CS118 Project 2 Report

Baixiao Huang
UID:504313981
Zhixin Wen
UID: 504277662

Design:

Our protocol is implemented on top of UDP. We uses GobackN to acheive reliable data transfer.

Header Design:

16 bit field	16 bit field
16 bits checksum	unused field
Sequence Number	Sequence Number
ACK Number	ACK Number
Data Length	Command
Data……...	Data……….

From above table, we can see that header is of size 128 bits. Or 16 bytes.
There are 6 command type
0: DATA Packet is used for data transfer.
1: REQUEST Packet is a request for file
2: ACK Packet is an ACK packet
3: FIRST_PACKET Packet is first data packet
4: LAST_PACKET Packet is last data packet
5: LAST_ACK Packet is last ACK for last packet

These commands tell either sender or receiver what type of the packet is.

GobackN Design:

Receiver:
Receiver first send a request packet (with command 1) and put file name in data field, send it through the UDP port to destination address. Receiver then wait for sender to send back first packet with command field of FIRST_PACKET. Then, for every new packet receiver receives, it check for integrity of packet by calculating the checksum. Checksum is compared with checksum field of the packet. If data is not corrupted, receiver save the data to file and returns an ACK packet with ACK number of next expected packet sequence number (next expected packet sequence number is equal to currently received packet sequence number plus the data length of the packet). Otherwise, when data is corrupted, it drops the packet and sends back an ACK packet with ACK number of currently expected packet sequence number. When receiver encounter an uncorrupted packet with command field LAST_PACKET, it returns packet with command field LAST_ACK. If LAST_ACK is lost or corrupted (can be detected if sender still keep sending packet), receiver send another LAST_ACK packet back, until packet is not lost or corrupted.

Sender:
Sender wait for incoming request. If there is uncorrupted incoming request packet(wiht command field of REQUEST). It look for the file in it’s disk, if file is not found, sender sends an empty file back. Then if reciever receive an empty packet, it can know a file does not exist. If file does exist, sender start to maintain a window. A window contains the sequence number available to be sent. And sender will burst a whole window of data to receiver, starts a timer and then wait for the ACK. If a uncorrupted in-order ACK is received, window moves forward and new packet will be sent. If sender wait for long period of time without receiving a uncorrupted in-order ACK packet, sender’s timer will timeout and the whole window is sent out again. Window stops incrementing when sender sends out last byte of the file. The last packet will contain LAST_PACKET in command field. Sender will then wait for LAST_ACK from the receiver. If timer timeout, last window will be sent again, then sender starts waiting again. If LAST_ACK is finally recieved, this transection ends, sender then wait for new request.

Simulation:

The simulator simulates data loss and data corruption. Data loss and data corruption probabilites are given as argument of the program. The simulation is done by putting a warper on socket function send() and sendTo(). We use random number to determine if a packet is lost or corrupted. In case that data is lost, sender/receiver just not to send the packet. And in case that the data is lost, sender/receiver modify the checksum field.

Difficulties:

We encounter some difficulties in implementing timer. After implementing timer, program fails to send the data. Also we encounter some problem in resend the whole window during timeout, a file seek cannot properly place the pointer to the start of window. But we managed to solve all those problems.

How to compile:

You compile by typing

```
$: make
```

How to run:

To run sender:

```
$: ./sender <port> <window size> <data loss probability> <data corruption probability>
```

To run receiver:

```
$: ./receiver <sender IP> <sender port> <filename> <data loss probability> <data corruption probability>
```