DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CS 208 : COMPUTER NETWORKS

LAB FILE

SUBMITTED TO
MS. GULL KAUR

SUBMITTED BY
KUNAL BANSAL
(23/CS/227)

INDEX

Experiment-1

Aim : To implement various framing techniques used in data communication to delineate frames.

Theory :

Bit Stuffing : Bit stuffing is a method used in data communication to avoid confusion between data and special control signals (like start or end markers). When a specific sequence of bits appears in the data an extra bit is added to break the pattern. This prevents the receiver from mistaking the data for control information. Once the data is received the extra bit is removed restoring the original message. This technique helps ensure accurate data transmission.

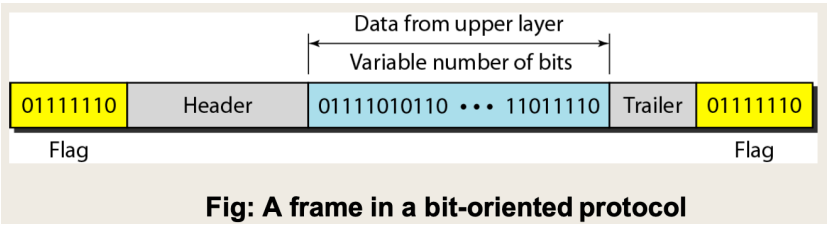


Fig: A frame in a bit-oriented protocol

Byte Stuffing : Byte stuffing is the same as bit stuffing the only difference is that instead of a single bit, one byte of data is added to the message to avoid confusion between data and special control signals. This ensures accurate message transmission without misinterpreting the data.

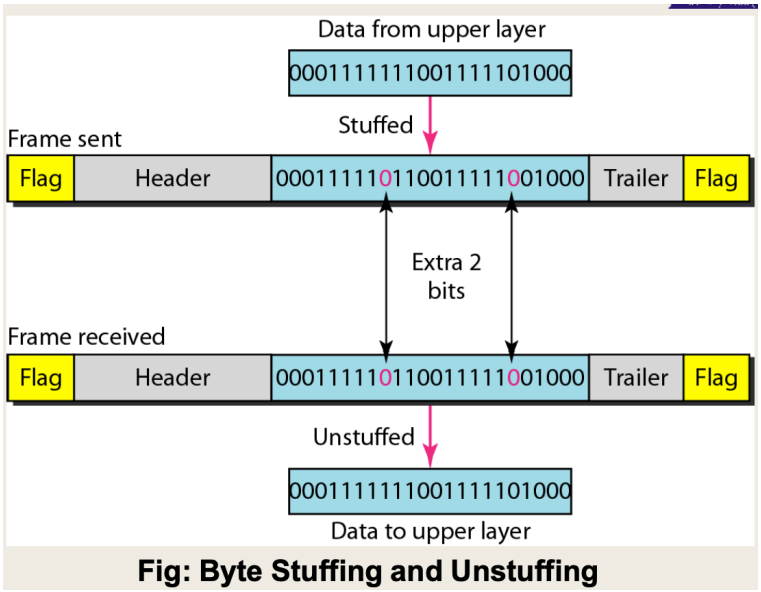


Fig: Byte Stuffing and Unstuffing

Character Stuffing : Character stuffing is a method used in data communication to ensure that special characters, which are used as control signals (such as start or end of a frame), are not misinterpreted as part of the data being transmitted.

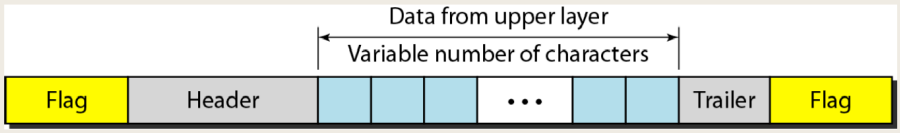


Fig: A frame in a character-oriented protocol

Code :

```
#include <iostream>
using namespace std;

int main() {
    // Case 1: Bit Framing
    string s;
    cout << "Input for Bit Stuffing: ";
    cin >> s;

    string head = "101";
    string tail = "101";
    string decoded = head;
    int counter = 0;

    for (int i = 0; i < s.size(); i++) {
        if (s[i] == '1') {
            decoded += '1';
            counter++;
            if (counter == 5) {
                decoded += '0';
                counter = 0;
            }
        } else {
            decoded += '0';
            counter = 0;
        }
    }

    decoded += tail;
    cout << "Decoded (Bit Stuffing): " << decoded << endl;

    // Case 2: Byte Framing
    string s2;
    cout << "Input for Byte Stuffing: ";
    cin >> s2;

    int byte_window = 4;
    string flag = "1111";
    string escape = "0000";
    string decoder2 = flag;

    for (int i = 0; i < s2.size(); i += byte_window) {
```

```
            string temp = s2.substr(i, byte_window);
            if (temp == flag) {
                decoder2 += escape;
                decoder2 += flag;
            } else if (temp == escape) {
                decoder2 += escape;
                decoder2 += escape;
            } else {
                decoder2 += temp;
            }
        }

        decoder2 += flag;
        cout << "Decoded (Byte Stuffing): " << decoder2 << endl;

        // Case 3: Character Framing
        string s3;
        cout << "Input for Character Stuffing: ";
        cin >> s3;

        char start_char = '#';
        char end_char = '#';
        string decoder3 = string(1, start_char);

        for (int i = 0; i < s3.size(); i++) {
            if (s3[i] == start_char || s3[i] == end_char) {
                decoder3 += start_char;
            }
            decoder3 += s3[i];
        }

        decoder3 += end_char;
        cout << "Decoded (Character Stuffing): " << decoder3 << endl;

        return 0;
    }
```

Output :

```
Input for Bit Stuffing: 111111110001111
Decoded (Bit Stuffing): 10111111011100011111101
Input for Byte Stuffing: 111100000111
Decoded (Byte Stuffing): 11110000111100000000001111111
Input for Character Stuffing: #hello#world#
Decoded (Character Stuffing): ###hello##world###
kunalbansal@KUNALs-MacBook-Pro CN LAB %
```

Experiment-2

Aim : To implement CRC generation and error detection. Display the codeword released by the sender. For error detection, send the original codeword and the corrupted codeword to show "Data Word" if the original is correct and "Invalid" if the codeword is corrupted in data communication.

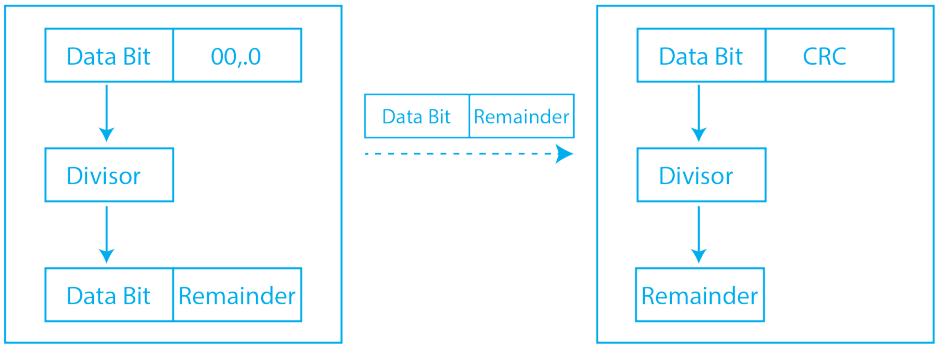
Theory :

Cyclic Redundancy Check (CRC) is an error-detection technique used in data communication to ensure data integrity. It involves appending a CRC code (remainder) to the original dataword before transmission. The CRC code is generated by performing modulo-2 division of the dataword (padded with zeros) by a predefined generator polynomial.

At the receiver's end, the received codeword undergoes the same division process; if the remainder is zero, the data is deemed error-free, otherwise, an error is detected.

Need for CRC in Data Communication

- Detects errors caused by noise or data corruption during transmission.
- Ensures reliable communication over unreliable channels.
- Efficient and widely used in network protocols like Ethernet, HDLC, & Wi-Fi.



CRC Working Illustration

Code :

```
#include <iostream>
using namespace std;

string mod2remainder(string s1, string s2)
{
    string result = "";
    for (int i = 1; i < s2.length(); i++)
    {
        if (s1[i] == s2[i])
            result += '0';
        else
            result += '1';
    }

    return result;
}

string mod2division(string dataword, string key)
{
    int ptr = key.length();
    string temp = dataword.substr(0, key.length());

    while (ptr < dataword.length())
    {
        string remainder = "";
        if (temp[0] == '1')
        {
            remainder = mod2remainder(temp, key);
        }
        else
        {
            remainder = mod2remainder(temp, string(key.length(),
'0'));
        }
        temp = remainder + dataword[ptr];
        ptr++;
    }

    if (temp[0] == '1')
    {
        return mod2remainder(temp, key);
    }
    else
    {
        return mod2remainder(temp, string(key.length(), '0'));
    }
}

string encodeData(string dataword, string key)
```

```
{
    dataword.append(key.length() - 1, '0');
    string remainder = mod2division(dataword, key);
    int i = (int)(remainder.length()) - 1, j = (int)
(dataword.length()) - 1;

    while (i >= 0 && j >= 0)
    {
        if (remainder[i] == dataword[j])
        {
            dataword[j] = '0';
        }
        else
        {
            dataword[j] = '1';
        }
        i--;
        j--;
    }

    return dataword;
}

bool decodeData(string encoded, string key)
{
    string remainder = mod2division(encoded, key);
    for (int i = 0; i < remainder.length(); i++)
    {
        if (remainder[i] == '1')
            return false;
    }
    return true;
}

int main(int argc, char const *argv[])
{
    cout << "Sender side..." << endl;

    string dataword;
    cout << "Enter Dataword : ";
    cin >> dataword;
    string key;
    cout << "Enter Key : ";
    cin >> key;

    cout << "The encoded string is : ";
    string encoded = encodeData(dataword, key);
    cout << encoded << endl;

    cout << "Receiver side..." << endl;

    bool decoding = decodeData(encoded, key);
```

```
    if (decoding)
    {
        cout << "Dataword" << endl;
        cout << "Data received successfully" << endl;
    }
    else
    {
        cout << "Invalid" << endl;
        cout << "Error in data" << endl;
    }

    cout << "Taking encoded data from user..." << endl;

    string encodedUser;
    cout << "Enter encoded string : ";
    cin >> encodedUser;
    string key2;
    cout << "Enter key : ";
    cin >> key2;

    decoding = decodeData(encodedUser, key2);
    if (decoding)
    {
        cout << "Dataword" << endl;
        cout << "Data received successfully" << endl;
    }
    else
    {
        cout << "Invalid" << endl;
        cout << "Error in data" << endl;
    }
    return 0;}
```

Output :

```
Sender side...
Enter Dataword : 1010
Enter Key : 1010
The encoded string is : 1010000
Reciever side...
Dataword
Data recieved successfully
Taking encoded data from user...
Enter encoded string : 101
Enter key : 10
Invalid
Error in data
kunalbansal@KUNALs-MacBook-Pro CN LAB %
```

Experiment-3

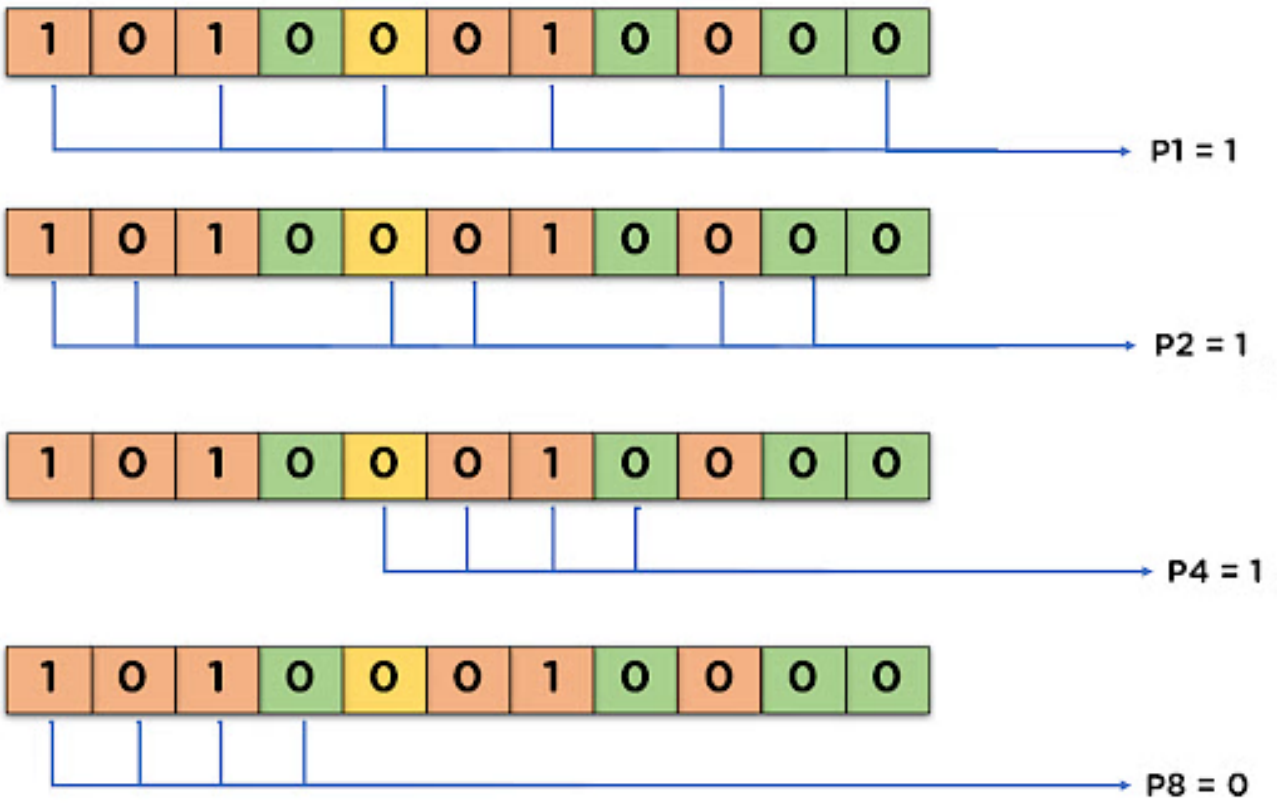
Aim : To implement error detection and correction using the Hamming Code. Show parity bits check bits and correct single-bit errors.

Theory :

Hamming code is an error-detection and correction technique used in data communication to detect and correct single-bit errors. It was developed by Richard Hamming and is widely used in computer memory systems and communication protocols.

Need for Hamming Code

- Detects and corrects single-bit errors without retransmission.
- Provides reliable data transmission over noisy channels.
- Uses redundant parity bits to encode the data, ensuring error correction.



Code :

```
#include <iostream>
#include <vector>
using namespace std;

bool detector(string encoded)
{
    // Calculating parity bits
    vector<int> parity(3);
    parity[0] = (encoded[0] - '0') ^ (encoded[2] - '0') ^
(encoded[4] - '0') ^ (encoded[6] - '0');
    parity[1] = (encoded[1] - '0') ^ (encoded[2] - '0') ^
(encoded[5] - '0') ^ (encoded[6] - '0');
    parity[2] = (encoded[3] - '0') ^ (encoded[4] - '0') ^
(encoded[5] - '0') ^ (encoded[6] - '0');

    int bit = parity[0] * 1 + parity[1] * 2 + parity[2] * 4;
    return bit;
}

vector<int> encoder(string input)
{
    // Using even parity Hamming(7,4)
    int n = 7;
    vector<int> encoded(n, 0);
    int it = 0;

    // Assigning data bits
    for (int i = 0; i < n; i++)
    {
        if (i != 0 && i != 1 && i != 3)
        {
            encoded[i] = input[it] - '0';
            it++;
        }
    }

    encoded[0] = encoded[2] ^ encoded[4] ^ encoded[6];
    encoded[1] = encoded[2] ^ encoded[5] ^ encoded[6];
    encoded[3] = encoded[4] ^ encoded[5] ^ encoded[6];

    return encoded;
}

int main()
{
    cout << "Enter a message (4 bits) : ";
    string input;
    cin >> input;

    vector<int> encodedMessage = encoder(input);
```

```
    cout << "Message (with Parity bits): ";
    for (int i = 0; i < encodedMessage.size(); i++)
        cout << encodedMessage[i];
    cout << endl;

    cout << "Enter received message (7 bits): ";
    string output;
    cin >> output;

    int errorBit = detector(output);
    if (errorBit == 0)
    {
        cout << "Message received correctly." << endl;
    }
    else
    {
        cout << "Message received incorrectly. Error detected at
bit position: " << errorBit << endl;
    }

    return 0;
}
```

Output :

```
Enter a message (4 bits) : 1010
Message (with Parity bits): 1011010
Enter received message (7 bits): 1011010
Message received correctly.
kunalbansal@KUNALs-MacBook-Pro CN LAB %
```

```
Enter a message (4 bits) : 1100
Message (with Parity bits): 0111100
Enter received message (7 bits): 011101
Message received incorrectly. Error detected at bit position: 1
kunalbansal@KUNALs-MacBook-Pro CN LAB %
```

Experiment-4

Aim : To implement the stop-and-wait protocol for unreliable data transmission. Handle three cases: when the frame and acknowledgement are delivered successfully, when the frame is lost, and when the acknowledgement is lost. Show the role of the timeout timer in the protocol.

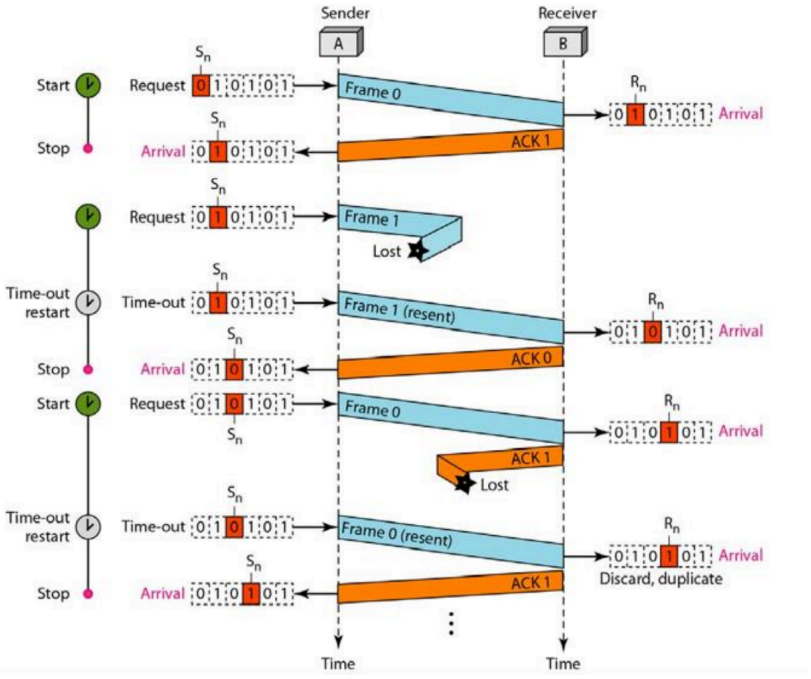
Theory :

The Stop-and-Wait protocol is a simple, reliable data-link layer protocol used in data communication. It ensures that frames (data packets) are sent and acknowledged correctly before transmitting the next frame. The protocol is widely used in systems where simplicity is preferred over efficiency, such as early ARQ (Automatic Repeat reQuest) mechanisms.

Working of Stop-and-Wait Protocol

1. The **sender** transmits a data frame and waits for an **acknowledgment (ACK)** from the receiver.
2. If the receiver successfully receives the frame, it sends an **ACK** back to the sender.
3. If the **ACK is received**, the sender transmits the next frame.
4. If no ACK is received within a certain timeout period, the sender **retransmits the frame**, assuming it was lost.

This ensures **reliable** communication by confirming the successful delivery of each frame before proceeding to the next one.



Code :

```
#include <iostream>
#include <cstdlib>
#include <ctime>
#include <thread>
#include <chrono>

using namespace std;

bool sendPacket(int packet)
{

    if (rand() % 10 < 3)
    {
        cout << "Packet " << packet << " lost! Resending...\n";
        return false;
    }
    cout << "Packet " << packet << " sent successfully!\n";
    return true;
}

bool receiveAck(int packet)
{

    if (rand() % 10 < 2)
    {
        cout << "ACK for Packet " << packet << " lost! Resending
packet...\n";
        return false;
    }
    cout << "ACK received for Packet " << packet << "!\n";
    return true;
}

int main()
{
    srand(time(0));

    int totalPackets = 5;
    int packet = 1;

    while (packet <= totalPackets)
    {
        bool sent = false, acked = false;

        while (!sent || !acked)
        {
            sent = sendPacket(packet);
            if (sent)
            {
                this_thread::sleep_for(chrono::milliseconds(500));
```

```
                acked = receiveAck(packet);
            }
            this_thread::sleep_for(chrono::milliseconds(500));
        }

        cout << "Packet " << packet << " successfully delivered.
\n\n";
        packet++;
    }

    cout << "All packets successfully transmitted!\n";
    return 0;
}
```

Output :

```
Packet 1 lost! Resending...
Packet 1 lost! Resending...
Packet 1 lost! Resending...
Packet 1 lost! Resending...
Packet 1 sent successfully!
ACK received for Packet 1!
Packet 1 successfully delivered.

Packet 2 lost! Resending...
Packet 2 sent successfully!
ACK received for Packet 2!
Packet 2 successfully delivered.

Packet 3 lost! Resending...
Packet 3 sent successfully!
ACK received for Packet 3!
Packet 3 successfully delivered.

Packet 4 sent successfully!
ACK for Packet 4 lost! Resending packet...
Packet 4 sent successfully!
ACK received for Packet 4!
Packet 4 successfully delivered.

Packet 5 lost! Resending...
Packet 5 sent successfully!
ACK received for Packet 5!
Packet 5 successfully delivered.

All packets successfully transmitted!
kunalbansal@KUNALs-MacBook-Pro CN LAB %
```

Experiment-5

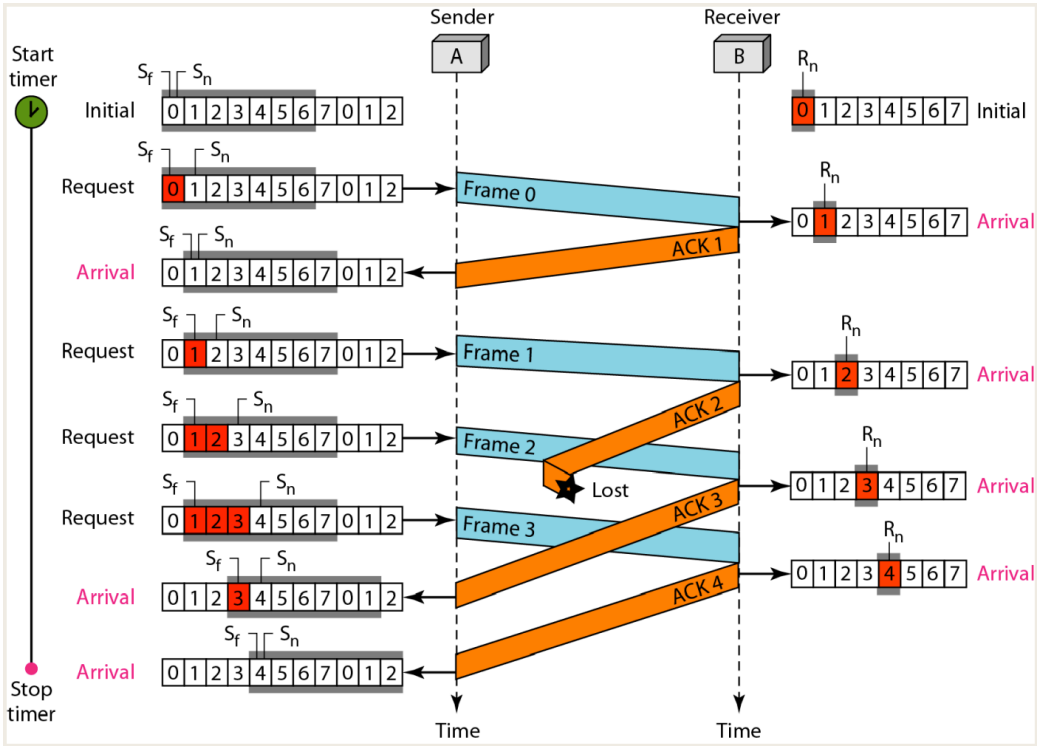
Aim : Implement Go-Back-N ARQ Protocol for efficient data transmission. Handle three cases: when the frame and acknowledgement are delivered successfully, when the frame is lost and when acknowledgement is lost. Show the role of the timeout timer in the protocol.

Theory :

Type: Sliding window protocol used for reliable data transmission in networks.
Sender Window: Can send multiple frames (up to a fixed window size, N) before requiring an acknowledgment (ACK).
Receiver Window: Always fixed to 1, meaning it only accepts frames in order.
ACK Mechanism:

- Receiver sends cumulative ACK for the last correctly received frame.
- If a frame is lost or corrupted, the receiver discards all subsequent frames until the missing one is received.

Retransmission: If an ACK for a frame is not received within a timeout, the sender retransmits all frames from that frame onwards.
Efficiency: Less efficient than Selective Repeat (SR) because it may retransmit unnecessary frames.
Used In: TCP (with some modifications), Wireless and Satellite Networks.



Code :

```
#include <iostream>
#include <ctime>
#define ll long long int
using namespace std;

void transmission(ll &i, ll &N, ll &tf)
{
  while (i <= tf)
  {
    int z = 0;
    for (int k = i; k < i + N && k <= tf; k++)
    {
      cout << "Sending Frame " << k << "..." << endl;
    }
    for (int k = i; k < i + N && k <= tf; k++)
    {
      int f = rand() % 2;
      if (!f)
      {
        cout << "Acknowledgment for Frame " << k << "..." << endl;
        z++;
      }
      else
      {
        cout << "Timeout!! Frame Number : " << k << " Not
Received" << endl;
        cout << "Retransmitting Window..." << endl;
        break;
      }
    }
    cout << "\n";
    i = i + z;
  }
}

int main()
{
  ll tf, N = 0;
  srand(time(0));
  cout << "Enter the Total number of frames : ";
  cin >> tf;
  cout << "Enter the Window Size : ";
  cin >> N;
  ll i = 1;
  transmission(i, N, tf);

  return 0;}
```

Output :

```
Enter the Total number of frames : 5
Enter the Window Size : 4
Sending Frame 1...
Sending Frame 2...
Sending Frame 3...
Sending Frame 4...
Timeout!! Frame Number : 1 Not Received
Retransmitting Window...

Sending Frame 1...
Sending Frame 2...
Sending Frame 3...
Sending Frame 4...
Acknowledgment for Frame 1...
Acknowledgment for Frame 2...
Timeout!! Frame Number : 3 Not Received
Retransmitting Window...

Sending Frame 3...
Sending Frame 4...
Sending Frame 5...
Timeout!! Frame Number : 3 Not Received
Retransmitting Window...

Sending Frame 3...
Sending Frame 4...
Sending Frame 5...
Acknowledgment for Frame 3...
Timeout!! Frame Number : 4 Not Received
Retransmitting Window...

Sending Frame 4...
Sending Frame 5...
Acknowledgment for Frame 4...
Timeout!! Frame Number : 5 Not Received
Retransmitting Window...

Sending Frame 5...
Timeout!! Frame Number : 5 Not Received
Retransmitting Window...

Sending Frame 5...
Timeout!! Frame Number : 5 Not Received
Retransmitting Window...

Sending Frame 5...
Acknowledgment for Frame 5...

kunalbansal@KUNALs-MacBook-Pro CN LAB %
```

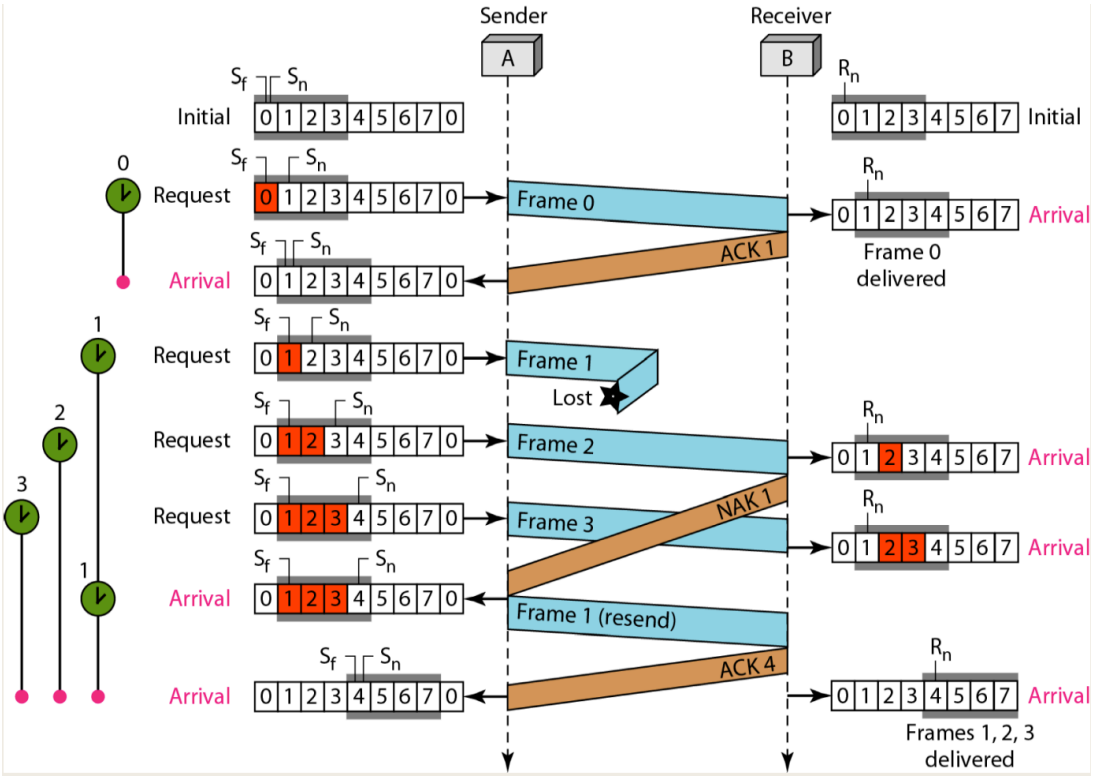
Experiment-6

Aim : Implement Selective Repeat ARQ Protocol for efficient data transmission. Handle three cases: when the frame and acknowledgement are delivered successfully, when the frame is lost and when acknowledgement is lost. Show the role of the timeout timer in the protocol.

Theory :

The **Selective Repeat ARQ** protocol ensures reliable data transmission by retransmitting only lost or unacknowledged frames, improving efficiency.

- **Successful Transmission**: If both the frame and its acknowledgment (ACK) are received correctly, the sender slides the window forward and transmits the next frame.
- **Frame Loss**: If a frame is lost, the receiver does not acknowledge it, triggering the sender's timeout timer, which leads to retransmission of only the lost frame.
- **Acknowledgment Loss**: If an ACK is lost, the sender's timeout expires, causing retransmission of the corresponding frame, but the receiver discards duplicates using sequence numbers.



Code :

```
#include <iostream>
#include <vector>
#include <cstdlib>
#include <ctime>
#include <unistd.h>

using namespace std;

const int WINDOW_SIZE = 4;
const int TOTAL_FRAMES = 10;

struct Frame {
    int id;
    bool acknowledged;
};

void transmitFrame(Frame &frame) {
    cout << "Transmitting frame " << frame.id << "...\n";
    int chance = rand() % 10;
    if (chance < 2) {
        cout<<"Timeout !! Frame Lost...."<<endl;
        cout << "Frame " << frame.id << " lost during
transmission!\n";
        return;
    }
    frame.acknowledged = false;
    cout << "Frame " << frame.id << " successfully transmitted."
<< endl;
}

bool checkAck(Frame &frame) {
    int chance = rand() % 10;
    if (chance < 2) {
        cout << "ACK for frame " << frame.id << " lost!\n";
        return false;
    }
    frame.acknowledged = true;
    cout << "ACK received for frame " << frame.id << endl;
    return true;
}

int main() {
    srand(time(0));
    vector<Frame> window;
    int frameIndex = 0;

    while (frameIndex < TOTAL_FRAMES || !window.empty()) {
        while (window.size() < WINDOW_SIZE && frameIndex <
TOTAL_FRAMES) {
```

```
            Frame frame = {frameIndex, false};
            transmitFrame(frame);
            window.push_back(frame);
            frameIndex++;
        }

        for (auto it = window.begin(); it != window.end();) {
            if (!it->acknowledged && checkAck(*it)) {
                it = window.erase(it);
            } else {
                ++it;
            }
        }
        sleep(1);
    }

    cout << "All frames successfully transmitted and acknowledged!
\n";
    return 0;
}
```

Output :

```
Transmitting frame 0...
Frame 0 successfully transmitted.
Transmitting frame 1...
Frame 1 successfully transmitted.
Transmitting frame 2...
Frame 2 successfully transmitted.
Transmitting frame 3...
Timeout !! Frame Lost....
Frame 3 lost during transmission!
ACK received for frame 0
ACK for frame 1 lost!
ACK received for frame 2
ACK received for frame 3
Transmitting frame 4...
Frame 4 successfully transmitted.
Transmitting frame 5...
Frame 5 successfully transmitted.
Transmitting frame 6...
Frame 6 successfully transmitted.
ACK received for frame 1
ACK for frame 4 lost!
ACK received for frame 5
ACK received for frame 6
Transmitting frame 7...
Timeout !! Frame Lost....
Frame 7 lost during transmission!
Transmitting frame 8...
Frame 8 successfully transmitted.
Transmitting frame 9...
Timeout !! Frame Lost....
Frame 9 lost during transmission!
ACK received for frame 4
ACK received for frame 7
ACK received for frame 8
ACK received for frame 9
All frames successfully transmitted and acknowledged!
kunalbansal@KUNALs-MacBook-Pro CN LAB %
```

Experiment-7

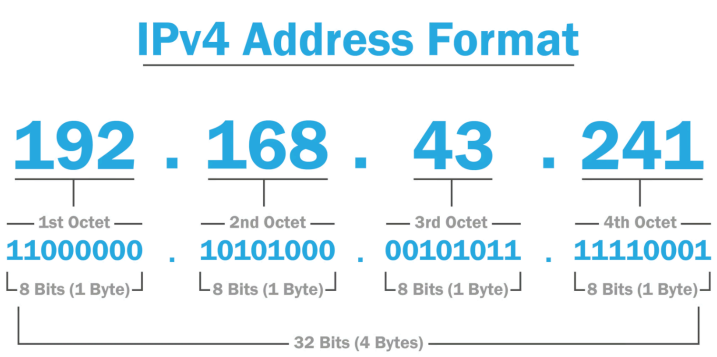
Aim : To parse and display network information from an IP address.

Theory :

Classful Addressing (IP Addressing)

Classful addressing was an early method of dividing the IP address space into five distinct classes (A, B, C, D, and E). Each class had a different range and purpose. The system divided the 32-bit IP address into two parts: Network ID and Host ID.

- **Class A (0.0.0.0 - 127.255.255.255)**
  - Network ID: The first 8 bits (1 byte) represent the network.
  - Host ID: The remaining 24 bits (3 bytes) represent the host.
- **Class B (128.0.0.0 - 191.255.255.255)**
  - Network ID: The first 16 bits (2 bytes) represent the network.
  - Host ID: The remaining 16 bits (2 bytes) represent the host.
- **Class C (192.0.0.0 - 223.255.255.255)**
  - Network ID: The first 24 bits (3 bytes) represent the network.
  - Host ID: The remaining 8 bits (1 byte) represent the host.
- **Class D (224.0.0.0 - 239.255.255.255)**
  - Purpose: Reserved for multicast addresses (used for sending data to multiple destinations).
  - Network ID: The full 32-bit address is used for identifying the multicast group.
- **Class E (240.0.0.0 - 255.255.255.255)**
  - Purpose: Reserved for experimental or future use (not used in typical networking).



Address Class	RANGE	Default Subnet Mask
A	1.0.0.0 to 126.255.255.255	255.0.0.0
B	128.0.0.0 to 191.255.255.255	255.255.0.0
C	192.0.0.0 to 223.255.255.255	255.255.255.0
D	224.0.0.0 to 239.255.255.255	Reserved for Multicasting
E	240.0.0.0 to 254.255.255.255	Experimental

Note: Class A addresses 127.0.0.0 to 127.255.255.255 cannot be used and is reserved for loopback testing.

Code :

```
#include<iostream>
using namespace std;

void print(string ip) {
    if (ip.length() != 32) {
        cout << "Invalid IP length. Must be 32 bits." << endl;
        return;
    }

    cout << "IP Address (Binary) : ";
    for (int i = 0; i < 32; i++) {
        cout << ip[i];
        if ((i + 1) % 8 == 0 && i != 31)
            cout << ".";
    }
    cout << endl;

    cout << "IP Address in Decimal : ";
    for (int i = 0; i < 32; i += 8) {
        string part = ip.substr(i, 8);
        int val = stoi(part, nullptr, 2);
        cout << val;
        if (i != 24)
            cout << ".";
    }
    cout << endl;}

int main(){
    string ip;
    cout<<"Enter IP Address (Binary Sequence without dots) : ";
    cin>>ip;
    cout<<endl<<"Processing......"<<endl;
    //class A
    if(ip.substr(0,1)=="0"){
        print(ip);
        cout<<"Class : A"<<endl;
        cout<<"Network_ID : "<<ip.substr(0,4)<<endl;
        cout<<"Network_ID (INT) :
"<<stoi(ip.substr(0,4),0,2)<<endl;
        cout<<"Host_ID : "<<ip.substr(4)<<endl;
        cout<<"Host_ID  (INT)  : "<<stoll(ip.substr(4),0,2)<<endl;
    }
    //class B
    else if(ip.substr(0,2)=="10"){
        print(ip);
        cout<<"Class : B"<<endl;
        cout<<"Network_ID : "<<ip.substr(0,16)<<endl;
        cout<<"Network_ID (INT) :
"<<stoi(ip.substr(0,16),0,2)<<endl;
        cout<<"Host_ID : "<<ip.substr(16)<<endl;
```

```
        cout<<"Host_ID  (INT)  :
"<<stoll(ip.substr(16),0,2)<<endl;

    }
    //class C
    else if(ip.substr(0,3)=="110"){
        print(ip);
        cout<<"Class : C"<<endl;
        cout<<"Network_ID : "<<ip.substr(0,24)<<endl;
        cout<<"Network_ID (INT) :
"<<stoi(ip.substr(0,24),0,2)<<endl;
        cout<<"Host_ID : "<<ip.substr(24)<<endl;
        cout<<"Host_ID  (INT)  :
"<<stoll(ip.substr(24),0,2)<<endl;

    }
    //class D
    else if(ip.substr(0,4)=="1110"){
        print(ip);
        cout<<"Class : D"<<endl;
        cout<<"Network_ID : "<<ip.substr(0,32)<<endl;
        cout<<"Network_ID (INT) :
"<<stoll(ip.substr(0,32),0,2)<<endl;
        cout<<"Host_ID : "<<"Not Applicable"<<endl;
        cout<<"Host_ID  (INT)  : "<<"Not Applicable"<<endl;
    }
    //class E
    else if(ip.substr(0,4)=="1111"){
        cout<<"Class : E"<<endl;
        cout<<"Network_ID : "<<ip.substr(0,32)<<endl;
        cout<<"Network_ID (INT) :
"<<stoll(ip.substr(0,32),0,2)<<endl;
        cout<<"Host_ID : "<<"Not Applicable"<<endl;
        cout<<"Host_ID  (INT)  : "<<"Not Applicable"<<endl;

    }
    else{
        cout<<"Error in IP Sequence..."<<endl;}}
```

Output :

```
Processing......
IP Address (Binary) : 11000000.10101000.00101011.11110001
IP Address in Decimal : 192.168.43.241
Class : C
Network_ID : 110000001010100000101011
Network_ID (INT) : 12625963
Host_ID : 11110001
Host_ID  (INT)  : 241
```

Experiment-8

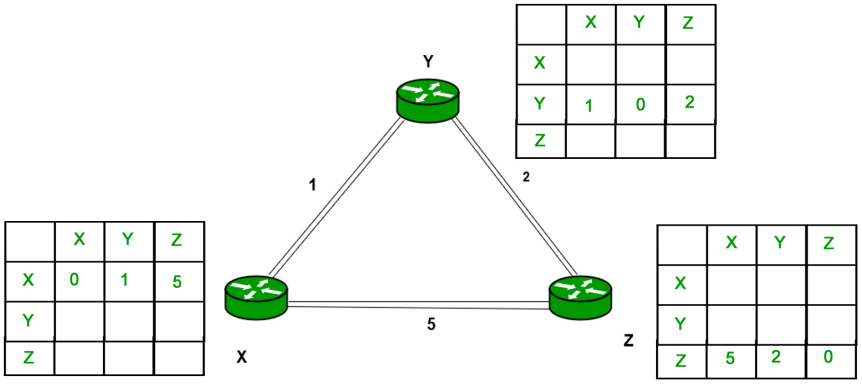
Aim : To implement the Distance Vector Routing Algorithm for routing table calculation

Theory :

The **Distance Vector Routing Algorithm** is a dynamic routing protocol that determines the shortest path between nodes in a network using the **Bellman-Ford algorithm**. Each node maintains a routing table with distances to all other nodes and exchanges this information with its directly connected neighbors. The routing table is updated iteratively based on the formula

$$d(i, j) = \min\{ d(i, j), d(i, k) + d(k, j) \}$$

until convergence is reached. While simple and efficient for small networks, the algorithm suffers from **slow convergence** and the **count-to-infinity problem**. It is commonly used in **Routing Information Protocol (RIP)** for network communication.



Code :

```
#include <iostream>
using namespace std;

#define INF 1e9

class Edge {
public:
    int src, dest, weight;
    Edge(int s, int d, int w) {
        src = s;
        dest = d;
        weight = w;
    }
};

class DistanceVectorRouting {
private:
    int V;
    vector<Edge> edges;

public:
    DistanceVectorRouting(int vertices) {
        V = vertices;
    }

    void addEdge(int src, int dest, int weight) {
        edges.push_back(Edge(src, dest, weight));
        edges.push_back(Edge(dest, src, weight));
    }

    void bellmanFord(int src) {
        vector<int> dist(V, INF);
        vector<int> nextHop(V, -1);
        dist[src] = 0;
        nextHop[src] = src;

        // Relax all edges (V-1) times
        for (int i = 0; i < V - 1; i++) {
            for (int j = 0; j < edges.size(); j++) {
                int u = edges[j].src;
                int v = edges[j].dest;
                int w = edges[j].weight;

                if (dist[u] != INF && dist[u] + w < dist[v]) {
                    dist[v] = dist[u] + w;
                    nextHop[v] = (nextHop[u] == src) ? v :
nextHop[u]; // Track next hop
                }
            }
```

```
        }

        // Print Routing Table
        cout << "Routing Table for Node " << src << ":\n";
        cout << "Destination\tNext Hop\tCost\n";
        cout << "------------------------------\n";
        for (int i = 0; i < V; i++) {
            if (i == src) {
                cout << i << "\t\t" << "-" << "\t\t" << 0 << "\n";
            } else if (dist[i] == INF) {
                cout << i << "\t\t" << "-" << "\t\t∞\n";
            } else {
                cout << i << "\t\t" << nextHop[i] << "\t\t" <<
dist[i] << "\n";
            }
        }
        cout << "------------------------------\n";
    }

    void calculateRoutingTables() {
        for (int i = 0; i < V; i++) {
            bellmanFord(i);
        }
    }
};

int main() {
    int V, E;
    cout << "Enter number of nodes and links: ";
    cin >> V >> E;

    DistanceVectorRouting dvr(V);

    cout << "Enter edges (source destination cost):\n";
    for (int i = 0; i < E; i++) {
        int u, v, w;
        cin >> u >> v >> w;
        dvr.addEdge(u, v, w);
    }

    dvr.calculateRoutingTables();

    return 0;
}
```

Output :

```
Enter number of nodes and links: 4 5
Enter edges (source destination cost):
0 1 4
0 2 1
1 2 2
1 3 5
2 3 3
Routing Table for Node 0:
Destination     Next Hop        Cost
-------------------------------------
0               -               0
1               2               3
2               2               1
3               2               4
-------------------------------------
Routing Table for Node 1:
Destination     Next Hop        Cost
-------------------------------------
0               2               3
1               -               0
2               2               2
3               3               5
-------------------------------------
Routing Table for Node 2:
Destination     Next Hop        Cost
-------------------------------------
0               0               1
1               1               2
2               -               0
3               3               3
-------------------------------------
Routing Table for Node 3:
Destination     Next Hop        Cost
-------------------------------------
0               2               4
1               1               5
2               2               3
3               -               0
-------------------------------------
kunalbansal@KUNALs-MacBook-Pro CN LAB %
```

Experiment-9

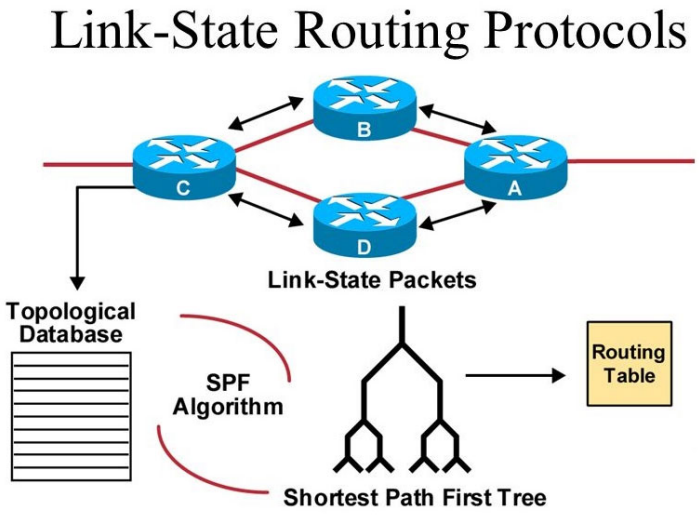
Aim : To implement the Link State Routing Algorithm for routing table calculation

Theory :

The Link State Algorithm is a shortest path routing algorithm used in computer networks. It is based on Dijkstra's Algorithm and works by maintaining a complete topology of the network. Each router follows these steps:

1. **Neighbor Discovery:** Routers identify their direct neighbors and measure link costs.
2. **Link State Advertisement (LSA):** Each router floods the network with its link-state information.
3. **Topology Database Construction:** Every router builds a global view of the network.
4. **Shortest Path Calculation:** Using Dijkstra's Algorithm, the router computes the shortest paths from itself to all other nodes.

Routing Table Update: The computed paths are stored in the routing table for packet forwarding.



Code :

```
#include <iostream>
#include <queue>
#include <vector>
#include <algorithm>

using namespace std;

#define INF 1e9

class Graph {
private:
    int V;
    vector<vector<pair<int, int> > > adjList;

public:
    Graph(int vertices) {
        V = vertices;
        adjList.resize(V);
    }

    void addEdge(int src, int dest, int weight) {
        adjList[src].push_back(make_pair(dest, weight));
        adjList[dest].push_back(make_pair(src, weight));
    }

    void dijkstra(int src) {
        vector<int> dist(V, INF);
        vector<int> nextHop(V, -1);
        vector<bool> visited(V, false);
        priority_queue<pair<int, int>, vector<pair<int, int> >,
greater<pair<int, int> > > pq;

        dist[src] = 0;
        pq.push(make_pair(0, src));
        nextHop[src] = src;

        while (!pq.empty()) {
            int u = pq.top().second;
            pq.pop();

            if (visited[u]) continue;
            visited[u] = true;

            for (int i = 0; i < adjList[u].size(); i++) {
                int v = adjList[u][i].first;
                int weight = adjList[u][i].second;

                if (dist[u] + weight < dist[v]) {
                    dist[v] = dist[u] + weight;
                    pq.push(make_pair(dist[v], v));
```

```
                    nextHop[v] = (nextHop[u] == src) ? v :
nextHop[u];
                }
            }
        }


        cout << "Routing Table for Node " << src << ":\n";
        cout << "Destination\tNext Hop\tCost\n";
        cout << "------------------------------\n";
        for (int i = 0; i < V; i++) {
            if (i == src) {
                cout << i << "\t\t" << "-" << "\t\t" << 0 << "\n";
            } else if (dist[i] == INF) {
                cout << i << "\t\t" << "-" << "\t\t∞\n"; //
Unreachable node
            } else {
                cout << i << "\t\t" << nextHop[i] << "\t\t" <<
dist[i] << "\n";
            }
        }
        cout << "------------------------------\n";
    }

    void calculateRoutingTables() {
        for (int i = 0; i < V; i++) {
            dijkstra(i);
        }
    }
};

int main() {
    int V, E;
    cout << "Enter number of nodes and links: ";
    cin >> V >> E;

    Graph graph(V);

    cout << "Enter edges (source destination cost):\n";
    for (int i = 0; i < E; i++) {
        int u, v, w;
        cin >> u >> v >> w;
        graph.addEdge(u, v, w);
    }

    graph.calculateRoutingTables();

    return 0;
}
```

Output :

```
Enter number of nodes and links: 4 5
Enter edges (source destination cost):
0 1 4
0 2 1
1 2 2
1 3 5
2 3 3
Routing Table for Node 0:
Destination     Next Hop        Cost
-----------------------------------
0               -               0
1               2               3
2               2               1
3               2               4
-----------------------------------
Routing Table for Node 1:
Destination     Next Hop        Cost
-----------------------------------
0               2               3
1               -               0
2               2               2
3               3               5
-----------------------------------
Routing Table for Node 2:
Destination     Next Hop        Cost
-----------------------------------
0               0               1
1               1               2
2               -               0
3               3               3
-----------------------------------
Routing Table for Node 3:
Destination     Next Hop        Cost
-----------------------------------
0               2               4
1               1               5
2               2               3
3               -               0
-----------------------------------
kunalbansal@KUNALs-MacBook-Pro CN LAB %
```

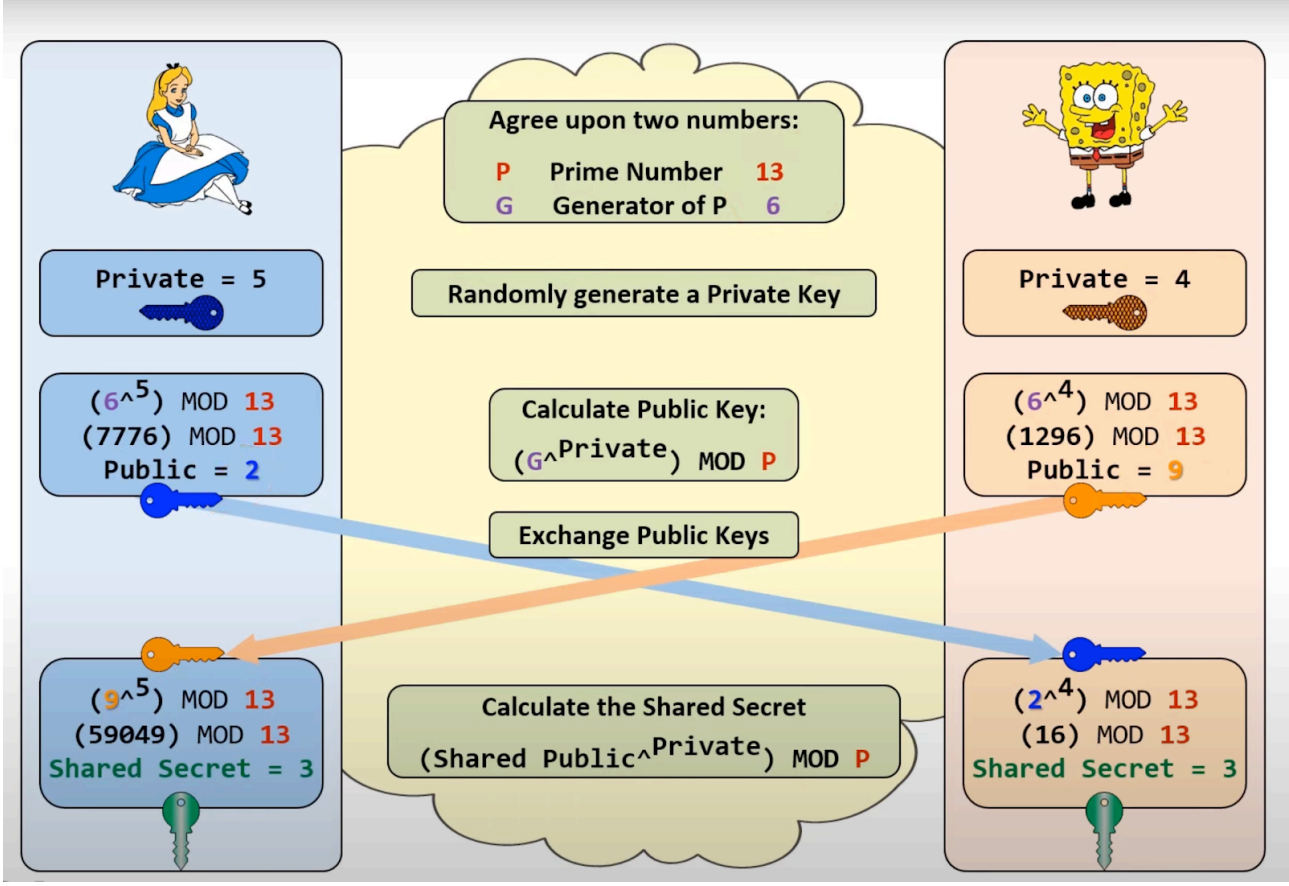
Experiment-10

Aim : To implement the Diffie-Hellman key exchange algorithm for secure key exchange over an insecure channel.

Theory :

The **Diffie-Hellman key exchange** (also known as exponential key exchange) is a method for securely exchanging cryptographic keys over an insecure channel. It is a fundamental building block of many secure communication protocols, including **SSL**/**TLS** and SSH.

The **Diffie-Hellman** key exchange works by allowing two parties (Alice and Bob) to agree on a shared secret key over an insecure channel, without any other party being able to intercept the key or learn anything about it.



Code :

```
#include<iostream>
using namespace std;

int main(){
    int p;
    int g;
    cout<<"Enter Prime No(p) & Primitive Root No(g) :";
    cin>>p>>g;
    int a,b;
    cout<<endl<<"Enter A's private key : ";
    cin>>a;
    cout<<endl<<"Enter B's private key  : ";
    cin>>b;
    long long a_public = pow(g,a);
    long long b_public = pow(g,b);
    cout<<endl<<"Data shared by A publicaly : "<<a_public%p;
    cout<<endl<<"Data shared by B publicaly : "<<b_public%p;
    cout<<endl<<"Data Interpreted by A: "<<int(pow(b_public%p,a))
%p;
    cout<<endl<<"Data Interpreted by B: "<<int(pow(a_public%p,b))
%p<<endl;


}
```

Output :

```
Enter Prime No(p) & Primitive Root No(g) :97 10

Enter A's private key : 8

Enter B's private key  : 5

Data shared by A publicaly : 81
Data shared by B publicaly : 90
Data Interpreted by A: 65
Data Interpreted by B: 65
kunalbansal@KUNALs-MacBook-Pro CN LAB %
```