UCLA Com Sci 118, Project 2

Spring 2019

Zixuan Lu, 304990072, luzixuanaa@gmail.com

Lihang Liu, 604972806, lliu0809@g.ucla.edu

Overall Design

First of all, we implemented packet.h to construct packets. It also gives us information about a packet, such as its sequence number, ack number, flags or payloads;

Next, we implemented server.cpp and client.cpp.
Server.cpp opens a socket on a given port number and waits for clients to connect to it. Once a client connects to the server, they will start a three-way handshaking. When the connection is done, client starts to send file in packets to the server, and the server stores the file with consecutive id numbers. When the transmission is done, the client automatically sends a FIN packet to the server. After receiving the FIN packet, the server replies with an ACK and its own FIN packet. As soon as the client gets the ACK packet, it will enter a 2 seconds countdown, after which it will exit the connection. During the 2 seconds countdown, the client replies to any FIN packet, but drops non-FIN ones. However, the server will continue to run and wait for the next connection from another client.
We used tc command to help to test the cases of data loss. Using diff command, we can know that the file we got from the connection was correct and uncorrupted. Using the script provided by TA, we got the following log file and cwnd/ssthresh window:

```
SEND 10680 0 512 5120 SYN
RECV 7706 10681 512 5120 ACK SYN
SEND 10681 7707 512 5120
RECV 7707 11193 512 5120 ACK
SEND 11193 7708 1024 5120
SEND 11705 7708 1024 5120
RECV 7708 11705 1024 5120 ACK
SEND 12217 7709 1536 5120
SEND 12729 7709 1536 5120
RECV 7709 12217 1536 5120 ACK
SEND 13241 7710 2048 5120
SEND 13753 7710 2048 5120
RECV 7710 12729 2048 5120 ACK
SEND 14265 7711 2560 5120
SEND 14777 7711 2560 5120
RECV 7711 13241 2560 5120 ACK
SEND 15289 7712 3072 5120
SEND 15801 7712 3072 5120
RECV 7713 14265 3072 5120 ACK
SEND 16313 7714 3584 5120
SEND 16825 7714 3584 5120
SEND 17337 7714 3584 5120
RECV 7714 14777 3584 5120 ACK
SEND 17849 7715 4096 5120
SEND 18361 7715 4096 5120
RECV 7715 15289 4096 5120 ACK
SEND 18873 7716 4608 5120
```

Figure 1: screenshot of the log file from the client side

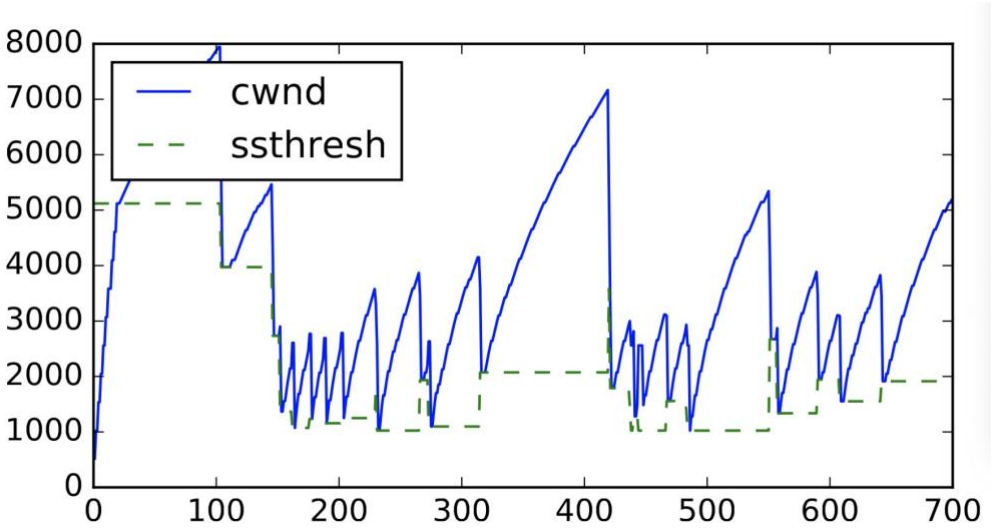

Figure 2: cwnd and ssthresh example

Implementation Description

1) Header format
   The Header of our packet is defined in packet.h, consisting of SeqNum, AckNum, len (for payload length), and flags (which specify the type of the packet). We also made paddings in it to make it 12 bytes;

2) Message
   For every message received, we call constructPacket() to turn the buffer into packets; then we print the packet derived, extract its data (seq num, ack num, flags etc.), and print them to std::out;
   For every message the client/server is sending, we use DeConstructPacket() to turn the packet into buffer; before doing so, we need to specify AckNum, SeqNum and flags of the pakcets;

3) Timeout
   As per the spec, we set the timeout value for packets to be 0.5 seconds; when no data was acked for more than 0.5s, we retransmit and take congestion control actions;

4) Window-based protocol
   We followed the instructions on the spec for Slow Start and Congestion Avoidance. cwnd is initialized to 512 and has maximum value of 10240; ssthresh is initialized to 5120; after timeout, ssthresh is set to maximum of

cwnd/2 or 1024 and cwnd is set to 512. We also implemented the extra credit part of Fast Retransmit and Fast Recovery.

Difficulties encountered

During the process of implementing this project, we encountered many difficulties. While we cannot remember all of them right now, we will discuss some of which that we can still recall.

Bugs during initializing and closing connection

Initializing connection and closing the connection is quite straightforward, yet we still encountered some bugs during the process.

One is that, in the process of three-way handshaking, SeqNum for consecutive packets are not correct. This was a little hard to debug, but we later found out that it was because we incremented SeqNum improperly. Somewhere in the code, we incremented SeqNum once after sending the packet, but then incremented it again when receiving its corresponding ACK packet.

We addressed and improved this issue by setting a convention that, we will increase SeqNum only after we send out the packet.

Arbitrarily large SeqNum occurred during file transfer

When we were testing file transferring without packet loss on large files (10 mb file), in some cases, sequence number for packets got really large (way beyond maximum sequence number) and resulted in segmentation fault in the server side. We spent some time trying to figure this problem out and found that the reason for the bug is that, in C++, char is turned into signed when building, yet we want it to be unsigned. To solve this undefined behavior, in packet.h, we casted the corresponding bit and everything worked properly.

Client could not get and print last FIN packet

When testing, we also found that during the closing stage of the connection, there were some unexpected behaviors. For example, some packets are received twice, while some were never received. This was because that Lihang and Zixuan implemented different parts of the project separately, and there were some miscommunications. When Zixuan finished sending the file, he sent a FIN packet, while Lihang also implemented the connection to send another FIN packet when

closing connection. This was kind of hard to debug, since we were not completely familiar with what the other person wrote in his part, but we still found the reason for the issue later by printing messages at different stages of the program and figured what went wrong.

There were also some other bugs we got when debugging, but probably smaller ones which we are not going to further discuss here in this report.