

INTRODUCTION TO COMPUTER NETWORKS

- A Computer Network is a set of computers connected together for the purpose of sharing resources.
- The most common resource shared today is connection to the Internet. Other shared resources can include a printer or a file server.

The OSI Model:

The Open Systems Interconnection (OSI) model is a reference tool for understanding data communications between any two networked systems. It divides the communications processes into seven layers. Each layer both performs specific functions to support the layers above it and offers services to the layers below it. The three lowest layers focus on passing traffic through the network to an end system. The top four layers come into play in the end system to complete the process.

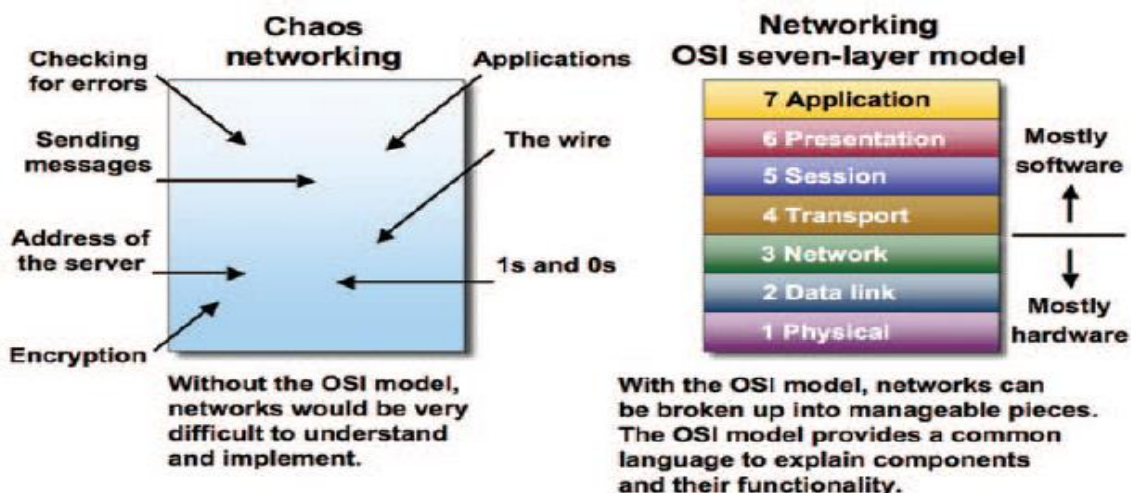
This white paper will provide you with an understanding of each of the seven layers, including their functions and their relationships to each other. This will provide you with an overview of the network process, which can then act as a framework for understanding the details of computer networking.

Since the discussion of networking often includes talk of “extra layers”, this paper will address these unofficial layers as well.

Finally, this paper will draw comparisons between the theoretical OSI model and the functional TCP/IP model.

Although TCP/IP has been used for network communications before the adoption of the OSI model, it supports the same functions and features in a differently layered arrangement.

An Overview of the OSI Model



A networking model offers a generic means to separate computer networking functions into multiple layers.

Each of these layers relies on the layers below it to provide supporting capabilities and performs support to the layers above it. Such a model of layered functionality is also called a “protocol stack” or “protocol suite”

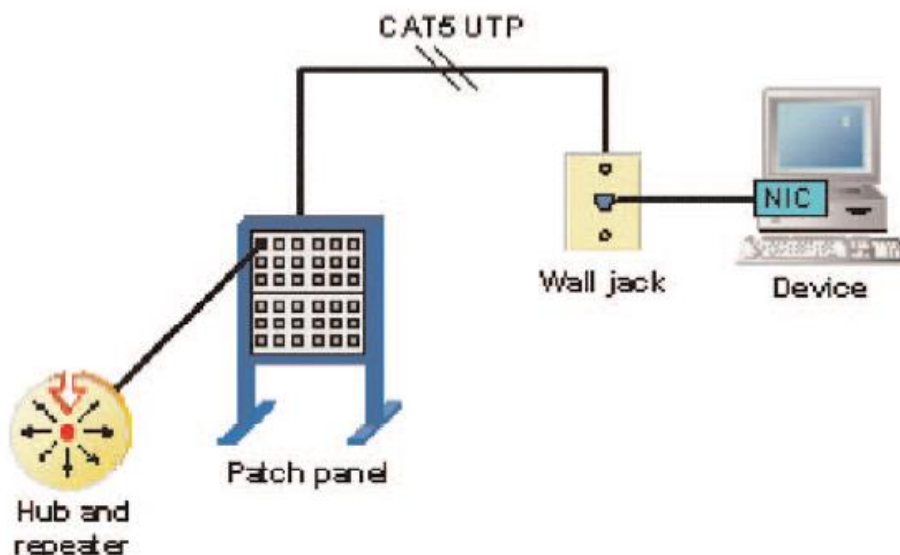
Protocols, or rules, can do their work in either hardware or software or, as with most protocol stacks, in a combination of the two. The nature of these stacks is that the lower layers do their work in hardware or firmware (software that runs on specific hardware chips) while the higher layers work in software.

The Open System Interconnection model is a seven-layer structure that specifies the requirements for communications between two computers. The ISO (International Organization for Standardization) standard 7498-1 defined this model. This model allows all network elements to operate together, no matter who created the protocols and what computer vendor supports them.

The main benefits of the OSI model include the following:

- Helps users understand the big picture of networking
- Helps users understand how hardware and software elements function together
- Makes troubleshooting easier by separating networks into manageable pieces
- Defines terms that networking professionals can use to compare basic functional relationships on different networks
- Helps users understand new technologies as they are developed
- Aids in interpreting vendor explanations of product functionality

Layer 1 – The Physical Layer



The physical layer of the OSI model defines connector and interface specifications, as well as the medium (cable) requirements. Electrical, mechanical, functional, and procedural specifications are provided for sending a bit stream on a computer network.

Components of the physical layer include:

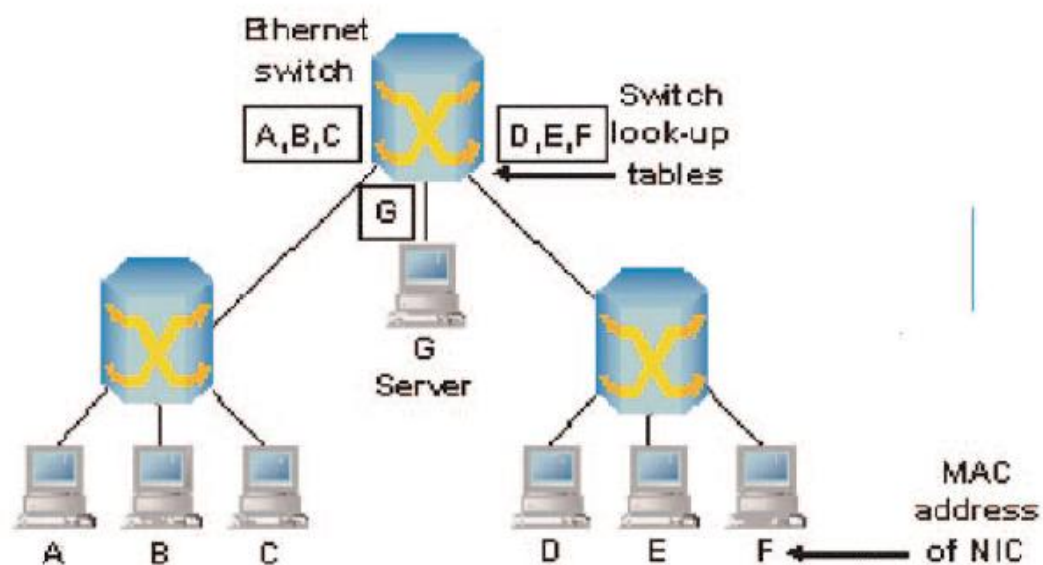
- Cabling system components
- Adapters that connect media to physical interfaces

- Connector design and pin assignments
- Hub, repeater, and patch panel specifications
- Wireless system components
- Parallel SCSI (Small Computer System Interface)
- Network Interface Card (NIC)

In a LAN environment, Category 5e UTP (Unshielded Twisted Pair) cable is generally used for the physical layer for individual device connections. Fiber optic cabling is often used for the physical layer in a vertical or riser backbone link. The IEEE, EIA/TIA, ANSI, and other similar standards bodies developed standards for this layer.

Note: The Physical Layer of the OSI model is only part of a LAN (Local Area Network).

Layer 2 – The Data Link Layer



Layer 2 of the OSI model provides the following functions:

- Allows a device to access the network to send and receive messages
- Offers a physical address so a device's data can be sent on the network
- Works with a device's networking software when sending and receiving messages
- Provides error-detection capability

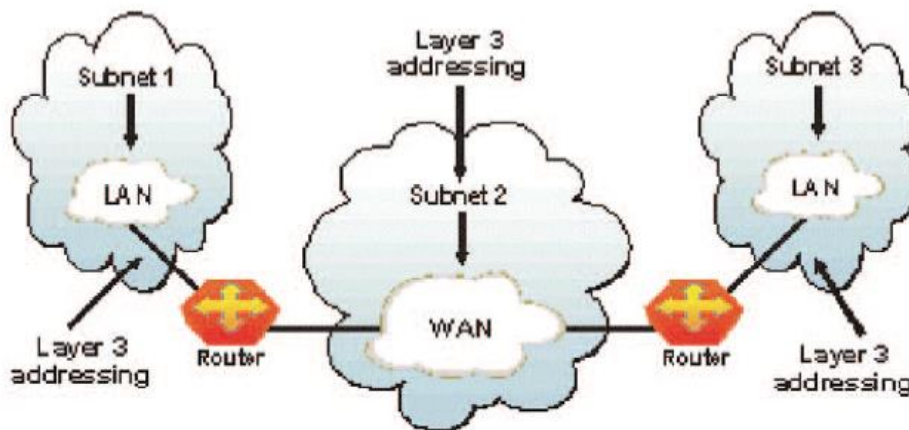
Common networking components that function at layer 2 include:

- Network interface cards
- Ethernet and Token Ring switches
- Bridges

NICs have a layer 2 or MAC address. A switch uses this address to filter and forward traffic, helping relieve congestion and collisions on a network segment.

Bridges and switches function in a similar fashion; however, bridging is normally a software program on a CPU, while switches use Application-Specific Integrated Circuits (ASICs) to perform the task in dedicated hardware, which is much faster.

Layer 3 – The Network Layer



Layer 3, the network layer of the OSI model, provides an end-to-end logical addressing system so that a packet of data can be routed across several layer 2 networks (Ethernet, Token Ring, Frame Relay, etc.). Note that network layer addresses can also be referred to as logical addresses.

Initially, software manufacturers, such as Novell, developed proprietary layer 3 addressing. However, the networking industry has evolved to the point that it requires a common layer 3 addressing system. The Internet Protocol (IP) addresses make networks easier to both set up and connect with one another. The Internet uses IP addressing to provide connectivity to millions of networks around the world.

To make it easier to manage the network and control the flow of packets, many organizations separate their network layer addressing into smaller parts known as subnets. Routers use the network or subnet portion of the IP addressing to route traffic between different networks. Each router must be configured specifically for the networks or subnets that will be connected to its interfaces.

Routers communicate with one another using routing protocols, such as Routing Information Protocol (RIP) and Open version of Shortest Path First (OSPF), to learn of other networks that are present and to calculate the best way to reach each network based on a variety of criteria (such as the path with the fewest routers). Routers and other networked systems make these routing decisions at the network layer.

When passing packets between different networks, it may become necessary to adjust their outbound size to one that is compatible with the layer 2 protocol that is being used. The network layer accomplishes this via a process known as fragmentation. A router's network layer is usually responsible for doing the fragmentation. All reassembly of fragmented packets happens at the network layer of the final destination system.

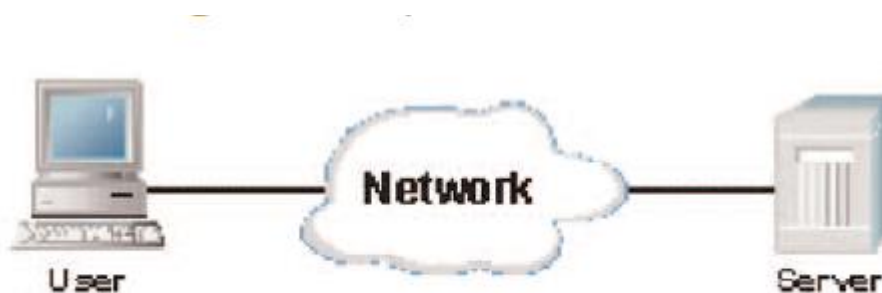
Two of the additional functions of the network layer are diagnostics and the reporting of logical variations in normal network operation. While the network layer diagnostics may be initiated by any networked system, the system

discovering the variation reports it to the original sender of the packet that is found to be outside normal network operation.

The variation reporting exception is content validation calculations. If the calculation done by the receiving system does not match the value sent by the originating system, the receiver discards the related packet with no report to the sender. Retransmission is left to a higher layer's protocol.

Some basic security functionality can also be set up by filtering traffic using layer 3 addressing on routers or other similar devices.

Layer 4 – The Transport Layer



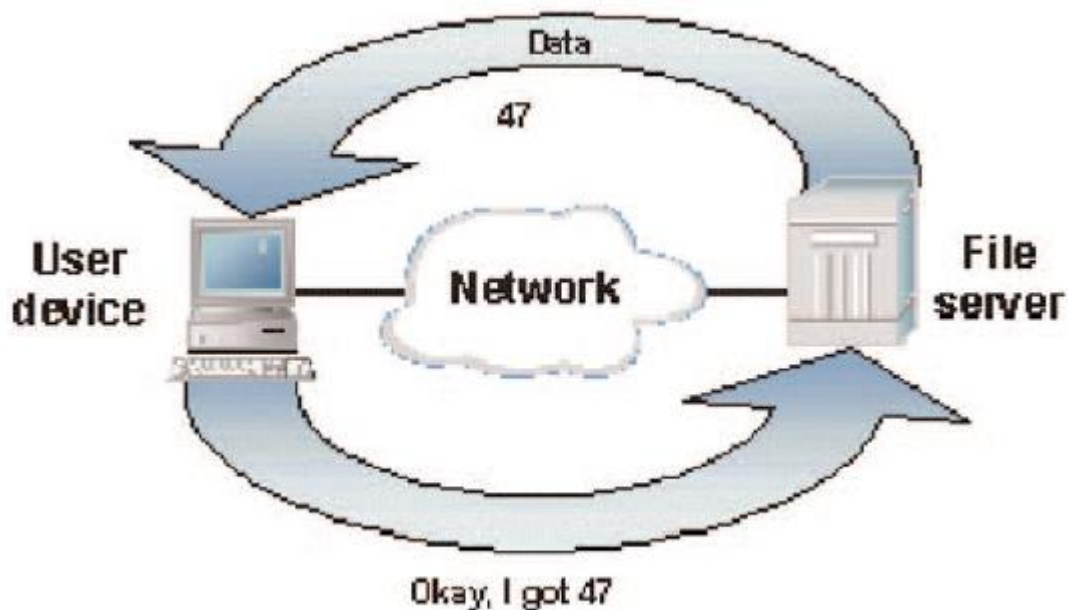
Layer 4, the transport layer of the OSI model, offers end-to-end communication between end devices through a network. Depending on the application, the transport layer either offers reliable, connection-oriented or connectionless, best-effort communications.

Some of the functions offered by the transport layer include:

- Application identification
- Client-side entity identification
- Confirmation that the entire message arrived intact
- Segmentation of data for network transport
- Control of data flow to prevent memory overruns
- Establishment and maintenance of both ends of virtual circuits
- Transmission-error detection
- Realignment of segmented data in the correct order on the receiving side
- Multiplexing or sharing of multiple sessions over a single physical link

The most common transport layer protocols are the connection-oriented TCP Transmission Control Protocol (TCP) and the connectionless UDP User Datagram Protocol (UDP).

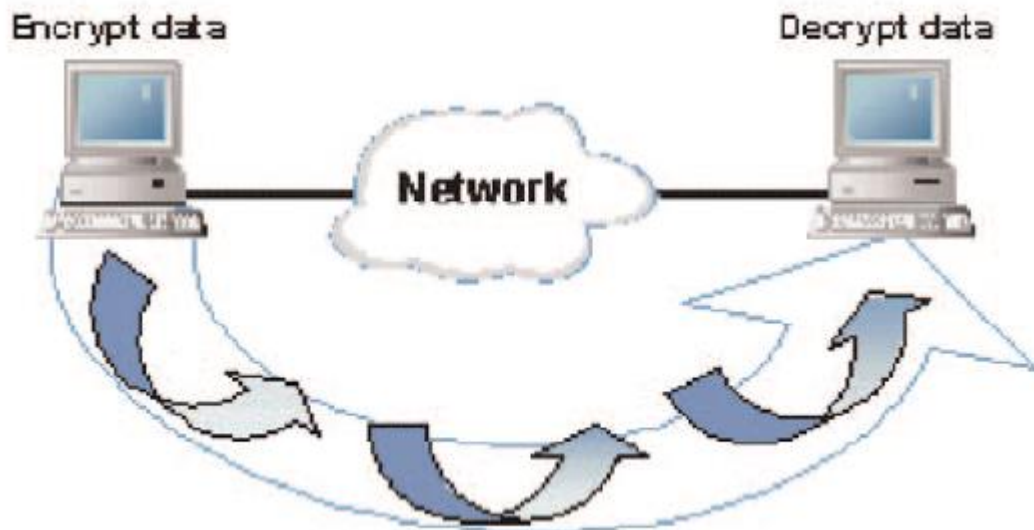
Layer 5 – The Session Layer



Layer 5, the session layer, provides various services, including tracking the number of bytes that each end of the session has acknowledged receiving from the other end of the session. This session layer allows applications functioning on devices to establish, manage, and terminate a dialog through a network. Session layer functionality includes:

- Virtual connection between application entities
- Synchronization of data flow
- Creation of dialog units
- Connection parameter negotiations
- Partitioning of services into functional groups
- Acknowledgements of data received during a session
- Retransmission of data if it is not received by a device

Layer 6 – The Presentation Layer



Layer 6, the presentation layer, is responsible for how an application formats the data to be sent out onto the network. The presentation layer basically allows an application to read (or understand) the message.

Examples of presentation layer functionality include:

- Encryption and decryption of a message for security
- Compression and expansion of a message so that it travels efficiently
- Graphics formatting
- Content translation
- System-specific translation

Layer 7 – The Application Layer



Layer 7, the application layer, provides an interface for the end user operating a device connected to a network.

This layer is what the user sees, in terms of loading an application (such as Web browser or e-mail); that is, this application layer is the data the user views while using these applications.

Examples of application layer functionality include:

- Support for file transfers
- Ability to print on a network
- Electronic mail
- Electronic messaging
- Browsing the World Wide Web

Layers 8, 9, and 10

Whether a designed to be a humorous extension or a secret technician code, layers 8, 9, and 10 are not officially part of the OSI model. They refer to the non-technical aspects of computer networking that often interfere with the smooth design and operation of the network.

Layer 8 is usually considered the “office politics” layer. In most organizations, there is at least one group who is favored, at least temporarily, by management and receives “special” treatment. When it comes to networking, this may mean that this group always has the latest and/or fastest equipment and highest speed network links.

Layer 9 is generally referred to as the “blinders” layer. This layer applies to organizational managers who have already decided, usually with little or no current information, to dictate a previously successful network plan.

They may say things such as:

“It worked in my last company, so we will use it here.”

“Everybody says this is the right solution.”

“I read in an airline magazine that this was the best way to do it so that is what we will do.”

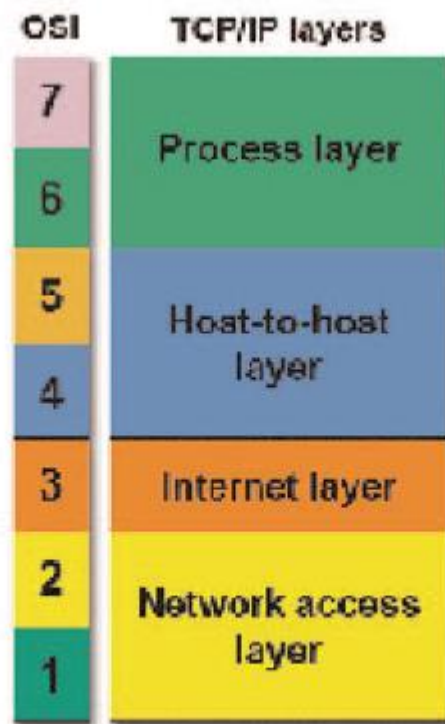
What these managers seem to forget is that they are paying a highly qualified staff to provide them with useful information. These managers bypass planning in order to make a quick decision.

Layer 10, the “user” layer, is in every organization. But users are much more than a layer. While they are one of the reasons the network exists, users can also be a big part of the need for troubleshooting. This is especially true when the users have computers at home and have decided to “help” the network administrator or manager by making changes to the network without consulting the network staff. Equally challenging is the user who “didn’t do anything” when the network segment in his/her immediate vicinity suddenly stopped working. In these cases, the layer 10 identification coincides with layer 10 troubles (and the “ID10T” label some technicians have used).

TCP/IP

Model

Overview



The OSI model describes computer networking in seven layers. While there have been implementations of networking protocol that use those seven layers, most networks today use TCP/IP. But, networking professionals continue to describe networking functions in relation to the OSI layer that performs those tasks.

The TCP/IP model uses four layers to perform the functions of the seven-layer OSI model.

The network access layer is functionally equal to a combination of OSI physical and data link layers (1 and 2).

The Internet layer performs the same functions as the OSI network layer (3).

Things get a bit more complicated at the host-to-host layer of the TCP/IP model. If the host-to-host protocol is TCP, the matching functionality is found in the OSI transport and session layers (4 and 5). Using UDP equates to the functions of only the transport layer of the OSI model.

The TCP/IP process layer, when used with TCP, provides the functions of the OSI model's presentation and application layers (6 and 7). When the TCP/IP transport layer protocol is UDP, the process layer's functions are equivalent to OSI session, presentation, and application layers (5, 6, and 7).

WEEK 1

QUESTION: Connect the computers in Local Area Network.

AIM: To Connect the computers in Local Area Network.

PROCEDURE:

Step 1: Right Click on folder. Select PROPERTIES.

Step 2: Check the “Share this folder” and click “permissions”.

Step 3: Select “Everyone” as Group or user name and give “Allow” and “Deny” permission and click Apply and Ok.

Step 5: “Turn off window Firewall” is selected in public network location settings.

Step 6: Press “Windows + R” to open “run” and type two backslash with followed by remote computer IPv4.

Eg: - [\\125.200.20.153](#)

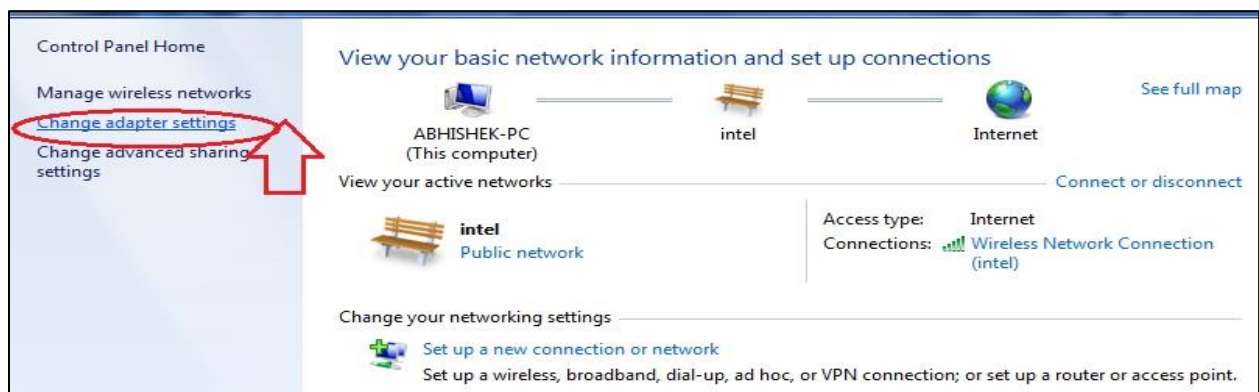
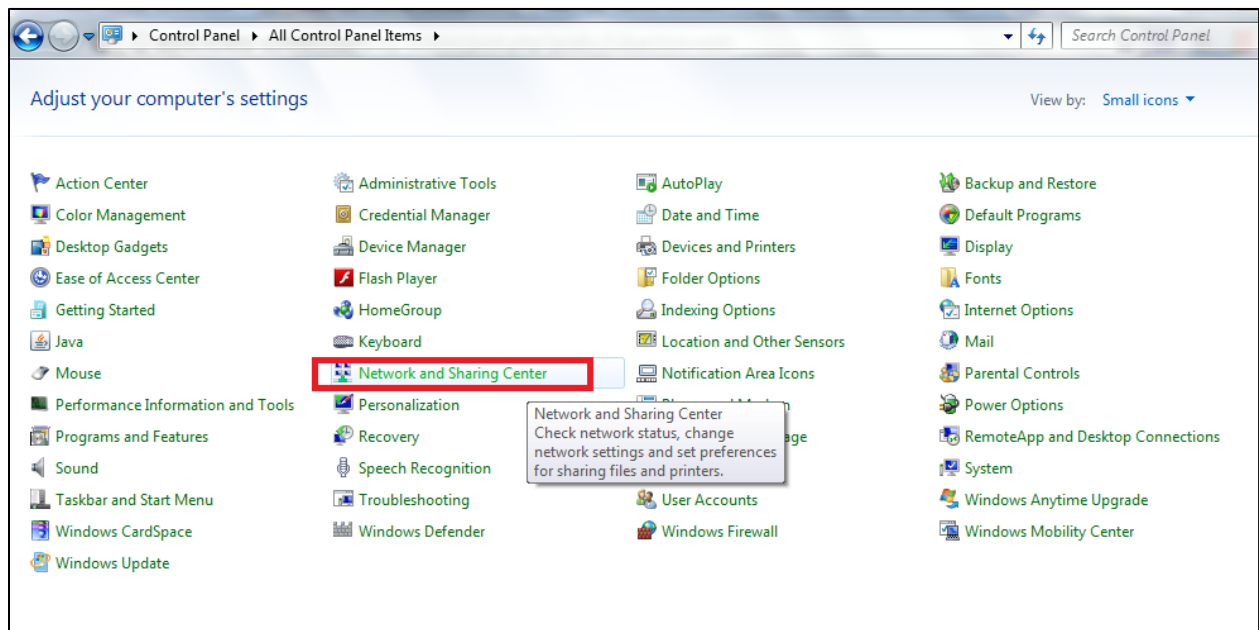
Step 7: Enter user_name and password if asked, and press Enter.

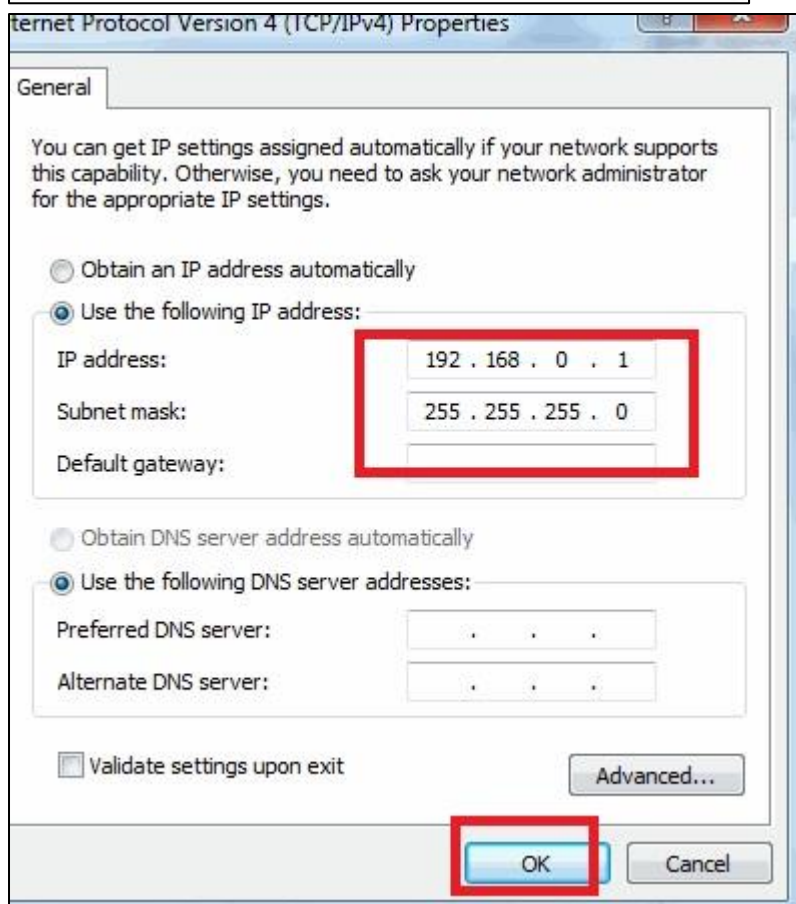
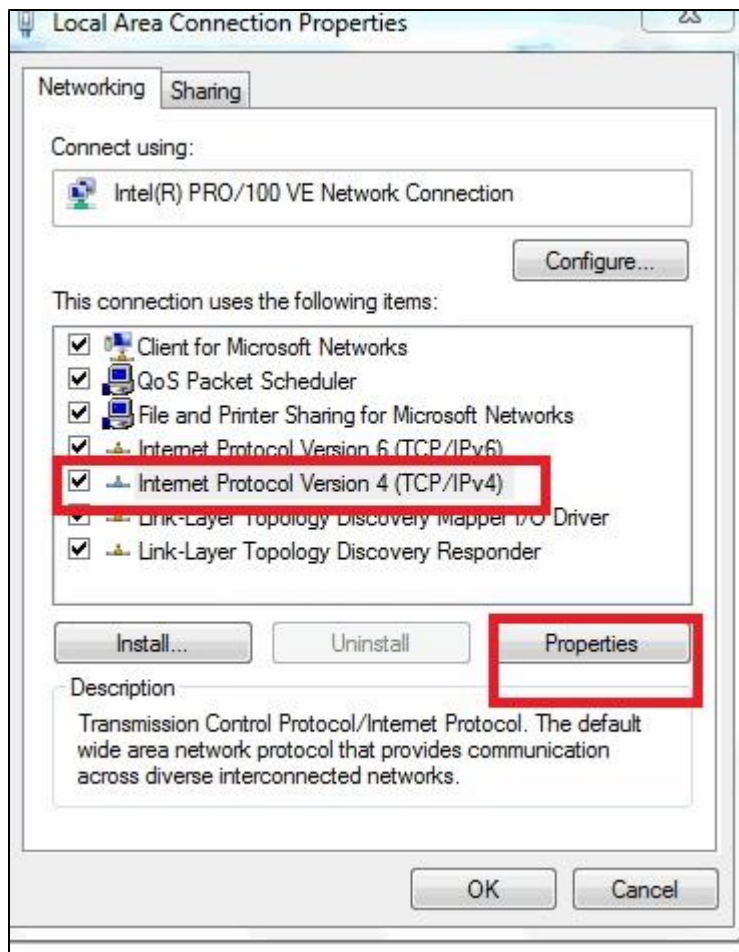
Step 8: Get access to share folder of remote system and copy a file from our system to remote system.

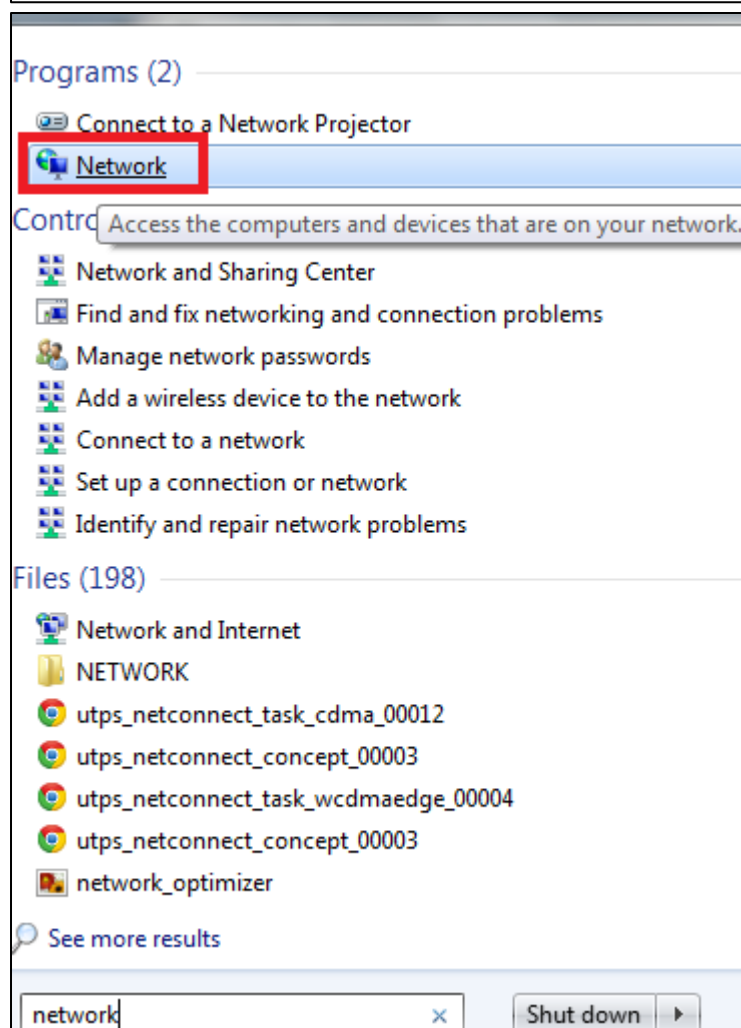
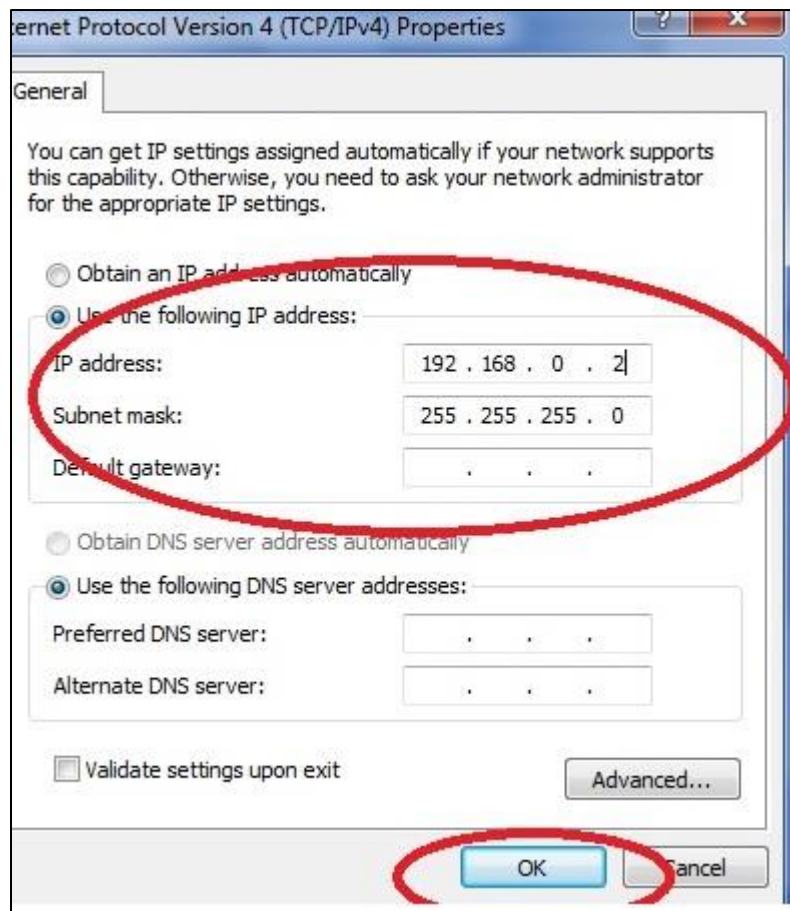
Step 9: The same procedure can be followed cut, delete, etc if they are allowed to performed such operations.

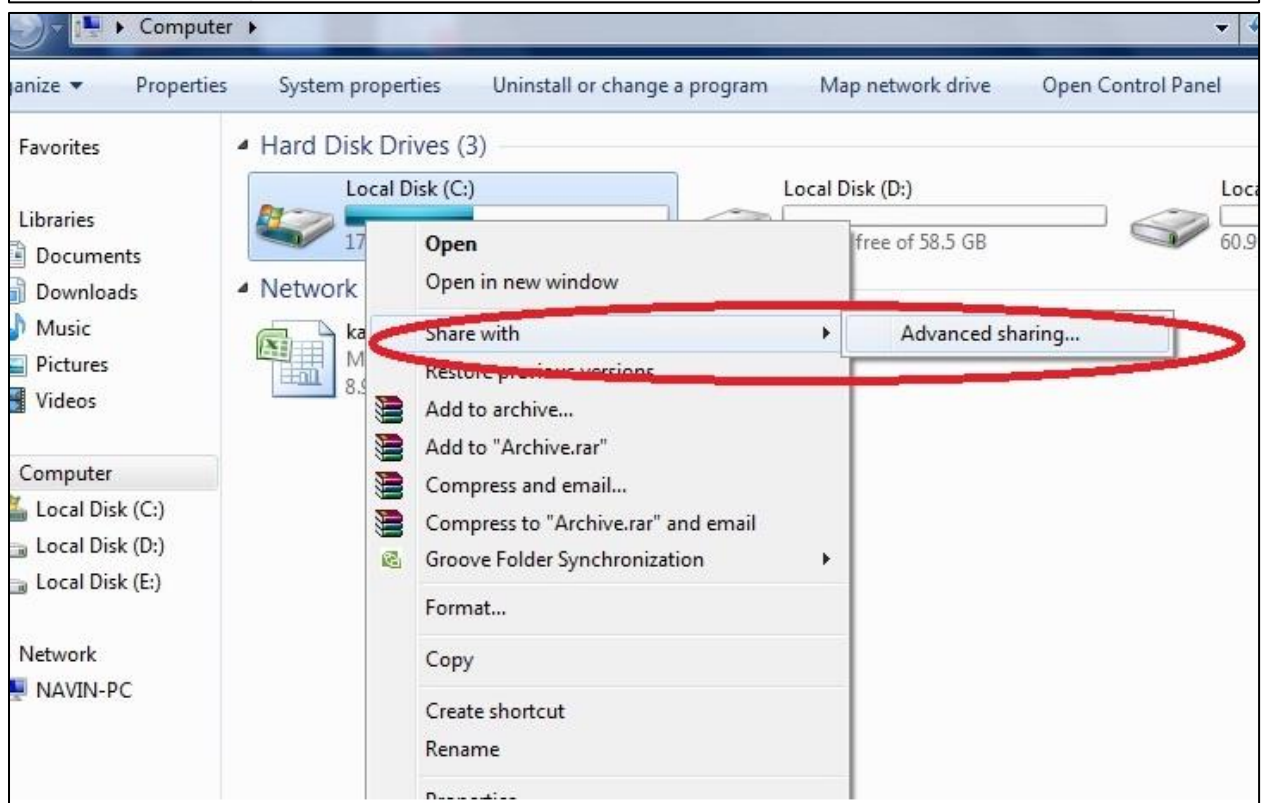
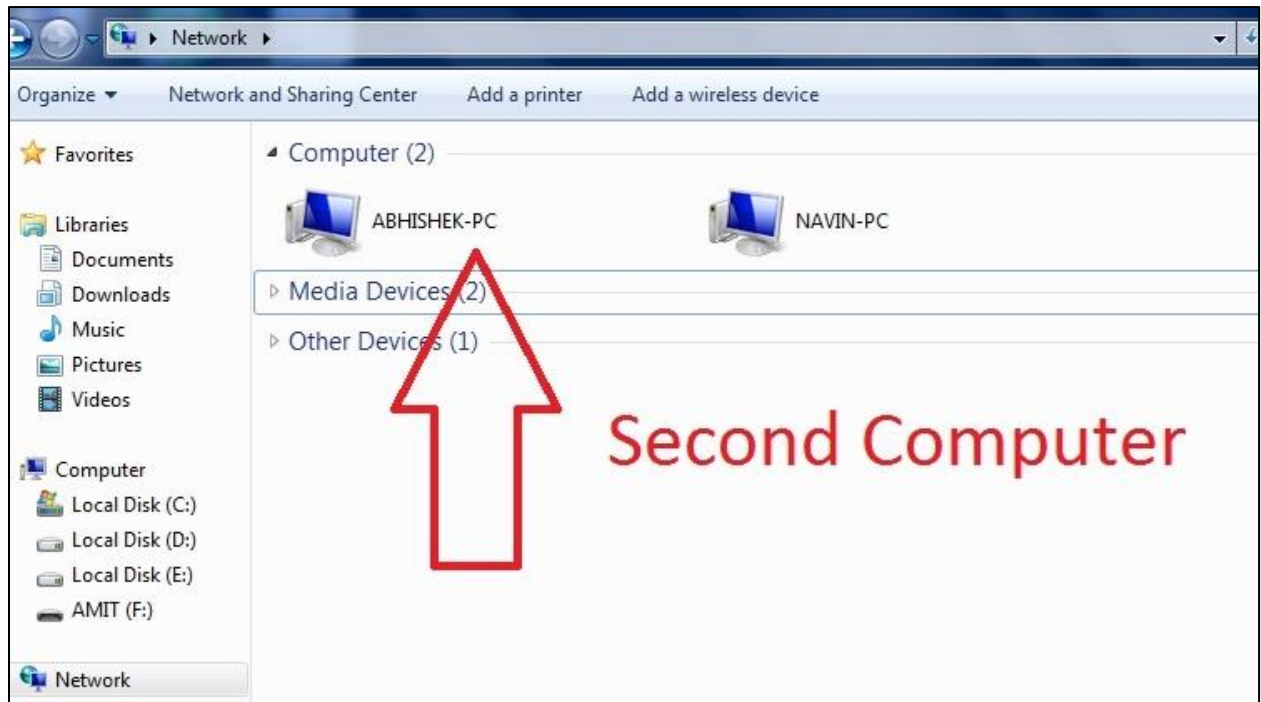
Note: These Shared folders can be accessed by the one system at a time.

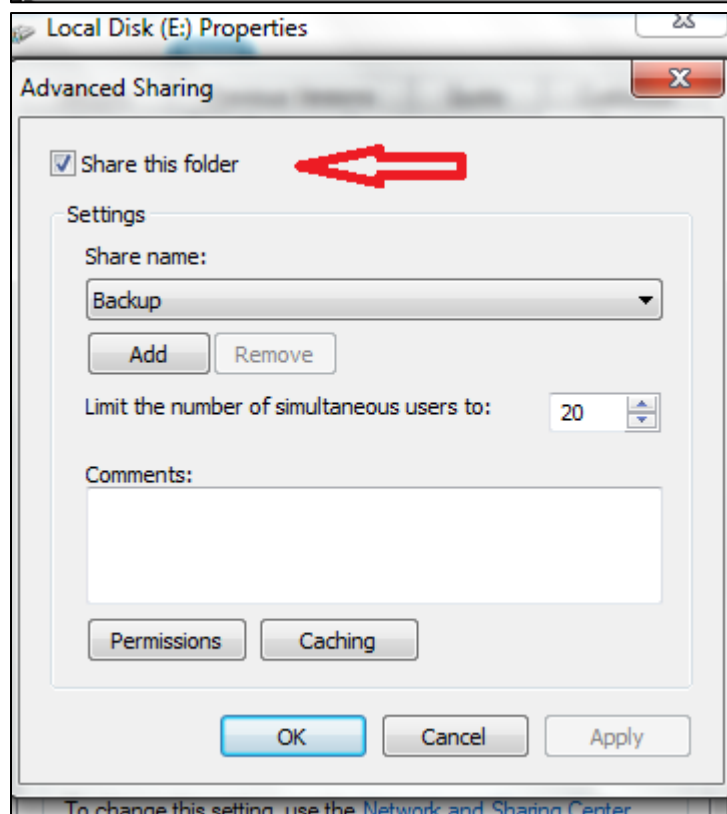
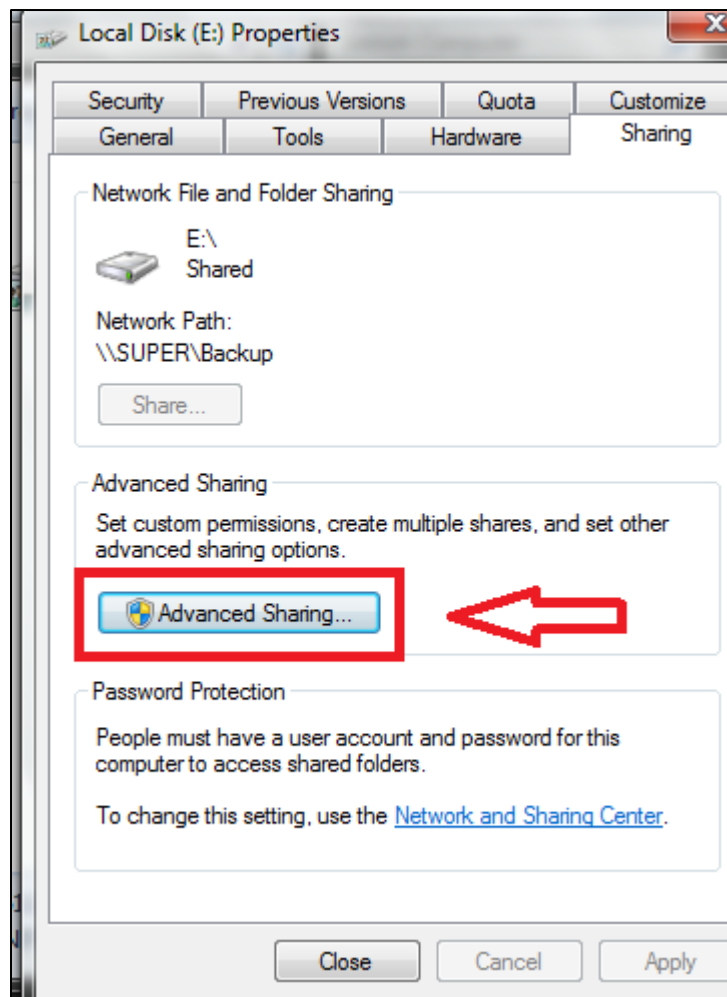
OUTPUT:











WEEK 2

QUESTION: Implement CRC technique for error handling.

AIM : To implement Cyclic Redundancy Check (CRC) using C-program.

DESCRIPTION : The polynomial code, also known as a CRC (Cyclic Redundancy Check). Polynomial codes are based upon treating bit strings as representations of polynomials with coefficients of 0 and 1 only. A k-bit frame is regarded as the coefficient list for a polynomial with k terms, ranging from x^{k-1} to x^0 . Such a polynomial is said to be of degree k - 1. The high-order (leftmost) bit is the coefficient of x^{k-1} ; the next bit is the coefficient of x^{k-2} , and so on. For example, 110001 has 6 bits and thus represents a six-term polynomial with coefficients 1, 1, 0, 0, 0, and 1: $x^5 + x^4 + x^0$.

SOURCE CODE :

```
#include <stdio.h>

#include <malloc.h>

#include <string.h>

char x_or(char a, char b) {
    return (a=='0' && b=='0' || a=='1' && b=='1')?'0':'1';
}

void fun(char * temp, char * g, int g_l) {
    for(int i=0; i<g_l-1; i++)
        temp[i] = x_or(temp[i+1], g[i+1]);
    return;
}

void crc_code(char * f, char * g, int pos) {
    int g_l = strlen(g), f_l = strlen(f), i, j;
    if(pos == 0) {
        for(i=f_l; i<f_l+g_l-1; i++)
            f[i] = '0';
        f[i] = '\0';
    }
    char * temp = (char*)malloc(100);
    for(i=0; i<g_l; i++)
        temp[i] = f[i];
    temp[i] = '\0';
```

```

j=g_1;
while(1) {
    // printf("\n%s", temp);
    if(temp[0] == '1') {
        fun(temp, g, g_1);
        if(j == f_1+g_1-1 && pos == 0) {
            j = f_1;
            for(i=0; i<g_1-1; i++)
                f[j++] = temp[i];
            f[j] = '\0';
            printf("\nThe crc is: ");
            for(int k=0; k<g_1-1; k++)
                printf("%c", temp[k]);
            return;
        }
        if(j == f_1 && pos == 1) {
            for(i=0; i<g_1-1; i++)
                if(temp[i] != '0') {
                    printf("\nError\nThe remainder is ");
                    for(int k=0; k<g_1-1; k++)
                        printf("%c", temp[k]);
                    printf(" instead of 0's");
                    return;
                }
            printf("\nData Transferred Successfully!!!");
            return;
        }
        temp[g_1-1] = f[j];
        temp[g_1] = '\0';
    }
    else {
        for(i=0; i<g_1-1; i++)

```

```

        temp[i] = temp[i+1];
    if(j == f_l+g_l-1 && pos == 0) {
        j = f_l;
        for(i=0; i<g_l-1; i++)
            f[j++] = temp[i];
        f[j] = '\0';
        printf("\nThe crc is: ");
        for(int k=0; k<g_l-1; k++)
            printf("%c", temp[k]);
        return;
    }
    if(j == f_l && pos == 1) {
        for(i=0; i<g_l-1; i++)
            if(temp[i] != '0') {
                printf("\nError\nThe remainder is ");
                for(int k=0; k<g_l-1; k++)
                    printf("%c", temp[k]);
                printf(" instead of 0's");
                return;
            }
        printf("\nData Transferred Successfully!!!");
        return;
    }
    temp[g_l-1] = f[j];
    temp[g_l] = '\0';
}
j++;
}
return;
}

void crc_encode(char * f, char * g) {
    crc_code(f, g, 0);
}

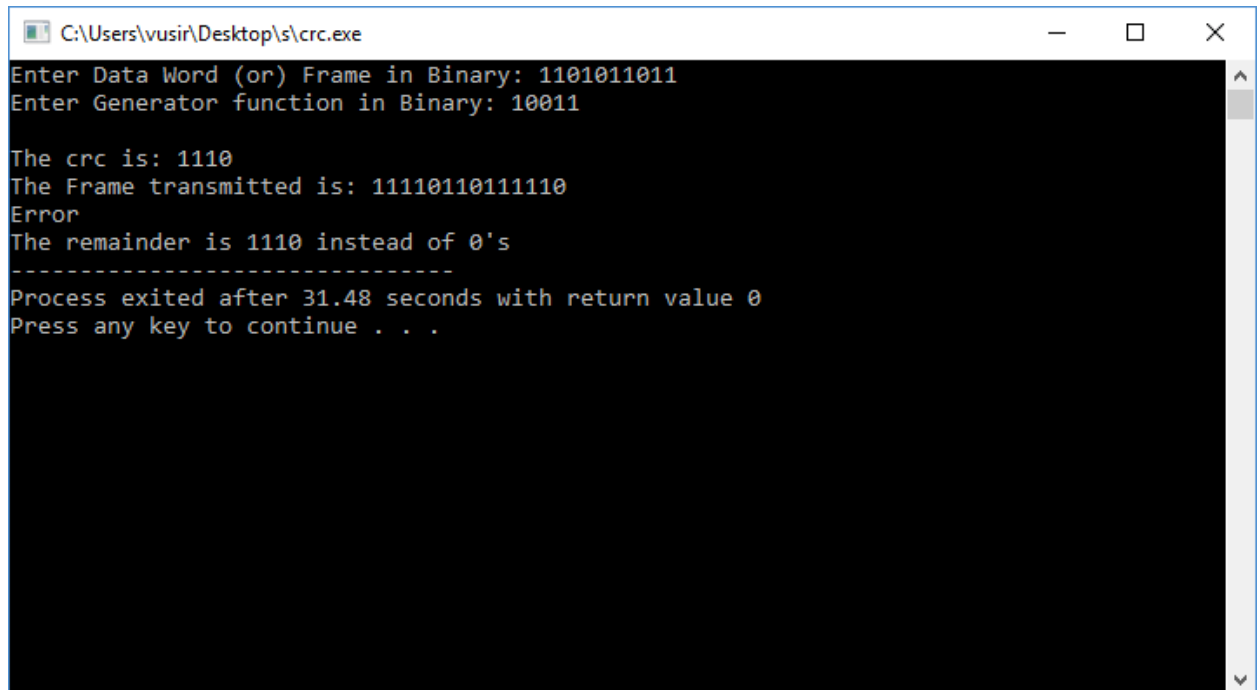
```

```

}
void crc_decode(char * f, char * g) {
    crc_code(f, g, 1);
}
int main() {
    char * f = (char*)malloc(100);
    char * g = (char*)malloc(100);
    printf("Enter Data Word (or) Frame in Binary: ");
    scanf("%s", f);
    printf("Enter Generator function in Binary: ");
    scanf("%s", g);
    crc_encode(f, g);
    f[2] = '1';
    printf("\nThe Frame transmitted is: %s", f);
    crc_decode(f, g);
    return 0;
}

```

OUTPUT:



```
C:\Users\vusir\Desktop\s\crc.exe
Enter Data Word (or) Frame in Binary: 1101011011
Enter Generator function in Binary: 10011

The crc is: 1110
The Frame transmitted is: 11110110111110
Error
The remainder is 1110 instead of 0's
-----
Process exited after 31.48 seconds with return value 0
Press any key to continue . . .
```


QUESTION: Implement Error detection technique for error handling.

AIM: To implement Hamming Code using C-program.

DESCRIPTION: This theoretical lower limit can, in fact, be achieved using a method due to Hamming (1950). The bits of the codeword are numbered consecutively, starting with bit 1 at the left end, bit 2 to its immediate right, and so on. The bits that are powers of 2 (1, 2, 4, 8, 16, etc.) are check bits. The rest (3, 5, 6, 7, 9, etc.) are filled up with the m data bits. Each check bit forces the parity of some collection of bits, including itself, to be even (or odd). A bit may be included in several parity computations. To see which check bits the data bit in position k contributes to, rewrite k as a sum of powers of 2. For example, $11 = 1 + 2 + 8$ and $29 = 1 + 4 + 8 + 16$. A bit is checked by just those check bits occurring in its expansion (e.g., bit 11 is checked by bits 1, 2, and 8).

SOURCE CODE :

```
#include <stdio.h>

#include <string.h>

#include <math.h>

#include <malloc.h>

int i, j, k=0, v;

void find_r_encode(int m, int * r) {

    int fact=4;

    while(m+(*r)+1 > fact) {

        fact *= 2;

        (*r)++;

    }

    return;

}

void find_r_decode(int q, int * r) {

    int fact=4;

    while(q+1 > fact) {

        fact *= 2;

        (*r)++;

    }

    return;

}
```

```

void x_or(char * temp, char * val) {
    for(i=0; i<strlen(temp); i++)
        temp[i] = (temp[i]=='0' && val[i]=='0' || temp[i]=='1' &&
val[i]=='1') ? '0' : '1';
    return;
}

void final_frame(char * f, char * temp, char * ans, int r, int len) {
    j = r-1;
    k=0;
    for(i=0; i<len; i++) {
        float p = log(len-i)/log(2);
        ans[i] = (p != (int)p) ? f[k++] : temp[j--];
    }
    ans[i] = '\0';
    return;
}

void correction(char * temp, int r) {
    int fa = 1, fla = 0;
    for(i=0; i<r; i++) {
        fla += fa * (temp[i]-48);
        fa *= 2;
    }
    if(fla!=0)
        printf("\n\n%dth bit is Wrong in the data Transmission.", fla);
    else
        printf("\n\nData Transferred Successfully!!!");
    return;
}

void encode(char * f, char * ans) {
    int r=2, len;
    find_r_encode(strlen(f), &r);
    len = strlen(f) + r;

```

```

char * temp = (char*)malloc(100);
char * val = (char*)malloc(100);
for(i=0; i<r; i++)
    temp[i] = '0';
temp[i] = '\0';
for(i=0; i<len; i++) {
    v = len-i;
    float p = log(len-i)/log(2);
    if(p != (int)p && f[k++] == '1') {
        for(j=0; j<r; j++) {
            val[j] = v%2 + 48;
            v /= 2;
        }
        val[j] = '\0';
        x_or(temp, val);
    }
}
final_frame(f, temp, ans, r, len);
}

void decode(char *f) {
    int r=2, len = strlen(f);
    find_r_decode(len, &r);
    char * temp = (char*)malloc(100);
    char * val = (char*)malloc(100);
    for(i=0; i<r; i++)
        temp[i] = '0';
    temp[i] = '\0';
    for(i=0; i<len; i++) {
        v = len-i;
        if(f[i] == '1') {
            for(j=0; j<r; j++) {
                val[j] = v%2 + 48;

```

```

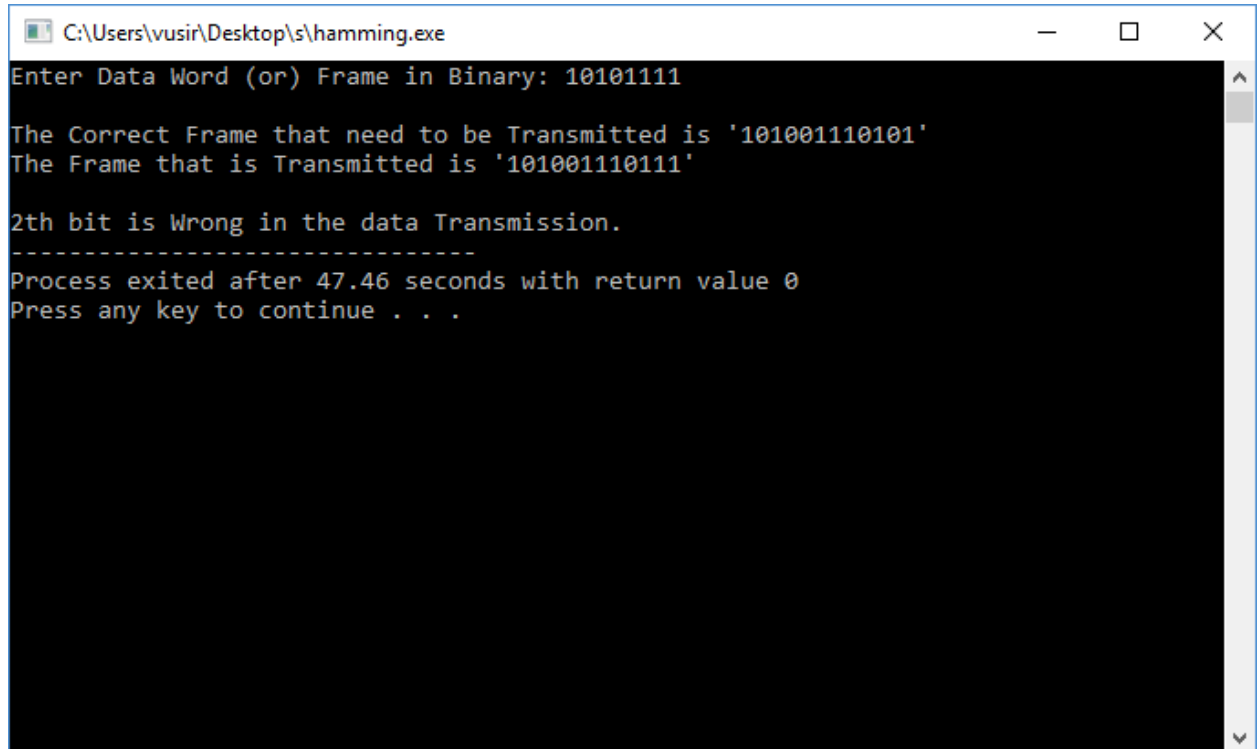
        v /= 2;
    }
    val[j] = '\0';
    x_or(temp, val);
}
}
correction(temp, r);
}

void only_decode(int dr) {
    char * ans = (char*)malloc(100);
    scanf("%s", ans);
    decode(ans);
    return;
}

int main() {
    char * f = (char*)malloc(100);
    printf("Enter Data Word (or) Frame in Binary: ");
    scanf("%s", f);
    char * ans = (char*)malloc(100);
    /**encode**/
    encode(f, ans);
    printf("\nThe Correct Frame that need to be Transmitted is '%s'", ans);
    /** some data has changed let it be 2nd position**/
    ans[strlen(ans)-2] = '1';
    printf("\nThe Frame that is Transmitted is '%s'", ans);
    /**decode**/
    decode(ans);
    // only_decode(4);
    return 0;
}

```

OUTPUT:



```
C:\Users\vusir\Desktop\s\hamming.exe
Enter Data Word (or) Frame in Binary: 10101111

The Correct Frame that need to be Transmitted is '101001110101'
The Frame that is Transmitted is '101001110111'

2th bit is Wrong in the data Transmission.
-----
Process exited after 47.46 seconds with return value 0
Press any key to continue . . .
```

WEEK 3

QUESTION: Simulation of Stop and Wait protocol

AIM: To Simulate Stop and Wait protocol

DESCRIPTION:

Characteristics

- Used in Connection-oriented communication.
- It offers error and flow control
- It is used in Data Link and Transport Layers
- Stop and Wait ARQ mainly implements Sliding Window Protocol concept with Window Size 1

Useful Terms:

- Propagation Delay: Amount of time taken by a packet to make a physical journey from one router to another router.
- Propagation Delay = (Distance between routers) / (Velocity of propagation)
- RoundTripTime (RTT) = 2* Propagation Delay
- TimeOut (TO) = 2* RTT
- Time To Live (TTL) = 2* TimeOut. (Maximum TTL is 180 seconds)

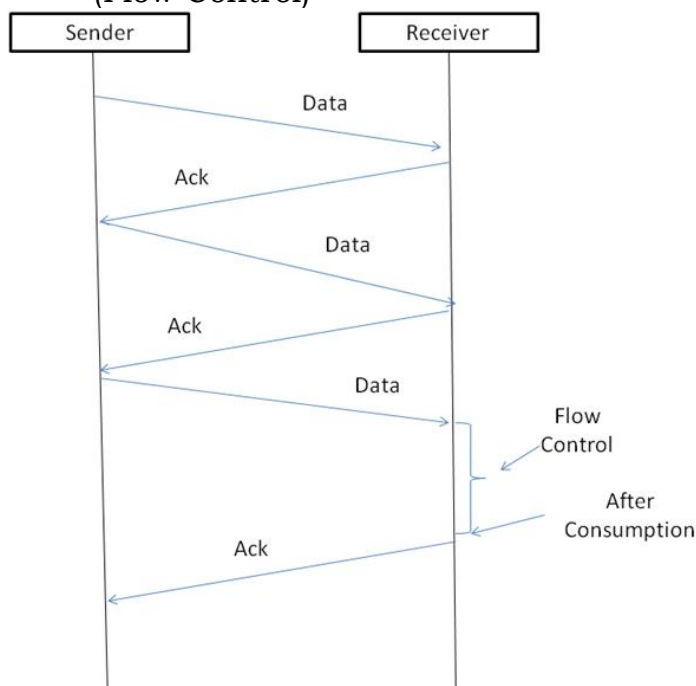
Simple Stop and Wait:

Sender:

- Rule 1) Send one data packet at a time.
- Rule 2) Send next packet only after receiving acknowledgement for previous.

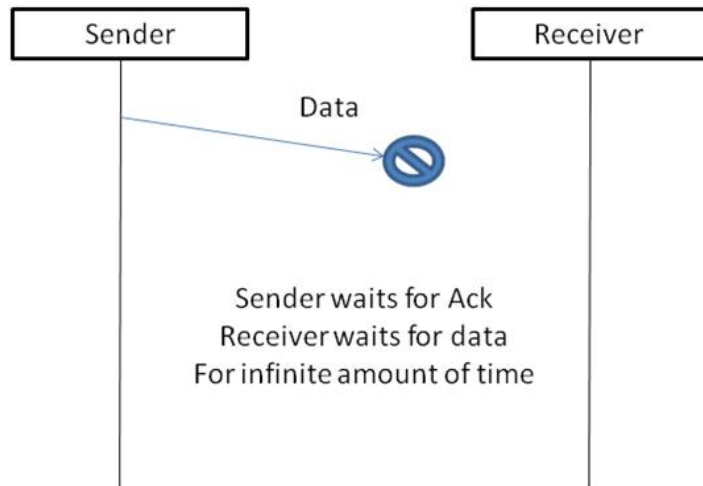
Receiver:

- Rule 1) Send acknowledgement after receiving and consuming of data packet.
- Rule 2) After consuming packet acknowledgement need to be sent (Flow Control)

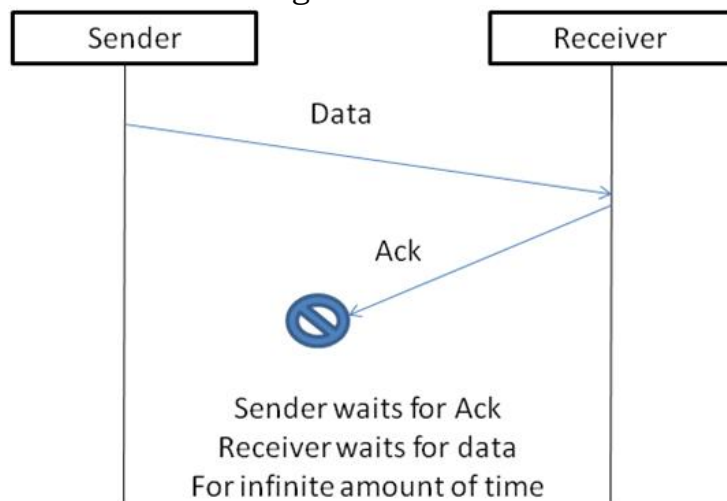


Problems:

1. Lost Data



2. Lost Acknowledgement:

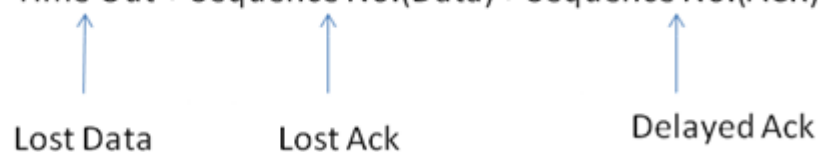


3. Delayed Acknowledgement/Data: After timeout on sender side, a long delayed acknowledgement might be wrongly considered as acknowledgement of some other recent packet.

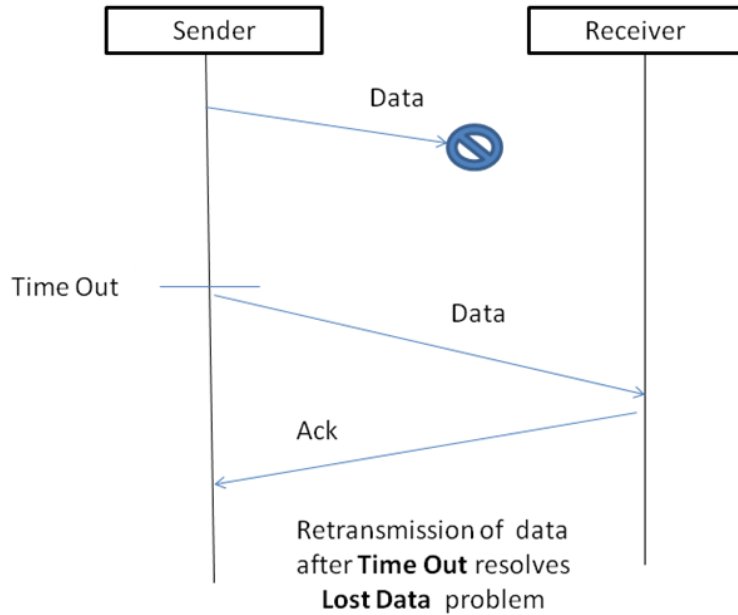
Stop and Wait for ARQ (Automatic Repeat Request)

Above 3 problems are resolved by Stop and Wait for ARQ (Automatic Repeat Request) that does both error control and flow control.

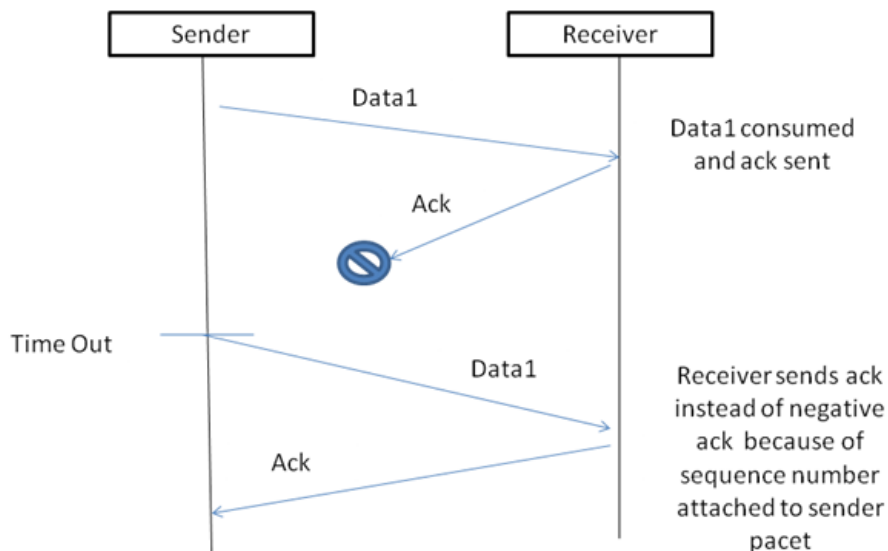
Stop (and) Wait + Time Out + Sequence No.(Data) + Sequence No.(ACK)



1. Time Out:



2. Sequence Number (Data)



3. Delayed Acknowledgement:

This is resolved by introducing sequence number for acknowledgment also.

Working of Stop and Wait for ARQ:

- 1) Sender A sends a data frame or packet with sequence number 0.
- 2) Receiver B, after receiving data frame, sends an acknowledgment with sequence number 1 (sequence number of next expected data frame or packet)

There is only one-bit sequence number that implies that both sender and receiver have a buffer for one frame or packet only.

SOURCE CODE:

Sender:

```
#include <stdio.h>
int i;
int main() {
    int flag = 0;
    char frame[8];
    FILE *f1, *f2;
    while(1) {
        f1 = fopen("file1.txt", "r");
        char ch = fgetc(f1);
        fclose(f1);
        if(ch == '1') {
            printf("\nDidn't received ACK in time.");
        }
        if(ch == '0') {
            printf((flag!=0)? "\nACK received\n": "");
            flag = 1;
            printf("\nEnter frame data: ");
            scanf("%s", frame);

            f2 = fopen("file2.txt", "w");
            fprintf(f2, "%s", frame);
            fclose(f2);

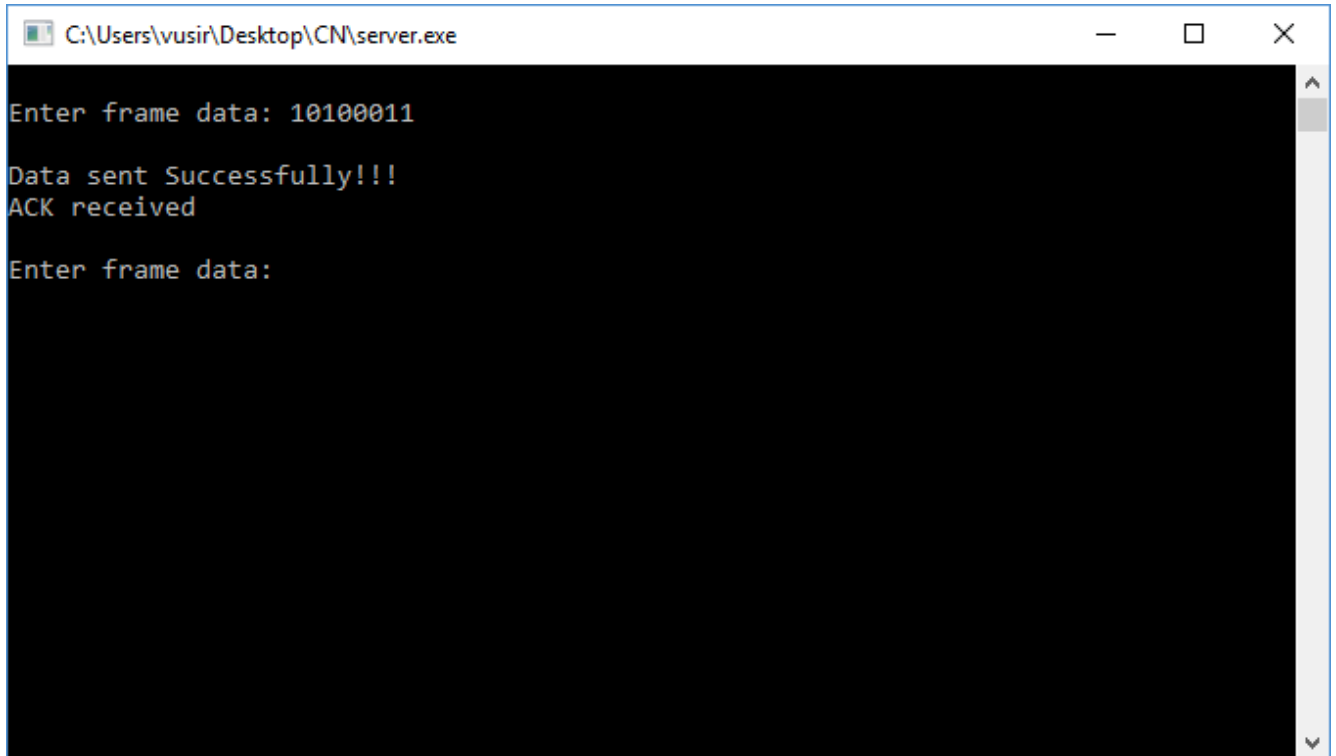
            f1 = fopen("file1.txt", "w");
            putc('1', f1);
            fclose(f1);
            Sleep(1000);
            printf("\nData sent Successfully!!!");
            Sleep(1000);
        }
    }
    return 0;
}
```

Receiver:

```
#include <stdio.h>
#include <malloc.h>
int i;
int main() {
    char * frame = (char*)malloc(9);
    FILE *f1, *f2;
    while(1) {
        f1 = fopen("file1.txt", "r");
        char ch = fgetc(f1);
        if(ch == '1') {
            f2 = fopen("file2.txt", "r");
            fscanf(f2, "%s", frame);
            fclose(f2);

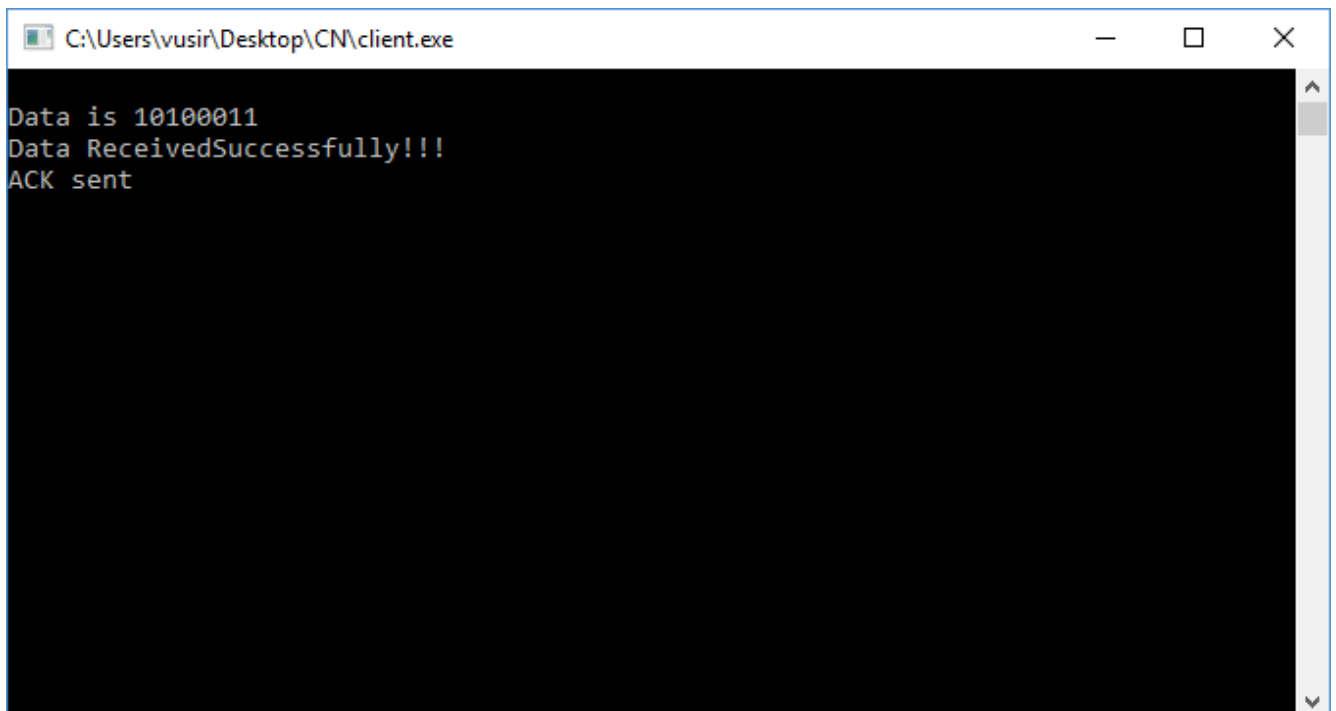
            printf("\nData is ");
            puts(frame);
            f1 = fopen("file1.txt", "w");
            putc('0', f1);
            printf("Data Received Successfully!!!");
            Sleep(1000);
            printf("\nACK sent\n");
        }
        fclose(f1);
    }
    return 0;
}
```

OUTPUT:



```
C:\Users\vusir\Desktop\CN\server.exe

Enter frame data: 10100011
Data sent Successfully!!!
ACK received
Enter frame data:
```



```
C:\Users\vusir\Desktop\CN\client.exe

Data is 10100011
Data ReceivedSuccessfully!!!
ACK sent
```


QUESTION: Simulation of Go back N protocol

AIM: To Simulate of Go back N protocol

DESCRIPTION:

Go Back n is a connection-oriented protocol in which the transmitter has a window of sequence numbers that may be transmitted without acknowledgment. The receiver will only accept the next sequence number it is expecting - other sequence numbers are silently ignored.

The protocol simulation shows a time-sequence diagram with users A and B, protocol entities A and B that support them, and a communications medium that carries messages. Users request data transmissions with `DatReq(DATAN)`, and receive data transmissions as `DatInd(DATAN)`. Data messages are simply numbered `DATA0`, `DATA1`, etc. without explicit content. The transmitting protocol sends the protocol message `DT(n)` that gives only the sequence number, not the data. Once sequence numbers reach a maximum number (like 7), they wrap back round to 0. An acknowledgement `AK(n)` means that the `DT` message numbered `n` is the next one expected (i.e. all messages up to but not including this number have been received). Since sequence numbers wrap round, an acknowledgment with sequence number 1 refers to messages 0, 1, 7, 6, etc. Note that if a `DT` message is received again due to re-transmission, it is acknowledged but discarded.

The protocol has a maximum number of messages that can be sent without acknowledgment. If this window becomes full, the protocol is blocked until an acknowledgment is received for the earliest outstanding message. At this point, the transmitter is clear to send more messages.

The receiver delivers the protocol messages `DT(n)` to the user in order. Any received out of order are ignored.

SOURCE CODE:

Sender:

```
#include<stdio.h>

#include<stdlib.h>

int main(){
    FILE *fp = NULL;
    int n;
    printf("Enter window size: ");
    scanf("%d", &n);
    char frame[n][8];
    fp = fopen("file1.txt", "w");
    putc('O', fp);
    fclose(fp);
    while (1){
        fp = fopen("file1.txt", "r");
        char ch = fgetc(fp);
        fclose(fp);
        if (ch == 'O'){
            fp = fopen("file1.txt", "w");
            fprintf(fp, "5\n");
            printf("Enter %d frames of 4 bit: \n", n);
            int i;
            for (i = 0; i < n; i++){
                scanf("%s", frame[i]);
                fflush(stdin);
                fprintf(fp, frame[i]);
                fprintf(fp, "\n");
            }
            fclose(fp);
        }
        else if (ch == '-'){
            fp = fopen("file1.txt", "r");
```

```

        int pos;
        ch = fgetc(fp);
        ch = fgetc(fp);
        pos = ch - '0';
        fclose(fp);

        fp = fopen("file1.txt", "w");
        fprintf(fp, "1");
        fprintf(fp, "\n");
        printf("Retransmission of following frames\n");
        int i;
        for (i = pos; i < n; i++){
            fprintf(fp, frame[i]);
            printf("%s\n", frame[i]);
            fprintf(fp, "\n");
        }
        fclose(fp);
    }
}
return 0;
}

```

Receiver:

```
#include<stdio.h>

#include<stdlib.h>

int main(){

    FILE *fp = NULL;

    int n;

    printf("Enter window size: ");

    scanf("%d", &n);

    while (1){

        fp = fopen("file1.txt", "r");

        char ch = fgetc(fp);

        fclose(fp);

        if (ch >= '1'){

            fp = fopen("file1.txt", "r");

            char ch = fgetc(fp);

            while ((ch = fgetc(fp)) != EOF)

                printf("%c", ch);

            int error_pos;

            printf("\nEnter the number of frame that is error, -1 for
no error\n");

            scanf("%d", &error_pos);

            if (error_pos > -1 && error_pos < n){

                fp = fopen("file1.txt", "w");

                fprintf(fp, "-");

                char p[2];

                itoa(error_pos, p, 10);

                fprintf(fp, p);

                fclose(fp);

            }

            if (error_pos == -1){

                fp = fopen("file1.txt", "w");

                fprintf(fp, "0");

            }

        }

    }

}
```

```
        fclose(fp);
    }
}
return 0;
}
```

OUTPUT:

```
C:\Users\vusir\Desktop\CN\go_back_n\server.exe
Enter window size: 4
Enter 4 frames of 4 bit
1010
1100
1011
0111
Retransmission of following frames
1011
0111
Enter 4 frames of 4 bit
```

```
C:\Users\vusir\Desktop\CN\go_back_n\client.exe
Enter window size: 4

1010
1100
1011
0111

Enter the number of frame that is error, -1 for no error
2

1011
0111

Enter the number of frame that is error, -1 for no error
-1
```

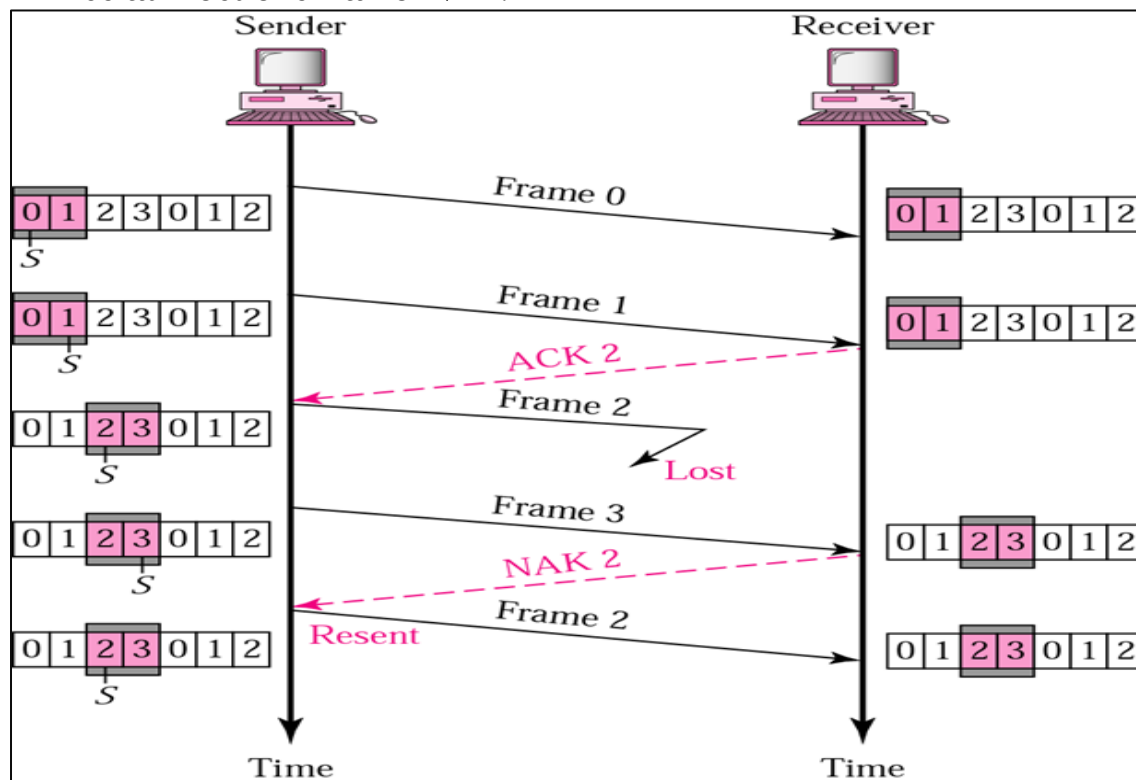
QUESTION: Simulation of Selective Repeat protocol

AIM: To Simulate Selective Repeat protocol

DESCRIPTION:

This protocol (SRP) is mostly identical to GBN protocol, except that buffers are used and the receiver, and the sender, each maintains a window of size. SRP works better when the link is very unreliable. Because in this case, retransmission tends to happen more frequently, selectively retransmitting frames is more efficient than retransmitting all of them. SRP also requires a full duplex link. backward acknowledgments are also in progress.

- Sender's Windows (Ws) = Receiver's Windows (Wr).
- Window size should be less than or equal to half the sequence number in SR protocol. This is to avoid packets being recognized incorrectly. If the window size is greater than half the sequence number space, then if an ACK is lost, the sender may send new packets that the receiver believes are retransmissions.
- The sender can transmit new packets as long as their number is within W of all unACKed packets.
- Sender retransmits un-ACKed packets after a timeout – Or upon a NAK if NAK is employed.
- Receiver ACKs all correct packets.
- Receiver stores correct packets until they can be delivered in order to the higher layer.
- In Selective Repeat ARQ, the size of the sender and receiver window must be at most one-half of 2^m .



SOURCE CODE:

Sender:

```
#include<stdio.h>
#include<stdlib.h>
int main(){
    FILE *fp = NULL;
    int n;
    printf("Enter window size: ");
    scanf("%d", &n);
    char frame[n][8];
    fp = fopen("file1.txt", "w");
    putc('O', fp);
    fclose(fp);
    while (1){
        fp = fopen("file1.txt", "r");
        char ch = fgetc(fp);
        fclose(fp);
        if (ch == 'O'){
            fp = fopen("file1.txt", "w");
            fprintf(fp, "%d\n", n);
            printf("Enter %d frames of 4 bit: \n", n);
            int i;
            for (i = 0; i < n; i++){
                scanf("%s", frame[i]);
                fflush(stdin);
                fprintf(fp, frame[i]);
                fprintf(fp, "\n");
            }
            fclose(fp);
        }
        else if (ch == '-') {
            fp = fopen("file1.txt", "r");
            int pos;
            ch = fgetc(fp);
            ch = fgetc(fp);
            pos = ch - 'O';
            fclose(fp);

            fp = fopen("file1.txt", "w");
            fprintf(fp, "1\n");
            printf("Retransmission of following frames\n");
            fprintf(fp, frame[pos]);
            printf("%s\n", frame[pos]);
            fprintf(fp, "\n");
            fclose(fp);
        }
    }
    return 0;
}
```


Receiver Code:

```
#include<stdio.h>
#include<stdlib.h>
int main(){
    FILE *fp = NULL;
    int n;
    printf("Enter window size: ");
    scanf("%d", &n);
    while (1){
        fp = fopen("file1.txt", "r");
        char ch = fgetc(fp);
        fclose(fp);
        if (ch >= '1'){
            fp = fopen("file1.txt", "r");
            char ch = fgetc(fp);
            while ((ch = fgetc(fp))!= EOF)
                printf("%c", ch);
            int error_pos;
            printf("\nEnter the number of frame that is error, -1 for
no error\n");
            scanf("%d", &error_pos);
            if (error_pos >-1 && error_pos<n){
                fp = fopen("file1.txt", "w");
                fprintf(fp, "-%d", error_pos);
                fclose(fp);
            }
            if (error_pos == -1){
                fp = fopen("file1.txt", "w");
                fprintf(fp, "0");
                fclose(fp);
            }
        }
    }
    return 0;
}
```

OUTPUT:

```
C:\Users\vusir\Desktop\CN\selective_repeat\server.exe
Enter window size: 4
Enter 4 frames of 4 bit:
1010
1100
0011
1011
Retransmission of following frames
0011
Retransmission of following frames
1011
Enter 4 frames of 4 bit:
```

```
C:\Users\vusir\Desktop\CN\selective_repeat\client.exe
Enter window size: 4
1010
1100
0011
1011

Enter the number of frame that is error, -1 for no error
2

0011

Enter the number of frame that is error, -1 for no error
3

1011

Enter the number of frame that is error, -1 for no error
-1
```

WEEK 4

QUESTION: Configure a network using Distance Vector Routing Algorithm.

AIM: To Configure a network using Distance Vector Routing Algorithm

DESCRIPTION:

In computer communication theory relating to packet-switched networks, a distance-vector routing protocol is one of the two major classes of intradomain routing protocols, the other major class being the link-state protocol. Distance-vector routing protocols use the Bellman-Ford algorithm, Ford-Fulkerson algorithm, or DUAL FSM (in the case of Cisco System's protocols) to calculate paths.

A distance-vector routing protocol requires that a router inform its neighbors of topology changes periodically. Compared to link-state protocols, which require a router to inform all the nodes in a network of topology changes, distance-vector routing protocols have less computational complexity and message overhead.^[citation needed]

The term *distance vector* refers to the fact that the protocol manipulates *vectors* (arrays) of distances to other nodes in the network. The distance vector algorithm was the original ARPANET routing algorithm and was also used on the Internet under the name of RIP (Routing Information Protocol).

Examples of distance-vector routing protocols include RIPv1 and RIPv2, IGRP, EIGRP, and Babel.

Distance vector routing protocols, such as RIP, are susceptible to a convergence problem known as the count-to-infinity problem. This problem is a consequence of the fact that distance vector routing protocols exchange routing information only with their neighbors. Here, it may happen that, after the failure of a link, information about routes that use the failed link are propagated a long time after the failure has occurred. This results in a slow convergence of the routing tables. Each time the router exchange RIP packets, the cost of a path that uses the failed link increases, but it takes a long time until all routers realize that routes through the failed link is unavailable.

The goal of this part of the lab is to observe the count-to-infinity problem. RIP has a number of protocol features that try to avoid the count-to-infinity problem. These features will be disabled. Still, since the count-to-infinity problem requires that routing updates occur in a certain order, the count-to-infinity problem is not always observable

SOURCE CODE:

```
#include<stdio.h>

struct node
{
    unsigned dist[20];
    unsigned from[20];
}rt[10];

int main()
{
    int costmat[20][20];
    int nodes,i,j,k,count=0;
    printf("\nEnter the number of nodes : ");
    scanf("%d",&nodes); //Enter the nodes
    printf("\nEnter the cost matrix :\n");
    for(i=0;i<nodes;i++)
    {
        for(j=0;j<nodes;j++)
        {
            scanf("%d",&costmat[i][j]);
            costmat[i][i]=0;
            rt[i].dist[j]=costmat[i][j]; //initialise the distance equal to cost matrix
            rt[i].from[j]=j;
        }
    }
    do
    {
        count=0;
        for(i=0;i<nodes;i++) //We choose arbitrary vertex k and we calculate
the direct distance from the node i to k using the cost matrix

        //and add the distance from k to node j
        for(j=0;j<nodes;j++)
        for(k=0;k<nodes;k++)
```

```

        if(rt[i].dist[j]>costmat[i][k]+rt[k].dist[j])
        { // We calculate the minimum distance
            rt[i].dist[j]=rt[i].dist[k]+rt[k].dist[j];
            rt[i].from[j]=k;
            count++;
        }break;
    }while(count!=0);
    for(i=0;i<nodes;i++)
    {
        printf("\n\n For router %d\n",i+1);
        for(j=0;j<nodes;j++)
        {
            printf("\t\nnode %d via %d Distance %d\n",j+1,rt[i].from[j]+1,rt[i].dist[j]);
        }
    }
    printf("the finalmatrix is :");
    printf("\n");
    for(i=0;i<nodes;i++)
    {
        for(j=0;j<nodes;j++)
        {
            printf("\t %d",rt[i].dist[j]);
        }
        printf("\n");
    }
    printf("\n\n");
    return 0;
}

```

OUTPUT:

The screenshot shows the Dev-C++ IDE with the file `C:\Users\hymavathi\Desktop\divvec.c` open. The program is running, and the output window displays the following text:

```
Enter the number of nodes : 2
Enter the cost matrix :
0 6
3 0

For router 1
node 1 via 1 Distance 0
node 2 via 2 Distance 6

For router 2
node 1 via 1 Distance 3
node 2 via 2 Distance 0 the finalmatrix is :
0 6
3 0
```

The IDE interface includes a menu bar (File, Edit, Search, View, Project, Execute, Tools, AStyle, Window, Help), a toolbar, and a status bar at the bottom showing line and column numbers.

The screenshot shows the Dev-C++ IDE with the file `C:\Users\hymavathi\Desktop\divvec.c` open. The program is running, and the output window displays the following text:

```
Enter the number of nodes : 3
Enter the cost matrix :
0 2 5
1 0 3
5 4 0

For router 1
node 1 via 1 Distance 0
node 2 via 2 Distance 2
node 3 via 2 Distance 5

For router 2
node 1 via 1 Distance 1
node 2 via 2 Distance 0
node 3 via 3 Distance 3

For router 3
node 1 via 1 Distance 5
node 2 via 2 Distance 4
node 3 via 3 Distance 0 the finalmatrix is :
0 2 5
1 0 3
5 4 0
```

The IDE interface includes a menu bar (File, Edit, Search, View, Project, Execute, Tools, AStyle, Window, Help), a toolbar, and a status bar at the bottom showing line and column numbers. A warning message is visible at the bottom of the output window:

```
- warnings: 0
- Output Filename: C:\Users\hymavathi\Desktop\divvec.exe
- Output Size: 130.6298828125 K1B
- Compilation Time: 1.98s
```

WEEK 5

QUESTION: Implement Address Resolution Protocol (ARP) and Reverse Address Resolution Protocol (RARP).

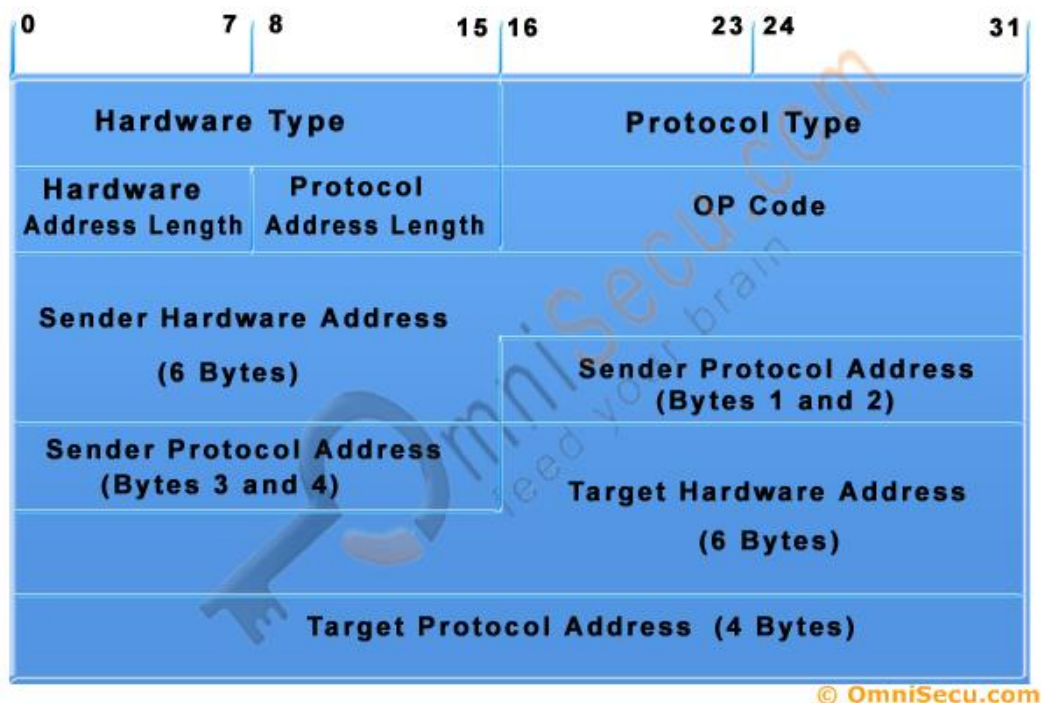
AIM: To implement Address Resolution Protocol (ARP) and Reverse Address Resolution Protocol (RARP).

DESCRIPTION:

ARP:

The address resolution protocol (arp) is a protocol used by the Internet Protocol (IP) [RFC826], specifically IPv4, to map IP network addresses to the hardware addresses used by a data link protocol. The protocol operates below the network layer as a part of the interface between the OSI network and OSI link layer. It is used when IPv4 is used over Ethernet.

The term address resolution refers to the process of finding an address of a computer in a network. The address is "resolved" using a protocol in which a piece of information is sent by a client process executing on the local computer to a server process executing on a remote computer. The information received by the server allows the server to uniquely identify the network system for which the address was required and therefore to provide the required address. The address resolution procedure is completed when the client receives a response from the server containing the required address.



An Ethernet network uses two hardware addresses which identify the source and destination of each frame sent by the Ethernet. The destination address (all 1's) may also identify a broadcast packet (to be sent to all connected computers). The hardware address is also known as the Medium Access Control (MAC) address, in reference to the standards which define Ethernet. Each computer network interface card is allocated a globally unique 6 byte link address when the factory manufactures the card (stored in a PROM). This

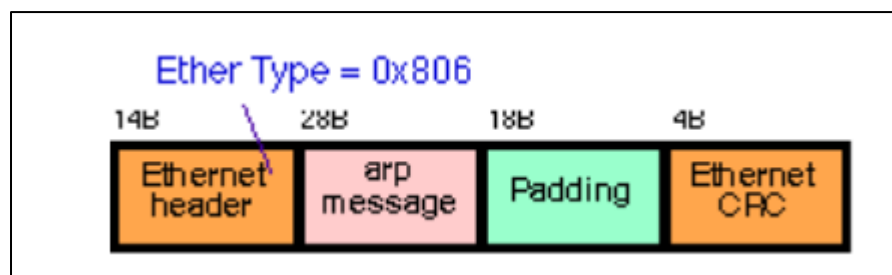
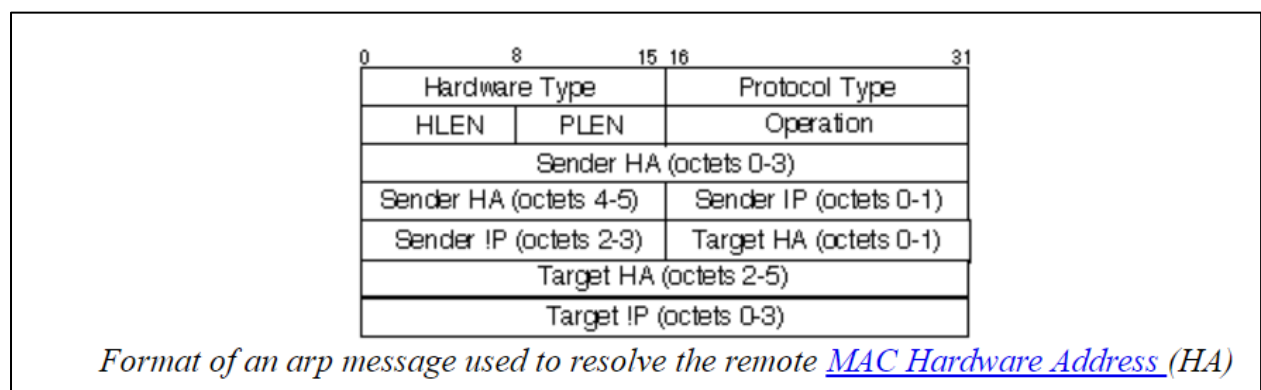
is the normal link source address used by an interface. A computer sends all packets which it creates with its own hardware source link address and receives all packets which match the same hardware address in the destination field or one (or more) pre-selected broadcast/multicast addresses.

The Ethernet address is a link layer address and is dependent on the interface card which is used. IP operates at the network layer and is not concerned with the link addresses of individual nodes which are to be used. The address resolution protocol (arp) is therefore used to translate between the two types of address. The arp client and server processes operate on all computers using IP over Ethernet. The processes are normally implemented as part of the software driver that drives the network interface card.

There are four types of arp messages that may be sent by the arp protocol. These are identified by four values in the "operation" field of an arp message. The types of message are:

1. ARP request
2. ARP reply
3. RARP request
4. RARP reply

The format of an arp message is shown below:



What is it for Arp translates IP numbers into hardware addresses?

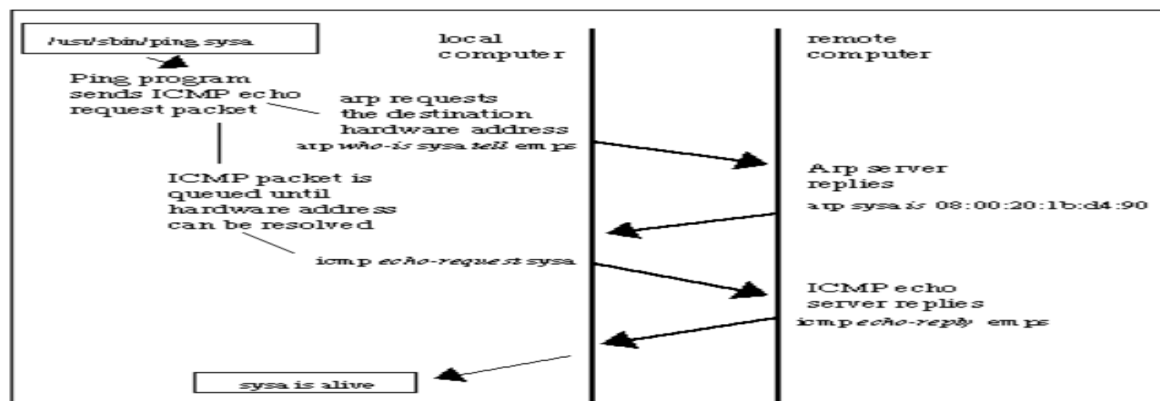
How ARP works: Send a packet from the querying host with an Ethernet broadcast address asking the target host with the given IP address to respond. All hosts on the physical network receive this packet, and the one with the given IP number responds. Then the original querying host knows the physical address of the target host. Does not use IP; uses's physical frames.

Common ARP improvements: Keep a cache of recently received translations. Remember that these addresses are quite small, and the space needed to store them is also small. Store both the physical and IP addresses of all ARP broadcasting hosts. Then every host who receives a broadcast ARP request can know the address translation of the sender. This is especially important for the receiver of the broadcast.

How to Write ARP Software: There are two parts. The first part uses the cache to map IP -> physical addresses. The second part fills the cache with mapping upon request from the first part.

Security: Can you fool ARP software. Yes, by polluting the network with your own answers.

Interesting question: To send machine A some data, you broadcast seeking machine A. Would it not be easier just to broadcast the data. That would for sure reduce the total number of packets sent, at the cost of changing many unicasts to broadcasts. What if someone answers an ARP request for you, and lies about who they are? Who answers an ARP for a machine.



RARP:

RARP (Reverse Address Resolution Protocol) is a protocol by which a physical machine in a local area network can request to learn its IP address from a gateway server's Address Resolution Protocol (ARP) table or cache. A network administrator creates a table in a local area network's gateway router that maps the physical machine (or Media Access Control - MAC address) addresses to corresponding Internet Protocol addresses. When a new machine is set up, its RARP client program requests from the RARP server on the router to be sent its IP address. Assuming that an entry has been set up in the router table, the RARP server will return the IP address to the machine which can store it for future use. RARP is available for Ethernet, Fiber Distributed-Data Interface, and token ring LANs. What is it for: Diskless clients don't have a place to store their IP number. Rarp translates machines addresses into IP numbers.

SOURCE CODE:

Client:

```
import java.io.*;
import java.net.*;
public class Carp
{
    public static void main(String args[])
    {
        try
        {
            Socket s=new Socket("localhost",80);
            DataOutputStream dos=new
            DataOutputStream(s.getOutputStream()); DataInputStream d=new
            DataInputStream(System.in); System.out.println("localhost ip");

            InetAddress ip=InetAddress.getLocalHost();
            System.out.println("IP:
            "+ip.getHostAddress()); String
            r=ip.getHostAddress(); dos.writeUTF(r);

            DataInputStream dis=new
            DataInputStream(s.getInputStream());

            dos.flush();
            dos.close();
            s.close();
        }
        catch(Exception e)
        {
            e.printStackTrace();
        }
    }
}
```

```

//Server side Source code--- ARP
import java.io.*;
import java.net.*;
public class Sarp
{
public static void main(String args[])
{
try
{
ServerSocket ss=new ServerSocket(80);
Socket s=ss.accept();
DataInputStream dis=new
DataInputStream(s.getInputStream()); String
str=(String)dis.readUTF(); System.out.println(str);

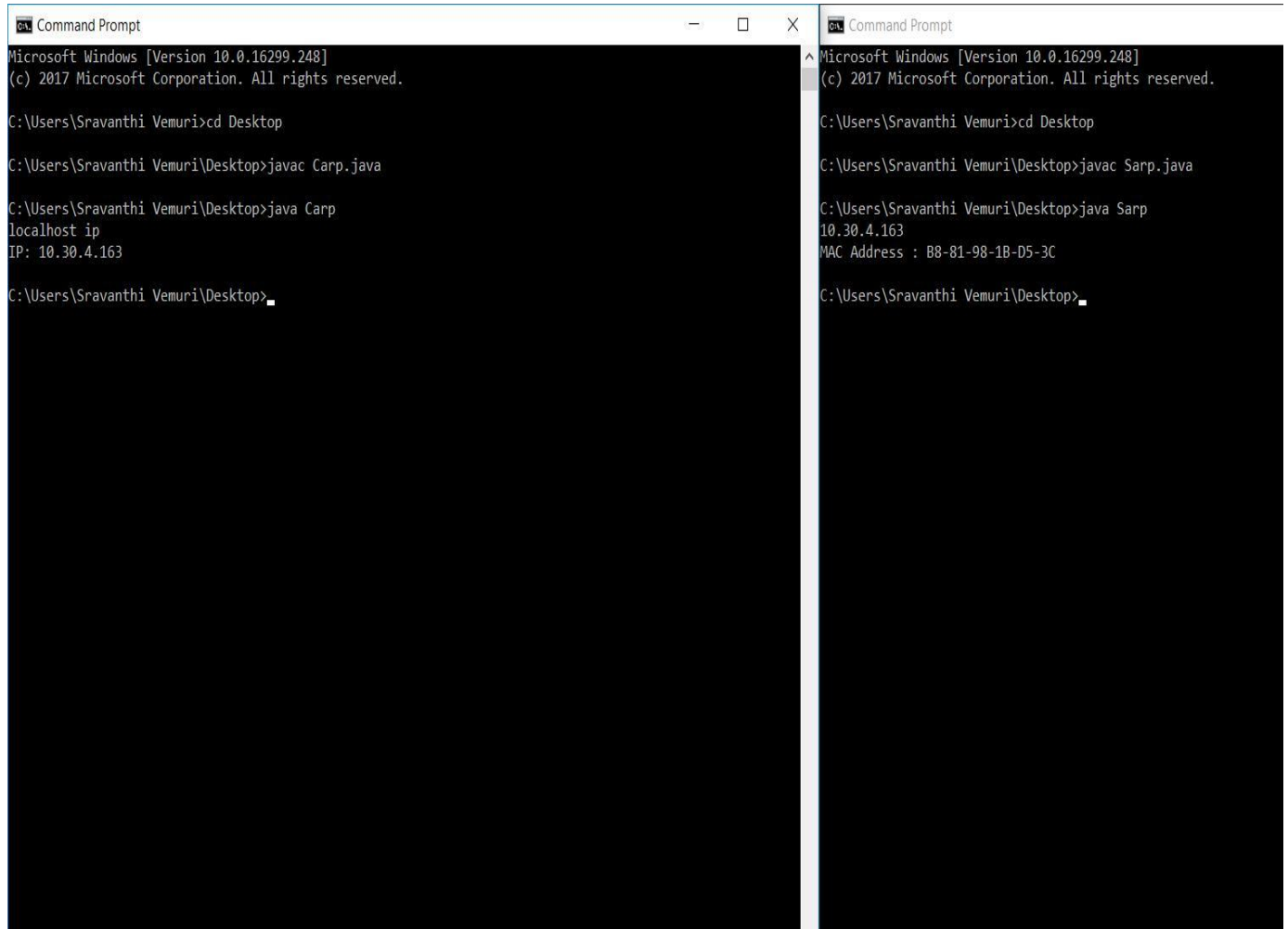
InetAddress ip=InetAddress.getByName(str);
DataOutputStream dos=new
DataOutputStream(s.getOutputStream()); NetworkInterface ni =
NetworkInterface.getByInetAddress(ip); if (ni!=null)
{
byte[] mac = ni.getHardwareAddress();
if (mac != null)
{

System.out.print("MAC Address : ");
for (int i=0; i<mac.length; i++)
{
System.out.format("%02X%s", mac[i], (i<mac.length - 1) ? "-" : "")
}
System.out.println();
}
else

```

```
{  
    System.out.println("Address doesn't exist or is not accessible/");  
}  
}  
else  
{  
    System.out.println("Network Interface for the specified address is not  
found");  
}  
ss.close();  
}  
catch(Exception e)  
{  
    e.printStackTrace();  
}  
}  
}
```

OUTPUT:



The image displays two side-by-side Windows Command Prompt windows. Both windows show the same initial text: "Microsoft Windows [Version 10.0.16299.248] (c) 2017 Microsoft Corporation. All rights reserved." and the directory path "C:\Users\Sravanthi Vemuri>cd Desktop".

The left window shows the execution of "javac Carp.java" followed by "java Carp", which outputs "localhost ip" and "IP: 10.30.4.163".

The right window shows the execution of "javac Sarp.java" followed by "java Sarp", which outputs "10.30.4.163" and "MAC Address : B8-81-98-1B-D5-3C".

```
Command Prompt
Microsoft Windows [Version 10.0.16299.248]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\Sravanthi Vemuri>cd Desktop

C:\Users\Sravanthi Vemuri\Desktop>javac Carp.java

C:\Users\Sravanthi Vemuri\Desktop>java Carp
localhost ip
IP: 10.30.4.163

C:\Users\Sravanthi Vemuri\Desktop>

Command Prompt
Microsoft Windows [Version 10.0.16299.248]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\Sravanthi Vemuri>cd Desktop

C:\Users\Sravanthi Vemuri\Desktop>javac Sarp.java

C:\Users\Sravanthi Vemuri\Desktop>java Sarp
10.30.4.163
MAC Address : B8-81-98-1B-D5-3C

C:\Users\Sravanthi Vemuri\Desktop>
```

```

//Source code—client---RARP
import java.io.*;
import java.net.*;
import java.util.*;
class Crarp
{
public static void main(String args[])
{
try
{
BufferedReader br=new BufferedReader(new
InputStreamReader(System.in)); Socket s=new Socket("localhost",80);

DataInputStream d=new DataInputStream(s.getInputStream());
DataOutputStream dos=new
DataOutputStream(s.getOutputStream());
System.out.println("Enter the Physical Addres (MAC):"); String
mac=br.readLine();

dos.writeBytes(mac+'\n');
s.close();
}
catch (Exception e)
{
System.out.println("Exception :"+e);
}
}
}

```

//Source code—Server side—RARP

```
import java.io.*;
import java.net.*;
import java.util.*;
class Srarp
{
public static void main(String args[])
{
try
{
ServerSocket ss=new ServerSocket(80);
Socket s=ss.accept();
while(true)
{
DataInputStream dis=new DataInputStream(s.getInputStream());
DataOutputStream dos=new
DataOutputStream(s.getOutputStream()); String str=dis.readLine();

String ip[]={"10.30.40.02","127.20.00.01"};
String mac[]={"0A-5B-D3-B2","5D-3C-2B-1A"};
for(int i=0;i<mac.length;i++) {

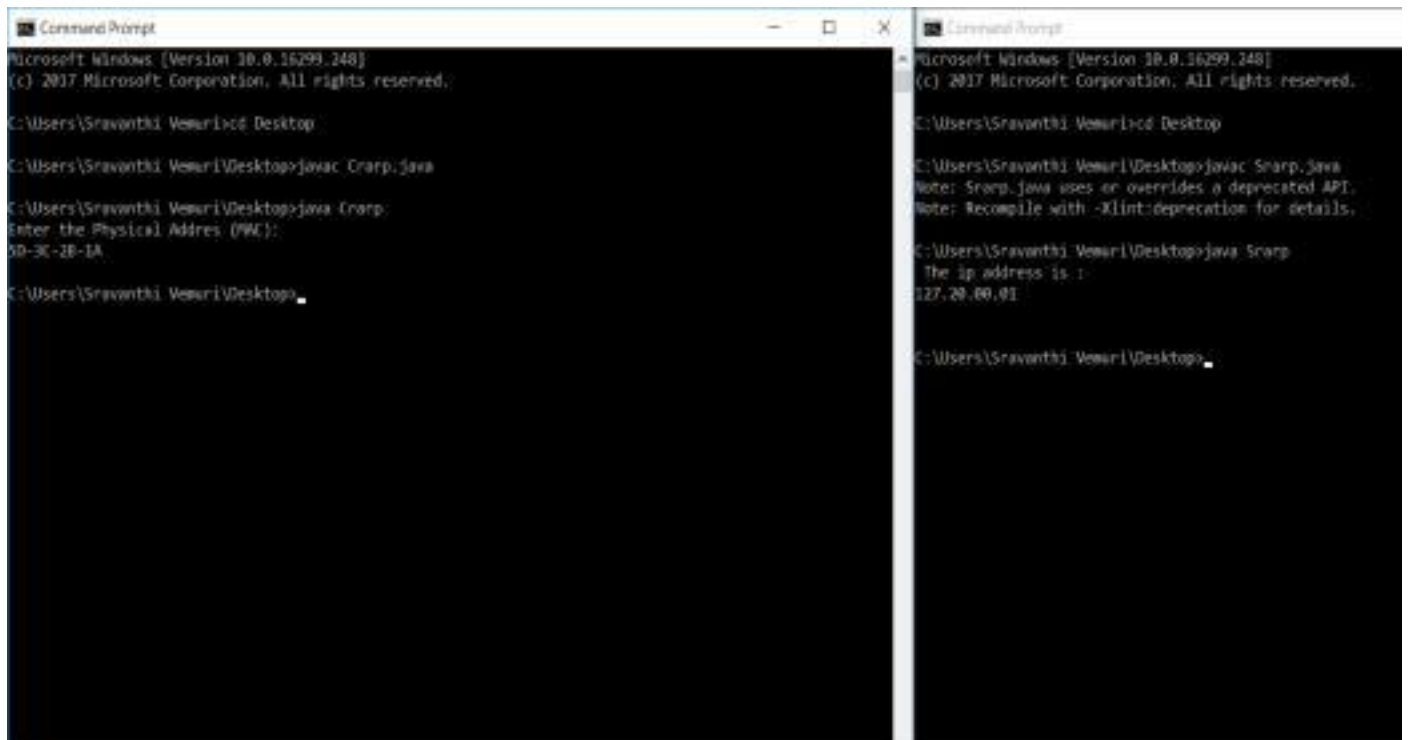
if(str.equals(mac[i]))
{
System.out.println(" The ip address
is:"); System.out.println(ip[i]); break;

}

}
```

```
}  
  
ss.close();  
}  
}  
catch(Exception e)  
{  
System.out.println();  
}  
}  
}
```


OUTPUT:

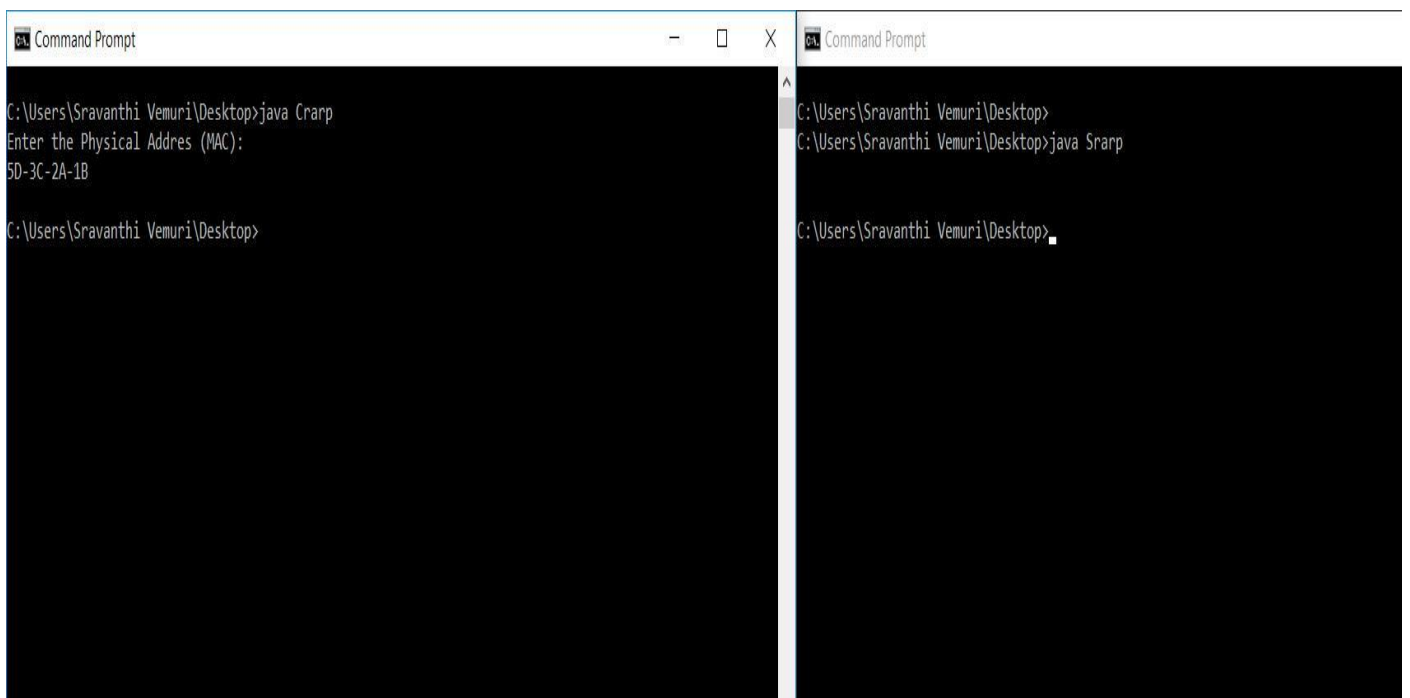


```
Microsoft Windows [Version 10.0.16299.248]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\Sravanthi Vemuri>cd Desktop
C:\Users\Sravanthi Vemuri\Desktop>javac Crarp.java
C:\Users\Sravanthi Vemuri\Desktop>java Crarp
Enter the Physical Address (MAC):
5D-3C-2B-1A
C:\Users\Sravanthi Vemuri\Desktop>
```

```
Microsoft Windows [Version 10.0.16299.248]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\Sravanthi Vemuri>cd Desktop
C:\Users\Sravanthi Vemuri\Desktop>javac Snarp.java
Note: Snarp.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
C:\Users\Sravanthi Vemuri\Desktop>java Snarp
The ip address is :
127.20.00.01
C:\Users\Sravanthi Vemuri\Desktop>
```



```
C:\Users\Sravanthi Vemuri\Desktop>java Crarp
Enter the Physical Address (MAC):
5D-3C-2A-1B
C:\Users\Sravanthi Vemuri\Desktop>
```

```
C:\Users\Sravanthi Vemuri\Desktop>
C:\Users\Sravanthi Vemuri\Desktop>java Snarp
C:\Users\Sravanthi Vemuri\Desktop>
```

WEEK 6

QUESTION: Using RSA algorithm Encrypt the data while sending it and Decrypt while receiving.

AIM: Using RSA algorithm Encrypt the data while sending it and Decrypt while receiving.

DESCRIPTION:

Encryption is a one of the ways to achieve data security. Encryption is converting the data in the plain text into an unreadable text called the cipher text. *Decryption* is converting the cipher text back to plain text.

This encryption/decryption of data is part of *cryptography*. Encryption and decryption generally require the use of some secret information, referred to as a *key*, which is used in converting plain text to cipher text and vice versa.

Symmetric key cryptography refers to encryption methods in which both the sender and receiver share the same key.

Asymmetric key cryptography (also known as public key cryptography) refers to a system that requires two separate keys, one of which is secret and one of which is public. Although different, the two parts of the key pair are mathematically linked. One key encrypts the plain text, and the other decrypts the cipher text. Neither key can perform both functions. The public key, or the key used to encrypt information can be freely distributed.

RSA is one of the algorithm for public-key cryptography that is based on factoring large integers. RSA stands for Ron Rivest, Adi Shamir and Leonard Adleman, who first publicly described it.

Following example shows how to encrypt/decrypt information using RSA algorithm in Java.

The *KeyPairGenerator* class instance is used to generate the pair of public and private key for RSA algorithm and are saved into the files.

The *Cipher* class instance is used encrypt/decrypt information using the pair of keys generated above.

SOURCE CODE:**SENDER:**

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
int main()
{
    int a=3,b=11,p,e,x,n;
    int z;
    int c;
    n=a*b;
    z=(a-1)*(b-1);
    int d=7;
    while((e*d)%z!=1)
        e++;
    FILE* fp=fopen("hai.txt","w");
    printf("encryption\n");
    printf("enter the message");
    scanf("%d",&p);
    x=pow(p,e);
    c=x%n;
    printf("encrypted msg is %d",c);
    fprintf(fp,"%d",c);
    return 0;
}
```

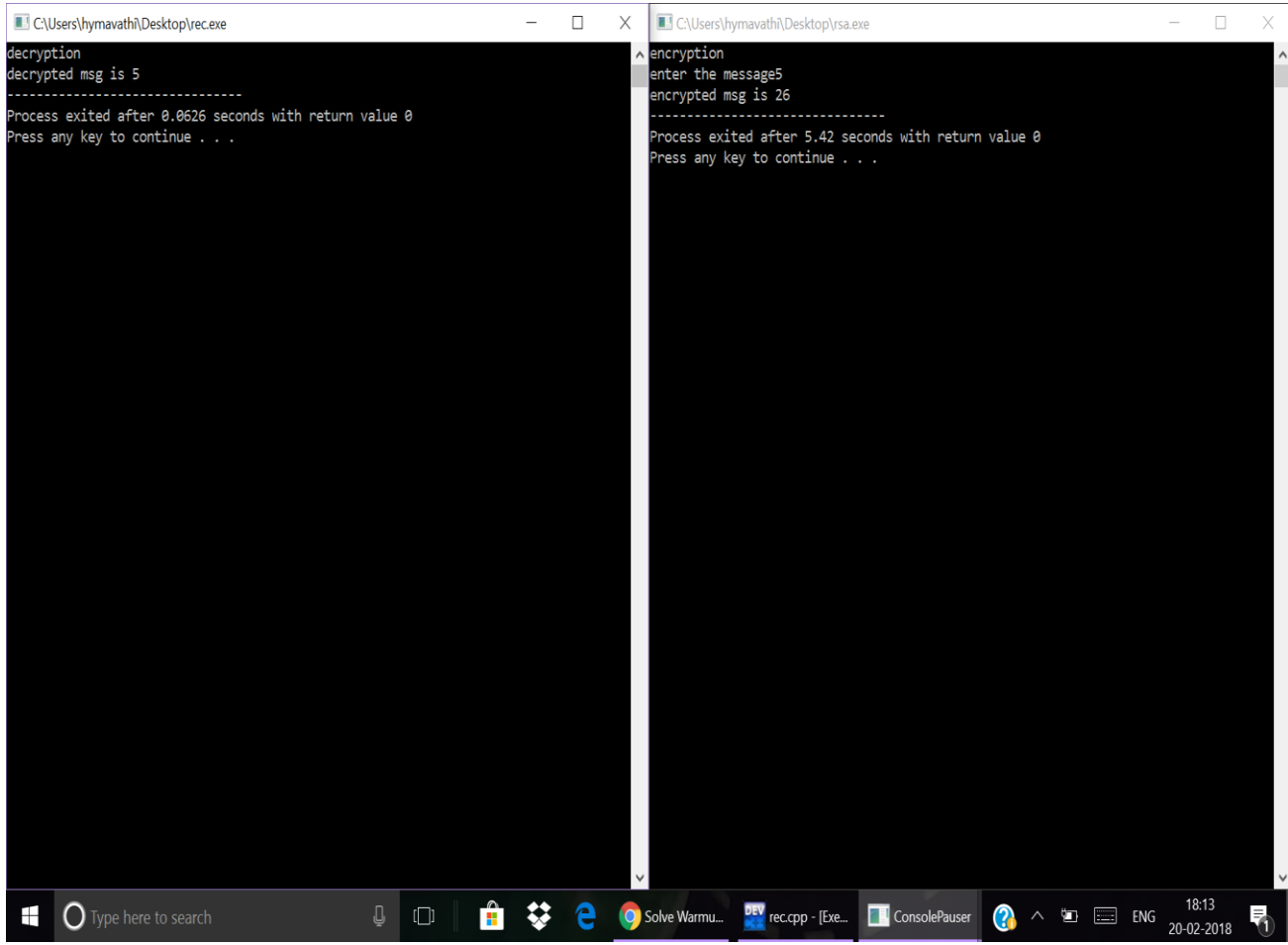
RECEIVER:

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
#include<stdlib.h>

int main()
{
    int a=3,b=11,p,n;
    int z;
    int c;
    n=a*b;
    int d=7;
    long long int x=0;
    FILE* fp=fopen("hai.txt","r");

    printf("decryption\n");
    char str[10];
    fgets(str,256,fp);
    c=atoi(str);
    x=pow(c,d);
    p=x%n;
    printf("decrypted msg is %d",p);
    return 0;
}
```

OUTPUT:



The image shows two side-by-side Windows command prompt windows. The left window, titled 'C:\Users\hymavathi\Desktop\rec.exe', displays the output of a decryption process. The right window, titled 'C:\Users\hymavathi\Desktop\rsa.exe', displays the output of an encryption process. Both windows show the message 'Process exited after 0.0626 seconds with return value 0' and 'Press any key to continue . . .'. The Windows taskbar at the bottom shows the search bar, task view button, and several open applications including 'Solve Warmu...', 'rec.cpp - [Exe...', and 'ConsolePauser'. The system clock indicates 18:13 on 20-02-2018.

```
C:\Users\hymavathi\Desktop\rec.exe
decryption
decrypted msg is 5
-----
Process exited after 0.0626 seconds with return value 0
Press any key to continue . . .

C:\Users\hymavathi\Desktop\rsa.exe
encryption
enter the message5
encrypted msg is 26
-----
Process exited after 5.42 seconds with return value 0
Press any key to continue . . .
```

WEEK 7

QUESTION: NS-2 Installation Procedure

AIM: Installing Network Simulator 2 (NS2) on Ubuntu 17.04

Introduction:

Network simulators are tools used to simulate discrete events in a network and which helps to predict the behaviors of a computer network. Generally, the simulated networks have entities like links, switches, hubs, applications, etc. Once the simulation model is complete, it is executed to analyse the performance. Administrators can then customize the simulator to suit their needs. Network simulators typically come with support for the most popular protocols and networks in use today, such as WLAN, UDP, TCP, IP, WAN, etc.

Most simulators that are available today are based on a GUI application like the NCTUNS while some others incl. NS2 are CLI based. Simulating the network involves configuring the state elements like links, switches, hubs, terminals, etc. and also the events like packet drop rate, delivery status and so on. The most important output of the simulations are the trace files. Trace files log every packet, every event that occurred in the simulation and are used for analysis. Network simulators can also provide other tools to facilitate visual analysis of trends and potential trouble spots. Most of the simulation is performed in discrete time intervals where events that are in the queue are processed one after the other in an order.

Since simulation is a complex task, we cannot guarantee that all the simulators can provide exact or accurate results for all the different type of information. Examples of network simulators are: ns, NCTUNS, NetSim, etc.

ns2 is a name for series of discrete event network simulators like ns-1, ns-2 and ns-3. All of them are discrete-event network simulators, primarily used in research and teaching. ns2 is free software, publicly available under the GNU GPLv2 license for research, development, and use.

This post deals with the installation of "ns2" also called the "network simulator 2" in Ubuntu 17.04.

PROCEDURE:

Download and Extract ns2:

Download the all in one package for ns2 from <http://sourceforge.net/projects/nsnam/files/latest/download>

The package downloaded will be named "ns-allinone-2.35.tar.gz". Copy it to the home folder. Then in a terminal use the following two commands to extract the contents of the package.:

```
cd ~/
tar -xvzf ns-allinone-2.35.tar.gz
```

All the files will be extracted into a folder called "ns-allinone-2.35".

Building the dependencies:

Ns2 requires a few packages to be pre installed. It also requires the GCC- version 4.3 to work correctly. So install all of them by using the following command:

```
sudo apt-get update

sudo apt-get dist-upgrade

sudo apt-get update

sudo apt-get install gcc-4.4

sudo apt-get install build-essential autoconf automake

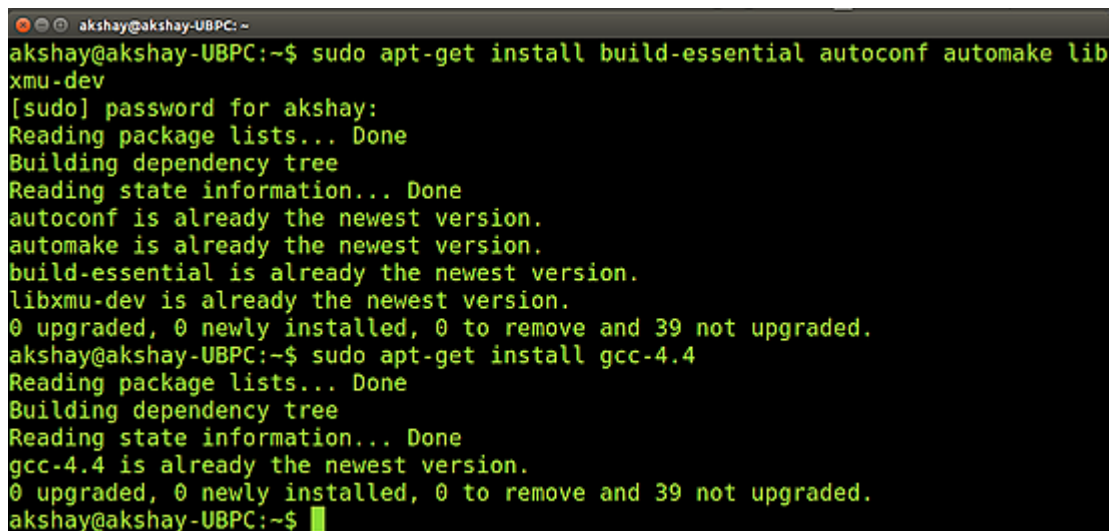
sudo apt-get install tcl8.5-dev tk8.5-dev

sudo apt-get install perl xgraph libxt-dev libx11-dev libxmu-dev
```

One of the dependencies mentioned is the compiler GCC-4.3, which is no longer available, and thus we have to install GCC-4.4 version. The version 4.4 is the oldest we can get. To do that, use the following command:

```
sudo apt-get install gcc-4.4
```

The image below shows the output of executing both the above commands. If you have all the dependencies pre-installed, as I did, the output will look like the image below:

A terminal window screenshot showing the execution of two apt-get commands. The first command is 'sudo apt-get install build-essential autoconf automake libxmu-dev'. The output shows that all these packages are already the newest versions. The second command is 'sudo apt-get install gcc-4.4'. The output shows that gcc-4.4 is also already the newest version. The terminal text is as follows:

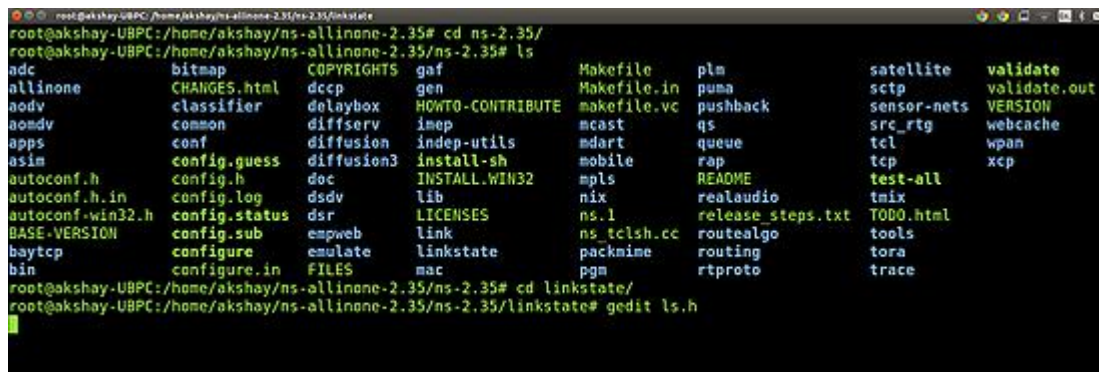
```
akshay@akshay-UBPC:~$ sudo apt-get install build-essential autoconf automake libxmu-dev
[sudo] password for akshay:
Reading package lists... Done
Building dependency tree
Reading state information... Done
autoconf is already the newest version.
automake is already the newest version.
build-essential is already the newest version.
libxmu-dev is already the newest version.
0 upgraded, 0 newly installed, 0 to remove and 39 not upgraded.
akshay@akshay-UBPC:~$ sudo apt-get install gcc-4.4
Reading package lists... Done
Building dependency tree
Reading state information... Done
gcc-4.4 is already the newest version.
0 upgraded, 0 newly installed, 0 to remove and 39 not upgraded.
akshay@akshay-UBPC:~$
```

Once the installation is over , we have to make a change in the "ls.h" file. Use the following steps to make the changes:
Navigate to the folder "linkstate", use the following command. Here it is assumed that the ns folder extracted is in the home folder of your system.

```
cd ~/ns-allinone-2.35/ns-2.35/linkstate
```

Now open the file named "ls.h" and scroll to the 137th line. In that change the word "**erase**" to "**this->erase**". The image below shows the line 137 (highlighted in the image below) after making the changes to the ls.h file.To open the file use the following command:

```
gedit ls.h
```



```
root@akshay-UBPC:/home/akshay/ns-allinone-2.35/ns-2.35/linkstate# ls
adc          bitmap      COPYRIGHTS  gaf          Makefile    plm          satellite   validate
allinone     CHANGES.html dccp        gen          Makefile.in puma        sctp        validate.out
aodv         classifier  delaybox    HOWTO-CONTRIBUTE Makefile.vc pushback    sensor-nets VERSION
aodv         common      diffserv    imcp         mcast       qs          src_rtg     webcache
apps         conf        diffusion    indep-utils  mdart       queue       tel         wpan
asim         config.guess diffusion3    install-sh   mobile      rap         tcp         xcp
autoconf.h   config.h     doc         INSTALL.WIN32 mpls        README      test-all
autoconf.h.in config.log    dsdv        lib          nix         realaudio   tmix
autoconf-win32.h config.status dsr          LICENSES     ns.l         release_steps.txt TODO.html
BASE-VERSION config.sub    espweb      link          ns_tclsh.cc routealgo   tools
baytcp       configure    emulate     linkstate     packmine    routing     tora
bin          configure.in FILES        mac           pgm          rtproto     trace
```

Save that file and close it.

```
// this next typedef of iterator seems extraneous but is required by gcc-2.96
typedef typename map<Key, T, less<Key> >::iterator iterator;
typedef pair<iterator, bool> pair_iterator_bool;
iterator insert(const Key & key, const T & item) {
    typename baseMap::value_type v(key, item);
    pair_iterator_bool ib = baseMap::insert(v);
    return ib.second ? ib.first : baseMap::end();
}

void eraseAll() { this->erase(baseMap::begin(), baseMap::end()); }
T* findPtr(Key key) {
    iterator it = baseMap::find(key);
    return (it == baseMap::end()) ? (T *)NULL : &((*it).second);
}
```

Now there is one more step that has to be done. We have to tell the ns which version of GCC will be used. To do so, go to your ns folder and type the following command:

```
Sudo gedit ns-allinone-2.35/otcl-1.14/Makefile.in
```



```

akshay@akshay-UBPC: ~/ns-allinone-2.35/otcl-1.14
akshay@akshay-UBPC:~/ns-allinone-2.35$ cd otcl-1.14/
akshay@akshay-UBPC:~/ns-allinone-2.35/otcl-1.14$ gedit Makefile.in

```

In the file, change `CC= @CC@` to `CC=gcc-4.4`, as shown in the image below.

```

*Makefile.in x
#
# try ./configure first to fill in all the definitions corresponding
# to your system, but you always can edit the sections below manually.
#
CC=      gcc-4.4
CFLAGS=  @CFLAGS@
RANLIB=  @RANLIB@
INSTALL= @INSTALL@

#
# how to compile, link, and name shared libraries
#
SHLIB_LD=    @SHLIB_LD@
SHLIB_CFLAGS= @SHLIB_CFLAGS@
SHLIB_SUFFIX= @SHLIB_SUFFIX@
SHLD_FLAGS=  @DL_LD_FLAGS@
DL_LIBS=     @DL_LIBS@

```

Installation:

Now we are ready to install ns2. To do so we first require root privileges and then we can run the install script. Use the following two commands:

```
sudo su cd ~/ns-allinone-2.35/./install
```

The following is a snap of these commands:

```

root@akshay-UBPC: /home/akshay/ns-allinone-2.35
akshay@akshay-UBPC:~$ cd ns-allinone-2.35/
akshay@akshay-UBPC:~/ns-allinone-2.35$ ls
cweb      install      ns-2.35      sgb          tk8.5.10
dei80211mr-1.1.4  INSTALL.WIN32  otcl-1.14    tcl8.5.10    xgraph-12.2
gt-itm     nam-1.15      README      tclcl-1.20   zlib-1.2.3
akshay@akshay-UBPC:~/ns-allinone-2.35$ sudo su
[sudo] password for akshay:
root@akshay-UBPC:/home/akshay/ns-allinone-2.35# ./install

```

The image below shows how it looks upon successful execution

```
Please put /home/akshay/ns-allinone-2.35/bin:/home/akshay/ns-allinone-2.35/tcl8.5.10/unix:/home/akshay/ns-allinone-2.35/tk8.5.10/unix
into your PATH environment; so that you'll be able to run itm/tclsh/wish/xgraph.

IMPORTANT NOTICES:

(1) You MUST put /home/akshay/ns-allinone-2.35/otcl-1.14, /home/akshay/ns-allinone-2.35/lib,
into your LD_LIBRARY_PATH environment variable.
If it complains about X libraries, add path to your X libraries
into LD_LIBRARY_PATH.
If you are using csh, you can set it like:
    setenv LD_LIBRARY_PATH <paths>
If you are using sh, you can set it like:
    export LD_LIBRARY_PATH=<paths>

(2) You MUST put /home/akshay/ns-allinone-2.35/tcl8.5.10/library into your TCL_LIBRARY environmental
variable. Otherwise ns/nam will complain during startup.

After these steps, you can now run the ns validation suite with
cd ns-2.35; ./validate

For trouble shooting, please first read ns problems page
http://www.isi.edu/nsnam/ns/ns-problems.html. Also search the ns mailing list archive
for related posts.

root@akshay-UBPC:/home/akshay/ns-allinone-2.35#
```

It took almost 6 minutes to build and install ns2 on my system. But before we run it, we need to add the build path to the environment path.

Setting the Environment Path:

The final step is to tell the system, where the files for ns2 are installed or present. To do that, we have to set the environment path using the ".bashrc" file. In that file, we need to add a few lines at the bottom. The things to be added are given below. But for the path indicated below, many of those lines have **"/home/akshay/ns-allinone-2.35/..."**, but that is where I have my extracted folder. Make sure you replace them with your path. For example, if you have installed it in a folder **"/home/abc"**, then replace **"/home/akshay/ns-allinone-2.35/otcl-1.14"** with **"/home/abc/ns-allinone-2.35/otcl-1.14"**.

Do this for all the required lines.

```
sudo gedit ~/.bashrc
```

Lines to be added:

```
# LD_LIBRARY_PATH
OTCL_LIB=/home/akshay/ns-allinone-2.35/otcl-1.14
NS2_LIB=/home/akshay/ns-allinone-2.35/lib
X11_LIB=/usr/X11R6/lib
USR_LOCAL_LIB=/usr/local/lib
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$OTCL_LIB:$NS2_LIB:$X11_LIB:$US
R_LOCAL_LIB
# TCL_LIBRARY
TCL_LIB=/home/akshay/ns-allinone-2.35/tcl8.5.10/library
USR_LIB=/usr/lib
```

```
export TCL_LIBRARY=$TCL_LIB:$USR_LIB
# PATH
XGRAPH=/home/akshay/ns-allinone-2.35/bin:/home/akshay/ns-allinone-2.35/tcl8.5.10/unix:/home/akshay/ns-allinone-2.35/tk8.5.10/unix
#the above two lines beginning from xgraph and ending with unix should come on the same line
NS=/home/akshay/ns-allinone-2.35/ns-2.35/
NAM=/home/akshay/ns-allinone-2.35/nam-1.15/
PATH=$PATH:$XGRAPH:$NS:$NAM
```

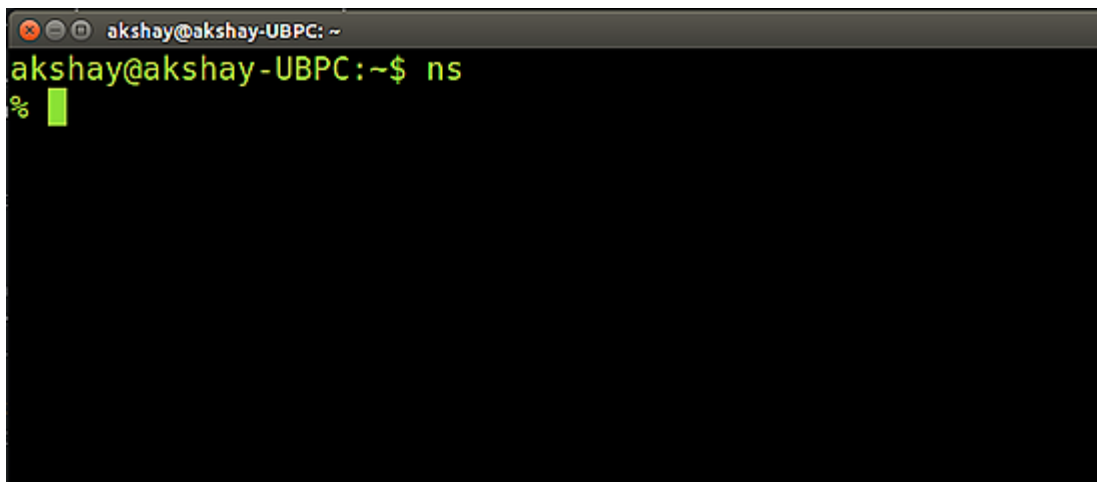
Once the changes have been made, save the file and restart the system.

Running ns2:

Once the system has restarted, open a terminal and start ns2 by using the following command:

```
ns
```

If the installation is correct then the terminal looks like the image below :



Steps to uninstall ns2:

Delete the directory 'ns-allinone-2.35'.

Delete the ns related files from '/user/local/bin'.

Edit the '.bashrc file' and remove the paths that you have added during the installation.

Restart the system and run following commands one by one

```
sudo apt-get autoremove
```

```
sudo apt-get clean
```

WEEK 8

QUESTION: Design a Sample Topology Using NS-2

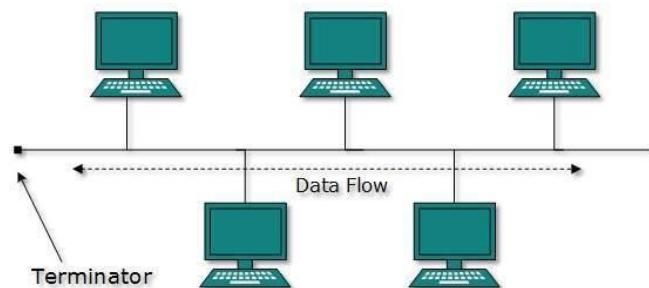
AIM: To Design a Sample Topology Using NS-2

DESCRIPTION:

Network Topology is the schematic description of a network arrangement, connecting various nodes (sender and receiver) through lines of connection. Network topology is the topological structure of a network and may be depicted physically or logically. *Physical topology* is the placement of the various components of a network, including device location and cable installation, while *logical topology* illustrates how data flows within a network. Distances between nodes, physical interconnections, transmission rates, or signal types may differ between two networks, yet their topologies may be identical. The various network topologies are:

1 BUS TOPOLOGY

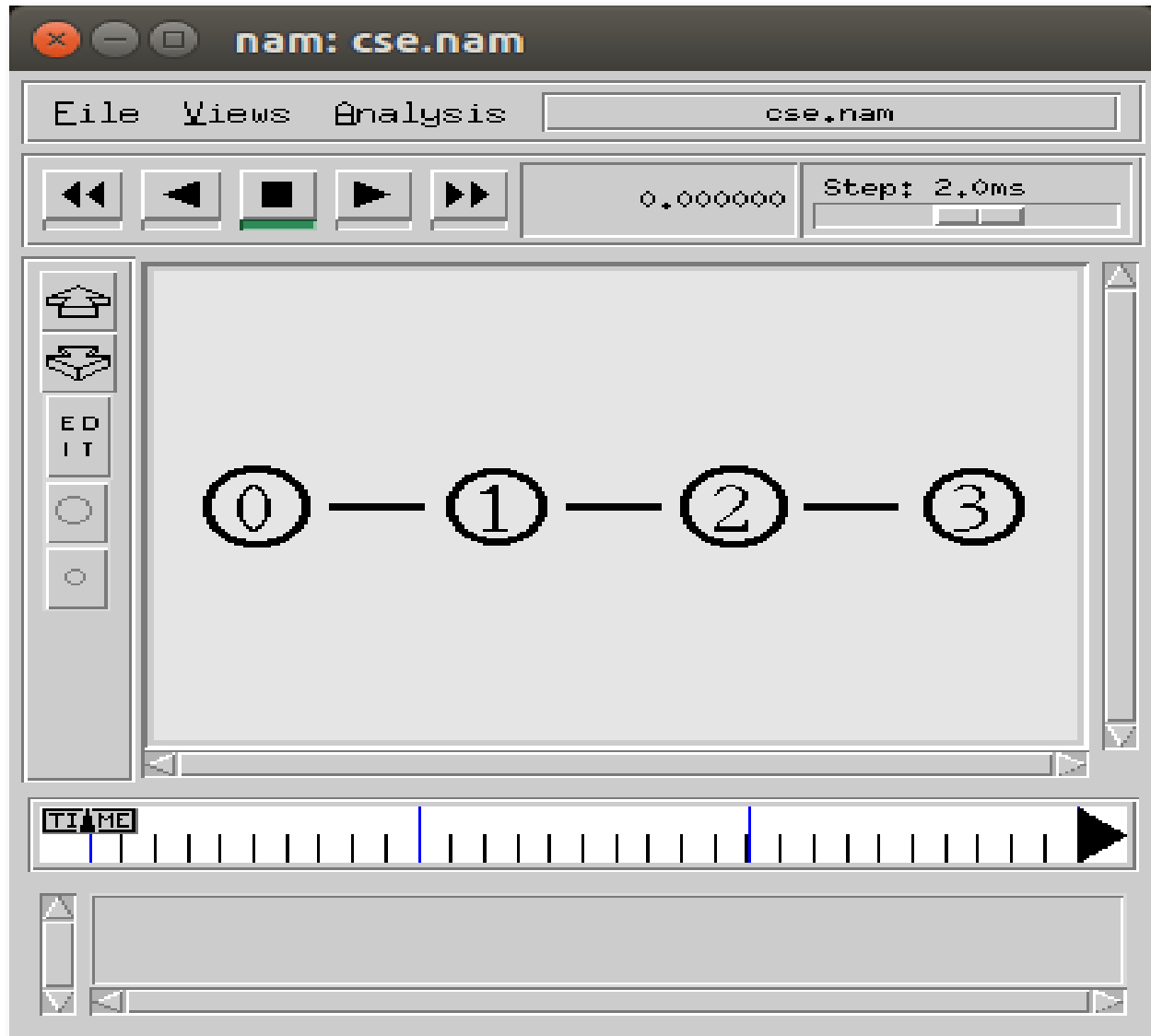
Bus networks share a common connection that extends to all devices. This network topology is used in small networks, and it is simple to understand. Every computer and network device connect to the same cable, so if the cable fails, the whole network is down, but the cost of setting up the network is reasonable. This type of networking is cost effective.



SOURCE CODE:

```
set ns [new Simulator]
set nf [open cse.nam w]
$ns namtrace-all $nf
set nd [open cse.tr w]
$ns trace-all $nd
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
$ns duplex-link $n1 $n2 10Mbps 10ms DropTail
$ns duplex-link $n2 $n3 10Mbps 10ms DropTail
$ns duplex-link $n3 $n4 10Mbps 10ms DropTail
$ns duplex-link-op $n1 $n2 orient right
$ns duplex-link-op $n2 $n3 orient right
$ns duplex-link-op $n3 $n4 orient right
proc finish {} {
global ns nf nd
$ns flush-trace
close $nf
exec nam cse.nam &
exit 0
}
$ns at 5.0 "finish"
$ns run
```

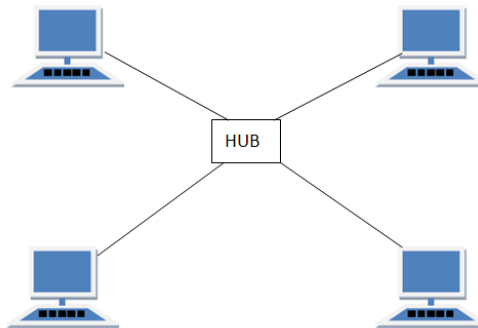
OUTPUT:



2 STAR TOPOLOGY

In the star network topology, there is a central computer or server to which all the workstations are directly connected. Every workstation is indirectly connected to every other through the central computer named Hub. As in Bus topology, hub acts as single point of failure. If hub fails, connectivity of all hosts to all other hosts fails.

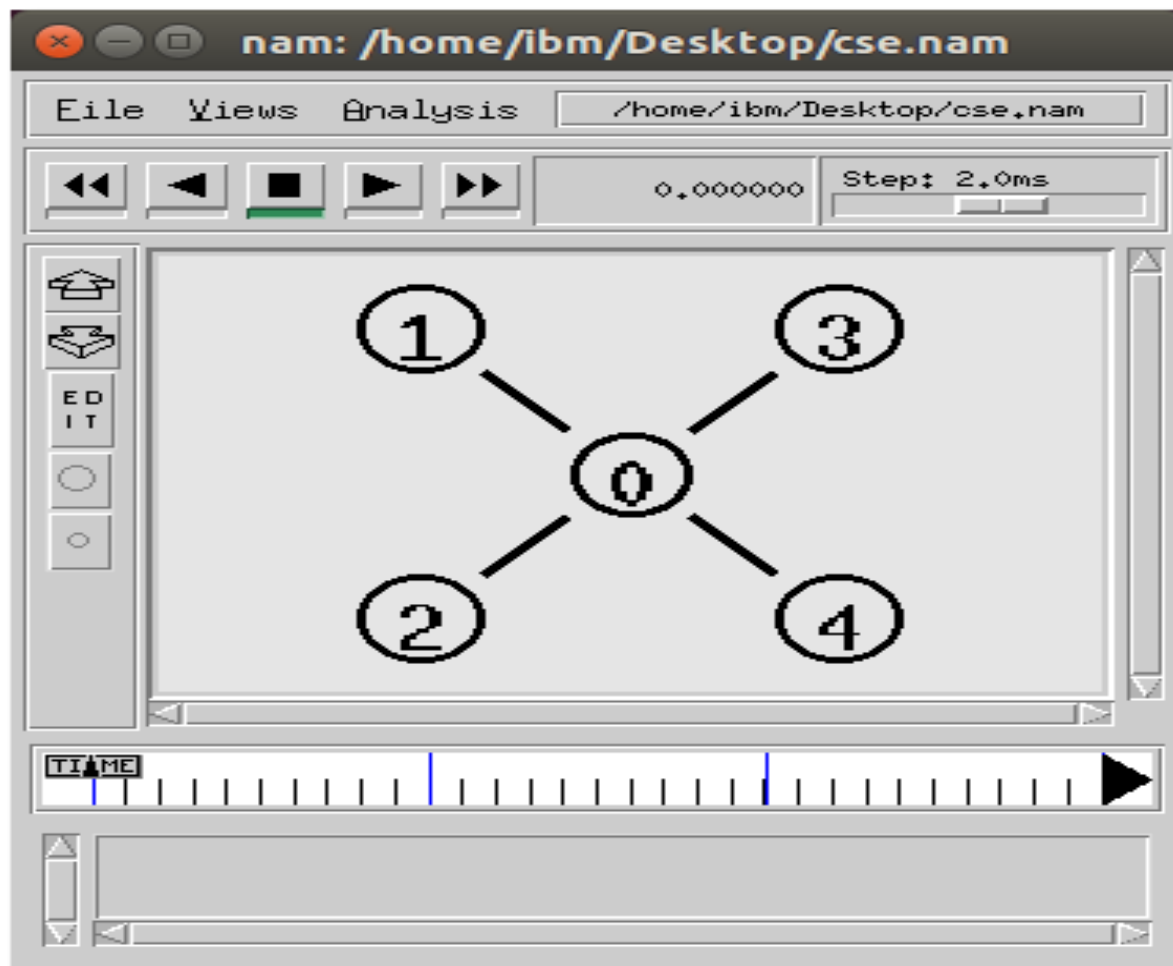
Every communication between hosts, takes place through only the hub. Star topology is not expensive as to connect one more host, only one cable is required and configuration is simple.



SOURCE CODE:

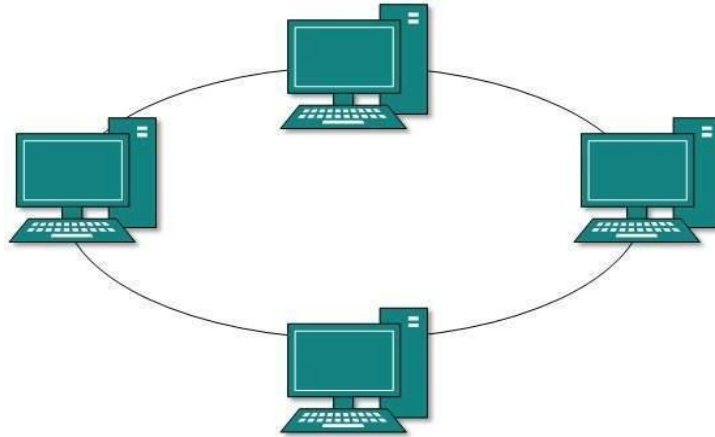
```
set ns [new Simulator]
set nf [open cse.nam w]
$ns namtrace-all $nf
set nd [open cse.tr w]
$ns trace-all $nd
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n0 [$ns node]
$ns duplex-link $n1 $n2 10Mbps 10ms DropTail
$ns duplex-link $n1 $n3 10Mbps 10ms DropTail
$ns duplex-link $n1 $n4 10Mbps 10ms DropTail
$ns duplex-link $n1 $n0 10Mbps 10ms DropTail
$ns duplex-link-op $n2 $n1 orient right-down
$ns duplex-link-op $n3 $n1 orient right-up
$ns duplex-link-op $n0 $n1 orient left-up
$ns duplex-link-op $n4 $n1 orient left-down
proc finish {} {
global ns nf nd
$ns flush-trace
close $nf
exec nam cse.nam &
exit 0
}
$ns at 5.0 "finish"
$ns run
```

OUTPUT:



3 RING TOPOLOGY

In ring topology, each host machine connects to exactly two other machines, creating a circular network structure. When one host tries to communicate or send message to a host which is not adjacent to it, the data travels through all intermediate hosts. To connect one more host in the existing structure, the administrator may need only one more extra cable.

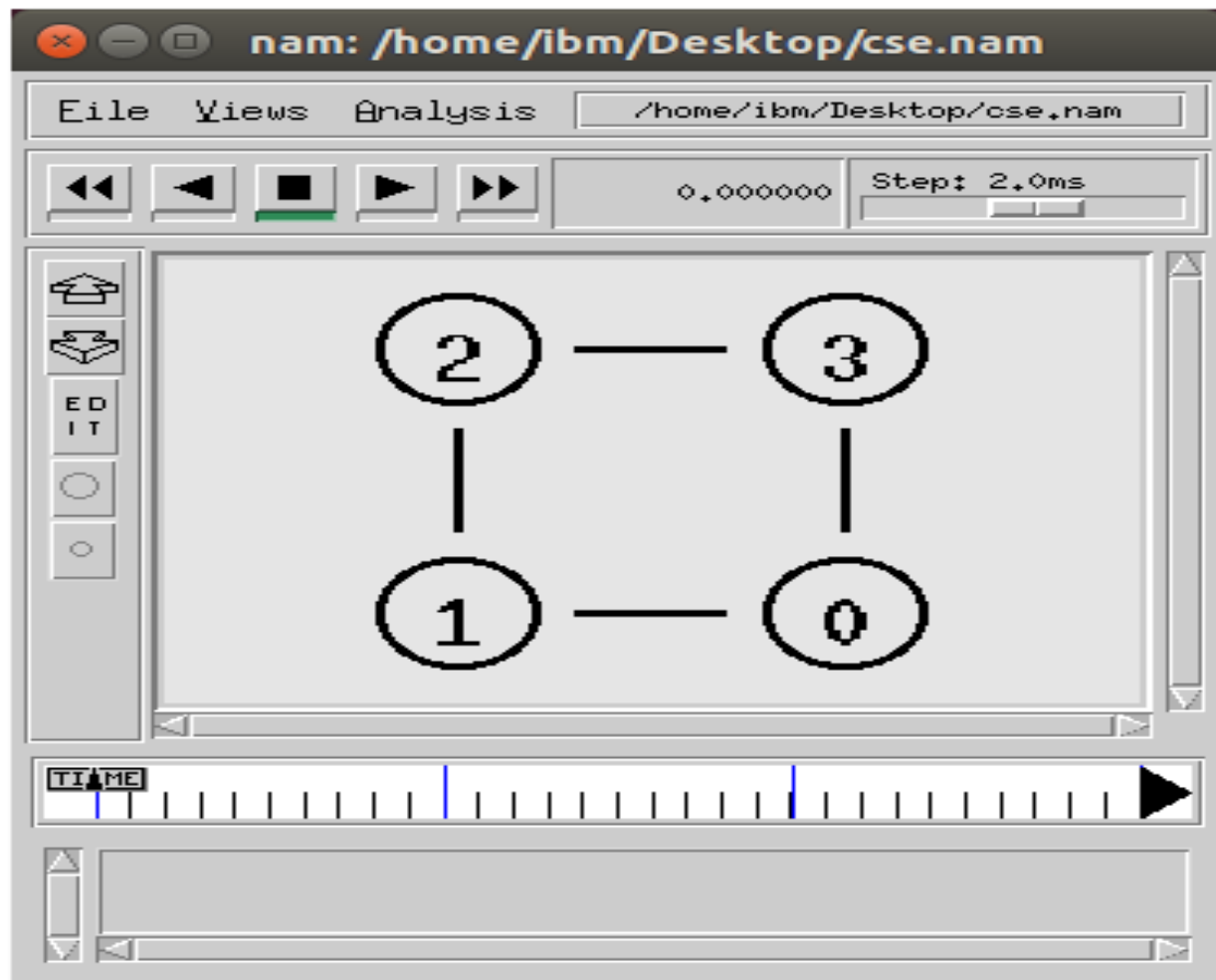


Failure of any host results in failure of the whole ring. Thus, every connection in the ring is a point of failure. There are methods which employ one more backup ring.

SOURCE CODE:

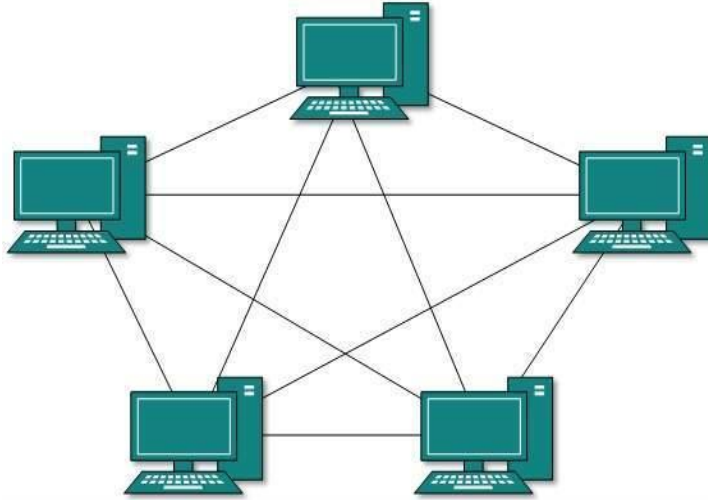
```
set ns [new Simulator]
set nf [open cse.nam w]
$ns namtrace-all $nf
set nd [open cse.tr w]
$ns trace-all $nd
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
$ns duplex-link $n1 $n2 10Mbps 10ms DropTail
$ns duplex-link $n2 $n3 10Mbps 10ms DropTail
$ns duplex-link $n3 $n4 10Mbps 10ms DropTail
$ns duplex-link $n4 $n1 10Mbps 10ms DropTail
$ns duplex-link-op $n2 $n1 orient right
$ns duplex-link-op $n3 $n2 orient down
$ns duplex-link-op $n4 $n3 orient left
$ns duplex-link-op $n1 $n4 orient up
proc finish {} {
global ns nf nd
$ns flush-trace
close $nf
exec nam cse.nam &
exit 0
}
$ns at 5.0 "finish"
$ns run
```

OUTPUT:



4 MESH TOPOLOGY

Mesh network topology provides redundant communication paths between some or all devices in a partial or full mesh. In full mesh topology, every device is connected to all the other devices. In a partial mesh topology, some of the connected devices or systems are connected to all the others, but some of the devices only connect to a few other devices. Mesh is robust and troubleshooting is relatively easy. However, installation and configuration are more complicated than with the star, ring and bus topologies.



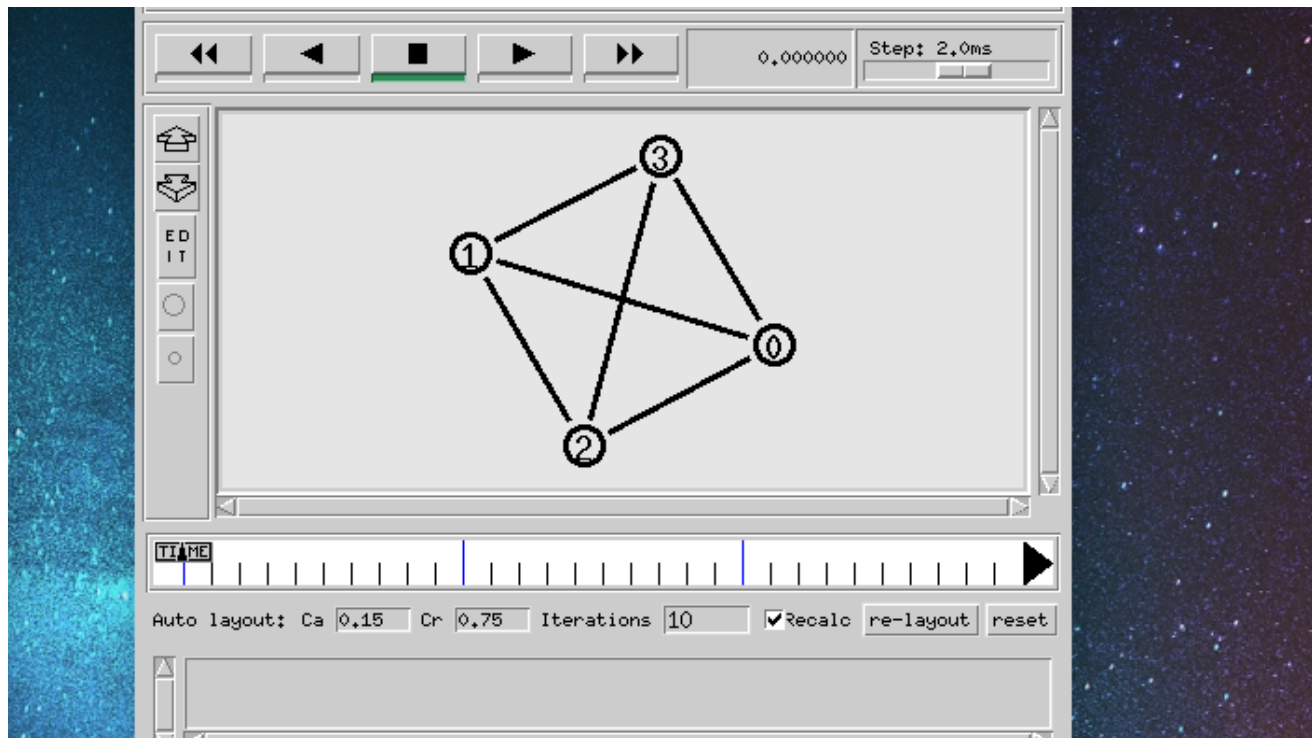
SOURCE CODE:

```
set ns [new Simulator]
set nf [open cse.nam w]
$ns namtrace-all $nf
set nd [open cse.tr w]
$ns trace-all $nd
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
$ns duplex-link $n1 $n2 10Mbps 10ms DropTail
$ns duplex-link $n2 $n3 10Mbps 10ms DropTail
$ns duplex-link $n3 $n4 10Mbps 10ms DropTail
$ns duplex-link $n4 $n1 10Mbps 10ms DropTail
$ns duplex-link $n3 $n1 10Mbps 10ms DropTail
$ns duplex-link $n2 $n4 10Mbps 10ms DropTail
$ns duplex-link-op $n2 $n1 orient right
$ns duplex-link-op $n3 $n2 orient down
$ns duplex-link-op $n4 $n3 orient left
$ns duplex-link-op $n1 $n4 orient up
$ns duplex-link-op $n3 $n1 orient right-down
$ns duplex-link-op $n2 $n4 orient right-up
proc finish {} {
global ns nf nd
$ns flush-trace
close $nf
exec nam cse.nam &
exit 0
}
```

\$ns at 5.0 "finish"

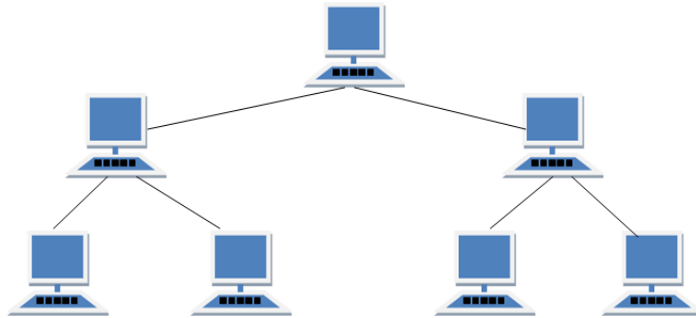
\$ns run

OUTPUT:



5 TREE TOPOLOGY

Also known as Hierarchical Topology, this is the most common form of network topology in use presently. This topology imitates as extended Star topology and inherits properties of bus topology. It has a root node and all other nodes are connected to it forming a hierarchy. It should at least have three levels to the hierarchy.



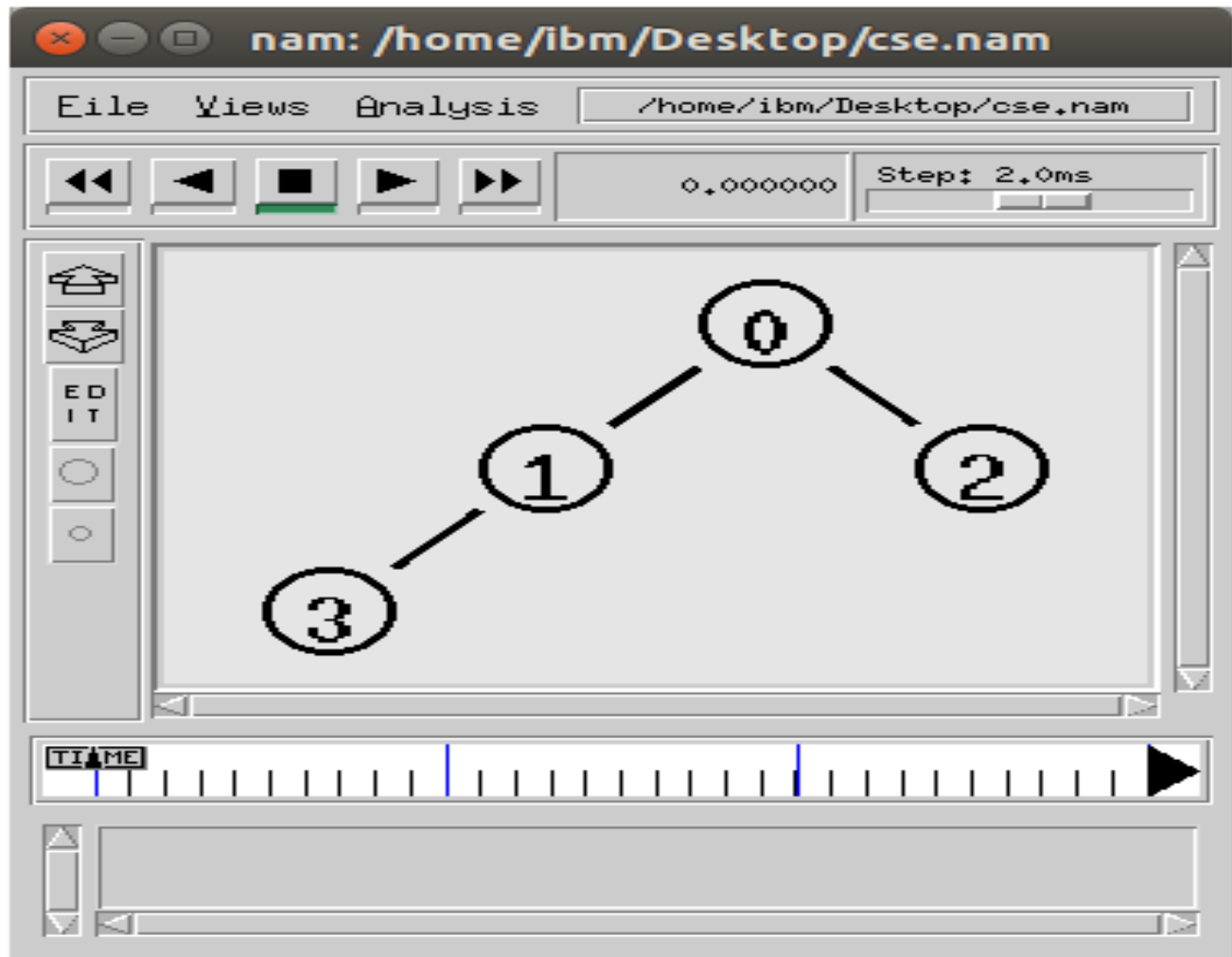
SOURCE CODE:

```
set ns [new Simulator]
set nf [open cse.nam w]
$ns namtrace-all $nf
set nd [open cse.tr w]
$ns trace-all $nd
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
$ns duplex-link $n1 $n2 10Mbps 10ms DropTail
$ns duplex-link $n1 $n3 10Mbps 10ms DropTail
$ns duplex-link $n2 $n4 10Mbps 10ms DropTail
#$ns duplex-link $n4 $n1 10Mbps 10ms DropTail
$ns duplex-link-op $n1 $n2 orient left-down
$ns duplex-link-op $n1 $n3 orient right-down
$ns duplex-link-op $n2 $n4 orient left-down
#$ns duplex-link-op $n1 $n4 orient up

proc finish {} {
global ns nf nd
$ns flush-trace
close $nf
exec nam cse.nam &
exit 0
}
$ns at 5.0 "finish"
```

\$ns run

OUTPUT:



WEEK 9

QUESTION: Implement Transmission Control Protocol (TCP) and User Datagram Protocol(UDP) using NS-2.

AIM: Implement Transmission Control Protocol (TCP) and User Datagram Protocol(UDP) using NS-2.

DESCRIPTION:

TCP is a connection-oriented protocol. This means that before the client and server can start to send data to each other, they first need to handshake and establish a TCP connection. One end of the TCP connection is attached to the client socket and the other end is attached to a server socket. When creating the TCP connection, we associate with it the client socket address (IPaddress and port number) and the server socket address (IPaddress and port number). With the TCP connection established, when one side wants to send data to the other side, it just drops the data into the TCP connection via its socket.

With the server process running, the client process can initiate a TCP connection to the server. This is done in the client program by creating a TCP socket. When the client creates its TCP socket, it specifies the address of the welcoming socket in the server, namely, the IP address of the server host and the port number of the socket. After creating its socket, the client initiates a three-way handshake and establishes a TCP connection with the server.

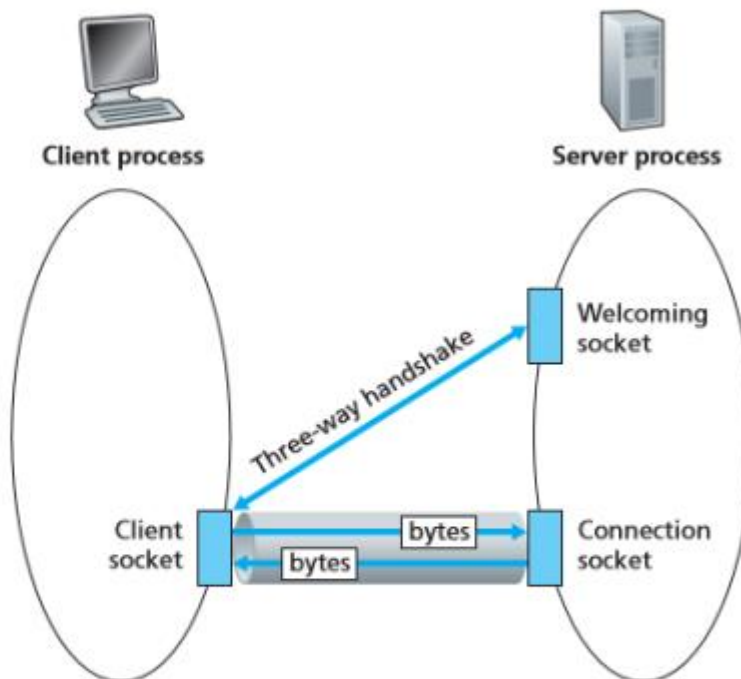


Figure 2.29 ♦ The TCPServer process has two sockets

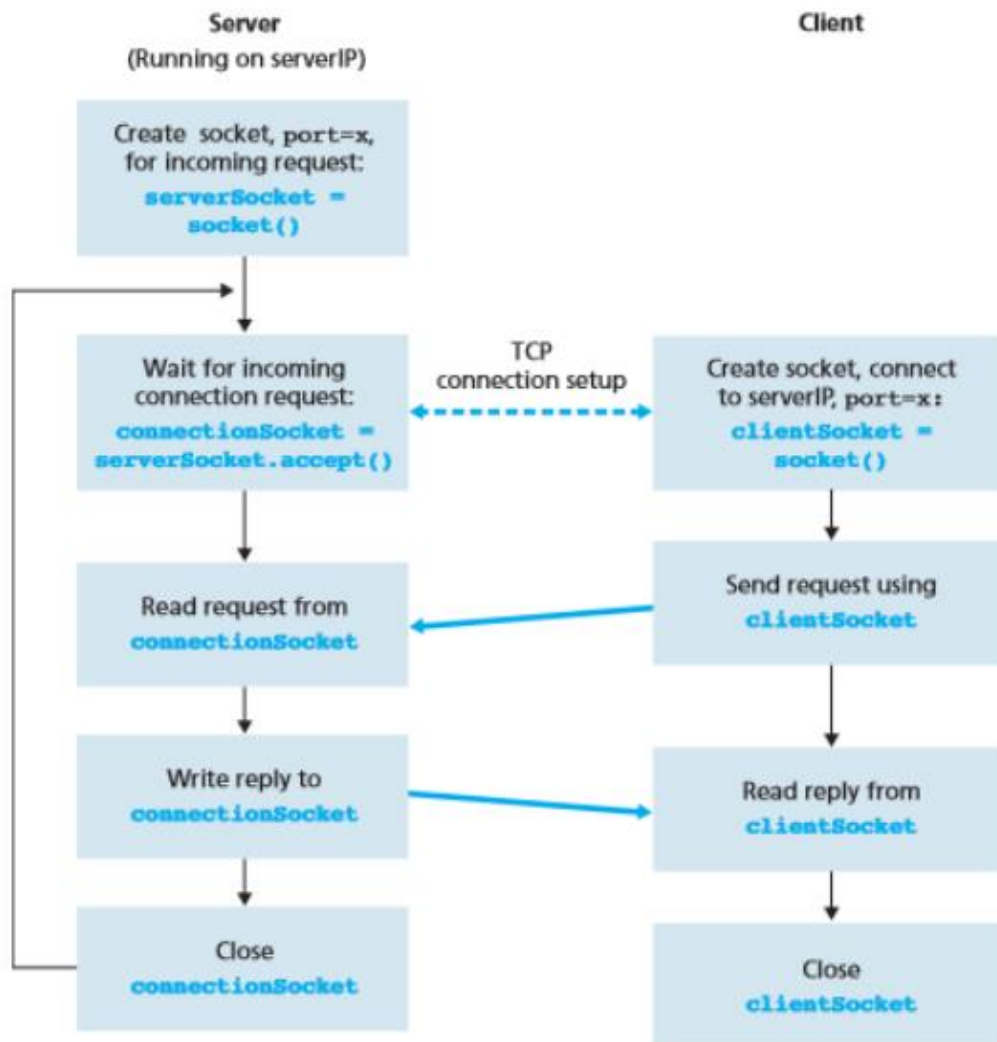


Figure 2.30 ♦ The client-server application using TCP

UDP:-

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

DESCRIPTION:

Using User Datagram Protocol, Applications can send data/message to the other hosts without prior communications or channel or path. This means even if the destination host is not available, application can send data. There is no guarantee that the data is received in the other side. Hence it's not a reliable service.

UDP is appropriate in places where delivery of data doesn't matters during data transition.

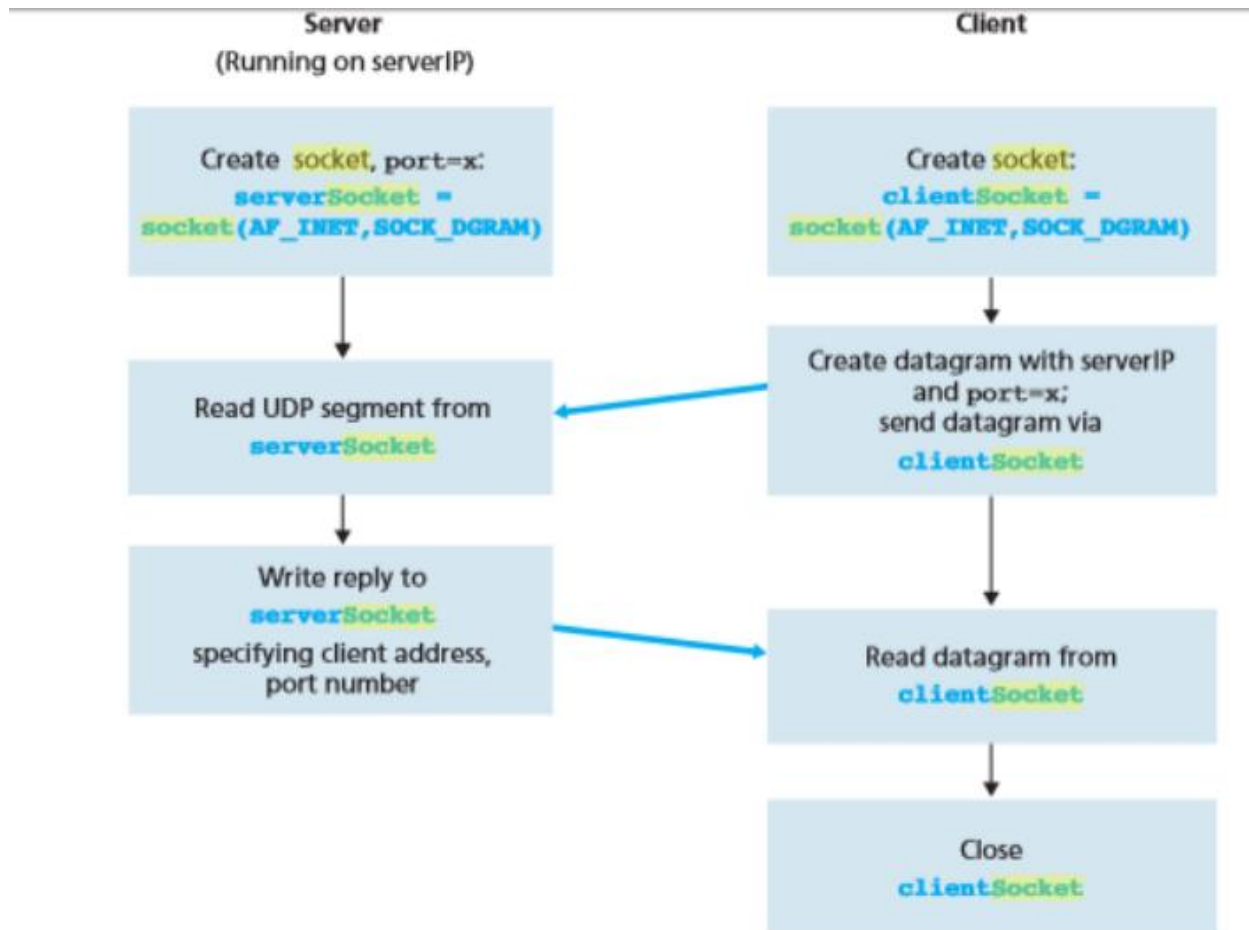


Figure 2.28 ♦ The client-server application using UDP

SOURCE CODE:

TCP for various networks.

1 Simple client-server model using TCP

```
set ns [new Simulator]
set nf [open cs.nam w]
$ns namtrace-all $nf
set nd [open cs.tr w]
$ns trace-all $nd
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
$ns duplex-link $n0 $n1 10Mbps 10ms DropTail
$ns duplex-link-op $n0 $n1 orient right
$ns duplex-link $n1 $n2 10Mbps 10ms DropTail
$ns duplex-link-op $n1 $n2 orient right
$ns duplex-link $n2 $n3 10Mbps 10ms DropTail
$ns duplex-link-op $n2 $n3 orient right
$n0 shape box
$n1 shape hexagon
$ns at 0.1 "$n0 label client"
$ns at 0.2 "$n1 label server"
$ns at 0.1 "$n0 color blue"
$ns at 0.2 "$n1 color red"
set tcp0 [new Agent/TCP]
$ns attach-agent $n0 $tcp0
set Sink [new Agent/Null]
$ns attach-agent $n3 $Sink
$ns connect $tcp0 $Sink
```

```
set cbr0 [new Application/Traffic/CBR]
```

```
$cbr0 set packetSize _ 500
```

```
$cbr0 set interval _ 0.005
```

```
$cbr0 attach-agent $tcp0
```

```
$ns at 0.5 "$cbr0 start"
```

```
$ns at 2.5 "$cbr0 stop"
```

```
proc finish {} {
```

```
global ns nf nd
```

```
$ns flush-trace
```

```
close $nf
```

```
exec nam cs.nam &
```

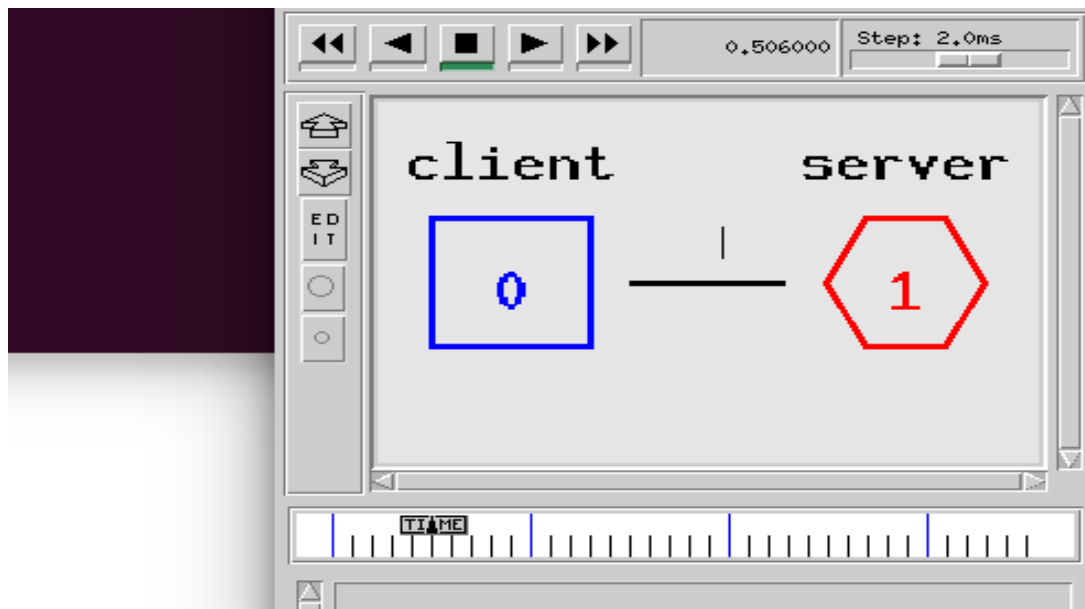
```
exit 0
```

```
}
```

```
$ns at 5.0 "finish"
```

```
$ns run
```


OUTPUT:

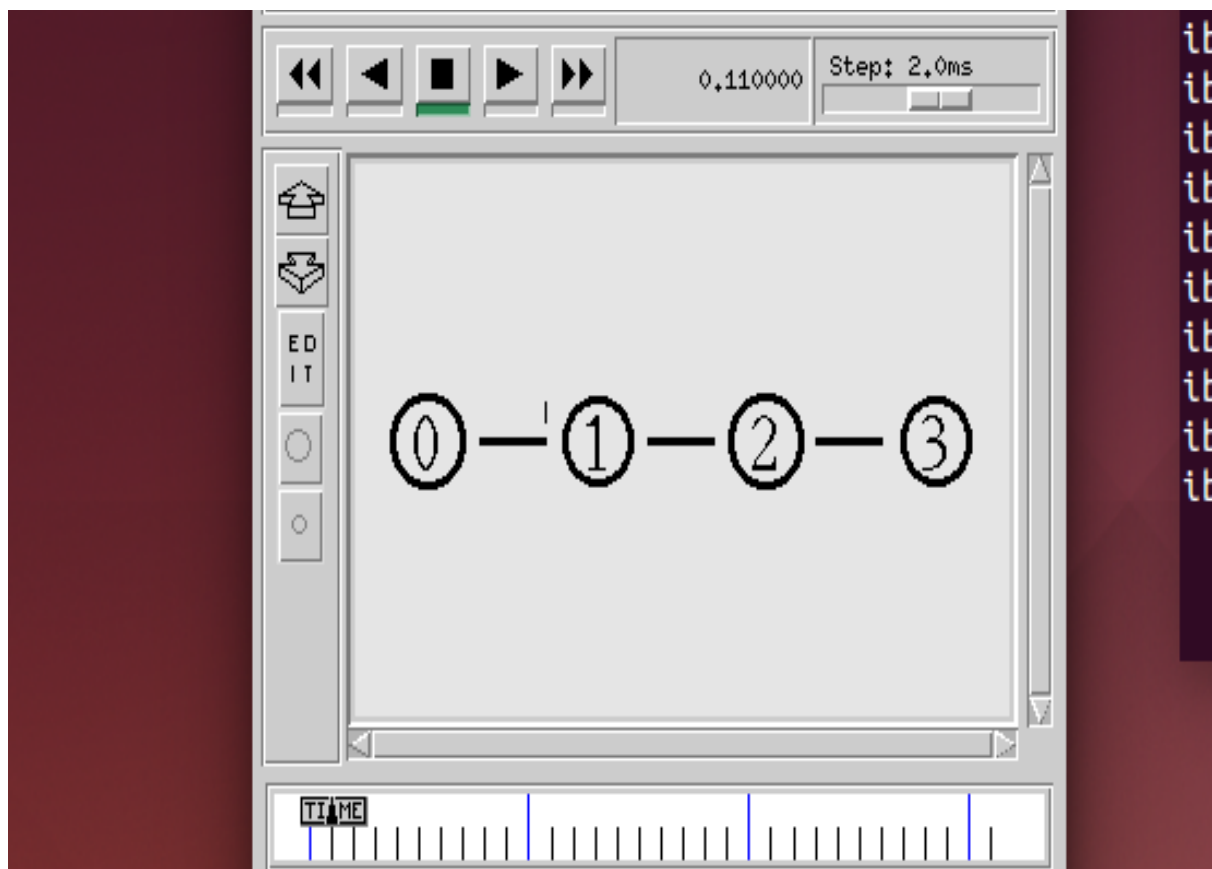


2 Data transfer among set of nodes:

```
set ns [new Simulator]
set nf [open cse.nam w]
$ns namtrace-all $nf
set nd [open cse.tr w]
$ns trace-all $nd
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
$ns duplex-link $n0 $n1 10Mbps 10ms DropTail
$ns duplex-link-op $n0 $n1 orient right
$ns duplex-link $n1 $n2 10Mbps 10ms DropTail
$ns duplex-link-op $n1 $n2 orient right
$ns duplex-link $n2 $n3 10Mbps 10ms DropTail
$ns duplex-link-op $n2 $n3 orient right
set udp0 [new Agent/TCP]
$ns attach-agent $n0 $udp0
set Sink [new Agent/Null]
$ns attach-agent $n3 $Sink
$ns connect $udp0 $Sink
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize _ 1500
$cbr0 set interval _ 0.005
$cbr0 attach-agent $udp0
$ns at 0.1 "$cbr0 start"
$ns at 0.4 "$cbr0 stop"
proc finish {} {
global ns nf nd
```

```
$ns flush-trace
close $nf
exec nam cse.nam &
exit 0
}
$ns at 5.0 "finish"
$ns run
```

OUTPUT:



UDP Programs

```
set ns [new Simulator]

set nf [open cs.nam w]

$ns namtrace-all $nf

set nd [open cs.tr w]

$ns trace-all $nd

set n0 [$ns node]

set n1 [$ns node]

$ns duplex-link $n0 $n1 10Mbps 10ms DropTail

$ns duplex-link-op $n0 $n1 orient right

$n0 shape box

$n1 shape hexagon

$ns at 0.1 "$n0 label client"

$ns at 0.2 "$n1 label server"

$ns at 0.1 "$n0 color blue"

$ns at 0.2 "$n1 color red"

set udp0 [new Agent/UDP]

$ns attach-agent $n0 $udp0

set udp1 [new Agent/Null]

$ns attach-agent $n1 $udp1

$ns connect $udp0 $udp1

set cbr0 [new Application/Traffic/CBR]

$cbr0 set packetSize _ 500

$cbr0 set interval _ 0.005
```

```
$cbr0 attach-agent $udp0
```

```
$ns at 0.5 "$cbr0 start"
```

```
$ns at 2.5 "$cbr0 stop"
```

```
proc finish {} {
```

```
global ns nf nd
```

```
$ns flush-trace
```

```
close $nf
```

```
exec nam cs.nam &
```

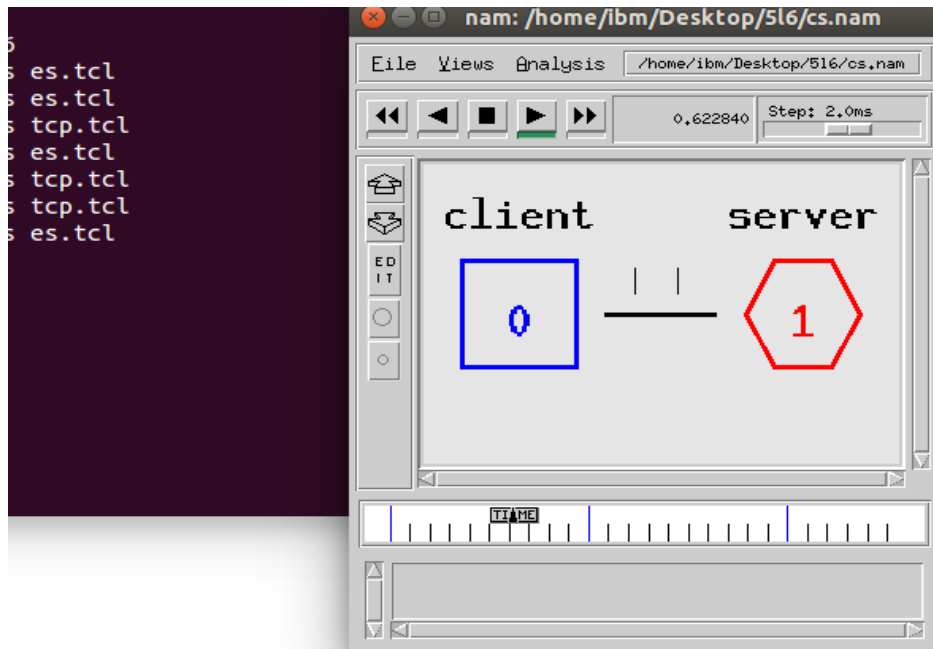
```
exit 0
```

```
}
```

```
$ns at 5.0 "finish"
```

```
$ns run
```

OUTPUT:



```
set ns [new Simulator]

set nf [open cse.nam w]

$ns namtrace-all $nf

set nd [open cse.tr w]

$ns trace-all $nd

set n0 [$ns node]

set n1 [$ns node]

set n2 [$ns node]

set n3 [$ns node]

$ns duplex-link $n0 $n1 10Mbps 10ms DropTail

$ns duplex-link-op $n0 $n1 orient right

$ns duplex-link $n1 $n2 10Mbps 10ms DropTail

$ns duplex-link-op $n1 $n2 orient right

$ns duplex-link $n2 $n3 10Mbps 10ms DropTail

$ns duplex-link-op $n2 $n3 orient right

set udp0 [new Agent/UDP]

$ns attach-agent $n0 $udp0

set Sink [new Agent/Null]

$ns attach-agent $n3 $Sink

$ns connect $udp0 $Sink

set cbr0 [new Application/Traffic/CBR]

$cbr0 set packetSize _ 500

$cbr0 set interval _ 0.005

$cbr0 attach-agent $udp0

$ns at 0.1 "$cbr0 start"
```

```
$ns at 0.4 "$cbr0 stop"
```

```
proc finish {} {
```

```
global ns nf nd
```

```
$ns flush-trace
```

```
close $nf
```

```
exec nam cse.nam &
```

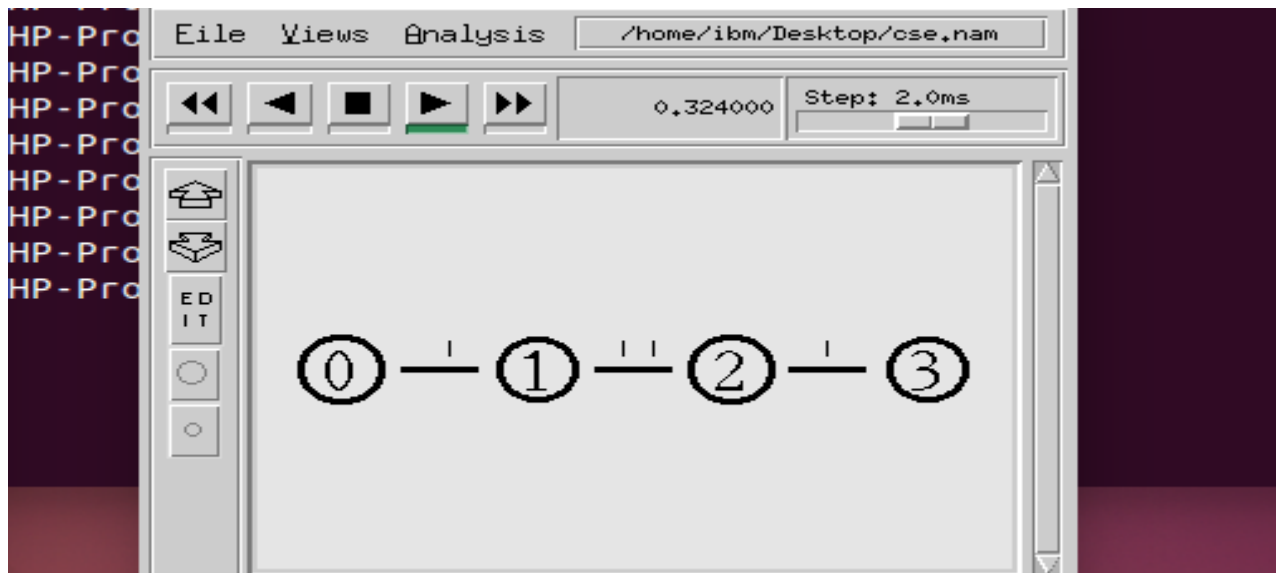
```
exit 0
```

```
}
```

```
$ns at 5.0 "finish"
```

```
$ns run
```

OUTPUT:



WEEK 10

QUESTION: Design Network using loop constraint

AIM: Design Network using loop constraint

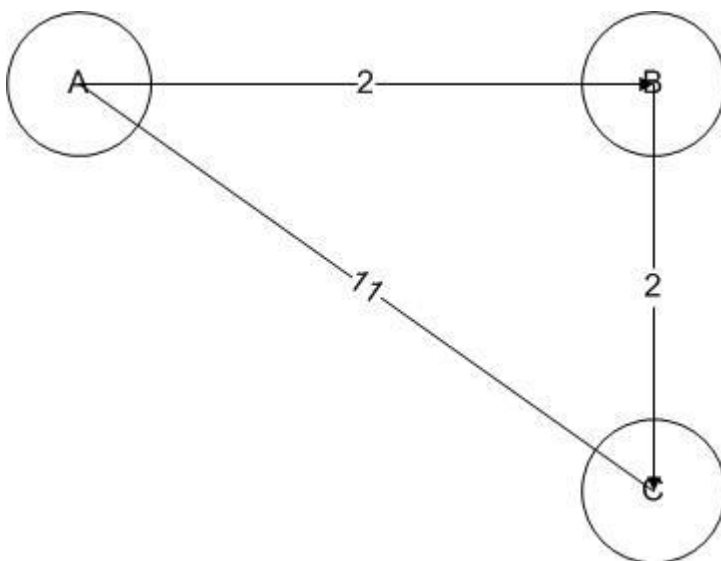
DESCRIPTION:

A routing loop is a common problem with various types of networks, particularly computer networks. They are formed when an error occurs in the operation of the routing algorithm, and as a result, in a group of nodes, the path to a particular destination forms a loop.

In the simplest version, a routing loop of size two, node A thinks that the path to some destination (call it C) is through its neighbouring node, node B. At the same time, node B thinks that the path to C starts at node A.

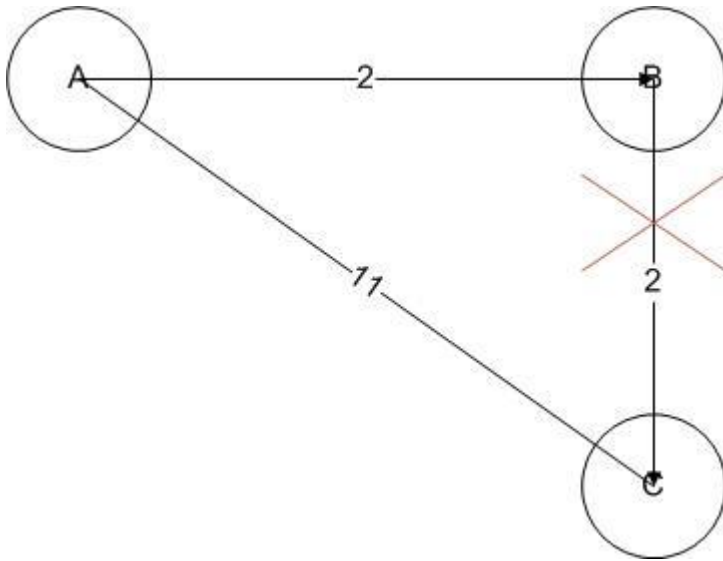
Thus, whenever traffic for C arrives at either A or B, it will loop endlessly between A and B, unless some mechanism exists to prevent that behaviour.

For example, in the network given below, **node A** is transmitting data to node **C** via node **B**. If the link between nodes **B** and **C** goes down and **B** has not yet informed node **A** about the breakage, node **A** transmits the data to node **B** assuming that the link **A-B-C** is operational and of lowest cost. Node **B** knows of the broken link and tries to reach node **C** via node **A**, thus sending the original data back to node **A**. Furthermore, node **A** receives the data that it originated back from node **B** and consults its routing table. Node **A**'s routing table will say that it can reach node **C** via node **B** (because it still has not been informed of the break) thus sending its data back to node **B** creating an infinite loop.



Consider now what happens if both the link from A to C and the link from B to C vanish at the same time (this can happen if node C has crashed). A believes that C is still reachable through B, and B believes that C is reachable through A. In a simple reachability protocol, such as EGP, the routing loop will persist forever.

In a naive distance vector protocol, such as the routing information protocol, the loop will persist until the metrics for C reach *infinity* (the maximum number of routers that a packet can traverse in RIP is 15. The value 16 is considered infinity and the packet is discarded).



In a link-state routing protocol, such as OSPF or IS-IS, a routing loop disappears as soon as the new network topology is flooded to all the routers within the routing area. Assuming a sufficiently reliable network, this happens within a few seconds.

Newer distance-vector routing protocols (BGP, EIGRP, DSDV, Babel) have built-in loop prevention: they use algorithms that assure that routing loops can never happen, not even transiently. Older routing protocols (RIP) do not implement the newest forms of loop prevention and only implement mitigations such as split horizon, route poisoning, holddown timer, Trigger Update and Define maximum hops.

WEEK 11

QUESTION: Design a Network using Packet Tracer

AIM: Design a Network using Packet Tracer

DESCRIPTION:

To perform the CCNA hands-on lab exercises, first, you need to [download](#) Cisco Packet Tracer. In the previous post, we explained the process how to download Cisco Packet Tracer. [Click here to know how to download Cisco Packet Tracer](#). Once it is downloaded, install it using the default selections. After the installation, you can create a network topology to perform the hands-on lab exercises. In this post, we will discuss how to create a network topology using Cisco Packet Tracer.

Create a Network Topology

You can easily create a network topology using Cisco Packet Tracer. In the following sections, we are going to explain how to create a network topology that will contain four PCs, two switches, and two routers.

Adding PCs in Cisco Packet Tracer

To add PCs in Cisco Packet Tracer, you need to perform the following steps:

1. In the Cisco Packet Tracer console, click on the PC icon, click Generic, and then click in the logical view area to add a Generic PC.
2. Repeat the same step to add three more Generic PCs in the logical view area, as shown in the following figure.

SOURCE CODE:

Trace to tcl code:

```
import sys
input_file=input("Enter Trace file : ")
fp=open(input_file,"r")
name=input_file.split(".")
name=name[0]
nodes=set()
pairs=[]
for line in fp:
    l1=line.split()
    a=list(l1[2])
    a.append(l1[3])
    nodes.add(l1[2])
    nodes.add(l1[3])
    pairs.append(a)
pairs = list(set(map(tuple,pairs)))
pairs=filter(None, pairs)
print("Total Nodes in Trace File are :", end="")
print(nodes)
print("Total connection pairs are :")
out_file=name+".tcl"
ofp=open(out_file,"w")
ofp.write("set ns [new Simulator]\n")
ofp.write("set nf [open "+name+".nam w]\n")
ofp.write("$ns namtrace-all $nf\n")
ofp.write("set nd [open "+name+".tr r]\n")
for i in nodes:
    ofp.write("set n")
```

```

    ofp.write(i)
    ofp.write(" [$ns node]\n")
for i in pairs:
    print(i)
    ofp.write("$ns duplex-link $n")
    ofp.write(i[0])
    #print(i)
    ofp.write(" $n")
    ofp.write(i[1])
    ofp.write(" 10Mbps 10ms DropTail\n")
ofp.write("proc finish {} {\n")
ofp.write("  global ns nf nd\n")
ofp.write("  $ns flush-trace\n")
ofp.write("  close $nf\n")
ofp.write("  exec nam "+name+".nam &\n")
ofp.write("  exit 0\n")
ofp.write("}\n")
ofp.write("")
ofp.write("$ns at 5.0 \"finish\" \n")
ofp.write("$ns run\n")

```

Trace file:

```
+ 1 0 1 tcp 40 ----- 0 0.0 2.0 0 0
- 1 0 1 tcp 40 ----- 0 0.0 2.0 0 0
r 1.05016 0 1 tcp 40 ----- 0 0.0 2.0 0 0
+ 1.05016 1 2 tcp 40 ----- 0 0.0 2.0 0 0
- 1.05016 1 2 tcp 40 ----- 0 0.0 2.0 0 0
r 1.10032 1 2 tcp 40 ----- 0 0.0 2.0 0 0
+ 1.10032 2 1 ack 40 ----- 0 2.0 0.0 0 1
- 1.10032 2 1 ack 40 ----- 0 2.0 0.0 0 1
r 1.15048 2 1 ack 40 ----- 0 2.0 0.0 0 1
+ 1.15048 1 0 ack 40 ----- 0 2.0 0.0 0 1
- 1.15048 1 0 ack 40 ----- 0 2.0 0.0 0 1
r 1.20064 1 0 ack 40 ----- 0 2.0 0.0 0 1
+ 1.20064 0 1 tcp 1040 ----- 0 0.0 2.0 1 2
- 1.20064 0 1 tcp 1040 ----- 0 0.0 2.0 1 2
+ 1.20064 0 1 tcp 1040 ----- 0 0.0 2.0 2 3
- 1.2048 0 1 tcp 1040 ----- 0 0.0 2.0 2 3
r 1.2548 0 1 tcp 1040 ----- 0 0.0 2.0 1 2
+ 1.2548 1 2 tcp 1040 ----- 0 0.0 2.0 1 2
- 1.2548 1 2 tcp 1040 ----- 0 0.0 2.0 1 2
r 1.25896 0 1 tcp 1040 ----- 0 0.0 2.0 2 3
+ 1.25896 1 2 tcp 1040 ----- 0 0.0 2.0 2 3
- 1.25896 1 2 tcp 1040 ----- 0 0.0 2.0 2 3
r 1.30896 1 2 tcp 1040 ----- 0 0.0 2.0 1 2
+ 1.30896 2 1 ack 40 ----- 0 2.0 0.0 1 4
- 1.30896 2 1 ack 40 ----- 0 2.0 0.0 1 4
r 1.31312 1 2 tcp 1040 ----- 0 0.0 2.0 2 3
+ 1.31312 2 1 ack 40 ----- 0 2.0 0.0 2 5
- 1.31312 2 1 ack 40 ----- 0 2.0 0.0 2 5
r 1.35912 2 1 ack 40 ----- 0 2.0 0.0 1 4
+ 1.35912 1 0 ack 40 ----- 0 2.0 0.0 1 4
- 1.35912 1 0 ack 40 ----- 0 2.0 0.0 1 4
r 1.36328 2 1 ack 40 ----- 0 2.0 0.0 2 5
```

```

+ 1.36328 1 0 ack 40 ----- 0 2.0 0.0 2 5
- 1.36328 1 0 ack 40 ----- 0 2.0 0.0 2 5
r 1.40928 1 0 ack 40 ----- 0 2.0 0.0 1 4
+ 1.40928 0 1 tcp 1040 ----- 0 0.0 2.0 3 6
- 1.40928 0 1 tcp 1040 ----- 0 0.0 2.0 3 6
+ 1.40928 0 1 tcp 1040 ----- 0 0.0 2.0 4 7
r 1.41344 1 0 ack 40 ----- 0 2.0 0.0 2 5
+ 1.41344 0 1 tcp 1040 ----- 0 0.0 2.0 5 8
+ 1.41344 0 1 tcp 1040 ----- 0 0.0 2.0 6 9
- 1.41344 0 1 tcp 1040 ----- 0 0.0 2.0 4 7
- 1.4176 0 1 tcp 1040 ----- 0 0.0 2.0 5 8
- 1.42176 0 1 tcp 1040 ----- 0 0.0 2.0 6 9
r 1.46344 0 1 tcp 1040 ----- 0 0.0 2.0 3 6
+ 1.46344 1 2 tcp 1040 ----- 0 0.0 2.0 3 6
- 1.46344 1 2 tcp 1040 ----- 0 0.0 2.0 3 6
r 1.4676 0 1 tcp 1040 ----- 0 0.0 2.0 4 7
+ 1.4676 1 2 tcp 1040 ----- 0 0.0 2.0 4 7
- 1.4676 1 2 tcp 1040 ----- 0 0.0 2.0 4 7
r 1.47176 0 1 tcp 1040 ----- 0 0.0 2.0 5 8
+ 1.47176 1 2 tcp 1040 ----- 0 0.0 2.0 5 8
- 1.47176 1 2 tcp 1040 ----- 0 0.0 2.0 5 8
r 1.47592 0 1 tcp 1040 ----- 0 0.0 2.0 6 9
+ 1.47592 1 2 tcp 1040 ----- 0 0.0 2.0 6 9
- 1.47592 1 2 tcp 1040 ----- 0 0.0 2.0 6 9
r 1.5176 1 2 tcp 1040 ----- 0 0.0 2.0 3 6
+ 1.5176 2 1 ack 40 ----- 0 2.0 0.0 3 10
- 1.5176 2 1 ack 40 ----- 0 2.0 0.0 3 10
r 1.52176 1 2 tcp 1040 ----- 0 0.0 2.0 4 7
+ 1.52176 2 1 ack 40 ----- 0 2.0 0.0 4 11
- 1.52176 2 1 ack 40 ----- 0 2.0 0.0 4 11
r 1.52592 1 2 tcp 1040 ----- 0 0.0 2.0 5 8
+ 1.52592 2 1 ack 40 ----- 0 2.0 0.0 5 12
- 1.52592 2 1 ack 40 ----- 0 2.0 0.0 5 12

```

```

r 1.53008 1 2 tcp 1040 ----- 0 0.0 2.0 6 9
+ 1.53008 2 1 ack 40 ----- 0 2.0 0.0 6 13
- 1.53008 2 1 ack 40 ----- 0 2.0 0.0 6 13
r 1.56776 2 1 ack 40 ----- 0 2.0 0.0 3 10
+ 1.56776 1 0 ack 40 ----- 0 2.0 0.0 3 10
- 1.56776 1 0 ack 40 ----- 0 2.0 0.0 3 10
r 1.57192 2 1 ack 40 ----- 0 2.0 0.0 4 11
+ 1.57192 1 0 ack 40 ----- 0 2.0 0.0 4 11
- 1.57192 1 0 ack 40 ----- 0 2.0 0.0 4 11
r 1.57608 2 1 ack 40 ----- 0 2.0 0.0 5 12
+ 1.57608 1 0 ack 40 ----- 0 2.0 0.0 5 12
- 1.57608 1 0 ack 40 ----- 0 2.0 0.0 5 12
r 1.58024 2 1 ack 40 ----- 0 2.0 0.0 6 13
+ 1.58024 1 0 ack 40 ----- 0 2.0 0.0 6 13
- 1.58024 1 0 ack 40 ----- 0 2.0 0.0 6 13
r 1.61792 1 0 ack 40 ----- 0 2.0 0.0 3 10
+ 1.61792 0 1 tcp 1040 ----- 0 0.0 2.0 7 14
- 1.61792 0 1 tcp 1040 ----- 0 0.0 2.0 7 14
+ 1.61792 0 1 tcp 1040 ----- 0 0.0 2.0 8 15
r 1.62208 1 0 ack 40 ----- 0 2.0 0.0 4 11
+ 1.62208 0 1 tcp 1040 ----- 0 0.0 2.0 9 16
+ 1.62208 0 1 tcp 1040 ----- 0 0.0 2.0 10 17
- 1.62208 0 1 tcp 1040 ----- 0 0.0 2.0 8 15
r 1.62624 1 0 ack 40 ----- 0 2.0 0.0 5 12
+ 1.62624 0 1 tcp 1040 ----- 0 0.0 2.0 11 18
+ 1.62624 0 1 tcp 1040 ----- 0 0.0 2.0 12 19
- 1.62624 0 1 tcp 1040 ----- 0 0.0 2.0 9 16
r 1.6304 1 0 ack 40 ----- 0 2.0 0.0 6 13
+ 1.6304 0 1 tcp 1040 ----- 0 0.0 2.0 13 20
+ 1.6304 0 1 tcp 1040 ----- 0 0.0 2.0 14 21
- 1.6304 0 1 tcp 1040 ----- 0 0.0 2.0 10 17
- 1.63456 0 1 tcp 1040 ----- 0 0.0 2.0 11 18
- 1.63872 0 1 tcp 1040 ----- 0 0.0 2.0 12 19

```

```

- 1.64288 0 1 tcp 1040 ----- 0 0.0 2.0 13 20
- 1.64704 0 1 tcp 1040 ----- 0 0.0 2.0 14 21
r 1.67208 0 1 tcp 1040 ----- 0 0.0 2.0 7 14
+ 1.67208 1 2 tcp 1040 ----- 0 0.0 2.0 7 14
- 1.67208 1 2 tcp 1040 ----- 0 0.0 2.0 7 14
r 1.67624 0 1 tcp 1040 ----- 0 0.0 2.0 8 15
+ 1.67624 1 2 tcp 1040 ----- 0 0.0 2.0 8 15
- 1.67624 1 2 tcp 1040 ----- 0 0.0 2.0 8 15
r 1.6804 0 1 tcp 1040 ----- 0 0.0 2.0 9 16
+ 1.6804 1 2 tcp 1040 ----- 0 0.0 2.0 9 16
- 1.6804 1 2 tcp 1040 ----- 0 0.0 2.0 9 16
r 1.68456 0 1 tcp 1040 ----- 0 0.0 2.0 10 17
+ 1.68456 1 2 tcp 1040 ----- 0 0.0 2.0 10 17
- 1.68456 1 2 tcp 1040 ----- 0 0.0 2.0 10 17
r 1.68872 0 1 tcp 1040 ----- 0 0.0 2.0 11 18
+ 1.68872 1 2 tcp 1040 ----- 0 0.0 2.0 11 18
- 1.68872 1 2 tcp 1040 ----- 0 0.0 2.0 11 18
r 1.69288 0 1 tcp 1040 ----- 0 0.0 2.0 12 19
+ 1.69288 1 2 tcp 1040 ----- 0 0.0 2.0 12 19
- 1.69288 1 2 tcp 1040 ----- 0 0.0 2.0 12 19
r 1.69704 0 1 tcp 1040 ----- 0 0.0 2.0 13 20
+ 1.69704 1 2 tcp 1040 ----- 0 0.0 2.0 13 20
- 1.69704 1 2 tcp 1040 ----- 0 0.0 2.0 13 20
r 1.7012 0 1 tcp 1040 ----- 0 0.0 2.0 14 21
+ 1.7012 1 2 tcp 1040 ----- 0 0.0 2.0 14 21
- 1.7012 1 2 tcp 1040 ----- 0 0.0 2.0 14 21
r 1.72624 1 2 tcp 1040 ----- 0 0.0 2.0 7 14
+ 1.72624 2 1 ack 40 ----- 0 2.0 0.0 7 22
- 1.72624 2 1 ack 40 ----- 0 2.0 0.0 7 22
r 1.7304 1 2 tcp 1040 ----- 0 0.0 2.0 8 15
+ 1.7304 2 1 ack 40 ----- 0 2.0 0.0 8 23
- 1.7304 2 1 ack 40 ----- 0 2.0 0.0 8 23
r 1.73456 1 2 tcp 1040 ----- 0 0.0 2.0 9 16

```



```

+ 1.73456 2 1 ack 40 ----- 0 2.0 0.0 9 24
- 1.73456 2 1 ack 40 ----- 0 2.0 0.0 9 24
r 1.73872 1 2 tcp 1040 ----- 0 0.0 2.0 10 17
+ 1.73872 2 1 ack 40 ----- 0 2.0 0.0 10 25
- 1.73872 2 1 ack 40 ----- 0 2.0 0.0 10 25
r 1.74288 1 2 tcp 1040 ----- 0 0.0 2.0 11 18
+ 1.74288 2 1 ack 40 ----- 0 2.0 0.0 11 26
- 1.74288 2 1 ack 40 ----- 0 2.0 0.0 11 26
r 1.74704 1 2 tcp 1040 ----- 0 0.0 2.0 12 19
+ 1.74704 2 1 ack 40 ----- 0 2.0 0.0 12 27
- 1.74704 2 1 ack 40 ----- 0 2.0 0.0 12 27
r 1.7512 1 2 tcp 1040 ----- 0 0.0 2.0 13 20
+ 1.7512 2 1 ack 40 ----- 0 2.0 0.0 13 28
- 1.7512 2 1 ack 40 ----- 0 2.0 0.0 13 28
r 1.75536 1 2 tcp 1040 ----- 0 0.0 2.0 14 21
+ 1.75536 2 1 ack 40 ----- 0 2.0 0.0 14 29
- 1.75536 2 1 ack 40 ----- 0 2.0 0.0 14 29
r 1.7764 2 1 ack 40 ----- 0 2.0 0.0 7 22
+ 1.7764 1 0 ack 40 ----- 0 2.0 0.0 7 22
- 1.7764 1 0 ack 40 ----- 0 2.0 0.0 7 22
r 1.78056 2 1 ack 40 ----- 0 2.0 0.0 8 23
+ 1.78056 1 0 ack 40 ----- 0 2.0 0.0 8 23
- 1.78056 1 0 ack 40 ----- 0 2.0 0.0 8 23
r 1.78472 2 1 ack 40 ----- 0 2.0 0.0 9 24
+ 1.78472 1 0 ack 40 ----- 0 2.0 0.0 9 24
- 1.78472 1 0 ack 40 ----- 0 2.0 0.0 9 24
r 1.78888 2 1 ack 40 ----- 0 2.0 0.0 10 25
+ 1.78888 1 0 ack 40 ----- 0 2.0 0.0 10 25
- 1.78888 1 0 ack 40 ----- 0 2.0 0.0 10 25
r 1.79304 2 1 ack 40 ----- 0 2.0 0.0 11 26
+ 1.79304 1 0 ack 40 ----- 0 2.0 0.0 11 26
- 1.79304 1 0 ack 40 ----- 0 2.0 0.0 11 26
r 1.7972 2 1 ack 40 ----- 0 2.0 0.0 12 27

```

```

+ 1.7972 1 0 ack 40 ----- 0 2.0 0.0 12 27
- 1.7972 1 0 ack 40 ----- 0 2.0 0.0 12 27
r 1.80136 2 1 ack 40 ----- 0 2.0 0.0 13 28
+ 1.80136 1 0 ack 40 ----- 0 2.0 0.0 13 28
- 1.80136 1 0 ack 40 ----- 0 2.0 0.0 13 28
r 1.80552 2 1 ack 40 ----- 0 2.0 0.0 14 29
+ 1.80552 1 0 ack 40 ----- 0 2.0 0.0 14 29
- 1.80552 1 0 ack 40 ----- 0 2.0 0.0 14 29
r 1.82656 1 0 ack 40 ----- 0 2.0 0.0 7 22
+ 1.82656 0 1 tcp 1040 ----- 0 0.0 2.0 15 30
- 1.82656 0 1 tcp 1040 ----- 0 0.0 2.0 15 30
+ 1.82656 0 1 tcp 1040 ----- 0 0.0 2.0 16 31
r 1.83072 1 0 ack 40 ----- 0 2.0 0.0 8 23
+ 1.83072 0 1 tcp 1040 ----- 0 0.0 2.0 17 32
+ 1.83072 0 1 tcp 1040 ----- 0 0.0 2.0 18 33
- 1.83072 0 1 tcp 1040 ----- 0 0.0 2.0 16 31
r 1.83488 1 0 ack 40 ----- 0 2.0 0.0 9 24
+ 1.83488 0 1 tcp 1040 ----- 0 0.0 2.0 19 34
+ 1.83488 0 1 tcp 1040 ----- 0 0.0 2.0 20 35
- 1.83488 0 1 tcp 1040 ----- 0 0.0 2.0 17 32
r 1.83904 1 0 ack 40 ----- 0 2.0 0.0 10 25
+ 1.83904 0 1 tcp 1040 ----- 0 0.0 2.0 21 36
+ 1.83904 0 1 tcp 1040 ----- 0 0.0 2.0 22 37
- 1.83904 0 1 tcp 1040 ----- 0 0.0 2.0 18 33
r 1.8432 1 0 ack 40 ----- 0 2.0 0.0 11 26
+ 1.8432 0 1 tcp 1040 ----- 0 0.0 2.0 23 38
+ 1.8432 0 1 tcp 1040 ----- 0 0.0 2.0 24 39
- 1.8432 0 1 tcp 1040 ----- 0 0.0 2.0 19 34
r 1.84736 1 0 ack 40 ----- 0 2.0 0.0 12 27
+ 1.84736 0 1 tcp 1040 ----- 0 0.0 2.0 25 40
+ 1.84736 0 1 tcp 1040 ----- 0 0.0 2.0 26 41
- 1.84736 0 1 tcp 1040 ----- 0 0.0 2.0 20 35
r 1.85152 1 0 ack 40 ----- 0 2.0 0.0 13 28

```

```

+ 1.85152 0 1 tcp 1040 ----- 0 0.0 2.0 27 42
+ 1.85152 0 1 tcp 1040 ----- 0 0.0 2.0 28 43
- 1.85152 0 1 tcp 1040 ----- 0 0.0 2.0 21 36
r 1.85568 1 0 ack 40 ----- 0 2.0 0.0 14 29
+ 1.85568 0 1 tcp 1040 ----- 0 0.0 2.0 29 44
+ 1.85568 0 1 tcp 1040 ----- 0 0.0 2.0 30 45
- 1.85568 0 1 tcp 1040 ----- 0 0.0 2.0 22 37
- 1.85984 0 1 tcp 1040 ----- 0 0.0 2.0 23 38
- 1.864 0 1 tcp 1040 ----- 0 0.0 2.0 24 39
- 1.86816 0 1 tcp 1040 ----- 0 0.0 2.0 25 40
- 1.87232 0 1 tcp 1040 ----- 0 0.0 2.0 26 41
- 1.87648 0 1 tcp 1040 ----- 0 0.0 2.0 27 42
- 1.88064 0 1 tcp 1040 ----- 0 0.0 2.0 28 43
r 1.88072 0 1 tcp 1040 ----- 0 0.0 2.0 15 30
+ 1.88072 1 2 tcp 1040 ----- 0 0.0 2.0 15 30
- 1.88072 1 2 tcp 1040 ----- 0 0.0 2.0 15 30
- 1.8848 0 1 tcp 1040 ----- 0 0.0 2.0 29 44
r 1.88488 0 1 tcp 1040 ----- 0 0.0 2.0 16 31
+ 1.88488 1 2 tcp 1040 ----- 0 0.0 2.0 16 31
- 1.88488 1 2 tcp 1040 ----- 0 0.0 2.0 16 31
- 1.88896 0 1 tcp 1040 ----- 0 0.0 2.0 30 45
r 1.88904 0 1 tcp 1040 ----- 0 0.0 2.0 17 32
+ 1.88904 1 2 tcp 1040 ----- 0 0.0 2.0 17 32
- 1.88904 1 2 tcp 1040 ----- 0 0.0 2.0 17 32
r 1.8932 0 1 tcp 1040 ----- 0 0.0 2.0 18 33
+ 1.8932 1 2 tcp 1040 ----- 0 0.0 2.0 18 33
- 1.8932 1 2 tcp 1040 ----- 0 0.0 2.0 18 33
r 1.89736 0 1 tcp 1040 ----- 0 0.0 2.0 19 34
+ 1.89736 1 2 tcp 1040 ----- 0 0.0 2.0 19 34
- 1.89736 1 2 tcp 1040 ----- 0 0.0 2.0 19 34
r 1.90152 0 1 tcp 1040 ----- 0 0.0 2.0 20 35
+ 1.90152 1 2 tcp 1040 ----- 0 0.0 2.0 20 35
- 1.90152 1 2 tcp 1040 ----- 0 0.0 2.0 20 35

```

```

r 1.90568 0 1 tcp 1040 ----- 0 0.0 2.0 21 36
+ 1.90568 1 2 tcp 1040 ----- 0 0.0 2.0 21 36
- 1.90568 1 2 tcp 1040 ----- 0 0.0 2.0 21 36
r 1.90984 0 1 tcp 1040 ----- 0 0.0 2.0 22 37
+ 1.90984 1 2 tcp 1040 ----- 0 0.0 2.0 22 37
- 1.90984 1 2 tcp 1040 ----- 0 0.0 2.0 22 37
r 1.914 0 1 tcp 1040 ----- 0 0.0 2.0 23 38
+ 1.914 1 2 tcp 1040 ----- 0 0.0 2.0 23 38
- 1.914 1 2 tcp 1040 ----- 0 0.0 2.0 23 38
r 1.91816 0 1 tcp 1040 ----- 0 0.0 2.0 24 39
+ 1.91816 1 2 tcp 1040 ----- 0 0.0 2.0 24 39
- 1.91816 1 2 tcp 1040 ----- 0 0.0 2.0 24 39
r 1.92232 0 1 tcp 1040 ----- 0 0.0 2.0 25 40
+ 1.92232 1 2 tcp 1040 ----- 0 0.0 2.0 25 40
- 1.92232 1 2 tcp 1040 ----- 0 0.0 2.0 25 40
r 1.92648 0 1 tcp 1040 ----- 0 0.0 2.0 26 41
+ 1.92648 1 2 tcp 1040 ----- 0 0.0 2.0 26 41
- 1.92648 1 2 tcp 1040 ----- 0 0.0 2.0 26 41
r 1.93064 0 1 tcp 1040 ----- 0 0.0 2.0 27 42
+ 1.93064 1 2 tcp 1040 ----- 0 0.0 2.0 27 42
- 1.93064 1 2 tcp 1040 ----- 0 0.0 2.0 27 42
r 1.9348 0 1 tcp 1040 ----- 0 0.0 2.0 28 43
+ 1.9348 1 2 tcp 1040 ----- 0 0.0 2.0 28 43
- 1.9348 1 2 tcp 1040 ----- 0 0.0 2.0 28 43
r 1.93488 1 2 tcp 1040 ----- 0 0.0 2.0 15 30
+ 1.93488 2 1 ack 40 ----- 0 2.0 0.0 15 46
- 1.93488 2 1 ack 40 ----- 0 2.0 0.0 15 46
r 1.93896 0 1 tcp 1040 ----- 0 0.0 2.0 29 44
+ 1.93896 1 2 tcp 1040 ----- 0 0.0 2.0 29 44
- 1.93896 1 2 tcp 1040 ----- 0 0.0 2.0 29 44
r 1.93904 1 2 tcp 1040 ----- 0 0.0 2.0 16 31
+ 1.93904 2 1 ack 40 ----- 0 2.0 0.0 16 47
- 1.93904 2 1 ack 40 ----- 0 2.0 0.0 16 47

```

r 1.94312 0 1 tcp 1040 ----- 0 0.0 2.0 30 45
+ 1.94312 1 2 tcp 1040 ----- 0 0.0 2.0 30 45
- 1.94312 1 2 tcp 1040 ----- 0 0.0 2.0 30 45
r 1.9432 1 2 tcp 1040 ----- 0 0.0 2.0 17 32
+ 1.9432 2 1 ack 40 ----- 0 2.0 0.0 17 48
- 1.9432 2 1 ack 40 ----- 0 2.0 0.0 17 48
r 1.94736 1 2 tcp 1040 ----- 0 0.0 2.0 18 33
+ 1.94736 2 1 ack 40 ----- 0 2.0 0.0 18 49
- 1.94736 2 1 ack 40 ----- 0 2.0 0.0 18 49
r 1.95152 1 2 tcp 1040 ----- 0 0.0 2.0 19 34
+ 1.95152 2 1 ack 40 ----- 0 2.0 0.0 19 50
- 1.95152 2 1 ack 40 ----- 0 2.0 0.0 19 50
r 1.95568 1 2 tcp 1040 ----- 0 0.0 2.0 20 35
+ 1.95568 2 1 ack 40 ----- 0 2.0 0.0 20 51
- 1.95568 2 1 ack 40 ----- 0 2.0 0.0 20 51
r 1.95984 1 2 tcp 1040 ----- 0 0.0 2.0 21 36
+ 1.95984 2 1 ack 40 ----- 0 2.0 0.0 21 52
- 1.95984 2 1 ack 40 ----- 0 2.0 0.0 21 52
r 1.964 1 2 tcp 1040 ----- 0 0.0 2.0 22 37
+ 1.964 2 1 ack 40 ----- 0 2.0 0.0 22 53
- 1.964 2 1 ack 40 ----- 0 2.0 0.0 22 53
r 1.96816 1 2 tcp 1040 ----- 0 0.0 2.0 23 38
+ 1.96816 2 1 ack 40 ----- 0 2.0 0.0 23 54
- 1.96816 2 1 ack 40 ----- 0 2.0 0.0 23 54
r 1.97232 1 2 tcp 1040 ----- 0 0.0 2.0 24 39
+ 1.97232 2 1 ack 40 ----- 0 2.0 0.0 24 55
- 1.97232 2 1 ack 40 ----- 0 2.0 0.0 24 55
r 1.97648 1 2 tcp 1040 ----- 0 0.0 2.0 25 40
+ 1.97648 2 1 ack 40 ----- 0 2.0 0.0 25 56
- 1.97648 2 1 ack 40 ----- 0 2.0 0.0 25 56
r 1.98064 1 2 tcp 1040 ----- 0 0.0 2.0 26 41
+ 1.98064 2 1 ack 40 ----- 0 2.0 0.0 26 57
- 1.98064 2 1 ack 40 ----- 0 2.0 0.0 26 57

```

r 1.9848 1 2 tcp 1040 ----- 0 0.0 2.0 27 42
+ 1.9848 2 1 ack 40 ----- 0 2.0 0.0 27 58
- 1.9848 2 1 ack 40 ----- 0 2.0 0.0 27 58
r 1.98504 2 1 ack 40 ----- 0 2.0 0.0 15 46
+ 1.98504 1 0 ack 40 ----- 0 2.0 0.0 15 46
- 1.98504 1 0 ack 40 ----- 0 2.0 0.0 15 46
r 1.98896 1 2 tcp 1040 ----- 0 0.0 2.0 28 43
+ 1.98896 2 1 ack 40 ----- 0 2.0 0.0 28 59
- 1.98896 2 1 ack 40 ----- 0 2.0 0.0 28 59
r 1.9892 2 1 ack 40 ----- 0 2.0 0.0 16 47
+ 1.9892 1 0 ack 40 ----- 0 2.0 0.0 16 47
- 1.9892 1 0 ack 40 ----- 0 2.0 0.0 16 47
r 1.99312 1 2 tcp 1040 ----- 0 0.0 2.0 29 44
+ 1.99312 2 1 ack 40 ----- 0 2.0 0.0 29 60
- 1.99312 2 1 ack 40 ----- 0 2.0 0.0 29 60
r 1.99336 2 1 ack 40 ----- 0 2.0 0.0 17 48
+ 1.99336 1 0 ack 40 ----- 0 2.0 0.0 17 48
- 1.99336 1 0 ack 40 ----- 0 2.0 0.0 17 48
r 1.99728 1 2 tcp 1040 ----- 0 0.0 2.0 30 45
+ 1.99728 2 1 ack 40 ----- 0 2.0 0.0 30 61
- 1.99728 2 1 ack 40 ----- 0 2.0 0.0 30 61
r 1.99752 2 1 ack 40 ----- 0 2.0 0.0 18 49
+ 1.99752 1 0 ack 40 ----- 0 2.0 0.0 18 49
- 1.99752 1 0 ack 40 ----- 0 2.0 0.0 18 49
r 2.00168 2 1 ack 40 ----- 0 2.0 0.0 19 50
+ 2.00168 1 0 ack 40 ----- 0 2.0 0.0 19 50
- 2.00168 1 0 ack 40 ----- 0 2.0 0.0 19 50
r 2.00584 2 1 ack 40 ----- 0 2.0 0.0 20 51
+ 2.00584 1 0 ack 40 ----- 0 2.0 0.0 20 51
- 2.00584 1 0 ack 40 ----- 0 2.0 0.0 20 51
r 2.01 2 1 ack 40 ----- 0 2.0 0.0 21 52
+ 2.01 1 0 ack 40 ----- 0 2.0 0.0 21 52
- 2.01 1 0 ack 40 ----- 0 2.0 0.0 21 52

```

```

r 2.01416 2 1 ack 40 ----- 0 2.0 0.0 22 53
+ 2.01416 1 0 ack 40 ----- 0 2.0 0.0 22 53
- 2.01416 1 0 ack 40 ----- 0 2.0 0.0 22 53
r 2.01832 2 1 ack 40 ----- 0 2.0 0.0 23 54
+ 2.01832 1 0 ack 40 ----- 0 2.0 0.0 23 54
- 2.01832 1 0 ack 40 ----- 0 2.0 0.0 23 54
r 2.02248 2 1 ack 40 ----- 0 2.0 0.0 24 55
+ 2.02248 1 0 ack 40 ----- 0 2.0 0.0 24 55
- 2.02248 1 0 ack 40 ----- 0 2.0 0.0 24 55
r 2.02664 2 1 ack 40 ----- 0 2.0 0.0 25 56
+ 2.02664 1 0 ack 40 ----- 0 2.0 0.0 25 56
- 2.02664 1 0 ack 40 ----- 0 2.0 0.0 25 56
r 2.0308 2 1 ack 40 ----- 0 2.0 0.0 26 57
+ 2.0308 1 0 ack 40 ----- 0 2.0 0.0 26 57
- 2.0308 1 0 ack 40 ----- 0 2.0 0.0 26 57
r 2.03496 2 1 ack 40 ----- 0 2.0 0.0 27 58
+ 2.03496 1 0 ack 40 ----- 0 2.0 0.0 27 58
- 2.03496 1 0 ack 40 ----- 0 2.0 0.0 27 58
r 2.0352 1 0 ack 40 ----- 0 2.0 0.0 15 46
+ 2.0352 0 1 tcp 1040 ----- 0 0.0 2.0 31 62
- 2.0352 0 1 tcp 1040 ----- 0 0.0 2.0 31 62
+ 2.0352 0 1 tcp 1040 ----- 0 0.0 2.0 32 63
r 2.03912 2 1 ack 40 ----- 0 2.0 0.0 28 59
+ 2.03912 1 0 ack 40 ----- 0 2.0 0.0 28 59
- 2.03912 1 0 ack 40 ----- 0 2.0 0.0 28 59
r 2.03936 1 0 ack 40 ----- 0 2.0 0.0 16 47
+ 2.03936 0 1 tcp 1040 ----- 0 0.0 2.0 33 64
+ 2.03936 0 1 tcp 1040 ----- 0 0.0 2.0 34 65
- 2.03936 0 1 tcp 1040 ----- 0 0.0 2.0 32 63
r 2.04328 2 1 ack 40 ----- 0 2.0 0.0 29 60
+ 2.04328 1 0 ack 40 ----- 0 2.0 0.0 29 60
- 2.04328 1 0 ack 40 ----- 0 2.0 0.0 29 60
r 2.04352 1 0 ack 40 ----- 0 2.0 0.0 17 48

```

```

+ 2.04352 0 1 tcp 1040 ----- 0 0.0 2.0 35 66
+ 2.04352 0 1 tcp 1040 ----- 0 0.0 2.0 36 67
- 2.04352 0 1 tcp 1040 ----- 0 0.0 2.0 33 64
r 2.04744 2 1 ack 40 ----- 0 2.0 0.0 30 61
+ 2.04744 1 0 ack 40 ----- 0 2.0 0.0 30 61
- 2.04744 1 0 ack 40 ----- 0 2.0 0.0 30 61
r 2.04768 1 0 ack 40 ----- 0 2.0 0.0 18 49
+ 2.04768 0 1 tcp 1040 ----- 0 0.0 2.0 37 68
+ 2.04768 0 1 tcp 1040 ----- 0 0.0 2.0 38 69
- 2.04768 0 1 tcp 1040 ----- 0 0.0 2.0 34 65
r 2.05184 1 0 ack 40 ----- 0 2.0 0.0 19 50
+ 2.05184 0 1 tcp 1040 ----- 0 0.0 2.0 39 70
- 2.05184 0 1 tcp 1040 ----- 0 0.0 2.0 35 66
r 2.056 1 0 ack 40 ----- 0 2.0 0.0 20 51
+ 2.056 0 1 tcp 1040 ----- 0 0.0 2.0 40 71
- 2.056 0 1 tcp 1040 ----- 0 0.0 2.0 36 67
r 2.06016 1 0 ack 40 ----- 0 2.0 0.0 21 52
+ 2.06016 0 1 tcp 1040 ----- 0 0.0 2.0 41 72
- 2.06016 0 1 tcp 1040 ----- 0 0.0 2.0 37 68
r 2.06432 1 0 ack 40 ----- 0 2.0 0.0 22 53
+ 2.06432 0 1 tcp 1040 ----- 0 0.0 2.0 42 73
- 2.06432 0 1 tcp 1040 ----- 0 0.0 2.0 38 69
r 2.06848 1 0 ack 40 ----- 0 2.0 0.0 23 54
+ 2.06848 0 1 tcp 1040 ----- 0 0.0 2.0 43 74
- 2.06848 0 1 tcp 1040 ----- 0 0.0 2.0 39 70
r 2.07264 1 0 ack 40 ----- 0 2.0 0.0 24 55
+ 2.07264 0 1 tcp 1040 ----- 0 0.0 2.0 44 75
- 2.07264 0 1 tcp 1040 ----- 0 0.0 2.0 40 71
r 2.0768 1 0 ack 40 ----- 0 2.0 0.0 25 56
+ 2.0768 0 1 tcp 1040 ----- 0 0.0 2.0 45 76
- 2.0768 0 1 tcp 1040 ----- 0 0.0 2.0 41 72
r 2.08096 1 0 ack 40 ----- 0 2.0 0.0 26 57
+ 2.08096 0 1 tcp 1040 ----- 0 0.0 2.0 46 77

```



```

- 2.08096 0 1 tcp 1040 ----- 0 0.0 2.0 42 73
r 2.08512 1 0 ack 40 ----- 0 2.0 0.0 27 58
+ 2.08512 0 1 tcp 1040 ----- 0 0.0 2.0 47 78
- 2.08512 0 1 tcp 1040 ----- 0 0.0 2.0 43 74
r 2.08928 1 0 ack 40 ----- 0 2.0 0.0 28 59
+ 2.08928 0 1 tcp 1040 ----- 0 0.0 2.0 48 79
- 2.08928 0 1 tcp 1040 ----- 0 0.0 2.0 44 75
r 2.08936 0 1 tcp 1040 ----- 0 0.0 2.0 31 62
+ 2.08936 1 2 tcp 1040 ----- 0 0.0 2.0 31 62
- 2.08936 1 2 tcp 1040 ----- 0 0.0 2.0 31 62
r 2.09344 1 0 ack 40 ----- 0 2.0 0.0 29 60
+ 2.09344 0 1 tcp 1040 ----- 0 0.0 2.0 49 80
- 2.09344 0 1 tcp 1040 ----- 0 0.0 2.0 45 76
r 2.09352 0 1 tcp 1040 ----- 0 0.0 2.0 32 63
+ 2.09352 1 2 tcp 1040 ----- 0 0.0 2.0 32 63
- 2.09352 1 2 tcp 1040 ----- 0 0.0 2.0 32 63
r 2.0976 1 0 ack 40 ----- 0 2.0 0.0 30 61
+ 2.0976 0 1 tcp 1040 ----- 0 0.0 2.0 50 81
- 2.0976 0 1 tcp 1040 ----- 0 0.0 2.0 46 77
r 2.09768 0 1 tcp 1040 ----- 0 0.0 2.0 33 64
+ 2.09768 1 2 tcp 1040 ----- 0 0.0 2.0 33 64
- 2.09768 1 2 tcp 1040 ----- 0 0.0 2.0 33 64
- 2.10176 0 1 tcp 1040 ----- 0 0.0 2.0 47 78
r 2.10184 0 1 tcp 1040 ----- 0 0.0 2.0 34 65
+ 2.10184 1 2 tcp 1040 ----- 0 0.0 2.0 34 65
- 2.10184 1 2 tcp 1040 ----- 0 0.0 2.0 34 65
- 2.10592 0 1 tcp 1040 ----- 0 0.0 2.0 48 79
r 2.106 0 1 tcp 1040 ----- 0 0.0 2.0 35 66
+ 2.106 1 2 tcp 1040 ----- 0 0.0 2.0 35 66
- 2.106 1 2 tcp 1040 ----- 0 0.0 2.0 35 66
- 2.11008 0 1 tcp 1040 ----- 0 0.0 2.0 49 80
r 2.11016 0 1 tcp 1040 ----- 0 0.0 2.0 36 67
+ 2.11016 1 2 tcp 1040 ----- 0 0.0 2.0 36 67

```

```

- 2.11016 1 2 tcp 1040 ----- 0 0.0 2.0 36 67
- 2.11424 0 1 tcp 1040 ----- 0 0.0 2.0 50 81
r 2.11432 0 1 tcp 1040 ----- 0 0.0 2.0 37 68
+ 2.11432 1 2 tcp 1040 ----- 0 0.0 2.0 37 68
- 2.11432 1 2 tcp 1040 ----- 0 0.0 2.0 37 68
r 2.11848 0 1 tcp 1040 ----- 0 0.0 2.0 38 69
+ 2.11848 1 2 tcp 1040 ----- 0 0.0 2.0 38 69
- 2.11848 1 2 tcp 1040 ----- 0 0.0 2.0 38 69
r 2.12264 0 1 tcp 1040 ----- 0 0.0 2.0 39 70
+ 2.12264 1 2 tcp 1040 ----- 0 0.0 2.0 39 70
- 2.12264 1 2 tcp 1040 ----- 0 0.0 2.0 39 70
r 2.1268 0 1 tcp 1040 ----- 0 0.0 2.0 40 71
+ 2.1268 1 2 tcp 1040 ----- 0 0.0 2.0 40 71
- 2.1268 1 2 tcp 1040 ----- 0 0.0 2.0 40 71
r 2.13096 0 1 tcp 1040 ----- 0 0.0 2.0 41 72
+ 2.13096 1 2 tcp 1040 ----- 0 0.0 2.0 41 72
- 2.13096 1 2 tcp 1040 ----- 0 0.0 2.0 41 72
r 2.13512 0 1 tcp 1040 ----- 0 0.0 2.0 42 73
+ 2.13512 1 2 tcp 1040 ----- 0 0.0 2.0 42 73
- 2.13512 1 2 tcp 1040 ----- 0 0.0 2.0 42 73
r 2.13928 0 1 tcp 1040 ----- 0 0.0 2.0 43 74
+ 2.13928 1 2 tcp 1040 ----- 0 0.0 2.0 43 74
- 2.13928 1 2 tcp 1040 ----- 0 0.0 2.0 43 74
r 2.14344 0 1 tcp 1040 ----- 0 0.0 2.0 44 75
+ 2.14344 1 2 tcp 1040 ----- 0 0.0 2.0 44 75
- 2.14344 1 2 tcp 1040 ----- 0 0.0 2.0 44 75
r 2.14352 1 2 tcp 1040 ----- 0 0.0 2.0 31 62
+ 2.14352 2 1 ack 40 ----- 0 2.0 0.0 31 82
- 2.14352 2 1 ack 40 ----- 0 2.0 0.0 31 82
r 2.1476 0 1 tcp 1040 ----- 0 0.0 2.0 45 76
+ 2.1476 1 2 tcp 1040 ----- 0 0.0 2.0 45 76
- 2.1476 1 2 tcp 1040 ----- 0 0.0 2.0 45 76
r 2.14768 1 2 tcp 1040 ----- 0 0.0 2.0 32 63

```

```

+ 2.14768 2 1 ack 40 ----- 0 2.0 0.0 32 83
- 2.14768 2 1 ack 40 ----- 0 2.0 0.0 32 83
r 2.15176 0 1 tcp 1040 ----- 0 0.0 2.0 46 77
+ 2.15176 1 2 tcp 1040 ----- 0 0.0 2.0 46 77
- 2.15176 1 2 tcp 1040 ----- 0 0.0 2.0 46 77
r 2.15184 1 2 tcp 1040 ----- 0 0.0 2.0 33 64
+ 2.15184 2 1 ack 40 ----- 0 2.0 0.0 33 84
- 2.15184 2 1 ack 40 ----- 0 2.0 0.0 33 84
r 2.15592 0 1 tcp 1040 ----- 0 0.0 2.0 47 78
+ 2.15592 1 2 tcp 1040 ----- 0 0.0 2.0 47 78
- 2.15592 1 2 tcp 1040 ----- 0 0.0 2.0 47 78
r 2.156 1 2 tcp 1040 ----- 0 0.0 2.0 34 65
+ 2.156 2 1 ack 40 ----- 0 2.0 0.0 34 85
- 2.156 2 1 ack 40 ----- 0 2.0 0.0 34 85
r 2.16008 0 1 tcp 1040 ----- 0 0.0 2.0 48 79
+ 2.16008 1 2 tcp 1040 ----- 0 0.0 2.0 48 79
- 2.16008 1 2 tcp 1040 ----- 0 0.0 2.0 48 79
r 2.16016 1 2 tcp 1040 ----- 0 0.0 2.0 35 66
+ 2.16016 2 1 ack 40 ----- 0 2.0 0.0 35 86
- 2.16016 2 1 ack 40 ----- 0 2.0 0.0 35 86
r 2.16424 0 1 tcp 1040 ----- 0 0.0 2.0 49 80
+ 2.16424 1 2 tcp 1040 ----- 0 0.0 2.0 49 80
- 2.16424 1 2 tcp 1040 ----- 0 0.0 2.0 49 80
r 2.16432 1 2 tcp 1040 ----- 0 0.0 2.0 36 67
+ 2.16432 2 1 ack 40 ----- 0 2.0 0.0 36 87
- 2.16432 2 1 ack 40 ----- 0 2.0 0.0 36 87
r 2.1684 0 1 tcp 1040 ----- 0 0.0 2.0 50 81
+ 2.1684 1 2 tcp 1040 ----- 0 0.0 2.0 50 81
- 2.1684 1 2 tcp 1040 ----- 0 0.0 2.0 50 81
r 2.16848 1 2 tcp 1040 ----- 0 0.0 2.0 37 68
+ 2.16848 2 1 ack 40 ----- 0 2.0 0.0 37 88
- 2.16848 2 1 ack 40 ----- 0 2.0 0.0 37 88
r 2.17264 1 2 tcp 1040 ----- 0 0.0 2.0 38 69

```

```

+ 2.17264 2 1 ack 40 ----- 0 2.0 0.0 38 89
- 2.17264 2 1 ack 40 ----- 0 2.0 0.0 38 89
r 2.1768 1 2 tcp 1040 ----- 0 0.0 2.0 39 70
+ 2.1768 2 1 ack 40 ----- 0 2.0 0.0 39 90
- 2.1768 2 1 ack 40 ----- 0 2.0 0.0 39 90
r 2.18096 1 2 tcp 1040 ----- 0 0.0 2.0 40 71
+ 2.18096 2 1 ack 40 ----- 0 2.0 0.0 40 91
- 2.18096 2 1 ack 40 ----- 0 2.0 0.0 40 91
r 2.18512 1 2 tcp 1040 ----- 0 0.0 2.0 41 72
+ 2.18512 2 1 ack 40 ----- 0 2.0 0.0 41 92
- 2.18512 2 1 ack 40 ----- 0 2.0 0.0 41 92
r 2.18928 1 2 tcp 1040 ----- 0 0.0 2.0 42 73
+ 2.18928 2 1 ack 40 ----- 0 2.0 0.0 42 93
- 2.18928 2 1 ack 40 ----- 0 2.0 0.0 42 93
r 2.19344 1 2 tcp 1040 ----- 0 0.0 2.0 43 74
+ 2.19344 2 1 ack 40 ----- 0 2.0 0.0 43 94
- 2.19344 2 1 ack 40 ----- 0 2.0 0.0 43 94
r 2.19368 2 1 ack 40 ----- 0 2.0 0.0 31 82
+ 2.19368 1 0 ack 40 ----- 0 2.0 0.0 31 82
- 2.19368 1 0 ack 40 ----- 0 2.0 0.0 31 82
r 2.1976 1 2 tcp 1040 ----- 0 0.0 2.0 44 75
+ 2.1976 2 1 ack 40 ----- 0 2.0 0.0 44 95
- 2.1976 2 1 ack 40 ----- 0 2.0 0.0 44 95
r 2.19784 2 1 ack 40 ----- 0 2.0 0.0 32 83
+ 2.19784 1 0 ack 40 ----- 0 2.0 0.0 32 83
- 2.19784 1 0 ack 40 ----- 0 2.0 0.0 32 83
r 2.20176 1 2 tcp 1040 ----- 0 0.0 2.0 45 76
+ 2.20176 2 1 ack 40 ----- 0 2.0 0.0 45 96
- 2.20176 2 1 ack 40 ----- 0 2.0 0.0 45 96
r 2.202 2 1 ack 40 ----- 0 2.0 0.0 33 84
+ 2.202 1 0 ack 40 ----- 0 2.0 0.0 33 84
- 2.202 1 0 ack 40 ----- 0 2.0 0.0 33 84
r 2.20592 1 2 tcp 1040 ----- 0 0.0 2.0 46 77

```

+ 2.20592 2 1 ack 40 ----- 0 2.0 0.0 46 97
- 2.20592 2 1 ack 40 ----- 0 2.0 0.0 46 97
r 2.20616 2 1 ack 40 ----- 0 2.0 0.0 34 85
+ 2.20616 1 0 ack 40 ----- 0 2.0 0.0 34 85
- 2.20616 1 0 ack 40 ----- 0 2.0 0.0 34 85
r 2.21008 1 2 tcp 1040 ----- 0 0.0 2.0 47 78
+ 2.21008 2 1 ack 40 ----- 0 2.0 0.0 47 98
- 2.21008 2 1 ack 40 ----- 0 2.0 0.0 47 98
r 2.21032 2 1 ack 40 ----- 0 2.0 0.0 35 86
+ 2.21032 1 0 ack 40 ----- 0 2.0 0.0 35 86
- 2.21032 1 0 ack 40 ----- 0 2.0 0.0 35 86
r 2.21424 1 2 tcp 1040 ----- 0 0.0 2.0 48 79
+ 2.21424 2 1 ack 40 ----- 0 2.0 0.0 48 99
- 2.21424 2 1 ack 40 ----- 0 2.0 0.0 48 99
r 2.21448 2 1 ack 40 ----- 0 2.0 0.0 36 87
+ 2.21448 1 0 ack 40 ----- 0 2.0 0.0 36 87
- 2.21448 1 0 ack 40 ----- 0 2.0 0.0 36 87
r 2.2184 1 2 tcp 1040 ----- 0 0.0 2.0 49 80
+ 2.2184 2 1 ack 40 ----- 0 2.0 0.0 49 100
- 2.2184 2 1 ack 40 ----- 0 2.0 0.0 49 100
r 2.21864 2 1 ack 40 ----- 0 2.0 0.0 37 88
+ 2.21864 1 0 ack 40 ----- 0 2.0 0.0 37 88
- 2.21864 1 0 ack 40 ----- 0 2.0 0.0 37 88
r 2.22256 1 2 tcp 1040 ----- 0 0.0 2.0 50 81
+ 2.22256 2 1 ack 40 ----- 0 2.0 0.0 50 101
- 2.22256 2 1 ack 40 ----- 0 2.0 0.0 50 101
r 2.2228 2 1 ack 40 ----- 0 2.0 0.0 38 89
+ 2.2228 1 0 ack 40 ----- 0 2.0 0.0 38 89
- 2.2228 1 0 ack 40 ----- 0 2.0 0.0 38 89
r 2.22696 2 1 ack 40 ----- 0 2.0 0.0 39 90
+ 2.22696 1 0 ack 40 ----- 0 2.0 0.0 39 90
- 2.22696 1 0 ack 40 ----- 0 2.0 0.0 39 90
r 2.23112 2 1 ack 40 ----- 0 2.0 0.0 40 91

```

+ 2.23112 1 0 ack 40 ----- 0 2.0 0.0 40 91
- 2.23112 1 0 ack 40 ----- 0 2.0 0.0 40 91
r 2.23528 2 1 ack 40 ----- 0 2.0 0.0 41 92
+ 2.23528 1 0 ack 40 ----- 0 2.0 0.0 41 92
- 2.23528 1 0 ack 40 ----- 0 2.0 0.0 41 92
r 2.23944 2 1 ack 40 ----- 0 2.0 0.0 42 93
+ 2.23944 1 0 ack 40 ----- 0 2.0 0.0 42 93
- 2.23944 1 0 ack 40 ----- 0 2.0 0.0 42 93
r 2.2436 2 1 ack 40 ----- 0 2.0 0.0 43 94
+ 2.2436 1 0 ack 40 ----- 0 2.0 0.0 43 94
- 2.2436 1 0 ack 40 ----- 0 2.0 0.0 43 94
r 2.24384 1 0 ack 40 ----- 0 2.0 0.0 31 82
+ 2.24384 0 1 tcp 1040 ----- 0 0.0 2.0 51 102
- 2.24384 0 1 tcp 1040 ----- 0 0.0 2.0 51 102
r 2.24776 2 1 ack 40 ----- 0 2.0 0.0 44 95
+ 2.24776 1 0 ack 40 ----- 0 2.0 0.0 44 95
- 2.24776 1 0 ack 40 ----- 0 2.0 0.0 44 95
r 2.248 1 0 ack 40 ----- 0 2.0 0.0 32 83
+ 2.248 0 1 tcp 1040 ----- 0 0.0 2.0 52 103
- 2.248 0 1 tcp 1040 ----- 0 0.0 2.0 52 103
r 2.25192 2 1 ack 40 ----- 0 2.0 0.0 45 96
+ 2.25192 1 0 ack 40 ----- 0 2.0 0.0 45 96
- 2.25192 1 0 ack 40 ----- 0 2.0 0.0 45 96
r 2.25216 1 0 ack 40 ----- 0 2.0 0.0 33 84
+ 2.25216 0 1 tcp 1040 ----- 0 0.0 2.0 53 104
- 2.25216 0 1 tcp 1040 ----- 0 0.0 2.0 53 104
r 2.25608 2 1 ack 40 ----- 0 2.0 0.0 46 97
+ 2.25608 1 0 ack 40 ----- 0 2.0 0.0 46 97
- 2.25608 1 0 ack 40 ----- 0 2.0 0.0 46 97
r 2.25632 1 0 ack 40 ----- 0 2.0 0.0 34 85
+ 2.25632 0 1 tcp 1040 ----- 0 0.0 2.0 54 105
- 2.25632 0 1 tcp 1040 ----- 0 0.0 2.0 54 105
r 2.26024 2 1 ack 40 ----- 0 2.0 0.0 47 98

```

```

+ 2.26024 1 0 ack 40 ----- 0 2.0 0.0 47 98
- 2.26024 1 0 ack 40 ----- 0 2.0 0.0 47 98
r 2.26048 1 0 ack 40 ----- 0 2.0 0.0 35 86
+ 2.26048 0 1 tcp 1040 ----- 0 0.0 2.0 55 106
- 2.26048 0 1 tcp 1040 ----- 0 0.0 2.0 55 106
r 2.2644 2 1 ack 40 ----- 0 2.0 0.0 48 99
+ 2.2644 1 0 ack 40 ----- 0 2.0 0.0 48 99
- 2.2644 1 0 ack 40 ----- 0 2.0 0.0 48 99
r 2.26464 1 0 ack 40 ----- 0 2.0 0.0 36 87
+ 2.26464 0 1 tcp 1040 ----- 0 0.0 2.0 56 107
- 2.26464 0 1 tcp 1040 ----- 0 0.0 2.0 56 107
r 2.26856 2 1 ack 40 ----- 0 2.0 0.0 49 100
+ 2.26856 1 0 ack 40 ----- 0 2.0 0.0 49 100
- 2.26856 1 0 ack 40 ----- 0 2.0 0.0 49 100
r 2.2688 1 0 ack 40 ----- 0 2.0 0.0 37 88
+ 2.2688 0 1 tcp 1040 ----- 0 0.0 2.0 57 108
- 2.2688 0 1 tcp 1040 ----- 0 0.0 2.0 57 108
r 2.27272 2 1 ack 40 ----- 0 2.0 0.0 50 101
+ 2.27272 1 0 ack 40 ----- 0 2.0 0.0 50 101
- 2.27272 1 0 ack 40 ----- 0 2.0 0.0 50 101
r 2.27296 1 0 ack 40 ----- 0 2.0 0.0 38 89
+ 2.27296 0 1 tcp 1040 ----- 0 0.0 2.0 58 109
- 2.27296 0 1 tcp 1040 ----- 0 0.0 2.0 58 109
r 2.27712 1 0 ack 40 ----- 0 2.0 0.0 39 90
+ 2.27712 0 1 tcp 1040 ----- 0 0.0 2.0 59 110
- 2.27712 0 1 tcp 1040 ----- 0 0.0 2.0 59 110
r 2.28128 1 0 ack 40 ----- 0 2.0 0.0 40 91
+ 2.28128 0 1 tcp 1040 ----- 0 0.0 2.0 60 111
- 2.28128 0 1 tcp 1040 ----- 0 0.0 2.0 60 111
r 2.28544 1 0 ack 40 ----- 0 2.0 0.0 41 92
+ 2.28544 0 1 tcp 1040 ----- 0 0.0 2.0 61 112
- 2.28544 0 1 tcp 1040 ----- 0 0.0 2.0 61 112
r 2.2896 1 0 ack 40 ----- 0 2.0 0.0 42 93

```

```

+ 2.2896 0 1 tcp 1040 ----- 0 0.0 2.0 62 113
- 2.2896 0 1 tcp 1040 ----- 0 0.0 2.0 62 113
r 2.29376 1 0 ack 40 ----- 0 2.0 0.0 43 94
+ 2.29376 0 1 tcp 1040 ----- 0 0.0 2.0 63 114
- 2.29376 0 1 tcp 1040 ----- 0 0.0 2.0 63 114
r 2.29792 1 0 ack 40 ----- 0 2.0 0.0 44 95
+ 2.29792 0 1 tcp 1040 ----- 0 0.0 2.0 64 115
- 2.29792 0 1 tcp 1040 ----- 0 0.0 2.0 64 115
r 2.298 0 1 tcp 1040 ----- 0 0.0 2.0 51 102
+ 2.298 1 2 tcp 1040 ----- 0 0.0 2.0 51 102
- 2.298 1 2 tcp 1040 ----- 0 0.0 2.0 51 102
r 2.30208 1 0 ack 40 ----- 0 2.0 0.0 45 96
+ 2.30208 0 1 tcp 1040 ----- 0 0.0 2.0 65 116
- 2.30208 0 1 tcp 1040 ----- 0 0.0 2.0 65 116
r 2.30216 0 1 tcp 1040 ----- 0 0.0 2.0 52 103
+ 2.30216 1 2 tcp 1040 ----- 0 0.0 2.0 52 103
- 2.30216 1 2 tcp 1040 ----- 0 0.0 2.0 52 103
r 2.30624 1 0 ack 40 ----- 0 2.0 0.0 46 97
+ 2.30624 0 1 tcp 1040 ----- 0 0.0 2.0 66 117
- 2.30624 0 1 tcp 1040 ----- 0 0.0 2.0 66 117
r 2.30632 0 1 tcp 1040 ----- 0 0.0 2.0 53 104
+ 2.30632 1 2 tcp 1040 ----- 0 0.0 2.0 53 104
- 2.30632 1 2 tcp 1040 ----- 0 0.0 2.0 53 104
r 2.3104 1 0 ack 40 ----- 0 2.0 0.0 47 98
+ 2.3104 0 1 tcp 1040 ----- 0 0.0 2.0 67 118
- 2.3104 0 1 tcp 1040 ----- 0 0.0 2.0 67 118
r 2.31048 0 1 tcp 1040 ----- 0 0.0 2.0 54 105
+ 2.31048 1 2 tcp 1040 ----- 0 0.0 2.0 54 105
- 2.31048 1 2 tcp 1040 ----- 0 0.0 2.0 54 105
r 2.31456 1 0 ack 40 ----- 0 2.0 0.0 48 99
+ 2.31456 0 1 tcp 1040 ----- 0 0.0 2.0 68 119
- 2.31456 0 1 tcp 1040 ----- 0 0.0 2.0 68 119
r 2.31464 0 1 tcp 1040 ----- 0 0.0 2.0 55 106

```



```

+ 2.31464 1 2 tcp 1040 ----- 0 0.0 2.0 55 106
- 2.31464 1 2 tcp 1040 ----- 0 0.0 2.0 55 106
r 2.31872 1 0 ack 40 ----- 0 2.0 0.0 49 100
+ 2.31872 0 1 tcp 1040 ----- 0 0.0 2.0 69 120
- 2.31872 0 1 tcp 1040 ----- 0 0.0 2.0 69 120
r 2.3188 0 1 tcp 1040 ----- 0 0.0 2.0 56 107
+ 2.3188 1 2 tcp 1040 ----- 0 0.0 2.0 56 107
- 2.3188 1 2 tcp 1040 ----- 0 0.0 2.0 56 107
r 2.32288 1 0 ack 40 ----- 0 2.0 0.0 50 101
+ 2.32288 0 1 tcp 1040 ----- 0 0.0 2.0 70 121
- 2.32288 0 1 tcp 1040 ----- 0 0.0 2.0 70 121
r 2.32296 0 1 tcp 1040 ----- 0 0.0 2.0 57 108
+ 2.32296 1 2 tcp 1040 ----- 0 0.0 2.0 57 108
- 2.32296 1 2 tcp 1040 ----- 0 0.0 2.0 57 108
r 2.32712 0 1 tcp 1040 ----- 0 0.0 2.0 58 109
+ 2.32712 1 2 tcp 1040 ----- 0 0.0 2.0 58 109
- 2.32712 1 2 tcp 1040 ----- 0 0.0 2.0 58 109
r 2.33128 0 1 tcp 1040 ----- 0 0.0 2.0 59 110
+ 2.33128 1 2 tcp 1040 ----- 0 0.0 2.0 59 110
- 2.33128 1 2 tcp 1040 ----- 0 0.0 2.0 59 110
r 2.33544 0 1 tcp 1040 ----- 0 0.0 2.0 60 111
+ 2.33544 1 2 tcp 1040 ----- 0 0.0 2.0 60 111
- 2.33544 1 2 tcp 1040 ----- 0 0.0 2.0 60 111
r 2.3396 0 1 tcp 1040 ----- 0 0.0 2.0 61 112
+ 2.3396 1 2 tcp 1040 ----- 0 0.0 2.0 61 112
- 2.3396 1 2 tcp 1040 ----- 0 0.0 2.0 61 112
r 2.34376 0 1 tcp 1040 ----- 0 0.0 2.0 62 113
+ 2.34376 1 2 tcp 1040 ----- 0 0.0 2.0 62 113
- 2.34376 1 2 tcp 1040 ----- 0 0.0 2.0 62 113
r 2.34792 0 1 tcp 1040 ----- 0 0.0 2.0 63 114
+ 2.34792 1 2 tcp 1040 ----- 0 0.0 2.0 63 114
- 2.34792 1 2 tcp 1040 ----- 0 0.0 2.0 63 114
r 2.35208 0 1 tcp 1040 ----- 0 0.0 2.0 64 115

```

```

+ 2.35208 1 2 tcp 1040 ----- 0 0.0 2.0 64 115
- 2.35208 1 2 tcp 1040 ----- 0 0.0 2.0 64 115
r 2.35216 1 2 tcp 1040 ----- 0 0.0 2.0 51 102
+ 2.35216 2 1 ack 40 ----- 0 2.0 0.0 51 122
- 2.35216 2 1 ack 40 ----- 0 2.0 0.0 51 122
r 2.35624 0 1 tcp 1040 ----- 0 0.0 2.0 65 116
+ 2.35624 1 2 tcp 1040 ----- 0 0.0 2.0 65 116
- 2.35624 1 2 tcp 1040 ----- 0 0.0 2.0 65 116
r 2.35632 1 2 tcp 1040 ----- 0 0.0 2.0 52 103
+ 2.35632 2 1 ack 40 ----- 0 2.0 0.0 52 123
- 2.35632 2 1 ack 40 ----- 0 2.0 0.0 52 123
r 2.3604 0 1 tcp 1040 ----- 0 0.0 2.0 66 117
+ 2.3604 1 2 tcp 1040 ----- 0 0.0 2.0 66 117
- 2.3604 1 2 tcp 1040 ----- 0 0.0 2.0 66 117
r 2.36048 1 2 tcp 1040 ----- 0 0.0 2.0 53 104
+ 2.36048 2 1 ack 40 ----- 0 2.0 0.0 53 124
- 2.36048 2 1 ack 40 ----- 0 2.0 0.0 53 124
r 2.36456 0 1 tcp 1040 ----- 0 0.0 2.0 67 118
+ 2.36456 1 2 tcp 1040 ----- 0 0.0 2.0 67 118
- 2.36456 1 2 tcp 1040 ----- 0 0.0 2.0 67 118
r 2.36464 1 2 tcp 1040 ----- 0 0.0 2.0 54 105
+ 2.36464 2 1 ack 40 ----- 0 2.0 0.0 54 125
- 2.36464 2 1 ack 40 ----- 0 2.0 0.0 54 125
r 2.36872 0 1 tcp 1040 ----- 0 0.0 2.0 68 119
+ 2.36872 1 2 tcp 1040 ----- 0 0.0 2.0 68 119
- 2.36872 1 2 tcp 1040 ----- 0 0.0 2.0 68 119
r 2.3688 1 2 tcp 1040 ----- 0 0.0 2.0 55 106
+ 2.3688 2 1 ack 40 ----- 0 2.0 0.0 55 126
- 2.3688 2 1 ack 40 ----- 0 2.0 0.0 55 126
r 2.37288 0 1 tcp 1040 ----- 0 0.0 2.0 69 120
+ 2.37288 1 2 tcp 1040 ----- 0 0.0 2.0 69 120
- 2.37288 1 2 tcp 1040 ----- 0 0.0 2.0 69 120
r 2.37296 1 2 tcp 1040 ----- 0 0.0 2.0 56 107

```

```

+ 2.37296 2 1 ack 40 ----- 0 2.0 0.0 56 127
- 2.37296 2 1 ack 40 ----- 0 2.0 0.0 56 127
r 2.37704 0 1 tcp 1040 ----- 0 0.0 2.0 70 121
+ 2.37704 1 2 tcp 1040 ----- 0 0.0 2.0 70 121
- 2.37704 1 2 tcp 1040 ----- 0 0.0 2.0 70 121
r 2.37712 1 2 tcp 1040 ----- 0 0.0 2.0 57 108
+ 2.37712 2 1 ack 40 ----- 0 2.0 0.0 57 128
- 2.37712 2 1 ack 40 ----- 0 2.0 0.0 57 128
r 2.38128 1 2 tcp 1040 ----- 0 0.0 2.0 58 109
+ 2.38128 2 1 ack 40 ----- 0 2.0 0.0 58 129
- 2.38128 2 1 ack 40 ----- 0 2.0 0.0 58 129
r 2.38544 1 2 tcp 1040 ----- 0 0.0 2.0 59 110
+ 2.38544 2 1 ack 40 ----- 0 2.0 0.0 59 130
- 2.38544 2 1 ack 40 ----- 0 2.0 0.0 59 130
r 2.3896 1 2 tcp 1040 ----- 0 0.0 2.0 60 111
+ 2.3896 2 1 ack 40 ----- 0 2.0 0.0 60 131
- 2.3896 2 1 ack 40 ----- 0 2.0 0.0 60 131
r 2.39376 1 2 tcp 1040 ----- 0 0.0 2.0 61 112
+ 2.39376 2 1 ack 40 ----- 0 2.0 0.0 61 132
- 2.39376 2 1 ack 40 ----- 0 2.0 0.0 61 132
r 2.39792 1 2 tcp 1040 ----- 0 0.0 2.0 62 113
+ 2.39792 2 1 ack 40 ----- 0 2.0 0.0 62 133
- 2.39792 2 1 ack 40 ----- 0 2.0 0.0 62 133
r 2.40208 1 2 tcp 1040 ----- 0 0.0 2.0 63 114
+ 2.40208 2 1 ack 40 ----- 0 2.0 0.0 63 134
- 2.40208 2 1 ack 40 ----- 0 2.0 0.0 63 134
r 2.40232 2 1 ack 40 ----- 0 2.0 0.0 51 122
+ 2.40232 1 0 ack 40 ----- 0 2.0 0.0 51 122
- 2.40232 1 0 ack 40 ----- 0 2.0 0.0 51 122
r 2.40624 1 2 tcp 1040 ----- 0 0.0 2.0 64 115
+ 2.40624 2 1 ack 40 ----- 0 2.0 0.0 64 135
- 2.40624 2 1 ack 40 ----- 0 2.0 0.0 64 135
r 2.40648 2 1 ack 40 ----- 0 2.0 0.0 52 123

```

```

+ 2.40648 1 0 ack 40 ----- 0 2.0 0.0 52 123
- 2.40648 1 0 ack 40 ----- 0 2.0 0.0 52 123
r 2.4104 1 2 tcp 1040 ----- 0 0.0 2.0 65 116
+ 2.4104 2 1 ack 40 ----- 0 2.0 0.0 65 136
- 2.4104 2 1 ack 40 ----- 0 2.0 0.0 65 136
r 2.41064 2 1 ack 40 ----- 0 2.0 0.0 53 124
+ 2.41064 1 0 ack 40 ----- 0 2.0 0.0 53 124
- 2.41064 1 0 ack 40 ----- 0 2.0 0.0 53 124
r 2.41456 1 2 tcp 1040 ----- 0 0.0 2.0 66 117
+ 2.41456 2 1 ack 40 ----- 0 2.0 0.0 66 137
- 2.41456 2 1 ack 40 ----- 0 2.0 0.0 66 137
r 2.4148 2 1 ack 40 ----- 0 2.0 0.0 54 125
+ 2.4148 1 0 ack 40 ----- 0 2.0 0.0 54 125
- 2.4148 1 0 ack 40 ----- 0 2.0 0.0 54 125
r 2.41872 1 2 tcp 1040 ----- 0 0.0 2.0 67 118
+ 2.41872 2 1 ack 40 ----- 0 2.0 0.0 67 138
- 2.41872 2 1 ack 40 ----- 0 2.0 0.0 67 138
r 2.41896 2 1 ack 40 ----- 0 2.0 0.0 55 126
+ 2.41896 1 0 ack 40 ----- 0 2.0 0.0 55 126
- 2.41896 1 0 ack 40 ----- 0 2.0 0.0 55 126
r 2.42288 1 2 tcp 1040 ----- 0 0.0 2.0 68 119
+ 2.42288 2 1 ack 40 ----- 0 2.0 0.0 68 139
- 2.42288 2 1 ack 40 ----- 0 2.0 0.0 68 139
r 2.42312 2 1 ack 40 ----- 0 2.0 0.0 56 127
+ 2.42312 1 0 ack 40 ----- 0 2.0 0.0 56 127
- 2.42312 1 0 ack 40 ----- 0 2.0 0.0 56 127
r 2.42704 1 2 tcp 1040 ----- 0 0.0 2.0 69 120
+ 2.42704 2 1 ack 40 ----- 0 2.0 0.0 69 140
- 2.42704 2 1 ack 40 ----- 0 2.0 0.0 69 140
r 2.42728 2 1 ack 40 ----- 0 2.0 0.0 57 128
+ 2.42728 1 0 ack 40 ----- 0 2.0 0.0 57 128
- 2.42728 1 0 ack 40 ----- 0 2.0 0.0 57 128
r 2.4312 1 2 tcp 1040 ----- 0 0.0 2.0 70 121

```

+ 2.4312 2 1 ack 40 ----- 0 2.0 0.0 70 141
- 2.4312 2 1 ack 40 ----- 0 2.0 0.0 70 141
r 2.43144 2 1 ack 40 ----- 0 2.0 0.0 58 129
+ 2.43144 1 0 ack 40 ----- 0 2.0 0.0 58 129
- 2.43144 1 0 ack 40 ----- 0 2.0 0.0 58 129
r 2.4356 2 1 ack 40 ----- 0 2.0 0.0 59 130
+ 2.4356 1 0 ack 40 ----- 0 2.0 0.0 59 130
- 2.4356 1 0 ack 40 ----- 0 2.0 0.0 59 130
r 2.43976 2 1 ack 40 ----- 0 2.0 0.0 60 131
+ 2.43976 1 0 ack 40 ----- 0 2.0 0.0 60 131
- 2.43976 1 0 ack 40 ----- 0 2.0 0.0 60 131
r 2.44392 2 1 ack 40 ----- 0 2.0 0.0 61 132
+ 2.44392 1 0 ack 40 ----- 0 2.0 0.0 61 132
- 2.44392 1 0 ack 40 ----- 0 2.0 0.0 61 132
r 2.44808 2 1 ack 40 ----- 0 2.0 0.0 62 133
+ 2.44808 1 0 ack 40 ----- 0 2.0 0.0 62 133
- 2.44808 1 0 ack 40 ----- 0 2.0 0.0 62 133
r 2.45224 2 1 ack 40 ----- 0 2.0 0.0 63 134
+ 2.45224 1 0 ack 40 ----- 0 2.0 0.0 63 134
- 2.45224 1 0 ack 40 ----- 0 2.0 0.0 63 134
r 2.45248 1 0 ack 40 ----- 0 2.0 0.0 51 122
+ 2.45248 0 1 tcp 1040 ----- 0 0.0 2.0 71 142
- 2.45248 0 1 tcp 1040 ----- 0 0.0 2.0 71 142
r 2.4564 2 1 ack 40 ----- 0 2.0 0.0 64 135
+ 2.4564 1 0 ack 40 ----- 0 2.0 0.0 64 135
- 2.4564 1 0 ack 40 ----- 0 2.0 0.0 64 135
r 2.45664 1 0 ack 40 ----- 0 2.0 0.0 52 123
+ 2.45664 0 1 tcp 1040 ----- 0 0.0 2.0 72 143
- 2.45664 0 1 tcp 1040 ----- 0 0.0 2.0 72 143
r 2.46056 2 1 ack 40 ----- 0 2.0 0.0 65 136
+ 2.46056 1 0 ack 40 ----- 0 2.0 0.0 65 136
- 2.46056 1 0 ack 40 ----- 0 2.0 0.0 65 136
r 2.4608 1 0 ack 40 ----- 0 2.0 0.0 53 124

+ 2.4608 0 1 tcp 1040 ----- 0 0.0 2.0 73 144
- 2.4608 0 1 tcp 1040 ----- 0 0.0 2.0 73 144
r 2.46472 2 1 ack 40 ----- 0 2.0 0.0 66 137
+ 2.46472 1 0 ack 40 ----- 0 2.0 0.0 66 137
- 2.46472 1 0 ack 40 ----- 0 2.0 0.0 66 137
r 2.46496 1 0 ack 40 ----- 0 2.0 0.0 54 125
+ 2.46496 0 1 tcp 1040 ----- 0 0.0 2.0 74 145
- 2.46496 0 1 tcp 1040 ----- 0 0.0 2.0 74 145
r 2.46888 2 1 ack 40 ----- 0 2.0 0.0 67 138
+ 2.46888 1 0 ack 40 ----- 0 2.0 0.0 67 138
- 2.46888 1 0 ack 40 ----- 0 2.0 0.0 67 138
r 2.46912 1 0 ack 40 ----- 0 2.0 0.0 55 126
+ 2.46912 0 1 tcp 1040 ----- 0 0.0 2.0 75 146
- 2.46912 0 1 tcp 1040 ----- 0 0.0 2.0 75 146
r 2.47304 2 1 ack 40 ----- 0 2.0 0.0 68 139
+ 2.47304 1 0 ack 40 ----- 0 2.0 0.0 68 139
- 2.47304 1 0 ack 40 ----- 0 2.0 0.0 68 139
r 2.47328 1 0 ack 40 ----- 0 2.0 0.0 56 127
+ 2.47328 0 1 tcp 1040 ----- 0 0.0 2.0 76 147
- 2.47328 0 1 tcp 1040 ----- 0 0.0 2.0 76 147
r 2.4772 2 1 ack 40 ----- 0 2.0 0.0 69 140
+ 2.4772 1 0 ack 40 ----- 0 2.0 0.0 69 140
- 2.4772 1 0 ack 40 ----- 0 2.0 0.0 69 140
r 2.47744 1 0 ack 40 ----- 0 2.0 0.0 57 128
+ 2.47744 0 1 tcp 1040 ----- 0 0.0 2.0 77 148
- 2.47744 0 1 tcp 1040 ----- 0 0.0 2.0 77 148
r 2.48136 2 1 ack 40 ----- 0 2.0 0.0 70 141
+ 2.48136 1 0 ack 40 ----- 0 2.0 0.0 70 141
- 2.48136 1 0 ack 40 ----- 0 2.0 0.0 70 141
r 2.4816 1 0 ack 40 ----- 0 2.0 0.0 58 129
+ 2.4816 0 1 tcp 1040 ----- 0 0.0 2.0 78 149
- 2.4816 0 1 tcp 1040 ----- 0 0.0 2.0 78 149
r 2.48576 1 0 ack 40 ----- 0 2.0 0.0 59 130

+ 2.48576 0 1 tcp 1040 ----- 0 0.0 2.0 79 150
- 2.48576 0 1 tcp 1040 ----- 0 0.0 2.0 79 150
r 2.48992 1 0 ack 40 ----- 0 2.0 0.0 60 131
+ 2.48992 0 1 tcp 1040 ----- 0 0.0 2.0 80 151
- 2.48992 0 1 tcp 1040 ----- 0 0.0 2.0 80 151
r 2.49408 1 0 ack 40 ----- 0 2.0 0.0 61 132
+ 2.49408 0 1 tcp 1040 ----- 0 0.0 2.0 81 152
- 2.49408 0 1 tcp 1040 ----- 0 0.0 2.0 81 152
r 2.49824 1 0 ack 40 ----- 0 2.0 0.0 62 133
+ 2.49824 0 1 tcp 1040 ----- 0 0.0 2.0 82 153
- 2.49824 0 1 tcp 1040 ----- 0 0.0 2.0 82 153
r 2.5024 1 0 ack 40 ----- 0 2.0 0.0 63 134

OUTPUT:

