Channabasaveshwara Institute of Technology
(Affiliated to VTU, Belgavi & Approved by AICTE, New Delhi)
(NAAC Accredited & ISO 9001:2015 Certified Institution)
NH 216 (B.H. Road), Gubbi, Tumkur – 572 216. Karnataka.

Department of Information Science & Engineering

LAB MANUAL (2025–26)

COMPUTER NETWORKS (BCS502)

Name :

Usn:

PRACTICAL COMPONENT OF CN - IPCC

Sl.NO Experiments

1. Implement three nodes point – to – point network with duplex links between them. Set the queue size, vary the bandwidth, and find the number of packets dropped.
2. Implement transmission of ping messages/trace route over a network topology consisting of 6 nodes and find the number of packets dropped due to congestion.
3. Implement an Ethernet LAN using n nodes and set multiple traffic nodes and plot congestion window for different source / destination.
4. Develop a program for error detecting code using CRC-CCITT (16- bits).
5. Develop a program to implement a sliding window protocol in the data link layer.
6. Develop a program to find the shortest path between vertices using the Bellman-Ford and path vector routing algorithm.
7. Using TCP/IP sockets, write a client – server program to make the client send the file name and to make the server send back the contents of the requested file if present.
8. Develop a program on a datagram socket for client/server to display the messages on client side, typed at the server side.
9. Develop a program for a simple RSA algorithm to encrypt and decrypt the data.
10. Develop a program for congestion control using a leaky bucket algorithm

1.Implement three nodes point – to – point network with duplex links between them. Set the queue size, vary the bandwidth, and find the number of packets dropped.
TCL:

```
#Create a simulator object

set ns [ new Simulator ]

#Open the nam trace file

set tf [ open lab1.tr w ]

$ns trace-all $tf

#Open the nam trace file

set nf [ open lab1.nam w ]

$ns namtrace-all $nf

#Define a 'finish' procedure

proc finish { } {

global ns nf tf

$ns flush-trace

exec nam lab1.nam &

close $tf

close $nf

exit 0

}

#Creating nodes

set n0 [$ns node]

set n1 [$ns node]

set n2 [$ns node]

set n3 [$ns node]

#Define different colors and labels for data flows

$ns color 1 "red"

$ns color 2 "blue"
```

```
$n0 label "Source/udp0"
$n1 label "Source/udp1"
$n2 label "Router"
$n3 label "Destination/Null"
#Create link between nodes
$ns duplex-link $n0 $n2 100Mb 300ms DropTail
$ns duplex-link $n1 $n2 100Mb 300ms DropTail
$ns duplex-link $n2 $n3 1Mb 300ms DropTail
#Set queue size of links
$ns set queue-limit $n0 $n2 50
$ns set queue-limit $n1 $n2 50
$ns set queue-limit $n2 $n3 5
#Setup a UDP connection
set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0
# Create a CBR traffic source and attach it to udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.005
$cbr0 attach-agent $udp0
#Create a UDP agent and attach it to node n1
set udp1 [new Agent/UDP]
$udp1 set class_ 2
$ns attach-agent $n1 $udp1
# Create a CBR traffic source and attach it to udp1
set cbr1 [new Application/Traffic/CBR]
$cbr1 set packetSize_ 500
```

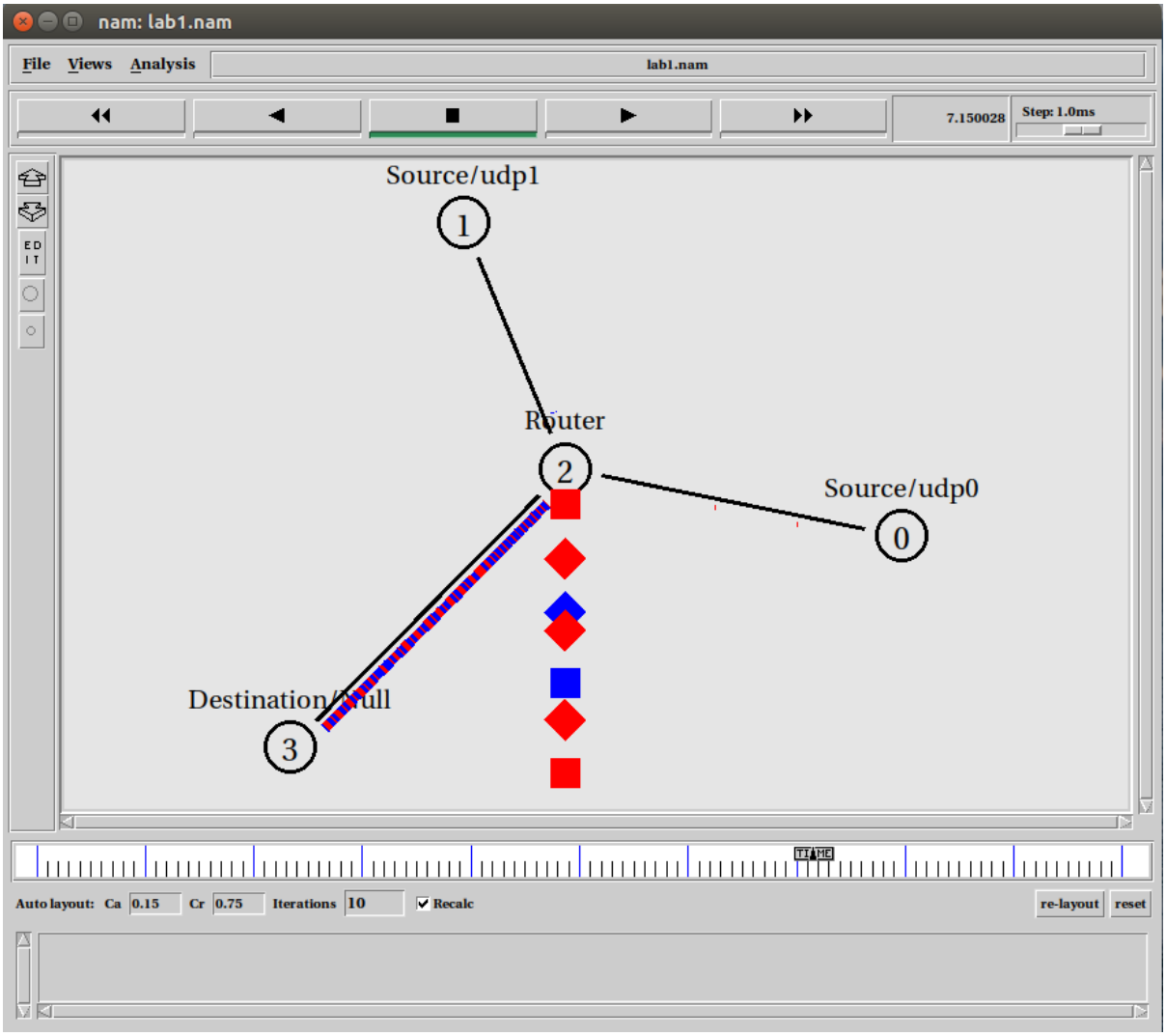
```
$cbr1 set interval_ 0.005
$cbr1 attach-agent $udp1
#Create a Null agent (a traffic sink) and attach it to node n3
set null0 [new Agent/Null]
$ns attach-agent $n3 $null0
#Connect the traffic sources with the traffic sink
$ns connect $udp0 $null0
$ns connect $udp1 $null0
#Schedule events for the CBR agents
$ns at 0.5 "$cbr0 start"
$ns at 1.0 "$cbr1 start"
$ns at 4.0 "$cbr1 stop"
$ns at 4.5 "$cbr0 stop"
#Call the finish procedure after 5 seconds of simulation time
$ns at 5.0 "finish"
#Run the simulation
$ns run
```

AWK:

```
BEGIN{
count=0;
}
{
if($1=="d")
count++
}
END{
printf("The Total no of Packets Drop is :%d\n\n", count)
```

}

Simulated Network:



2.Implement transmission of ping messages/trace route over a network topology consisting of 6 nodes and find the number of packets dropped due to congestion.

```
set ns [new Simulator]
set tf [open lab2.tr w]
$ns trace-all $tf
set nf [open lab2.nam w]
$ns namtrace-all $nf
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
set n6 [$ns node]
$n0 label "Ping0"
$n4 label "Ping4"
$n5 label "Ping5"
$n6 label "Ping6"
$n2 label "Router"
$ns color 1 "red"
$ns color 2 "green"
$ns duplex-link $n0 $n2 100Mb 300ms DropTail
$ns duplex-link $n1 $n2 1Mb 300ms DropTail
$ns duplex-link $n3 $n2 1Mb 300ms DropTail

$ns duplex-link $n5 $n2 100Mb 300ms DropTail
$ns duplex-link $n2 $n4 1Mb 300ms DropTail
$ns duplex-link $n2 $n6 1Mb 300ms DropTail
$ns queue-limit $n0 $n2 5
$ns queue-limit $n2 $n4 3
$ns queue-limit $n2 $n6 2
$ns queue-limit $n5 $n2 5
#The below code is used to connect between the ping agents to the node n0,
#n4 , n5 and n6.
set ping0 [new Agent/Ping]
$ns attach-agent $n0 $ping0
set ping4 [new Agent/Ping]
$ns attach-agent $n4 $ping4
set ping5 [new Agent/Ping]
$ns attach-agent $n5 $ping5
set ping6 [new Agent/Ping]
$ns attach-agent $n6 $ping6
$ping0 set packetSize_ 50000
$ping0 set interval_ 0.0001
$ping5 set packetSize_ 60000
$ping5 set interval_ 0.00001
```

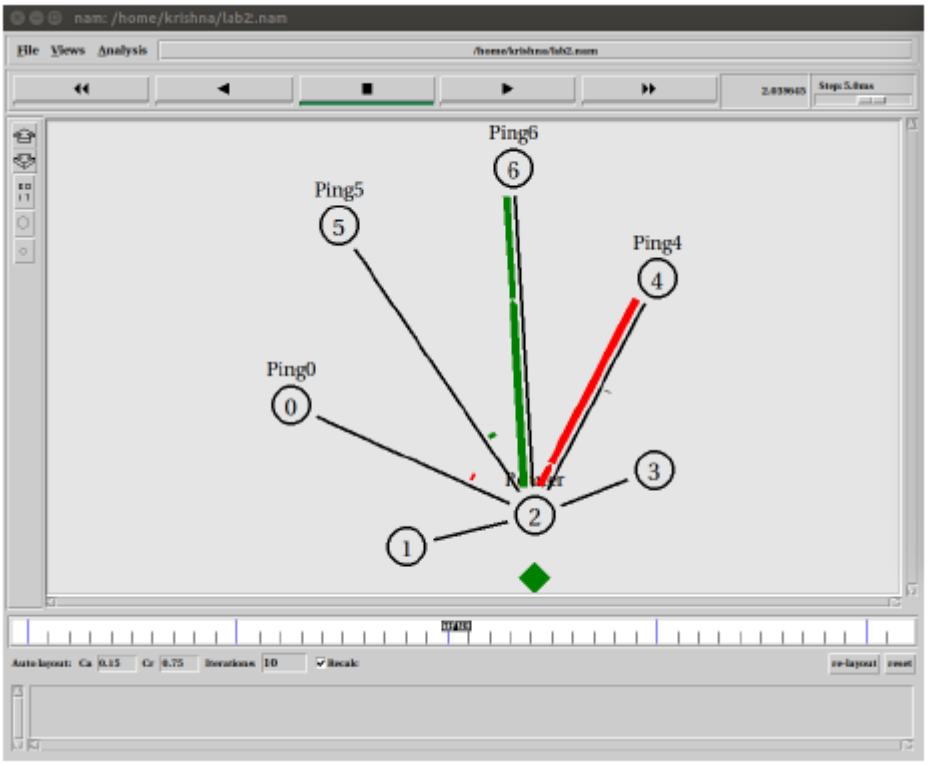
```
$ping0 set class_ 1
$ping5 set class_ 2
$ns connect $ping0 $ping4
$ns connect $ping5 $ping6
#Define a 'recv' function for the class 'Agent/Ping'
#The below function is executed when the ping agent receives a reply from
the destination
Agent/Ping instproc recv {from rtt} {
$self instvar node_
puts " The node [$node_ id] received an reply from $from with round trip
time of $rtt"
}
proc finish {} {
global ns nf tf
exec nam lab2.nam &
$ns flush-trace
close $tf
close $nf
exit 0
}
#Schedule events
$ns at 0.1 "$ping0 send"
$ns at 0.2 "$ping0 send"
$ns at 0.3 "$ping0 send"
$ns at 0.4 "$ping0 send"
$ns at 0.5 "$ping0 send"
$ns at 0.6 "$ping0 send"
$ns at 0.7 "$ping0 send"
$ns at 0.8 "$ping0 send"
$ns at 0.9 "$ping0 send"
$ns at 1.0 "$ping0 send"
$ns at 1.1 "$ping0 send"
$ns at 1.2 "$ping0 send"
$ns at 1.3 "$ping0 send"
$ns at 1.4 "$ping0 send"
$ns at 1.5 "$ping0 send"
$ns at 1.6 "$ping0 send"
$ns at 1.7 "$ping0 send"
$ns at 1.8 "$ping0 send"
$ns at 0.1 "$ping5 send"
$ns at 0.2 "$ping5 send"
$ns at 0.3 "$ping5 send"
$ns at 0.4 "$ping5 send"
$ns at 0.5 "$ping5 send"
$ns at 0.6 "$ping5 send"
$ns at 0.7 "$ping5 send"
$ns at 0.8 "$ping5 send"
$ns at 0.9 "$ping5 send"
$ns at 1.0 "$ping5 send"
```

```
$ns at 1.1 "$ping5 send"
$ns at 1.2 "$ping5 send"
$ns at 1.3 "$ping5 send"
$ns at 1.4 "$ping5 send"
$ns at 1.5 "$ping5 send"
$ns at 1.6 "$ping5 send"
$ns at 1.7 "$ping5 send"
$ns at 1.8 "$ping5 send"
$ns at 5.0 "finish"
$ns run
```

AWK:

```
BEGIN{
count=0;
}
{
if($1=="d")
count++;
}
END{
printf("The Total no of Packets Drop is :%d\n\n", count);
}
```

Topology:



3.Implement an Ethernet LAN using n nodes and set multiple traffic nodes and plot congestion
window for different source / destination.

```
#Make a NS simulator
set ns [new Simulator]
set tf [open lab3.tr w]
$ns trace-all $tf
set nf [open lab3.nam w]
$ns namtrace-all $nf
# Create the nodes,color and label
set n0 [$ns node]
$n0 color "magenta"
$n0 label "src1"
set n1 [$ns node]
$n1 color "red"
set n2 [$ns node]
$n2 color "magenta"
$n2 label "src2"
set n3 [$ns node]
$n3 color "blue"
$n3 label "dest2"
set n4 [$ns node]
$n4 shape square
set n5 [$ns node]
$n5 color "blue"
$n5 label "dest1"

#Creates a lan from a set of nodes given by <nodelist>. Bandwidth, delay
#characteristics along with the link-layer, Interface queue, Mac layer and
#channel type for the lan also needs to be defined.
$ns make-lan "$n0 $n1 $n2 $n3 $n4" 50Mb 100ms LL Queue/DropTail
Mac/802_3
# Create the link
$ns duplex-link $n4 $n5 1Mb 1ms DropTail
# Create the node position
$ns duplex-link-op $n4 $n5 orient right
# Add a TCP sending module to node n0
set tcp0 [new Agent/TCP]
$ns attach-agent $n0 $tcp0
# Setup a FTP traffic generator on "tcp0"
set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0
$ftp0 set packetSize_ 500
$ftp0 set interval_ 0.0001
# Add a TCP receiving module to node n5
set sink0 [new Agent/TCPSink]
$ns attach-agent $n5 $sink0
```

```
# Direct traffic from "tcp0" to "sink1"
$ns connect $tcp0 $sink0
# Add a TCP sending module to node n2
set tcp1 [new Agent/TCP]
$ns attach-agent $n2 $tcp1

# Setup a FTP traffic generator on "tcp1"
set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1
$ftp1 set packetSize_ 600
$ftp1 set interval_ 0.001
# Add a TCP receiving module to node n3
set sink1 [new Agent/TCPSink]
$ns attach-agent $n3 $sink1
# Direct traffic from "tcp1" to "sink1"
$ns connect $tcp1 $sink1
set file1 [open file1.tr w]
$tcp0 attach $file1
set file2 [open file2.tr w]
$tcp1 attach $file2
$tcp0 trace cwnd_
$tcp1 trace cwnd_
# Define a 'finish' procedure
proc finish { } {
global ns nf tf
$ns flush-trace
close $tf
close $nf
exec nam lab3.nam &
exit 0
}

# Schedule start/stop times
$ns at 0.1 "$ftp0 start"
$ns at 5 "$ftp0 stop"
$ns at 7 "$ftp0 start"
$ns at 0.2 "$ftp1 start"
$ns at 8 "$ftp1 stop"
$ns at 14 "$ftp0 stop"
$ns at 10 "$ftp1 start"
$ns at 15 "$ftp1 stop"
# Set simulation end time
$ns at 16 "finish"
$ns run
```

AWK:

```
BEGIN {
}
```

```
{
if($6=="cwnd_")
printf("%f\t%f\t\n",$1,$7);
}
END {
}
```

4.Develop a program for error detecting code using CRC-CCITT (16- bits).

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

// CRC-CCITT (16-bit) generator polynomial
const char GENERATOR[] = "10001000000100001"; // x^16 + x^12 + x^5 + 1

// Function to perform XOR operation on two binary strings
void xor_binary(char* result, const char* a, const char* b, int len) {
   for (int i = 0; i < len; i++) {
      result[i] = ((a[i] - '0') ^ (b[i] - '0')) + '0';
   }
   result[len] = '\0';
}

// Function to calculate CRC
void calculate_crc(char* data, char* crc_output) {
   int data_len = strlen(data);
   int gen_len = strlen(GENERATOR);
   int remainder_len = gen_len - 1;

   // Append 16 zeros to the data
   char* temp_data = (char*)malloc(data_len + remainder_len + 1);
   strcpy(temp_data, data);
   for (int i = 0; i < remainder_len; i++) {
      strcat(temp_data, "0");
   }

   char* current_remainder = (char*)malloc(gen_len + 1);
   strncpy(current_remainder, temp_data, gen_len);
   current_remainder[gen_len] = '\0';

   for (int i = 0; i < data_len; i++) {
      if (current_remainder[0] == '1') {
         xor_binary(current_remainder, current_remainder, GENERATOR, gen_len);
      }

      // Shift left and bring in the next bit
      for (int j = 0; j < gen_len - 1; j++) {
         current_remainder[j] = current_remainder[j + 1];
      }
      current_remainder[gen_len - 1] = temp_data[gen_len + i];
   }

   // The final remainder is the CRC
   strncpy(crc_output, current_remainder, remainder_len);
   crc_output[remainder_len] = '\0';
```

```
    free(temp_data);
    free(current_remainder);
}
```

5.Develop a program to implement a sliding window protocol in the data link layer.

Go-Back-N ARQ

```
#include<stdio.h>

int main()

{

        int windowsize,sent=0,ack,i;

        printf("enter window size\n");

        scanf("%d",&windowsize);

        while(1)

        {

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

                        {

                                printf("Frame %d has been transmitted.\n",sent);

                                sent++;

                                if(sent == windowsize)

                                        break;

                        }

                        printf("\nPlease enter the last Acknowledgement received.\n");

                        scanf("%d",&ack);


                        if(ack == windowsize)

                                break;

                        else

                                sent = ack;

        }

return 0;

}
```

Here's a simple **Go-Back-N ARQ** implementation using C socket programming to simulate communication between a client (sender) and a server (receiver).

Overview:

- **Go-Back-N ARQ** allows the sender to send multiple packets (window size) without waiting for individual ACKs.
- If a packet is lost or an ACK is not received, all packets starting from the lost packet are retransmitted.
- The server randomly simulates ACK loss or successful reception.

Program Structure:

- **Server**: Simulates ACK reception with a chance of ACK loss, acknowledging packets up to the first lost packet.
- **Client**: Sends packets in a sliding window fashion, retransmitting the entire window if an ACK is lost.

Server - Receiver

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <time.h>

#define PORT 8080
#define BUFFER_SIZE 1024
#define LOSS_PROBABILITY 30  // 30% chance of ACK loss

int main() {
   int server_fd, new_socket;
   struct sockaddr_in address;
   int addrlen = sizeof(address);
   char buffer[BUFFER_SIZE] = {0};
   int ack;

   srand(time(0));  // Random seed for ACK simulation

   // Create socket
   if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
      perror("Socket creation failed");
      exit(EXIT_FAILURE);
   }

   address.sin_family = AF_INET;
```

```
    address.sin_addr.s_addr = INADDR_ANY;
    address.sin_port = htons(PORT);

    // Bind socket
    if (bind(server_fd, (struct sockaddr *)&address, sizeof(address)) < 0) {
        perror("Bind failed");
        exit(EXIT_FAILURE);
    }

    // Listen for client
    if (listen(server_fd, 3) < 0) {
        perror("Listen failed");
        exit(EXIT_FAILURE);
    }

    printf("Server: Waiting for connection...\n");

        if ((new_socket = accept(server_fd, (struct sockaddr *)&address,
(socklen_t*)&addrlen)) < 0) {
        perror("Accept failed");
        exit(EXIT_FAILURE);
    }

    printf("Server: Connection established.\n");

    while (1) {
        memset(buffer, 0, BUFFER_SIZE);
        int valread = read(new_socket, buffer, BUFFER_SIZE);
        if (valread == 0) break;

        ack = atoi(buffer);
        printf("Server: Received packet %d\n", ack);

        // Simulate ACK loss
        if (rand() % 100 < LOSS_PROBABILITY) {
            printf("Server: ACK for packet %d lost!\n\n", ack);
        } else {
            sleep(1);  // Simulate processing delay
            printf("Server: ACK sent for packet %d\n\n", ack);
            memset(buffer,0,BUFFER_SIZE);
            sprintf(buffer, "%d", ack);  // Send ACK
            send(new_socket, buffer, strlen(buffer)+1, 0);
        }
    }

    close(new_socket);
    close(server_fd);
    return 0;
}
```

Client - Sender

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/time.h>

#define PORT 8080
#define BUFFER_SIZE 1024
#define TIMEOUT 3  // Timeout in seconds
#define WINDOW_SIZE 4  // Sliding window size
#define TOTAL_PACKETS 10  // Number of packets to send

int main() {
   int sock = 0;
   struct sockaddr_in serv_addr;
   char buffer[BUFFER_SIZE] = {0};
   struct timeval tv;

   if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
      perror("Socket creation failed");
      exit(EXIT_FAILURE);
   }

   serv_addr.sin_family = AF_INET;
   serv_addr.sin_port = htons(PORT);

   if (inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr) <= 0) {
      perror("Invalid address/ Address not supported");
      exit(EXIT_FAILURE);
   }

   if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0) {
      perror("Connection failed");
      exit(EXIT_FAILURE);
   }

   printf("Client: Connected to server.\n");

   tv.tv_sec = TIMEOUT;
   tv.tv_usec = 0;
   setsockopt(sock, SOL_SOCKET, SO_RCVTIMEO, (const char*)&tv, sizeof(tv));

   int base = 1;
   int next_to_send = 1;
   int ack, packets_acked = 0;
```

```
   while (packets_acked < TOTAL_PACKETS) {
      // Send all packets in the window
      while (next_to_send < base + WINDOW_SIZE && next_to_send <= TOTAL_PACKETS)
{
         memset(buffer,0,BUFFER_SIZE);
         printf("Client: Sending packet %d\n", next_to_send);
         sprintf(buffer, "%d", next_to_send);
         send(sock, buffer, strlen(buffer)+1, 0);
         next_to_send++;
    }

      // Wait for ACK
      memset(buffer, 0, BUFFER_SIZE);
      int valread = read(sock, buffer, BUFFER_SIZE);

      if (valread > 0) {
         ack = atoi(buffer);
         printf("Client: ACK received for packet %d\n", ack);

         // Slide the window if ACK corresponds to the base
         if (ack == base) {
            base = ack + 1;
            packets_acked = ack;
         }
      } else {
         printf("Client: Timeout! Retransmitting from packet %d...\n", base);
         next_to_send = base;  // Reset next_to_send to base for retransmission
      }
   }

   printf("Client: All packets sent successfully.\n");
   close(sock);
   return 0;
}
```

How to Run the Program:

1. **Compile the Server and Client**

```
gcc server.c -o server
gcc client.c -o client
```

2. **Run the Server**

```
./server
```

3. **Run the Client** (in a new terminal)

./client

How It Works:

- **Client** sends packets in batches (window size of 4).
- If an ACK is not received within the timeout period, the client retransmits starting from the first unacknowledged packet.
- **Server** randomly decides to send ACKs or simulate ACK loss.
- This process continues until all packets are successfully acknowledged.

Testing Process:

1. **Run the server** first.
2. **Run the client** in a separate terminal.
3. Observe the gradual packet transmission and retransmissions based on ACK loss.

Key Points:

- **Sliding window** approach for efficiency.
- Efficient retransmission by avoiding the need to retransmit all packets, only those starting from the missing ACK.
- Simulates **real-world network issues** like packet loss and retransmission.

Code Explanations - Receiver Server

1. Headers and Macros:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <time.h>
```

- **Purpose:** These headers provide necessary functions for socket programming and general operations.
  - stdio.h – Standard I/O functions (printf, perror).
  - stdlib.h – For general utilities (exit, atoi, rand).
  - string.h – For string handling (memset, strlen).
  - unistd.h – For system calls (read, write, close).
  - arpa/inet.h – Socket-related functions (inet_addr, sockaddr_in).
  - time.h – For randomization (time, srand).

```
#define PORT 8080
#define BUFFER_SIZE 1024
```

```
#define LOSS_PROBABILITY 30
```

- **PORT** – The server listens on port 8080.
- **BUFFER_SIZE** – Defines the size of the buffer used for sending/receiving data (1 KB).
- **LOSS_PROBABILITY** – Simulates a 30% chance that the ACK will be "lost" (simulated by not sending an acknowledgment).

---

2. Variable Declarations:

```
int server_fd, new_socket;
struct sockaddr_in address;
int addrlen = sizeof(address);
char buffer[BUFFER_SIZE] = {0};
int ack;
```

- **server_fd** – File descriptor for the server socket.
- **new_socket** – File descriptor for the client connection (after accept()).
- **address** – Stores server address information (IPv4, port, IP).
- **addrlen** – Length of the address structure.
- **buffer** – Temporary buffer for sending and receiving data.
- **ack** – Packet number for which an acknowledgment (ACK) is sent.

---

3. Random Seed Initialization:

```
srand(time(0));
```

- **Purpose:** Initializes the random number generator for simulating packet loss.
- **srand(time(0))** – Seeds randomness based on the current time to ensure that every execution results in different ACK loss patterns.

---

4. Socket Creation:

```
if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
   perror("Socket creation failed");
   exit(EXIT_FAILURE);
}
```

- **socket(AF_INET, SOCK_STREAM, 0)** – Creates a TCP socket (SOCK_STREAM indicates a stream-based connection).

- **Error Handling:** If socket creation fails, an error is printed (perror), and the program exits.

---

5. Server Address Configuration:

```
address.sin_family = AF_INET;
address.sin_addr.s_addr = INADDR_ANY;
address.sin_port = htons(PORT);
```

- **sin_family** – Specifies the address family (AF_INET for IPv4).
- **sin_addr.s_addr** – Binds the server to listen on all available network interfaces (INADDR_ANY).
- **sin_port** – Converts the port number to network byte order (htons(PORT)). This ensures compatibility across systems with different byte ordering.

---

6. Binding the Socket to the Address:

```
if (bind(server_fd, (struct sockaddr *)&address, sizeof(address)) < 0) {
   perror("Bind failed");
   exit(EXIT_FAILURE);
}
```

- **bind** – Binds the server socket to the specified IP and port.
- **Error Handling:** If the bind fails (e.g., if the port is already in use), the server exits.

---

7. Listening for Incoming Connections:

```
if (listen(server_fd, 3) < 0) {
   perror("Listen failed");
   exit(EXIT_FAILURE);
}
```

- **listen** – Puts the server in a listening state for incoming connections.
- **3** – Maximum number of pending connections in the queue.
- **Error Handling:** If listen fails, the server exits.

---

8. Accepting Client Connections:

```
printf("Server: Waiting for connection...\n");

if ((new_socket = accept(server_fd, (struct sockaddr *)&address, (socklen_t*)&addrlen))
< 0) {
   perror("Accept failed");
   exit(EXIT_FAILURE);
}
```

- **accept** – Waits for a client to connect. It blocks execution until a connection request arrives.
- **new_socket** – A new socket for handling the specific client connection (different from the server socket).
- **Error Handling:** If accept fails, the server exits.

---

9. Connection Established:

```
printf("Server: Connection established.\n");
```

- Indicates that the client successfully connected.

---

10. Receiving and Sending ACKs:

```
while (1) {
   memset(buffer, 0, BUFFER_SIZE);
   int valread = read(new_socket, buffer, BUFFER_SIZE);
   if (valread == 0) break;
```

- **while (1)** – Infinite loop to keep receiving packets until the client disconnects.
- **memset** – Clears the buffer before each packet reception to avoid residual data.
- **read** – Reads data from the client into the buffer.
- **valread == 0** – If no data is received (client disconnects), the loop breaks.

---

11. Processing Received Packets:

```
ack = atoi(buffer);
printf("Server: Received packet %d\n", ack);
```

- **atoi** – Converts the received packet number (in string format) to an integer.

---

12. Simulating ACK Loss:

```
if (rand() % 100 < LOSS_PROBABILITY) {
   printf("Server: ACK for packet %d lost!\n\n", ack);
}
```

- **rand() % 100 < 30** – Simulates ACK loss by generating a random number between 0 and 99. If the value falls below 30 (30% chance), the ACK is "lost".
- **No ACK Sent:** The server prints the loss message but does not send an acknowledgment.

---

13. Sending ACKs (if not lost):

```
else {
   sleep(1);
   printf("Server: ACK sent for packet %d\n\n", ack);
   memset(buffer, 0, BUFFER_SIZE);
   sprintf(buffer, "%d", ack);
   send(new_socket, buffer, strlen(buffer) + 1, 0);
}
```

- **sleep(1)** – Simulates network delay by pausing for 1 second before sending an ACK.
- **sprintf** – Converts the integer ack to a string format for transmission.
- **send** – Sends the ACK back to the client.

---

14. Closing Sockets:

```
close(new_socket);
close(server_fd);
return 0;
```

- **close** – Closes both the client (new_socket) and server (server_fd) sockets to release resources.

Summary:

- The server accepts a connection from the client and continuously listens for incoming packets.
- For each packet received, it simulates ACK loss with a 30% probability.
- If ACK is not "lost," the server sends it back to the client after a small delay.

- This simulates the behavior of a Go-Back-N ARQ protocol where packets may need retransmission if ACKs are not received.

Code Explanations Sender - Client

1. Headers and Macros:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/time.h>
```

- **Standard C Headers:**
  - stdio.h – For input/output functions (printf, sprintf).
  - stdlib.h – For general functions like exit(), atoi().
  - string.h – For string manipulation (strlen, memset).
  - unistd.h – For system calls (read, write, close).
  - arpa/inet.h – For socket functions (inet_pton, sockaddr_in).
  - sys/time.h – For handling timeouts with setsockopt().

```
#define PORT 8080
#define BUFFER_SIZE 1024
#define TIMEOUT 3  // Timeout in seconds
#define WINDOW_SIZE 4  // Sliding window size
#define TOTAL_PACKETS 10  // Number of packets to send
```

- **Macros:**
  - PORT – Defines the port number used for communication.
  - BUFFER_SIZE – Sets the buffer size for sending and receiving data.
  - TIMEOUT – Specifies how long the client will wait for an ACK before retransmitting.
  - WINDOW_SIZE – Number of packets that can be sent before waiting for ACKs.
  - TOTAL_PACKETS – Total number of packets to send.

---

2. Socket Creation and Connection:

```
int sock = 0;
struct sockaddr_in serv_addr;
char buffer[BUFFER_SIZE] = {0};
struct timeval tv;
```

- **Socket Variables:**
  - sock – The socket descriptor.
  - serv_addr – Stores server address information.
  - buffer – Temporary buffer for sending and receiving data.
  - tv – Timeout structure for setting the socket read timeout.

```
if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
   perror("Socket creation failed");
   exit(EXIT_FAILURE);
}
```

- **Create Socket:**
  - socket(AF_INET, SOCK_STREAM, 0) – Creates a TCP socket.
  - AF_INET – IPv4 address family.
  - SOCK_STREAM – Stream socket (TCP).
  - perror – Prints error if socket creation fails.

---

3. Server Address Setup:

```
serv_addr.sin_family = AF_INET;
serv_addr.sin_port = htons(PORT);

if (inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr) <= 0) {
   perror("Invalid address/ Address not supported");
   exit(EXIT_FAILURE);
}
```

- **Configure Server Address:**
  - sin_family – Set to AF_INET (IPv4).
  - sin_port – Converts the port to network byte order using htons().
  - inet_pton – Converts IP address (localhost 127.0.0.1) from text to binary.

---

4. Connect to Server:

```
if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0) {
   perror("Connection failed");
   exit(EXIT_FAILURE);
}
printf("Client: Connected to server.\n");
```

- **Establish Connection:**
  - connect() – Connects to the server using the specified address.
  - If the connection fails, an error message is displayed.

5. Set Timeout:

```
tv.tv_sec = TIMEOUT;
tv.tv_usec = 0;
setsockopt(sock, SOL_SOCKET, SO_RCVTIMEO, (const char*)&tv, sizeof(tv));
```

- **Timeout Setup:**
  - setsockopt – Configures the socket to timeout if no data is received within 3 seconds (TIMEOUT).

---

6. Sliding Window Variables:

```
int base = 1;
int next_to_send = 1;
int ack, packets_acked = 0;
```

- **Sliding Window Variables:**
  - base – The first unacknowledged packet.
  - next_to_send – Tracks the next packet to be sent.
  - ack – Stores received ACKs.
  - packets_acked – Total packets acknowledged so far.

---

7. Sending Packets (Sliding Window):

```
while (packets_acked < TOTAL_PACKETS) {
   // Send packets in the window
   while (next_to_send < base + WINDOW_SIZE && next_to_send <= TOTAL_PACKETS) {
      printf("Client: Sending packet %d\n", next_to_send);
      sprintf(buffer, "%d", next_to_send);
      send(sock, buffer, strlen(buffer)+1, 0);
      next_to_send++;
   }
```

- **Send Packets Within Window:**
  - while (next_to_send < base + WINDOW_SIZE) – Send packets until the window is full.
  - sprintf(buffer, "%d", next_to_send) – Prepare packet number as a string.
  - send() – Send the packet to the server.
  - Increment next_to_send after each packet.

---

8. Wait for ACKs:

```
  while (1) {
    memset(buffer, 0, BUFFER_SIZE);
    int valread = read(sock, buffer, BUFFER_SIZE);

    if (valread > 0) {
      ack = atoi(buffer);
      printf("Client: ACK received for packet %d\n", ack);

      if (ack >= base) {
        base = ack + 1;
        packets_acked = ack;
      }
    } else {
      printf("Client: Timeout! Retransmitting from packet %d...\n", base);
      next_to_send = base;
      break;
    }
  }
}
```

- **ACK Handling (Loop Until Timeout):**
  - read() – Wait for the server to send an ACK.
  - atoi(buffer) – Convert received string to integer (ACK number).
  - If ACK is received, slide the window by adjusting base and increment packets_acked.
  - **Timeout Handling:**
    - If no ACK is received within 3 seconds, the client retransmits packets starting from the base.

---

9. Completion:

```
printf("Client: All packets sent successfully.\n");
close(sock);
return 0;
```

- **Completion Message:**
  - Once all packets are acknowledged, the client displays a success message and closes the socket.

How It Works:

1. **Window-Based Transmission:**

   - Packets are sent in chunks (size of the window).
   - The client waits for ACKs for the sent packets.
2. **Timeouts and Retransmission:**
   - If an ACK is not received within 3 seconds, the client retransmits starting from the first unacknowledged packet (base).
3. **Efficiency:**
   - Multiple packets can be sent before waiting for ACKs (Go-Back-N mechanism).
   - The window slides forward when consecutive ACKs are received.

Key Concepts:

- **Sliding Window:** Allows multiple packets to be in-flight at once.
- **Timeout & Retransmission:** Ensures lost packets are resent.
- **Go-Back-N ARQ:** If a packet is lost, all packets from that point are retransmitted.

6. Develop a program to find the shortest path between vertices using the Bellman-Ford and path vector routing algorithm.

```
// Bellman Ford Algorithm in C

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

#define INFINITY 99999

//struct for the edges of the graph
struct Edge {
  int u;  //start vertex of the edge
  int v;  //end vertex of the edge
  int w;  //weight of the edge (u,v)
};

//Graph - it consists of edges
struct Graph {
  int V;       //total number of vertices in the graph
  int E;       //total number of edges in the graph
  struct Edge *edge;  //array of edges
};

void bellmanford(struct Graph *g, int source);
void display(int arr[], int size);

int main(void) {
  //create graph
  struct Graph *g = (struct Graph *)malloc(sizeof(struct Graph));
  g->V = 4;  //total vertices
  g->E = 5;  //total edges

  //array of edges for graph
  g->edge = (struct Edge *)malloc(g->E * sizeof(struct Edge));

  //------- adding the edges of the graph
  /*
                edge(u, v)
                where  u = start vertex of the edge (u,v)
                              v = end vertex of the edge (u,v)

                w is the weight of the edge (u,v)
        */

  //edge 0 --> 1
  g->edge[0].u = 0;
  g->edge[0].v = 1;
  g->edge[0].w = 5;
```

```
  //edge 0 --> 2
  g->edge[1].u = 0;
  g->edge[1].v = 2;
  g->edge[1].w = 4;

  //edge 1 --> 3
  g->edge[2].u = 1;
  g->edge[2].v = 3;
  g->edge[2].w = 3;

  //edge 2 --> 1
  g->edge[3].u = 2;
  g->edge[3].v = 1;
  g->edge[3].w = 6;

  //edge 3 --> 2
  g->edge[4].u = 3;
  g->edge[4].v = 2;
  g->edge[4].w = 2;

  bellmanford(g, 0);  //0 is the source vertex

  return 0;
}

void bellmanford(struct Graph *g, int source) {
  //variables
  int i, j, u, v, w;

  //total vertex in the graph g
  int tV = g->V;

  //total edge in the graph g
  int tE = g->E;

  //distance array
  //size equal to the number of vertices of the graph g
  int d[tV];

  //predecessor array
  //size equal to the number of vertices of the graph g
  int p[tV];

  //step 1: fill the distance array and predecessor array
  for (i = 0; i < tV; i++) {
    d[i] = INFINITY;
    p[i] = 0;
  }

  //mark the source vertex
```

```
  d[source] = 0;

  //step 2: relax edges |V| - 1 times
  for (i = 1; i <= tV - 1; i++) {
    for (j = 0; j < tE; j++) {
      //get the edge data
      u = g->edge[j].u;
      v = g->edge[j].v;
      w = g->edge[j].w;

      if (d[u] != INFINITY && d[v] > d[u] + w) {
        d[v] = d[u] + w;
        p[v] = u;
      }
    }
  }

  //step 3: detect negative cycle
  //if value changes then we have a negative cycle in the graph
  //and we cannot find the shortest distances
  for (i = 0; i < tE; i++) {
    u = g->edge[i].u;
    v = g->edge[i].v;
    w = g->edge[i].w;
    if (d[u] != INFINITY && d[v] > d[u] + w) {
      printf("Negative weight cycle detected!\n");
      return;
    }
  }

  //No negative weight cycle found!
  //print the distance and predecessor array
  printf("Distance array: ");
  display(d, tV);
  printf("Predecessor array: ");
  display(p, tV);
}

void display(int arr[], int size) {
  int i;
  for (i = 0; i < size; i++) {
    printf("%d ", arr[i]);
  }
  printf("\n");
}
```

7.Using TCP/IP sockets, write a client – server program to make the client send the file name and to make the server send back the contents of the requested file if present.

TCP/IP sockets, Client - Server program to make the client send the file name and to make the server send back the contents of the requested file if present.

```
import java.net.*;
import java.io.*;
public class ContentsClient
{
 public static void main( String args[ ] ) throws Exception
 {
 Socket sock = new Socket("127.0.0.1", 4000);
 // reading the file name from keyboard. Uses input stream
 System.out.print("Enter the file name");
 BufferedReader keyRead = new BufferedReader(new InputStreamReader(System.in));
 String fname = keyRead.readLine();

 // sending the file name to server. Uses PrintWriter
 OutputStream ostream = sock.getOutputStream( );
 PrintWriter pwrite = new PrintWriter(ostream, true);
 pwrite.println(fname);

 // receiving the contents from server. Uses input stream
 InputStream istream = sock.getInputStream();
 BufferedReader socketRead = new BufferedReader(new InputStreamReader(istream));
 String str;
 while((str = socketRead.readLine()) != null) // reading line-by-line
 {
 System.out.println(str);
 }
 pwrite.close();
 socketRead.close();
 keyRead.close();
}
}
```

TCP/IP sockets, Client - Server program to make the client send the file name and to make the server send back the contents of the requested file if present.

```
import java.net.*;
import java.io.*;
public class ContentsServer
{
 public static void main(String args[]) throws Exception
 {
 // establishing the connection with the server
 ServerSocket sersock = new ServerSocket(4000);
```

```
System.out.println("Server ready for connection");
Socket sock = sersock.accept(); // binding with port: 4000
System.out.println("Connection is successful and wating for client request");

// reading the file name from client
InputStream istream = sock.getInputStream( );
BufferedReader fileRead =new BufferedReader(new InputStreamReader(istream));
String fname = fileRead.readLine( );

// reading file contents
BufferedReader contentRead = new BufferedReader(new FileReader(fname) );

// keeping output stream ready to send the contents
OutputStream ostream = sock.getOutputStream( );
PrintWriter pwrite = new PrintWriter(ostream, true);

String str;
while((str = contentRead.readLine()) != null) // reading line-by-line from file
{
pwrite.println(str); // sending each line to client
}
sock.close(); sersock.close(); // closing network sockets
pwrite.close(); fileRead.close(); contentRead.close();
}
}
```

8. Develop a program on a datagram socket for client/server to display the messages on client side, typed at the server side.

Sender.java

```
import java.io.*;
import java.net.*;
public class Sender
{
 public static void main(String[] args) throws IOException
 {
 InetAddress addr = InetAddress.getByName(args[0]);
 byte[] buf = args[1].getBytes();
 DatagramPacket packet = new DatagramPacket(buf, buf.length, addr, 4444);
 DatagramSocket socket = new DatagramSocket();
 socket.send(packet);
 }
}
```

Receiver.java

```
import java.io.*;
import java.net.*;
public class Receiver
{
 public static void main(String[] args) throws IOException
 {
 DatagramSocket socket = new DatagramSocket(4444);
 byte[] buf = new byte[256];
 DatagramPacket packet = new DatagramPacket(buf, buf.length);
 System.out.println("Waiting ...");
 socket.receive(packet);
 String s = new String(packet.getData(), 0, packet.getLength());
 System.out.println(packet.getAddress().getHostName() + ": " + s);
}
}
```

Datagram socket for client/server to display the messages on client side,
typed at the server side.

- Compile the program.
- Start the receiver by running “java Receiver”.
- Assuming that the receiver is running on a host with IP address 127.0.0.1
  Start the sender by running:

java Sender 127.0.0.1 “My String”

- The receiver program should now display the string “My String”.
- Repeat this exercise, with the difference, that you run the sender and receiver on two different hosts.

9. Develop a program for a simple RSA algorithm to encrypt and decrypt the data.

```
// Java Program to Implement the RSA Algorithm
import java.math.*;
import java.util.*;

class RSA {
   public static void main(String args[])
   {
      int p, q, n, z, d = 0, e, i;

      // The number to be encrypted and decrypted
      int msg = 12;
      double c;
      BigInteger msgback;

      // 1st prime number p
      p = 3;

      // 2nd prime number q
      q = 11;
      n = p * q;
      z = (p - 1) * (q - 1);
      System.out.println("the value of z = " + z);

      for (e = 2; e < z; e++) {

         // e is for public key exponent
         if (gcd(e, z) == 1) {
            break;
         }
      }
      System.out.println("the value of e = " + e);
      for (i = 0; i <= 9; i++) {
         int x = 1 + (i * z);

         // d is for private key exponent
         if (x % e == 0) {
            d = x / e;
            break;
         }
      }
      System.out.println("the value of d = " + d);
      c = (Math.pow(msg, e)) % n;
      System.out.println("Encrypted message is : " + c);

      // converting int value of n to BigInteger
      BigInteger N = BigInteger.valueOf(n);

      // converting float value of c to BigInteger
```

```
        BigInteger C = BigDecimal.valueOf(c).toBigInteger();
        msgback = (C.pow(d)).mod(N);
        System.out.println("Decrypted message is : "
                        + msgback);
    }

    static int gcd(int e, int z)
    {
        if (e == 0)
            return z;
        else
            return gcd(z % e, e);
    }
}
```

10.Develop a program for congestion control using a leaky bucket algorithm.

```
import java.util.Scanner;
public class LeakyBucket {
public static void main(String[] args) throws InterruptedException {
int n, incoming, outgoing, store=0, bucketsize;
Scanner scan = new Scanner(System.in);
System.out.println("Enter bucket size, outgoing rate, number of inputs and incoming
size");
 bucketsize = scan.nextInt();
 outgoing = scan.nextInt();
 n = scan.nextInt();
 incoming = scan.nextInt();

 while(n!=0)
 {
 System.out.println("Incoming size is " + incoming);
 if(incoming <= (bucketsize-store))
 {
 store+=incoming;
 System.out.println("Bucket buffer size is " + store +" out of " + bucketsize);
 }
 else
 {
 System.out.println("Packet loss : " + (incoming-(bucketsize-store)));
 store=bucketsize;

 System.out.println("Bucket buffer size is " + store +" out of " + bucketsize);
 }

 store-=outgoing;
 System.out.println("After outgoing: " + store + " packets left out of " + bucketsize
 + "in buffer");

 n--;
 Thread.sleep(3000);
 }
 scan.close();
 }
}
```