ADITYA

COLLEGE OF ENGINEERING & TECHNOLOGY

AN AUTONOMOUS INSTITUTION

ADB Road, Surampalem. Kakinada.Dist., (A.P.)

Department of

.....

Name: ..... PIN No. | | | | | | | | | | |

Certified that this is the bonafide record of practical work done by

Mr./Ms. .....

a student of ..... ***with PIN No.*** .....

in the ..... ***Laboratory during the year*** .....

No. of Practicals Conducted [ ] No. of Practicals Attended [ ]

Signature - of the Incharge Signature - Head of the Department

Submitted for the practical examination held on .....

Examiner-1 Examiner-2

ADITYA COLLEGE OF ENGINEERING & TECHNOLOGY(A)
(An AUTONOMOUS Institution)
Approved by AICTE, New Delhi * Permanently Affiliated to JNTUK, Kakinada
Accredited by **NBA*** Accredited by **NAAC A+** Grade with CGPA of 3.40
Recognized by UGC Under Sections 2(f) and 12(B) of the UGC Act, 1956
Aditya Nagar, ADB Road, Surampalem, Gandepalli Mandal, Kakinada District - 533437, A.P
Ph. 99591 76665, Email: office@acet.ac.in, www.acet.ac.in

ADITYA COLLEGE OF ENGINEERING AND TECHNOLOGY

DEPARTMENT of CSE-

ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

COMPUTER NETWORKS LAB LAB

MANUAL

Class : III Year II Semester

Branch : CSE-AI&ML

Prepared By : K. Rojarani, M.Tech (PhD),Assistant Professor.

Academic Year: 2024-25

Pointer

S No	Date	Name of the Experiment	Page No	Remark

Pointer

S No	Date	Name of the Experiment	Page No	Remark

Institute Vision:
To induce higher planes of learning by imparting technical education with

- International standards
- Applied research
- Creative Ability
- Value based instruction and

to emerge as a premiere institute.

Institute Mission:

Achieving academic excellence by providing globally acceptable technical education by forecasting technology through

- Innovative Research And development
- Industry Institute Interaction
- Empowered Manpower

Department Vision:

To be a recognized Computer Science and Engineering hub striving to meet the growing needs of the Industry and Society.

Department Mission:

M1:
Imparting Quality Education through state-of-the-art infrastructure with industry collaboration.

M2:
Enhance Teaching Learning Process to disseminate knowledge.

M3:
Organize Skill based, Industrial and Societal Events for overall Development.

Program Educational Outcomes (PEO's):

PEO 1: Proficiency in AI & ML Applications

Expertise in designing, developing, and deploying AI and ML solutions for healthcare, finance, and autonomous system problems.

PEO 2: Lifelong Learning and Adaptability

Adapt to evolving technologies and industry trends in AI and ML through continuous learning and self-improvement.

PEO 3: Effective Communication and Collaboration

Possess strong communication and teamwork skills, enabling them to effectively collaborate with interdisciplinary teams and solve complex problems using AI and ML concepts.

Program Specific Outcome (PSO's):

PSO 1: AI and ML System Development

Design, develop, and implement AI and ML systems by applying fundamental principles, algorithms, and techniques.

PSO 2: Data-driven Decision-Making

Collect, preprocess, and analyze large and diverse datasets to extract meaningful insights using AI and ML techniques to support data-driven decision-making processes in various domains.

Program Outcomes (PO's):

1. Engineering Knowledge:
Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

2. Problem Analysis:

Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

3. Design/development of solutions:

Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

4. Conduct investigations of complex problems:

Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

5. Modern tool usage:

Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

6. The engineer and society:

Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice. **7. Environment and sustainability:**

Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

8. Ethics:

Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

9. Individual and team work:

Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

10. Communication:

Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write

effective reports and design documentation, make effective presentations, and give and receive clear instructions.

11. Project management and finance:

Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

12. Life-long learning:

Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Course Outcomes:

CO#	Course Outcomes	Blooms Taxonomy level
C321.1	Summarize the fundamental protocols and types of various computer networks.	Understand
C321.2	Demonstrate data link layer concepts and its protocols in computer networks	Understand
C321.3	Illustrate routing algorithms at network layer in various networking environments	Apply
C321.4	Demonstrate different transport layer protocols	Apply
C321.5	Illustrate various Application Layer protocols and applications in diverse networking environments.	Apply
C321.1	Summarize the fundamental protocols and types of various computer networks.	Understand

List of Experiments:

1. Study of Network devices in detail and connect the computers in Local Area Network.
2. Write a Program to implement the data link layer farming methods such as
i) Character stuffing ii) bit stuffing.
3. Write a Program to implement data link layer farming method checksum.
4. Write a program for Hamming Code generation for error detection and correction.
5. Write a Program to implement on a data set of characters the three CRC polynomials CRC 12, CRC 16 and CRC CCIP.
6. Write a Program to implement Sliding window protocol for Goback N.
7. Write a Program to implement Sliding window protocol for Selective repeat.
8. Write a Program to implement Stop and Wait Protocol.
9. Write a program for congestion control using leaky bucket algorithm
10. Write a Program to implement Dijkstra's algorithm to find shortest path through the graph.
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).

EXPERIMENT -1

Study of Network Devices in Detail and Connecting Computers in a Local Area Network (LAN)

1. Introduction to Network Devices

Network devices are essential components that enable communication between computers and other devices in a network. In a **Local Area Network (LAN)**, various devices work together to ensure data transmission and connectivity. Below are some commonly used network devices and their functions:

---

2. Essential Network Devices

a) Router:

- Function: Connects multiple networks and directs data packets to their destination.
- Features: Dynamic routing, Network Address Translation (NAT), and firewall capabilities.
- Use Case: Connecting a LAN to the internet.

b) Switch:

- Function: Connects multiple devices within a LAN and forwards data only to the intended recipient.
- Features: VLAN support, MAC address learning, and high-speed data transfer.
- Use Case: Interconnecting computers, printers, and servers within a network.

c) Hub:

- Function: Broadcasts data to all connected devices without filtering.
- Features: Simple, inexpensive, and basic connectivity.
- Use Case: Small or less complex networks.

d) Modem (Modulator-Demodulator):

- Function: Converts digital signals to analog for transmission over phone lines and vice versa.
- Features: DSL, Cable, and Fiber modems.
- Use Case: Connecting a LAN to the internet through an ISP.

e) Access Point (AP):Function: Extends wireless coverage by connecting Wi-Fi devices to a wired LAN.

- Features: Dual-band support, security protocols (WPA3).
- Use Case: Creating wireless networks within a LAN.

f) Network Interface Card (NIC):

- Function: Connects a computer to a network using wired (Ethernet) or wireless (Wi-Fi) methods.
- Features: Unique MAC address and Ethernet port.
- Use Case: Physical network connection on desktops and laptops.

---

3. Connecting Computers in a Local Area Network

(LAN) Step 1: Gather Required Devices and Cables

- Router or Switch
- Ethernet cables (Cat5e or Cat6)
- Network Interface Cards (NIC)
- Wireless Access Point (optional)

Step 2: Physical Connection

- Connect the switch to the router using an Ethernet cable.
- Connect each computer to the switch using Ethernet cables.
- Optionally, connect a wireless access point to the switch.

Step 3: Configure IP Addresses

- Use **Dynamic Host Configuration Protocol (DHCP)** on the router to assign IP addresses automatically.
- Alternatively, assign **Static IPAddresses** manually within the same subnet.

Step 4: Network Configuration

- Ensure that all computers are set to the same **IP subnet** (e.g., 192.168.1.x).
- Configure the **Gateway and DNS Server** as the router's IP address.

Step 5: Enable File and Printer Sharing

- On Windows:
  - Go to **Network and Sharing Center**.
  - Enable **Network Discovery** and **File and Printer Sharing**.
- On Linux:

- Use **Samba** to share files across the network.

Step 6: Testing the Network

- Use the **ping** command to check connectivity:
- ping 192.168.1.2
- Test file sharing and printing between connected devices.

EXPERIMENT -2

Write a Program to implement the data link layer farming methods such as

i) Character stuffing ii) bit stuffing.

Character stuffing: The framing method gets around the problem of resynchronization after an error by having each frame start with the ASCII character sequence DLE STX and the sequence DLE ETX. If the destination ever losses the track of the frame boundaries all it has to do is look for DLESTX or DLE ETX characters to figure out. The data link layer on the receiving end removes the DLE before the data are given to the network layer. This technique is called character stuffing.

PROGRAM:

```
#include<stdio.h>
#include<string.h>
void main()
{
        int a[20],b[30],i,j,k,count,n;
        printf("Enter frame length:");
        scanf("%d",&n);
        printf("Enter input frame (0's & 1's only):");
        for(i=0;i<n;i++)
                scanf("%d",&a[i]);
                i=0; count=1; j=0;
                while(i<n)
                {
                if(a[i]==1)
                        {
                        b[j]=a[i];
                                for(k=i+1;a[k]==1 && k<n && count<5;k++)
                                {
                                j++;
                                b[j]=a[k]; count++; if(count==5)
                                        {
                                        j++;
```

```
                                        b[j]=0;
                                    }
                                    i=k;
                                }
                        }
                        else
                        {

                                b[j]=a[i];
                        }
                        i++;
                        j++;
                }

                printf("After stuffing the frame is:");
                for(i=0;i<j;i++)
                printf("%d",b[i]);
}
```

OUTPUT:

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

PROGRAM:

```
#include<stdio.h>
#include<string.h>
main()
{
    char a[30], fs[50] = " ", t[3], sd, ed, x[3], s[3], d[3], y[3];
     int i, j, p = 0, q = 0;
     clrscr();
     printf("Enter characters to be stuffed:");
     scanf("%s", a);
     printf("\nEnter a character that represents starting delimiter:");
     scanf(" %c", &sd);
     printf("\nEnter a character that represents ending delimiter:");
     scanf(" %c", &ed);
     x[0] = s[0] = s[1] = sd;
     x[1] = s[2] = '\0';
     y[0] = d[0] = d[1] = ed;
     d[2] = y[1] = '\0';
     strcat(fs, x);
     for(i = 0; i < strlen(a); i++)
     {
     t[0] = a[i];
        t[1] = '\0';
        if(t[0] == sd)
            strcat(fs, s);
        else if(t[0]==ed)
```

```
            strcat(fs, d);
            else
            strcat(fs, t);
        }
        strcat(fs, y);
        printf("\n After stuffing:%s", fs);
        getch()
    }
```

OUTPUT:

EXPERIMENT -3

AIM: Write a Program to implement data link layer farming method checksum.

Description:In checksum error detection schema the data is divided into k segments each of m bits. In the sender's end the segment are added using 1's complement arithmetic to get the sum. The sum sender is complemented to get the checksum. Now, checksum segment is sent along with the data segment. At receiver's end, all received segment are added using 1's complement arithmetic to get the sum. The sun is complemented. If the result is 0 data is accepted otherwise data is rejected.

PROGRAM:

```
#include<stdio.h>
#include<math.h>
int sender(int arr[10],int n)
{
   int checksum,sum=0,i;
   printf("\n****SENDER
   SIDE****\n"); for(i=0;i<n;i++)
   sum+=arr[i];
   printf("SUM IS: %d",sum);
   checksum=~sum;     //1's complement of
   sum printf("\nCHECKSUM
   IS:%d",checksum); return checksum;
}

void receiver(int arr[10],int n,int sch)
{
   int checksum,sum=0,i;
   printf("\n\n****RECEIVER SIDE****\n");
   for(i=0;i<n;i++)
      sum+=arr[i];
   printf("SUM
   IS:%d",sum);
```

```
        sum=sum+sch;
    checksum=~sum;     //1's complement of sum
    printf("\nCHECKSUM IS:%d",checksum);
}
void main()
{
    int n,sch,rch;
    printf("\nENTER SIZE OF THE STRING:");
    scanf("%d",&n);
    int arr[n];
    printf("ENTER THE ELEMENTS OF THE ARRAY TO CALCULATE
CHECKSUM:\n");
    for(int i=0;i<n;i++)
    {
        scanf("%d",&arr[i]);
    }
    sch=sender(arr,n);
    receiver(arr,n,sch);
}
```

OUTPUT:

EXPERIMENT -4

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

DESCRIPTION: Hamming code is error-detection and error-correction code which is used to find and correct errors in a code while transmission in data communication. The original data bits are mixed with some bits called redundant bits from the sender sides. Then on the receiver side, the Hamming codes are decoded to find the errors while communication. Hamming code can only detect 2-bit error and can correct a single bit error only.

PROGRAM:

```
#include<stdio.h>
void main()
{ int data[10];
   int dataatrec[10],c,c1,c2,c3,i;
   printf("Enter 4 bits of data one by one\n");
   scanf("%d",&data[0]);
   scanf("%d",&data[1]);
   scanf("%d",&data[2]);
   scanf("%d",&data[4]);
   //Calculation of even parity
   data[6]=data[0]^data[2]^data[4];
        data[5]=data[0]^data[1]^data[4];
        data[3]=data[0]^data[1]^data[2];
        printf("\nEncoded data is\n");
     for(i=0;i<7;i++)
     printf("%d",data[i]);
     printf("\n\nEnter received data bits one by one\n");
     for(i=0;i<7;i++)
     scanf("%d",&dataatrec[i]);
    c1=dataatrec[6]^dataatrec[4]^dataatrec[2]^dataatrec[0];
```

```
        c2=dataatrec[5]^dataatrec[4]^dataatrec[1]^dataatrec[0];
        c3=dataatrec[3]^dataatrec[2]^dataatrec[1]^dataatrec[0];
        c=c3*4+c2*2+c1 ;
  if(c==0) {
                printf("\nNo error while transmission of data\n");
  }
        else {
                printf("\nError on position %d",c);

                printf("\nData sent : ");
     for(i=0;i<7;i++)
        printf("%d",data[i]);
                printf("\nData received : ");
     for(i=0;i<7;i++)
        printf("%d",dataatrec[i]);
                printf("\nCorrect message is\n");
                //if errorneous bit is 0 we complement it else vice versa
                if(dataatrec[7-c]==0)
                        dataatrec[7-c]=1;
     else
                        dataatrec[7-c]=0;

                for (i=0;i<7;i++) {
                        printf("%d",dataatrec[i]);
                }
        }
}
```

OUTPUT:

EXPERIMENT -5

AIM:Write a Program to implement on a data set of characters the three CRC polynomials CRC 12, CRC 16 and CRC CCIP.

DESCRIPTION: CRC method can detect a single burst of length n, since only one bit per column will be changed, a burst of length n+1 will pass undetected, if the first bit is inverted, the last bit is inverted and all other bits are correct. If the block is badly garbled by a long burst or by multiple shorter burst, the probability that any of the n columns will have the correct parity that is 0.5.So the probability of a bad block being expected when it should not be 2 power(-n). This scheme is sometimes known as Cyclic Redundancy Code.

PROGRAM:

```
#include<stdio.h>
#include<string.h>
// length of the generator polynomial
#define N strlen(gen_poly)
// data to be transmitted and received
char data[28];
// CRC value
char check_value[28];
// generator polynomial
char gen_poly[10];
// variables
int data_length,i,j;
// function that performs XOR operation
void XOR(){
    // if both bits are the same, the output is 0
    // if the bits are different the output is 1
    for(j = 1;j < N; j++)
    check_value[j] = (( check_value[j] == gen_poly[j])?'0':'1');
}
// Function to check for errors on the receiver side
void receiver(){
```

```
// get the received data
```

```
printf("Enter the received data: ");
scanf("%s", data);
    printf("\n---------------------------------\n");
    printf("Data received: %s", data);
// Cyclic Redundancy Check
    crc();
// Check if the remainder is zero to find the error
    for(i=0;(i<N-1) && (check_value[i]!='1');i++);
        if(i<N-1)
            printf("\nError detected\n\n");
        else
            printf("\nNo error detected\n\n");
}
void crc(){
    // initializing check_value
    for(i=0;i<N;i++)
        check_value[i]=data[i];
    do{
    // check if the first bit is 1 and calls XOR function
        if(check_value[0]=='1')
            XOR();
// Move the bits by 1 position for the next computation
        for(j=0;j<N-1;j++)
            check_value[j]=check_value[j+1];
        // appending a bit from data
        check_value[j]=data[i++];
    }while(i<=data_length+N-1);
// loop until the data ends}
}
```

```
int main()
{
// get the data to be transmitted printf("\nEnter
data to be transmitted: "); scanf("%s",data);
    printf("\n Enter the Generating polynomial: ");
    // get the generator polynomial
    scanf("%s",gen_poly);
    // find the length of data
    data_length=strlen(data);
    // appending n-1 zeros to the data
    for(i=data_length;i<data_length+N-1;i++)
        data[i]='0';
    printf("\n----------------------------------------");
// print the data with padded zeros
    printf("\n Data padded with n-1 zeros : %s",data);
    printf("\n----------------------------------------");
// Cyclic Redundancy Check
    crc();
// print the computed check value
    printf("\nCRC or Check value is : %s",check_value);
// Append data with check_value(CRC)
    for(i=data_length;i<data_length+N-1;i++)
        data[i]=check_value[i-data_length];
    printf("\n----------------------------------------");
// printing the final data to be sent
    printf("\n Final data to be sent : %s",data);
    printf("\n----------------------------------------\n");
// Calling the receiver function to check errors
receiver();
return 0;}
```

EXPERIMENT -6

AIM:Write a Program to implement Sliding window protocol for Go back N.

DESCRIPTION: **Sliding Window Protocol** is actually a theoretical concept in which we have only talked about what should be the sender window size (1+2a) in order to increase the efficiency of stop and wait arq. Now we will talk about the practical implementations in which we take care of what should be the size of receiver window. Practically it is implemented in two protocols namely :

1. Go Back N (GBN)
2. Selective Repeat (SR)

Go Back N (GBN) Protocol

The three main characteristic features of GBN are:

1. Sender Window Size (WS)

It is N itself. If we say the protocol is GB10, then Ws = 10. N should be always greater than 1 in order to implement pipelining. For N = 1, it reduces to Stop and Wait protocol.

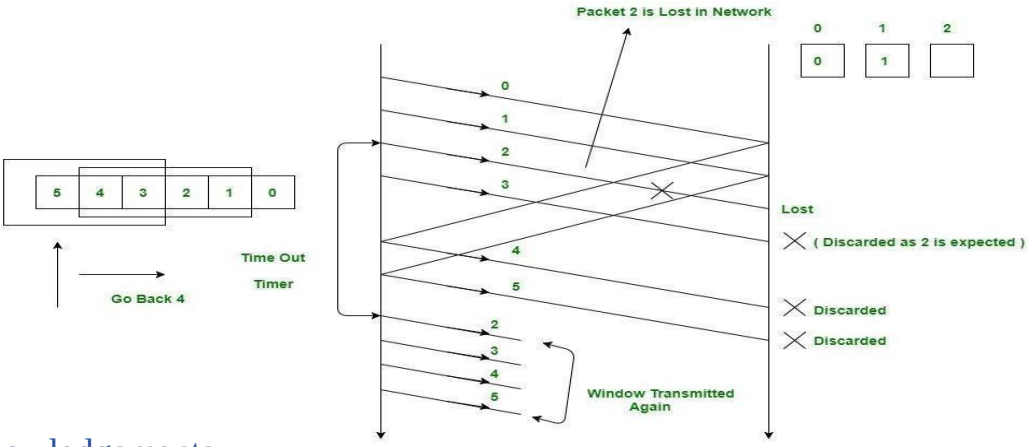
2. Efficiency Of GBN = N/(1+2a)

where a = Tp/Tt
If B is the bandwidth of the channel,
then Effective Bandwidth or
Throughput
= Efficiency * Bandwidth
= (N/(1+2a)) * B

3. Receiver Window Size (WR)

WR is always 1 in GBN.

Now what exactly happens in GBN, we will explain with a help of example. Consider the diagram given below. We have sender window size of 4. Assume that we have lots of sequence numbers just for the sake of explanation. Now the sender has sent the packets 0, 1, 2 and 3. After acknowledging the packets 0 and 1, receiver is now expecting packet 2 and sender window has also slided to further transmit the packets 4 and 5. Now suppose the packet 2 is lost in the network, Receiver will discard all the packets which sender has transmitted after packet 2 as it is expecting sequence number of 2. On the sender side for every packet send there is a time out timer which will expire for packet number 2. Now from the last transmitted packet 5 sender will go back to the packet number 2 in the current window and transmit all the packets till packet number 5. That's why it is called Go Back N. Go back means sender has to go back N places from the last transmitted packet in the unacknowledged window and not from the point where the packet is lost.



Acknowledgements

There are 2 kinds of acknowledgements namely:

- **Cumulative Ack**: One acknowledgement is used for many packets. The main advantage is traffic is less. A disadvantage is less reliability as if one ack is the loss that would mean that all the packets sent are lost.
- **Independent Ack**: If every packet is going to get acknowledgement independently. Reliability is high here but a disadvantage is that traffic is also high since for every packet we are receiving independent ack.

PROGRAM:

```
#include  <stdio.h>

#include <stdlib.h>

#include <time.h>

void sendFrames(int start, int windowSize, int totalFrames)

{

int i, ack, lostFrame = -1;

   srand(time(0)); // Seed for randomness
   while (start < totalFrames) {

      printf("\nSending frames: ");

      for (i = start; i < start + windowSize && i < totalFrames; i++)

      {

      printf("[Frame %d] ", i);

      }

      printf("\n");


       // Simulating frame loss randomly
 if (rand() % 5 == 0) { // 20% chance of loss
```

```
        lostFrame = start + (rand() % windowSize);
            printf("Frame %d lost!\n", lostFrame);
        }

        // Receiving Acknowledgments
        for (i = start; i < start + windowSize && i < totalFrames; i++) {
            if (i == lostFrame) {
                printf("ACK %d not received. Retransmitting from Frame %d\n", i, i);
                start = i; // Retransmit from lost frame

                break;
            } else {
                printf("ACK %d received.\n", i);
            }
        }

        if (i == start + windowSize || i >= totalFrames) {
            start += windowSize;
        }
    }
    printf("\nAll frames sent successfully!\n");
}

int main() {
    int windowSize, totalFrames;
    printf("\n--- Go-Back-N ARQ Simulation ---\n");
    printf("Enter window size: ");
    scanf("%d", &windowSize);
    printf("Enter total number of frames: ");
    scanf("%d", &totalFrames);
```

```
    sendFrames(0, windowSize, totalFrames);
    return 0;
}
```

OUTPUT:

EXPERIMENT -7

AIM:Write a Program to implement Sliding window protocol for Selective repeat.

DESCRIPTION: Selective Repeat ARQ is also form of ARQ protocol in which only suspected or damaged or lost data frames are only retransmitted. This technique is similar to Go-Back-N ARQ though much more efficient than the Go-Back-N ARQ technique due to reason that it reduces number of retransmission. In this, the sender only retransmits frames for which NAK is received. But this technique is used less because of more complexity at sender and receiver and each frame must be needed to acknowledged individually.

PROGRAM:

```
#include<stdio.h>

#include <stdlib.h>

#include <time.h>

#define WINDOW_SIZE 4

#define TOTAL_FRAMES 10
typedef

    struct { int

    frame_no;

    int is_acknowledged;

} Frame;
void send_frames(Frame frames[], int start, int

    window_size) { printf("Sending frames: ");

    for (int i = start; i < start + window_size && i < TOTAL_FRAMES; i++) {

        printf("%d ", frames[i].frame_no);

    }

    printf("\n");

}
void receive_ack(Frame frames[], int start, int window_size) {

    for (int i = start; i < start + window_size && i < TOTAL_FRAMES;

        i++) { if (rand() % 5 != 0) { // Simulating 80% success rate

            frames[i].is_acknowledged = 1;

            printf("Acknowledgment received for frame %d\n", frames[i].frame_no);
```

```
        } else {
            printf("Frame %d lost, needs retransmission\n", frames[i].frame_no);
        }
    }}
int main() {
    Frame frames[TOTAL_FRAMES];
    srand(time(0));
    // Initialize frames
    for (int i = 0; i < TOTAL_FRAMES; i++) {
        frames[i].frame_no = i;
        frames[i].is_acknowledged = 0;
    }

    int base = 0;
    while (base < TOTAL_FRAMES) {
        send_frames(frames, base, WINDOW_SIZE);
        receive_ack(frames, base, WINDOW_SIZE);
        // Move base to the next unacknowledged frame
        while (base < TOTAL_FRAMES && frames[base].is_acknowledged) {
            base++;
        }
    }
    printf("All frames successfully transmitted!\n");
    return 0;
}
```

EXPERIMENT -8

AIM:Write a Program to implement Stop and Wait Protocol.

DESCRIPTION: The error control mechanism is divided into two categories, i.e., Stop and Wait ARQ and sliding window. The sliding window is further divided into two categories, i.e., Go Back N, and Selective Repeat. Based on the usage, the people select the error control mechanism whether it is **stop and wait** or **sliding window**.Here stop and wait means, whatever the data that sender wants to send, he sends the data to the receiver. After sending the data, he stops and waits until he receives the acknowledgment from the receiver. The stop and wait protocol is a flow control protocol where flow control is one of the services of the data link layer.

It is a data-link layer protocol which is used for transmitting the data over the noiseless channels. It provides unidirectional data transmission which means that either sending or receiving of data will take place at a time. It provides flow-control mechanism but does not provide any error control mechanism.The idea behind the usage of this frame is that when the sender sends the frame then he waits for the acknowledgment before sending the next frame.

PROGRAM:

```
#include <stdio.h>

#include

<stdlib.h>

#include <unistd.h>  // For sleep function

#include <time.h>    // For random number generation
// Simulate sending a

packet void

send_packet(int frame) {

   printf("Sending frame %d...\n", frame);

}
// Simulate receiving an acknowledgment (with random

loss) int receive_ack() {

   int success = rand() % 2; // 50% chance of

   success return success;

}
int main() {

   srand(time(NULL)); // Seed for random number generation


    int total_frames = 5;  // Total number of frames to send
```

```
    int frame = 0;

    while (frame < total_frames) {
        send_packet(frame);
        sleep(1); // Simulate transmission delay

        if (receive_ack()) {

            printf("Acknowledgment received for frame %d.\n", frame);
            frame++;
        } else {
            printf("Acknowledgment lost for frame %d. Retransmitting...\n", frame);
        }
    }

    printf("All frames sent successfully.\n");
    return 0;
}
```

OUTPUT:

EXPERIMENT -9

AIM:Write a program for congestion control using leaky bucket algorithm

DESCRIPTION: The main concept of the leaky bucket algorithm is that the output data flow remains constant despite the variant input traffic, such as the water flow in a bucket with a small hole at the bottom. In case the bucket contains water (or packets) then the output flow follows a constant rate, while if the bucket is full any additional load will be lost because of spill over. In a similar way if the bucket is empty the output will be zero. From network perspective, leaky bucket consists of a finite queue (bucket) where all the incoming packets are stored in case there is space in the queue, otherwise the packets are discarded. In order to regulate the output flow, leaky bucket transmits one packet from the queue in a fixed time (e.g. at every clock tick)

The congesting control algorithms are basically divided into two groups: open loop and closed loop.

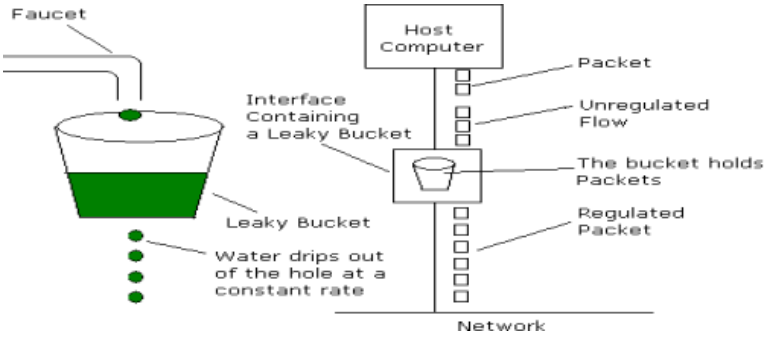
Open loop solutions attempt to solve the problem by good design, in essence, to make sure it does not occur in the first place. Once the system is up and running, midcourse corrections are not made. Open loop algorithms are further divided into ones that act at source versus ones that act at the destination.

Closed loop solutions are based on the concept of a feedback loop if there is any congestion. Closed loop algorithms are also divided into two sub categories: explicit feedback and implicit feedback. In explicit feedback algorithms, packets are sent back from the point of congestion to warn the source. In implicit algorithm, the source deduces the existence of congestion by making local observation, such as the time needed for acknowledgment to come back.

The presence of congestion means that the load is (temporarily) greater than the resources (in part of the system) can handle. For subnets that use virtual circuits internally, these methods can be used at the network layer.

The other method is the leaky bucket algorithm. Each host is connected to the network by an interface containing a leaky bucket, that is, a finite internal queue. If a packet arrives at the queue when it is full, the packet is discarded. In other words, if one or more process are already queued, the new packet is unceremoniously discarded. This arrangement can be built into the hardware interface or simulated by the host operating system. In fact it is nothing other than a single server queuing system with constant service time.

The host is allowed to put one packet per clock tick onto the network. This mechanism turns an uneven flow of packet from the user process inside the host into an even flow of packet onto the network, smoothing out bursts and greatly reducing the chances of congestion



PROGRAM:

```
#include <stdio.h>

#include <stdlib.h>
#include <unistd.h>  // for sleep() function

#define BUCKET_SIZE 10  // Maximum capacity of the bucket
#define OUTPUT_RATE 3   // Rate at which data is sent out
void leakyBucket(int packetSize)
{
static int bucketContent = 0;
   if (packetSize > BUCKET_SIZE) {
      printf("Packet size (%d) too large to fit in the bucket. Packet discarded.\n", packetSize);
   } else {
      if (bucketContent + packetSize > BUCKET_SIZE) {
         printf("Bucket overflow! Packet of size %d discarded.\n", packetSize);
      } else {
         bucketContent += packetSize;
         printf("Packet of size %d added to the bucket. Current bucket content: %d\n",
packetSize, bucketContent);
      }
   }

   // Simulate output of data at the specified rate
   if (bucketContent > 0) {
```

```
        int sent = (bucketContent < OUTPUT_RATE) ? bucketContent : OUTPUT_RATE;
        bucketContent -= sent;
        printf("Sent %d units of data. Remaining bucket content: %d\n", sent, bucketContent);

    } else {
        printf("Bucket is empty. No data sent.\n");
    }
}

int main() {
    int n, packetSize;

    printf("Enter the number of incoming packets: ");
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        printf("Enter size of packet %d: ", i + 1);
        scanf("%d", &packetSize);
        leakyBucket(packetSize);
        sleep(1);  // Simulate time delay between packet arrivals
    }

    // Drain the remaining content from the bucket
    while (bucketContent > 0) {
        leakyBucket(0);
        sleep(1);
    }

    return 0;
}
```

OUTPUT:

EXPERIMENT -10

AIM:Write a Program to implement Dijkstra's algorithm to find shortest path through the graph.

DESCRIPTION: Dijkstra's algorithm is very similar to Prim's algorithm for minimum spanning tree. Like Prim's MST, we generate a SPT (shortest path tree) with given source as root. We maintain two sets, one set contains vertices included in shortest path tree, and other set includes vertices not yet included in shortest path tree.

PROGRAM:

```
#include <stdio.h>

#include <limits.h>

 // Number of vertices in the graph

 #define V 9

// A utility function to find the vertex with minimum distance value, from

// the set of vertices not yet included in shortest path tree

int minDistance(int dist[], int sptSet[]) {

    // Initialize min value

    int min = INT_MAX,

    min_index; int v;

    for (v = 0; v < V; v++)

        if (sptSet[v] == 0 && dist[v] <= min) min = dist[v], min_index = v;

    return min_index;

}

// A utility function to print the constructed distance

array void printSolution(int dist[], int n)

{

    printf("Vertex  Distance from Source\n"); int i;

    for (i = 0; i < V; i++)

        printf("%d \t\t %d\n", i, dist[i]);

}
```

```
// Funtion that implements Dijkstra's single source shortest path algorithm
// for a graph represented using adjacency matrix representation

void dijkstra(int graph[V][V], int src) {
   int dist[V];
   // The output array.  dist[i] will hold the shortest
   // distance from src to i
   int sptSet[V]; // sptSet[i] will 1 if vertex i is included in shortest
   // path tree or shortest distance from src to i is finalized
   // Initialize all distances as INFINITE and stpSet[] as 0
   int i, count, v;
   for (i = 0; i < V; i++)
      dist[i] = INT_MAX, sptSet[i] = 0;
   // Distance of source vertex from itself is always 0
   dist[src] = 0;
   // Find shortest path for all vertices
   for (count = 0; count < V - 1; count++) {
      // Pick the minimum distance vertex from the set of vertices not
      // yet processed. u is always equal to src in first iteration.
      int u = minDistance(dist, sptSet);
      // Mark the picked vertex as processed
      sptSet[u] = 1;
      // Update dist value of the adjacent vertices of the picked vertex.
      for (v = 0; v < V; v++)
         // Update dist[v] only if is not in sptSet, there is an edge from
         // u to v, and total weight of path from src to  v through u is
         // smaller than current value of dist[v]
         if (!sptSet[v] && graph[u][v] && dist[u] != INT_MAX && dist[u]
               + graph[u][v] < dist[v])
            dist[v] = dist[u] + graph[u][v];
   }
```

```
    // print the constructed distance array
    printSolution(dist, V);
}

// driver program to test above function int
main() {
    /* Let us create the example graph discussed above */ int
    graph[V][V] = {{0, 4, 0, 0, 0, 0, 0, 8, 0},
                   {4, 0, 8, 0, 0, 0, 0, 11, 0},
                   {0, 8, 0, 7, 0, 4, 0, 0, 2},
                   {0, 0, 7, 0, 9, 14, 0, 0, 0},
                   {0, 0, 0, 9, 0, 10, 0, 0, 0},
                   {0, 0, 4, 0, 10, 0, 2, 0, 0},
                   {0, 0, 0, 14, 0, 2, 0, 1, 6},
                   {8, 11, 0, 0, 0, 0, 1, 0, 7},
                   {0, 0, 2, 0, 0, 0, 6, 7, 0}
                  };

    dijkstra(graph, 0);

    return 0;
}
```

OUTPUT:

EXPERIMENT -11

AIM:Write a Program to implement Distance vector routing algorithm by obtaining routing table at each node

DESCRIPTION:

- In distance vector routing, each router periodically shares its knowledge about the entire internet with its neighbours.
- The three keys to understanding how this algorithm works are as follows:
  1. Sharing knowledge about the entire autonomous system.
  2. Sharing only with neighbors.
  3. Sharing at regular intervals.
- Every router keeps a routing table that has one entry for each destination network which the router aware.

PROGRAM:

```
#include<stdio.h>
int dist[50][50],temp[50][50],n,i,j,k,x; void
dvr();
int main()
{
   printf("\nEnter the number of nodes : ");
   scanf("%d",&n);
   printf("\nEnter the distance matrix :\n");
   for(i=0;i<n;i++)
   {
      for(j=0;j<n;j++)
      {
         scanf("%d",&dist[i][j]);
         dist[i][i]=0;
         temp[i][j]=j;
      }
      printf("\n");
          }
    dvr();
```

```
printf("enter value of i &j:");

scanf("%d",&i);

scanf("%d",&j); printf("enter the new
cost");

scanf("%d",&x);

dist[i][j]=x;
```