TCP

ALGORITHM

TCP Server –

1. using create(), Create TCP socket.
2. using bind(), Bind the socket to server address.
3. using listen(), put the server socket in a passive mode, where it waits for the client to approach the server to make a connection
4. using accept(), At this point, connection is established between client and server, and they are ready to transfer data.
5. Go back to Step 3.

```c
#include <stdio.h>
#include <string.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
void main()
{
    struct sockaddr_in client, server;
    int s, n, sock;
    char b1[10] = "", b2[10] = "Hello";
    s = socket(AF_INET, SOCK_STREAM, 0);
    server.sin_family = AF_INET;
    server.sin_port = 2000;
    server.sin_addr.s_addr = inet_addr("127.0.0.1");
    bind(s, (struct sockaddr *)&server, sizeof server);
    listen(s, 1);
    n = sizeof client;
    sock = accept(s, (struct sockaddr *)&client, &n);
    for (;;)
    {
        recv(sock, b1, sizeof b1, 0);
        if (strcmp(b1, "end") == 0)
            break;
        printf("\nClient:%s", b1);
        printf("\nserver:");
        scanf("%s", b2);
        send(sock, b2, sizeof b2, 0);
        if (strcmp(b2, "end") == 0)
            break;
```

```
    }
    close(sock);
    close(s);
}
```

TCP Client –

1. Create TCP socket.
2. Connect newly created client socket to server.

```
#include <unistd.h>
#include <stdio.h>
#include <string.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
void main()
{
    struct sockaddr_in client, server;
    int s, sock;
    char b1[10] = "", b2[10] = "Hello";
    s = socket(AF_INET, SOCK_STREAM, 0);
    server.sin_family = AF_INET;
    server.sin_port = 2000;
    server.sin_addr.s_addr = inet_addr("127.0.0.1");
    connect(s, (struct sockaddr *)&server, sizeof server);
    for (;;)
    {
        printf("\nClient:");
        scanf("%s", b2);
        send(s, b2, sizeof b2, 0);
        if (strcmp(b2, "end") == 0)
            break;
        recv(s, b1, sizeof b1, 0);
        if (strcmp(b1, "end") == 0)
            break;
        printf("\nserver:%s", b1);
    }
    close(s);
}
```

UDP Server:

Server

1. Create a UDP socket.
2. Bind the socket with the proper IP (Internet Protocol) adress and the port number.
3. Wait for the datagram packet from the client.
4. Process the datagram and send the reply.
5. Finish.

```
#include<arpa/inet.h>
#include<netinet / in.h>
#include<stdio.h>
#include<string.h>
#include<sys / socket.h>
#include<sys / stat.h>
#include<sys / types.h> void main()
{
    struct
        sockaddr_in
            server,
        client;
    int s, n;
    char b1[10], b2[10];
    s = socket(AF_INET, SOCK_DGRAM, 0);
    server.sin_family = AF_INET;
    server.sin_port = 3000;
    server.sin_addr.s_addr = inet_addr("127.0.0.1");
    bind(s, (struct sockaddr *)&server, sizeof(server));
    n = sizeof(client);
    while (1)
    {
        recvfrom(s, b1, sizeof(b1), 0, (struct sockaddr *)&client, &n);
        if (!(strcmp(b1, "end")))
            break;
        printf("client:%s\n", b1);
        printf("server :");
        scanf("%s", b2);
        sendto(s, b2, sizeof(b1), 0, (struct sockaddr *)&client, n);
    }
```

UDP Client:

1. Create a UDP socket.
2. Send a message to the server.
3. Wait for the reply from the server.
4. Process the packet.
5. Finish.

```c
#include <arpa/inet.h>
#include <netinet/in.h>
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/stat.h>
#include <sys/types.h>
void main()
{
    struct
        sockaddr_in server,
        client;
    int s, n;
    char b1[10], b2[10];
    s = socket(AF_INET, SOCK_DGRAM, 0);
    server.sin_family = AF_INET;
    server.sin_port = 3000;
    server.sin_addr.s_addr = inet_addr("127.0.0.1");
    n = sizeof(server);
    while (1)
    {
        printf("\nClient:");
        scanf("%s", b2);
        sendto(s, b2, sizeof(b2), 0, (struct sockaddr *)&server, n);
        if (!(strcmp(b2, "end")))
            break;
        recvfrom(s, b1, sizeof(b1), 0, NULL, NULL);
        printf("\nserver :%s\n", b1);
    }
}
```

Distance Vector Routing:

1. Start
2. By convention, the distance of the node to itself is assigned to zero and when a node is unreachable the distance is accepted as 999.
3. Accept the input distance matrix from the user that represents the distance between each node in the network.
4. Store the distance between nodes in a suitable variable.
5. Calculate the minimum distance between two nodes by iterating.
    - If the distance between two nodes is larger than the calculated alternate available path, replace the existing distance with the calculated distance.
6. Print the shortest path calculated.
7. Stop.

```c
#include <stdio.h>
void main()
{
    int number_of_nodes, i, j, k, x;
    printf("Enter the number of nodes:");
    scanf("%d", &number_of_nodes);
    int routing_table[number_of_nodes][number_of_nodes];
    printf("Enter the routing table:\n");
    for (i = 0; i < number_of_nodes; i++)
        for (j = 0; j < number_of_nodes; j++)
        {
            printf("[%d][%d]: ", i, j);
            scanf("%d", &routing_table[i][j]);
        }
    for (x = 0; x < number_of_nodes; x++)
        for (i = 0; i < number_of_nodes; i++)
            for (j = 0; j < number_of_nodes; j++)
                for (k = 0; k < number_of_nodes; k++)
                    if (routing_table[i][j] > routing_table[i][k] +
routing_table[k][j])
                        routing_table[i][j] = routing_table[i][k] +
routing_table[k][j];
    printf("\nDistance Vector Table:\n");
    for (i = 0; i < number_of_nodes; i++)
    {
        for (j = 0; j < number_of_nodes; j++)
            printf("%d\t", routing_table[i][j]);
        printf("\n");
    }
}
```

Leaky Bucket:

Algorithm:

1. Start

2. Let STORE = 0

3. Read bucket size (BUCKETSIZE), outgoing rate (OUTGOING) and number of inputs (N) from user

4. While N ≠ 0, do

a. Read packet size to INCOMING

b. If INCOMING ≤ (BUCKETSIZE - STORE), then

i. STORE = STORE + INCOMING

ii. Print STORE

c. Else, then

a. Print INCOMING - (BUCKETSIZE - STORE) as "Dropped number of packets"

b. STORE = BUCKETSIZE

c. Print STORE

d. STORE = STORE - OUTGOING

e. If STORE < 0, then

a. STORE = 0

f. Print STORE

g. Decrement N by 1

5. Stop

```c
#include <stdio.h>
int main()
{
    int inc, outg, b_size, n, store = 0;
    printf("Enter bucket size, outgoing rate and number of inputs:");
    scanf("%d %d %d", &b_size, &outg, &n);
    while (n != 0)
    {
        printf("Enter the incoming packet size :");
        scanf("%d", &inc);
        printf("Incoming packet size :- %d\n", inc);
        if (inc <= (b_size - store))
        {
            store += inc;
            printf("Bucket buffer size is %d out of %d\n", store, b_size);
        }
        else
        {
```

```
            printf("Dropped number of packets :- %d\n", inc - (b_size - store));
            store += b_size - store;
            printf("Bucket buffer size is %d out of %d\n", store, b_size);
        }
        store = store - outg;
        printf("After outgoing,%d packets left out of %d in buffer\n", store, b_size);
        printf("\n");
        n--;
    }
}
```