CS 118
Project 2 Report

Zhen Feng
Seasnet Login:zfeng
FNU Pramono
Seasnet Login:fnu

Implementation description:

For this project 2, we are implementing GBN protocol over the UDP and to make the data transfer become a reliable data transfer, we basically built the transfer detection in the following ways.

1.Header format:

```
struct pkt
{
  // define the 1 means request
  // 2 means normal data
  // 3 means Acknowledgement
  // 4 means coonection is over
  // 5 means "retransmitting ack" to inform sender to do retransmit
  int type; // 4
  int errortype; // size 4, 0 means corrupted,  1 means good
  int sequence; // size 4
  int loss; // size 4, 0 means loss, 1 means good
  size_t length; // size 8
  char buffer[1000];
};
```

We use 1024 byte as the basic transfer unit between the sender and receiver, then this packet include header information to indicate some property about this packet. First is "type". It has 5 types. Type=1 means request, it indicates the file request and the data in this type of packet is the requested file name. Type=2 the data description to indicates this packet is just a normal data segment. Type = 3 means ack, which is used by receiver to indicate the successful packet receiving. Type= 5 is retransmit ack, which is used by receiver to tell sender that the packet specified by the sequence number carried by retransmit ack should be transmit again. Second is "errortype", when errortype is 1, it means this packet is good not corrupted and 0 means corrupted. This is just like a checksum. Third one is the "sequence" number, which indicates the order of the packet. Ack with seq# m means that receiver has successfully receive up to #m packet. The forth one is loss to indicate the packet is loss or not. Fifth one is the "length", it shows the size of file data which is actually carried by this packet.

One more thing I want to mention is that to let the packet carry some flag to indicate loss or corrupted is actually more reasonable, since in the real life, corrupted or loss usually happen halfway during the packet transmitting and receiver and sender are good to receive and send so normally packet itself is the reason to cause corrupted or loss. And another is that we try to use the rand function in C to randomize loss and corrupted by accept the packet or not accept upon receiving at First. However due to the some internal "rand" function library implementation issue, that way doesn't work properly, so we choose to let the packet carry the flag of loss and corruption. And this way can also simulate the loss and corruption and works pretty well, so I stick with this implementation.

2.Timeout mechanism:

To maintain a reliable data transfer, we have to set up a timer for each packet to detect whether that packet is loss or not. We use the select function and it is a built-in library

function which works like a trigger. Which means if the current socket doesn't receive anything during some specific time, then select function will return 0 and it will return some positive number if it receive some message in the time period.

So if the time is out, our sender will just retransmit all unacked packets in the current window to the receiver.

3. Sender and receiver accept mechanism:

For the sender side, at the very beginning, after the request, the sender will send maximum of packets stored in the current window to the receiver and every time the sender receive an good normal ack with a sequence number #m, then the sender will just send the packet with sequence number #m+1 and kick out all the packet with seq number less then m+1 of the current window. If the sender receives a packet with loss flag or corruption flag then sender will just ignore it and do nothing. If it receive a good retransmitting ack with seq # m, then the sender will retransmit the packet with number #m+1 after timeout. The sender doesn't retransmit immediately because GBN will generate many duplicate retransmit acks so the sender should do retransmit after receiving all those "retransmitting ack". If the sender doesn't receive any ack after some time period, the sender will just retransmit all unacked packets in the current window to the receiver.

For the receiver side, if the receiver receives the packet, which is loss then the receiver will just ignore it. If the receiver receive a corrupted packet, then receiver knows that there is actually a packet arrived but it is not good to use, so it is better for sender to retransmit that packet. Therefore the receiver will transmit a "retransmit ack" with the #m which is the largest successful receiving sequence number that the receiver currently get. If the packet is actually good but the sequence number it carries is equal to one plus the largest successful receiving sequence number that the receiver get before this newly arrived packet, then receiver knows it is the packet that is good to use, so the receiver will just receive it and increase the largest successful receiving sequence number by one. If the sequence number the new packet carries doesn't satisfy the above situation, there will be two ways to handle. If the sequence number is larger, then the receiver will do the something as if it receives the corrupted packet. if the number is actually smaller, then the receiver will just ignore it because the receiver has received it already.

Difficult we met:

We did encounter several difficulties for our project. First to set up the UDP we need to set up the socket from the scratch. However we refer to the first project from our first project. The other difficulties we find out that is how to simulate the data so that it will be loss or corrupted. We have to come up with our own probability generated function in order to make the file loss or corrupted.

One of other common difficulties is getting familiar with the library that we are going to use, for this difficulties we only need to spend some amount of times to read the documentation of the library.

The most difficult part is actually the part of simulating the loss and corruption. Since at first we want to let sender or receiver to accept some thing in probability to implement the loss and corruption, however the rand library function in C doesn't work properly is it is being constantly called by only the main function so we try to bind the loss and error type in the packet to make it work

Last but not least is that when he handle the retransmission ack and ask the server to retransmit the file. For some reasons we have segmentation fault in this part of having not the correct logic. But we have fixed it and using the right logic, now everything works well.

Conclusion:

In conclusion, this is one of the project that provide you a lot of knowledge of networking. It is worth to spend some times dealing with this project and gain some knowledge from the project.