3D3 Data Link Protocol Simulation

Hugh Lavery – 14313812

23rd Feb 2017

Description

The purpose of this project was to simulate the transfer of raw data (alphanumeric characters) from a client to a server. The data is transferred in a binary string.

Along with the 8 input characters (1 byte per char) there is a header consisting of 2 bytes containing the sequence number and payload length. The trailer contains 2 bytes to store the checksum for the input data calculated using the CRC16 algorithm that will ensure data integrity.

The server/client can hold 5 frames in its buffer. The server verifies the data is not corrupt by calculating the checksum. It then sends an acknowledgement (ACK with frame number) back along with the frame number and both the server and client remove that frame from the buffer.

If the packet is corrupt the server sends a negative acknowledgement (NAK with frame number) and the client resends the frame and waits for an ACK. This is known as selective repeat mode. The server can now write the data to the output file. It does this until all of the frames have been received.



Code

Transmitter

#include <stdio.h>

#include <stdlib.h>

#include <stdint.h>

#include <string>

#include <bitset>

#include <iostream>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <netdb.h>

#include <arpa/inet.h>

#include <unistd.h>

#define POLY 0x8408

const int INPUT = 1024;

const int PAYLOADLENGTH = 8; //number of bytes per payload

const int MAXPACKETS = INPUT/PAYLOADLENGTH;

using namespace std;

struct dataFrame{

bitset <8\*2> header; //sequence number, Payload Length

bitset <PAYLOADLENGTH\*8> coredata;

bitset <16> trailer; //checksum

dataFrame\* next;

int seq;

};

unsigned short crc16(char \*data\_p, unsigned short length){

unsigned char i;

unsigned int data;

unsigned int crc = 0xffff;

if (length == 0)

return (~crc);

do

{

for (i=0, data=(unsigned int)0xff & \*data\_p++;

i < 8;

i++, data >>= 1)

{

if ((crc & 0x0001) ^ (data & 0x0001))

crc = (crc >> 1) ^ POLY;

else crc >>= 1;

}

} while (--length);

crc = ~crc;

data = crc;

crc = (crc << 8) | (data >> 8 & 0xff);

return (crc);

}

char gremlin(char data){

int num = 0;

num = rand()%20;

if (num == 5){

data = ' ';

}

return data;

}

int main(){

char data[INPUT], buffer[256], tempchararray[3], temp[PAYLOADLENGTH];

FILE \*fp;

int clientSocket, portNum, n, QUE = 0, sentSuccesfully =0, i=0;

string tempChar, core, checksum, headerT;;

unsigned short crc;

dataFrame\* head = NULL;

dataFrame\* tempFrame = NULL;

struct sockaddr\_in serverAddr;

struct hostent \*serverName;

portNum = 12000;

//reading in data to be transmitted

fp = fopen("ASCIIdata.txt", "r");

for (int i = 0; i<1024; i++){

data[i] = fgetc(fp);

}

//connecting to socket

clientSocket = socket(AF\_INET, SOCK\_STREAM, 0);

if (clientSocket < 0)

{

printf("Error: Unable to open the socket");

return -1;

}

serverName = gethostbyname("localhost"); // host name

bzero((char \*) &serverAddr, sizeof(serverAddr));

serverAddr.sin\_family = AF\_INET;

bcopy(

(char \*)serverName->h\_addr,

(char \*)&serverAddr.sin\_addr.s\_addr,

serverName->h\_length

);

//conecting to reciever

serverAddr.sin\_port = htons(portNum);

serverAddr.sin\_addr.s\_addr = INADDR\_ANY;

if (connect(clientSocket,(struct sockaddr \*) &serverAddr,sizeof(serverAddr)) < 0)

{

printf("ERROR connecting");

return -1;

}

// prepare data to send

while(sentSuccesfully < MAXPACKETS){

//fill the buffer

while(QUE < 5){

//split data into frame to be sent

for(int j=0; j < PAYLOADLENGTH; j++){

temp [j] = data[(PAYLOADLENGTH\*i)+ j];

bitset<8> coreFrame(temp[j]);

core += coreFrame.to\_string();

}

string tempstr = temp;

//Calculate CRC16

crc = (crc16(temp, 8));

bitset<16> tempTrailer(crc);

checksum = to\_string(crc);

//Prepare data to be sent

bitset <8\*PAYLOADLENGTH> tempCore(core);

bitset<8> tempHeader1(i+1);

bitset<8> tempHeader2(64);

headerT = tempHeader1.to\_string() + tempHeader2.to\_string();

bitset<16> tempHeader(headerT);

//store in buffer until ACK recieved

if(QUE == 0){

head = new dataFrame;

head->coredata = tempCore;

head->header = tempHeader;

head->trailer = tempTrailer;

head->seq = i+1;

head->next = NULL;

QUE++;

}

else{

dataFrame\* temp = head;

//go to end of list

while (temp->next) {

temp = temp->next;

}

temp->next = new dataFrame;

temp = temp->next;

temp->coredata = tempCore;

temp->header = tempHeader;

temp->trailer = tempTrailer;

temp->seq = i+1;

temp->next = NULL;

QUE++;

}

core = "";

checksum= "";

crc = 0;

//apply gremlin function on random bit

tempCore[18] = gremlin(tempCore[18]);

//send the data to the reciever

printf("Sending Frame: %i \n", i+1);

string buff = headerT + tempCore.to\_string() + tempTrailer.to\_string();

bzero(buffer,256);

strcpy(buffer, buff.c\_str());

n = sendto(clientSocket, &buffer, sizeof(buffer), 0,(struct sockaddr\*) &serverAddr, sizeof(serverAddr));

if (n < 0)

{

printf("ERROR writing to socket\n");

return -1;

}

i++;

cout << "\n\n";

}

//When the reciever sends an ACK back for a particular frame

//remove it from the buffer or send back frame if NAK recieved

printf("waiting for ACK/NAK\n");

bzero(buffer,256);

n = recvfrom(clientSocket,&buffer,sizeof(buffer),0,(struct sockaddr\*) &serverAddr, (socklen\_t \*)&serverAddr);

if (n < 0)

{

printf("ERROR reading from socket\n");

return -1;

}

//Parse the reply

string recieved(buffer);

string reply ="";

bitset<24> replyBits(recieved.substr(0,24));

bitset<8> frameNumBit(recieved.substr(24,8));

int frameNum = (int)frameNumBit.to\_ulong();

reply = replyBits.to\_string();

for(int i=0; i <3; i++){

tempChar = reply.substr(i\*8,8);

bitset<8> replyChar(tempChar);

tempchararray[i] = char(replyChar.to\_ulong());

}

reply = tempchararray;

//remove frame acknowleged from the buffer

if(reply == "ACK"){

if(head->seq == frameNum){

printf("frame %i has been ACKed \n", frameNum);

QUE--;

sentSuccesfully++;

//delete node from linked list

dataFrame\* p = head;

head = head->next;

delete p;

}

else{

//check rest of linked list for frame

tempFrame = head;

dataFrame\* t = NULL;

while(tempFrame->seq != frameNum){

t = tempFrame;

tempFrame = tempFrame->next;

}

if(tempFrame->seq == frameNum){

printf("frame %i has been ACKed \n", frameNum);

QUE--;

sentSuccesfully++;

//delete node

dataFrame\* tp = tempFrame;

t->next = tempFrame->next;

delete tp;

}

else{

printf("ERROR: Frame %i Not found in list\n", frameNum);

}

}

}

//resend frame that was corrupted

else{

printf("Frame %i has been NAKed\n", frameNum);

if(head->seq == frameNum){

string buff = head->header.to\_string() + head->coredata.to\_string() + head->trailer.to\_string();

bzero(buffer,256);

strcpy(buffer, buff.c\_str());

printf("Resending frame\n");

n = sendto(clientSocket, &buffer, sizeof(buffer), 0,(struct sockaddr\*) &serverAddr, sizeof(serverAddr));

if (n < 0)

{

printf("ERROR writing to socket\n");

return -1;

}

}

else{

tempFrame = head;

while(tempFrame->seq != frameNum){

tempFrame = tempFrame->next;

}

if(tempFrame->seq != frameNum){

printf("error not in resend list\n");

}

else{

string buff = tempFrame->header.to\_string() + tempFrame->coredata.to\_string() + tempFrame->trailer.to\_string();

bzero(buffer,256);

strcpy(buffer, buff.c\_str());

printf("Resending frame\n");

n = sendto(clientSocket, &buffer, sizeof(buffer), 0,(struct sockaddr\*) &serverAddr, sizeof(serverAddr));

if (n < 0)

{

printf("ERROR writing to socket\n");

return -1;

}

}

}

}

reply = "";

}

printf("All frames have been acknowledged\n\n");

close(clientSocket);

return 0;

}

Receiver

//

// reciever.cpp

//

//

// Created by Hugh Lavery on 19/02/2017.

//

//

#include <stdio.h>

#include <stdlib.h>

#include <stdint.h>

#include <string>

#include <bitset>

#include <iostream>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <netdb.h>

#include <arpa/inet.h>

#include <unistd.h>

#define POLY 0x8408

const int PAYLOADLENGTH = 8;

using namespace std;

struct dataFrame{

int seq;

string data;

bitset <8\*2> header; //sequence number, Payload Length

bitset <PAYLOADLENGTH\*8> coredata;

bitset <16> trailer; //checksum

dataFrame\* next;

int sourceCRC;

};

unsigned short crc16(char \*data\_p, unsigned short length){

unsigned char i;

unsigned int data;

unsigned int crc = 0xffff;

if (length == 0)

return (~crc);

do

{

for (i=0, data=(unsigned int)0xff & \*data\_p++;

i < 8;

i++, data >>= 1)

{

if ((crc & 0x0001) ^ (data & 0x0001))

crc = (crc >> 1) ^ POLY;

else crc >>= 1;

}

} while (--length);

crc = ~crc;

data = crc;

crc = (crc << 8) | (data >> 8 & 0xff);

return (crc);

}

void writeToFile(char\* characters){

FILE \*fp;

fp = fopen("output.txt", "a+");

char t;

for(int i =0; i <PAYLOADLENGTH; i++){

t = characters[i];

fputc(t,fp);

}

}

char gremlin(char data){

int num = 0;

num = rand()%20;

if (num < 5){

data = ' ';

}

return data;

}

int main(){

int serverSocket, portNum, clientAddrLen, n, count =1, seqNumint =0, lengthNumint=0, QUE =0, crcNumint, checksumint;

char buffer[256], tempcharArray[PAYLOADLENGTH], ack[3], nak[3];

struct sockaddr\_in serverAddr, clientAddr;

string headerStr, seqNumStr, lengthNumStr, ACK, NAK, data, coredata, crcStr, coretempStr, characterTemp, checksum, reply;

portNum = 12000;

dataFrame \*head = NULL;

unsigned short crc;

dataFrame \*pTemp;

FILE \*fp;

fp = fopen("output.txt", "w");

ack[0] = 'A';

ack[1] = 'C';

ack[2] = 'K';

nak[0] = 'N';

nak[1] = 'A';

nak[2] = 'K';

//Preparing ACK and NAK replies

for (int i=0; i<3;i++){

bitset<8> ackBits(ack[i]);

ACK += ackBits.to\_string();

bitset<8> nakBits(nak[i]);

NAK += nakBits.to\_string();

}

//Opening socket

serverSocket = socket(AF\_INET, SOCK\_STREAM, 0);

if (serverSocket < 0){

printf("Error: Could not open socket");

return -1;

}

//Getting transmitter data

bzero((char\*) &serverAddr, sizeof(serverAddr));

serverAddr.sin\_family = AF\_INET;

serverAddr.sin\_port = htons(portNum);

serverAddr.sin\_addr.s\_addr = INADDR\_ANY;

if(bind(serverSocket, (struct sockaddr\*) &serverAddr, sizeof(serverAddr)) < 0)

{

printf("Error: Socket is in use\n");

return -1;

}

listen(serverSocket, 5);

clientAddrLen = sizeof(clientAddr);

//accepting incoming connect

serverSocket = accept(serverSocket, (struct sockaddr\*) &clientAddr, (socklen\_t \*)&clientAddr);

count =1;

while(count < 129){

//Fill buffer

while(QUE < 5){

usleep(100);

bzero(buffer, 256);

printf("Waiting to recieve frame %i\n", count);

//Recieving frame

n = recvfrom(serverSocket, &buffer, sizeof(buffer), 0, (struct sockaddr\*) &clientAddr, (socklen\_t \*)&clientAddr);

if(n < 0)

{

printf("Error: Could not read from socket");

return -1;

}

//Parsing Frame number from incoming header

data = buffer;

seqNumStr = data.substr(0,8);

bitset<8> seqNum(seqNumStr);

seqNumint =(int)(seqNum.to\_ulong());

//Parsing length of input data

lengthNumStr = data.substr(8,8);

bitset<8> lengthNum(lengthNumStr);

lengthNumint = (int)(lengthNum.to\_ulong());

//data extraction

coredata = data.substr(16,(lengthNumint));

bitset<64> coreBits(coredata);

//gremlin on data recieved

coreBits[30] = gremlin(coreBits[30]);

//crc extraction

crcStr = data.substr((16+lengthNumint), 16);

bitset<16> crcNum(crcStr);

crcNumint = (int)crcNum.to\_ulong();

//store in buffer

if (QUE == 0){

head = new dataFrame;

head->data = data;

bitset<8> tempSeq(seqNum);

bitset<8> tempLength(lengthNum);

string headerT = tempSeq.to\_string() + tempLength.to\_string();

bitset<16> headerBits(headerT);

head->header = headerBits;

head->coredata = coreBits;

head->trailer = crcNumint;

head->seq = seqNumint;

head->next = NULL;

head->sourceCRC = crcNumint;

QUE++;

}

else if(QUE <= 5){

dataFrame\* temp = head;

//go to end of list

while (temp->next) {

temp = temp->next;

}

temp->next = new dataFrame;

temp = temp->next;

temp->data = data;

bitset<8> tempSeq(seqNum);

bitset<8> tempLength(lengthNum);

string headerT = tempSeq.to\_string() + tempLength.to\_string();

bitset<16> headerBits(headerT);

temp->header = headerBits;

temp->coredata = coreBits;

temp->trailer = crcNum;

temp->seq = seqNumint;

temp->next = NULL;

temp->sourceCRC = crcNumint;

QUE++;

}

}

//Find next Frame in sequence and ensure data integrity

//before writing to output file (cheking head of linked list first)

if(head->seq == count){

coretempStr = head->coredata.to\_string();

for (int j =0; j<PAYLOADLENGTH; j++){

characterTemp = coretempStr.substr(PAYLOADLENGTH\*j, 8);

bitset <8> tempchar(characterTemp);

tempcharArray[j] = char(tempchar.to\_ulong());

}

//Calculating CRC16 on core data and checking against

//transmitters CRC16 value stored in the trailer

coretempStr = tempcharArray;

crc = (crc16(tempcharArray, 8));

bitset <16> sourceCRCBits(crc);

checksum = to\_string(crc);

checksumint = (int)sourceCRCBits.to\_ulong();

//if CRCs are the same send back ACK with Frame number

if (checksumint == head->sourceCRC){

printf("Success crcs are the same for frame %i \nSending ACK\n", count);

bzero(buffer, 256);

reply = "";

bitset<8> countBits(count);

reply = ACK + countBits.to\_string();

strcpy(buffer, reply.c\_str());

n = sendto(serverSocket, &buffer, sizeof(buffer), 0, (struct sockaddr\*) &clientAddr, sizeof(clientAddr));

if(n < 0)

{

printf("Error: Could not write to socket");

}

//delete node from list

dataFrame \*p = head;

head = head->next;

delete p;

QUE--;

printf("writing to file\n\n");

//Write data out to file

writeToFile(tempcharArray);

count++;

}

else{//crc's are not same so data is corrupted

printf("Data has been corrupted. Sending NAK\n");

bzero(buffer, 256);

bitset<8> countBits(count);

reply = NAK + countBits.to\_string();

strcpy(buffer, reply.c\_str());

n = sendto(serverSocket, &buffer, sizeof(buffer), 0, (struct sockaddr\*) &clientAddr, sizeof(clientAddr));

if(n < 0)

{

printf("Error: Could not write to socket");

}

//delete node from list

dataFrame \*p = head;

head = head->next;

delete p;

QUE--;

}

}

else{// Check rest of list

dataFrame\* temp = head;

//go to end of list

while (temp->next && temp->seq != count) {

pTemp = temp;

temp = temp->next;

}

if(temp->seq == count){

coretempStr = temp->coredata.to\_string();

for (int j =0; j<PAYLOADLENGTH; j++){

characterTemp = coretempStr.substr(PAYLOADLENGTH\*j, 8);

bitset <8> tempchar(characterTemp);

tempcharArray[j] = char(tempchar.to\_ulong());

}

coretempStr = tempcharArray;

crc = (crc16(tempcharArray, 8));

bitset <16> sourceCRCBits(crc);

checksum = to\_string(crc);

checksumint = (int)sourceCRCBits.to\_ulong();

if (checksumint == temp->sourceCRC){

printf("Success crcs are the same for frame %i \nSending ACK\n", count);

//send back ack and correctly delete node from list

bzero(buffer, 256);

bitset<8> countBits(count);

reply = ACK + countBits.to\_string();

strcpy(buffer, reply.c\_str());

n = sendto(serverSocket, &buffer, sizeof(buffer), 0, (struct sockaddr\*) &clientAddr, sizeof(clientAddr));

if(n < 0)

{

printf("Error: Could not write to socket");

}

//delete node from list

dataFrame \*p = temp;

pTemp->next = temp->next;

delete p;

QUE--;

//Write out to file

printf("Writing to file\n\n");

writeToFile(tempcharArray);

count++;

}

else{//CRCs are not same so data is corrupted send NAK

printf("Data has been corrupted. Sending NAK\n");

bzero(buffer, 256);

bitset<8> countBits(count);

reply = NAK + countBits.to\_string();

strcpy(buffer, reply.c\_str());

n = sendto(serverSocket, &buffer, sizeof(buffer), 0, (struct sockaddr\*) &clientAddr, sizeof(clientAddr));

if(n < 0)

{

printf("Error: Could not write to socket");

}

//delete node from list

dataFrame \*p = temp;

pTemp->next = temp->next;

delete p;

QUE--;

}

}

}

}

printf("Frames all recived and printed out in order\n\n");

return 0;

}