

COMPUTER NETWORKS LAB MANUAL (CSPC2207)

Institute: Government College of Engineering Kalahandi,
Bhawanipatna

Course Code: CSPC2207

Course Title: Computer Networks Lab

Branch: Computer Science & Engineering

Semester: 4th Sem.

Designed by: Dr. Rajendra Prasad Nayak
Assistant Professor



List of Experiments

1. Implement the data link layer framing methods such as character, character-stuffing and bit stuffing
2. Write a program to compute CRC code for the polynomials CRC-12, CRC-16 and CRC CCIP
3. Develop a simple data link layer that performs the flow control using the sliding window protocol, and loss recovery using the Go-Back-N mechanism
4. Take an example subnet of hosts and obtain a broadcast tree for the subnet.
5. Implement distance vector routing algorithm for obtaining routing tables at each node.
6. Implement data encryption and data decryption.
7. Write a program for congestion control using Leaky bucket algorithm
8. Write a program for frame sorting technique used in buffers.
9. Wireshark
 - Packet Capture Using Wire shark
 - Starting Wire shark
 - Viewing Captured Traffic
 - Analysis and Statistics & Filters.
10. Do the following using NS2 Simulator
 - NS2 Simulator-Introduction
 - Simulate to Find the Number of Packets Dropped
 - Simulate to Find the Number of Packets Dropped by TCP/UDP
 - Simulate to Find the Number of Packets Dropped due to Congestion
 - Simulate to Compare Data Rate& Throughput.
 - Simulate to Plot Congestion for Different Source/Destination
 - Simulate to Determine the Performance with respect to Transmission of Packets

Experiment 1: Implement the data link layer framing methods such as character count, character stuffing, and bit stuffing

1. Character Count Framing

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

void charCountFraming(char data[]) {
    int len = strlen(data);
    printf("Framed Data: %d%s\n", len, data);
}

int main() {
    char data[100];
    printf("Enter the data: ");
    scanf("%s", data);
    charCountFraming(data);
    return 0;
}
```

2. Character Stuffing

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

void charStuffing(char data[]) {
    char FLAG[] = "FLAG";
    char ESC[] = "ESC";
    char result[200] = "";
    char temp[200] = "";

    // Scan and stuff
    for (int i = 0; i < strlen(data); i++) {
        if (data[i] == 'F' && strncmp(&data[i], FLAG, 4) == 0) {
            strcat(temp, ESC);
            strcat(temp, FLAG);
            i += 3; // Skip the rest of FLAG
        } else if (data[i] == 'E' && strncmp(&data[i], ESC, 3) == 0) {
            strcat(temp, ESC);
            strcat(temp, ESC);
            i += 2; // Skip the rest of ESC
        } else {
            strncat(temp, &data[i], 1);
        }
    }
}
```

```

        // Add FLAG at start and end
        strcpy(result, FLAG);
        strcat(result, temp);
        strcat(result, FLAG);

        printf("Stuffed Frame: %s\n", result);
    }

int main() {
    char data[100];
    printf("Enter the data (use FLAG or ESC to test stuffing): ");
    scanf("%s", data);
    charStuffing(data);
    return 0;
}

```

3. Bit Stuffing

```

c
CopyEdit
#include <stdio.h>
#include <string.h>

void bitStuffing(char data[]) {
    char stuffed[200] = "";
    int count = 0;
    strcat(stuffed, "01111110"); // Start flag

    for (int i = 0; i < strlen(data); i++) {
        strncat(stuffed, &data[i], 1);
        if (data[i] == '1') {
            count++;
            if (count == 5) {
                strcat(stuffed, "0");
                count = 0;
            }
        } else {
            count = 0;
        }
    }

    strcat(stuffed, "01111110"); // End flag

    printf("Stuffed Data: %s\n", stuffed);
}

int main() {
    char data[150];
    printf("Enter the bit stream (only 0s and 1s): ");
    scanf("%s", data);
    bitStuffing(data);
    return 0;
}

```

✔Sample Outputs:

Character Count Framing:

Enter the data: HELLO
Framed Data: 5HELLO

Character Stuffing:

Enter the data (use FLAG or ESC to test stuffing): FLAGHELLOESC
Stuffed Frame: FLAGEScFLAGHELLOESCEScFLAG

Bit Stuffing:

Enter the bit stream: 01111101111110
Stuffed Data: 01111110011111001111100101111110

✔Result:

All three framing methods were implemented in **C language** and tested successfully.

Experiment 2: Write a program to compute CRC code for the polynomials CRC-12, CRC-16, and CRC-CCITT

Aim:

To implement a **C program** that computes the **Cyclic Redundancy Check (CRC)** code for the following generator polynomials:

1. **CRC-12:** $x^{12} + x^{11} + x^3 + x^2 + x + 1 \rightarrow 1100000001111$
 2. **CRC-16:** $x^{16} + x^{15} + x^2 + 1 \rightarrow 11000000000000101$
 3. **CRC-CCITT:** $x^{16} + x^{12} + x^5 + 1 \rightarrow 10001000000100001$
-

Theory:

CRC is an error-detecting code used in digital networks and storage devices. It treats the data as a binary number and divides it by a **generator polynomial** using modulo-2 division. The **remainder** becomes the CRC code.

Algorithm:

1. Take input data in binary.
 2. Append $n-1$ zeros to the data, where n is the length of the generator polynomial.
 3. Perform modulo-2 division (XOR).
 4. The remainder is the CRC code.
 5. Append the CRC to the original data for transmission.
-

C Program:

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

void xorOperation(char *data, char *key, int keylen) {
    for (int i = 0; i < keylen; i++) {
        data[i] = (data[i] == key[i]) ? '0' : '1';
    }
}

void crc(char *input, char *key) {
    int datalen = strlen(input);
    int keylen = strlen(key);
    char temp[128];
    char remainder[128];
    char appended_data[128];

    strcpy(temp, input);

    // Append n-1 zeros to data
    for (int i = 0; i < keylen - 1; i++) {
        temp[datalen + i] = '0';
    }
    temp[datalen + keylen - 1] = '\0';
    strcpy(appended_data, temp);

    for (int i = 0; i <= strlen(appended_data) - keylen; ) {
        for (int j = 0; j < keylen; j++) {
            remainder[j] = appended_data[i + j];
        }

        if (remainder[0] == '1')
            xorOperation(remainder, key, keylen);
        else
            xorOperation(remainder, "0000000000000000", keylen); // Dummy 0s

        // Replace bits in original data
        for (int j = 1; j < keylen; j++) {
            appended_data[i + j] = remainder[j];
        }

        // Move to next bit with '1'
        while (appended_data[i] != '1' && i < strlen(appended_data))
            i++;
    }

    // Extract the CRC from the last (keylen-1) bits
    printf("CRC: ");
    for (int i = strlen(input); i < strlen(input) + keylen - 1; i++) {
        printf("%c", appended_data[i]);
    }

    // Final transmitted message = data + CRC
    printf("\nTransmitted Message: %s", input);
    for (int i = strlen(input); i < strlen(input) + keylen - 1; i++) {
```

```

        printf("%c", appended_data[i]);
    }
    printf("\n\n");
}

int main() {
    char data[100];

    // CRC Polynomials
    char crc12[] = "1100000001111"; // CRC-12
    char crc16[] = "11000000000000101"; // CRC-16
    char crcCCITT[] = "10001000000100001"; // CRC-CCITT

    printf("Enter the binary data: ");
    scanf("%s", data);

    printf("\n--- CRC-12 ---\n");
    crc(data, crc12);

    printf("\n--- CRC-16 ---\n");
    crc(data, crc16);

    printf("\n--- CRC-CCITT ---\n");
    crc(data, crcCCITT);

    return 0;
}

```

Sample Output:

```

Enter the binary data: 11010011101100

--- CRC-12 ---
CRC: 100100110111
Transmitted Message: 11010011101100100100110111

--- CRC-16 ---
CRC: 0001011100001010
Transmitted Message: 110100111011000001011100001010

--- CRC-CCITT ---
CRC: 1010010111110110
Transmitted Message: 110100111011001010010111110110

```

✓Result:

CRC codes for CRC-12, CRC-16, and CRC-CCITT polynomials were successfully computed using modulo-2 division in C.

? Viva Questions:

1. What is CRC used for?
 2. How does modulo-2 division differ from normal division?
 3. Why are zeros appended to the data in CRC?
 4. Can CRC detect burst errors?
 5. What is the role of the generator polynomial?
-
-

Computer Network Lab Manual _ GCEK Bhawanipatna _ Dr. R. P. Nayak

Experiment 3: Develop a simple data link layer that performs flow control using Sliding Window Protocol and loss recovery using Go-Back-N

Aim:

To simulate a **Sliding Window Protocol** with **Go-Back-N ARQ (Automatic Repeat Request)** for flow control and loss recovery.

Theory:

- **Sliding Window Protocol** is a method of flow control in which the sender can send multiple frames before needing an acknowledgment.
 - **Go-Back-N ARQ** is an error control strategy where:
 - The sender sends multiple frames (up to a window size).
 - If an error/loss occurs in any frame, all subsequent frames are retransmitted starting from the lost frame.
-

Assumptions:

- Simulated transmission with no real network.
 - Manual input for lost frame.
 - Acknowledgment simulated.
-

Algorithm:

1. Sender sends N frames at a time.
 2. Receiver checks if a frame is lost or received.
 3. If a frame is lost, receiver discards that and all next frames.
 4. Sender resends from the lost frame.
-

C Program:

```
#include <stdio.h>

int main() {
    int frames[50], n, window, i, j, lost;

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

    for (i = 0; i < n; i++) {
        frames[i] = i;
    }

    printf("Enter window size: ");
    scanf("%d", &window);

    printf("Assume frames are numbered from 0 to %d\n", n - 1);
    printf("Enter the frame number that is lost (or -1 if none): ");
    scanf("%d", &lost);

    printf("\n--- Sender Side ---\n");
    for (i = 0; i < n; i += window) {
        printf("\nSending frames: ");
        for (j = i; j < i + window && j < n; j++) {
            printf("%d ", frames[j]);
        }

        printf("\n--- Receiver Side ---\n");
        for (j = i; j < i + window && j < n; j++) {
            if (frames[j] == lost) {
                printf("Frame %d lost. Sending NAK...\n", lost);
                printf("Sender resending from frame %d...\n", lost);
                i = lost - window; // Retransmit from lost frame in next loop
                break;
            } else {
                printf("Frame %d received and acknowledged.\n", frames[j]);
            }
        }

        if (lost == -1) {
            printf("All frames received correctly in this window.\n");
        }

        if (j != i + window) break; // Exit loop if retransmission required
    }

    printf("\nTransmission Complete.\n");
    return 0;
}
```

Sample Output:

```
Enter number of frames to send: 6
Enter window size: 3
Assume frames are numbered from 0 to 5
Enter the frame number that is lost (or -1 if none): 4
```

```
--- Sender Side ---
```

```
Sending frames: 0 1 2
```

```
--- Receiver Side ---
```

```
Frame 0 received and acknowledged.
```

```
Frame 1 received and acknowledged.
```

```
Frame 2 received and acknowledged.
```

```
All frames received correctly in this window.
```

```
Sending frames: 3 4 5
```

```
--- Receiver Side ---
```

```
Frame 3 received and acknowledged.
```

```
Frame 4 lost. Sending NAK...
```

```
Sender resending from frame 4...
```

```
Sending frames: 4 5
```

```
--- Receiver Side ---
```

```
Frame 4 received and acknowledged.
```

```
Frame 5 received and acknowledged.
```

```
All frames received correctly in this window.
```

```
Transmission Complete.
```

✔Result:

Successfully simulated **Sliding Window Protocol with Go-Back-N** mechanism using C.

? Viva Questions:

1. What is the purpose of the sliding window?
 2. How does Go-Back-N handle errors?
 3. What happens when a frame is lost in Go-Back-N?
 4. Compare Go-Back-N and Selective Repeat.
 5. What are the pros and cons of Go-Back-N?
-
-

Experiment 4: Take an example subnet of hosts and obtain a broadcast tree for the subnet

Aim:

To construct a **broadcast tree** (also called a spanning tree) from a given subnet graph representing connected hosts/nodes.

Theory:

- A **Broadcast Tree** ensures that a message originating from one node reaches all others without loops or duplication.
 - It is commonly represented by a **Minimum Spanning Tree (MST)**.
 - MST connects all nodes with the minimum total weight (distance/cost) and no cycles.
 - Algorithms like **Prim's** or **Kruskal's** are used to find MSTs in weighted graphs.
-

Algorithm (Prim's Algorithm for MST):

1. Start from any arbitrary node.
 2. At each step, add the smallest-weight edge that connects a new node to the tree.
 3. Repeat until all nodes are included.
-

C Program using Prim's Algorithm:

```
#include <stdio.h>
#define INF 9999
#define MAX 10

int main() {
    int cost[MAX][MAX], visited[MAX], num_nodes, i, j, edges = 0;
    int min_cost = 0, u, v, min;

    printf("Enter number of nodes in the subnet: ");
    scanf("%d", &num_nodes);

    printf("Enter the adjacency matrix (enter 9999 if no edge):\n");
    for (i = 0; i < num_nodes; i++) {
        for (j = 0; j < num_nodes; j++) {
            scanf("%d", &cost[i][j]);
        }
    }
}
```

```

        visited[i] = 0;
    }

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

    printf("\nBroadcast Tree Edges (Minimum Spanning Tree):\n");

    while (edges < num_nodes - 1) {
        min = INF;

        for (i = 0; i < num_nodes; i++) {
            if (visited[i]) {
                for (j = 0; j < num_nodes; j++) {
                    if (!visited[j] && cost[i][j] < min) {
                        min = cost[i][j];
                        u = i;
                        v = j;
                    }
                }
            }
        }

        printf("Edge %d: (%d - %d) Cost: %d\n", edges + 1, u, v, min);
        visited[v] = 1;
        min_cost += min;
        edges++;
    }

    printf("\nTotal Cost of Broadcast Tree: %d\n", min_cost);

    return 0;
}

```

Sample Input:

```

Enter number of nodes in the subnet: 4
Enter the adjacency matrix:
0 2 9999 6
2 0 3 8
9999 3 0 1
6 8 1 0

```

Sample Output:

Broadcast Tree Edges (Minimum Spanning Tree):

```

Edge 1: (0 - 1) Cost: 2
Edge 2: (1 - 2) Cost: 3
Edge 3: (2 - 3) Cost: 1

```

Total Cost of Broadcast Tree: 6

✔Result:

Successfully generated the **Broadcast Tree** using **Prim's Algorithm** for the given subnet.

? Viva Questions:

1. What is a broadcast tree?
 2. What is the difference between spanning tree and minimum spanning tree?
 3. Why do we need MST for broadcasting?
 4. What is the time complexity of Prim's algorithm?
 5. Can a subnet have more than one MST?
-

Experiment 5: Implement Distance Vector Routing Algorithm for obtaining routing tables at each node

Aim:

To simulate the **Distance Vector Routing Algorithm** to compute routing tables for each node in a subnet.

Theory:

- Distance Vector Routing is a **dynamic routing algorithm**.
 - Each router maintains a **vector (table)** of minimum distances to every other router.
 - Routers exchange this vector periodically with neighbors.
 - The **Bellman-Ford algorithm** is used to update the vectors based on neighbors' vectors.
-

Key Concepts:

- Each router assumes cost to all other nodes as ∞ except to itself (0).
 - Routers update their tables after comparing with neighbors.
 - Update continues until no change occurs in any table (i.e., convergence).
-

C Program: Distance Vector Routing

```
#include <stdio.h>
#define INFINITY 9999
#define MAX 10

int main() {
    int cost[MAX][MAX], dist[MAX][MAX], next_hop[MAX][MAX];
    int nodes, i, j, k, updated;

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

    printf("Enter the cost matrix (9999 for infinity):\n");
    for (i = 0; i < nodes; i++) {
        for (j = 0; j < nodes; j++) {
            scanf("%d", &cost[i][j]);
            if (i == j)
                dist[i][j] = 0;
        }
    }
}
```

```

        else
            dist[i][j] = cost[i][j];

        if (cost[i][j] != INFINITY && i != j)
            next_hop[i][j] = j;
        else
            next_hop[i][j] = -1;
    }
}

// Distance Vector Algorithm (Bellman-Ford)
do {
    updated = 0;
    for (i = 0; i < nodes; i++) {
        for (j = 0; j < nodes; j++) {
            for (k = 0; k < nodes; k++) {
                if (dist[i][j] > cost[i][k] + dist[k][j]) {
                    dist[i][j] = cost[i][k] + dist[k][j];
                    next_hop[i][j] = k;
                    updated = 1;
                }
            }
        }
    }
} while (updated);

// Display final routing tables
for (i = 0; i < nodes; i++) {
    printf("\nRouting table for Router %d:\n", i);
    printf("Destination\tNext Hop\tDistance\n");
    for (j = 0; j < nodes; j++) {
        if (i != j) {
            printf("%d\t\t%d\t\t%d\n", j, next_hop[i][j], dist[i][j]);
        }
    }
}

return 0;
}

```

Sample Input:

```

Enter the number of nodes: 3
Enter the cost matrix:
0 2 7
2 0 1
7 1 0

```

Sample Output:

```

Routing table for Router 0:
Destination      Next Hop      Distance
1                 1              2
2                 1              3

```

Routing table for Router 1:

Destination	Next Hop	Distance
0	0	2
2	2	1

Routing table for Router 2:

Destination	Next Hop	Distance
0	1	3
1	1	1

✔Result:

Successfully implemented **Distance Vector Routing Algorithm** and generated routing tables for all routers in the subnet.

? Viva Questions:

1. What is the difference between distance vector and link state routing?
 2. What is the role of the Bellman-Ford algorithm in distance vector routing?
 3. What causes the count-to-infinity problem?
 4. How often are routing tables exchanged in DV routing?
 5. Can distance vector routing be used in large networks?
-

Experiment 6: Implement Data Encryption and Data Decryption

Aim:

To implement **basic encryption and decryption** techniques in C using:

- **Caesar Cipher** (for alphabetic text)
 - **XOR Cipher** (for binary/text data)
-

Theory:

1. Caesar Cipher:

- A substitution cipher that shifts characters by a fixed key.
- Encryption: $\text{cipher}[i] = (\text{plain}[i] + \text{key}) \% 26$
- Decryption: $\text{plain}[i] = (\text{cipher}[i] - \text{key} + 26) \% 26$

2. XOR Cipher:

- Uses XOR operation with a key.
 - Same function for encryption and decryption.
 - Encryption/Decryption: $\text{cipher}[i] = \text{plain}[i] \wedge \text{key}$
-

C Program (Both Ciphers Included)

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>

void caesarEncrypt(char text[], int key) {
    char ch;
    printf("Encrypted (Caesar): ");
    for (int i = 0; text[i] != '\0'; ++i) {
        ch = text[i];
        if (isalpha(ch)) {
            char base = isupper(ch) ? 'A' : 'a';
            ch = (ch - base + key) % 26 + base;
        }
        printf("%c", ch);
    }
    printf("\n");
}
```

```

void caesarDecrypt(char text[], int key) {
    char ch;
    printf("Decrypted (Caesar): ");
    for (int i = 0; text[i] != '\0'; ++i) {
        ch = text[i];
        if (isalpha(ch)) {
            char base = isupper(ch) ? 'A' : 'a';
            ch = (ch - base - key + 26) % 26 + base;
        }
        printf("%c", ch);
    }
    printf("\n");
}

void xorCipher(char text[], char key) {
    printf("Output (XOR): ");
    for (int i = 0; text[i] != '\0'; ++i) {
        printf("%c", text[i] ^ key);
    }
    printf("\n");
}

int main() {
    char text[100];
    int choice, caesarKey;
    char xorKey;

    printf("Enter the text: ");
    scanf("%[^\n]", text); // Allow spaces

    printf("\nChoose:\n1. Caesar Cipher\n2. XOR Cipher\nChoice: ");
    scanf("%d", &choice);

    if (choice == 1) {
        printf("Enter Caesar key (shift): ");
        scanf("%d", &caesarKey);
        caesarEncrypt(text, caesarKey);
        caesarDecrypt(text, caesarKey);
    } else if (choice == 2) {
        printf("Enter XOR key (any character): ");
        scanf("%c", &xorKey);
        xorCipher(text, xorKey); // Encryption
        xorCipher(text, xorKey); // Decryption
    } else {
        printf("Invalid choice.\n");
    }

    return 0;
}

```

Sample Output:

Enter the text: HELLO

Choose:

```
1. Caesar Cipher
2. XOR Cipher
Choice: 1
Enter Caesar key (shift): 3
```

```
Encrypted (Caesar): KHOOR
Decrypted (Caesar): HELLO
vbnet
CopyEdit
Enter the text: HELLO
```

```
Choose:
1. Caesar Cipher
2. XOR Cipher
Choice: 2
Enter XOR key (any character): A
```

```
Output (XOR): )$-,
Output (XOR): HELLO
```

✔Result:

Successfully implemented **encryption and decryption** using **Caesar Cipher** and **XOR Cipher** in C.

? Viva Questions:

1. What is the Caesar cipher and how does it work?
 2. How is XOR cipher different from substitution ciphers?
 3. Is Caesar cipher secure? Why or why not?
 4. What is the benefit of XOR cipher in binary communication?
 5. How can keys be shared securely in encryption?
-

Experiment 7: Write a program for congestion control using the Leaky Bucket Algorithm

Aim:

To implement the **Leaky Bucket algorithm** for **congestion control** in computer networks.

Theory:

- **Leaky Bucket** is a traffic shaping and congestion control mechanism.
 - It regulates data transmission by allowing packets to flow out of the bucket at a constant rate, regardless of bursty arrivals.
 - If incoming packets exceed the bucket capacity, they are discarded (dropped).
-

Parameters:

- **Bucket Size:** Max capacity (in packets or bytes)
 - **Incoming Packet Size:** User-supplied list
 - **Output Rate:** Constant rate of packet transmission per time unit
-

Algorithm:

1. Initialize bucket capacity and output rate.
 2. For each time unit:
 - Add incoming packet to the bucket if space allows.
 - Transmit packets from the bucket up to the output rate.
 - Drop packets if overflow occurs.
-

C Program: Leaky Bucket Simulation

```
#include <stdio.h>

#define MAX 50

int main() {
    int n, bucket_size, output_rate, input_packets[MAX];
    int i, stored = 0, dropped = 0;

    printf("Enter number of time units: ");
    scanf("%d", &n);

    printf("Enter incoming packet size at each unit:\n");
    for (i = 0; i < n; i++) {
        printf("Time %d: ", i + 1);
        scanf("%d", &input_packets[i]);
    }

    printf("Enter bucket size: ");
    scanf("%d", &bucket_size);

    printf("Enter output rate: ");
    scanf("%d", &output_rate);

    printf("\nTime\tIncoming\tStored\tDropped\tTransmitted\n");

    for (i = 0; i < n; i++) {
        printf("%d\t", i + 1);
        printf("%d\t\t", input_packets[i]);

        // Add incoming to stored
        if (input_packets[i] + stored <= bucket_size) {
            stored += input_packets[i];
            dropped = 0;
        } else {
            dropped = (input_packets[i] + stored) - bucket_size;
            stored = bucket_size;
        }

        // Transmit up to output rate
        int transmitted = (stored >= output_rate) ? output_rate : stored;
        stored -= transmitted;

        printf("%d\t%d\t%d\n", stored, dropped, transmitted);
    }

    return 0;
}
```

Sample Input & Output:

```
Enter number of time units: 5
Enter incoming packet size at each unit:
Time 1: 4
Time 2: 7
Time 3: 2
Time 4: 6
Time 5: 5
Enter bucket size: 10
Enter output rate: 4
```

Output:

Time	Incoming	Stored	Dropped	Transmitted
1	4	0	0	4
2	7	3	1	4
3	2	1	0	4
4	6	3	0	4
5	5	4	0	4

✓Result:

Implemented the **Leaky Bucket algorithm** successfully to simulate congestion control.

? Viva Questions:

1. What is the purpose of the leaky bucket algorithm?
 2. How is it different from the token bucket algorithm?
 3. What happens if the input rate exceeds the bucket size?
 4. Can this algorithm prevent congestion in bursty traffic?
 5. How is fairness achieved using leaky bucket?
-

Experiment 8: Write a program for frame sorting technique used in buffers

Aim:

To simulate **frame sorting in a buffer**, where frames arrive **out of order** and must be sorted before processing.

Theory:

- In network communication, **frames may arrive out of order** due to differing paths, delays, or transmission errors.
 - A **frame buffer** temporarily stores these frames.
 - The system **sorts the buffered frames** by their sequence numbers before delivering them to the application layer.
-

Example:

- Received out-of-order frames: [3, 1, 2]
 - Sorted output: [1, 2, 3]
-

Assumptions:

- Each frame has a unique **sequence number**.
 - Sorting is based on this sequence number (simple integer).
-

C Program: Frame Sorting in Buffers

```
#include <stdio.h>

void sortFrames(int frames[], int n) {
    int temp;
    // Bubble Sort for simplicity
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (frames[j] > frames[j + 1]) {
                temp = frames[j];
                frames[j] = frames[j + 1];
                frames[j + 1] = temp;
            }
        }
    }
}

int main() {
    int frames[50], n, i;

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

    printf("Enter the sequence numbers of frames (out of order):\n");
    for (i = 0; i < n; i++) {
        scanf("%d", &frames[i]);
    }

    printf("\nFrames in buffer (unsorted):\n");
    for (i = 0; i < n; i++) {
        printf("%d ", frames[i]);
    }

    sortFrames(frames, n);

    printf("\n\nFrames after sorting:\n");
    for (i = 0; i < n; i++) {
        printf("%d ", frames[i]);
    }

    printf("\n\nFrames delivered in order.\n");

    return 0;
}
```

Sample Output:

```
Enter the number of frames received: 5
Enter the sequence numbers of frames (out of order):
3 1 5 2 4

Frames in buffer (unsorted):
3 1 5 2 4
```

Frames after sorting:
1 2 3 4 5
Frames delivered in order.

✓Result:

Simulated and sorted the **received out-of-order frames** using a buffer in C.

? Viva Questions:

1. Why do frames arrive out of order in a network?
 2. What is the role of sequence numbers?
 3. What happens if a frame is missing?
 4. What data structure is ideal for buffering frames?
 5. How can sorting delay affect real-time applications?
-

Experiment 9: Wireshark – Packet Capture and Analysis

Aim:

To use **Wireshark**, a network protocol analyzer, to capture, inspect, and analyze network traffic in real-time.

Objectives:

- Start and stop a packet capture session
 - Analyze the captured packets
 - Use filters to inspect specific traffic
 - View protocol statistics and detailed packet breakdowns
-

Tools Required:

- A computer with **Wireshark** installed
 - Internet/network connection
 - Administrator privileges
-

Theory:

Wireshark is a free and open-source packet analyzer. It captures live packets from the network interface and allows analysis of:

- Protocols (TCP, UDP, HTTP, etc.)
 - Headers (IP, MAC, TCP sequence numbers)
 - Payload data
 - Communication between hosts
-

Steps:

➤1. Starting Wireshark:

- Open **Wireshark**
- Select a network interface (e.g., Wi-Fi, Ethernet)
- Click **Start Capture**

➤2. Capturing Traffic:

- Perform some network activity (open a website, ping a server)
- Let Wireshark capture traffic for a few seconds
- Click **Stop Capture** (red square button)

➤3. Viewing Captured Packets:

Each packet has:

- **No.:** Serial number of the packet
- **Time:** Time since capture started
- **Source/Destination:** IP or MAC addresses
- **Protocol:** The protocol used (TCP, UDP, HTTP, etc.)
- **Info:** Summary of the packet content

➤4. Detailed Analysis:

Click any packet to inspect:

- **Frame Info:** Overall packet size and interface
- **Ethernet Header**
- **IP Header**
- **TCP/UDP Details**
- **Application layer data** (e.g., HTTP GET requests)

➤5. Using Filters:

Filter	Description
http	Show only HTTP traffic
ip.addr == 192.168.1.1	Filter packets with a specific IP
tcp.port == 80	Filter packets on port 80
icmp	Show only ICMP packets (e.g., ping)
dns	Show DNS traffic

Use the **filter bar** at the top and press **Enter**.

➤6. Protocol Hierarchy and Statistics:

- Go to Statistics > Protocol Hierarchy
 - See which protocols were captured and their relative sizes
 - Use Statistics > Conversations to see connections between IPs
-

Screenshots to Include in Lab Record:

1. Wireshark interface with live capture
 2. Detailed breakdown of a selected packet
 3. Filter applied (e.g., http)
 4. Protocol hierarchy view
-

✓Result:

Successfully captured and analyzed live network traffic using **Wireshark**, including the use of filters and protocol statistics.

? Viva Questions:

1. What is Wireshark used for?
 2. How does packet capturing work?
 3. Can you capture encrypted data with Wireshark?
 4. What is the difference between TCP and UDP traffic in Wireshark?
 5. How are filters applied in Wireshark?
-

Experiment 10: NS2 Simulator – Network Simulation Tasks

Aim:

To use the NS2 (Network Simulator 2) to simulate network behavior and analyze:

- Packet drops
 - Throughput
 - Congestion
 - TCP vs UDP performance
-

Tools Required:

- Ubuntu/Linux system
 - NS2 installed (ns, nam, xgraph tools)
-

Theory:

NS2 is a discrete event network simulator used for:

- Simulating wired/wireless networks
 - Studying TCP/UDP protocols
 - Visualizing network behavior (via NAM)
 - Graphing performance metrics (via XGraph)
-

Setup Steps:

➤1. Basic TCL Script Structure

```
# Create Simulator object
set ns [new Simulator]

# Open trace and NAM files
set nf [open out.nam w]
$ns namtrace-all $nf
```

```
set tf [open out.tr w]
$ns trace-all $tf

# Create nodes
set n0 [$ns node]
set n1 [$ns node]

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

# Set up TCP connection
set tcp [new Agent/TCP]
$ns attach-agent $n0 $tcp

set sink [new Agent/TCPSink]
$ns attach-agent $n1 $sink
$ns connect $tcp $sink

# Set up traffic
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ftp start

# Define finish
$ns at 5.0 "finish"
proc finish {} {
    global ns nf tf
    $ns flush-trace
    close $nf
    close $tf
    exec nam out.nam &
    exit 0
}

# Run simulation
$ns run
```

Sub-Experiments:

✓1. Simulate to Find the Number of Packets Dropped

- Add a queue with `DropTail`
- Analyze `out.tr` for lines with `d` (drop)

Command:

```
grep "^d" out.tr | wc -l
```

✓2. Simulate to Find Packets Dropped by TCP/UDP

- Use both TCP and UDP flows in script
- Filter trace with:

```
bash
CopyEdit
grep "^d.*tcp" out.tr | wc -l
grep "^d.*udp" out.tr | wc -l
```

✔3. Simulate Packet Drops due to Congestion

- Use low bandwidth/high load links
 - Observe high number of drops in DropTail or RED queue
-

✔4. Simulate to Compare Data Rate & Throughput

- $\text{Throughput} = (\text{Total Bytes Received} * 8) / \text{Total Simulation Time}$
 - Parse `out.tr` for received packets at sink
-

✔5. Simulate Congestion for Different Source/Destination

- Add 3 or more flows with overlapping paths
 - Monitor queue and drop behavior
-

✔6. Simulate Performance Based on Packet Transmission

- Vary packet size, interval, link bandwidth
 - Evaluate output using:
 - NAM animations
 - Throughput calculation
-

Result Summary Format:

Metric	Value
Total Packets Sent	2000
Total Packets Received	1900
Total Packets Dropped	100
Throughput (kbps)	760 kbps

Screenshots to Include in Record:

1. NAM animation (packet flow and drops)
 2. Output of `out.tr` analysis (grep commands)
 3. Throughput/Drop Rate Graph using `xgraph`
-

✓Result:

Successfully simulated various **network behaviors using NS2**, including:

- Packet drops
 - Congestion
 - TCP vs UDP performance
 - Throughput evaluation
-

? Viva Questions:

1. What is NS2 and what is it used for?
2. How does NS2 differ from real-world testing?
3. How is packet drop detected in trace files?
4. What is the role of `nam` and `xgraph`?
5. How do you simulate congestion in NS2?