Networks

RELIABLE DATA TRANSFER ASSIGNMENT 2

Toka Alaa Ahmed
ID(14)

Server

Overall Organization

- Create server socket
- Bind socket with Address
- Receive data using recvFrom from a client
- Fork a child to handle this client through reading the file name found in packet sent by the client, and sending back the number of packets expected.
- Start transferring packets to client using (checksum, congestion control) principles.
- Send empty packet to indicate the completion of file transferring.

Data structure:

serverMain.cpp: start running server.

Server:

read server arguments from an input file.
Create a socket , bind it and wait for a client.
Fork a child to handle the client request.
Create data sender object to send the requested data.
Print the time spent in data transferring.
Kill child and wait for another client .

DataSender: (transfer data with congestion control)

get data size of requested file.
If seqNo < base + windowsize → send packet with probability plp to client
If seqNo >= base + windowsize → check if acknowledgments are received
check time out packets
repeat until all data is transferred
print cwnds for analysis

major functions:

```
receive_acks():
   if duplicate ack:
         dupAckCount++;
         if(dupAckCount == MAX_DUP_ACKS){//fast recovery
```

```
            ssthresh = window size / 2;
            if(ssthresh == 0){
                  ssthresh = 1;
            }
            window size = ssthresh + MAX_DUP_ACKS;
            Resend first unacked packet

      } else if (dupAckCount > MAX_DUP_ACKS){
            window size+=1;
      }


if new ack:
      if(dupAckCount > MAX_DUP_ACKS){//fast recovery
         window size = ssthresh;
       }
      dupAckCount = 0;

      if(windowSize >= ssthresh){// congestion avoidance

         while (base <= ack_pck.ackno) {
                  base++;
                  miniWindow++;
                  if(miniWindow >= window size){
                        window size+=1;
                        miniWindow = 0;
                  }
            }

      }else {//slow start

            while (base <= ack_pck.ackno) {
                  base++;
                  window size++;
            }

      }
```

```
send_packet():
      -generate random number rd in the range [0,1)
      -if rd < loss_prob → drop packet
      -else :
            corrupt_packet()
            send packet to client

send_file():
      -get data size of requested file.
      -If seqNo < base + windowsize → send_packet()
      -If seqNo >= base + windowsize → check if acks are received
      -check time out packets
      -repeat until all data is transferred
      -print cwnds for analysis

check_timeout():
      -check all unacked packets
      -if current time – sent time > TIMEOUT:
            -resend packet
            -update packet’s sent time
            -ssthresh = window size/2;
            -window size = 1;
            -if(ssthresh == 0){
```

```
                        ssthresh = 1;
                  }

corrupt_packet():
      -generate random number rd in the range [0,1)
      -if rd < CORRUPT_PROB → return packet with corruption
      -else:return packet without corruption
```

Client

Overall Organization

- Create client socket.
- Create a packet with the name of the file requested and sends the packet to the server.
- Starts receiving packets from server and sending acks back until receiving an empty packet(data size = 0, packet size = headers size).

Data structure:

clientMain.cpp: start running client.

Client:

create client socket.
create a packet with the name of the file requested and sends the packet to the server.
create data receiver object to receive packets.
print transferring time.
close socket.

DataReceiver:

receive packets from server.
write data to a file.

major functions:

```
receive_message():
  while not finished:
      if (pck.seqno == base && pck.verifyChecksum()) {// accept packet
            only if it has the expected seqno
            if (pck.len != HEADERS_SIZE) {// still not the last packet
                        base++;
                        write_data(pck);
                        send ack packet
            } else {// last packet
                        base++;
                        send ack packet
                        finished = true;
                  }

      }else if (pck.seqno != base){// Receiving out of order packet
        // send duplicate ack
           ack last received packet

      }else{//corrupted packet was received
           do nothing
      }

      write_data()
```

Data structure:

-DataPacket:

```
public:
    unsigned char* create_packet(const char * data, uint16_t size, uint32_t seq_num);
    bool create_packet(const unsigned char* buff);
    bool verifyChecksum();

    uint16_t check_sum;
    uint16_t len;
    uint32_t seqno;
    char data[PCK_DATA_SIZE];
    unsigned char buffer[PCK_SIZE];
 private:
    uint16_t computeCheckSum();
```

-AckPacket:

```
public:
    unsigned char* create_packet(uint32_t ack_num);
    bool create_packet(const unsigned char* buff);
    bool verifyChecksum();
    uint16_t check_sum;
    uint16_t len;
    uint32_t ackno;

private:
    unsigned char buffer[HEADERS_SIZE];
    uint16_t computeCheckSum();
```

-Parser:

parse input files of server and client and check if they are valid.

-DataSender:

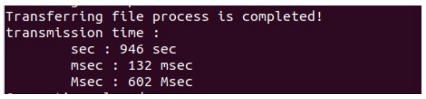
```
    float loss_prob;
    int windowSize;
    int miniWindow = 0;
    unsigned int sentData = 0;
    uint32_t seqNo = 0;
    uint32_t base = 0;
    int ssthresh = 32;
    int dupAckCount = 0;
    vector<pair<uint32_t, double>> sent_time;
    vector<DataPacket>unacked_packets;
    vector<unsigned char*>corrupted;
    vector<int>cwnd_for_analysis;
```

Bonus part:

Error detection and checksumming with 16 bit in Internet checksum
const plp is used → `CORRUPT_PROB = 0.05`

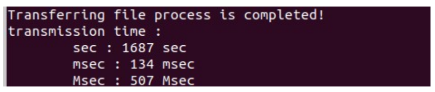
Testing examples:
– large file: 4888486 bytes
– No of packets = 9547
– max size of data in a packet = 512 bytes

plp = 0.01



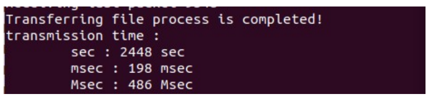
```
Transferring file process is completed!
transmission time :
        sec : 946 sec
        msec : 132 msec
        Msec : 602 Msec
```

plp = 0.05

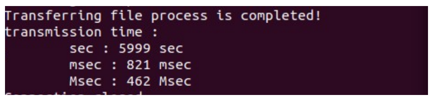


```
Transferring file process is completed!
transmission time :
        sec : 1687 sec
        msec : 134 msec
        Msec : 507 Msec
```

plp = 0.1


```
Transferring file process is completed!
transmission time :
        sec : 2448 sec
        msec : 198 msec
        Msec : 486 Msec
```

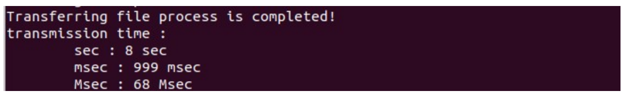
plp = 0.3

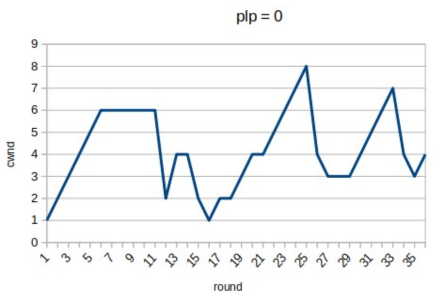

```
Transferring file process is completed!
transmission time :
        sec : 5999 sec
        msec : 821 msec
        Msec : 462 Msec
```

graphs:

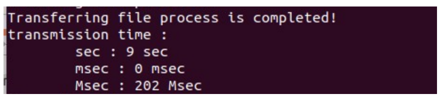
- file size: 45004 bytes
- No of packets = 87
- max size of data in a packet = 512 bytes

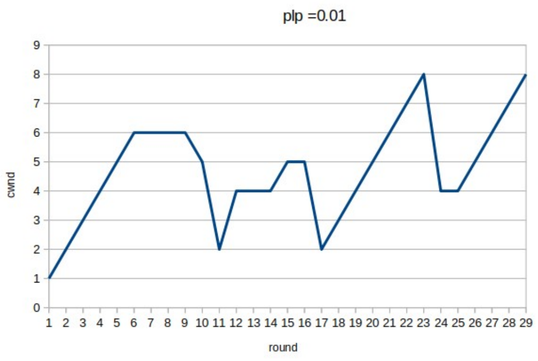
plp = 0


```
Transferring file process is completed!
transmission time :
        sec : 8 sec
        msec : 999 msec
        Msec : 68 Msec
```

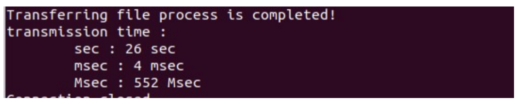


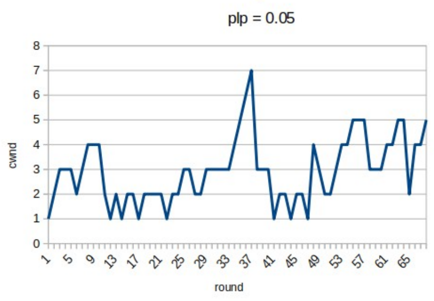
plp = 0.01


```
Transferring file process is completed!
transmission time :
        sec : 9 sec
        msec : 0 msec
        Msec : 202 Msec
```

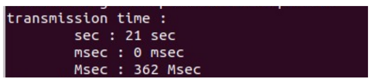


plp = 0.05

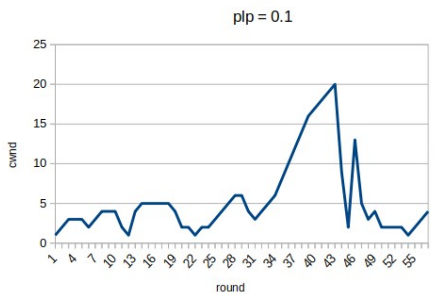

```
Transferring file process is completed!
transmission time :
        sec : 26 sec
        msec : 4 msec
        Msec : 552 Msec
```



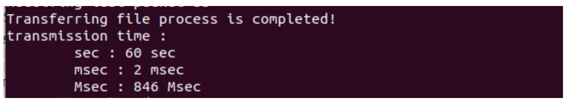
plp = 0.1



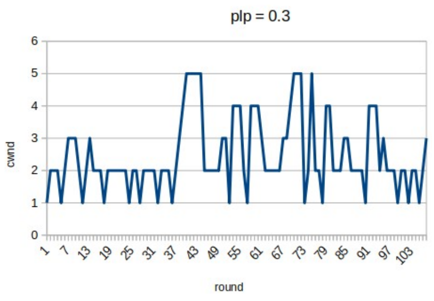
```
transmission time :
        sec : 21 sec
        msec : 0 msec
        Msec : 362 Msec
```



plp = 0.3



```
Transferring file process is completed!
transmission time :
        sec : 60 sec
        msec : 2 msec
        Msec : 846 Msec
```


plp = 0.3
cwnd
round