Computer Science 118 Winter 2018

Computer Network Fundamentals

University of California, Los Angeles

Project 2

Simple Window-based Reliable Data Transfer

by

Shiqi Wang (304582601)

Chenyu Xu (204584747)

March 12th 2018

1. Introduction

UDP is the simplest design of the transport protocol, because it is connectionless and unreliable, so it provides “No Service” and is robust. However, people find it is useful to make a reliable data transfer, that is, we can make sure we will deliver all the data, and the client can align all the data in the right order. Thus, some methods have been proposed by people, including Go-Back-N, Selective Repeat and now popular TCP. Here, we try to implement the reliable transfer protocol based on UDP and Selective Repeat, to cover all the problems when the transfer is delayed, out of order or encounters loss.

2. High-Level Description of the Server Design

1. packet header format:

```
class Packet{
public:
    ... ...
private:
    //header size: 4 + 2 + 2 bytes
    char SYNbit;    //0 or 1
    char FINbit;    //0 or 1
    char ERRbit;    //file cannot open or doesn't exist: 0 or 1()
    char REQbit;    //true if server receives requesting file name; if client
receives close connecting ack
    short SEQ;      //-1 or num
    short ACK;      //-1 or num

    //file content
    std::string data;
};
```

We decided to use four flags to represent certain special use packets except regular data packets, and we use *short* for sequence and ACK numbers because the range is 0 to 30720, and we want to maintain an 8 bytes header. After the 8 bytes header is the data.

2. timeout design:

We use *<time.h>* library functions to create timer. We use *timer_create(...)* to create our timer for each sending packet. Then, we use *timer_settime(...)* to set the timeout for each packet. When the timeout has been triggered, we write a signal handler to find which

timer has been triggered, and retransmit the corresponding packet. Also, when we received an ACK before the timeout, we delete the corresponding timer.

3. window-based design:
We limit our window size, so during transformation, at most *windowsize / max_packetsize* packets can be sent simultaneously. Thus, after SYN, at most 5120/1024 packets can be delivered immediately, and we keep our window to a upper limit 5, and move the window forward unless we receive the ACK for the first packet of the window. If the file is small, we will just use the smallest number of packets to transfer instead.

3. Difficulties during the project

1. Although it seems trivial to use threads and loops to implement timer, we think the signal timer may be better because it will consume fewer resources. However, this design complexes the code, and we encountered memory bugs which confused us a lot of time. Finally, it is in a good shape.
2. The FIN/FIN-ACK closing algorithm seems to suffer the Byzantine Generals' Problem, that if there happens to some specific packet losing scenario, the server or the client will hang there. In order to solve this problem, we first make sure the FIN-ACK from client to server is received by server, then we add an additional timer to client: after 10RTO, if client does not receive anything from server, it will close. (to handle the last FIN-ACK loss from server to client)

4. Conclusion

In this project, we have explored the reliable data transfer ideas, and tried to implement the basic functionalities with UDP unreliable transport protocol. Also, we have included many designs on packets, buffers and timers. It is a good practice to understand the transport protocol better and be familiar with C++.