NATIONAL INSTITUTE OF TECHNOLOGY SILCHAR

Cachar, Assam

B.Tech. Vth Sem

Subject Code: CS-311

Subject Name: Computer Network Laboratory

Submitted By:

Name : Subhojit Ghimire

Sch. Id. : 1912160

Branch : CSE – B

Q.3. Write a Socket Program to implement "Go-Back-N" Protocol using TCP.

AIM: TO IMPLEMENT "GO-BACK-N" PROTOCOL USING TCP IN C

THEORY: 1. GO-BACK-N PROTOCOL: It is a connection oriented transmission. It uses sliding window method for reliable and sequential delivery of data frames. This data link layer protocol provides for sending multiple frames before receiving the acknowledgement for the first frame. The Go-Back-N protocol works for both the sender and the receiver side to ensure reliable data transfer.

2. SLIDING WINDOW PROTOCOL: It is a feature-based data transmission protocols. Sliding window protocols are used where reliable in-order delivery of packets is required, such as in the Data Link layer (OSI model) as well as in Transmission Control Protocol (TCP). By placing limits on the number of packets that can be transmitted or received at any given time, a sliding window protocol allows an unlimited number of packets to be communicated using fixed-size sequence numbers.

3. The Sender: Better known as client, the sender has a sequence of frames to send. The sender starts by sending the first frame, initially consisting of the base and the next packet to send. While there are more packets to send and the value for the next

packet to send is smaller than the summation of base and window size, the sender sends the packet pointed by pointer of next packet to send variable, and also increments the pointer. Meanwhile, the base pointer is incremented after receiving acknowledgement packets from the receiver.

4. The Receiver: Better known as the server, the receiver only keeps track of the expected sequence number to receive next. There is no receive buffer; the out of order packets are simply discarded, and so are the corrupted packets. The receiver always sends the acknowledgement for the last in-order packet received upon reception of a new packet (whether successfully or unsuccessfully).

The entire operation can be broken down into following parts:

1. Connection Setup: First, the connection is set up with a 3-way handshake between the sender and the receiver.

2. Data Transmission: Once the connection is established, the data is sent in DATA packets by the sender. Each DATA packet is acknowledged by the receiver by sending an ACK packet.

3. Connection Teardown: Once the data transmission is completed, the connection is torn down by both the sender and the receiver by sending FIN packets.

CODE:

// TCP GBN SENDER AND RECEIVER.

// RECEIVER:

```
# include <iostream>
# include <sys/types.h>
# include <sys/socket.h>
# include <netinet/in.h>
# include <arpa/inet.h>
# include <cstring>
# include <time.h>
# include <cstdlib>
# include <unistd.h>

# define windowSize 3
# define frameNumber 50
# define lossRate 10
# define PORT 8080

using namespace std;

char buffer [10];
char timeOutMsg [] = "Time Out!";

void errorExit (string errorSignal) {
        cout << "FAIL: " << errorSignal << endl;
        exit (0);
}
```

```
void gbnGenerate (int ack) {
    int ii = 0;
    int jj, kk = ack, ll;
    while (kk > 0) {
        ++ii;
        kk = kk/10;
    }
    ll = ii--;
    while (ack > 0) {
        kk = ack % 10;
        buffer [ii--] = kk + 48;
        ack = ack / 10;
    }
    buffer[ll] = '\0';
}

int main () {
    int sockfd = socket (AF_INET, SOCK_STREAM, 0);
    struct sockaddr_in gbnServAddr;
    if (sockfd != -1)
        cout << " SUCCESS: SOCKET CREATED \n";
    else
        errorExit (" SOCKET CREATING ");
    bzero (&gbnServAddr, sizeof (gbnServAddr));

    gbnServAddr.sin_family = AF_INET;
    gbnServAddr.sin_addr.s_addr = htonl (INADDR_ANY);
    gbnServAddr.sin_port = htons (PORT);
```

```
if (!(bind (sockfd, (struct sockaddr *) &gbnServAddr,
                    sizeof (gbnServAddr))))
        cout << "SUCCESS : SOCKET BINDED \n";
else
        errorExit ("SOCKET BINDING");

Struct sockaddr_in gbnClntAddr;
unsigned int cliAddrLen;
if (!(listen (sockfd, 5)))
        cout << " SUCCESS: SERVER LISTENING \n";
else
        errorExit ("SERVER LISTENING");
 cliAddrLen = sizeof (gbnClntAddr);

int connfd = accept (sockfd, (struct sockaddr *) &gbnClntAddr,
                                &cliAddrLen);
if (connfd >= 0)
        cout << "SUCCESS: CLIENT ACCEPTED \n\n";
else
        errorExit ("SERVER ACCEPTING");

unsigned int timeOut = (unsigned int) time (NULL);
srand (timeOut);
recv (connfd, buffer, sizeof (buffer), 0);

int buffValue = atoi (buffer);
int ii, frameDrop;
int ack = 1;
```

```
while (1) {
        for (ii = 0; ii < windowSize; ++ii) {
            recv (connfd, buffer, sizeof (buffer), 0);
            if (strncmp (buffer, timeOutMsg, 10) == 0)
                break;
        }
        ii = 0;
        while (ii < windowSize) {
            frameDrop = rand() % frameNumber;
            if (frameDrop < lossRate) {
                send (connfd, timeOutMsg, sizeof (
                                    timeOutMsg), 0);
                break;
            }
            else {
                gbnGenerate (ack);
                if (ack <= buffValue + 1) {
                    cout << "\nFRAME RECEIVED: ";
                    cout << buffer;
                    send (connfd, buffer, sizeof(buffer),0);
                }
                else break;
                ++ack;
            }
            if (ack > buffValue) break;
            ++ii;
        }
}
cout << "SUCCESS: SERVER EXIT\n"; close (sockfd); return 0;
```

// SENDER:

```
# include <iostream>
# include <sys/types.h>
# include <sys/socket.h>
# include <netinet/in.h>
# include <arpa/inet.h>
# include <cstring>
# include <time.h>
# include <cstdlib>
# include <unistd.h>

# define PORT 8080
# define windowSize 3

using namespace std;

char buffer [10];
char timeOutMsg [] = "Time Out!";

void errorExit (string errorSignal) {
        cout << "FAIL: " << errorSignal << endl;
        exit (0);
}
```

```
void gbnGenerate (int ack) {
    int ii = 0, jj, kk = ack, ll;
    while (kk > 0) {
        ++ii;
        kk = kk / 10;
    }
    ll = ii--;
    while (ack > 0) {
        kk = ack % 10;
        buffer [ii--] = kk + 48;
        ack = ack / 10;
    }
    buffer [ll] = '\0';
}

int main () {
    int sockfd = socket (AF_INET, SOCK_STREAM, 0);
    struct sockaddr_in gbnServAddr;
    if (sockfd != -1)
        printf ("SUCCESS: SOCKET CREATED\n");
    else
        errorExit ("SOCKET CREATION");
    bzero (&gbnServAddr, sizeof (gbnServAddr));

    gbnServAddr.sin_family = AF_INET;
    gbnServAddr.sin_addr.s_addr = htonl (INADDR_ANY)
                                  // htonl (INADDR_ANY);
    gbnServAddr.sin_port = htons (PORT);
```

```
if (!(connect (sockfd, (struct sockaddr*) &gbnServAddr,
                                    sizeof (gbnServAddr))))
        printf ("SUCCESS: CONNECTED TO SERVER \n\n");
else
        errorExit ("SERVER CONNECTION");

int frameNumber;
cout << endl;
cout << "Enter the number of frames:  ";
cin >> frameNumber;
cout << endl;
gbnGenerate (frameNumber);
send (sockfd, buffer, sizeof (buffer), 0);

int ii, windowTemp, ack = 1, bufferValue = 0, gbN = 0;
while (1) {
    for (ii = 0; ii < windowSize; ++ii) {
        gbnGenerate (ack);
        send (sockfd, buffer, sizeof (buffer), 0);
        if (ack <= frameNumber) {
            cout << "FRAME SENT  " << ack << endl;
            ++ack;
        }
    }
    sleep (3);
    ii = 0;
    windowTemp = windowSize;
```

```
while (ii < windowSize) {
    recv (sockfd, buffer, sizeof (buffer), 0);
    bufferValue = atoi (buffer);
    if (strcmp (buffer, timeOutMsg) == 0) {
        gbN = ack - windowTemp;
        if ( gbN < frameNumber) {
            cout << "\n\n TIME OUT. RESENDING";
            cout << " FRAMES " << gbN << " ONWARDS";
            cout << endl;
        }
        sleep (3);
        break;
    }
    else {
        if (bufferValue <= frameNumber) {
            cout << "FRAME ACKNOWLEDGED. FRAME";
            cout << " SEQUENCE :  " << buffer << endl;
            --windowTemp;
        }
        else break;
    }
    if (bufferValue > frameNumber) break;
    ++ii;
}
if (windowTemp == 0 && ack > frameNumber) {
    send (sockfd, timeOutMsg, sizeof (timeOutMsg), 0);
    break;
}
```

```
        else {
            ack = ack - windowTemp;
            windowTemp = windowSize;
        }
        if (atoi(buffer) == frameNumber)
            break;
    }
    cout << " SUCCESS: CLIENT EXIT " << endl;
    close (sockfd);
    return 0;
}
```

// OUTPUT EXPLANATION:

The socket port in this Go-Back-N Protocol Implementation was pre-defined as 8080. The loss rate (ie, the chance to drop frame) was pre-taken as 10%. The total number of frames to be accepted was limited to 50, that means the receiver cannot accept more than 50 data frames from the sender.

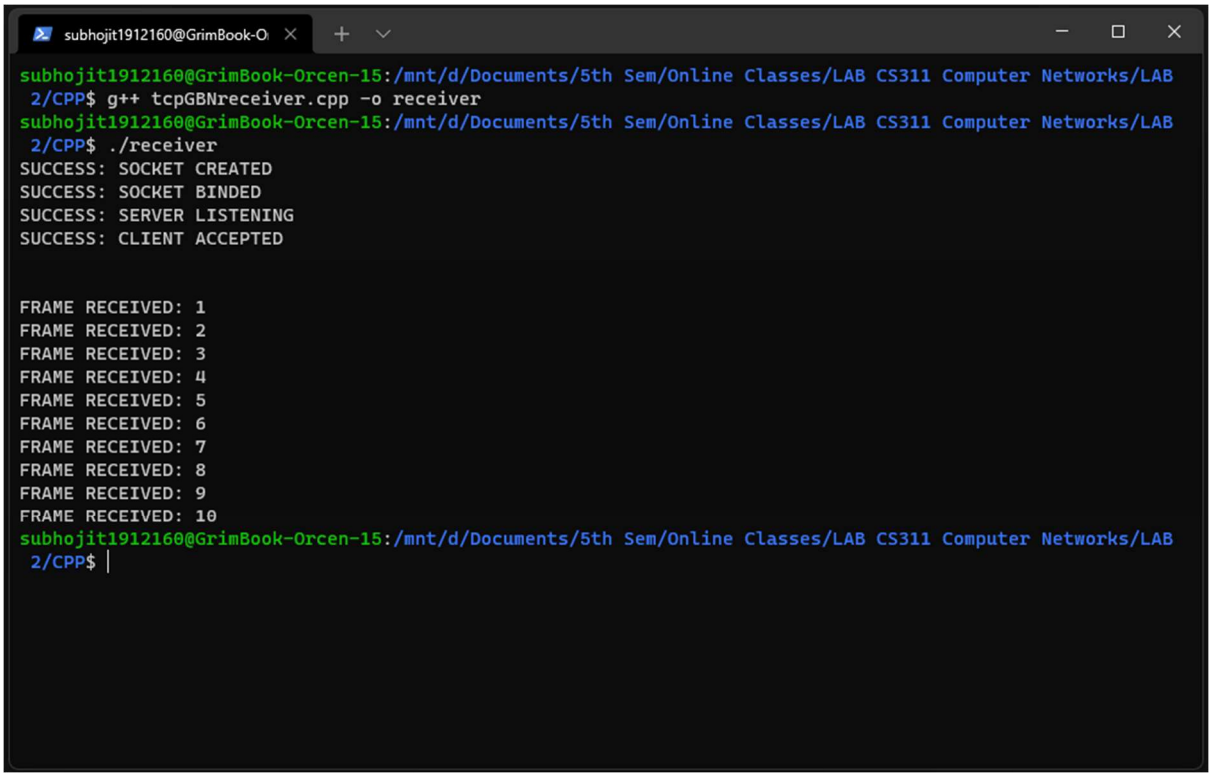
Firstly, the receiver programme is compiled and run. The we compile and run the sender programme. Once the connection is established between the sender and the receiver, we give in the number of data frames we wish to work with as the input in the sender terminal. Then we enter the window size. Window size ensures how many data packets are sent at a time. Here we have taken window size as 3, so we first send 3 data frames back to back, without awaiting the receiver's acknowledgement. After sending, we wait for

the receiver to acknowledge the data packets. If the receiver successfully receives the data packet, the server terminal updates with the frame Number it has received, and the sender terminal also updates with the "frame acknowledged" message. If the receiver fails to receive data; in our case, receiver suffers data drop for frame 3; the next three frames starting from frame 3 will be resent. The data drop is completely random, that means, there is a possibility that all the data frames will be received with no error at all, and also a propability all the data frames will suffer drop and have to be resent all over again. The process will continue until the receiver receives all the data frames and the sender receives acknowledgement for all the sent data packets, ensuring successful transmission of data packets.

Finally, the receiver and the sender program will close the connection and terminate successfully, (in our case, I have forcefully terminated the terminal using Ctrl+C command due to unforeseen bug issues).

OUTPUT:

// TCP GO-BACK-N RECEIVER



// TCP Go-BACK-N SENDER