CSE156/L Programming Lab 4: Simple Reliable File Replication

Serjo Barron
sepbarro
Winter 2024

Program Design (Server):

1. Parse arguments: port number and droppc (drop percentage)
2. Check if the port is valid
3. Call srand(time(NULL)) to set random seed
4. Call “start_server()” which does the following:
    a. Create a server socket and bind it to the port number
    b. Run an infinite loop that calls “process_packet()” which does the following:
        i. Receive bytes from the client(s)
        ii. Log received DATA packet
        iii. Case 1: first packet received (contains outfile path)
            1. No droppc is applied
            2. Combine outfile path with root folder for complete file destination
            3. Send ACK to the client to confirm the packet was received
            4. The buffered data (which contains outfile path) is not written
            5. Log ACK packet
            6. Increment packet sequence number
        iv. Case 2: all subsequent packets received
            1. Calculate random number (range 0-99) using rand()
            2. If the random number < droppc:
                a. Log dropped packet
                b. Exit process_packet()
            3. If the packet is not dropped:
                a. Open the complete outfile path
                b. Write buffered data to the outfile path in append bytes mode
                c. Close outfile
                d. Send ACK to the client to confirm the packet was received
                e. Log ACK packet
                f. Increment packet sequence number
5. The program will close the server socket when terminated by the user

Program Design (Client):

1. Parse arguments: number of servers, server configuration file, MTU, window size, input path, output path
2. Open server configuration file
3. Allocate memory for each server (as an array)
4. Iterate over the number of servers (as indices):
    a. Set server IP, MTU, window size, infile path, and outfile path for each server
5. Call “start_client()” which does the following:

    a. Allocate memory for threads for each server
    b. Iterate over the number of servers:
        i. Create thread
    c. Join back created threads to the main thread
    d. Free thread memory
6. Each thread does the following:
    a. Open infile path in reading bytes mode
    b. Truncate (empty) file if it exists
    c. Create a socket, and initialize it according to the server IP and port number
    d. Create a sender thread that does the following:
        i. Send outfile path to server as the first packet
        ii. 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 the server using the socket as a connection point
            3. Log DATA packet
            4. Send ACK packet to the server
            5. Log ACK packet
        iii. Close infile
        iv. Close socket
        v. Join the sender thread back to the thread
7. Once all threads are done working program exits successfully

Program Instructions:

1. Go to the 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> <root folder path>
    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 <# of servers> <server configuration file> <MTU> <winsz> <input path> <output path>
    b. The server configuration file should be formatted as:
        i. [server IP] [port number]
        ii. Each IP and port number pair should be in its own line
    c. The server IP may be “127.0.0.1” localhost
    d. 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)
    e. The MTU must be at least 1 byte
    f. The window size (winsz) must be at least 1

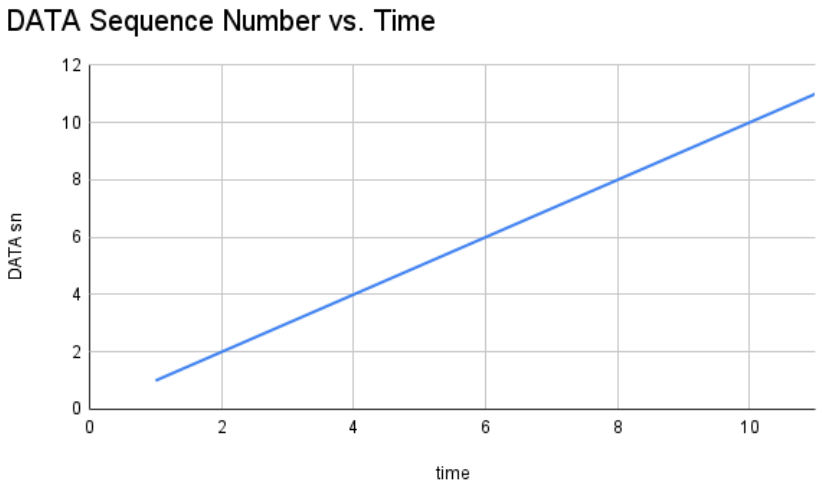
g. The infile path must exist before executing the client

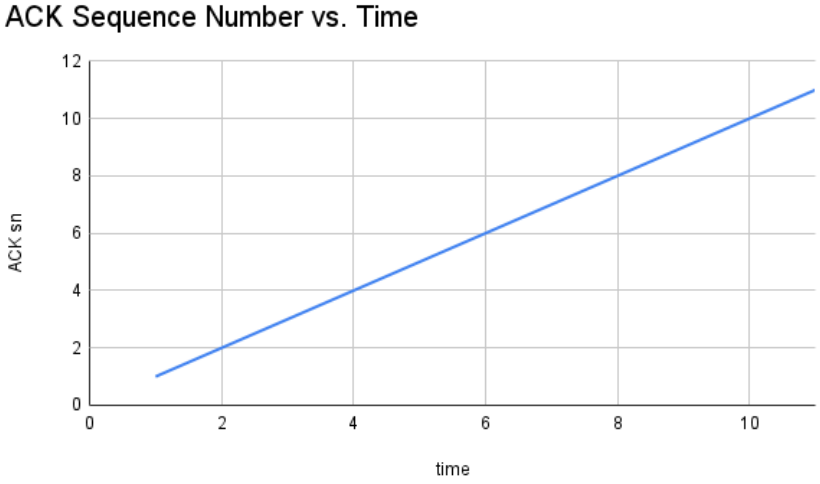
Test Cases:

1. Invalid input: argument count
    a. ./myserver 8000 0
        i. Missing root folder path (4 expected, 3 given)
    b. ./myclient 2 8 8 infile.txt outfile.txt
        i. Missing server configuration file (7 expected, 6 given)
2. Invalid input: negative port number
    a. ./myserver -8080 0 folder0
        i. The server socket cannot be bound to a negative port number
3. Invalid input: well-known port number
    a. ./myserver 80 0 folder0
        i. Cannot bind to a well-known port number, these are reserved
4. Invalid input: out-of-range port number
    a. ./myserver 65536 0 folder0
        i. The port number is out of range of dynamic and ephemeral ports (1024-65535)
5. Invalid input: MTU < 1
    a. ./myclient 2 servconf 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 2 servconf 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 2 servconf 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 2 servconf 8 8 infile.txt outfile.txt
        i. servconf contains:
            1. 10.0.0 8000
            2. 10.1.9 8001
        ii. sendto() will fail due to not being able to initialize the socket connection properly
9. Invalid input: 0 replications
    a. ./myserver 8000 0 folder0
    b. ./myserver 8001 0 folder1
    c. ./myserver 8002 0 folder2
    d. ./myclient 0 servconf 8 8 infile.txt outfile.txt
        i. servconf contains:
            1. 127.0.0.1 8000
            2. 127.0.0.1 8001

            3. 127.0.0.1 8002
        ii. The infile cannot be replicated to any servers due to 0 being specified, which in turn means no bytes are being sent/received
10. Valid input: in-range port number
    a. ./myserver 8000 0 folder0
        i. The server is running, port number is within the range of 1024-65535
11. Valid input: 1 replication
    a. ./myserver 8000 0 folder0
    b. ./myserver 8001 0 folder1
    c. ./myserver 8002 0 folder2
    d. ./myclient 1 servconf 8 8 infile.txt outfile.txt
        i. servconf contains:
            1. 127.0.0.1 8000
            2. 127.0.0.1 8001
            3. 127.0.0.1 8002
        ii. The infile is replicated to only one server, which is the first one contained in servconf on port 8001
12. Valid input: 3 replications
    a. ./myserver 8000 0 folder0
    b. ./myserver 8001 0 folder1
    c. ./myserver 8002 0 folder2
    d. ./myclient 3 servconf 8 8 infile.txt outfile.txt
        i. servconf contains:
            1. 127.0.0.1 8000
            2. 127.0.0.1 8001
            3. 127.0.0.1 8002
        ii. The infile is replicated to all 3 servers under their respective folders

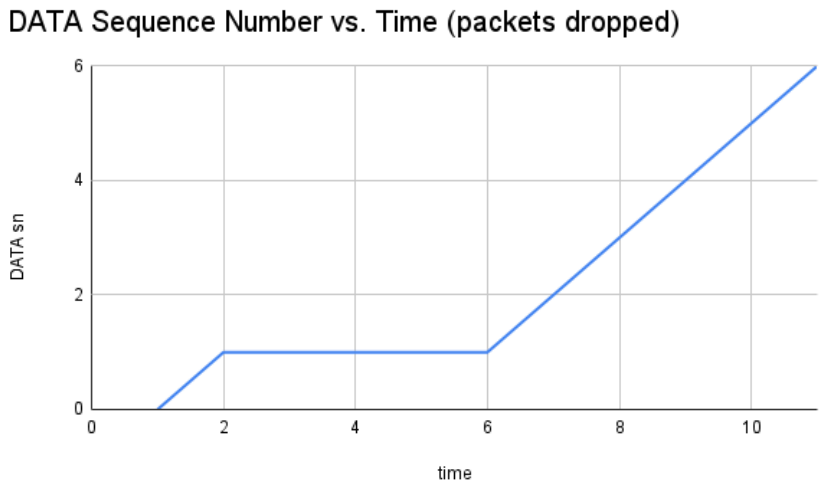
Graphs:

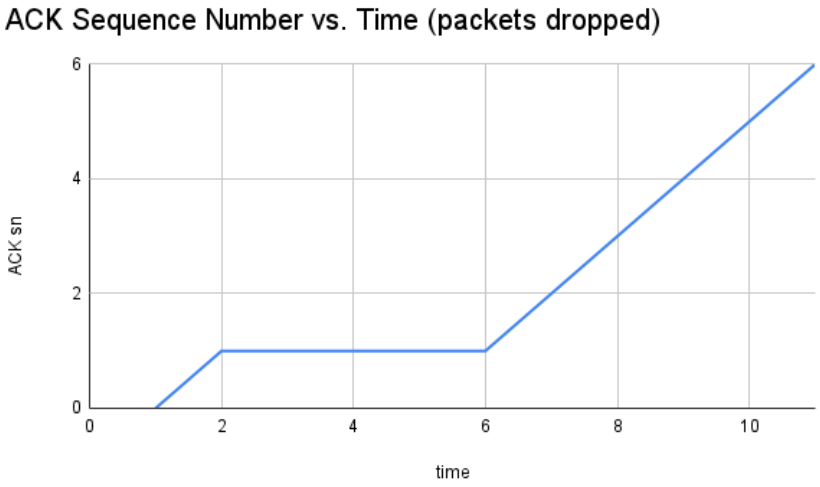




As seen above, in an ideal scenario where no packets are dropped by the server which would require retransmission on the client's end the ACK and DATA sequence numbers overtime will exhibit a simple linear graph. This occurs since the transmission rate will remain constant and sequential, so when DATA 0 is sent the server response gives ACK 0 before DATA 1 is sent.

Let's examine what would happen if we let the server drop packets, such as setting droppc = 50 at server start-up.





As seen above, when DATA sn 1 is sent, the graph flatlines, indicating 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 can be transmitted without being dropped.