

Computer Networks Laboratory

B.Tech. 5th Semester



Name : Chandan Kumar
Roll Number : 22ETCS002301
Programme : B.Tech in CSE
Department : Computer Science and Engineering

Faculty of Engineering & Technology
Ramaiah University of Applied Sciences



Ramaiah University of Applied Sciences

Private University Established in Karnataka State by Act No. 15 of 2013

Faculty	Engineering & Technology
Programme	B. Tech. in Computer Science and Engineering/ Information Science and Engineering/ Artificial Intelligence and Machine Learning
Year/Semester	3 rd Year /5 th Semester
Name of the Laboratory	Computer Networks Laboratory
Laboratory Code	CSL301A

List of Experiments

1. Error Detection using Parity
2. Error Detection using CRC
3. Neighbour Table Determination
4. Distance Vector Routing
5. ARQ Mechanisms in DLL
6. Socket Programming-I
7. DLL ARQ Mechanisms using TCP Sockets

Index Sheet

No	Lab Experiment	Lab document	Viva
1	Error Detection using Parity	Lab document 10% of Lab CE	
2	Error Detection using CRC	Lab document 10% of Lab CE	
3	Neighbour Table Determination	Lab document 10% of Lab CE	
4	Distance Vector Routing	Lab document 10% of Lab CE	
5	ARQ Mechanisms in DLL	Lab document 10% of Lab CE	
6	Socket Programming-I	Lab document 10% of Lab CE	
7	DLL ARQ Mechanisms using TCP Sockets	Lab document 10% of Lab CE	
8	Lab Internal Test	10% of Lab CE	

S. No	Experiments Name	Page No
1	Error Detection using Parity	5-9
2	Error Detection using CRC	10-15
3	Neighbor Table Determination	16-21
4	Distance Vector Routing	22-27
5	ARQ Mechanisms in DLL	28-33
6	Socket Programming-I	34-37
7	DLL ARQ Mechanisms using TCP Sockets	38-41

Laboratory 1

Title of the Laboratory Exercise: Error Detection using Parity

1. Introduction and Purpose of Experiment

The purpose of this experiment is to introduce students to the concept of error detection in computer networks using parity bits. Error detection plays a crucial role in ensuring the integrity of transmitted data. In this lab, students will learn how to implement and verify the effectiveness of parity bit-based error detection techniques. They will gain hands-on experience in detecting and correcting single-bit errors, which is fundamental in network communication protocols. Additionally, this exercise will lay the foundation for more advanced error detection and correction techniques explored in subsequent labs.

2. Aim and Objectives

Aim

- To apply Parity check rules for error detection

Objectives

At the end of this lab, the student will be able to

- Apply 1D and 2D parity rules for error detection
- Analyze the difference between 1D and 2D parity and their limitations

3. Experimental Procedure

Problem statement: You are required to write separate programs to demonstrate the use of 1D and 2D parity. Take the input bit streams (max five) of 7 bit each from the user. Your programs should calculate the parity and display the input and output bit streams.

- Analysis: While analyzing your program, you are required to address the following points:
 - Why can this method not be used to correct errors?
 - How are 1D and 2D parity different?
 - What are the limitations of this method of error detection?

4. Algorithm/Flowchart

Part A : 1D parity

1. START

2. Input number of data bits n
3. Input data bits - input[]
4. Enter 1 for odd parity and 2 for even parity.
5. Count total number of 1's in the given data bits.
6. For odd parity:

 if count == even, then parity bit = 1;

 if count == odd, then parity bit = 0;
7. For even parity:

 If count == even then parity bit = 0;

 If count == odd then parity bit = 1;
8. input[n+1] = parity bit
9. print input[].
10. END

Part B : 2D parity

1. START
2. input number of data bits n
3. Input number of data sets m.
4. create a 2D array of size [m+1][n+1].
5. store the data bits in the array.
6. Input the user's choice of even or odd parity.
7. calculate the even or odd parity row wise and column wise.
8. print the data that will be finally transmitted to the screen.
9. END

5. Program

Below is the screenshot of the code for 1D parity which is written in C program.

```

source code
1  #include <stdio.h>
2  #include <string.h>
3  void main()
4  {
5      char input[100];
6      printf("give data bro: ");
7      scanf("%s", input);
8      int n = strlen(input);
9      int parity = 0;
10     for (int i = 0; i < n; i++)
11     {
12         int k = input[i] - '0';
13         if (k == 1)
14         {
15             parity++;
16         }
17         else if (k != 0)
18         {
19             printf("why bro, Enter '0' or '1' only \n");
20             return;
21         }
22     }
23     char parity[2] = "";
24     if (parity % 2 == 0)
25     {
26         parity[0] = '0';
27     }
28     else
29     {
30         parity[0] = '1';
31     }
32     printf("data = %s\n", input);
33     printf("Parity = %s\n", parity);
34     strcat(input, parity);
35     printf("data with parity is %s\n", input);
36 }

```

Below is the screenshot of the code for 2D even or odd parity which is written in C program.

```

main.c
1  #include <stdio.h>
2  #include <string.h>
3
4  void main()
5  {
6      char bit_stream1[10], bit_stream2[10], bit_stream3[10], bit_stream4[10], bit_stream5[10];
7      char *streams[] = {bit_stream1, bit_stream2, bit_stream3, bit_stream4, bit_stream5};
8      char col_parities[5] = {0, 0, 0, 0, 0};
9      char row_parities[7] = {0, 0, 0, 0, 0, 0, 0};
10
11     for (int i = 0; i < 5; i++)
12     {
13         printf("enter the bit stream %d (max 7 bits): ", i + 1);
14         scanf("%9s", streams[i]);
15         printf("you entered %9s\n", streams[i]);
16         if (strlen(streams[i]) > 7)
17         {
18             printf("max 7 bits ");
19             return;
20         }
21         int k = strlen(streams[i]);
22         for (int j = 0; j < k; j++)
23         {
24             if (streams[i][j] - '0' == 1)
25             {
26                 col_parities[i]++;
27                 row_parities[j]++;
28             }
29             else if (streams[i][j] - '0' != 0)
30             {
31                 printf("enter only '0' or '1' please ");
32             }
33         }
34     }
35     char row_par[10] = "";
36     int par = 0;
37     for (int i = 0; i < 7; i++)
38     {
39         if (row_parities[i] % 2 == 0)
40         {
41             (strcat(row_par, "0"));
42         }

```

```

43         else
44         {
45             strcat(row_par, "1");
46             par++;
47         };
48     }
49
50     for (int i = 0; i < 5; i++)
51     {
52         (col_parities[i] % 2 == 0) ? (strcat(streams[i], "0")) : (strcat(streams[i], "1"));
53         printf("data = %.7s,column parity = %c \n", streams[i], streams[i][7]);
54     }
55
56     ((par % 2 == 0) ? (strcat(row_par, "0")) : (strcat(row_par, "1")));
57     printf("row parities = %.7s,column parity of row parity = %c \n", row_par, row_par[7]);
58     printf("final transmitted data is\n");
59     for (int i = 0; i < 5; i++)
60     {
61         printf("%s", streams[i]);
62     }
63     printf("%s", row_par);
64     printf("\n");
65 }

```

6. Results

Below is the screenshot of the output of the program to check for 1D even or odd parity.

```

enter the data: 1100110
data = 1100110
Parity = 0
data with parity is 11001100

```

Below is the screenshot of the output of the program to check for 2D even or odd parity.

```

enter the bit stream 1 (max 7 bits): 1100101
you entered 1100101
enter the bit stream 2 (max 7 bits): 1110001
you entered 1110001
enter the bit stream 3 (max 7 bits): 1010101
you entered 1010101
enter the bit stream 4 (max 7 bits): 0101010
you entered 0101010
enter the bit stream 5 (max 7 bits): 1001001
you entered 1001001
data = 1100101,column parity = 0
data = 1110001,column parity = 0
data = 1010101,column parity = 0
data = 0101010,column parity = 1
data = 1001001,column parity = 1
row parities = 0100010,column parity of row parity = 0
final transmitted data is
1100101011100010101010010101011001001101000100

```


7. Analysis and Discussions

The transfer of binary information through communication mediums can be disrupted by external noise, leading to errors when bits change from 1 to 0 or vice versa. Error detection and correction codes, like parity bits, help ensure efficient data transfer. A parity bit is an additional bit added to a binary message to make the number of 1's either odd (odd parity) or even (even parity). In even parity, a 1 is appended if the count of 1's is odd; otherwise, a 0 is added. In odd parity, a 1 is appended to make the count of 1's odd.

Parity can detect only an odd number of errors and does not identify or correct specific errors. It is limited to error detection, as even errors can result in matching parity. Additionally, the distinction between 1D and 2D arrays is highlighted: 1D arrays organize data in a list, while 2D arrays represent data in rows and columns like a table.

8. Conclusions

We can conclude that the parity check method is not the most efficient error detection method out there as it cannot detect multiple errors and also correct the errors if there are any.

9. Comments

1. Limitations of the experiment

- Parity check method cannot be used to correct errors.
- It is not suitable for detection of multiple errors(eg. two, fours, sixes etc)
- This method cannot be used to reveal the location of the erroneous bit. And hence this method cannot be used to correct the error.

2. Limitations of the results obtained

- The results obtained have not been corrected

3. Learning

- We learned about the 1D and 2D parity check method and how they detect errors in the data transferred.

Laboratory 2

Title of the Laboratory Exercise: Error Detection using CRC

1. Introduction and Purpose of Experiment

The purpose of this experiment is to build upon the concepts learned in the previous lab and delve deeper into error detection techniques in computer networks. Cyclic Redundancy Check (CRC) is a widely used method for detecting errors in transmitted data. In this lab, students will gain practical experience in implementing CRC algorithms and understanding how they can be used to detect various types of errors, including burst errors. Through hands-on exercises, students will learn to calculate and verify CRC codes, enhancing their understanding of error detection mechanisms crucial for reliable data communication in networks. This lab will provide a solid foundation for more advanced topics in error detection and correction.

2. Aim and Objectives

Aim

- To apply CRC for error detection

Objectives

At the end of this lab, the student will be able to

- Apply CRC to develop codes for error detection
- Analyse how this CRC is able to detect bit errors irrespective of their length and position in the data

3. Experimental Procedure

- i. Analyse the problem statement
- ii. Design an algorithm for the given problem statement and develop a flowchart/pseudo-code
- iii. Implement the algorithm in Python language
- iv. Execute the Python program
- v. Test the implemented program
- vi. Document the Results
- vii. Analyse and discuss the outcomes of the experiment

4. Question

Problem statement: You are required to write a program that uses CRC to detect burst errors in transmitted data. Initially, write the program using the CRC example you studied in class. Your final program should ask the user to input data and choose a generator polynomial from the list given in the figure below. Your program is required to calculate the checksum and the transmitted data. Subsequently, the user enters the received data. Applying the same generator polynomial on the received data should result in a remainder of 0.

Name	Polynomial	Application
CRC-8	$x^8 + x^2 + x + 1$	ATM header
CRC-10	$x^{10} + x^9 + x^5 + x^4 + x^2 + 1$	ATM AAL
CRC-16	$x^{16} + x^{12} + x^5 + 1$	HDLC
CRC-32	$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$	LANs

Analysis: While analysing your program, you are required to address the following points:

- How is this method different from 2D parity scheme that you have implemented previously?
- What are the limitations of this method of error detection?

5. Calculations/Computations/Algorithms

1. Input Selection:

- Prompt the user to select the type of CRC to calculate: CRC-8, CRC-10, CRC-16, or CRC-32.
- Based on the user's choice, assign the corresponding generator polynomial.

2. Input Data:

- Ask the user to input the data (binary string) for which the CRC needs to be calculated.

3. Data Preparation:

- Determine the lengths of the input data and the selected generator polynomial.
- Append zeros to the data equal to the length of the generator polynomial minus one. This padded data simulates the input message with space for the CRC remainder.

4. Division Simulation (Modulo-2 Arithmetic):

- Copy the first segment of the padded data (equal to the generator length) as the initial **dividend**.
- Perform the division as follows:
 - If the first bit of the dividend is 1, perform an XOR operation between the dividend and the generator polynomial.
 - Shift the dividend left by removing its first bit and append the next bit from the padded data to the end of the dividend.
 - Repeat this process until all bits of the padded data are processed.

5. CRC Calculation:

- At the end of the division process, the remaining bits in the dividend represent the CRC.

6. Output:

- Print the padded data for verification.
- Display the computed CRC (remainder).

7. End.

6. Program

```

source code
1  #include <stdio.h>
2  #include <string.h>
3
4  // Function to perform XOR operation
5  void xor_operation(char *dividend, const char *generator, int len) {
6      for (int i = 0; i < len; i++) {
7          dividend[i] = (dividend[i] == generator[i]) ? '0' : '1';
8      }
9  }
10
11 // Function to perform CRC
12 void crc(char data[], const char generator[]) {
13     int data_len = strlen(data);
14     int gen_len = strlen(generator);
15     int total_len = data_len + gen_len - 1;
16
17     // Append zeros to the data (simulate the message with padding for CRC).
18     char padded_data[total_len + 1];
19     strcpy(padded_data, data);
20     for (int i = data_len; i < total_len; i++) {
21         padded_data[i] = '0';
22     }
23     padded_data[total_len] = '\0';
24
25     printf("Padded data: %s\n", padded_data);
26
27     char dividend[gen_len];
28     strncpy(dividend, padded_data, gen_len);
29     dividend[gen_len] = '\0';
30
31     for (int i = 0; i <= total_len - gen_len; i++) {
32         if (dividend[0] == '1') {
33             // Perform XOR with the generator polynomial
34             xor_operation(dividend, generator, gen_len);
35         }
36
37         // Shift left and bring in the next bit from padded_data
38         memmove(dividend, dividend + 1, gen_len - 1);
39         dividend[gen_len - 1] = padded_data[i + gen_len];
40         dividend[gen_len] = '\0';
41     }
42
43     printf("Remainder (CRC): %s\n", dividend);
44 }

```

```

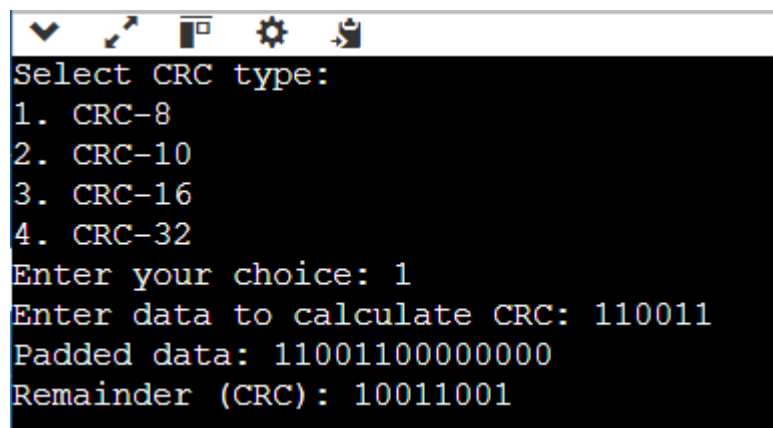
45
46 ▾ int main() {
47     char data[256]; // Input data buffer
48     int choice;
49     const char *generator;
50
51     // Menu to select CRC type
52     printf("Select CRC type:\n");
53     printf("1. CRC-8\n");
54     printf("2. CRC-10\n");
55     printf("3. CRC-16\n");
56     printf("4. CRC-32\n");
57     printf("Enter your choice: ");
58     scanf("%d", &choice);
59
60     // Assign generator polynomial based on choice
61 ▾ switch (choice) {
62     case 1:
63         generator = "100000111"; // x^8 + x^2 + x + 1
64         break;
65     case 2:
66         generator = "11000110101"; // x^10 + x^9 + x^5 + x^4 + x^2 + 1
67         break;
68     case 3:
69         generator = "1100000000000101"; // x^16 + x^12 + x^5 + 1
70         break;
71     case 4:
72         generator = "100000100110000010001110110110111"; // x^32 + ...
73         break;
74     default:
75         printf("Invalid choice.\n");
76         return 1;
77 }
78
79 // Input data from the user
80 printf("Enter data to calculate CRC: ");
81 scanf("%s", data);
82
83 // Calculate and print CRC
84 crc(data, generator);
85
86 return 0;
87 }

```

7. Presentation of Results

Below is the screenshot of the output of the program when there is no error in the data that was transmitted :

CRC-8

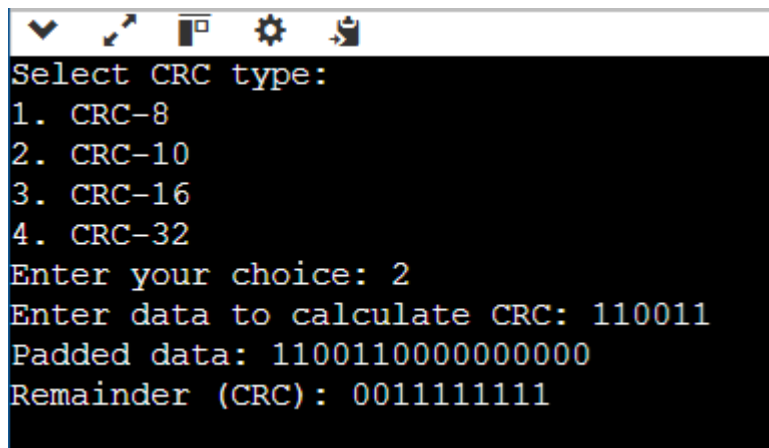


```

Select CRC type:
1. CRC-8
2. CRC-10
3. CRC-16
4. CRC-32
Enter your choice: 1
Enter data to calculate CRC: 110011
Padded data: 11001100000000
Remainder (CRC): 10011001

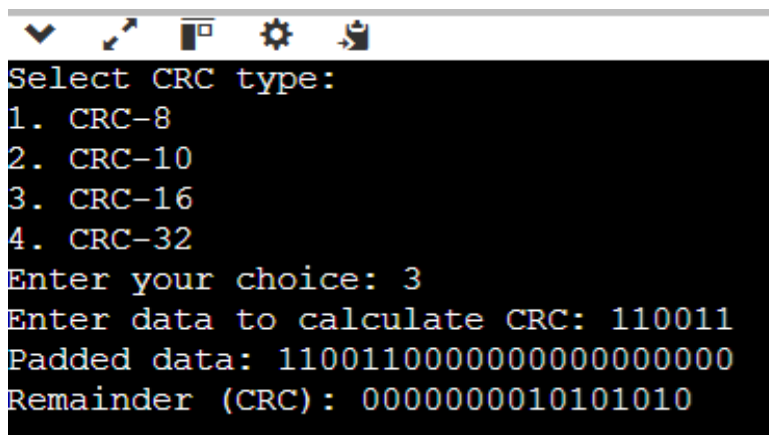
```

CRC-10



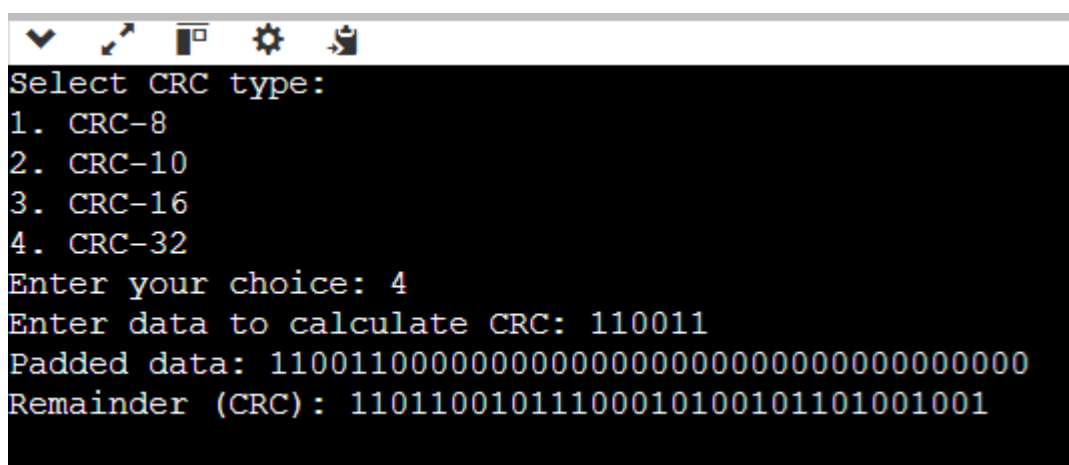
```
▼ ↗ 📄 ⚙️ 📋
Select CRC type:
1. CRC-8
2. CRC-10
3. CRC-16
4. CRC-32
Enter your choice: 2
Enter data to calculate CRC: 110011
Padded data: 1100110000000000
Remainder (CRC): 0011111111
```

CRC-16



```
▼ ↗ 📄 ⚙️ 📋
Select CRC type:
1. CRC-8
2. CRC-10
3. CRC-16
4. CRC-32
Enter your choice: 3
Enter data to calculate CRC: 110011
Padded data: 110011000000000000000000
Remainder (CRC): 0000000010101010
```

CRC-32



```
▼ ↗ 📄 ⚙️ 📋
Select CRC type:
1. CRC-8
2. CRC-10
3. CRC-16
4. CRC-32
Enter your choice: 4
Enter data to calculate CRC: 110011
Padded data: 1100110000000000000000000000000000000000000000000000000
Remainder (CRC): 11011001011100010100101101001001
```

8. Conclusions

In this lab experiment we learned how the CRC error detection method actually works and what are its advantages. It is a powerful error detecting mechanism which outperforms the simple parity check method and 2D parity check method.

9. Comments

1. Limitations of Experiments

- CRC is not suitable for protecting against intentional alteration of data, and overflow of data is possible in CRC.
- Only binary data can be entered and not other types of data. v

2. Limitations of Results

- The results obtained simply tell us if our data was sent successfully without being corrupted.
- It doesn't provide information as to which bit was corrupted and doesn't correct it either

3. Learning happened

- We learned how to code a sample CRC error detection program. We learned how it is more powerful than our simple 1D and 2D parity check

Laboratory 3

Title of the Laboratory Exercise: Neighbour Table Determination

1. Introduction and Purpose of Experiment

The objective of this experiment is to familiarize students with the concept of neighbor table determination in computer networks. A neighbor table, also known as an adjacency table, is a vital data structure used in network routing protocols to keep track of directly connected devices. In this lab, students will learn how to construct and update neighbor tables through practical exercises. They will gain hands-on experience in using network monitoring tools and protocols to discover and maintain information about neighboring devices. Understanding neighbor tables is essential for effective routing and plays a crucial role in network management and optimization. This lab will lay the groundwork for more advanced routing protocols and network troubleshooting techniques.

2. Aim and Objectives

Aim

- To create neighbour table for a given network topology

Objectives

At the end of this lab, the student will be able to

- Generate neighbour table for all the nodes in a given topology.
- Analyse how this is useful in the process of routing data

3. Experimental Procedure

- Analyse the problem statement
- Design an algorithm for the given problem statement and develop a flowchart/pseudo-code
- Implement the algorithm in Python language
- Execute the Python program
- Test the implemented program
- Document the Results
- Analyse and discuss the outcomes of the experiment

4. Questions

Problem statement: You are required to write a program that calculates neighbor table for all the nodes in a given network. Consider a network with 10 nodes that is deployed in an area of 500 m². Your program should initially determine the distance between each node and all other nodes. Then the range of the nodes is given as input to the user. Using this range information, determine the neighbors of all the nodes.

Analysis: While analyzing your program, you are required to address the following points:

- How this is useful in the process of routing data?
- For a 3D topology, how would your program need to be changed?

5. Calculations/Computations/Algorithms

1. START
2. At first we take the length and breadth of the rectangular area as input from the user.
3. Then we create ten nodes with random coordinates.
4. After that we calculate the distance between each and every node and present them on the screen. We use the formula $\sqrt{(x1 - x2)^2 + (y1 - y2)^2}$ to calculate the distance between any two nodes.
5. Then we ask the user to input the range of each node.
6. After that we check each and every node's distance from each other and if it is lesser or equal to the given range we consider that node as a neighbour to the other node.
7. Then we print all the neighbour's of all the nodes.
8. STOP

6. Program

Below is the screenshot of the program in C language:

```
main.c
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  #include <math.h>
5  #include <time.h>
6
7  int i, j, n;
8  int x[15], y[15];
9  float distance[10][10];
```

```

11 void createNetwork() {
12     int x_range, y_range;
13     printf("Enter the number of nodes (max 10): ");
14     scanf("%d", &n);
15     printf("Enter X-Coordinate range: ");
16     scanf("%d", &x_range);
17     printf("Enter Y-Coordinate range: ");
18     scanf("%d", &y_range);
19
20     srand(time(NULL));
21
22     for (i = 0; i < n; i++) {
23         x[i] = 1 + rand() % x_range;
24         y[i] = 1 + rand() % y_range;
25     }
26
27     printf("\nX\tY\n");
28     for (i = 0; i < n; i++) {
29         printf("(%d, %d)\n", x[i], y[i]);
30     }
31 }
32
33 void computeDistance() {
34     for (i = 0; i < n; i++) {
35         for (j = 0; j < n; j++) {
36             distance[i][j] = sqrt(((x[j] - x[i]) * (x[j] - x[i])) + ((y[j] - y[i]) * (y[j] - y[i])));
37         }
38     }
39
40     printf("\nDistance Matrix:\n\t");
41     for (i = 0; i < n; i++) {
42         printf("\t%d", i + 1);
43     }
44     printf("\n");
45
46     for (i = 0; i < n; i++) {
47         printf("%d\t", i + 1);
48         for (j = 0; j < n; j++) {
49             printf("%f\t", distance[i][j]);
50         }
51         printf("\n");
52     }
53 }
54
55 void findNeighbor() {
56     float range;
57     printf("\nEnter the neighbor range: ");
58     scanf("%f", &range);
59
60     for (i = 0; i < n; i++) {
61         printf("\nNeighbors of %d are:\t", i + 1);
62         for (j = 0; j < n; j++) {
63             if (i == j) {
64                 continue;
65             }
66             if (distance[i][j] <= range) {
67                 printf("%d (%f)\t", j + 1, distance[i][j]);
68             }
69         }
70         printf("\n");
71     }
72 }
73
74 int main(int argc, char* argv[]) {
75     createNetwork();
76     computeDistance();
77     findNeighbor();
78     return EXIT_SUCCESS;
79 }
80

```

7. Presentation of Results

```

input
Enter the number of nodes (max 10): 10
Enter X-Coordinate range: 500
Enter Y-Coordinate range: 500

X      Y
(204, 354)
(41, 107)
(98, 395)
(99, 377)
(433, 148)
(292, 255)
(313, 336)
(24, 423)
(76, 479)
(182, 185)

Distance Matrix:
      1      2      3      4      5      6      7      8      9      10
1  0.000000  295.935791  113.652977  107.489532  308.021118  132.457535
2  295.935791  0.000000  293.586456  276.159363  394.138306  291.384613
3  113.652977  293.586456  0.000000  18.027756  416.213898  239.240463
4  107.489532  276.159363  18.027756  0.000000  404.965424  228.326523
5  308.021118  394.138306  416.213898  404.965424  0.000000  177.002823
6  132.457535  291.384613  239.240463  228.326523  177.002823  0.000000
7  110.476242  355.562927  222.948425  217.892181  223.033630  83.677956
8  192.771881  316.456940  79.120163  87.982956  492.854950  316.303650
9  178.910599  373.642883  86.833176  104.560989  486.836731  311.178406
10 170.425934  161.136581  226.176926  209.172180  253.712433  130.384048

```

```

Enter the neighbor range: 100

Neighbors of 1 are:
Neighbors of 2 are:
Neighbors of 3 are:      4 (18.027756)      8 (79.120163)      9 (86.833176)
Neighbors of 4 are:      3 (18.027756)      8 (87.982956)
Neighbors of 5 are:
Neighbors of 6 are:      7 (83.677956)
Neighbors of 7 are:      6 (83.677956)
Neighbors of 8 are:      3 (79.120163)      4 (87.982956)      9 (76.419891)
Neighbors of 9 are:      3 (86.833176)      8 (76.419891)
Neighbors of 10 are:

```

8. Analysis and Discussions

1. How this is useful in the process of routing data?

The program aids in routing data by:

- Simulating Network Topology: Models the positions of nodes in 2D/3D space, vital for networks like WSNs or IoT.
- Distance Calculation: Helps estimate communication cost (e.g., power, latency) between nodes.
- Neighbor Discovery: Identifies nearby nodes for establishing connections and forwarding data.
- Routing Path Selection: Provides data for algorithms like greedy forwarding, shortest path, or flooding to optimize data transfer.

2. For a 3D topology, how would your program need to be changed?

To adapt the code for 3D coordinates, you'll need to incorporate an additional dimension, the z-coordinate, into the logic. Here's what needs to change:

Changes for 3D:

- Add an Array for z-Coordinates:** Declare an array `int z[15]` to store the z-coordinates of the nodes.

```
6  int i, j, n;
7  int x[15], y[15], z[15]; // Added z[] for 3D coordinates
8  float distance[10][10];
```

- Modify createNetwork to Generate z-Coordinates:** Add a range for z-coordinates and generate random z-values.

```
10 void createNetwork() {
11     int x_range, y_range, z_range;
12     printf("Enter the number of nodes (max 10): ");
13     scanf("%d", &n);
14     printf("Enter X-Coordinate range: ");
15     scanf("%d", &x_range);
16     printf("Enter Y-Coordinate range: ");
17     scanf("%d", &y_range);
18     printf("Enter Z-Coordinate range: ");
19     scanf("%d", &z_range);
20
21     srand(time(NULL));
22
23     for (i = 0; i < n; i++) {
24         x[i] = 1 + rand() % x_range;
25         y[i] = 1 + rand() % y_range;
26         z[i] = 1 + rand() % z_range; // Randomly generating z-coordinate
27     }
28
29     printf("\nX\tY\tZ\n");
30     for (i = 0; i < n; i++) {
31         printf("(%d, %d, %d)\n", x[i], y[i], z[i]);
32     }
33 }
```

- Update the Distance Formula in computeDistance:** Modify the distance calculation to include the z-dimension.

```

35 void computeDistance() {
36     for (i = 0; i < n; i++) {
37         for (j = 0; j < n; j++) {
38             distance[i][j] = sqrt(((x[j] - x[i]) * (x[j] - x[i])) +
39                                   ((y[j] - y[i]) * (y[j] - y[i])) +
40                                   ((z[j] - z[i]) * (z[j] - z[i]))); // Updated distance formula for 3D
41         }
42     }
43
44     printf("\nDistance Matrix:\n\t");
45     for (i = 0; i < n; i++) {
46         printf("\t%d", i + 1);
47     }
48     printf("\n");
49
50     for (i = 0; i < n; i++) {
51         printf("%d\t", i + 1);
52         for (j = 0; j < n; j++) {
53             printf("%f\t", distance[i][j]);
54         }
55         printf("\n");
56     }
57 }

```

- d. **Keep findNeighbor Unchanged:** The neighbor-finding logic remains the same because it already compares distances. The distances now account for 3D space.

Final Notes:

- These changes extend the program's capability to work in 3D space, adding a z-coordinate dimension to node positions.
- Ensure proper bounds for the z-coordinate range when entering data.
- The output will display 3D coordinates and their corresponding distance matrix.

9. Conclusions

We can conclude that a good routing algorithm is greatly needed to properly manage the network traffic in today's day. We learned how to calculate the distance between each and every node in a 2-D topology and that also gave us an idea as to the changes we must make in our program to do the same for nodes in a 3-D topology

10. Comments

1. Limitations of Experiments

- a. The program won't be working for nodes in a 3-D topology

2. Learning happened

- a. We learned how to calculate the distance between each and every node in a 2-D topology and that also gave us an idea as to the changes we must make in our program to do the same for nodes in a 3-D topology.

Laboratory 4

Title of the Laboratory Exercise: Distance Vector Routing

1. Introduction and Purpose of Experiment

The purpose of this experiment is to introduce students to the fundamentals of distance vector routing algorithms in computer networks. Distance vector routing is a key method used to determine the best path for data packets in a network. In this lab, students will gain practical experience in implementing and simulating distance vector routing protocols, such as the Bellman-Ford algorithm. They will learn how routers exchange routing information and update their routing tables based on distance metrics. Through hands-on exercises and simulations, students will develop a deep understanding of how distance vector routing algorithms work, enabling them to analyse and optimize network performance. This lab sets the stage for exploring more advanced routing protocols and network optimization techniques..

2. Aim and Objectives

Aim

- To generate routing tables for a network of routers using Distance Vector Routing

Objectives

At the end of this lab, the student will be able to

- Generate routing tables for a given network using Distance Vector Routing
- Analyze the reasons why Distance Vector Routing is adaptive in nature

3. Experimental Procedure

- i. Analyse the problem statement
- ii. Design an algorithm for the given problem statement and develop a flowchart/pseudo-code
- iii. Implement the algorithm in Python language
- iv. Execute the Python program
- v. Test the implemented program
- vi. Document the Results
- vii. Analyse and discuss the outcomes of the experiment

4. Questions

Problem statement: You are required to write a program that can generate routing tables for a network of routers. Take the number of nodes and the adjacency matrix as input from user. Your program should use this adjacency matrix and create routing tables for all the nodes in the network. The routing table should consist of one entry per destination. This entry should contain the total cost and the outgoing line to reach that destination.

Analysis: While analyzing your program, you are required to address the following points:

1. Why is Distance Vector Routing classified as an adaptive routing algorithm?
 - a. Limitations of Distance Vector Routing

5. Calculations/Computations/Algorithms

Algorithm: Distance Vector Routing

1. **Start**
2. **Repeat for each node in the graph:**
 - Take a node as the source node.
3. **Initialize distances:**
 - Initialize the distance to all other nodes as infinity (∞), except for the distance to the source node, which is 0.
4. **Create an array dist[]:**
 - Create an array dist[] of size |V| (number of vertices), where all values are set to infinity except for dist[src] = 0 (distance from source to itself).
5. **Iterate |V| - 1 times:**
 - Perform |V| - 1 iterations, where |V| is the number of vertices in the graph.
 - **For each edge u-v:**
 1. If dist[v] > dist[u] + weight of edge u-v, update dist[v] as follows:
 - dist[v] = dist[u] + weight of edge u-v
6. **Check for negative weight cycles:**
 - After completing |V| - 1 iterations, check all edges to see if there is still any edge where dist[v] > dist[u] + weight of edge u-v.
 1. If such an edge exists, report that "**Graph contains a negative weight cycle**" and terminate the algorithm.
7. **Compute the shortest path:**
 - The array dist[] will now contain the shortest path distances from the source node to all other nodes.
8. **Stop**

6. Program

```

main.c
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int findShortestPath(int G[6][6], int Vertices, int Edges, int edge[20][2], int Source) {
5      int i, j, u, v, k, distance[20], parent[20], flag = 1;
6
7      for (i = 0; i < Vertices; i++) {
8          distance[i] = 999;
9          parent[i] = -1;
10     }
11
12     distance[Source - 1] = 0;
13
14     for (i = 0; i < Vertices - 1; i++) {
15         for (k = 0; k < Edges; k++) {
16             u = edge[k][0];
17             v = edge[k][1];
18
19             if (distance[u] + G[u][v] < distance[v]) {
20                 distance[v] = distance[u] + G[u][v];
21                 parent[v] = u;
22             }
23         }
24     }
25
26     for (k = 0; k < Edges; k++) {
27         u = edge[k][0];
28         v = edge[k][1];
29
30         if (distance[u] + G[u][v] < distance[v]) {
31             flag = 0;
32             break;
33         }
34     }
35
36     printf("Destination\tCost\tPath\n");
37     if (flag) {
38         for (i = 0; i < Vertices; i++) {
39             int nodes[10], b = 9, next;
40             nodes[b--] = i;
41             next = parent[i];
42             nodes[b--] = next;
43
44             while (next != -1) {
45                 next = parent[next];
46                 nodes[b--] = next;
47             }
48
49             if (distance[i] == 999)
50                 printf("\n%d\t\t%d\t", i, nodes[-1]);
51             else
52                 printf("\n%d\t\t%d\t", i, distance[i]);
53
54             for (int h = 10; h > 0; h--) {
55                 if (nodes[h] == -1) {
56                     for (int g = h + 1; g < 10; g++) {
57                         if (g == 9)
58                             printf("%d", nodes[g]);
59                         else
60                             printf("%d-->", nodes[g]);
61                     }
62                     break;
63                 }
64             }
65         }
66     }
67     return flag;
68 }

```



```

70 int main() {
71     int Vertices = 6, edge[20][2];
72     int i, j, k = 0;
73
74     int graph[6][6] = {
75         {0, 1, 5, 0, 0, 0},
76         {1, 0, 3, 4, 0, 0},
77         {5, 3, 0, 5, 9, 0},
78         {0, 4, 5, 0, 2, 6},
79         {0, 0, 9, 2, 0, 3},
80         {0, 0, 0, 6, 3, 0}
81     };
82
83     printf("The Adjacency Matrix representation of graph\n");
84     for (i = 0; i < Vertices; i++) {
85         for (j = 0; j < Vertices; j++) {
86             printf("%d\t", graph[i][j]);
87             if (graph[i][j] != 0)
88                 edge[k][0] = i, edge[k++][1] = j;
89         }
90         printf("\n");
91     }
92
93     for (i = 0; i < Vertices; i++) {
94         printf("\n-----\n");
95         printf("\tSource vertex %d\n", i);
96         findShortestPath(graph, Vertices, k, edge, i + 1);
97     }
98     return 0;
99 }

```

7. Presentation of Results

The Adjacency Matrix representation of graph					
0	1	5	0	0	0
1	0	3	4	0	0
5	3	0	5	9	0
0	4	5	0	2	6
0	0	9	2	0	3
0	0	0	6	3	0

Source vertex 0		
Destination	Cost	Path
0	0	0
1	1	0-->1
2	4	0-->1-->2
3	5	0-->1-->3
4	7	0-->1-->3-->4
5	10	0-->1-->3-->4-->5

Source vertex 1		
Destination	Cost	Path
0	1	1-->0
1	0	1
2	3	1-->2
3	4	1-->3
4	6	1-->3-->4
5	9	1-->3-->4-->5

Source vertex 2		
Destination	Cost	Path
0	4	2-->1-->0
1	3	2-->1
2	0	2
3	5	2-->3
4	7	2-->3-->4
5	10	2-->3-->4-->5

Source vertex 3		
Destination	Cost	Path
0	5	3-->1-->0
1	4	3-->1
2	5	3-->2
3	0	3
4	2	3-->4
5	5	3-->4-->5

Source vertex 4		
Destination	Cost	Path
0	7	4-->3-->1-->0
1	6	4-->3-->1
2	7	4-->3-->2
3	2	4-->3
4	0	4
5	3	4-->5

Source vertex 5		
Destination	Cost	Path
0	10	5-->4-->3-->1-->0
1	9	5-->4-->3-->1
2	10	5-->4-->3-->2
3	5	5-->4-->3
4	3	5-->4
5	0	5

8. Analysis and Discussions

Why is Distance Vector Routing Adaptive?

Distance Vector Routing (DVR) is adaptive because:

1. **Dynamic Adjustments:** It automatically adjusts to changes in network topology (e.g., link failures).
2. **Periodic Updates:** Routers periodically exchange distance vectors, adapting to network conditions.
3. **Propagation of Information:** Routers update their paths based on neighbor information, minimizing the cost to reach destinations.

Limitations of Distance Vector Routing

1. **Slow Convergence:** Takes time to stabilize, especially in large networks.
2. **Count-to-Infinity Problem:** Vulnerable to routing loops when a link fails.
3. **Bandwidth Overhead:** Periodic updates create unnecessary traffic even without topology changes.
4. **Limited Knowledge:** Routers only know their neighbors, not the entire network.
5. **Scalability Issues:** Large networks require larger routing tables, leading to memory and performance problems.
6. **Incorrect Information:** Outdated or incorrect distance vectors can cause suboptimal routing.
7. **No Optimal Path Guarantee:** DVR may not always find the globally optimal path.

9. Conclusions

Understanding, Developing and Analyzing generation of routing tables for a network of routers using Distance Vector Routing.

10. Comments

1. Limitations of Experiments

- **Slow Convergence:** The algorithm takes longer to converge, especially in large networks.
- **Count-to-Infinity Problem:** The algorithm is vulnerable to count-to-infinity issues during network instability.
- **Traffic Overhead:** The periodic updates of hop counts result in unnecessary bandwidth usage, even when there are no changes in the network topology.
- **Larger Routing Tables:** In larger networks, routers maintain routing tables with information about all other routers, leading to increased memory usage and possible congestion on WAN links.

2. Limitations of Results

- **Network-Specific:** The results are tied to the specific network topology used in the experiment and do not account for variations in real-world networks.
- **Unverified for Negative Cycles:** The algorithm was not tested on graphs that include negative weight cycles, which may result in incorrect results.

3. Learning happened

- Gained an understanding of **Distance Vector Routing**, its advantages, and limitations in terms of network performance and scalability.

Laboratory 5

Title of the Laboratory Exercise: ARQ Mechanisms in DLL

1. Introduction and Purpose of Experiment

The purpose of this experiment is to delve into Automatic Repeat reQuest (ARQ) mechanisms, a crucial aspect of the Data Link Layer (DLL) in computer networks. ARQ protocols are employed to ensure reliable data transmission by detecting and correcting errors that may occur during the communication process. In this lab, students will gain hands-on experience in implementing and simulating various ARQ techniques, such as Stop-and-Wait, Go-Back-N, and Selective Repeat. They will learn how these protocols operate, how they handle retransmissions, and their impact on network performance. Through practical exercises and simulations, students will develop a comprehensive understanding of ARQ mechanisms, which are vital for achieving reliable data communication in networks. This lab paves the way for further exploration of error control mechanisms and protocols at the Data Link Layer.

2. Aim and Objectives

Aim

- To implement receiver algorithms for the different ARQ mechanisms at the Data Link Layer

Objectives

At the end of this lab, the student will be able to

- Implement receiver algorithms for the different ARQ mechanisms at the Data Link Layer
- Analyse the differences between the ARQ mechanisms

3. Experimental Procedure

- i. Analyse the problem statement
- ii. Design an algorithm for the given problem statement and develop a flowchart/pseudo-code
- iii. Implement the algorithm in Python language
- iv. Execute the Python program
- v. Test the implemented program
- vi. Document the Results
- vii. Analyse and discuss the outcomes of the experiment

4. Questions

Problem statement: You are required to write a program that can receive frames at the data link layer. Assume that the user is entering the frames as the transmitter. You are required to implement stop and wait, go back N and selective repeat ARQ mechanisms. Consider that you have to transmit and receive a total of 20 frames using $W_T=W_R=1$, $W_T=5$ and $W_R=1$ and $W_T=W_R=5$ for stop and wait, go back N and selective repeat respectively

Analysis: While analyzing your program, you are required to address the following points:

- Difference between stop and wait, go back N and selective repeat.
- Comparison of the disadvantages of the different ARQ mechanisms.

5. Calculations/Computations/Algorithms

Algorithm for Stop and Wait

- Step 1: Start
- Step 2: Take total number of frames from user
- Step 3: Send frames one by one
- Step 4: Ask for user whether respective frame is received or not
- Step 5: If not send negative ACK to sender and if it is then send ACK to sender
- Step 6: Stop

Algorithm for Go back N

- Step 1: Start
- Step 2: Take total number of frames from user
- Step 3: Take window size from user
- Step 4: Send frames in window, if frame is received then slide the window and send all frames in window
- Step 5: If frame is not received then send frames of window which comes after the frame is lost.
- Step 6: Stop

Algorithm for Selective Repeat

- Step 1: Start
- Step 2: Take total number of frames from user
- Step 3: Send all frames
- Step 4: Take number of frames received
- Step 5: Also take the which frames are received
- Step 6: Send only frames which are not received
- Step 7: Stop

6. Program

```

main.c
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void StopAndWait();
5  void GoBackN();
6  void SelectiveRepeat();
7
8  int main(int argc, char** argv) {
9      int val;
10     while (1) {
11         printf("\nEnter:\n1. Stop And Wait protocol\n2. Go back N protocol\n3. Selective Repeat protocol\n");
12         scanf("%d", &val);
13         switch (val) {
14             case 1: StopAndWait(); break;
15             case 2: GoBackN(); break;
16             case 3: SelectiveRepeat(); break;
17             default: return 0;
18         }
19     }
20     return EXIT_SUCCESS;
21 }
22
23 // Function to implement Stop and Wait ARQ algorithm
24 void StopAndWait() {
25     int n, i = 0, frame;
26     printf("Enter the total number of frames: ");
27     scanf("%d", &n);
28     while (i != n) {
29         printf("\nEnter received frame: ");
30         scanf("%d", &frame);
31         if (frame == i + 1) {
32             printf("Transmitting..... ACK to frame %d\n", frame);
33             i++;
34         } else {
35             printf("Negative ACK.... to frame %d\n", i + 1);
36         }
37     }
38 }
39
40 // Function to implement Go back N algorithm
41 void GoBackN() {
42     int n, i = 0, frame, size, t;
43     printf("Enter the total number of frames: ");
44     scanf("%d", &n);
45     printf("Enter window size of frames: ");
46     scanf("%d", &size);
47     printf("Sending frames: ");
48     for (int j = 0; j < size; j++) {
49         printf("%d, ", j + 1);
50     }
51     for (int j = 0; j < n; j++) {
52         printf("\nIs frame %d received (1 or 0): ", j + 1);
53         scanf("%d", &t);
54         if (t == 1) {
55             printf("Sending ACK to frame %d\n", j + 1);
56             printf("Sliding window: ");
57             for (int k = j + 1; k < j + 1 + size; k++) {
58                 if (k < n)
59                     printf("%d, ", k + 1);
60             }
61         } else {
62             printf("\nRetransmitting frames: ");
63             for (int m = j; m < j + size; m++) {
64                 if (m < n)
65                     printf("%d, ", m + 1);
66             }
67         }
68     }
69 }
70
71 // Function to implement Selective Repeat algorithm
72 void SelectiveRepeat() {
73     int n, m, size;
74     printf("Enter the total number of frames: ");
75     scanf("%d", &n);
76     printf("Sending frames: ");
77     for (int j = 0; j < n; j++) {
78         printf("%d, ", j + 1);
79     }
80     printf("\nEnter the number of frames received: ");
81     scanf("%d", &m);

```

```

82     int a[n];
83     printf("\nEnter received frames: ");
84     for (int i = 0; i < m; i++) {
85         scanf("%d", &a[i]);
86     }
87     for (int i = 0; i < n; i++) {
88         int trigger = 0;
89         for (int j = 0; j < m; j++) {
90             if (i + 1 == a[j]) {
91                 trigger = 1;
92                 break;
93             }
94         }
95         if (!trigger) {
96             printf("Retransmitting frame %d\n", i + 1);
97             a[m++] = i + 1;
98         }
99     }
100    // Sorting the received frames
101    for (int i = 0; i < m - 1; i++) {
102        for (int j = i + 1; j < m; j++) {
103            if (a[i] > a[j]) {
104                int temp = a[i];
105                a[i] = a[j];
106                a[j] = temp;
107            }
108        }
109    }
110    printf("\nSorted frames: ");
111    for (int i = 0; i < m; i++) {
112        printf("%d ", a[i]);
113    }
114    printf("\n");
115 }

```

7. Presentation of Results

Stop and Wait

```

Enter:
1. Stop And Wait protocol
2. Go back N protocol
3. Selective Repeat protocol
1
Enter the total number of frames: 5

Enter received frame: 1
Transmitting..... ACK to frame 1

Enter received frame: 3
Negative ACK.... to frame 2

Enter received frame: 2
Transmitting..... ACK to frame 2

Enter received frame: 3
Transmitting..... ACK to frame 3

Enter received frame: 5
Negative ACK.... to frame 4

Enter received frame: 4
Transmitting..... ACK to frame 4

Enter received frame: 5
Transmitting..... ACK to frame 5

```

Go back N algorithm

```
Enter:
1. Stop And Wait protocol
2. Go back N protocol
3. Selective Repeat protocol
2
Enter the total number of frames: 5
Enter window size of frames: 3
Sending frames: 1, 2, 3,
Is frame 1 received (1 or 0): 1
Sending ACK to frame 1
Sliding window: 2, 3, 4,
Is frame 2 received (1 or 0): 0

Retransmitting frames: 2, 3, 4,
Is frame 3 received (1 or 0): 2

Retransmitting frames: 3, 4, 5,
Is frame 4 received (1 or 0): 1
Sending ACK to frame 4
Sliding window: 5,
Is frame 5 received (1 or 0): 0

Retransmitting frames: 5,
```

Selective Repeat Algorithm

```
Enter:
1. Stop And Wait protocol
2. Go back N protocol
3. Selective Repeat protocol
3
Enter the total number of frames: 10
Sending frames: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
Enter the number of frames received: 4

Enter received frames: 2
8
1
3
Retransmitting frame 4
Retransmitting frame 5
Retransmitting frame 6
Retransmitting frame 7
Retransmitting frame 9
Retransmitting frame 10

Sorted frames: 1 2 3 4 5 6 7 8 9 10
```


8. Analysis and Discussions

Difference Between Stop-and-Wait, Go-Back-N, and Selective Repeat ARQ

1. Stop-and-Wait ARQ:

- Sends one frame at a time, waits for an acknowledgment (ACK).
- Low efficiency, minimal buffer needed.

2. Go-Back-N ARQ:

- Sends multiple frames, retransmits all frames after an error.
- Higher efficiency, requires sender buffer for N frames.

3. Selective Repeat ARQ:

- Sends multiple frames, retransmits only erroneous frames.
- Most efficient but complex, large buffers needed for reordering.

Disadvantages of ARQ Mechanisms

1. Stop-and-Wait:

- Low throughput, inefficient for high-latency networks.

2. Go-Back-N:

- Wastes bandwidth by retransmitting redundant frames after errors.

3. Selective Repeat:

- Complex implementation, requires large buffers for out-of-order frames.

9. Conclusions

C program for ARQ mechanism is developed and verified with result.

10. Comments

1. Limitations of Experiments

- Only simple logic is used may not work in complex cases.

2. Limitations of Results

- Results are based on user input for particular ARQ function.

3. Learning happened

- Learnt to create simple program to implement ARQ

4. Recommendations

- This program further modified like adding frame sequence number and Ack type.

Laboratory 6

Title of the Laboratory Exercise: Socket Programming-I

1. Introduction and Purpose of Experiment

The objective of this experiment is to introduce students to socket programming, a fundamental concept in network communication. Sockets provide a powerful mechanism for processes to communicate over a network, enabling data exchange between devices. In this lab, students will learn how to create and use sockets for establishing network connections. They will gain hands-on experience in developing basic client-server applications, allowing them to send and receive data over a network. Through practical exercises, students will become proficient in utilizing socket APIs to implement simple networked applications. This lab lays the foundation for more advanced socket programming techniques and network application development in subsequent labs.

2. Aim and Objectives

Aim

- To use TCP Sockets for Inter Process Communication

Objectives

At the end of this lab, the student will be able to

- Apply TCP Socket programming technique to establish IPC between remote processes
- Analyse the difference between sockets and other enabling techniques for IPC such as Pipes and Message Queues

3. Experimental Procedure

- i. Analyse the problem statement
- ii. Design an algorithm for the given problem statement and develop a flowchart/pseudo-code
- iii. Implement the algorithm in Python language
- iv. Execute the Python program
- v. Test the implemented program
- vi. Document the Results
- vii. Analyse and discuss the outcomes of the experiment

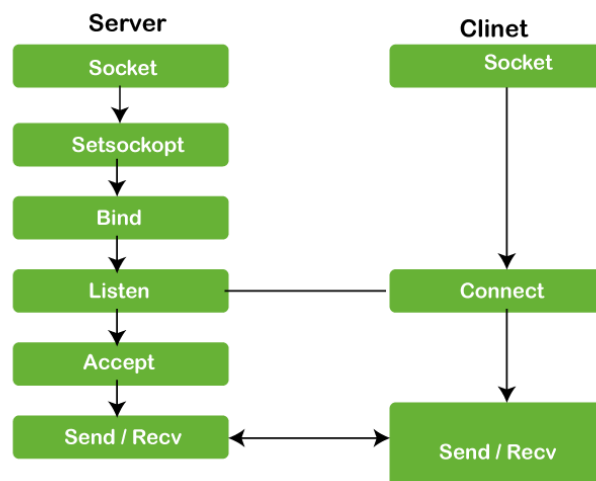
4. Questions

Problem statement: You are required to write programs to implement a TCP based echo server. The functionality of this server is that it should echo any data it receives from a client back to it.

Analysis: While analyzing your program, you are required to address the following points:

1. How is socket programming different from other techniques for IPC such as Pipes and Message Queues?
 - a. What happens if the number of incoming client requests exceeds the second argument of the `listen()` function in the server?

5. Calculations/Computations/Algorithms



6. Program

Server :-

```

server.py - Notepad
File Edit Format View Help
import time, socket, sys
print('Setup Server...')
time.sleep(1)
# Get the hostname, IP Address from socket and set Port
soc = socket.socket()
host_name = socket.gethostname()
ip = socket.gethostbyname(host_name)
port = 1234
soc.bind((host_name, port))
print(host_name, '({})'.format(ip))
name = input('Enter name: ')
soc.listen(1) # Try to locate using socket
print('Waiting for incoming connections...')
connection, addr = soc.accept()
print("Received connection from ", addr[0], "(", addr[1], ")\n")
print('Connection Established. Connected From: {}, {}'.format(addr[0], addr[1]))
# get a connection from client side
client_name = connection.recv(1024)
client_name = client_name.decode()
print(client_name + ' has connected.')
print('Press [bye] to leave the chat room')
connection.send(name.encode())
  
```

```

while True:
    message = input('Me > ')
    if message == '[bye]':
        message = 'Bye, leaving the chat room!'
        connection.send(message.encode())
        print('\n')
        break

    connection.send(message.encode())
    message = connection.recv(1024)
    message = message.decode()
    print(client_name, '>', message)

```

Client :-

client.py - Notepad

File Edit Format View Help

```

import time, socket, sys
print('Client Server...')
time.sleep(1)
#Get the hostname, IP Address from socket and set Port
soc = socket.socket()
shost = socket.gethostname()
ip = socket.gethostbyname(shost)
#get information to connect with the server
print(shost, '({})'.format(ip))
server_host = input('Enter server\'s IP address:')
name = input('Enter Client\'s name: ')
port = 1234
print('Trying to connect to the server: {}, ({}).format(server_host, port))
time.sleep(1)
soc.connect((server_host, port))
print("Connected...\n")
soc.send(name.encode())
server_name = soc.recv(1024)
server_name = server_name.decode()
print('{} has joined...'.format(server_name))
print('Enter [bye] to exit.')

while True:
    message = soc.recv(1024)
    message = message.decode()
    print(server_name, ">", message)
    message = input("Me > ")
    if message == '[bye]':
        message = "Leaving the Chat room"
        soc.send(message.encode())
        print("\n")
        break
    soc.send(message.encode())

```

7. Presentation of Results

Server :-

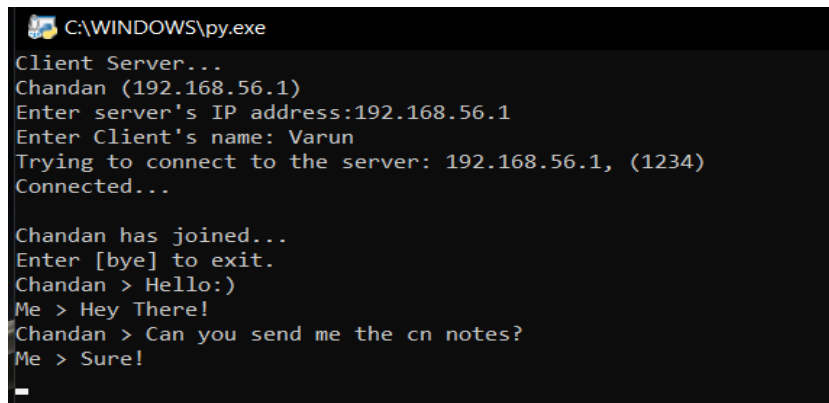
```

C:\WINDOWS\py.exe
Setup Server...
Chandan (192.168.56.1)
Enter name: Chandan
Waiting for incoming connections...
Received connection from 192.168.56.1 ( 54284 )

Connection Established. Connected From: 192.168.56.1, (192.168.56.1)
Varun has connected.
Press [bye] to leave the chat room
Me > Hello:)
Varun > Hey There!
Me > Can you send me the cn notes?
Varun > Sure!
Me >

```

Client :-



```
C:\WINDOWS\py.exe
Client Server...
Chandan (192.168.56.1)
Enter server's IP address:192.168.56.1
Enter Client's name: Varun
Trying to connect to the server: 192.168.56.1, (1234)
Connected...

Chandan has joined...
Enter [bye] to exit.
Chandan > Hello:)
Me > Hey There!
Chandan > Can you send me the cn notes?
Me > Sure!
```

8. Analysis and Discussions

Socket programming is more suitable for network-based communication, especially when interacting across different machines, whereas Pipes and Message Queues are better for local IPC within the same machine.

If the number of incoming client requests exceeds the backlog size specified in the second argument of `listen()`, additional requests are dropped. These clients may receive a "connection refused" error. To handle this, you can increase the backlog size, implement load balancing, or optimize the server to process connections faster.

9. Conclusions

Understanding and implementing the technique of socket programming

10. Comments

1. Limitations of Experiments

- Limited test scenarios and resource constraints impacted reliability.

2. Limitations of Results

- Results may lack generalizability and fail to cover edge cases.

3. Learning happened

- Gained hands-on experience with TCP sockets and connection management.

4. Recommendations

- Conduct stress testing, improve error handling, and optimize server scalability.

Laboratory 7

Title of the Laboratory Exercise: DLL ARQ Mechanisms using TCP Sockets

1. Introduction and Purpose of Experiment

This lab combines Data Link Layer ARQ mechanisms with TCP sockets. Students will integrate error control with TCP for reliable data transmission. They'll gain hands-on experience in implementing ARQ techniques within a TCP socket environment, learning to handle retransmissions and acknowledgments. This practical exercise builds a strong foundation for advanced network application development.

2. Aim and Objectives

Aim

- To use TCP Sockets to implement the ARQ mechanisms at the Data Link Layer

Objectives

At the end of this lab, the student will be able to

- Apply TCP Socket programming technique to implement the ARQ mechanisms at the Data Link Layer

3. Experimental Procedure

- Analyse the problem statement
- Design an algorithm for the given problem statement and develop a flowchart/pseudo-code
- Implement the algorithm in Python language
- Execute the Python program
- Test the implemented program
- Document the Results
- Analyse and discuss the outcomes of the experiment

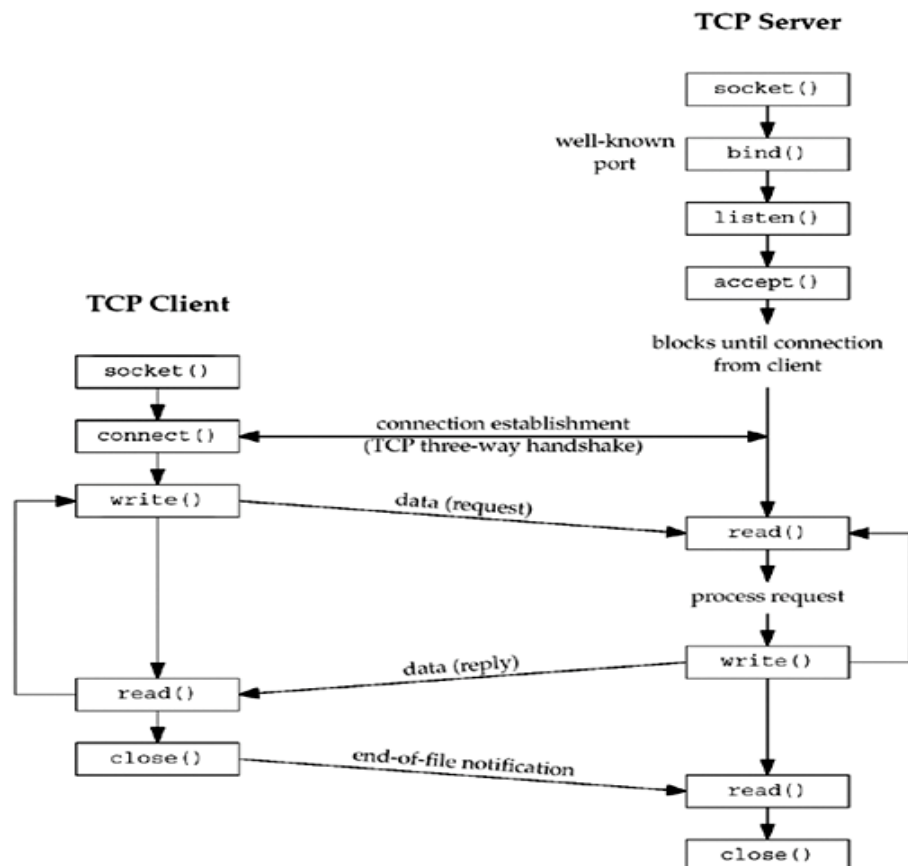
4. Questions

Problem statement: You are required to write programs to implement a TCP based server that receives frames sent to it by a client. The functionality of this server is that it should echo any data it receives from a client back to it. You are required to implement stop and wait, go back N and selective repeat ARQ mechanisms. Consider that you have to transmit and receive a total of 20 frames using $W_T=W_R=1$, $W_T=5$ and $W_R=1$ and $W_T=W_R=5$ for stop and wait, go back N and selective repeat respectively.

Analysis: While analyzing your program, you are required to address the following points:

- How does the functionality of the program differ when you have the `accept()` function call at the server within an infinite loop as opposed to having it outside?

5. Calculations/Computations/Algorithms



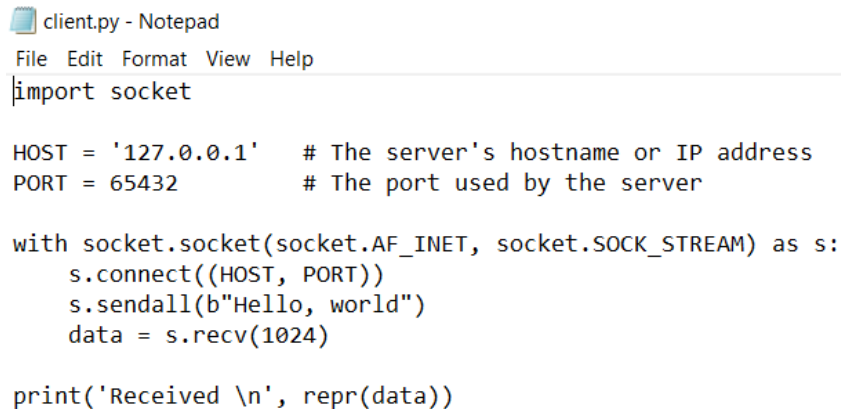
6. Program

Server :-

```
server.py - Notepad
File Edit Format View Help
import socket

HOST = '127.0.0.1' # Standard loopback interface address (localhost)
PORT = 65432       # Port to listen on (non-privileged ports are > 1023)

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.bind((HOST, PORT))
    s.listen()
    conn, addr = s.accept()
    with conn:
        print('Connected by', addr)
        while True:
            data = conn.recv(1024)
            if not data:
                break
            conn.sendall(data)|
```

Client :-


```

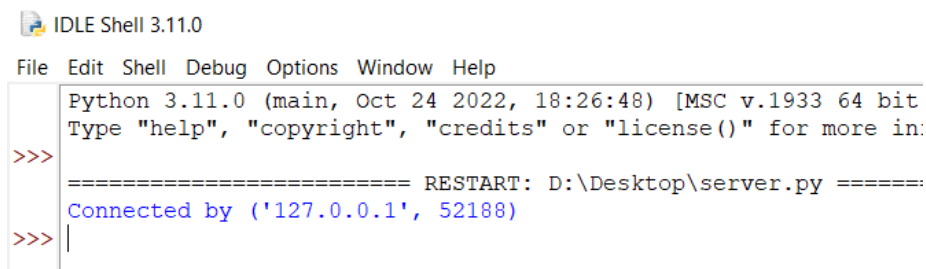
client.py - Notepad
File Edit Format View Help
import socket

HOST = '127.0.0.1'    # The server's hostname or IP address
PORT = 65432         # The port used by the server

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.connect((HOST, PORT))
    s.sendall(b"Hello, world")
    data = s.recv(1024)

print('Received \n', repr(data))

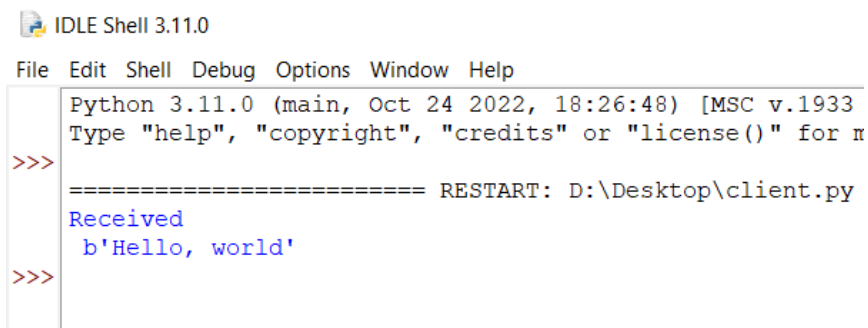
```

7. Presentation of Results**Server :-**


```

IDLE Shell 3.11.0
File Edit Shell Debug Options Window Help
Python 3.11.0 (main, Oct 24 2022, 18:26:48) [MSC v.1933 64 bit
Type "help", "copyright", "credits" or "license()" for more in:
>>>
===== RESTART: D:\Desktop\server.py =====
Connected by ('127.0.0.1', 52188)
>>>

```

Client :-


```

IDLE Shell 3.11.0
File Edit Shell Debug Options Window Help
Python 3.11.0 (main, Oct 24 2022, 18:26:48) [MSC v.1933
Type "help", "copyright", "credits" or "license()" for m
>>>
===== RESTART: D:\Desktop\client.py =====
Received
b'Hello, world'
>>>

```

8. Analysis and Discussions

- Use `accept()` inside an infinite loop for a robust server capable of handling multiple clients or continuous connections.
- Use `accept()` outside the loop for simple applications or single-client scenarios.

9. Conclusions

- Gained understanding of socket programming and the impact of `accept()` placement on server behaviour.

10. Comments

1. Limitations of Experiments

- The experiment was limited to single-client testing and didn't account for real-world conditions like network delays.

2. Limitations of Results

- The results don't cover performance under load, multiple simultaneous connections, or edge cases.

3. Learning happened

- Gained understanding of socket programming and the impact of `accept()` placement on server behaviour.

4. Recommendations

- Perform stress tests, implement multi-threading, and simulate real-world network conditions for better robustness.