

COMPUTER NETWORKS

Laboratory Manual

Subject Code: CO-306



SUBMITTED TO:

Kavinder Singh

SUBMITTED BY:

Zishnendu Sarker

2K19/CO/450

CO-A6(G3)

Experiment No: 01

AIM: - To write a program that performs Bit-Stuffing

Theory: -

Bit Stuffing: The technique allows data frames to contain an arbitrary number of bits and allows character codes with an arbitrary no of bits per character. Each frame begins and ends with the special bit pattern, 01111110, called a flag byte. Whenever the sender's data link layer encounters five consecutive ones in the data, it automatically stuffs a 0 bit into the outgoing bit stream. This bit stuffing is analogous to character stuffing, in which a DLE is stuffed into the outgoing character stream before DLE in the data.

CODE:

```
#include<iostream>
#include<vector>
using namespace std;
int main()
{
    vector <char> data;
    int n;
    cout<<"Enter the no of bits to be transferred: "<<endl;
    cin>>n;
    cout<<"Enter the data to be transferred: "<<endl;
    char b;int i;
    for(i=0;i<n;i++)
    {
        cin>>b;
        data.push_back(b);
    }
    i=0;int j;
    while(i<data.size())
    {
        if(data[i]=='1')
        {
            j=1;
```

```
while(data[++i]=='1' && j<5)
{
j++;
}
if(j==5)
{
data.insert(data.begin()+i,'0');
}
}
i++;
}
cout<<endl<<"Data after stuffing: "<<endl;
for(i=0;i<data.size();i++)
cout<<data[i];
cout<<endl;
return 0;
}
```

OUTPUT-:

```
Enter the no of bits to be transferred:
10
Enter the data to be transferred:
1 0 0 1 1 1 1 0 1

Data after stuffing:
10011111001

-----
Process exited after 21.31 seconds with return value 0
Press any key to continue . . .
```

LEARNING OUTCOME:

Bit Stuffing was studied, a C++ program for same was written and the results were verified by obtaining the output

EXPERIMENT NO: 2

AIM: - To write a program that performs Byte-Stuffing

Data Link Layer: The data link layer, or layer 2, is the second layer of the seven-layer OSI model of computer networking. This layer is the protocol layer that transfers data between nodes on a network segment across the physical layer. Data link layer is responsible for something called Framing, which is the division of stream of bits from network layer into manageable units (called frames). Frames could be of fixed size or variable size. In variable-size framing, we need a way to define the end of the frame and the beginning of the next frame.

Byte Stuffing: In Byte stuffing a process is followed which transforms a sequence of data bytes that may contain 'illegal' or 'reserved' values (such as packet delimiter) into a potentially longer sequence that contains no occurrences of those values. The extra length of the transformed sequence is typically referred to as the overhead of the algorithm. The COBS algorithm tightly bounds the worst-case overhead, limiting it to a minimum of one byte and a maximum of $\lceil n/254 \rceil$ bytes (one byte in 254, rounded up).

Byte Stuffing Mechanism

If the pattern of the flag byte is present in the message byte, there should be a strategy so that the receiver does not consider the pattern as the end of the frame. In character – oriented protocol, the mechanism adopted is byte stuffing.

In byte stuffing, a special byte called the escape character (ESC) is stuffed before every byte in the message with the same pattern as the flag byte. If the ESC sequence is found in the message byte, then another ESC byte is stuffed before it.

Note: Point to Point Protocol (PPP) is based on byte stuffing.

ALGORITHMS:

Byte Stuffing:

1. Start
2. Append 0 1 1 1 1 1 0 at the beginning of the string.
3. Check the data if character is present. If character is present in the string, insert another 0 1 1 1 1 1 0 in the string.
4. Transmit 0 1 1 1 1 1 1 0 at the end of the string.
5. Display the string.
6. Stop.

CODE:

```
#include<iostream.h>
using namespace std;
int main()
{
```

```

    int i,j,k,data[100],n;
    cout<<"Enter the no. of bits: "<<endl;
    cin>>n;
    cout<<"Enter the bits to be transmitted: "<<endl;
    for(i=0;i<n;i++)
    {
        cin>>data[i];
    }
    cout<<endl;
    cout<<<"The bits entered are: "<<endl;
    for(i=0;i<n;i++)
    {
        Cout<<data[i];
    }
    cout<<<"The byte stuffed array is: "<<endl;
    cout<<"01111110 ";
    for(i=0;i<n;i++)
    {
        if(data[i]==0 && data[i+1]==1 && data[i+2]==1 && data[i+3]==1 && data[i+4]==1
        && data[i+5] && data[i+6]==1 && data[i+7]==0)
        {
            cout<<"01111111001111110";
            i=i+7;
        }
        else cout<<data[i];
    }
    cout<<" 01111110"<<endl;
    cout<<endl;
    return 0;
}

```

OUTPUT:

```
Enter the no. of bits:
9
Enter the bits to be transmitted:
0
1
1
1
1
1
1
1
0
1

The bits entered are:
011111101
The byte stuffed array is:
01111110 01111110011111101 01111110
```

LEARNING OUTCOMES AND RESULTS:

We have learnt that in byte stuffing, start and end of frame are recognized with the help of flag bytes. Each frame starts with and ends with a flag byte. Two consecutive flag bytes indicate the end of one frame and start of the next one. The flag bytes used is named as “ESC” flag byte.

A frame delimited by flag bytes. This framing method is only applicable in 8-bit character codes which are a major disadvantage of this method as not all character codes use 8-bit characters e.g., Unicode.

EXPERIMENT NO: 3

AIM: -

To write a program that performs Cyclic Redundancy Check (CRC)

THEORY:-

CRC or Cyclic Redundancy Check is a method of detecting accidental changes/errors in the communication channel.

CRC uses Generator Polynomial which is available on both sender and receiver side. An example generator polynomial is of the form like $x^3 + x + 1$. This generator polynomial represents key 1011. Another example is $x^2 + 1$ that represents key 101.

- **n : Number of bits in data to be sent from sender side.**
- **k : Number of bits in the key obtained from generator polynomial.**

Sender Side (Generation of Encoded Data from Data and Generator Polynomial (or Key)):

1. The binary data is first augmented by adding $k-1$ zeros in the end of the data
2. Use *modulo-2 binary division* to divide binary data by the key and store remainder of division.
3. Append the remainder at the end of the data to form the encoded data and send the same

Receiver Side (Check if there are errors introduced in transmission)
Perform modulo-2 division again and if the remainder is 0, then there are no errors.

Modulo

2

Division:

The process of modulo-2 binary division is the same as the familiar division process we use for decimal numbers. Just that instead of subtraction, we use XOR here.

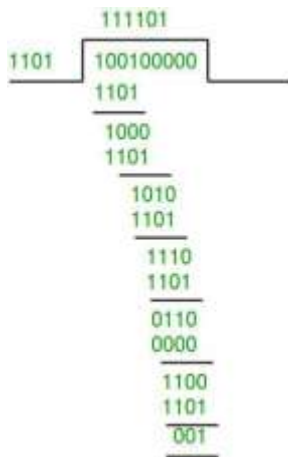
1. In each step, a copy of the divisor (or data) is XORed with the k bits of the dividend (or key).
2. The result of the XOR operation (remainder) is $(n-1)$ bits, which is used for the next step after 1 extra bit is pulled down to make it n bits long.
3. When there are no bits left to pull down, we have a result. The $(n-1)$ -bit remainder which is appended at the sender side.

Example:

Data word to be sent - 100100

Key - 1101 [Or generator polynomial $x^3 + x^2 + 1$]

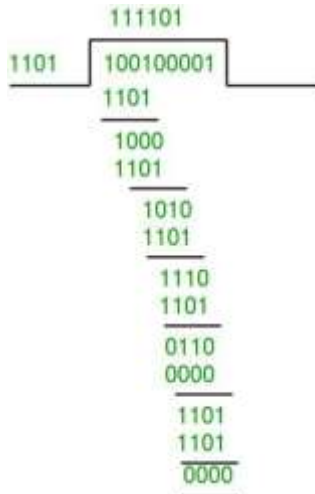
Sender Side:



Therefore, the remainder is 001 and hence the encoded data sent is 100100001.

Receiver Side:

Code word received at the receiver side 100100001



Therefore, the remainder is all zeros. Hence, the data received has no error.

CODE:

```

#include<iostream>
using namespace std;
void division(int temp[],int gen[],int nmb,int rgb){
for(int i=0;i<nmb;i++){
if (gen[0]==temp[i])
{
for(int j=0,k=i;j<rgb+1;j++,k++){
if(!(temp[k]^gen[j]))
temp[k]=0;
else
temp[k]=1;
}
}
}
}

```

```

int main(){
    int nmb,rgb;
    int msg[50],gen[50],temp[50];
    cout<<"At Sender's End "<<endl;
    cout<<"Enter the number of message bits : ";
    cin>>nmb;
    cout<<endl;
    cout<<"Enter the number of generator bits : ";
    cin>>rgb;
    cout<<endl;
    cout<<"Enter the message : ";
    for(int i=0;i<nmb;i++){
        cin>>msg[i];
    }
    cout<<endl;
    cout<<"Enter the generator : ";
    for(int i=0;i<rgb;i++){
        cin>>gen[i];
    }
    rgb--;
    for(int i=0;i<rgb;i++){
        msg[nmb+i] = 0;
    }
    for(int i=0;i<nmb+rgb;i++){
        temp[i] = msg[i];
    }
    division(temp,gen,nmb,rgb);
    cout<<endl;
    cout<<"Cyclic Redundancy Code : ";
    for(int i=0;i<rgb;i++){
        cout<<temp[nmb+i]<<" ";
        msg[nmb+i] = temp[nmb+i];
    }
    cout<<endl;
    cout<<endl<<"Transmitted Message : ";
    for(int i=0;i<nmb+rgb;i++){
        cout<<msg[i]<<" ";
    }
    cout<<endl<<endl;
    cout<<"At Receiver's End "<<endl;
    cout<<"Enter the received message : ";
    for(int i=0;i<nmb+rgb;i++){
        cin>>msg[i];
    }
    for(int i=0;i<nmb+rgb;i++){
        temp[i] = msg[i];
    }
    division(temp,gen,nmb,rgb);
    for(int i=0;i<rgb;i++){
        if(temp[nmb+i]){
            cout<<"\nError detected in received message.";
            return 0;
        }
    }
    cout<<"\nNo error in received Message"<<endl;;
}

```

```

cout<<"Received Message : ";
for(int i=0;i<nmb;i++){
cout<<msg[i]<<" ";
}
cout<<endl;
return 0;
}

```

Output :-

```

At Sender's End
Enter the number of message bits : 12

Enter the number of generator bits : 6

Enter the message : 1 1 0 1 1 0 0 0 0 0 1 1

Enter the generator : 1 0 0 0 1 1

Cyclic Redundancy Code : 0 1 0 1 1

Transmitted Message : 1 1 0 1 1 0 0 0 0 0 1 1 0 1 0 1 1

At Receiver's End
Enter the received message : 1 1 0 1 1 0 0 0 0 0 1 1 0 1 0 1 1

No error in received Message
Received Message : 1 1 0 1 1 0 0 0 0 0 1 1

-----
Process exited after 119.1 seconds with return value 0
Press any key to continue . . .

```

RESULT:-

A CRC-enabled device calculates a short, fixed-length binary sequence, known as the check value or CRC, for each block of data to be sent or stored and appends it to the data, forming a codeword. When a codeword is received or read, the device either compares its check value with one freshly calculated from the data block, or equivalently, performs a CRC on the whole codeword and compares the resulting check value with an expected residue constant. If the CRC values do not match, then the block contains a data error.

LEARNING OUTCOME:

Cyclic Redundancy Check was studied, a C++ program for same was written and the results were verified by obtaining the output.

EXPERIMENT NO: 4

AIM:

To write a program that performs Stop and Wait Protocol

THEORY:

Stop and Wait protocol is:

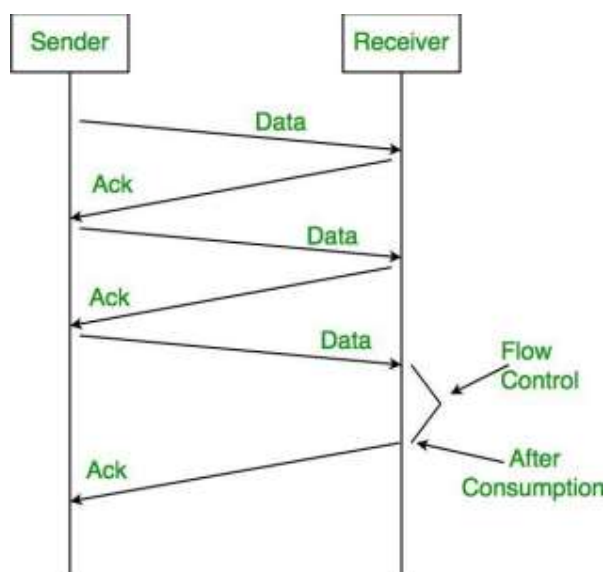
- Used in Connection-oriented communication. • It offers error and flow control.
- It is used in Data Link and Transport Layers.
- Stop and Wait ARQ mainly implements Sliding Window Protocol
- concept with Window Size 1.

Sender:

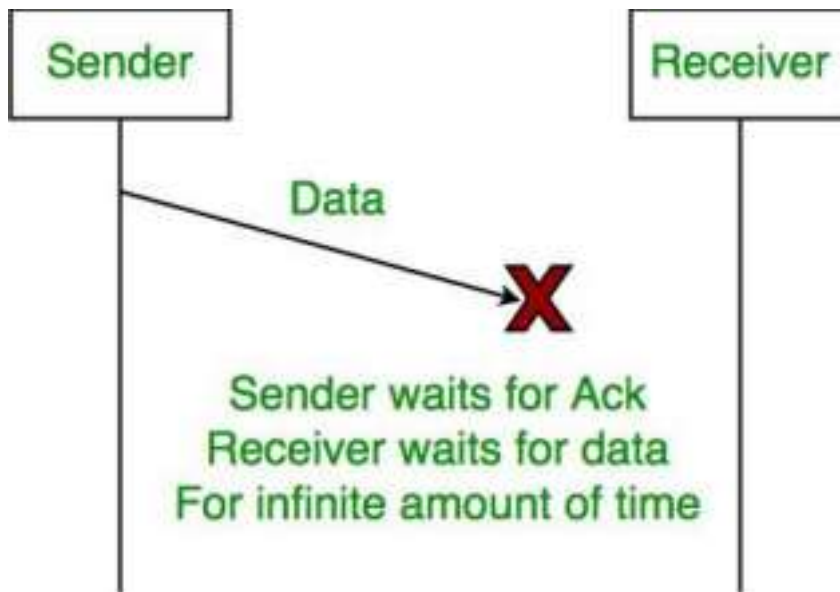
1. Send one data packet at a time.
2. Send next packet only after receiving acknowledgement for previous.

Receiver:

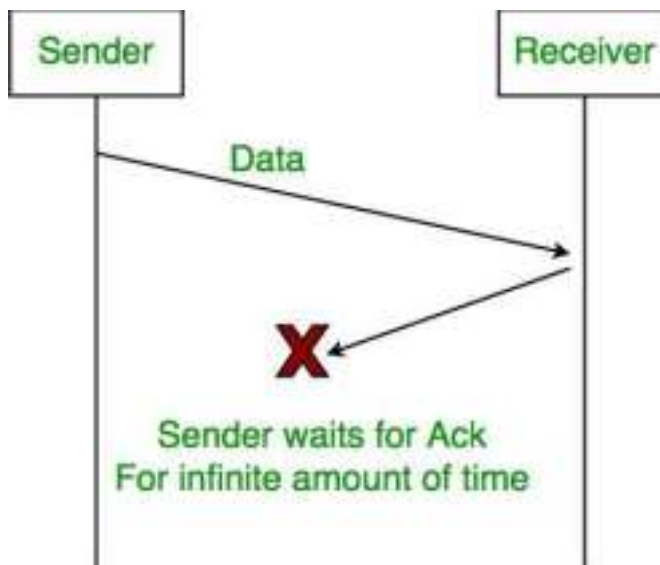
1. Send acknowledgement after receiving and consuming of data packet.
2. After consuming packet acknowledgement need to be sent (Flow Control)



1. Lost Data



2. Lost Acknowledgement:

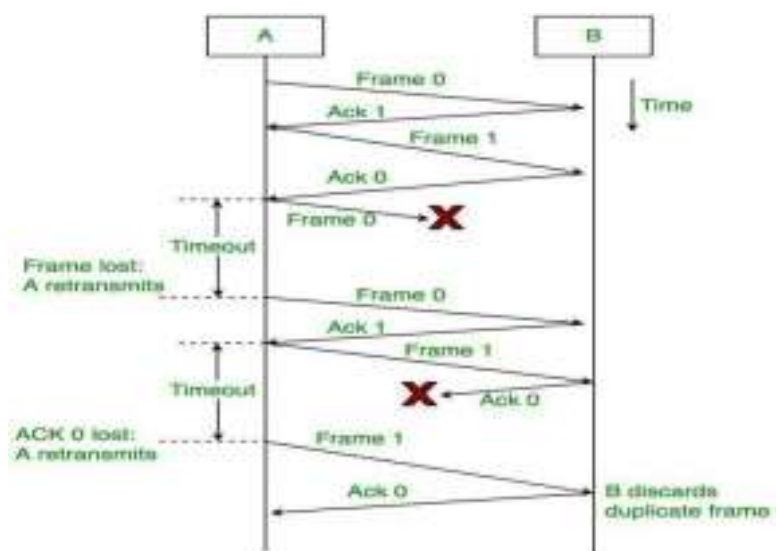


3. Delayed Acknowledgement/Data: After timeout on sender side, a long-delayed acknowledgement might be wrongly considered as acknowledgement of some other recent packet.

Stop and Wait ARQ (Automatic Repeat Request)

Above 3 problems are resolved by Stop and Wait ARQ (Automatic Repeat Request) that does both error control and flow control.

with sequence number 1 (sequence number of next expected data frame or packet) There is only one-bit sequence number that implies that both sender and receiver have buffer for one frame or packet only.



ALGORITHM:

At sender (Node A)

1. Accept packet from higher layer when available. 2. Assign number SN to it
3. Transmit packet SN in frame with sequence #SN 4. Wait for an error free frame from B
- a. If received and it contains RN greater than SN in the request #field, set SN 2 RN and go to 1
- b. If not received within given time, go to 2 (with initial condition SN equal to zero)

At receiver (Node B)

1. Whenever an error free frame is received from a with a sequence # equal to RN release the receive packet to higher layers and increment RN
2. At arbitrary times, but within bounded delay after receiving an error free frame from A, transmit a frame to A containing RN in the request # field. (with initial condition RN equal to zero)

CODE:

```
#include <conio.h>

#include <dos.h>

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <windows.h>
#define TIMEOUT 5

#define MAX_SEQ 1

#define TOT_PACKETS 8

#define inc(k) \

if (k < MAX_SEQ) \

k++; else

k = 0; typedef struct

{

int data;

} packet; typedef struct {

\\

int kind;

int seq;

int ack; packet info; int err;

} frame; frame DATA; typedef enum {

frame_arrival, err,

timeout, no_event

} event_type;

void from_network_layer(packet *);

void to_network_layer(packet *);

void to_physical_layer(frame *);

void from_physical_layer(frame *);

void wait_for_event_sender(event_type *);

void wait_for_event_reciever(event_type *); void reciever();

void sender();

int i = 1; //Data to be sent by sender char turn; //r , s int

DISCONNECT = 0;

void main() {

while (!DISCONNECT) {
```



```

sender(); Sleep(400); reciever();

}

getch(); }

void sender() {

static int fts = 0; static frame s; packet buf; event_type event; static int flag = 0; if (flag ==
0)

{
from_network_layer(&buf);

s.info = buf;
s.seq = fts;
printf("SENDER : Info = %d turn = 'r'; to_physical_layer(&s); flag = 1;

} wait_for_event_sender(&event); if (turn == 's')
{

if (event == frame_arrival) {

from_network_layer(&buf); inc(fts);
s.info = buf;
s.seq = fts;
printf("SENDER : Info = %d turn = 'r'; to_physical_layer(&s);

}
if (event == timeout) {

printf("SENDER : Resending Frame "); turn = 'r';
to_physical_layer(&s);

} }

}

void reciever() {

static int frame_expected = 0; frame r, s;
event_type event;

wait_for_event_reciever(&event); if (turn == 'r')
{

if (event == frame_arrival) {

from_physical_layer(&r);
if (r.seq == frame_expected) {

to_network_layer(&r.info);

inc(frame_expected); }

else

Seq No = %d\t", s.info, s.seq);

```

```

Seq No = %d\t", s.info, s.seq);

} }

printf("RECIEVER : Acknowledgement Resent\n"); turn = 's';
to_physical_layer(&s); }

if (event == err) {

}

printf("RECIEVER : Garbled Frame\n"); turn = 's'; //if frame not recieved

//sender shold send it again

void from_network_layer(packet *buf) {

(*buf).data = i;
i++; }

void to_physical_layer(frame *s) {

srand(time(NULL)); // 0 means error
s->err = rand() % 4; //non zero means no error DATA = *s; // probability of
error = 1/4

}

void to_network_layer(packet *buf) {

printf("RECIEVER :Packet %d recieved , Ack Sent\n", (*buf).data); if (i > TOT_PACKETS)
//if all packets recieved then disconnect {

DISCONNECT = 1;

printf("\nDISCONNECTED"); }

}

void from_physical_layer(frame *buf) {

*buf = DATA; }

void wait_for_event_sender(event_type *e) {

static int timer = 0;

if (turn == 's') {

timer++;
if (timer == TIMEOUT) {

*e = timeout;
printf("SENDER : Ack not recieved=> TIMEOUT\n"); timer = 0; return;

}
if (DATA.err == 0)

```

```

*e = err; else
{
timer = 0;

*e = frame_arrival; }

} }

void wait_for_event_reciever(event_type *e) {

if (turn == 'r') {

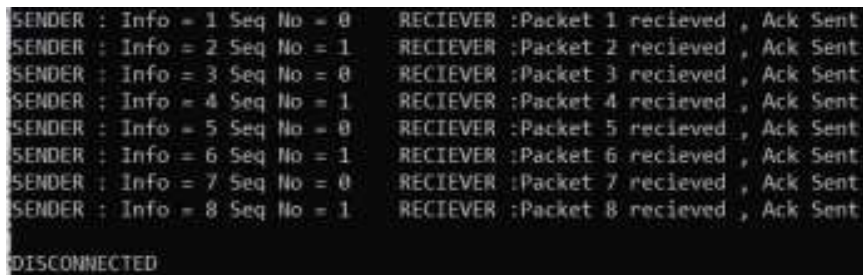
if (DATA.err == 0) *e = err;

else
*e = frame_arrival;

} }

```

OUTPUTS:



```

SENDER : Info = 1 Seq No = 0    RECIEVER :Packet 1 recieved , Ack Sent
SENDER : Info = 2 Seq No = 1    RECIEVER :Packet 2 recieved , Ack Sent
SENDER : Info = 3 Seq No = 0    RECIEVER :Packet 3 recieved , Ack Sent
SENDER : Info = 4 Seq No = 1    RECIEVER :Packet 4 recieved , Ack Sent
SENDER : Info = 5 Seq No = 0    RECIEVER :Packet 5 recieved , Ack Sent
SENDER : Info = 6 Seq No = 1    RECIEVER :Packet 6 recieved , Ack Sent
SENDER : Info = 7 Seq No = 0    RECIEVER :Packet 7 recieved , Ack Sent
SENDER : Info = 8 Seq No = 1    RECIEVER :Packet 8 recieved , Ack Sent

DISCONNECTED

```

DISCUSSION:

1. It uses link between sender and receiver as half duplex
2. Throughput = 1 Data packet/frame per RTT
3. If Bandwidth*Delay product is very high, then stop and wait protocol is not so useful. The sender has to keep waiting for acknowledgements before sending the processed next packet.
4. It is an example for “**Closed Loop OR connection oriented**” protocols
5. It is a special category of SWP where its window size is 1
6. Irrespective of number of packets sender is having stop and wait protocol requires only 2 sequence numbers 0 and 1

LEARNING OUTCOMES AND RESULTS:

The Stop and Wait ARQ solves main three problems but may cause big performance issues as sender always waits for acknowledgement even if it has next packet ready to send. Consider a situation where you have a high bandwidth connection and propagation delay is also high (you are connected to some server in some other country though a high speed connection). To solve this problem, we can send more than one packet at a time with a larger sequence number.

Thus, we implemented the Stop and Wait Protocol at both the sender's and the receiver's side.

EXPERIMENT NO: 5

AIM: -

To write a program that performs Sliding Window Protocol

THEORY:-

The sliding window is a technique for sending multiple frames at a time. It controls the data packets between the two devices where reliable and gradual delivery of data frames is needed. It is also used in TCP (Transmission Control Protocol).

In this technique, each frame has sent from the sequence number. The sequence numbers are used to find the missing data in the receiver end. The purpose of the sliding window technique is to avoid duplicate data, so it uses the sequence number.

Sliding window refers to an imaginary boxes that hold the frames on both sender and receiver side.

It provides the upper limit on the number of frames that can be transmitted before requiring an acknowledgment.

Frames may be acknowledged by receiver at any point even when window is not full on receiver side.

Frames may be transmitted by source even when window is not yet full on sender side.

The windows have a specific size in which the frames are numbered modulo- n , which means they are numbered from 0 to $n-1$. For e.g. if $n = 8$, the frames are numbered 0, 1, 2, 3, 4, 5, 6, 7, 0, 1, 2, 3, 4, 5, 6, 7, 0, 1,

The size of window is $n-1$. For e.g. In this case it is 7. Therefore, a maximum of $n-1$ frames may be sent before an acknowledgment.

When the receiver sends an ACK, it includes the number of next frame it expects to receive. For example in order to acknowledge the group of frames ending in frame 4, the receiver sends an ACK containing the number 5. When sender sees an ACK with number 5, it comes to know that all the frames up to number 4 have been received.



CODE:

```
#include <iostream>
using namespace std;
int main(){
```

```

int ws,fs;

cout<<"Enter window size: ";
cin>>ws;

cout<<"Enter Frame Size: ";
cin>>fs;

int frames[fs];

for(int i=1;i<=fs;i++){
    frames[i]=i;
}

int idx=1;

for(int i=1;i<=fs;i++){

    cout<<"\nFrames transmitted this second ";
    for(int j=i;j<i+ws&& j<=fs;j++){

        cout<<" "<<frames[j];

    }

    cout<<"\n Acknowledgement of frame "<<frames[idx++]<<" received";
}

return 0;
}

```

OUTPUT:

```

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\ayush> cd "D:\StudyMaterial\3rdYear\6th SEM\Colab" ; if ($?) { g++ slidingwindow.cpp -o slidingwindow } ; if ($?) { .\slidingwindow }
Enter window size: 3
Enter Frame Size: 12

Frames transmitted this second 1 2 3
Acknowledgement of frame 1 received
Frames transmitted this second 2 3 4
Acknowledgement of frame 2 received
Frames transmitted this second 3 4 5
Acknowledgement of frame 3 received
Frames transmitted this second 4 5 6
Acknowledgement of frame 4 received
Frames transmitted this second 5 6 7
Acknowledgement of frame 5 received
Frames transmitted this second 6 7 8
Acknowledgement of frame 6 received
Frames transmitted this second 7 8 9
Acknowledgement of frame 7 received
Frames transmitted this second 8 9 10
Acknowledgement of frame 8 received
Frames transmitted this second 9 10 11
Acknowledgement of frame 9 received
Frames transmitted this second 10 11 12
Acknowledgement of frame 10 received
Frames transmitted this second 11 12
Acknowledgement of frame 11 received
Frames transmitted this second 12
Acknowledgement of frame 12 received
PS D:\StudyMaterial\3rdYear\6th SEM\Colab>

```

LEARNING OUTCOME:

Sliding Window Protocol was studied, a C++ program for same was written and the results were verified by obtaining the output.

EXPERIMENT NO: 6

AIM: To implement and find class, network and host ID of given IPv4 address.

THEORY:

IP address is an address having information about how to reach a specific host, especially outside the LAN. An IP address is a 32-bit unique address having an address space of 2^{32} .

Generally, there are two notations in which IP address is written, dotted decimal notation and hexadecimal notation.

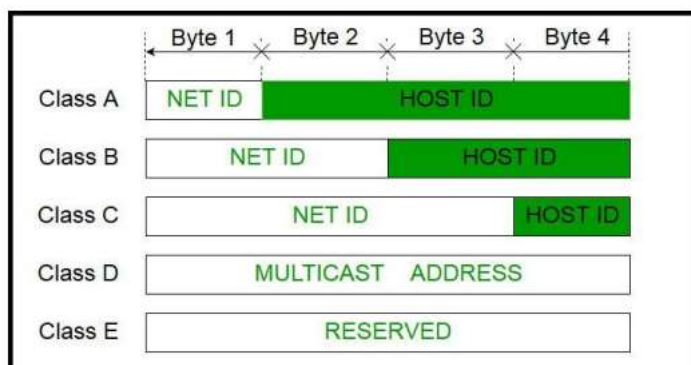
The 32-bit IP address is divided into five sub-classes. These are:

- Class A
- Class B
- Class C
- Class D
- Class E

Each of these classes has a valid range of IP addresses. Classes D and E are reserved for multicast and experimental purposes respectively. The order of bits in the first octet determine the classes of IP address. IPv4 address is divided into two parts:

- Network ID
- Host ID

The class of IP address is used to determine the bits used for network ID and host ID and the number of total networks and hosts possible in that particular class. Each ISP or network administrator assigns IP address to each device that is connected to its network.



ALGORITHM:

1. **For determining the class:** The idea is to check first octet of IP address. As we know, for class A first octet will range from 1 – 126, for class B first octet will range from 128 – 191, for class C first octet will range from 192- 223, for class D first octet will range from 224 – 239, for class E first octet will range from 240 – 255.
2. **For determining the Network and Host ID:** We know that Subnet Mask for Class A is 8, for Class B is 16 and for Class C is 24 whereas Class D and E is not

divided into Network and Host ID. For 2nd Example, first octet is 130. So, it belongs to Class B. Class B has subnet mask of 16. So, first 16 bit or first two octets is Network ID part and rest is Host ID part. Hence, Network ID is 130.45 and Host ID is 151.154

CODE:

```
#include<bits/stdc++.h>

using namespace std;

char findClass(vector<char> str)
{

vector<char> arr(4,0); int i = 0;
while (str[i] != '.')
{
arr[i] = str[i]; i++;
}
i--;
{
ip = ip + (str[i] - '0') * j; j = j * 10;
i--;
}

if (ip >= 1 && ip <= 127) return 'A';

else if (ip >= 128 && ip <= 191) return 'B';

else if (ip >= 192 && ip <= 223) return 'C';

else if (ip >= 224 && ip <= 239) return 'D';

else
return 'E';
}

// Finding Network ID
```



```

void networkID(vector<char> str, char ipClass)
{
    vector<char> net(12);

    if (ipClass == 'A')
    {

        int i = 0, j = 0; while (str[j] != '.')
        net[i++] = str[j++]; cout<<"Net Id is: "; for(auto it: net)
        cout<<it;
    }

    else if (ipClass == 'B')
    {
        int i = 0, j = 0, dot = 0;

        while (dot < 2)
        {
            net[i++] = str[j++]; if (str[j] == '.')
            dot++;
        }
        cout<<"Net Id is: "; for(auto it: net)
        cout<<it;
    }

    else if (ipClass == 'C')
    {
        int i = 0, j = 0, dot = 0;

        while (dot < 3)
        {
            net[i++] = str[j++]; if (str[j] == '.')
            dot++;
        }
        cout<<"Net Id is: "; for(auto it: net)

```

```

    cout<<it;
}

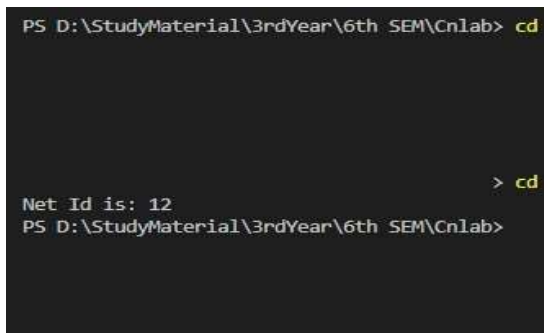
else
    cout<<"IP address of class E and D not availabe\n";
}

int main()
{
    string s="12.226.12.11"; vector<char> str;
    for(int i=0;i<s.size();i++){ str.push_back(s[i]);
    }
    char ipClass = findClass(str);

    networkID(str, ipClass); return 0;
}

```

OUTPUT:



```

PS D:\StudyMaterial\3rdYear\6th SEM\Cnlab> cd
> cd
Net Id is: 12
PS D:\StudyMaterial\3rdYear\6th SEM\Cnlab>

```

LEARNING OUTCOMES AND RESULTS:

We learnt about host id, net id and network class and their uses and significance in computer network.

