MTU must be at least 1 byte

e. The window size (winsz) must be at least 1
f. The infile path must exist before executing the client

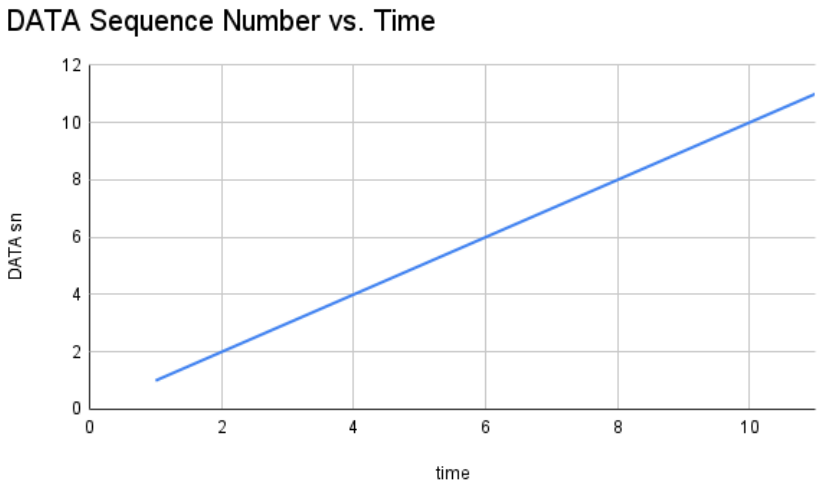
Test Cases:

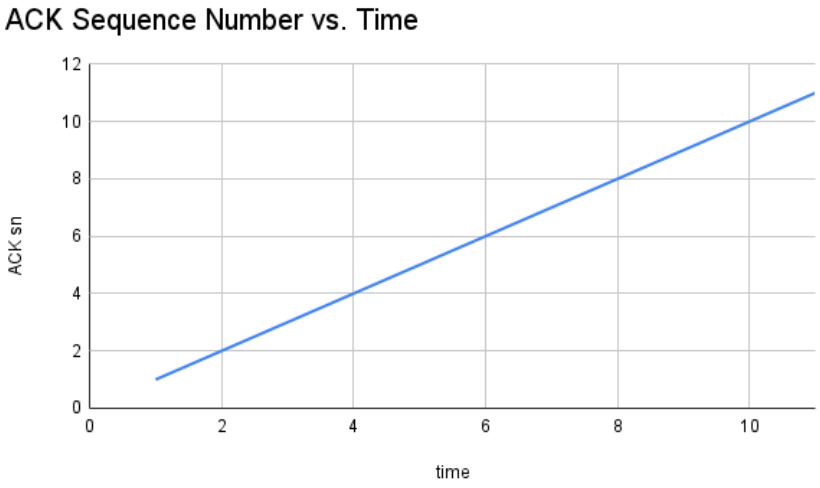
1. Invalid input: argument count
    a. ./myserver 8080
        i. Missing droppc (3 expected, 2 given)
    b. ./myclient 127.0.0.1 8080 8 infile.txt outfile.txt
        i. Missing window size (7 expected, 6 given)
2. Invalid input: negative port number
    a. ./myserver -8080 0
        i. The server socket cannot be bound to a negative port number
3. Invalid input: well-known port number
    a. ./myserver 80 0
        i. Cannot bind to a well-known port number, these are reserved
4. Invalid input: out-of-range port number
    a. ./myserver 65536 0
        i. The port number is out of range of dynamic and ephemeral ports (1024-65535)
5. Invalid input: MTU < 1
    a. ./myclient 127.0.0.1 8080 0 8 infile.txt outfile.txt
        i. If the MTU is less than 1, bytes cannot be read to be sent to the server
6. Invalid input: window size < 1
    a. ./myclient 127.0.0.1 8080 8 0 infile.txt outfile.txt
        i. If the window size is less than 1, no bytes from the client (sender) can be in transit to the server (receiver)
7. Invalid input: infile path does not exist
    a. ./myclient 127.0.0.1 8080 8 8 doesnotexist.txt outfile.txt
        i. The infile path does not exist, attempting fopen() on it will result in a NULL return
8. Invalid input: bad server IP
    a. ./myclient 10.0.0 8080 8 8 infile.txt outfile.txt
        i. sendto() will fail due to not being able to initialize the socket connection properly
9. Valid input: in-range port number
    a. ./myserver 8080 50
        i. The server is running, port number is within range 1024-65535
10. Valid input: droppc = 0
    a. ./myserver 8080 0
    b. ./myclient 127.0.0.1 8080 8 8 infile.txt outfile.txt
        i. droppc is disabled which means no packets can be dropped
        ii. The client will successfully transmit each packet on its first transmission and exits with success.
11. Valid input: droppc = 50

a. ./myserver 8080 50
b. ./myclient 127.0.0.1 8080 8 8 infile.txt outfile.txt
    i. droppc is enabled and set to 50, but the client retransmits the dropped packet and exits successfully after successful retransmission.

12. Valid input: droppc = 100
    a. ./myserver 8080 100
    b. ./myclient 127.0.0.1 8080 8 8 infile.txt outfile.txt
        i. droppc is enabled and set to 100 which means all packets are dropped
        ii. The client will attempt to retransmit the packet, but will be dropped every time until it reaches a retransmission count of 0 to which it exits with failure.

Graphs:

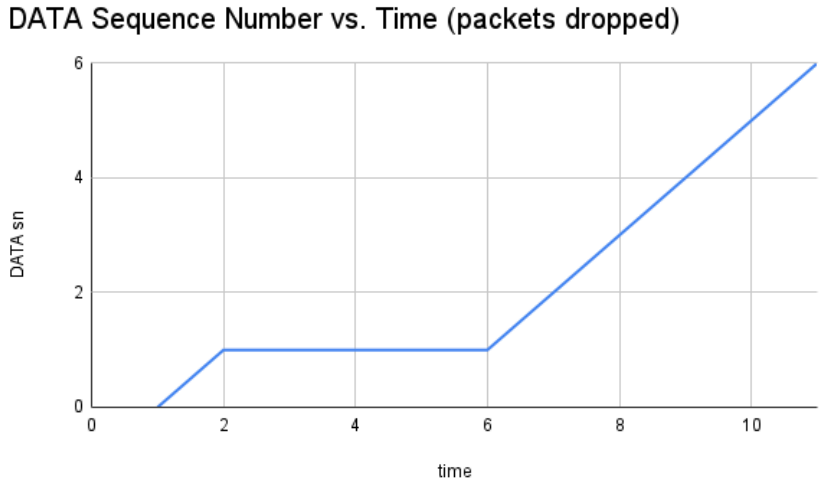


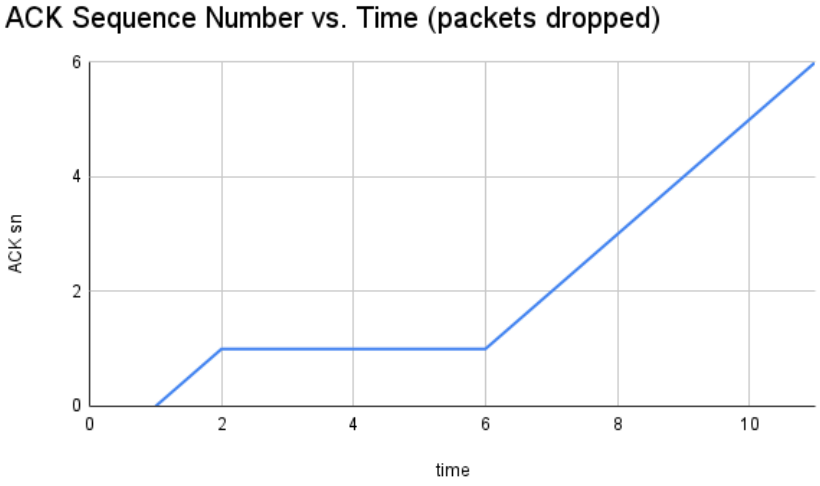


As seen above, in an ideal scenario where no packets are dropped by the server which would require retransmission on the clients end the ACK and DATA sequence numbers overtime

will exhibit a simple linear graph. This occurs since the transmission rate will remain constant and sequential, so when DATA 0 is sent the server response gives ACK 0 before DATA 1 is sent.

Lets examine what were to happen though if we were to let the server drop packets, such as setting droppc = 50 at server start up.





As seen above, at around the time DATA sn 1 is sent, the graph flatlines which indicates a packet has been dropped and is being retransmitted. Then around time = 6 the graph resumes increasing its sequence number indicating that the DATA 1 successfully got retransmitted and the rest of the packets are able to be transmitted without being dropped.