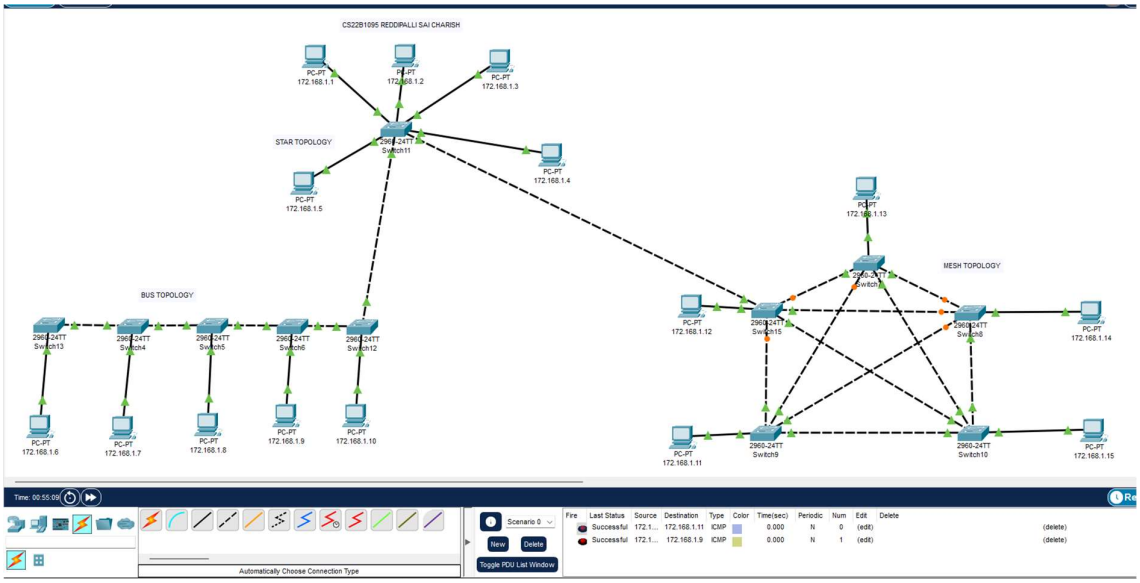
COMPUTER NETWORKS
LAB ASSIGNMENT-03

REDDIPALLI SAI CHARISH

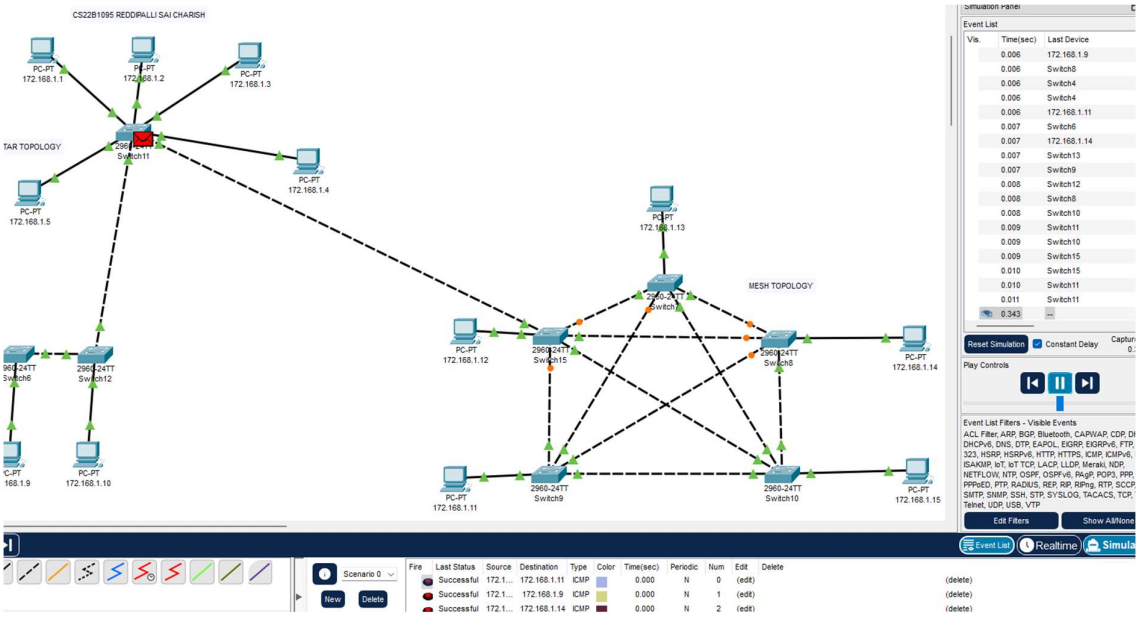
CS22B1095

QUESTION 1:

(i)Network Diagram



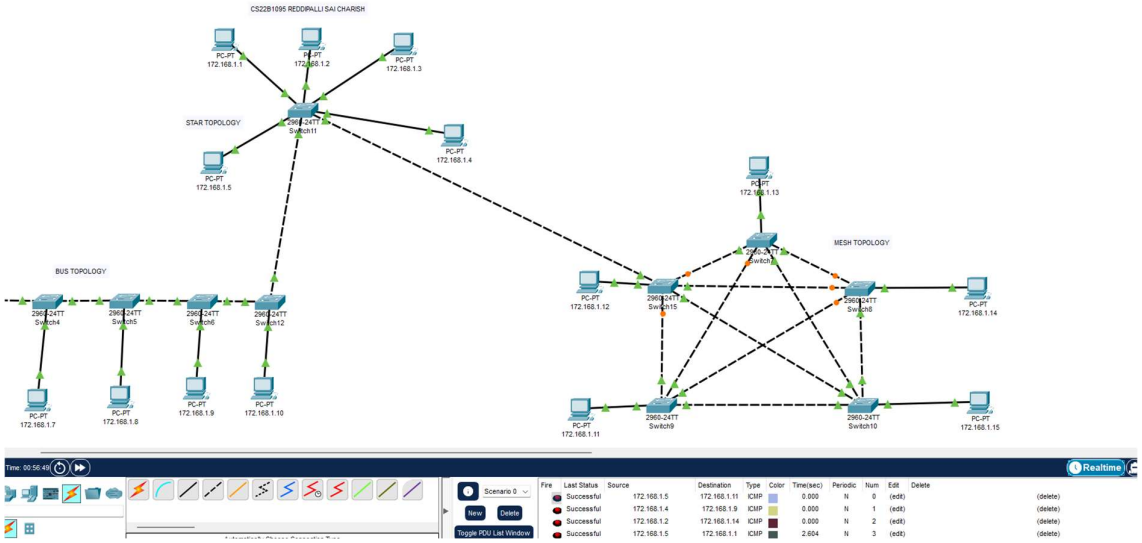
Simulation :



Packet Transfer Successful:

Fire	Last Status	Source	Destination	Type	Color	Time(sec)	Periodic	Num	Edit	Delete
	Successful	172.1...	172.168.1.11	ICMP		0.000	N	0	(edit)	(delete)
	Successful	172.1...	172.168.1.9	ICMP		0.000	N	1	(edit)	(delete)
	Successful	172.1...	172.168.1.14	ICMP		0.000	N	2	(edit)	(delete)

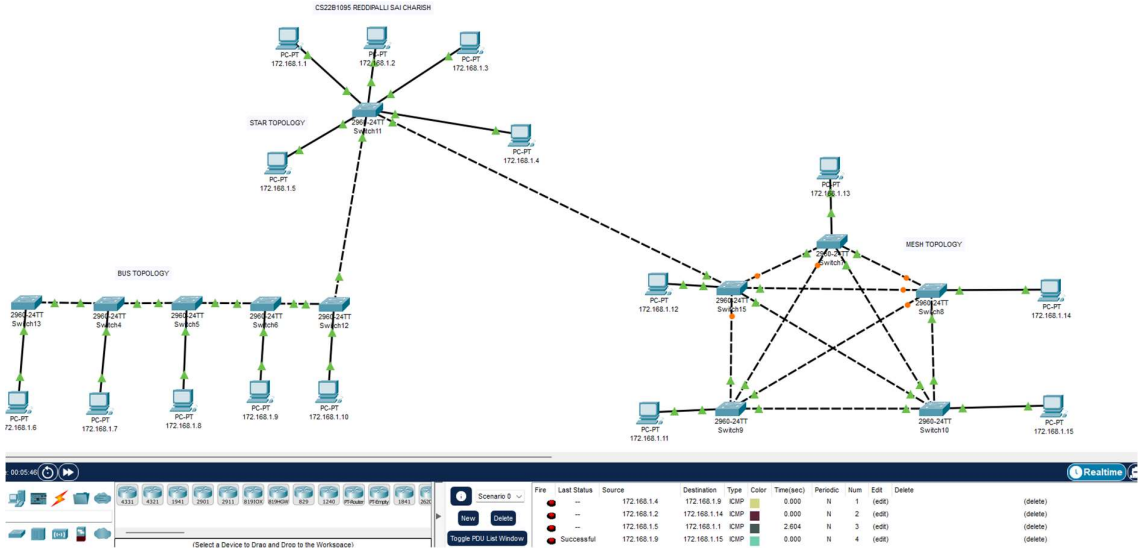
(ii)Intra topology and inter topology Packet transfer :



Star topology (172.168.1.5 → 172.168.1.1) -Intra TOPOLOGY

Star Topology → Bus Topology (172.168.1.4→ 172.168.1.9) - Inter Topology

Star Topology → Mesh Topology (172.168.1.5→ 172.168.1.11) - Inter Topology



Bus topology → Mesh Topology (172.168.1.9→172.168.1.15)-Inter topology

(iii) Analysing the Performance and Fault Tolerance:

TOPOLOGY	Performance	TOLERANCE
STAR	Good performance since each device is connected to a central hub; the hub manages data flow efficiently. However, performance may degrade if the central hub is overloaded.	Low fault tolerance. If the central hub fails, the entire network goes down. However, failure of a single device doesn't affect the rest of the network.
BUS	Moderate performance; as the number of devices increases, the performance may degrade due to data collisions.	Low fault tolerance. If the main cable (bus) fails, the entire network is disrupted. Failure of individual devices doesn't impact the rest of the network, but data collision issues can arise as more devices are added.
MESH	Excellent performance, as each device is connected to multiple other devices, providing multiple paths for data to travel.	High fault tolerance. If one connection or device fails, data can still be transmitted through other connections, ensuring network continuity.

Question 2:

TCP SERVER:

```
//CS22B1095

//REDDIPALLI SAI CHARISH

#include <netinet/in.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/socket.h>

#include <unistd.h>

#define PORT 8080

#define BUFFER_SIZE 1024

#define BACKLOG 5

int main() {

  int server_socket, client_socket;

  struct sockaddr_in server_addr, client_addr;

  socklen_t addr_len = sizeof(client_addr);

  char buffer[BUFFER_SIZE];

  char *response = "Hello from server!";

 // 1. Create a TCP socket

  server_socket = socket(AF_INET, SOCK_STREAM, 0);

  if (server_socket == 0) {

    perror("Socket failed");

    exit(EXIT_FAILURE);

  }


  // 2. Define server address and port

  server_addr.sin_family = AF_INET;

  server_addr.sin_addr.s_addr = INADDR_ANY;

  server_addr.sin_port = htons(PORT);


  // 3. Bind the socket to an IP address and port

  if (bind(server_socket, (struct sockaddr *)&server_addr, sizeof(server_addr)) < 0) {

    perror("Bind failed");

    close(server_socket);

    exit(EXIT_FAILURE);

  }
```

```
// 4. Listen for incoming connections
if (listen(server_socket, BACKLOG) < 0) {
    perror("Listen failed");
    close(server_socket);
    exit(EXIT_FAILURE);
}

printf(" Tcp Server is listening on port %d...\n", PORT);

// 5. Accept a connection from a client
client_socket = accept(server_socket, (struct sockaddr *)&client_addr, &addr_len);
if (client_socket < 0) {
    perror("Accept failed");
    close(server_socket);
    exit(EXIT_FAILURE);
}

// 6. Receive data from the client
int bytes_received = recv(client_socket, buffer, BUFFER_SIZE, 0);
if (bytes_received < 0) {
    perror("Receive failed");
    close(client_socket);
    close(server_socket);
    exit(EXIT_FAILURE);
}
buffer[bytes_received] = '\0';  // Null-terminate the received string
printf("Received from client: %s\n", buffer);

// 7. Send data to the client
if (send(client_socket, response, strlen(response), 0) < 0) {
    perror("Send failed");
    close(client_socket);
    close(server_socket);
    exit(EXIT_FAILURE);
}

printf("Response sent to client.\n");
```

```
    // 8. Close the client socket
    close(client_socket);

    // 9. Close the server socket
    close(server_socket);

    return 0;
}
```

TCP CLIENT:

```
//CS22B1095
//REDDIPALLI SAI CHARISH
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <unistd.h>
#include <arpa/inet.h>

#define PORT 8080
#define BUFFER_SIZE 1024

int main() {
    int client_socket;
    struct sockaddr_in server_addr;
    char *message = "Hello from client!";
    char buffer[BUFFER_SIZE] = {0};

    // 1. Create a TCP socket
    client_socket = socket(AF_INET, SOCK_STREAM, 0);
    if (client_socket < 0) {
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
    }

    // 2. Define server address and port
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(PORT);
```

```
    // Convert IPv4 address from text to binary form
    if (inet_pton(AF_INET, "127.0.0.1", &server_addr.sin_addr) <= 0) {
        perror("Invalid address / Address not supported");
        close(client_socket);
        exit(EXIT_FAILURE);
    }

    // 3. Connect to the server
    if (connect(client_socket, (struct sockaddr *)&server_addr, sizeof(server_addr)) < 0) {
        perror("Connection failed");
        close(client_socket);
        exit(EXIT_FAILURE);
    }

    // 4. Send data to the server
    if (send(client_socket, message, strlen(message), 0) < 0) {
        perror("Send failed");
        close(client_socket);
        exit(EXIT_FAILURE);
    }
    printf("Message sent to server: %s\n", message);

    // 5. Receive data from the server
    int bytes_received = recv(client_socket, buffer, BUFFER_SIZE, 0);
    if (bytes_received < 0) {
        perror("Receive failed");
        close(client_socket);
        exit(EXIT_FAILURE);
    }
    buffer[bytes_received] = '\0';  // Null-terminate the received string
    printf("Received from server: %s\n", buffer);

    // 6. Close the socket
    close(client_socket);

    return 0;
}
```

OUTPUTS:

```
charish@charish-Nitro-AN515-57:~/Desktop/Computer Networks $ gcc -o a tcp_server.c
charish@charish-Nitro-AN515-57:~/Desktop/Computer Networks $ ./a
 Tcp Server is listening on port 8080...
```

```
charish@charish-Nitro-AN515-57:~/Desktop/Computer Networks $ gcc -o b tcp_client.c
charish@charish-Nitro-AN515-57:~/Desktop/Computer Networks $ ./b
Message sent to server: Hello from client!
Received from server: Hello from server!
charish@charish-Nitro-AN515-57:~/Desktop/Computer Networks $
```

```
charish@charish-Nitro-AN515-57:~/Desktop/Computer Networks $ gcc -o a tcp_server.c
charish@charish-Nitro-AN515-57:~/Desktop/Computer Networks $ ./a
 Tcp Server is listening on port 8080...
Received from client: Hello from client!
Response sent to client.
```

UDP SERVER:

```
//CS22B1095
//REDDIPALLI SAI CHARISH
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <unistd.h>

#define PORT 8080
#define BUFFER_SIZE 1024

int main() {
   int server_socket;
   struct sockaddr_in server_addr, client_addr;
   char buffer[BUFFER_SIZE];
   char *response = "Hello from server!";
   socklen_t addr_len = sizeof(client_addr);

   // 1. Create a UDP socket
   server_socket = socket(AF_INET, SOCK_DGRAM, 0);
   if (server_socket < 0) {
      perror("Socket creation failed");
      exit(EXIT_FAILURE);
   }

   // 2. Define server address and port
   server_addr.sin_family = AF_INET;
   server_addr.sin_addr.s_addr = INADDR_ANY;
   server_addr.sin_port = htons(PORT);

   // 3. Bind the socket to an IP address and port
   if (bind(server_socket, (struct sockaddr *)&server_addr, sizeof(server_addr)) < 0) {
      perror("Bind failed");
      close(server_socket);
      exit(EXIT_FAILURE);
   }
```

```
    printf("UDP server is up and listening on port %d...\n", PORT);

    // 4. Receive data from a client
    int bytes_received = recvfrom(server_socket, buffer, BUFFER_SIZE, 0, (struct sockaddr *)&client_addr, &addr_len);
    if (bytes_received < 0) {
        perror("Receive failed");
        close(server_socket);
        exit(EXIT_FAILURE);
    }
    buffer[bytes_received] = '\0';  // Null-terminate the received string
    printf("Received from client: %s\n", buffer);

    // 5. Send data to the client
    if (sendto(server_socket, response, strlen(response), 0, (struct sockaddr *)&client_addr, addr_len) < 0) {
        perror("Send failed");
        close(server_socket);
        exit(EXIT_FAILURE);
    }

    printf("Response sent to client.\n");

    // 6. Close the socket
    close(server_socket);

    return 0;
}
```

UDP CLIENT:

```
//CS22B1095

//REDDIPALLI SAI CHARISH

#include <arpa/inet.h>

#include <netinet/in.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/socket.h>

#include <unistd.h>


#define PORT 8080

#define BUFFER_SIZE 1024


int main() {

  int client_socket;

  struct sockaddr_in server_addr;

  char *message = "Hello from client!";

  char buffer[BUFFER_SIZE];

  socklen_t addr_len = sizeof(server_addr);


  // 1. Create a UDP socket

  client_socket = socket(AF_INET, SOCK_DGRAM, 0);

  if (client_socket < 0) {

    perror("Socket creation failed");

    exit(EXIT_FAILURE);

  }


  // 2. Define server address and port

  server_addr.sin_family = AF_INET;

  server_addr.sin_port = htons(PORT);


  // Convert IPv4 address from text to binary form

  if (inet_pton(AF_INET, "127.0.0.1", &server_addr.sin_addr) <= 0) {

    perror("Invalid address / Address not supported");

    close(client_socket);

    exit(EXIT_FAILURE);

  }
```

```
    // 3. Send data to the server
    if (sendto(client_socket, message, strlen(message), 0, (struct sockaddr *)&server_addr, addr_len) < 0) {
        perror("Send failed");
        close(client_socket);
        exit(EXIT_FAILURE);
    }
    printf("Message sent to server: %s\n", message);

    // 4. Receive data from the server
    int bytes_received = recvfrom(client_socket, buffer, BUFFER_SIZE, 0, NULL, NULL);
    if (bytes_received < 0) {
        perror("Receive failed");
        close(client_socket);
        exit(EXIT_FAILURE);
    }
    buffer[bytes_received] = '\0';  // Null-terminate the received string
    printf("Received from server: %s\n", buffer);

    // 5. Close the socket
    close(client_socket);

    return 0;
}
```

OUTPUT: