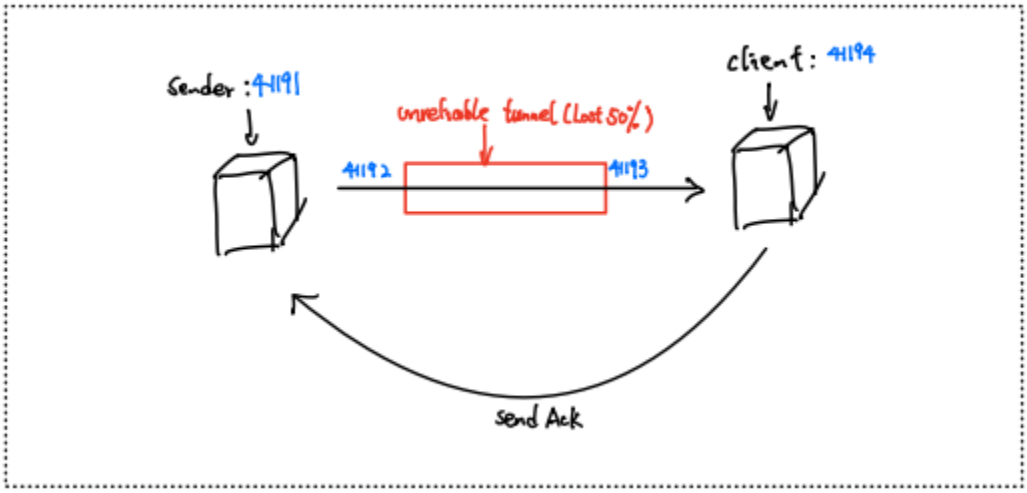
Simplex TCP - Design Documentation

The purpose of this project is to design a transport layer protocol, a.k.a. simplex version of TCP which could handle the unreliable data transmission. There are 5 parts that should be mentioned in this documentation.

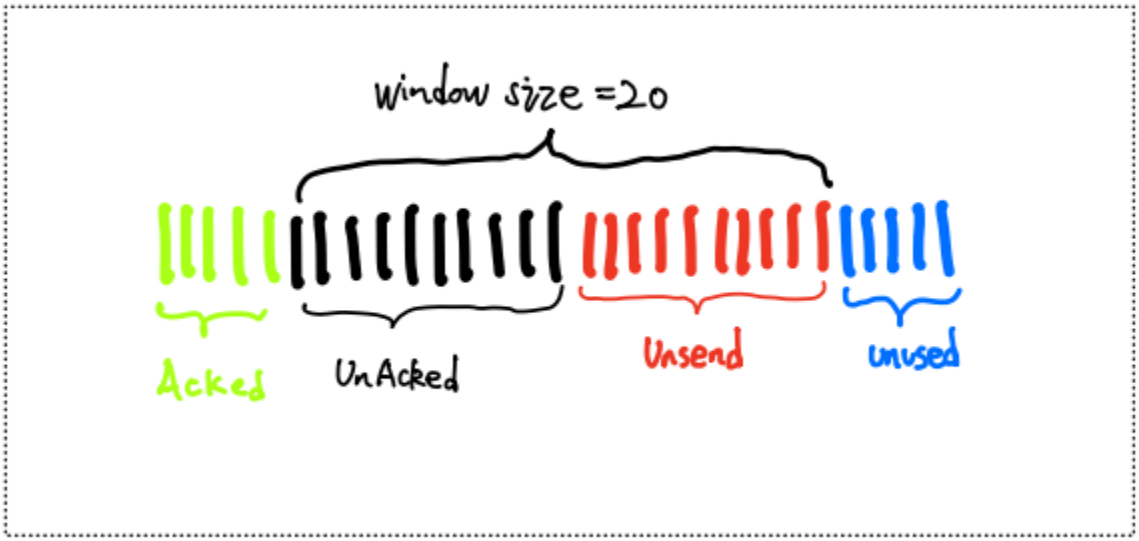
Project Framework

This system has 3 components, sender, newudp emulator, and receiver. It also should be highlighted that there are 4 ports ready for the function. For example, from the following graph we have 41194, this port is ready to receive the acks from the receiver and also serves as the source port to send packets to the receive port of emulator 41192. 41193 is the sending port of the emulator to the client 41194. And finally once the client receives the packet, it will send Ack to the sender.



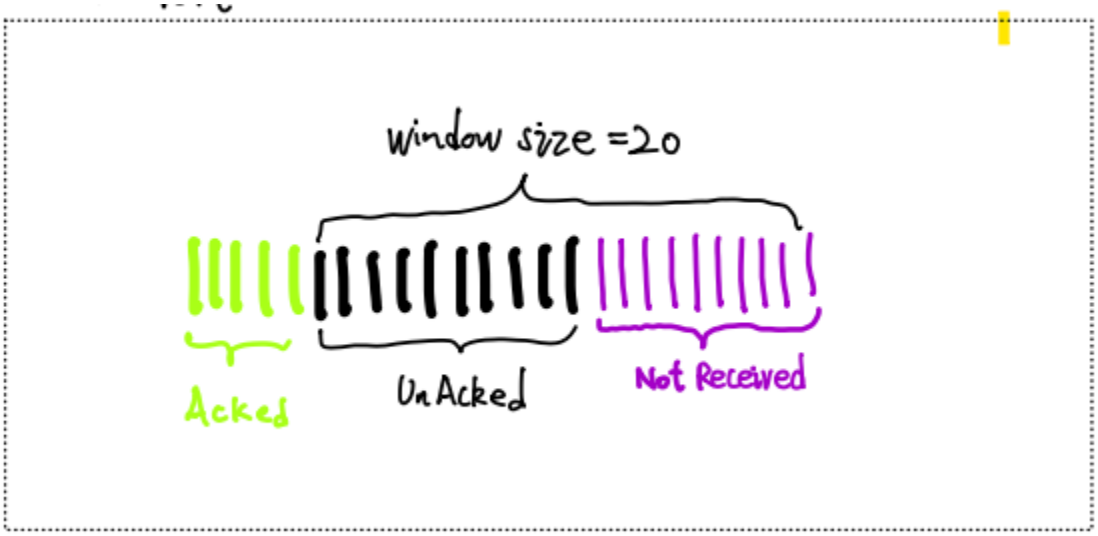
Sender Configuration

From the Sender part, for the data stream, for example from the figure below, there are 4 parts in the whole data read from the text file: acked, unacked, used, and unused data. With 20 window size for the unacked and unsend data on fly. And also after the trials many time, the retransmission time is set to 250ms is a suitable time, timeout is measured as 40ms derived from the formula TimeoutInterval = EstimatedRTT + 4*DevRTT.



Receiver Configuration

The receiver part is pretty similar to the sender. The only difference is that the receiver does not have unused data in the data stream, it only receives the data from the sender. Following is the decomposition of the data stream in sender and receiver.

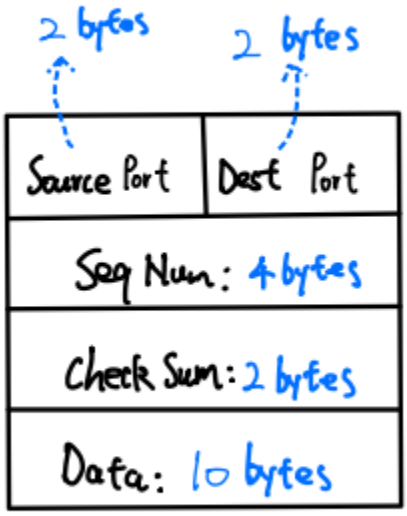


TCP Protocol

Even though the requirement is 20 bytes restriction for the whole packet size, it could be compressed nicely to utilize the size efficiency. There are 5 parts in total in the packet. With Source IP and Destination IP (the range for them is 0-65535), so 2 bytes could represent them individually. We should set a sequence number big enough to deliver all the packets, a 4 bytes number is enough (0 - 4294967295) for the packet numbers, the equivalent data size is 40GB. And the data we occupy 10 bytes every time to send 10 ASCII characters each time. The remaining 2 bytes are allocated for the checksum number (we'll discuss the checksum mechanism later).

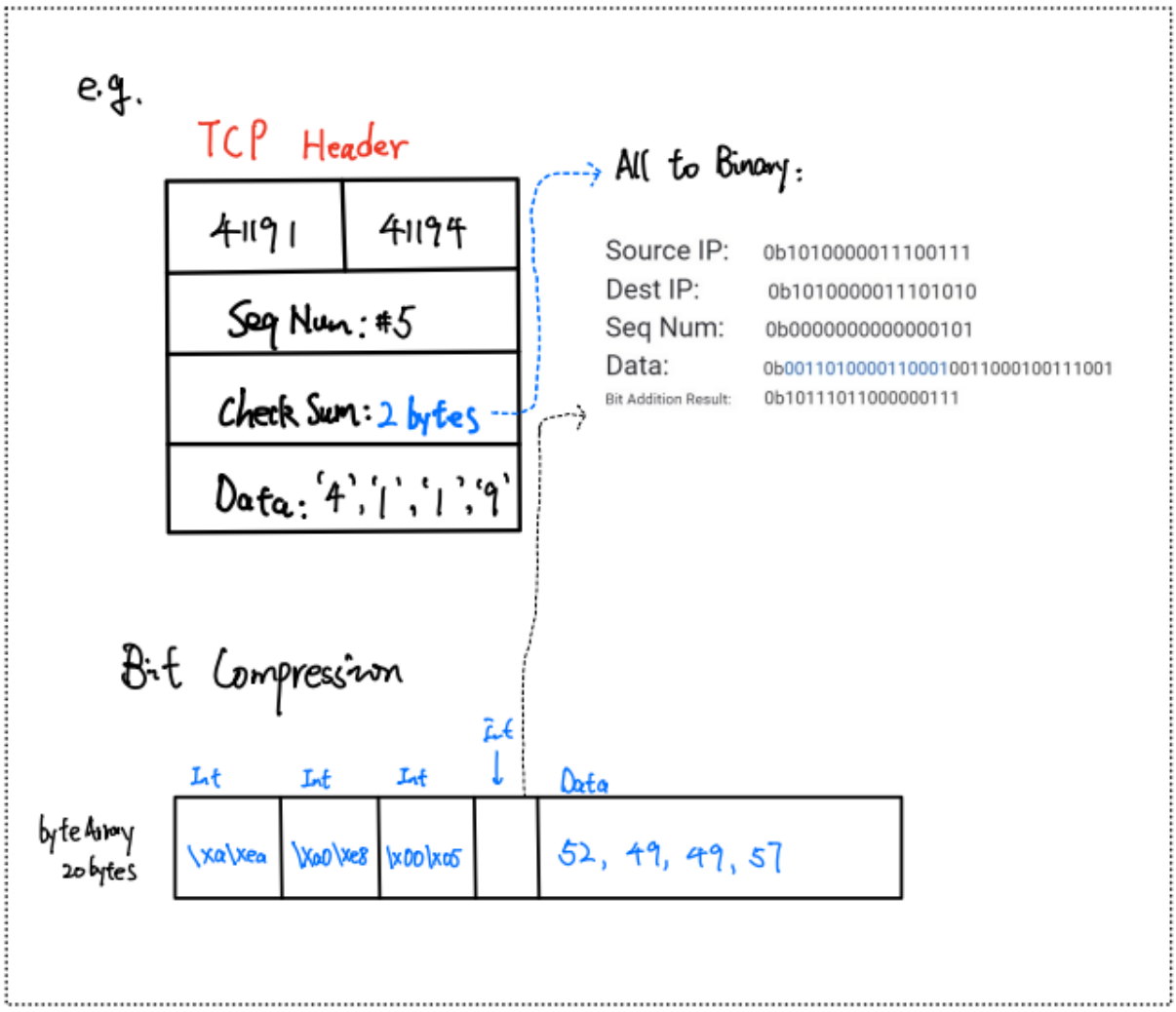
TCP Protocal

TCP Header



Error Detection

There are 3 possible situations we will encounter in the newudp tunnel: packet lost, packet disorder, bit error. The first error will be adjusted by our TCP timeout mechanism, the later 2 could be detected by our checksum mechanism. The following is a great example to illustrate how it works, suppose we have the following configuration, we will use the bit addition to add the (Source IP + Destination IP + Sequence Number + first 4 bytes of the Data) >> 2 bytes to get the 2 bytes checksum result for the receiver comparison. The following is the detailed explanation.



Sender Code

The sender should keep doing 2 tasks. First, it should choose the packet with the correct sequence. The selection should be based on: select the most recent expired/unsend packet. Then it should send this packet with a corresponding data sequence to the receiver. On the other hand, it should also keep receiving the Acks from the receiver and mark the Ack table to True for that specific Sequence Number. The technique to keep these two tasks running simultaneously is using threads, we keep two threads to run them individually. Also we need to lock the CUR_ACK_TF_TABLE once it needs to make a change.

Receiver Code

For the receiver part, we need to keep receiving the 20 bytes data from the sender, and once it receives it, it needs to recalculate the checksum and compare with the original checksum to prevent the packet corruption situation. The data structure to hold all packet data before it dumps to the file is a hash table with the sequence number as the key, data as the value. And once it receives all the data, the program terminates and dumps it to the output file.

Termination Mechanism

The termination mechanism is pretty simple, we just need to keep the first sequence number (Seq Num=0) stuffed with a packet number sent to the receiver. In the receiver, it should consistently keep detecting whether it received the Seq Num=0 packet, but finally it will receive. Then the termination condition will become the length of the hash table == packet number+1.