CSCI351 Project 4 : Buoy Networking

Name: Jennifer Liu

Date: May 1st, 2020

How does the Protocol Work?

First, the protocol needs to only do a 2 way handshake when opening a connection. The client sends the file name, and waits for the server to respond with an ACK. If it does not receive an ACK after a certain period of time, it considers the connection unavailable. It does not require a 3rd handshake because the order in which the packets arrive does not matter for this protocol as this logic is handled by the server. Hence, it does not need to exchange sequence numbers or acknowledge numbers. More details will be discussed later.

The content of this file is transferred by taking a large file (i.e 78KB image) and dividing the image into smaller pieces. The chunk size is variable. I have set my chunk size to be 5000 bytes. I send these individual chunks to the destination IP address and port. Once the client has sent all of its data, it will notify the server that it has completed its transaction using an ACK message. In my case, my ACK message was “Finished”.

The server keeps track of a few things on its end. There is an array called “receivedChunks” which keeps track of the file name of the chunks that it has successfully received from the client. The size of this array is the number of chunks there are. As the packets arrive on the server, it will update the receiveChunks to indicate which packets have been sent and which have not. Additionally, the server also caches an ArrayList containing the file name of the missing files and corrupted data.

When the server obtains a “Finished” ACK message, it will check for chunks that are missing or corrupted. The protocol determines whether the file is corrupted or not by comparing the checksum computed from the server side and the checksum that was sent from the client side. If they do not match, it means the file is corrupted. It sends a “MissingFile” ACK message to the client. This asks the client to resend these files. To handle server ACKs being dropped and client resends being dropped, I also included a timeout mechanism. If in the span of 5 seconds, I hear no response and the client still has not received a “Success” message from the server indicating that the server has successfully received all the chunks, the client sends another message to ask the Server whether everything is all settled. The Server will then send a message either indicating that it still has missing files or that it is done.

The format of the packet is shown in Figure 1.0. It is important to note that this protocol adds onto the UDP. That is why I do not include the source and destination IP and source and destination Port. These information can be obtained from UDP.

Resend Bit (0 or 1)	Filename (Variable Length)
Checksum (32 Byte)	Payload (Length of Chunk Size)

Figure 1.0

Resend Bit - Indicates whether this binary chunk is being resent or is the first time that the server sees it. If the resend bit is set, it will look at the missingFiles and corruptedFiles array, and remove the value that contains the file name, which can be obtained from the file name field. This way, we do not have to loop through the receiveChunk to see if there is anything missing each time.

Filename - The file name of chunk.
The file name format is as follows:
<file name>.part<partIdx>of<total number of chunks>.

An example, "t1.gif.part001of245"

Checksum - The checksum computed from sender
Payload - The binary data. This payload field will be the size of the chunk.

Why is this protocol more efficient than TCP/IP?

As you can see from the protocol in Figure 1.0, the header and information encoded is significantly less that of the TCP protocol. The protocol does not require any acknowledgement of any sequence number. The server side has logics that will handle when data is transferred out of order. It does so by keeping a receivedChunk array and it updates the file name of the part index as files arrive on the server end. For example, if the client sends "t1.gif.part001of245" as their file name, receivedChunk[0] would be updated with this file name. As long as the index value of receivedChunk is not null, I know the data arrived successfully. Once it determines that all files have been received, it will reassemble the image in the order of the receiveChunk.

This protocol also prevents head of line blocking, which can be a problem in TCP/IP. If a client sends a massive file, it will block all other files from preceding. This prevents that from happening since no file will be larger than chunk size + a bit more for file names and resend bits etc.

The protocol also sends significantly less packets because it does not need to send an acknowledgment for every message sent by the client. The server only responds when it needs a missing file from the client and when it needs to notify the client that it has obtained all the information to reconstruct the original file.