Experiment-1

1. Study of Network devices in detail and connect the computers in Local Area Network.

Network Devices Used

1. **Switch** – Connects multiple computers in a LAN using MAC addressing for data forwarding.
2. **Router** – Connects networks or subnets and provides IP-based routing.
3. **Hub** – A basic device that broadcasts signals to all connected systems (less efficient than switches).
4. **Network Interface Card (NIC)** – A hardware component in each PC allowing physical network connection through Ethernet cables.
5. **Cables**
    - Straight-through cable: Used to connect PCs to switches.
    - Crossover cable: Used to connect similar devices such as switch-to-switch or hub-to-hub.
6. **Server and Printer (optional)** – Used for resource sharing across the LAN.

Procedure

1. Select Devices
   Choose 4–5 PCs, 1 switch, 1 router, and necessary Ethernet cables.
2. Physical Connections
   Connect all PCs to the switch using straight-through cables. Connect the router to the switch using another straight-through cable.
3. IP Configuration
   Assign IP addresses manually to each PC:
    - PC1 → 192.168.1.1
    - PC2 → 192.168.1.2
    - PC3 → 192.168.1.3
    - PC4 → 192.168.1.4
      Router IP: 192.168.1.100 (gateway).

4. Network Verification
   Open Command Prompt and use:

   ping 192.168.1.2

5. Resource Sharing
   Configure one PC as a file or print server to demonstrate shared resource access

Experiment-2

(i) **Character stuffing** is a technique used in data communication (especially at the Data Link Layer) to ensure special "delimiter" characters within the data don't accidentally terminate or confuse the frame structure. Whenever a reserved character occurs within data, an extra escape character is inserted to distinguish it from an actual frame boundary.

How the Program Works

- **User Input:** Accepts the string to be transmitted and the chosen start/end delimiter characters.
- **Stuffing Logic:** Scans through the data:
  - If a data character matches the start or end delimiter, it is duplicated (i.e., "stuffed") to distinguish from real frame boundaries.
- **Framing:** The start and end delimiters are prepended and appended to the result.
- **Output:** Shows the final stuffed frame, ready for safe network transmission.

Program:

```
#include <stdio.h>

#include <string.h>

int main() {

    char input[30], stuffed[80] = "";

    char start_delim, end_delim;

    char temp[3], double_start[3], double_end[3];

    int i;

    // Input Section

    printf("Enter the data to be stuffed: ");

    scanf("%s", input);

    printf("Enter the starting delimiter character: ");

    scanf(" %c", &start_delim);

    printf("Enter the ending delimiter character: ");
```

```
    scanf(" %c", &end_delim);
    // Prepare delimiter substrings
    double_start[0] = double_start[1] = start_delim;
    double_start[2] = '\0';
    double_end[0] = double_end[1] = end_delim;
    double_end[2] = '\0';
    // Add starting delimiter
    strcat(stuffed, double_start);
    // Stuffing logic
    for(i = 0; i < strlen(input); i++) {
        temp[0] = input[i];
        temp[1] = '\0';
        if(input[i] == start_delim)
            strcat(stuffed, double_start);  // Stuff start delimiter again
        else if(input[i] == end_delim)
            strcat(stuffed, double_end);    // Stuff end delimiter again
        else
            strcat(stuffed, temp);
    }
    // Add ending delimiter
    strcat(stuffed, double_end);

    printf("Data after character stuffing: %s\n", stuffed);

    return 0;
}
```

Output 1:

Enter the data to be stuffed: **goodday**

Enter the starting delimiter character: **d**

Enter the ending delimiter character: **g**

Data after character stuffing: **dggooddddayg**

Output 2:

Enter the data to be stuffed: **VITCSE**

Enter the starting delimiter character: **A**

Enter the ending delimiter character: **Z**

Data after character stuffing: **AAVITCSEZZ**

(b) Bit Stuffing:

Bit stuffing is used in data communications to ensure a predefined pattern (like a flag 01111110 in HDLC protocols) does not accidentally appear in data. The rule for most protocols is: after five consecutive 1s in the data, insert a 0.

How It Works

- **Input:** The user provides the original bit stream.
- **Logic:** The program copies each bit to the output. If it finds five consecutive 1s, it inserts a 0 immediately after.
- **Output:** Shows the bit-stuffed stream ready for transmission.

Program:

```
#include <stdio.h>

#include <string.h>

int main() {

    char input[100], stuffed[200];

    int i, j = 0, count = 0;

    printf("Enter the bit stream (only 0s and 1s): ");

    scanf("%s", input);

    int len = strlen(input);

    for(i = 0; i < len; i++) {

        stuffed[j++] = input[i];


        if(input[i] == '1') {

            count++;

            if(count == 5) {

                stuffed[j++] = '0'; // Insert a 0 after five consecutive 1s

                count = 0;          // Reset the count
```

```
            }
        } else {
            count = 0; // Reset if the current bit is 0
        }
    }
    stuffed[j] = '\0';
    printf("Bit-stuffed output: %s\n", stuffed);
    return 0;
}
```

Output 1:

Enter the bit stream (only 0s and 1s): 0111111011

Bit-stuffed output: 01111101011

Output 2:

Enter the bit stream (only 0s and 1s): 01111110

Bit-stuffed output: 011111010

Experiment-3

write a program to implement data link layer framing method checksum.

Description: The **checksum** method is widely used in data link layer protocols for error detection. A checksum is calculated over the data and appended to the frame. The receiver recalculates the checksum and compares it with the received one to detect errors introduced during transmission.

Explanation:

- **Framing:** Uses 0x7E as the start-of-frame (SOF) and 0x7F as the end-of-frame (EOF) delimiter bytes (can be set as per protocol).
- **Checksum Calculation:** Sums all data bytes modulo 256 (one-byte checksum).
- **Sender:** Packs data, checksum, and delimiters into a frame.
- **Receiver:** Checks frame delimiters, extracts data, recalculates checksum, and compares with received checksum.
- **Error Detection:** If frame structure or checksum doesn't match, an error is detected.

Program:

```
#include <stdio.h>

#include <string.h>


// Function to calculate checksum (simple byte-sum modulo 256)

unsigned char calculate_checksum(const char *data) {

    unsigned int sum = 0;

    for (int i = 0; data[i] != '\0'; i++) {

        sum += (unsigned char)data[i];

    }

    return (unsigned char)(sum % 256); // Modulo 256 for one-byte checksum

}
```

```
// Function to simulate transmission (framing and checksum)
void send_frame(const char *data, unsigned char *frame, int *frame_len) {
    unsigned char checksum = calculate_checksum(data);
    int data_len = strlen(data);

    // Frame format: | SOF (Start of Frame) | DATA | CHECKSUM | EOF (End of Frame) |
    frame[0] = 0x7E; // SOF byte (01111110 in HDLC)
    memcpy(&frame[1], data, data_len);
    frame[1 + data_len] = checksum;
    frame[2 + data_len] = 0x7F; // EOF byte (arbitrary choice)
    *frame_len = 3 + data_len;
}

// Function to simulate receiver frame check
int receive_frame(const unsigned char *frame, int frame_len, char *out_data) {
    if (frame[0] != 0x7E || frame[frame_len - 1] != 0x7F) {
        printf("Frame error: Invalid framing bytes.\n");
        return 0;
    }
    int data_len = frame_len - 3;
    memcpy(out_data, &frame[1], data_len);
    out_data[data_len] = '\0';
    unsigned char received_checksum = frame[1 + data_len];
    unsigned char calc_checksum = calculate_checksum(out_data);
    if (received_checksum != calc_checksum) {
        printf("Checksum error!\n");
        return 0;
```

```
    }
    return 1;
}

int main() {
    char data[100];
    unsigned char frame[110];
    char received_data[100];
    int frame_len;

    // Sender Side
    printf("Enter data to send: ");
    scanf("%s", data);
    send_frame(data, frame, &frame_len);

    printf("Transmitted Frame (in hex): ");
    for (int i = 0; i < frame_len; i++)
        printf("%02X ", frame[i]);
    printf("\n");
 // (Change third character in data)
//frame[3] = 0x4D;
    // Receiver Side
    if (receive_frame(frame, frame_len, received_data)) {
        printf("Received data: %s\n", received_data);
        printf("No error detected in frame.\n");
    }
    return 0;
```

}

Output 1:

Enter data to send: HELLO

Transmitted Frame (in hex): 7E 48 45 4C 4C 4F 26 7F

Received data: HELLO

No error detected in frame.

Output 2:

Enter data to send: Hello

Transmitted Frame (in hex): 7E 48 65 6C 6C 6F F4 7F

Checksum error!

Experiment-4

Write a program for Hamming-Code generation for error detection and correction.

Program:

```
#include <stdio.h>
// Function to calculate parity bits and generate Hamming code
void generateHammingCode(int data[], int hcode[]) {
    // Assign data bits to proper positions in the code
    // Positions: 1 2 3 4 5 6 7 (P1,P2,D1,P4,D2,D3,D4)
    hcode[2] = data[0]; // D1
    hcode[4] = data[1]; // D2
    hcode[5] = data[2]; // D3
    hcode[6] = data[3]; // D4
    // Calculate parity bits for even parity
    hcode[0] = hcode[2] ^ hcode[4] ^ hcode[6]; // P1
    hcode[1] = hcode[2] ^ hcode[5] ^ hcode[6]; // P2
    hcode[3] = hcode[4] ^ hcode[5] ^ hcode[6]; // P4
}

int main() {
    int data[4], hcode[7];
    printf("Enter 4 data bits (space-separated): ");
    for(int i = 0; i < 4; i++)
        scanf("%d", &data[i]);
    generateHammingCode(data, hcode);

    printf("Generated 7-bit Hamming code: ");
```

```
for(int i = 0; i < 7; i++)
    printf("%d ", hcode[i]);
printf("\n");

// Optionally simulate an error
char opt;
printf("Simulate error? (y/n): ");
scanf(" %c", &opt);
if(opt == 'y' || opt == 'Y') {
    int pos;
    printf("Enter bit position to flip (1-7): ");
    scanf("%d", &pos);
    if(pos >= 1 && pos <=7) {
        hcode[pos-1] ^= 1;
        printf("Codeword after error: ");
        for(int i = 0; i < 7; i++)
            printf("%d ", hcode[i]);
        printf("\n");
    }
}
// Decode: error detection and correction
int p[3];
// parity checks: P1 (positions 0,2,4,6), P2 (1,2,5,6), P4 (3,4,5,6)
p[0] = hcode[0] ^ hcode[2] ^ hcode[4] ^ hcode[6];
p[1] = hcode[1] ^ hcode[2] ^ hcode[5] ^ hcode[6];
p[2] = hcode[3] ^ hcode[4] ^ hcode[5] ^ hcode[6];
int error_pos = p[2]*4 + p[1]*2 + p[0]*1; // binary to decimal
```

```
    if(error_pos == 0) {
        printf("No error detected in received code.\n");
    } else {
        printf("Error detected at bit position %d (counting from 1).\n", error_pos);
        hcode[error_pos-1] ^= 1; // Correct error
        printf("Corrected code: ");
        for(int i = 0; i < 7; i++)
            printf("%d ", hcode[i]);
        printf("\n");
    }
    printf("Extracted data bits: %d %d %d %d\n", hcode[2], hcode[4], hcode[5], hcode[6]);

    return 0;
}
```

Output 1:

Enter 4 data bits (space-separated): 1 0 0 1

Generated 7-bit Hamming code: 0 0 1 1 0 0 1

Simulate error? (y/n): n

No error detected in received code.

Extracted data bits: 1 0 0 1

Output 2:

Enter 4 data bits (space-separated): 1 0 0 1

Generated 7-bit Hamming code: 0 0 1 1 0 0 1

Simulate error? (y/n): y

Enter bit position to flip (1-7): 4

ERROR!

Codeword after error: 0 0 1 0 0 0 1

Error detected at bit position 4 (counting from 1).

Corrected code: 0 0 1 1 0 0 1

Extracted data bits: 1 0 0 1

Experiment-5

Write a program to implement on a data set of characters the three CRC polynomials CRC 12, CRC 16, CRC CCIP

Note:

- **CRC-12:** $x^{12} + x^{11} + x^3 + x^2 + x + 1$ (poly: 0x180F, used e.g. in SDLC)
- **CRC-16:** $x^{16} + x^{15} + x^2 + 1$ (poly: 0x8005, common CRC-16-IBM)
- **CRC-CCITT (CRC-16-CCITT):** $x^{16} + x^{12} + x^5 + 1$ (poly: 0x1021)

Program:

```
#include <stdio.h>

#include <stdint.h>

#include <string.h>


// initial CRC values

#define CRC16_INIT      0x0000

#define CRCCCITT_INIT   0xFFFF

#define CRC12_INIT      0x000


// CRC-12: x^12 + x^11 + x^3 + x^2 + x + 1 = 0x180F

uint16_t crc12(const uint8_t *data, size_t len) {

    uint16_t crc = CRC12_INIT;

    uint16_t poly = 0x180F; // 0001 1000 0000 1111

    for (size_t i = 0; i < len; i++) {

        crc ^= data[i] << 4;    // Align data with high end of crc (data is 8 bits, CRC is 12 bits)

        for (int j = 0; j < 8; j++) {

            if (crc & 0x800)    // If highest (12th) bit is set

                crc = (crc << 1) ^ poly;
```

```
            else
                crc <<= 1;
            crc &= 0xFFF;       // Mask to 12 bits
        }
    }
    return crc;
}
```

// CRC-16: x^16 + x^15 + x^2 + 1 = 0x8005 (CRC-16-IBM)

```
uint16_t crc16(const uint8_t *data, size_t len) {
    uint16_t crc = CRC16_INIT;
    uint16_t poly = 0x8005; // 1000 0000 0000 0101

    for (size_t i = 0; i < len; i++) {
        crc ^= data[i] << 8;
        for (int j = 0; j < 8; j++) {
            if (crc & 0x8000)
                crc = (crc << 1) ^ poly;
            else
                crc <<= 1;
        }
    }
    return crc;
}
```

```
// CRC-CCITT: x^16 + x^12 + x^5 + 1 = 0x1021
uint16_t crc_ccitt(const uint8_t *data, size_t len) {
    uint16_t crc = CRCCCITT_INIT;
    uint16_t poly = 0x1021;

    for (size_t i = 0; i < len; i++) {
        crc ^= data[i] << 8;
        for (int j = 0; j < 8; j++) {
            if (crc & 0x8000)
                crc = (crc << 1) ^ poly;
            else
                crc <<= 1;
        }
    }
    return crc;
}

int main() {
    char input[128];
    printf("Enter the data string: ");
    scanf("%127s", input);

    size_t len = strlen(input);

    uint16_t c12 = crc12((uint8_t*)input, len);
    uint16_t c16 = crc16((uint8_t*)input, len);
    uint16_t cccitt = crc_ccitt((uint8_t*)input, len);
```

```
    printf("\nCRC Results:\n");
    printf("CRC-12:    0x%03X (12 bits)\n", c12);
    printf("CRC-16:    0x%04X (16 bits)\n", c16);
    printf("CRC-CCITT: 0x%04X (16 bits)\n", cccitt);

    return 0;
}
```

Output 1:

```
Enter the data string: 1101011011
CRC Results:
CRC-12:    0xD47 (12 bits)
CRC-16:    0x0496 (16 bits)
CRC-CCITT: 0x9DEB (16 bits)
```

Output 2:

```
Enter the data string: 1010000

CRC Results:
CRC-12:    0x7F2 (12 bits)
CRC-16:    0x59B1 (16 bits)
CRC-CCITT: 0x439B (16 bits)
```

Experiment-6

Write a program to implement Sliding Window Protocol for Go-Back-N.

Program:

```
#include <stdio.h>

#include <stdlib.h>

#include <time.h>


#define MAX_FRAMES 50

#define WINDOW_SIZE 4


// Randomly decide if a frame is lost (simulate error)

int is_frame_lost() {

   // 20% loss chance

   return (rand() % 5) == 0;

}


int main() {

   int total_frames, sent = 0, ack = 0, to_send, i;


   srand((unsigned)time(NULL));

   printf("Enter total number of frames to send (max %d): ", MAX_FRAMES);

   scanf("%d", &total_frames);


   printf("\n--- Sending frames using Go-Back-N with window size %d ---\n",
WINDOW_SIZE);
```

```
while (ack < total_frames) {
    // Determine how many frames can be sent in this window
    to_send = 0;
    for (i = 0; i < WINDOW_SIZE && sent + i < total_frames; i++)
        to_send++;

    printf("\nSender window: [");
    for (i = 0; i < to_send; i++)
        printf("%d ", sent + i + 1);
    printf("]\n");

    // Simulate sending frames in the window
    int error_index = -1;
    for (i = 0; i < to_send; i++) {
        if (is_frame_lost()) {
            printf("Frame %d lost or corrupted!\n", sent + i + 1);
            error_index = i;
            break;
        } else {
            printf("Frame %d sent successfully.\n", sent + i + 1);
        }
    }
    // Receiver logic
    if (error_index == -1) {
        // All frames received correctly, ACK all
        printf("Receiver: ACK for all %d frames.\n", to_send);
        sent += to_send;
```

```
            ack += to_send;
        } else {
            // NACK for erroneous frame and all after it: Go-Back-N
            printf("Receiver: NACK for frame %d. Go-Back-N triggered.\n", sent + error_index +
1);
            printf("Receiver: Discards all frames after and incl. frame %d.\n", sent + error_index +
1);
            // Resend from the error frame
            sent += error_index;
            ack += error_index;
        }
    }

    printf("\nAll frames sent and acknowledged successfully!\n");
    return 0;
}
```

Output 1:

```
Enter total number of frames to send (max 50): 3


--- Sending frames using Go-Back-N with window size 4 ---


Sender window: [1 2 3 ]
Frame 1 sent successfully.
Frame 2 sent successfully.
Frame 3 sent successfully.
Receiver: ACK for all 3 frames.
```

All frames sent and acknowledged successfully!

Output 2:

Enter total number of frames to send (max 50): 4

--- Sending frames using Go-Back-N with window size 4 ---

Sender window: [1 2 3 4]

Frame 1 sent successfully.

Frame 2 sent successfully.

Frame 3 lost or corrupted!

Receiver: NACK for frame 3. Go-Back-N triggered.

Receiver: Discards all frames after and incl. frame 3.

Sender window: [3 4]

Frame 3 sent successfully.

Frame 4 sent successfully.

Receiver: ACK for all 2 frames.

All frames sent and acknowledged successfully!

Experiment-7

Write a program to implement Sliding Window Protocol for Selective Repeat.

Program:

```
#include <stdio.h>
#include <stdlib.h>

#define MAX_FRAMES 10

// Simulating frame transmission
typedef struct {
    int frame_no;
    int acked; // 0: not acked, 1: acked
} Frame;

void send_frame(int frame_no) {
    printf("Sender: Sent frame %d\n", frame_no);
}

void receive_frame(int frame_no) {
    printf("Receiver: Received frame %d\n", frame_no);
}

void send_ack(int frame_no) {
    printf("Receiver: Ack sent for frame %d\n", frame_no);
}

void receive_ack(int frame_no) {
```

```
    printf("Sender: Ack received for frame %d\n", frame_no);
}

int main() {
    int total_frames, window_size;
    int sender_base = 0;
    int next_frame_to_send = 0;
    int receiver_expected = 0;
    int ack[MAX_FRAMES] = {0};
    Frame window[MAX_FRAMES];

    printf("Enter total number of frames to send: ");
    scanf("%d", &total_frames);
    printf("Enter window size: ");
    scanf("%d", &window_size);

    // Initialize window
    for(int i=0; i<total_frames; ++i) {
        window[i].frame_no = i;
        window[i].acked = 0;
    }

    // Sender side
    while (sender_base < total_frames) {
        // Send frames in window
        while (next_frame_to_send < sender_base + window_size && next_frame_to_send <
total_frames) {
```

```
        if (window[next_frame_to_send].acked == 0) {
            send_frame(window[next_frame_to_send].frame_no);
        }
        next_frame_to_send++;
    }

    // Simulate receiver randomly dropping some frames
    int recv;
    printf("Enter received frame number (or -1 if lost): ");
    scanf("%d", &recv);

    if (recv >= 0 && recv < total_frames && window[recv].acked == 0) {
        receive_frame(recv);
        send_ack(recv);
        window[recv].acked = 1;
    }

    // Simulate sender receiving acks (all acks up to and including the last in-order acked
frame)
    printf("Enter ack number received by sender (-1 if none): ");
    int ack_no;
    scanf("%d", &ack_no);
    if (ack_no >= 0 && ack_no < total_frames && window[ack_no].acked == 1) {
        receive_ack(ack_no);
    }

    // Slide the window for every in-order ack
```

```
        while (sender_base < total_frames && window[sender_base].acked == 1) {
            sender_base++;
        }
    }

    printf("All frames sent and acknowledged!\n");
    return 0;
}
```

Output 1:

Enter total number of frames to send: 5

Enter window size: 3

Sender: Sent frame 0

Sender: Sent frame 1

Sender: Sent frame 2

Enter received frame number (or -1 if lost): 0

Receiver: Received frame 0

Receiver: Ack sent for frame 0

Enter ack number received by sender (-1 if none): 0

Sender: Ack received for frame 0

Sender: Sent frame 3

Enter received frame number (or -1 if lost): 2

Receiver: Received frame 2

Receiver: Ack sent for frame 2

Enter ack number received by sender (-1 if none): 2

Sender: Ack received for frame 2

Enter received frame number (or -1 if lost): 3

Receiver: Received frame 3

Receiver: Ack sent for frame 3

Enter ack number received by sender (-1 if none): -1

Enter received frame number (or -1 if lost): 1

Receiver: Received frame 1

Receiver: Ack sent for frame 1

Enter ack number received by sender (-1 if none): 1

Sender: Ack received for frame 1

Sender: Sent frame 4

Enter received frame number (or -1 if lost): 4

Receiver: Received frame 4

Receiver: Ack sent for frame 4

Enter ack number received by sender (-1 if none): 4

Sender: Ack received for frame 4

All frames sent and acknowledged!

Experiment-8

Write a program to implement Stop and Wait Protocol.

Program:

```
#include <stdio.h>


int main() {
   int total_frames;
   int frame, ack;


   printf("Enter the total number of frames to send: ");
   scanf("%d", &total_frames);


   frame = 0;
   while (frame < total_frames) {
      printf("Sender: Sending Frame %d\n", frame);
      // Simulate receiver input: frame received or lost (-1)
      printf("Receiver: Enter received frame number (or -1 if the frame is lost): ");
      int recv;
      scanf("%d", &recv);


      if (recv == frame) {
         printf("Receiver: Frame %d received. Sending ACK %d\n", recv, recv);
         ack = recv;
      } else {
         printf("Receiver: Frame lost or out-of-order. No ACK sent.\n");
         ack = -1;
      }
```

```c
        // Simulate Sender reading ACK
        if (ack == frame) {
            printf("Sender: ACK %d received. Proceeding to next frame.\n\n", ack);
            frame++;
        } else {
            printf("Sender: No valid ACK received. Retransmitting Frame %d...\n\n", frame);
            // Do not increment frame; retransmit the same frame
        }
    }

    printf("All frames sent and acknowledged!\n");
    return 0;
}
```

Output 1:

Enter the total number of frames to send: 5

Sender: Sending Frame 0

Receiver: Enter received frame number (or -1 if the frame is lost): -1

Receiver: Frame lost or out-of-order. No ACK sent.

Sender: No valid ACK received. Retransmitting Frame 0...

Sender: Sending Frame 0

Receiver: Enter received frame number (or -1 if the frame is lost): 0

Receiver: Frame 0 received. Sending ACK 0

Sender: ACK 0 received. Proceeding to next frame.

Sender: Sending Frame 1

Receiver: Enter received frame number (or -1 if the frame is lost): 1

Receiver: Frame 1 received. Sending ACK 1

Sender: ACK 1 received. Proceeding to next frame.

Sender: Sending Frame 2

Receiver: Enter received frame number (or -1 if the frame is lost): 2

Receiver: Frame 2 received. Sending ACK 2

Sender: ACK 2 received. Proceeding to next frame.

Sender: Sending Frame 3

Receiver: Enter received frame number (or -1 if the frame is lost): -1

Receiver: Frame lost or out-of-order. No ACK sent.

Sender: No valid ACK received. Retransmitting Frame 3...

Sender: Sending Frame 3

Receiver: Enter received frame number (or -1 if the frame is lost): 3

Receiver: Frame 3 received. Sending ACK 3

Sender: ACK 3 received. Proceeding to next frame.

Sender: Sending Frame 4

Receiver: Enter received frame number (or -1 if the frame is lost): 4

Receiver: Frame 4 received. Sending ACK 4

Sender: ACK 4 received. Proceeding to next frame.

All frames sent and acknowledged!\

Experiment-9

Write a program for congestion control using leaky bucket algorithm.

Program:

```
#include <stdio.h>

int main() {
    int bucket_capacity, output_rate, n, i;
    int input_packets[50], current_bucket = 0;

    printf("Enter bucket capacity: ");
    scanf("%d", &bucket_capacity);
    printf("Enter output rate: ");
    scanf("%d", &output_rate);
    printf("Enter number of time intervals: ");
    scanf("%d", &n);

    printf("Enter number of packets arriving at each interval:\n");
    for(i = 0; i < n; i++) {
        scanf("%d", &input_packets[i]);
    }

    printf("\nTime\tPackets Incoming\tPackets Sent\tPackets Left\tPackets Dropped\n");
    for(i = 0; i < n; i++) {
        printf("%d\t\t%d\t", i+1, input_packets[i]);
        if(input_packets[i] + current_bucket > bucket_capacity) {
            // Incoming packets overflow bucket
            int dropped = (input_packets[i] + current_bucket) - bucket_capacity;
            current_bucket = bucket_capacity;
            printf("%d\t\t%d\t\t%d\n", output_rate, current_bucket-output_rate, dropped);
        } else {
            current_bucket += input_packets[i];
            printf("%d\t\t%d\t\t0\n",   output_rate,   (current_bucket-output_rate   >   0)   ?
current_bucket-output_rate : 0);
        }
        if(current_bucket < output_rate)
            current_bucket = 0;
        else
            current_bucket -= output_rate
```

```
else
        current_bucket -= output_rate;
   }
   return 0;
}
```

Output:

Enter bucket capacity: 10

Enter output rate: 5

Enter number of time intervals: 4

Enter number of packets arriving at each interval:

6 7 2 8

Time	Packets Incoming	Packets Sent	Packets Left	Packets Dropped
1	6	5	1	0
2	7	5	3	0
3	2	5	0	0
4	8	5	3	1

Experiment-10

Write a C program to implement Dijkstra‘s algorithm to compute the shortest path.

Program:

```
#include <stdio.h>
#define MAX 100
#define INFINITY 9999

void dijkstra(int n, int graph[MAX][MAX], int start) {
    int distance[MAX], visited[MAX], i, j, count, min_dist, next_node;

    // Initialization
    for (i = 0; i < n; i++) {
        distance[i] = graph[start][i];
        visited[i] = 0;
    }
    distance[start] = 0;
    visited[start] = 1;
    count = 1;

    while (count < n) {
        min_dist = INFINITY;
        next_node = -1;
        for (i = 0; i < n; i++) {
            if (!visited[i] && distance[i] < min_dist) {
                min_dist = distance[i];
                next_node = i;
```

```
            }
        }
        if (next_node == -1) break;

        visited[next_node] = 1;
        for (i = 0; i < n; i++) {
            if (!visited[i] && graph[next_node][i] != INFINITY &&
                distance[next_node] + graph[next_node][i] < distance[i]) {
                distance[i] = distance[next_node] + graph[next_node][i];
            }
        }
        count++;
    }
    // Output shortest distances
    printf("\nShortest distances from source vertex %d:\n", start);
    for (i = 0; i < n; i++) {
        printf("To vertex %d : %d\n", i, distance[i]);
    }
}
int main() {
    int n, graph[MAX][MAX], i, j, start;
    printf("Enter number of vertices: ");
    scanf("%d", &n);
    printf("Enter the adjacency matrix (use %d for infinity/no edge):\n", INFINITY);
    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
            scanf("%d", &graph[i][j]);
```

```
    printf("Enter the source vertex (0 to %d): ", n - 1);
    scanf("%d", &start);


    dijkstra(n, graph, start);
    return 0;
}
```

```
else
        current_bucket -= output_rate;
    }
    return 0;
}
```

Output:

Enter number of vertices: 4

Enter the adjacency matrix (use 9999 for infinity/no edge):

0 3 9999 7

8 0 2 9999

5 9999 0 1

2 9999 9999 0

Enter the source vertex (0 to 3): 0

Shortest distances from source vertex 0:

To vertex 0 : 0

To vertex 1 : 3

To vertex 2 : 5

To vertex 3 : 6

Note: Replace 9999 with actual large value for "no edge" or infinity.

Experiment-11

Write a Program to implement Distance vector routing algorithm by obtaining routing table at each node (Take an example subnet graph with weights indicating delay between nodes).

Program:

```
#include <stdio.h>

#define MAX 10

#define INF 9999

int main() {

    int nodes, i, j, k, count = 0;

    int distance[MAX][MAX], via[MAX][MAX], updated[MAX][MAX];


    printf("Enter the number of nodes: ");

    scanf("%d", &nodes);


    printf("Enter the cost/delay adjacency matrix (%d for no link/infinite delay):\n", INF);

    for(i = 0; i < nodes; i++) {

        for(j = 0; j < nodes; j++) {

            scanf("%d", &distance[i][j]);

            if(i != j && distance[i][j] == 0) distance[i][j] = INF;

            via[i][j] = j;

        }

    }


    // Distance Vector Algorithm

    do {

        count = 0;
```

```
        for(i = 0; i < nodes; i++) {
            for(j = 0; j < nodes; j++) {
                for(k = 0; k < nodes; k++) {
                    if(distance[i][j] > distance[i][k] + distance[k][j]) {
                        distance[i][j] = distance[i][k] + distance[k][j];
                        via[i][j] = k;
                        count++;
                    }
                }
            }
        }
    } while(count != 0);
    // Display routing tables
    for(i = 0; i < nodes; i++) {
        printf("\nRouting table for node %d:\n", i);
        printf("Destination\tNext Hop\tTotal Cost\n");
        for(j = 0; j < nodes; j++) {
            if (i == j)
                printf("%d\t\t-\t\t0\n", j);
            else
                printf("%d\t\t%d\t\t%d\n", j, via[i][j], distance[i][j]);
        }
    }
    return 0;
}
```

Output:

1. **Input: Enter the number of nodes (e.g., 4).**
2. **Matrix: Enter an adjacency matrix (use 0 for self, actual delay for link, 9999 for no link).**

- **Example for 4 nodes:**
- 0 3 9999 7
- 8 0 2 9999
- 5 9999 0 1
- 2 9999 9999 0

Routing table for node 0:

Destination	Next Hop	Total Cost
0	-	0
1	1	3
2	1	5
3	2	6

Routing table for node 1:

Destination	Next Hop	Total Cost
1	-	0
0	0	8
2	2	2
3	2	3

Experiment-12

Write a Program to implement Broadcast tree by taking subnet of hosts.

Program:

```
#include <stdio.h>
#define MAX 20
#define INF 9999

int main() {
    int n, i, j, u, v, min, total_cost = 0;
    int graph[MAX][MAX], visited[MAX] = {0}, edges = 0;

    printf("Enter the number of hosts (nodes): ");
    scanf("%d", &n);

    printf("Enter the adjacency matrix (cost 0 for self, %d for no direct link):\n", INF);
    for(i = 0; i < n; i++)
        for(j = 0; j < n; j++)
            scanf("%d", &graph[i][j]);

    visited[0] = 1; // Start from host 0

    printf("\nEdges in the broadcast (spanning) tree:\n");
    while(edges < n - 1) {
        min = INF;
        for(i = 0; i < n; i++) {
            if(visited[i]) {
```

```
                for(j = 0; j < n; j++) {
                    if(!visited[j] && graph[i][j] < min && graph[i][j] != 0) {
                        min = graph[i][j];
                        u = i;
                        v = j;
                    }
                }
            }
        }
        printf("Host %d - Host %d : Cost = %d\n", u, v, min);
        total_cost += min;
        visited[v] = 1;
        edges++;
    }
    printf("Total cost of Broadcast Tree: %d\n", total_cost);
    return 0;
}
```

Output:

Enter the number of hosts (nodes): 4

Enter the adjacency matrix (cost 0 for self, 9999 for no direct link):

0 2 9999 6

2 0 3 8

9999 3 0 5

6 8 5 0

Edges in the broadcast (spanning) tree:

Host 0 - Host 1 : Cost = 2

Host 1 - Host 2 : Cost = 3

Host 2 - Host 3 : Cost = 5

Total cost of Broadcast Tree: 10

Experiment-13

Wireshark

i. Packet Capture Using Wire shark
ii. Starting Wire shark
iii. Viewing Captured Traffic
iv. Analysis and Statistics & Filters.

i. Packet Capture Using Wireshark

Wireshark is a network packet analyzer used to capture live traffic on a computer network.

- Launch Wireshark and identify active network interfaces such as Ethernet or Wi-Fi from the home screen.
- Select the interface showing live traffic and click Start Capturing Packets (icon of a blue shark fin) or press Ctrl + E to begin capture.
- Wireshark collects frames and displays them in real time as a list in the Packet List Pane.

ii. Starting Wireshark

- Open Wireshark via Start Menu or Terminal.
- Choose your network interface card (NIC) that carries internet traffic (for example, Wi-Fi 3 or eth0).
- Click on Capture → Start, or use the keyboard shortcut Ctrl + E to begin recording packets.
- Let it run briefly and then click Capture → Stop to finish.
- Save your capture as .pcap file for further analysis.

iii. Viewing Captured Traffic

The Wireshark main window has three panes for inspecting packets:

1. Packet List Pane – Displays each captured packet with details such as number, time, source, destination, protocol, and length.
2. Packet Details Pane – Shows the hierarchical structure of selected packet protocols (Ethernet, IP, TCP, HTTP, etc.).

3. Packet Bytes Pane – Displays raw data (in hexadecimal or ASCII) of the selected packet for low-level analysis.

Clicking on any packet updates the other panes instantly, providing detail down to field-level structure in network headers.

iv. Analysis and Statistics & Filters

Wireshark provides analytical tools and filtering options for traffic examination:

- Display Filters limit visible packets using syntax like ip.addr == 192.168.1.5, http, or tcp.port == 80.
- Statistics Menu includes features:
  - Protocol Hierarchy: shows proportion of Ethernet, IP, TCP, and application-layer traffic.
  - Conversations: lists communication pairs between devices.
  - IO Graphs: visualizes traffic volume over time.
- Follow TCP Stream reconstructs full client-server communication for one session.
- Expert Info highlights retransmissions, delays, or malformed packets.

Experiment-14

How to run Nmap scan

Requirements

- Nmap installed on Linux, Windows, or Mac
- Terminal/Command Prompt access
- Target IP (example: scanme.nmap.org or a local IP like 192.168.1.10)

Step-by-Step Lab Program

1. Basic Host Scan

This command discovers which devices are active on the subnet

nmap -sn 192.168.1.0/24

- Lists live hosts within the subnet, skips port scan.

2. Simple Port Scan

Scan open ports on a target device.

nmap scanme.nmap.org

- Displays which common ports (top 1000) are open and corresponding services.

3. Stealth SYN Scan

Performs a half-open TCP SYN scan for stealthier port detection.

nmap -sS scanme.nmap.org

- Detects open, closed, and filtered TCP ports.

4. Service and Version Detection

Finds out which services are running and their versions.

nmap -sV scanme.nmap.org

- Advanced scan, useful for service enumeration.

5. OS Detection and Aggressive Scan

Performs a scan including OS fingerprinting and traceroute.

nmap -A scanme.nmap.org

- Detailed results: OS, services, open ports, traceroute.

6. Scan Specific Ports

Scan only selected ports (e.g., 22, 80, 443).

nmap -p 22,80,443 scanme.nmap.org

- Saves time if searching for specific services

7. UDP Port Scan

Scans UDP services; slower but sometimes needed.

sudo nmap -sU scanme.nmap.org

- Run with sudo for privileged access.

Sample Output

After running a scan, Nmap lists:

- PORT | STATE | SERVICE
- 22/tcp | open | ssh
- 80/tcp | open | http
- 443/tcp | closed | https

Experiment-15

Operating System Detection using Nmap

Requirements

- Nmap installed on your machine (Linux/Windows/Mac)
- Target IP address (can be local or internet-facing)
- Root/administrator privileges (important for OS detection on Unix/Linux)

Step-by-Step Lab Program

1. Basic OS Detection

Run the following command in your terminal/command prompt:

sudo nmap -O <target_ip_address>

- Replace <target_ip_address> with the IP of the target host.
- The -O flag enables OS detection.
- sudo is required for many OS-fingerprinting probes, especially on Unix/Linux

2. Aggressive OS Detection

Run a more comprehensive scan that combines OS detection, version detection, traceroute, and script scanning:

sudo nmap -A <target_ip_address>

- The -A flag enables aggressive scanning, which increases accuracy but can be more detectable by firewalls.

3. OS Detection with Guessing

If Nmap fails to identify the OS exactly, you can add a guessing option:

sudo nmap -O --osscan-guess <target_ip_address>

- This tries to make an educated guess when there's no perfect match.

Sample Output

Starting Nmap scan at 2025-10-16 12:00 IST

Nmap scan report for 192.168.1.100

OS details: Linux 3.x, Ubuntu; or Microsoft Windows 10

Network Distance: 1 hop

...

- The output shows probable OS names and versions, along with confidence levels.

Experiment- 16

Do the following using NS2 Simulator

i. NS2 Simulator-Introduction

ii. Simulate to Find the Number of Packets Dropped

iii. Simulate to Find the Number of Packets Dropped by TCP/UDP

iv. Simulate to Find the Number of Packets Dropped due to Congestion

i. NS2 Simulator - Introduction

- NS2 is an event-driven simulation tool for networking research.
- Simulations are written in Tcl scripts.
- Typical steps:
  1. Create Simulator object: **set ns [new Simulator]**
  2. Define nodes, links, agents (TCP/UDP), and applications (FTP/CBR).
  3. Specify trace and NAM output files.
  4. Execute events and run: **$ns run**
  5. Analyze trace file for results.

ii. Simulate to Find the Number of Packets Dropped

Sample Tcl Script:

```
set ns [new Simulator]

set nf [open out.nam w]

$ns namtrace-all $nf

set tf [open out.tr w]

$ns trace-all $tf


set n0 [$ns node]

set n1 [$ns node]

$ns duplex-link $n0 $n1 1Mb 10ms DropTail


set udp [new Agent/UDP]
```

```
$ns attach-agent $n0 $udp
set null [new Agent/Null]
$ns attach-agent $n1 $null
$ns connect $udp $null

set cbr [new Application/Traffic/CBR]
$cbr attach-agent $udp
$ns at 0.1 "$cbr start"
$ns at 4.5 "$cbr stop"
$ns at 5.0 "finish"

proc finish {} {
   global ns tf nf
   $ns flush-trace
   close $tf
   close $nf
   exec nam out.nam &
   exit 0
}
$ns run
```

- Run simulation: **ns script.tcl**
- Analyze packet drops: **grep "^d" out.tr | wc -l** (counts lines starting with "d")

iii. Simulate to Find the Number of Packets Dropped by TCP/UDP

- Use both TCP and UDP agents in one simulation.
- After running, filter trace like:
    - Packet drops for UDP: **grep "^d.*UDP" out.tr | wc -l**
    - Packet drops for TCP: **grep "^d.*TCP" out.tr | wc -l**
- The trace file marks transport type per event, use AWK or grep for protocol-specific counts.

iv. Simulate to Find the Number of Packets Dropped due to Congestion

- Create a bottleneck by connecting multiple sources to one receiver via a single link with limited bandwidth.
- Example analysis (after simulation):

TCL

```
# After simulation:

grep "^d" out.tr | wc -l                    # Total drops

grep "^d.*<bottleneck node number>" out.tr | wc -l   # Drops at congestion point
```

- In the Tcl script, set up several sources (n0, n1) sending to a single receiver (n2) over a shared, bandwidth-limited link.
- Analyze trace to find drops at congested nodes.

Trace analysis method:

- Look for "d" (drop) events in trace; congestion usually occurs at links with high incoming rate vs. low bandwidth.

Additional Lab Experiments

TCP Chat Server (Multiplexed using select())

1. **Develop client-server applications in C for**:

- TCP and UDP chat
- RTT (Round Trip Time) measurement
- Multiplexed server with simultaneous connections.

```
// tcp_chat_server.c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include <sys/select.h>

#define PORT 8080
#define MAX_CLIENTS 10

int main() {
    int master_socket, new_socket, client_socket[MAX_CLIENTS], activity, valread, sd, max_sd;
    struct sockaddr_in address;
    fd_set readfds;
    char buffer[1025];
```

```
char *welcome_msg = "Welcome to the TCP Chat Server!\n";
socklen_t addrlen;

for (int i = 0; i < MAX_CLIENTS; i++) client_socket[i] = 0;

master_socket = socket(AF_INET, SOCK_STREAM, 0);
int opt = 1;
setsockopt(master_socket, SOL_SOCKET, SO_REUSEADDR, (char*)&opt, sizeof(opt));
address.sin_family = AF_INET;
address.sin_addr.s_addr = INADDR_ANY;
address.sin_port = htons(PORT);

bind(master_socket, (struct sockaddr *)&address, sizeof(address));
listen(master_socket, 3);
printf("TCP Chat Server listening on port %d...\n", PORT);

while (1) {
   FD_ZERO(&readfds);
   FD_SET(master_socket, &readfds);
   max_sd = master_socket;

   for (int i = 0; i < MAX_CLIENTS; i++) {
      sd = client_socket[i];
      if (sd > 0) FD_SET(sd, &readfds);
      if (sd > max_sd) max_sd = sd;
   }
```

```
activity = select(max_sd + 1, &readfds, NULL, NULL, NULL);

if (FD_ISSET(master_socket, &readfds)) {
    new_socket = accept(master_socket, (struct sockaddr *)&address, &addrlen);
    send(new_socket, welcome_msg, strlen(welcome_msg), 0);

    for (int i = 0; i < MAX_CLIENTS; i++) {
        if (client_socket[i] == 0) {
            client_socket[i] = new_socket;
            break;
        }
    }
}

for (int i = 0; i < MAX_CLIENTS; i++) {
    sd = client_socket[i];
    if (FD_ISSET(sd, &readfds)) {
        valread = read(sd, buffer, 1024);
        if (valread == 0) {
            close(sd);
            client_socket[i] = 0;
        } else {
            buffer[valread] = '\0';
            printf("Client: %s", buffer);
            for (int j = 0; j < MAX_CLIENTS; j++) {
                if (client_socket[j] != 0 && j != i)
                    send(client_socket[j], buffer, strlen(buffer), 0);
```

```
                    }
                }
            }
        }
    }
    return 0;
}
```

TCP Chat Client

```
// tcp_chat_client.c
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#define PORT 8080
int main() {
    int sock = 0;
    struct sockaddr_in serv_addr;
    char message[1024], buffer[1024];
    sock = socket(AF_INET, SOCK_STREAM, 0);
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(PORT);
    inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr);
    connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr));
```

```
    printf("Connected to TCP chat server.\n");


    while (1) {
        fd_set readfds;
        FD_ZERO(&readfds);
        FD_SET(0, &readfds); // stdin
        FD_SET(sock, &readfds);
        int maxfd = sock + 1;
        select(maxfd, &readfds, NULL, NULL, NULL);
        if (FD_ISSET(0, &readfds)) {
            fgets(message, sizeof(message), stdin);
            send(sock, message, strlen(message), 0);
        }
        if (FD_ISSET(sock, &readfds)) {
            int valread = read(sock, buffer, sizeof(buffer));
            buffer[valread] = '\0';
            printf("Server: %s", buffer);
        }
    }
    return 0;
}
```

UDP Chat Application

```
// udp_chat.c
#include <stdio.h>
```

```c
#include <string.h>
#include <arpa/inet.h>
#include <unistd.h>
#define PORT 9090
int main() {
    int sock;
    struct sockaddr_in addr;
    char buffer[1024], message[1024];
    sock = socket(AF_INET, SOCK_DGRAM, 0);
    addr.sin_family = AF_INET;
    addr.sin_port = htons(PORT);
    addr.sin_addr.s_addr = INADDR_ANY;
    bind(sock, (struct sockaddr*)&addr, sizeof(addr));
    printf("UDP chat started (Ctrl+C to exit)\n");

    while (1) {
        socklen_t len = sizeof(addr);
        recvfrom(sock, buffer, sizeof(buffer), 0, (struct sockaddr*)&addr, &len);
        printf("Client: %s", buffer);
        printf("You: ");
        fgets(message, sizeof(message), stdin);
        sendto(sock, message, strlen(message), 0, (struct sockaddr*)&addr, len);
    }
    close(sock);
    return 0;
}
```

RTT(Round Trip Time) Measurement

```
// tcp_rtt.c
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <time.h>

#define PORT 7070

int main() {
    int sock;
    struct sockaddr_in server;
    char msg[] = "ping", buffer[1024];
    clock_t start, end;
    double rtt;

    sock = socket(AF_INET, SOCK_STREAM, 0);
    server.sin_family = AF_INET;
    server.sin_port = htons(PORT);
    inet_pton(AF_INET, "127.0.0.1", &server.sin_addr);
    connect(sock, (struct sockaddr *)&server, sizeof(server));

    start = clock();
    send(sock, msg, strlen(msg), 0);
    recv(sock, buffer, sizeof(buffer), 0);
```

```
    end = clock();
    rtt = ((double)(end - start)) / CLOCKS_PER_SEC * 1000;
    printf("Round Trip Time: %.3f ms\n", rtt);

    close(sock);
    return 0;
}
```

2. **<u>Implement Link State Routing Algorithm</u>**

Program:

```
#include <stdio.h>
#define INFINITY 9999
#define MAX 10
void dijkstra(int cost[MAX][MAX], int n, int src);
int main() {
    int n, src;
    int cost[MAX][MAX];
    printf("Enter number of nodes: ");
    scanf("%d", &n);
    printf("Enter the cost adjacency matrix (use 9999 for no direct link):\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            scanf("%d", &cost[i][j]);
            if (cost[i][j] == 0 && i != j)
                cost[i][j] = INFINITY;
        }
    }

printf("Enter the source node (0 to %d): ", n - 1);
    scanf("%d", &src);
    dijkstra(cost, n, src);
    return 0;
}
void dijkstra(int cost[MAX][MAX], int n, int src) {
    int dist[MAX], visited[MAX], pred[MAX];
```

```
  int count, min, nextNode;
  // Initialization
  for (int i = 0; i < n; i++) {
    dist[i] = cost[src][i];
    pred[i] = src;
    visited[i] = 0;
  }
  dist[src] = 0;
  visited[src] = 1;
  cout = 1;
  while (count < n - 1) {
    min = INFINITY;
    for (int i = 0; i < n; i++)
      if (dist[i] < min && !visited[i]) {
        min = dist[i];
        nextNode = i;
      }

visited[nextNode] = 1;
    for (int i = 0; i < n; i++)
      if (!visited[i])
        if (min + cost[nextNode][i] < dist[i]) {
          dist[i] = min + cost[nextNode][i];
          pred[i] = nextNode;
        }
      count++;
  }
```

```
    printf("\nRouting Table for Node %d:\n", src);
    for (int i = 0; i < n; i++) {
        if (i != src) {
            printf("Shortest distance to node %d = %d\n", i, dist[i]);
            printf("Path: %d", i);
            int j = i;
            while (j != src) {
                j = pred[j];
                printf("<-%d", j);
            }
            printf("\n");
        }
    }
}
```

Input:

Number of routers(nodes) and the cost of adjacency matrix

0	2	9999	1
2	0	3	9999
9999	3	0	4
1	9999	4	0

Choose the source node. The Program applies the Dijkstra’s algorithm to calculate the shortest path to all nodes.

Shortest distance to node 1 = 2

Path: 1<-0

Shortest distance to node 2 = 5

Path: 2<-1<-0

...