NETWORK LAB REPORT

ASSIGNMENT- 2

JADAVPUR UNIVERSITY

NAME – SOULIB GHOSH

SECTION – A2

CLASS - BCSE – III

ROLL – 0016105010 47

Deadline – 21st February

Submission – 28th February

Problem Statement

Implement three data link layer protocols, Stop and Wait, Go Back N Sliding Window and Selective Repeat Sliding Window for flow control.

Sender, Receiver and Channel all are independent processes. There may be multiple Transmitter and Receiver processes, but only one Channel process. The channel process introduces random delay and/or bit error while transferring frames. Define your own frame format or you may use IEEE 802.3 Ethernet frame format.

Hints: Some points you may consider in your design.

Following functions may be required in Sender.

Send: This function, invoked every time slot at the sender, decides if the sender should (1) do nothing, (2) retransmit the previous data frame due to a timeout, or (3) send a new data frame. Also, you have to consider current network time measure in time slots.

Recv_Ack: This function is invoked whenever an ACK packet is received. Need to consider network time when the ACK was received, ack_num and timestamp are the sender's sequence number and timestamp that were echoed in the ACK. This function must call the timeout function.

Timeout: This function should be called by ACK method to compute the most recent data packet's round-trip time and then recompute the value of timeout.

Following functions may be required in Receiver.

Recv: This function at the receiver is invoked upon receiving a data frame from the sender.

Send_Ack: This function is required to build the ACK and transmit.

Sliding window: The sliding window protocols (Go-Back-N and Selective Repeat) extend the stop-and-wait protocol by allowing the sender to have multiple frames outstanding (i.e., unacknowledged) at any given time. The maximum number of unacknowledged frames at the sender cannot exceed its "window size". Upon receiving a frame, the receiver sends an ACK for the frame's sequence number. The receiver then buffers the received frames and delivers them in sequence number order to the application.

Performance metrics: Receiver Throughput (packets per time slot), RTT, bandwidth-delay product, utilization percentage.

Introduction:

In data communications, flow control is the process of managing the rate of data transmission between two nodes to prevent a fast sender from overwhelming a slow receiver. It provides a mechanism for the receiver to control the transmission speed, so that the receiving node is not overwhelmed with data from transmitting node. Flow control should be distinguished from congestion control, which is used for controlling the flow of data when congestion has actually occurred. Flow control mechanisms can be classified by whether or not the receiving node sends feedback to the sending node. Flow control is important because it is possible for a sending computer to transmit information at a faster rate than the destination computer can receive and process it. This can happen if the receiving computers have a heavy traffic load in comparison to the sending computer, or if the receiving computer has less processing power than the sending computer. Flow control also plays a key role in case of noisy channel. In case of noisy channel we cannot always guaranty that the receiver have received the packet properly. Flow control deals with such situation where the data packet is lost or the sender have received any distorted packet. Flow control mechanism is implemented in the Data Link Layer.

Among all the flow control mechanism, three major flow control mechanisms – Stop and Wait ARQ, Go back N ARQ and Selective Repeat ARQ are discussed. Among them the detailed implementation of Stop and Wait ARQ and Go back N ARQ are discussed.

Overview of the Flow Control Methods and Proposed Approach to Implement That:

First of all let us discuss the definition of two term sequence number, acknowledgement number, positive acknowledgment (ACK), negative acknowledgment (NACK) and retransmission.

To distinguish between two data frames a unique number is used for each data frames which is known as sequence number.

Each acknowledgement sent from the receiver side contains a unique number which specifies the next frame to be sent is known as acknowledgement number.

When the receiver receives a correct frame, it acknowledge it using positive acknowledgement.

When the receiver receives a damaged frame or a duplicate frame, it sends a negative acknowledgment back to the sender and the sender must retransmit the correct frame.

The sender maintains a clock and sets a timeout period. If an acknowledgement of a data-frame previously transmitted does not arrive before the timeout the sender retransmits the frame, thinking that the frame or its acknowledgement is lost in transit. This is known as retransmission.

Stop and Wait ARQ:

The following transition occur in Stop-and-Wait ARQ protocol:

- The sender maintains a timeout counter.
- When a frame is sent, the sender starts the timeout counter.
- If acknowledgement of frame comes in time, the sender transmits the next frame in queue.
- If acknowledgement does not come in time, the sender assumes that either the frame or its acknowledgement is lost in transit. Sender retransmits the frame and starts the timeout counter.
- If a negative acknowledgement is received, the sender retransmits the frame.

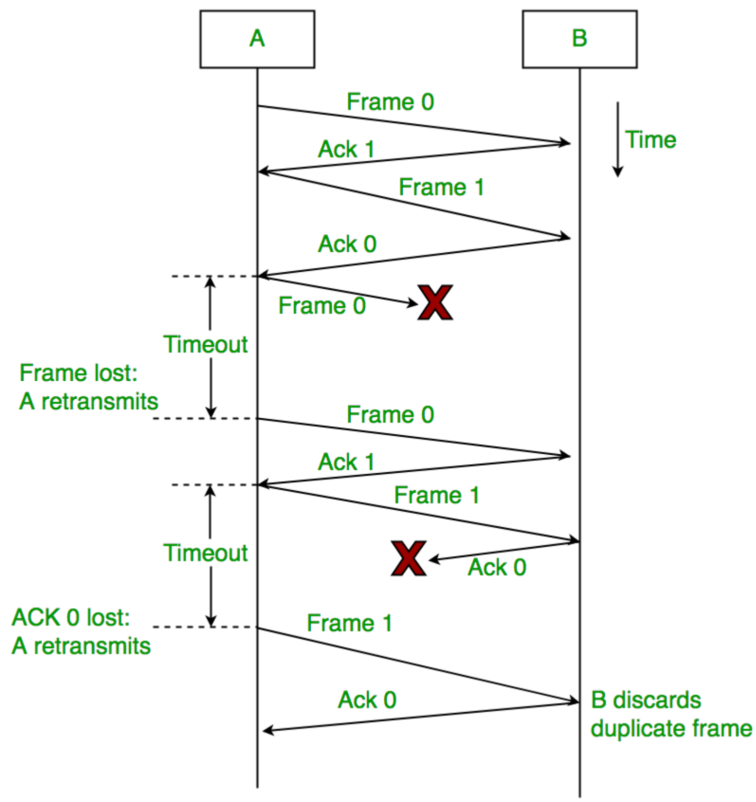

Figure 1: This figure explains how a stop and wait ARQ protocol works.

Limitations:

1) The process is very slow as the sender can send at most one new packet and wait until the time out or receiving acknowledgement.
2) The method is not robust because if the acknowledgement can get lost, when the receiver gets a packet, the receiver cannot tell if it is a retransmission or a new packet.
3) No pipeline like implementation is used.
4) Timer should be set for each frame.
5) Resource utilization is very poor as it consumes a lot of bandwidth.

Go Back N ARQ:

Stop and wait ARQ mechanism does not utilize the resources at their best. After sending packets and till the acknowledgement is received, the sender sits idle and does nothing. In Go-Back-N ARQ method, sender maintain a window or buffer. Some extra modifications are done on stop and wait ARQ protocol which are discussed. The rest processed are same.

The sending-window size enables the sender to send multiple frames without receiving the acknowledgement of the previous ones. The receiving-window enables the receiver to receive multiple frames and acknowledge them. The receiver keeps track of incoming frame's sequence number. When the sender sends all the frames in window, it checks up to what sequence number it has received positive acknowledgement. If all frames are positively acknowledged, the sender sends next set of frames. If sender finds that it has received NACK or has not receive any ACK for a particular frame, it retransmits all the frames after which it does not receive any positive ACK.

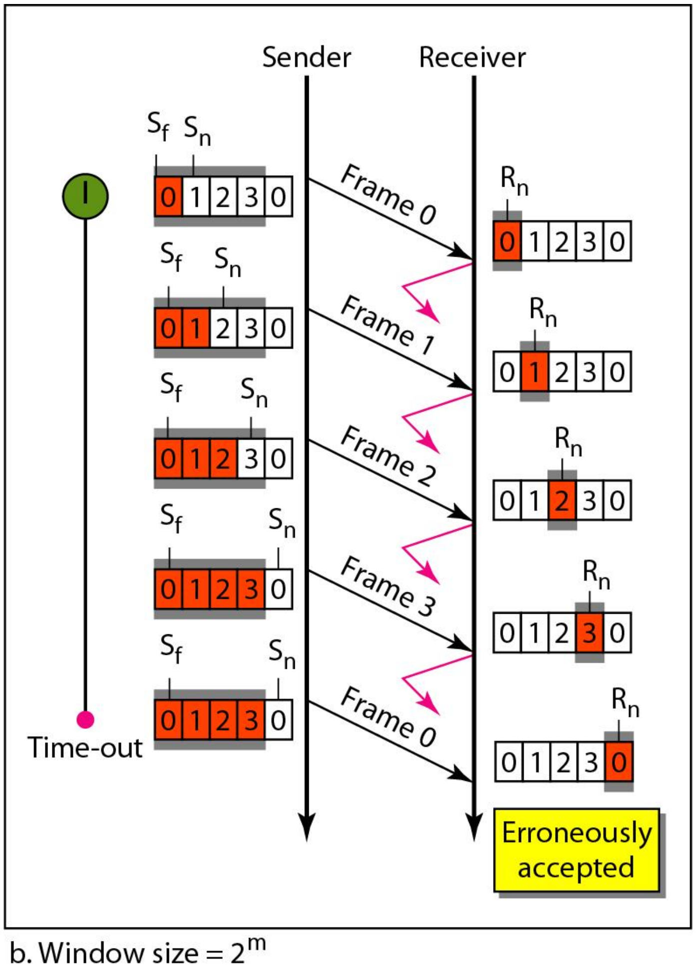

Figure 2: This figure analyzes the Go Back N ARQ protocol

Limitations:

1) Go-back-N simplifies the receiver implementation, since no buffering is needed. There is no buffer in the receiver side. The receiver can be better implemented to make the process more efficient.
2) Go back N ARQ is wasteful especially if the receiver is only missing one or two packets.
3) Scheme is inefficient when round trip delay is large and data transmission rate is high.
4) When RTT is large, for high number of NACK, a large amount of band width is wasted.

Selective Repeat ARQ:

The additional facility provided in the Selective Repeat ARQ protocol is that it contains a window or a buffer in the receiver side also which helps the receiver to process more than one frame frames at a time. Selective Repeat Protocol works better when the link is very unreliable. Because in this case, retransmission tends to happen more frequently, selectively retransmitting frames is more efficient than retransmitting all of them. In Selective-Repeat ARQ, the receiver while keeping track of sequence numbers, buffers the frames in memory and sends NACK for only frame which is missing or damaged. The sender in this case, sends only packet for which NACK is received.

Some points regarding Selective Repeat ARQ:

- Sender's Windows (Ws) = Receiver's Windows (Wr).
- Window size should be less than or equal to half the sequence number in Selective Repeat protocol. This is to avoid packets being recognized incorrectly. If the windows size is greater than half the sequence number space, then if an ACK is lost, the sender may send new packets that the receiver believes are retransmissions.
- Sender can transmit new packets as long as their number is with W of all unacknowledged packets.
- Sender retransmit unacknowledged packets after a timeout – Or upon a NAK if NAK is employed.
- Receiver acknowledges all correct packets.
- Receiver stores correct packets until they can be delivered in order to the higher layer.
- In Selective Repeat ARQ, the size of the sender and receiver window must be at most one-half of 2^m.

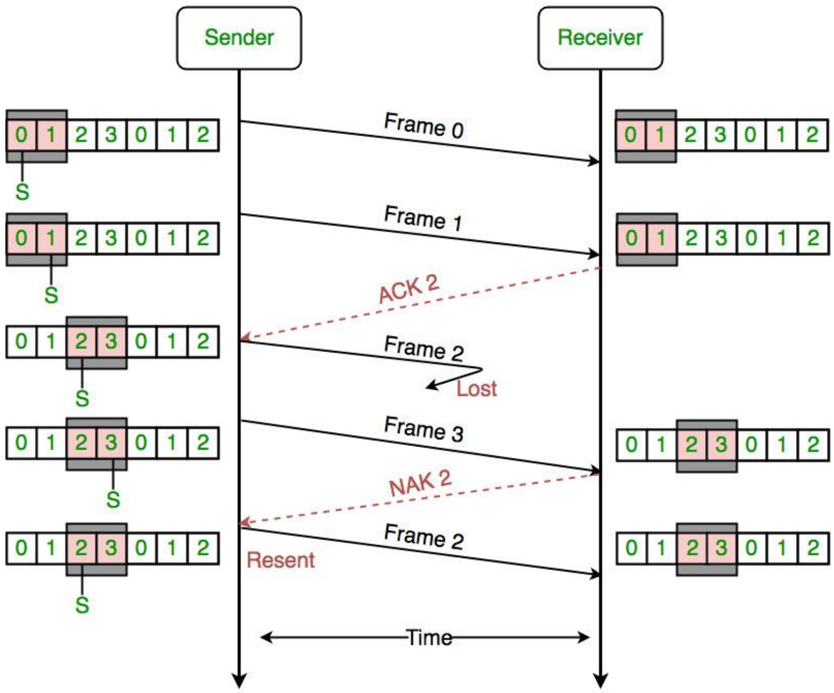

Figure 3: This figure demonstrates the Selective Repeat ARQ protocol

Limitations:

1) In this protocol, receiver may receive frames out sequence.
2) The design of sender and receiver side is very complex compared to other methods.

Proposed Approach:

To implement the stop and wait ARQ and Go back N ARQ protocol, the channel is kept same for the both cases. The purpose of data transfer is viewed as bit sharing via some shared memory. Three threads are created – sender, channel and receiver. To synchronize between them binary semaphore or mutex is used. Two message queue is used to connect channel with sender and receiver with channel. For shared memory there are Ordinary pipe, Named pipe or FIFO and Message queue. Message queue is selected because it is duplex that means we can send and receive via a message queue. Binary semaphore is used because we have only process per shared memory which are to be synchronized. The structure of the proposed approach is given below in figure 3.

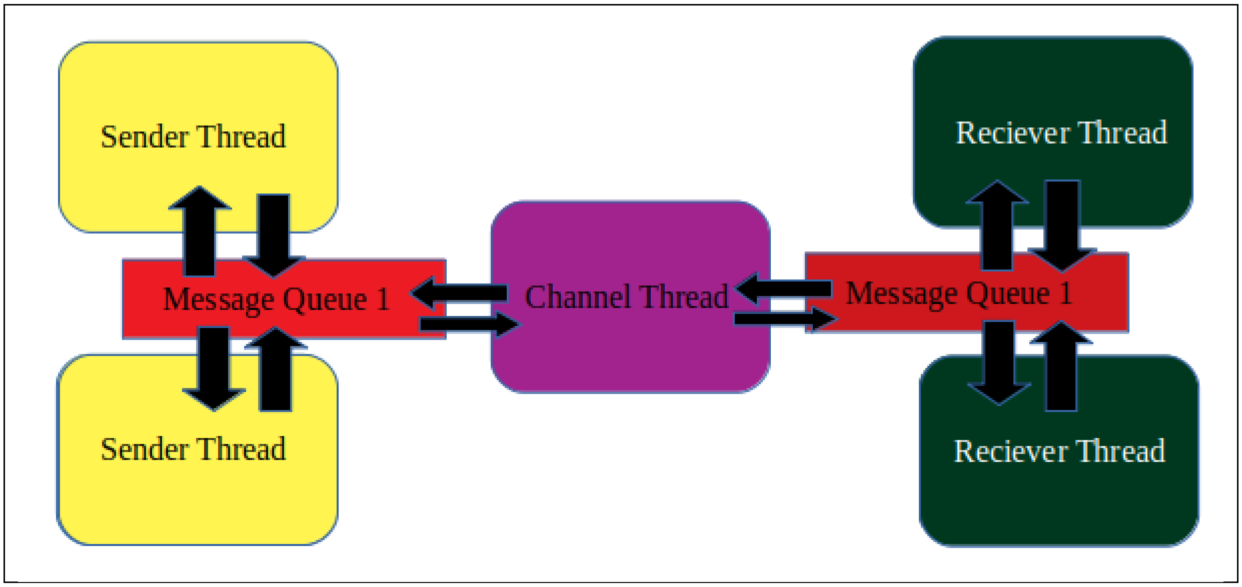

Figure 3: Diagram of the Proposed Schema for implementation of two flow control mechanism.

Some of the following cases and their solutions are mentioned below:

Channel Delay: To implement a channel delay, a random time is generated. For that time period the channel thread is kept sleep. If we generate completely random time then it may occur that the channel thread remain sleep for a huge time. To avoid those unnecessary situations, maximum and minimum sleep time is taken from the user. The random time will always generate between the minimum and maximum value.

Channel Error: In noisy channel any data packet may get lost. To implement that situation, percentage of error is taken from the user which indicate the percentage of frame which will get lost.

Error Handling: Error detection mechanism is employed with the flow control mechanism to ensure the authentication of the delivered packets. For error detection CRC 4 is used. In the sender side CRC encoder generates the CRC code word which is transferred via channel. In the receiver side, decoder decodes the code word to obtain the original data.

Time period of the timer: In general the waiting time period of the sender side is twice the propagation time. In this implementation calculation of propagation time is not possible. So, a fixed time period is considered which is 0.1 sec. It is to be noted that user must specify the maximum channel delay time less than 0.1 sec else all the packets will be timed out.

Example of a execution:

First of all the bit streams is divided into packets or frames. For error handling, CRC4 is used and the code word is generated. Then the sender thread use a message queue to send the data in channel thread. After that timer in the sender side is turned on.

The channel may reject the packet according to the error percentage supplied by the user. Then the channel will sleep during the time specified by the user. Then channel thread will use another message queue to send the data in receiver thread.

The receiver thread will receive the data and decode it. It will check whether the data is proper or not. If the data is appropriate then receiver will send acknowledgement to the channel thread via the message queue.

The channel thread will pass the acknowledgement to the sender thread via message queue. If the acknowledgement arrives before the time out then sender will send the next frame else it will resend the current frame. For Go back N ARQ if time out or NACK occurs sender will resend all the packets after that in the window or buffer.

Coding Implementation of the Proposed Approach:

To implement the mechanism, channel, encoding, protocol and decoding are same for all the cases. First of all the common code snippets are discussed then the sender and receiver code is explicitly discussed for each protocol.

Channel:

This code creates the channel thread.

```
#include "channel.h"
std:: mutex ofstream_lock;
int min_delay;
int max_delay;
int error_percent;
void channel_recieve_1(frame *r){
        msgform buf;
        msgrcv(msgId1,&buf,sizeof(buf.mtext),0,0);
        msgform_to_frame(r,&buf);
        ofstream_lock.lock();
        cout<<"----------------------------------------------------"<<endl;
        cout<<"Frame recieved FROM SENDER ready to send in the reciever side"<<endl;
        cout<<"\tSENDER:   "<<r->source_id<<endl;
           cout<<"TYPE OF FRAME :    ";
        if(r->type==1) cout<<"DATA BYTE";
        else if(r->type==2) cout<<" ACKNOWLEDGEMENT";
        else cout<<" NEGEATIVE ACKNOWLEDGEMENT";
        cout<<endl;
        cout<<"\tNUMBER:   "<<r->frame_no<<endl;
        cout<<"\tContent:  ";
        for(int i=0;i<MAX_PKT;i++) cout<<r->info.data[i];
        cout<<endl;
        cout<<"----------------------------------------------------"<<endl;
        ofstream_lock.unlock();
        return;
}
void channel_send_1(frame *s){
        msgform buf;
        frame_to_msgform(s,&buf);
        ofstream_lock.lock();
        cout<<"----------------------------------------------------"<<endl;
        cout<<"FRAME READY TO GO ON THE SENDER SIDE"<<endl;
```

```
        cout<<"\tSender:   "<<s->source_id<<endl;
            cout<<"\tTYPE :    ";
        if(s->type==1) cout<<"DATA BYTE";
        else if(s->type==2) cout<<"\t ACKNOWLEDGEMENT ";
        else cout<<" NEGEATIVE ACKNOWLEDGEMENT";
        cout<<endl;
        cout<<"\tNumber:   "<<s->frame_no<<endl;
        cout<<"---------------------------------------------------"<<endl;
        ofstream_lock.unlock();
        msgsnd(msgId4,&buf,sizeof(buf.mtext),0);
        return;
}
void channel_send_2(frame *s){
        msgform buf;
        frame_to_msgform(s,&buf);
        ofstream_lock.lock();
        cout<<"---------------------------------------------------"<<endl;
        cout<<"FROM SENDER READY TO GO TO RECIEVER"<<endl;
        cout<<"\t SENDER:   "<<s->source_id<<endl;
            cout<<"TYPE :    ";
        if(s->type==1) cout<<"DATA BYTE";
        else if(s->type==2) cout<<" ACKNOWLEDGEMENT";
        else cout<<" NEGEATIVE ACKNOWLEDGEMENT";
        cout<<endl;
        cout<<"\tNUMBER:   "<<s->frame_no<<endl;
        cout<<"\tCONTENT:  ";
        for(int i=0;i<MAX_PKT;i++) cout<<s->info.data[i];
        cout<<endl;
        cout<<"---------------------------------------------------"<<endl;
        ofstream_lock.unlock();
        msgsnd(msgId2,&buf,sizeof(buf.mtext),0);
        return ;
}
void channel_recieve_2(frame *r){
        //cout<<"recieve from reciever"<<endl;
        msgform buf;
        msgrcv(msgId3,&buf,sizeof(buf.mtext),0,0);
        msgform_to_frame(r,&buf);
        ofstream_lock.lock();
        cout<<"---------------------------------------------------"<<endl;
        cout<<"FROM RECIEVER"<<endl;
        cout<<"Sender:   "<<r->source_id<<endl;
        cout<<"Type :    ";
        if(r->type==1) cout<<"Data";
        else if(r->type==2) cout<<" ACKNOWLEDGEMENT";
        else cout<<" NEGEATIVE ACKNOWLEDGEMENT";
        cout<<endl;
        cout<<"\tNUMBER:   "<<r->frame_no<<endl;
        cout<<"---------------------------------------------------"<<endl;
        ofstream_lock.unlock();
        return;
```

```
}
void insert_error(frame *f){
            bool error;
            for(int i=0;i<MAX_PKT;i++){
                error= (rand()%100 < error_percent);
                if(error) f->info.data[i]=(f->info.data[i]) ^ 1;
            }
            return;
}
void delay(int milli_seconds)
{
  clock_t start_time = clock();
    while (clock() < start_time + milli_seconds) ;
    return;
}
void insert_delay(){
            int random=rand()%(max_delay-min_delay)+min_delay;
            delay(random);
            return ;
}
void to_reciever(){
            frame s;
            channel_recieve_1(&s);
            std::thread t(to_reciever);
            t.detach();
            insert_delay();
            insert_error(&s);
            channel_send_2(&s);
            return ;
}
void to_sender(){
            frame s;
            channel_recieve_2(&s);
            std::thread t(to_sender);
            t.detach();
            insert_delay();
            channel_send_1(&s);
            return;
}
int main(){
            cout<<"PLEASE ENTER MINIMUM DELAY:";
            cin>>min_delay;
            cout<<"PLEASE ENTER MAXIMUM DELAY:";
            cin>>max_delay;
            cout<<"PERCENTAGE OF ERROR: ";
            cin>>error_percent;
            frame f;
            srand(100);
                 std::thread s_to_r_thread(to_reciever);
                 s_to_r_thread.detach();
                  std:: thread r_to_s_thread(to_sender);
```

```
        r_to_s_thread.detach();
    while(true);
}
```

Protocol:

Protocol creates the message queue which will be used to establish connection between sender with channel and receiver with channel.

```
#include "protocol.h"
int msgId1=msgget(MSGKEY1,0777|IPC_CREAT);
int msgId2=msgget(MSGKEY2,0777|IPC_CREAT);
int msgId3=msgget(MSGKEY3,0777|IPC_CREAT);
int msgId4=msgget(MSGKEY4,0777|IPC_CREAT);

void frame_to_msgform(frame *f,msgform *buf){
    buf->mtype=f->source_id;
    memcpy(buf->mtext,f,sizeof(*f));
    return;
}
void msgform_to_frame(frame *f,msgform *buf){
    f->source_id=buf->mtype;
    memcpy(f,buf->mtext,sizeof(*f));
    return;
}
```

Encoding CRC:

Encoding CRC generates the CRC data word after using CRC4. This function is used in the sender side.

```
string generateCRCcodeword(const string& dataword){
    string zeroes="",augword=dataword,dividend,divisor;
    for(int i=0;i<CRC4.length();i++){
        zeroes.append("0");
        }
    for(int i=0;i<CRC4.length()-1;i++){
        augword.append("0");
        }
    dividend=augword.substr(0,CRC4.length());
    int pos=CRC4.length()-1;
    while(pos<augword.length()){
        if(dividend[0]=='0') divisor=zeroes;
        else divisor=CRC4;
        for(int i=0;i<CRC4.length();i++){
                dividend[i]=(char)((dividend[i]-48)^(divisor[i]-48)+48);
                }
        pos++;
        if(pos<augword.length()){
                dividend.erase(0,1);
                dividend.push_back(augword[pos]);
                }
        }
```

```
        int i=0;
        while(i<CRC4.length()-1){
            augword[augword.length()-1-i]=dividend[dividend.length()-1-i];
            i++;
            }
        return augword;
        }

string encodeCRC(string bitStream,int segSize){
        string dataword,codeword,streamCRC="";
        for(int i=0;i<bitStream.length();i+=segSize){
            dataword=bitStream.substr(i,segSize);
            codeword=generateCRCcodeword(dataword);
            streamCRC.append(codeword);
            }
        return streamCRC;
        }
```

Decoding the CRC:

This function decodes the data word to obtain the data. This function is used in the receiver side.

```
int generateCRCcodeword(const string& dataword){
    string zoeres="",augword=dataword,dividend,divisor;
    for(int i=0;i<CRC4.length();i++){
        zeroes.append("0");
        }
    dividend=augword.substr(0,CRC4.length());
    int pos=CRC4.length()-1;
    while(pos<augword.length()){
        if(dividend[0]=='0') divisor=zeroes;
        else divisor=CRC4;
        for(int i=0;i<CRC4.length();i++){
            dividend[i]=(char)((dividend[i]-48)^(divisor[i]-48)+48);
            }
        pos++;
        if(pos<augword.length()){
            dividend.erase(0,1);
            dividend.push_back(augword[pos]);
            }
        }
    return bin2dec(dividend);
    }
bool decodeCRC(string bitStream,int segSize){ decoding-------------
"<<endl<<"=====================================<<endl<<"  FRAME        SYNDROME    COMMENT
"<<endl;
        string codeword=bitStream,streamCRC="";
        int syndrome;
        bool corrupt;
    if(syndrome == 0) return false;
    else return true;
```

```
        }
```

The sender and receiver side code for each protocol is discussed below.

Stop-and-Wait Protocol:

Sender Side:

This code implements the sender side of the Stop and Wait ARQ protocol similarly as discussed before.

```
#include "protocol.h"
#include "encoder.h"
event_type event;
msgform ackbuf;
std:: mutex frame_lock,ack_lock,main_lock,timer_lock,ostream_lock;
ifstream fin("packets.txt");
int len,cur,ack_count=0;
bool timer_running=false;
void enable_network_layer(){
            fin.seekg(0,ios::end);
            len=fin.tellg();
            fin.seekg(0,ios::beg);
            return;
}

void from_network_layer(packet *p){
            fin.read((char*)&(p->data),sizeof(p->data));
            cur=fin.tellg();
            cout.flush();
            return ;
            }
void make_frame(frame *s,packet* buf,seq_nr sn){
            string CRC_codeword;
            s->source_id=1;
            s->dest_id=2;
            s->type=data;
            s->frame_no=sn;
            s->info=*buf;
            CRC_codeword=encodeCRC(string(s->info.data),MAX_PKT);
            memcpy(s->CRC,CRC_codeword.substr(MAX_PKT,CRC_codeword.length()-
MAX_PKT).c_str(),sizeof(CRC_SIZE));
            return;
}
void start_timer(){
            clock_t start_time = clock();
    while (true){
                        timer_lock.lock();
                        if(!(clock() < start_time + 100000) || !timer_running) break;
                        ack_lock.unlock();
                }
            if(clock()>=start_time+100000){
```

```
                event=time_out;
                ostream_lock.lock();
                cout<<"TIME OUT!"<<endl;
                cout<<"======================================================="<<endl;
                ostream_lock.unlock();
                main_lock.unlock();
            }
            else if(!timer_running) ack_lock.unlock();
    return;
}
void is_packet_available(){
    while(true){
                frame_lock.lock();
                if(cur!=len-1) event=send_request;
                else event=no_event;
                if(timer_running){
                            timer_lock.unlock();
                            }
                else{
                        ack_lock.unlock();
                }
        }
}
void ack_arrival_notification(){
        while(true){
        ack_lock.lock();
        //cout<<"u"<<endl;
        if(msgrcv(msgId4,&ackbuf,sizeof(ackbuf.mtext),2,IPC_NOWAIT) !=-1){
                event=frame_arrival;
            }
        //cout<<"ack unlock"<<endl;
        main_lock.unlock();
        }

        return;
}
void from_physical_layer_sender(frame *f){
        msgform_to_frame(f,&ackbuf);
        return;
}
void to_physical_layer_sender(frame *f){
        msgform buf;
        frame_to_msgform(f,&buf);
        msgsnd(msgId1,&buf,sizeof(buf.mtext),0);
        return;
}
void print_sent_frame(frame *f){
        ostream_lock.lock();
        cout<<"----------------------------------------------------"<<endl;
        cout<<"FRAME SENT"<<endl;
        cout<<"Sender:   "<<f->source_id<<endl;
```

```
        cout<<"Type :    ";
        if(f->type==1) cout<<"Data";
        else if(f->type==2) cout<<" Ack";
        else cout<<" Nak";
        cout<<endl;
        cout<<"Number:   "<<f->frame_no<<endl;
        cout<<"Content:  ";
        for(int i=0;i<MAX_PKT;i++) cout<<f->info.data[i];
        cout<<endl;
        cout<<"--------------------------------------------------"<<endl;
        ostream_lock.unlock();
        return;
}
void print_resent_frame(frame *f){
        ostream_lock.lock();
        cout<<"--------------------------------------------------"<<endl;
        cout<<"FRAME RESENT"<<endl;
        cout<<"Sender:   "<<f->source_id<<endl;
        cout<<"Type :    ";
        if(f->type==1) cout<<"Data";
        else if(f->type==2) cout<<" Ack";
        else cout<<" Nak";
        cout<<endl;
        cout<<"Number:   "<<f->frame_no<<endl;
        cout<<"Content:  ";
        for(int i=0;i<MAX_PKT;i++) cout<<f->info.data[i];
        cout<<endl;
        cout<<"--------------------------------------------------"<<endl;
        ostream_lock.unlock();
        return;
}
void print_acknowledgement(seq_nr SN){
        ostream_lock.lock();
        cout<<"ACK RECIEVED FOR FRAME "<<(SN+1)%2<<endl;
        cout<<"=================================================="<<endl;
        ostream_lock.unlock();
        return;
}
void send(){
        frame s,s_copy,r;
        seq_nr sn=0;
        packet buffer;
        bool can_send=true;
        enable_network_layer();
        main_lock.lock();
        ack_lock.lock();
        timer_lock.lock();
        std::thread request_sender(is_packet_available);
        request_sender.detach();
        std::thread ack_notif(ack_arrival_notification);
        ack_notif.detach();
```

```
        while(true){
            main_lock.lock();
            if(event==send_request && can_send){
                    from_network_layer(&buffer);
                    make_frame(&s,&buffer,sn);
                    s_copy=s;
                    to_physical_layer_sender(&s);
                    print_sent_frame(&s);
                    timer_running=true;
                    std::thread timer(start_timer);
                    timer.detach();
                    sn=(sn+1)%2;
                    can_send=false;
            }
            if(event==frame_arrival){
                    from_physical_layer_sender(&r);
                    if(r.frame_no==sn){
                            timer_running=false;
                            can_send=true;
                            print_acknowledgement(sn);
                            ack_count++;
                    }
            }
            if(event==time_out){
                    std::thread timer(start_timer);
                    timer.detach();
                    to_physical_layer_sender(&s);
                    print_resent_frame(&s);
            }
                    frame_lock.unlock();
        }
        return;
}
int main(){
        send();
        return 0;
}
```

Receiver Side:

This is the implementation of the receiver side.

```
#include "protocol.h"
#include "decoder.h"
event_type event;
msgform framebuf;
std:: mutex arrv_lock,main_lock,ostream_lock;
void frame_arrival_notification(){
        while(true){
        arrv_lock.lock();
```

```
        if(msgrcv(msgId2,&framebuf,sizeof(framebuf.mtext),1,IPC_NOWAIT)!=-1){
            event=frame_arrival;
            }
        else event=no_event;
        main_lock.unlock();
        }
        return;
}
bool is_corrupted(frame *f){
        string codeword="";
        bool is_corrupt;
        for(int i=0;i<MAX_PKT;i++) codeword.push_back(f->info.data[i]);
        for(int i=0;i<CRC_SIZE;i++) codeword.push_back(f->CRC[i]);
        if(decodeCRC(codeword,MAX_PKT+CRC_SIZE)){
                is_corrupt=true;
                ostream_lock.lock();
                cout<<"CORRUPT FRAME!!                                  "<<endl;

        cout<<"=========================================================================="<<endl<<endl;
                ostream_lock.unlock();
            }
        else is_corrupt=false;
        return is_corrupt;
}
void print_recieved_frame(frame *f){
        ostream_lock.lock();
        cout<<"------------------------------------------------------"<<endl;
        cout<<"FRAME RECIEVED:"<<endl;
        cout<<"Sender:   "<<f->source_id<<endl;
        cout<<"Type :    ";
        if(f->type==1) cout<<"Data";
        else if(f->type==2) cout<<" Ack";
        else cout<<" Nak";
        cout<<endl;
        cout<<"Number:   "<<f->frame_no<<endl;
        cout<<"Content:  ";
        for(int i=0;i<MAX_PKT;i++) cout<<f->info.data[i];
        cout<<endl;
        cout<<"------------------------------------------------------"<<endl;
        ostream_lock.unlock();
        return;
}
void from_physical_layer_reciever(frame *f){
        msgform buf;
        msgform_to_frame(f,&framebuf);
        return;
}
void to_physical_layer_reciever(frame *f){
        msgform buf;
        frame_to_msgform(f,&buf);
        msgsnd(msgId3,&buf,sizeof(buf.mtext),0);
```

```
        ostream_lock.lock();
        cout<<"CLEAN FRAME ,ACK SENT"<<endl;
        cout<<"========================================================"<<endl;
        ostream_lock.unlock();
}
void make_ack(frame *f,seq_nr Rn){
        f->source_id=2;
        f->dest_id=1;
        f->type=ack;
        f->frame_no=Rn;
        return;
}
void recieve(){
        frame r,s;
        seq_nr Rn=0;
        main_lock.lock();
        std::thread frame_arriv(frame_arrival_notification);
        frame_arriv.detach();
        while(true){
           main_lock.lock();
           if(event==frame_arrival){
                   from_physical_layer_reciever(&r);
                   print_recieved_frame(&r);
                   if(is_corrupted(&r)) goto label;
                   if(r.frame_no == Rn) Rn=(Rn+1)%2;
                   make_ack(&s,Rn);
                   to_physical_layer_reciever(&s);
           }
           label : arrv_lock.unlock();
        }
        return;
}
int main(){
        recieve();
        return 0;
}
```

Output of Stop and Wait ARQ Protocol:

Data: 1001101011010001100100010010010110100111010111111110

Sender side output:

```
----------------------------------------------------
FRAME SENT
Sender:   1
Type :    Data
Number:   0
Content:  10011001
----------------------------------------------------
TIME OUT!
====================================================
```

```
--------------------------------------------------
FRAME RESENT
Sender:   1
Type :    Data
Number:   0
Content:  10011001
--------------------------------------------------
TIME OUT!
====================================================
--------------------------------------------------
FRAME RESENT
Sender:   1
Type :    Data
Number:   0
Content:  10011001
--------------------------------------------------
ACK RECIEVED FOR FRAME 0
=======================================================
--------------------------------------------------
FRAME SENT
Sender:   1
Type :    Data
Number:   1
Content:  11100010
--------------------------------------------------
ACK RECIEVED FOR FRAME 1
=======================================================
--------------------------------------------------
FRAME SENT
Sender:   1
Type :    Data
Number:   0
Content:  00100100
--------------------------------------------------
ACK RECIEVED FOR FRAME 0
=======================================================
--------------------------------------------------
FRAME SENT
Sender:   1
Type :    Data
Number:   1
Content:  10000100
--------------------------------------------------
TIME OUT!
====================================================
--------------------------------------------------
FRAME RESENT
Sender:   1
Type :    Data
Number:   1
Content:  10000100
```

```
-----------------------------------------------------
ACK RECIEVED FOR FRAME 1
=====================================================
-----------------------------------------------------
FRAME SENT
Sender:   1
Type :    Data
Number:   0
Content:  11101011
-----------------------------------------------------
TIME OUT!
==================================================
-----------------------------------------------------
FRAME RESENT
Sender:   1
Type :    Data
Number:   0
Content:  11101011
-----------------------------------------------------
TIME OUT!
==================================================
-----------------------------------------------------
FRAME RESENT
Sender:   1
Type :    Data
Number:   0
Content:  11101011
-----------------------------------------------------
ACK RECIEVED FOR FRAME 0
=====================================================
-----------------------------------------------------
FRAME SENT
Sender:   1
Type :    Data
Number:   1
Content:  11111110
-----------------------------------------------------
ACK RECIEVED FOR FRAME 1
=====================================================
```

Channel Output:

```
PLEASE ENTER MINIMUM DELAY:1000
PLEASE ENTER MAXIMUM DELAY:10000
PERCENTAGE OF ERROR: 5
-----------------------------------------------------
Frame recieved FROM SENDER ready to send
in the reciever side
        SENDER:   1
TYPE OF FRAME :    DATA BYTE
        NUMBER:   0
        Content:  10011001
```

```
----------------------------------------------------
----------------------------------------------------
FROM SENDER READY TO GO TO RECIEVER
              SENDER:   1
TYPE :    DATA BYTE
              NUMBER:   0
              CONTENT:  00011001
----------------------------------------------------
----------------------------------------------------
Frame recieved FROM SENDER ready to send
in the reciever side
              SENDER:   1
TYPE OF FRAME :    DATA BYTE
              NUMBER:   0
              Content:  10011001
----------------------------------------------------
----------------------------------------------------
FROM SENDER READY TO GO TO RECIEVER
              SENDER:   1
TYPE :    DATA BYTE
              NUMBER:   0
              CONTENT:  10011101
----------------------------------------------------
----------------------------------------------------
Frame recieved FROM SENDER ready to send
in the reciever side
              SENDER:   1
TYPE OF FRAME :    DATA BYTE
              NUMBER:   0
              Content:  10011001
----------------------------------------------------
----------------------------------------------------
FROM SENDER READY TO GO TO RECIEVER
              SENDER:   1
TYPE :    DATA BYTE
              NUMBER:   0
              CONTENT:  10011001
----------------------------------------------------
----------------------------------------------------
FROM RECIEVER
Sender:   2
Type :     ACKNOWLEDGEMENT
              NUMBER:   1
----------------------------------------------------
----------------------------------------------------
FRAME READY TO GO ON THE SENDER SIDE
              Sender:   2
              TYPE :              ACKNOWLEDGEMENT
              Number:   1
----------------------------------------------------
----------------------------------------------------
```

```
Frame recieved FROM SENDER ready to send
in the reciever side
            SENDER:   1
TYPE OF FRAME :    DATA BYTE
            NUMBER:   1
            Content:  11100010
------------------------------------------------------
------------------------------------------------------
FROM SENDER READY TO GO TO RECIEVER
             SENDER:   1
TYPE :    DATA BYTE
            NUMBER:   1
            CONTENT:  11100010
------------------------------------------------------
------------------------------------------------------
FROM RECIEVER
Sender:   2
Type :     ACKNOWLEDGEMENT
            NUMBER:   0
------------------------------------------------------
------------------------------------------------------
FRAME READY TO GO ON THE SENDER SIDE
            Sender:   2
            TYPE :              ACKNOWLEDGEMENT
            Number:   0
------------------------------------------------------
------------------------------------------------------
Frame recieved FROM SENDER ready to send
in the reciever side
            SENDER:   1
TYPE OF FRAME :    DATA BYTE
            NUMBER:   0
            Content:  00100100
------------------------------------------------------
------------------------------------------------------
FROM SENDER READY TO GO TO RECIEVER
             SENDER:   1
TYPE :    DATA BYTE
            NUMBER:   0
            CONTENT:  00100100
------------------------------------------------------
------------------------------------------------------
FROM RECIEVER
Sender:   2
Type :     ACKNOWLEDGEMENT
            NUMBER:   1
------------------------------------------------------
------------------------------------------------------
FRAME READY TO GO ON THE SENDER SIDE
            Sender:   2
            TYPE :              ACKNOWLEDGEMENT
```

Number: 1

Frame recieved FROM SENDER ready to send
in the reciever side
SENDER: 1
TYPE OF FRAME : DATA BYTE
NUMBER: 1
Content: 10000100

FROM SENDER READY TO GO TO RECIEVER
SENDER: 1
TYPE : DATA BYTE
NUMBER: 1
CONTENT: 10000000

Frame recieved FROM SENDER ready to send
in the reciever side
SENDER: 1
TYPE OF FRAME : DATA BYTE
NUMBER: 1
Content: 10000100

FROM SENDER READY TO GO TO RECIEVER
SENDER: 1
TYPE : DATA BYTE
NUMBER: 1
CONTENT: 10000100

FROM RECIEVER
Sender: 2
Type : ACKNOWLEDGEMENT
NUMBER: 0

FRAME READY TO GO ON THE SENDER SIDE
Sender: 2
TYPE : ACKNOWLEDGEMENT
Number: 0

Frame recieved FROM SENDER ready to send
in the reciever side
SENDER: 1
TYPE OF FRAME : DATA BYTE
NUMBER: 0
Content: 11101011

```
----------------------------------------------------
----------------------------------------------------
FROM SENDER READY TO GO TO RECIEVER
             SENDER:   1
TYPE :    DATA BYTE
            NUMBER:   0
            CONTENT:  11101011
----------------------------------------------------
----------------------------------------------------
FROM RECIEVER
Sender:   2
Type :     ACKNOWLEDGEMENT
            NUMBER:   1
----------------------------------------------------
----------------------------------------------------
Frame recieved FROM SENDER ready to send
in the reciever side
            SENDER:   1
TYPE OF FRAME :    DATA BYTE
            NUMBER:   0
            Content:  11101011
----------------------------------------------------
----------------------------------------------------
Frame recieved FROM SENDER ready to send
in the reciever side
            SENDER:   1
TYPE OF FRAME :    DATA BYTE
            NUMBER:   0
            Content:  11101011
----------------------------------------------------
----------------------------------------------------
FROM SENDER READY TO GO TO RECIEVER
             SENDER:   1
TYPE :    DATA BYTE
            NUMBER:   0
            CONTENT:  11101011
----------------------------------------------------
----------------------------------------------------
FRAME READY TO GO ON THE SENDER SIDE
            Sender:   2
            TYPE :              ACKNOWLEDGEMENT
            Number:   1
----------------------------------------------------
----------------------------------------------------
FROM SENDER READY TO GO TO RECIEVER
             SENDER:   1
TYPE :    DATA BYTE
            NUMBER:   0
            CONTENT:  11101011
----------------------------------------------------
----------------------------------------------------
```

FROM RECIEVER
Sender: 2
Type : ACKNOWLEDGEMENT
NUMBER: 1

FROM RECIEVER
Sender: 2
Type : ACKNOWLEDGEMENT
NUMBER: 1

FRAME READY TO GO ON THE SENDER SIDE
Sender: 2
TYPE : ACKNOWLEDGEMENT
Number: 1

FRAME READY TO GO ON THE SENDER SIDE
Sender: 2
TYPE : ACKNOWLEDGEMENT
Number: 1

Frame recieved FROM SENDER ready to send
in the reciever side
SENDER: 1
TYPE OF FRAME : DATA BYTE
NUMBER: 1
Content: 11111110

FROM SENDER READY TO GO TO RECIEVER
SENDER: 1
TYPE : DATA BYTE
NUMBER: 1
CONTENT: 11111110

FROM RECIEVER
Sender: 2
Type : ACKNOWLEDGEMENT
NUMBER: 0

FRAME READY TO GO ON THE SENDER SIDE
Sender: 2
TYPE : ACKNOWLEDGEMENT
Number: 0

Receiver Output:

```
-----------------------------------------------------
FRAME RECIEVED:
Sender:   1
Type :    Data
Number:   0
Content:  00011001
-----------------------------------------------------
CORRUPT FRAME!!
=============================================================

-----------------------------------------------------
FRAME RECIEVED:
Sender:   1
Type :    Data
Number:   0
Content:  10011101
-----------------------------------------------------
CORRUPT FRAME!!
=============================================================

-----------------------------------------------------
FRAME RECIEVED:
Sender:   1
Type :    Data
Number:   0
Content:  10011001
-----------------------------------------------------
CLEAN FRAME ,ACK SENT
======================================================
-----------------------------------------------------
FRAME RECIEVED:
Sender:   1
Type :    Data
Number:   1
Content:  11100010
-----------------------------------------------------
CLEAN FRAME ,ACK SENT
======================================================
-----------------------------------------------------
FRAME RECIEVED:
Sender:   1
Type :    Data
Number:   0
Content:  00100100
-----------------------------------------------------
CLEAN FRAME ,ACK SENT
======================================================
-----------------------------------------------------
FRAME RECIEVED:
Sender:   1
Type :    Data
```

Number: 1
Content: 10000000

CORRUPT FRAME!!
==

FRAME RECIEVED:
Sender: 1
Type : Data
Number: 1
Content: 10000100

CLEAN FRAME ,ACK SENT
===

FRAME RECIEVED:
Sender: 1
Type : Data
Number: 0
Content: 11101011

CLEAN FRAME ,ACK SENT
===

FRAME RECIEVED:
Sender: 1
Type : Data
Number: 0
Content: 11101011

CLEAN FRAME ,ACK SENT
===

FRAME RECIEVED:
Sender: 1
Type : Data
Number: 0
Content: 11101011

CLEAN FRAME ,ACK SENT
===

FRAME RECIEVED:
Sender: 1
Type : Data
Number: 1
Content: 11111110

CLEAN FRAME ,ACK SENT
===

Go Back N ARQ:

Sender Side:

Implementation of the sender side of the Go back N ARQ protocol.

```
#include "protocol.h"
#include "encoder.h"
event_type event;
std::mutex ostream_lock,l1,l2,l3,l4;
ifstream fin("packets_goback.txt");
int len,cur;
queue<msgform> acks;
msgform ackbuf;
bool timer_running=false;
bool  cond2,cond3;
std::condition_variable cd;
std::mutex mu;
bool request_to_send=true,repeat_flag;
condition_variable cond;
int run_c;
void enable_network_layer(){
		fin.seekg(0,ios::end);
		len=fin.tellg();
		fin.seekg(0,ios::beg);
		return;
}
void from_network_layer(packet *p){
		fin.read((char*)&(p->data),sizeof(p->data));
		cur=fin.tellg();

		return ;
		}
void make_frame(frame *s,packet* buf,seq_nr sn){
		string CRC_codeword;
		s->source_id=3;
		s->dest_id=4;
		s->type=data;
		s->frame_no=sn;
		s->info=*buf;
		CRC_codeword=encodeCRC(string(s->info.data),MAX_PKT);
		memcpy(s->CRC,CRC_codeword.substr(MAX_PKT,CRC_codeword.length()-
MAX_PKT).c_str(),sizeof(CRC_SIZE));
		return;
}
void start_timer(){

		clock_t start_time = clock();
   while (true){
```

```
                        l2.lock();
                        if(!(clock() < start_time + 100000) || run_c==0) break;
                        l3.unlock();
                }
            if(clock()>=start_time+100000){
                event=time_out;
                ostream_lock.lock();
                cout<<"TIME OUT!"<<endl;
                cout<<"================================================="<<endl;
                ostream_lock.unlock();
                l3.unlock();
            }
            else if(run_c==0){
                timer_running=false;
                l4.unlock();
                }
    return;
}
void is_packet_available(){
   while(true){
                        l1.lock();
                        if(cur!=len-1) event=send_request;
                        else event=no_event;
                        if(timer_running){
                                        l2.unlock();
                                }
                        else{
                                l3.unlock();
                        }

            }
}
void ack_arrival_notification(){
            while(true){
                        l3.lock();
                bool run=run_c;
                        if(msgrcv(msgId4,&ackbuf,sizeof(ackbuf.mtext),4,IPC_NOWAIT) != -1){
                                if(run)run_c=true;
                                event=frame_arrival;
                        }
                        l4.unlock();
            }
            return;
}
void from_physical_layer_sender(frame *f){
            msgform buf;
            msgform_to_frame(f,&ackbuf);
            return;
}
void to_physical_layer_sender(frame *f){
            msgform buf;
```

```
        frame_to_msgform(f,&buf);
        msgsnd(msgId1,&buf,sizeof(buf.mtext),0);
        return;
}
void print_sent_frame(frame *f){
        ostream_lock.lock();
        cout<<"--------------------------------------------------"<<endl;
        cout<<"FRAME SENT"<<endl;
        cout<<"Sender:   "<<f->source_id<<endl;
        cout<<"Type :    "<<f->type<<endl;
        cout<<"Number:   "<<f->frame_no<<endl;
        for(int i=0;i<MAX_PKT;i++) cout<<f->info.data[i];
        cout<<endl;
        cout<<"--------------------------------------------------"<<endl;
        ostream_lock.unlock();
}
void print_resent_frame(frame *f){
        ostream_lock.lock();
        cout<<"--------------------------------------------------"<<endl;
        cout<<"FRAME RESENT"<<endl;
        cout<<"Sender:   "<<f->source_id<<endl;
        cout<<"Type :    "<<f->type<<endl;
        cout<<"Number:   "<<f->frame_no<<endl;
        for(int i=0;i<MAX_PKT;i++) cout<<f->info.data[i];
        cout<<endl;
        cout<<"--------------------------------------------------"<<endl;
        ostream_lock.unlock();
}
void print_acknowledgement(seq_nr SN){
        ostream_lock.lock();
        cout<<"ACK RECIEVED FOR FRAME "<<SN<<endl;
        cout<<"=================================================="<<endl;
        ostream_lock.unlock();
}
void send(){
        frame s,r;
        packet buffer;
        seq_nr Sn=0;
        seq_nr Sf=0;
        seq_nr Sw=7;
        map<seq_nr,frame> outstanding;
        bool timer_start=false;
        l2.lock();
        l3.lock();
        l4.lock();
        enable_network_layer();
        std::thread request_sender(is_packet_available);
        request_sender.detach();
        std::thread ack_notif(ack_arrival_notification);
        ack_notif.detach();
        while(true){
```

```
            l4.lock();
            if(event== send_request ){
                    if(Sn-Sf < Sw){
                            from_network_layer(&buffer);
                                    make_frame(&s,&buffer,Sn);
                            outstanding[Sn]=s;
                            to_physical_layer_sender(&s);
                            print_sent_frame(&s);
                            Sn=Sn+1;

                            if(!run_c){
                            run_c=true;
                            timer_running=true;
                            std::thread timer(start_timer);
                            timer.detach();
                            }
                    }
            }
            else if(event == frame_arrival){
                    event=no_event;
                    from_physical_layer_sender(&r);
                    seq_nr ack_no=r.frame_no;
                    if(ack_no>=Sf && ack_no<=Sn){
                            print_acknowledgement(ack_no);
                            Sf=ack_no;
                            if(ack_no ==Sn){
                                    run_c=false;
                                    }
                    }
            }
            else if(event== time_out){
                    run_c=true;
                    timer_running=true;
                    event=no_event;
                    std::thread timer(start_timer);
                    timer.detach();
                    seq_nr temp=Sf;
                    while(temp<Sn){
                                    frame to_resend=outstanding[temp];
                                    to_physical_layer_sender(&to_resend);
                                    print_resent_frame(&to_resend);
                                    temp=temp+1;
                    }
            }
        l1.unlock();
        }
}
int main(){
        send();
        return 0;
}
```

Receiver Side:

Implementation of the receiver side of Go back N ARQ protocol.

```
#include "protocol.h"
#include "decoder.h"
event_type event=time_out;
msgform framebuf;
bool cond;
std:: mutex ofstream_lock,mu;
void frame_arrival_notification(){
        while(true){
        msgrcv(msgId2,&framebuf,sizeof(framebuf.mtext),3,0);
        event=frame_arrival;
        }
        return;
}
bool is_corrupted(frame *f){
        string codeword="";
        bool is_corrupt;
        for(int i=0;i<MAX_PKT;i++) codeword.push_back(f->info.data[i]);
        for(int i=0;i<CRC_SIZE;i++) codeword.push_back(f->CRC[i]);
        if(decodeCRC(codeword,MAX_PKT+CRC_SIZE)){
                is_corrupt=true;
                ofstream_lock.lock();
                cout<<"CORRUPT FRAME!!"<<endl;

        cout<<"==========================================="<<endl<<endl;
                ofstream_lock.unlock();
          }
        else {
                is_corrupt=false;
                ofstream_lock.lock();
                ofstream_lock.unlock();
        }
        return is_corrupt;
}
void from_physical_layer_reciever(frame *f){
        msgform buf;
        msgform_to_frame(f,&framebuf);
        ofstream_lock.lock();
        cout<<"----------------------------------------------------"<<endl;
        cout<<"FRAME ARRIVED"<<endl;
        cout<<"Sender:   "<<f->source_id<<endl;
        cout<<"Type :    "<<f->type<<endl;
        cout<<"Number:   "<<f->frame_no<<endl;
        cout<<"Content:  ";
        for(int i=0;i<MAX_PKT;i++) cout<<f->info.data[i];
```

```
        cout<<endl;
        cout<<"--------------------------------------------------"<<endl;
        ofstream_lock.unlock();
        return;
}
void to_physical_layer_reciever(frame *f){
        msgform buf;
        frame_to_msgform(f,&buf);
        msgsnd(msgId3,&buf,sizeof(buf.mtext),0);
        ofstream_lock.lock();
        cout<<"CLEAN FRAME ,ACK SENT"<<endl;
        cout<<"=========================================================="<<endl;
        ofstream_lock.unlock();
}
void make_ack(frame *f,seq_nr Rn){
        f->source_id=4;
        f->dest_id=3;
        f->type=ack;
        f->frame_no=Rn;
        return;
}
void print_recieved_frame(frame *f){

        return;
}
void recieve(){
        frame r,s;
        seq_nr Rn=0;
        std::thread frame_arriv(frame_arrival_notification);
        frame_arriv.detach();
        while(true){
           if(event == frame_arrival){
                event=time_out;
                from_physical_layer_reciever(&r);
                if(is_corrupted(&r)) continue;
                if(r.frame_no == Rn){
                                Rn=Rn+1;
                                make_ack(&s,Rn);
                                to_physical_layer_reciever(&s);
                        }
           }
        }
        return;
}
int main(){
        recieve();
        return 0;
}
```

Go Back N ARQ Output:

Data:

10011100011000100010010011010110010011101011111111101010111011001011101010110101110011110000011111110101011101011100011101000000111111101011010
This data will be sent to the receiver side from the sender side.

Sender Side Output:

\---
FRAME SENT
Sender: 3
Type : 1
Number: 0
10011001
\---
\---
FRAME SENT
Sender: 3
Type : 1
Number: 1
11100010
\---
\---
FRAME SENT
Sender: 3
Type : 1
Number: 2
00100100
\---
\---
FRAME SENT
Sender: 3
Type : 1
Number: 3
10000100
\---
\---
FRAME SENT
Sender: 3
Type : 1
Number: 4
11101011
\---
ACK RECIEVED FOR FRAME 1
===
\---
FRAME SENT
Sender: 3

Type : 1
Number: 5
11111110

FRAME SENT
Sender: 3
Type : 1
Number: 6
10101110

FRAME SENT
Sender: 3
Type : 1
Number: 7
11001011

TIME OUT!
==

FRAME RESENT
Sender: 3
Type : 1
Number: 1
11100010

FRAME RESENT
Sender: 3
Type : 1
Number: 2
00100100

FRAME RESENT
Sender: 3
Type : 1
Number: 3
10000100

FRAME RESENT
Sender: 3
Type : 1
Number: 4
11101011

FRAME RESENT
Sender: 3

```
Type :    1
Number:   5
11111110
------------------------------------------------------
------------------------------------------------------
FRAME RESENT
Sender:   3
Type :    1
Number:   6
10101110
------------------------------------------------------
------------------------------------------------------
FRAME RESENT
Sender:   3
Type :    1
Number:   7
11001011
------------------------------------------------------
ACK RECIEVED FOR FRAME 2
==========================================================
------------------------------------------------------
FRAME SENT
Sender:   3
Type :    1
Number:   8
10101011
------------------------------------------------------
ACK RECIEVED FOR FRAME 3
==========================================================
------------------------------------------------------
FRAME SENT
Sender:   3
Type :    1
Number:   9
01011100
------------------------------------------------------
ACK RECIEVED FOR FRAME 4
==========================================================
------------------------------------------------------
FRAME SENT
Sender:   3
Type :    1
Number:   10
11110000
------------------------------------------------------
ACK RECIEVED FOR FRAME 5
==========================================================
------------------------------------------------------
FRAME SENT
Sender:   3
Type :    1
```

```
Number:   11
01111110
----------------------------------------------------
ACK RECIEVED FOR FRAME 6
========================================================
----------------------------------------------------
FRAME SENT
Sender:   3
Type :    1
Number:   12
10101110
----------------------------------------------------
ACK RECIEVED FOR FRAME 7
========================================================
----------------------------------------------------
FRAME SENT
Sender:   3
Type :    1
Number:   13
10000000
----------------------------------------------------
ACK RECIEVED FOR FRAME 8
========================================================
----------------------------------------------------
FRAME SENT
Sender:   3
Type :    1
Number:   14
00000000
----------------------------------------------------
ACK RECIEVED FOR FRAME 9
========================================================
----------------------------------------------------
FRAME SENT
Sender:   3
Type :    1
Number:   15
11111111
----------------------------------------------------
ACK RECIEVED FOR FRAME 10
========================================================
----------------------------------------------------
FRAME SENT
Sender:   3
Type :    1
Number:   16
01011010
----------------------------------------------------
ACK RECIEVED FOR FRAME 11
========================================================
ACK RECIEVED FOR FRAME 13
```

```
======================================================
ACK RECIEVED FOR FRAME 14
======================================================
ACK RECIEVED FOR FRAME 15
======================================================
TIME OUT!
===================================================
----------------------------------------------------
FRAME RESENT
Sender:   3
Type :    1
Number:   15
11111111
----------------------------------------------------
----------------------------------------------------
FRAME RESENT
Sender:   3
Type :    1
Number:   16
01011010
----------------------------------------------------
ACK RECIEVED FOR FRAME 16
======================================================
TIME OUT!
===================================================
----------------------------------------------------
FRAME RESENT
Sender:   3
Type :    1
Number:   16
01011010
----------------------------------------------------
ACK RECIEVED FOR FRAME 17
======================================================
```

Receiver Side Output:

```
----------------------------------------------------
FRAME ARRIVED
Sender:   3
Type :    1
Number:   0
Content:  10011001
----------------------------------------------------
CLEAN FRAME ,ACK SENT
=======================================================
----------------------------------------------------
FRAME ARRIVED
Sender:   3
Type :    1
Number:   1
Content:  11101010
```

CORRUPT FRAME!!
==

FRAME ARRIVED
Sender: 3
Type : 1
Number: 3
Content: 10000100

FRAME ARRIVED
Sender: 3
Type : 1
Number: 4
Content: 11101011

FRAME ARRIVED
Sender: 3
Type : 1
Number: 5
Content: 11111110

FRAME ARRIVED
Sender: 3
Type : 1
Number: 2
Content: 00100101

CORRUPT FRAME!!
==

FRAME ARRIVED
Sender: 3
Type : 1
Number: 6
Content: 10101110

FRAME ARRIVED
Sender: 3
Type : 1
Number: 7
Content: 11001011

FRAME ARRIVED

Sender: 3
Type : 1
Number: 1
Content: 11100010

CLEAN FRAME ,ACK SENT
==

FRAME ARRIVED
Sender: 3
Type : 1
Number: 2
Content: 00100100

CLEAN FRAME ,ACK SENT
==

FRAME ARRIVED
Sender: 3
Type : 1
Number: 3
Content: 10000100

CLEAN FRAME ,ACK SENT
==

FRAME ARRIVED
Sender: 3
Type : 1
Number: 4
Content: 11101011

CLEAN FRAME ,ACK SENT
==

FRAME ARRIVED
Sender: 3
Type : 1
Number: 5
Content: 11111110

CLEAN FRAME ,ACK SENT
==

FRAME ARRIVED
Sender: 3
Type : 1
Number: 6
Content: 10101110

CLEAN FRAME ,ACK SENT

===

FRAME ARRIVED
Sender: 3
Type : 1
Number: 7
Content: 11001011

CLEAN FRAME ,ACK SENT
===

FRAME ARRIVED
Sender: 3
Type : 1
Number: 8
Content: 10101011

CLEAN FRAME ,ACK SENT
===

FRAME ARRIVED
Sender: 3
Type : 1
Number: 9
Content: 01011100

CLEAN FRAME ,ACK SENT
===

FRAME ARRIVED
Sender: 3
Type : 1
Number: 10
Content: 11110000

CLEAN FRAME ,ACK SENT
===

FRAME ARRIVED
Sender: 3
Type : 1
Number: 11
Content: 01111110

CLEAN FRAME ,ACK SENT
===

FRAME ARRIVED
Sender: 3
Type : 1
Number: 12

```
Content:  10101110
----------------------------------------------------
CLEAN FRAME ,ACK SENT
==========================================================
----------------------------------------------------
FRAME ARRIVED
Sender:   3
Type :    1
Number:   13
Content:  10000000
----------------------------------------------------
CLEAN FRAME ,ACK SENT
==========================================================
----------------------------------------------------
FRAME ARRIVED
Sender:   3
Type :    1
Number:   14
Content:  00000000
----------------------------------------------------
CLEAN FRAME ,ACK SENT
==========================================================
----------------------------------------------------
FRAME ARRIVED
Sender:   3
Type :    1
Number:   15
Content:  10111111
----------------------------------------------------
CORRUPT FRAME!!
==================================================================

----------------------------------------------------
FRAME ARRIVED
Sender:   3
Type :    1
Number:   16
Content:  01011010
----------------------------------------------------
----------------------------------------------------
FRAME ARRIVED
Sender:   3
Type :    1
Number:   15
Content:  11111111
----------------------------------------------------
CLEAN FRAME ,ACK SENT
==========================================================
----------------------------------------------------
FRAME ARRIVED
Sender:   3
```

Type : 1
Number: 16
Content: 01001010

CORRUPT FRAME!!
==

FRAME ARRIVED
Sender: 3
Type : 1
Number: 16
Content: 01011010

CLEAN FRAME ,ACK SENT
===

Strength of the Proposed Method:

1) The channel delay is implemented using a random time. Some other parameters can be introduced which will be responsible to calculate the channel delay. Those parameters can be changed accordingly to maintain realistic effect or to visualize effect of specifics case where propagation time is to be examined.
2) Noise in channel is also taken into account using an error rate taken from the user. Various values can be used to monitor the effect of a noisy channel or noiseless channel. Besides, we can also examine the effect on the evaluation metrics when the noise in the channel varies.
3) There is also a scope of error handling. Here CRC4 is used. The error handling module can be changed to implement other error handling module. We just need to change a function, no need to change the entire system.

Limitations of the Proposed Method:

1) The proposed module does not contain selective repeat ARQ. There is a future plan to incorporate this protocol with this method.

Test Cases:

Here the test cases are just a sequence of bits containing 0 and 1. Every message is considered to be in binary format. The sequence is divided into frames. Each frame is send to the receiver side. To verify the method there is no special test cases. The method is examined with a sequence of 1 and 0.

Comments:

The assignment is bit difficult as there are concepts of thread, shared memory and process synchronization. The main thing which I learned from this assignment is the implementation details of three popular flow control mechanism. The assignment would be much interesting if we socket is used. Additionally, it would be more interesting if we can run the sender and receiver in separate machine and communicate between two nodes. Then we could get a proper occurrence of noise in the medium and channel delay.

END