Dewan VS Institute Of Engineering and Technology

N.H. 58, Delhi-Roorkee Highway, Baghpat Road Bypass Crossing,
Meerut-250005, U.P(India)

DEWAN VS GROUP OF
INSTITUTIONS INDIA

Department of Computer Science & Engineering

B. Tech (Session 2023-24)

Even-Semester

COMPUTER NETWORK LAB

(BCS-653)

L T P
0 0 2

DEWAN VS GROUP OF INSTITUTIONS INDIA

Department Of Computer Science & Engineering

Vision of the Institute

To be an outstanding institution in the country imparting technical education, providing need based, value based and career based programmes and producing self-reliant, self-sufficient technocrats, capable of meeting new challenges.

Mission of the Institute

To educate young aspirants in various technical fields to fulfill global requirement of human resources by providing sustainable quality education, training and invigorating environment, also molding them into skilled competent and socially responsible citizens who will lead the building of a powerful nation.

Vision of Department

To be an excellent department that imparts value based quality education and uplifts innovative research in the ever-changing field of technology.

Mission of Department

1. To fulfill the requirement of skilled human resources with focus on quality education.
2. To create globally competent and socially responsible technocrats by providing value and need based training.
3. To improve Industry-Institution Interaction and encourage the innovative research activities.

Program Educational Objectives

1. Students will have the successful careers in the field of computer science and allied sectors as an innovative engineer.
2. Students will continue to learn and advance their careers through participation in professional activities, attainment of professional certification and seeking advance studies.
3. Students will be able to demonstrate a commitment to life-long learning.
4. Students will be ready to serve society in any manner and become a responsible and aware citizen.
5. Establishing students in a leadership role in any field.

Program Outcomes

1. **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

2. **Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

3. **Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations

4. **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

5. **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

6. **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

7. **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

9. **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

10. **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

11. **Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

12. **Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Program Specific Outcomes

1. Ability to apply and analyze computational concepts in the areas related to algorithms, machine learning, cloud computing, web designing and web services.
2. Ability to apply standard practices and methodologies in software development and project management.
3. Ability to employ fundamental concepts and emerging technologies for innovative research activities, carrier opportunities & zeal for higher studies.

Course Outcomes

Subject Name: Computer Networks Lab **Subject code: BCS-653**

The students are expected to be able to demonstrate the following knowledge, skills and attitudes after completing this course:

S.N.		Course Outcomes
1.	To understand the basic concepts of network devices and connectivity	Understand
2.	To design and configure a network using Cisco Packet Tracer.	Apply
3.	To implement a client/server chatting program using socket programming.	Apply
4.	To analyze network traffic using Wire shark Tool.	Analyze

DEWAN VS INSTITIUTE OF ENGINEERING & TECHNOLOGY, MEERUT

COLLEGE CODE (311)

DEPARTMENT OF CSE

COMPUTER NETWORK LAB (BCS-653)

VI SEM

S. No	Name of the practical
1.	Running and using services / commands like ping, trace route, nslookup, arp etc. (Verify the connectivity of your workstation to the internet.)
2.	To study network CONNECTING DEVICES.
Write a C program:	
3.	For OSI model simulation.
4.	For ALOHA and Slotted ALOHA.
5.	To count Even and Odd Parity.
6.	For stuffing & De- stuffing of Bits.
7.	To implement Cyclic Redundancy Check (CRC).
8.	To implement RSA Algorithm (Encryption and Decryption).
9.	For IPv4, Implementation of decimal to binary and binary to decimal.
10.	For Leaky Bucket and Token Bucket.
Value Addition:	
11.	To analyze the packet transmission in the network by using Wireshark.
12.	Implementation of Socket programming using UDP and TCP.
13.	To build a network and analyzing the packet transmission by using packet tracer.

(Mr. Ashwani Chauhan)

Practical no. 1

Objective: Running and using services / commands like ping, trace route, nslookup, arp etc **. (Verify the connectivity of your workstation to the internet).**

Experiment

1. Verify the connectivity of your workstation to the internet.
2. Open the Command Prompt of the operating system using either of the following methods:

   Click on **Start > All Programs > Accessories > Command Prompt** or

   Click on **Start > Run**, enter **cmd** (short for command) and click on **ok**.

   A Command Prompt screen should open.
3. Gather TCP/IP configuration information: Type **ipconfig** (short for IP configuration) and press **Enter**. The screen will show the IP address, subnet mask, and default gateway for your computer's connection.

Notice the values in the Command Prompt. The IP address and the default gateway should be in the same network or subnet, otherwise this host would not be able to communicate outside the network. In figure 1, the subnet mask tells us that the first three octets of the IP address and the default gateway must be the same in order to be in the same network.

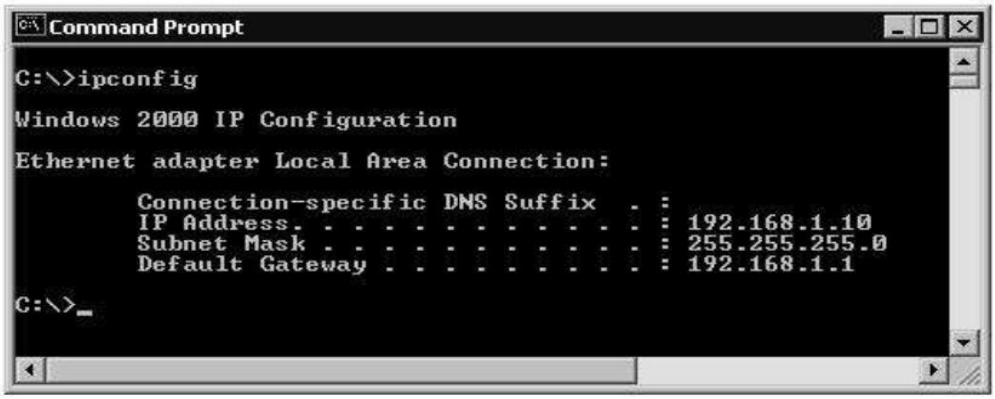


Fig. 1. The TCP/IP configuration information of a workstation

4. Check more detailed TCP/IP configuration information: Type **ip config /all** and press **Enter**. What are the DNS and DHCP server addresses? What are their functions? What is the MAC of the network interface card?

5. Ping the IP address of another computer. Note that for the ping and tracert commands to work the PC firewalls have to be disabled. Why do you think this is so? Ask the IP address of the workstation that is being used by another group of students. Then type **ping,** space, and the IP address that you received, then press **Enter**. Notice the outputs. Figure 2 shows a successful result of a ping to a given IP address.

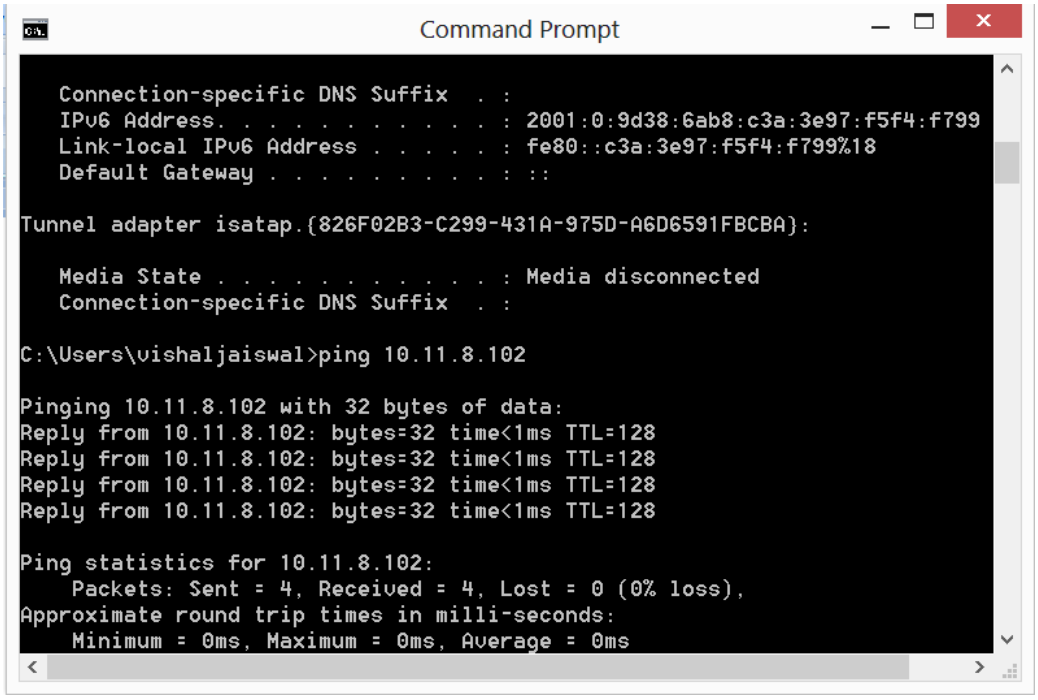


Fig. 2. A successful result of a ping to a certain IP address

6. Ping the IP address of the gateway router from the details that have been observed in the output of step 4 above. If the ping is successful, it means that there is a physical connectivity to the router on the local network and probably the rest of the world.

7. Ping the Loopback IP address of your computer. Type the following command: ping **127.0.0.1**. The IP address 127.0.0.1 is reserved for loopback testing. If the ping is successful, then TCP/IP is properly installed and functioning on this computer.

8. You can also ping using names like websites. Ping the IP address of the cisco website. Type **ping,** space and **www.cisco.com**, then press **Enter**. Notice the outputs. A DNS server will resolve the name to an IP address and the ping will be successful only in the existence of the DNS server.

9. Ping www.ee.uct.ac.za and observe the results. Is there a difference in time between the results shown by pinging www.cisco.com and www.ee.uct.ac.za. If so why and if not why?

10. Trace the route to the Cisco website. Type **tracert www.cisco.com** and press **enter**. In a successful output, you will see listings of all routers the tracert requests had to pass through to get to the destination.

11. Trace the route to the website of the Department of Electrical Engineering. Type **tracert www.ee.uct.ac.za** and press **enter**. The output should take less time than that of step 9. In the same way gmail.com can be traced out as shown in figure 3.

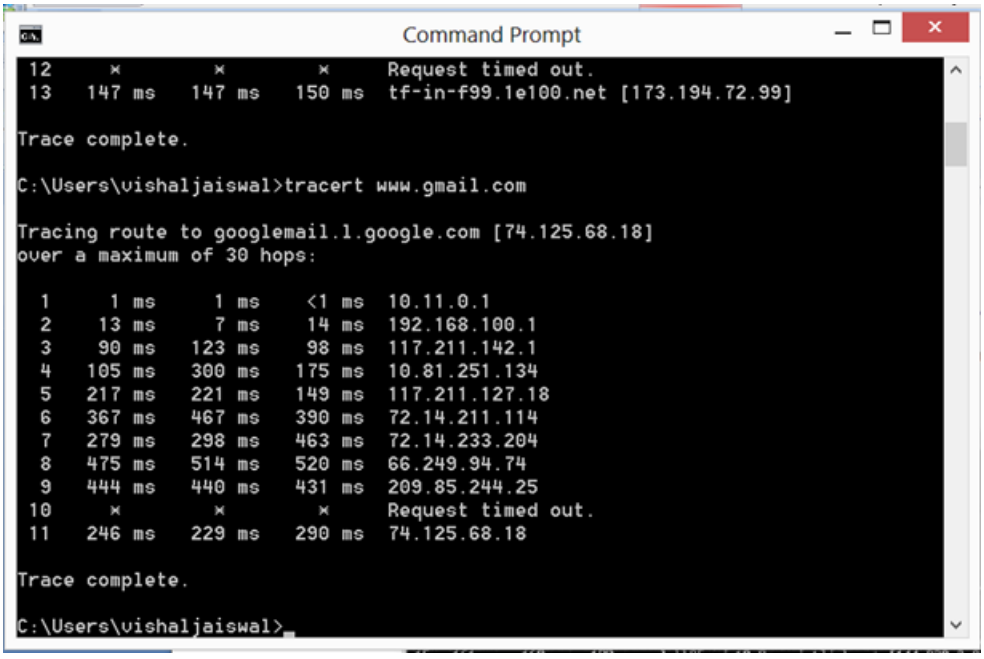


Fig. 3. A traceroute output

Practical no. 2

Objective: To study of Network CONNECTING DEVICES.

Passive Hubs

A passive hub is just a connector. It connects the wires coming from different branches. In a star-topology Ethernet LAN, a passive hub is just a point where the signals coming from different stations collide; the hub is the collision point. This type of a hub is part of the media; its location in the Internet model is below the physical layer.

Repeaters

A repeater is a device that operates only in the physical layer. Signals that carry information within a network can travel a fixed distance before attenuation endangers the integrity of the data. A repeater receives a signal and, before it becomes too weak or corrupted, regenerates the original bit pattern. The repeater then sends the refreshed signal as shown in figure 4.

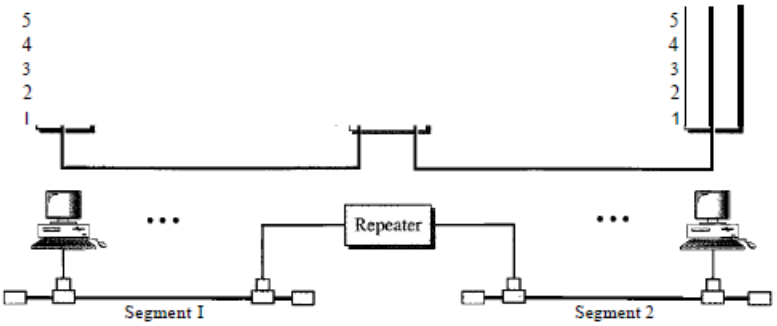


Fig. 4. A repeater connecting two segments of a LAN

A repeater does not actually connect two LANs; it connects two segments of the same LAN. The segments connected are still part of one single LAN. A repeater is not a device that can connect two LANs of different protocols.

- A repeater connects segments of a LAN.

- A repeater forwards every frame
- It has no filtering capability.
- A repeater is a regenerator, not an amplifier.

Active Hubs

An active hub is actually a multipart repeater. It is normally used to create connections between stations in a physical star topology. We have seen examples of hubs in some Ethernet implementations (lOBase-T, for example). However, hubs can also be used to create multiple levels of hierarchy, as shown in figure 5. The hierarchical use of hubs removes the length limitation of 10Base-T (100 m).

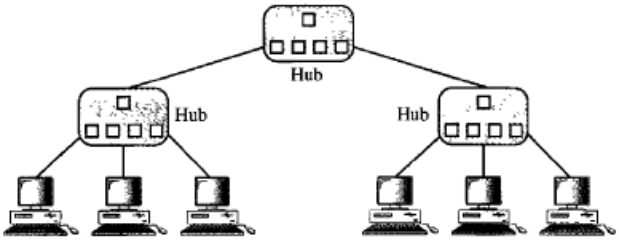


Fig. 5. Hub Connectivity

Bridges

A bridge operates in both the physical and the data link layer. As a physical layer device, it regenerates the signal it receives. As a data link layer device, the bridge can check the physical (MAC) addresses (source and destination) contained in the frame.

Transparent Bridges

A transparent bridge is a bridge in which the stations are completely unaware of the bridge's existence. If a bridge is added or deleted from the system, reconfiguration of the stations is unnecessary. According to the IEEE 802.1 d specification, a system equipped with transparent bridges must meet three criteria:

I. Frames must be forwarded from one station to another.

2. The forwarding table is automatically made by learning frame movements in the network.

3. Loops in the system must be prevented.

Two-Layer Switches

When we use the term *switch,* we must be careful because a switch can mean two different things. We must clarify the term by adding the level at which the device operates. We can have a two-layer switch or a three-layer switch. A **three-layer switch** is used at the network layer; it is a kind of router. The **two-layer switch** performs at the physical and data link layers.

A two-layer switch is a bridge, a bridge with many ports and a design that allows better (faster) performance. A bridge with a few ports can connect a few LANs together. A bridge with many ports may be able to allocate a unique port to each station, with each station on its own independent entity. This means no competing traffic (no collision, as we saw in Ethernet).

A two-layer switch, as a bridge does, makes a filtering decision based on the MAC address of the frame it received. However, a two-layer switch can be more sophisticated. It can have a buffer to hold the frames for processing. It can have a switching factor that forwards the frames faster. Some new two-layer switches, called *cut-through* switches, have been designed to forward the frame as soon as they check the MAC addresses in the header of the frame.

Routers

A router is a three-layer device that routes packets based on their logical addresses (host-to-host addressing). A router normally connects LANs and WANs in the Internet and has a routing table that is used for making decisions about the route. The routing tables are normally dynamic and are updated using routing protocols.

Three-Layer Switches

A three-layer switch is a router, but a faster and more sophisticated. The switching fabric in a three-layer switch allows faster table lookup and forwarding. In this book, we use the terms *router* and *three-layer switch* interchangeably.

Gateway

A gateway is normally a computer that operates in all five layers of the Internet or seven layers of OSI model. A gateway takes an application message, reads it, and interprets it. This means that it can be used as a connecting device between two internetworks that use different models. For example, a network designed to use the OSI model can be connected to another network using the Internet model. The gateway connecting the two systems can take a frame as it arrives from the first system, move it up to the OSI application layer, and remove the message.

Program No. 3

Objective: OSI model simulation.

PROGRAM DEFINITION: This is an open system interconnection program that transmit message from sender to receiver through server different layers.

PROGRAM DESCRIPTION:

The OSI Model deals with connecting open system. This model does not specify the exact services and protocols to use in each layer. Therefore, the OSI Model is not a network architecture. This model has seven layers. They are Physical layer, Data link layer, Presentation layer, Network layer, Session layer, Transport layer and Application layer. At sender side, each layer adds the header. The length of string i.e., number of bytes are not restricted up to Session layer.

ALGORITHM:

1. Read the input string and address.
2. Add application header.
3. Print the string.
4. Add the presentation layer header.
5. Print the string.
6. Add the Session layer header.
7. Print the string.
8. Add the Transport layer header.
9. Print the string.

10. Add the Network layer header.

11. Print the string.

12. Add the Data link layer header.

13. Print the string

14. Add the physical layer header.

15. Print the string.

INPUT: Enter the string: hai

OUTPUT: TRANSMITTER:

APPLICATION LAYER: AH hai

PRESENTATION LAYER: PHAH hai

SESSION LAYER: SHPHAH hai

TRANSPORT LAYER: THSHPHAH hai

NETWORK LAYER: NHTHSHPHAH hai

DATALINK LAYER: DHNHTHSHPHAH hai

MESSAGE ENTERED INTO PHYSICAL LAYER AND TRANSMITTED.

Program No. 4

Objective: Write C program to find the throughput of Pure Aloha and Slotted Aloha.

Problem Statement: C program to implement Pure Aloha

Problem Text: Write C program to find the throughput of Pure Aloha.

Input Description: Take input value of G (Channel load per packet time)

Output Description: Output will be a throughput of Pure Aloha.

Example: In Pure Aloha, Probability of successful transmission of data packet

$= G \times e^{-2G}$

Input: G=1/2

Output: S=0.184

Test Cases:

S.No	Input	Output
1.	G=1/2	S=0.184

Question-3

Problem Statement: C program to implement Slotted Aloha

Problem Text: Write C program to find the throughput of Slotted Aloha.

Input Description: Take input value of G (Channel load per packet time)

Output Description: Output will be a throughput of Slotted Aloha.

Example: In Slotted Aloha, Probability of successful transmission of data packet

$= G \times e^{-G}$

Input: G=1

Output: S=0.368

Test Cases:

S.No	Input	Output
1.	G=1	S=0.368

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

// Function to calculate the throughput of Pure Aloha
double pure_aloha_throughput(double G) {
   // Throughput S = G * e^(-2G)
   return G * exp(-2 * G);
}

// Function to calculate the throughput of Slotted Aloha
double slotted_aloha_throughput(double G) {
   // Throughput S = G * e^(-G)
   return G * exp(-G);
}

int main() {
   double G;

   // Get the offered load (G) from the user
   printf("Enter the average number of frames generated per frame transmission time (G): ");
   // The value of G should generally be between 0 and 1 for reasonable throughput,
   // although the formulas work for any non-negative G.
   if (scanf("%lf", &G) != 1 || G < 0) {
      printf("Invalid input. Please enter a non-negative number for G.\n");
      return 1;
```

```
    }

    // Calculate throughput for both protocols
    double S_pure = pure_aloha_throughput(G);
    double S_slotted = slotted_aloha_throughput(G);

    // Display the results
    printf("\n--- ALOHA Throughput Results (for G = %.4lf) ---\n", G);
    printf("Pure Aloha Throughput (S_pure): %.4lf (or %.2lf%%)\n", S_pure, S_pure * 100.0);
    printf("Slotted Aloha Throughput (S_slotted): %.4lf (or %.2lf%%)\n", S_slotted, S_slotted * 100.0);

    // Display maximum theoretical throughputs for context
    printf("\n--- Maximum Theoretical Throughputs ---\n");
    printf("Max Pure Aloha Throughput: 0.184 (18.4%%) at G = 0.5\n");
    printf("Max Slotted Aloha Throughput: 0.368 (36.8%%) at G = 1.0\n");

    return 0;
}
```

Program No. 5

Objective: Program to count Even and Odd Parity.

Parity: Parity of a number refers to whether it contains an odd or even number of 1-bits. The number has "odd parity", if it contains odd number of 1-bits and is "even parity" if it contains even number of 1-bits.
Main idea of the below solution is – Loop while n is not 0 and in loop unset one of the set bits and invert parity.C PROGRAM

Algorithm: getParity(n)
1. Initialize parity = 0
2. Loop while n != 0
a. Invert parity
parity = !parity
b. Unset rightmost set bit
n = n & (n-1)
3. return parity

Example:
Initialize: n = 13 (1101) parity = 0

n = 13 & 12 = 12 (1100) parity = 1
n = 12 & 11 = 8 (1000) parity = 0
n = 8 & 7 = 0 (0000) parity = 1

C PROGRAM

```
#include <stdio.h>

int main() {
    int num, count = 0;
    printf("Enter an integer: ");
    scanf("%d", &num);

    int temp = num;
    while (temp > 0) {
        if (temp & 1) // Check if the last bit is 1
            count++;
        temp >>= 1; // Right shift to check the next bit
    }

    printf("Number of 1s: %d\n", count);
    if (count % 2 == 0)
        printf("Parity: Even\n");
    else
        printf("Parity: Odd\n");

    return 0;
}
```

Program No. 6

Objective: Program for stuffing & De- stuffing of Bits.

(1) Write a program to implement bit stuffing & De-stuffing.

(2) Write a program to implement character stuffing & De-stuffing.

(1) Write a program to implement bit stuffing & De-stuffing.

Resources: Turbo C, C++.

Bit Stuffing and Destuffing

- Include <iostream.h>,<conio.h>,<io.h> files both in transmitter & receiver programs.
- During the transmission, attach a flag pattern (01111110) at the start & end of data unit.
- If transmitter sees five consecutive one's in data, it stuffs zero bit in data.
- At the receiving end, whenever in data it finds five consecutive one's and the next bit are zero then the receiver will de stuff that zero bit. e.g. If the Pattern to be transmitted is 00011110111110000, then at the transmitter side will be 000111101111100000 because as 5 consecutive 1's are detected, one 0 should be stuffed and at the receiver side again as it will detect 0 after 5 consecutive 1's , it will de-stuff it.

C PROGRAM

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

void performBitStuffing(char* data, char* stuffedData) {
   int i = 0, j = 0, count = 0;
   int len = strlen(data);

   while (i < len) {
      if (data[i] == '1') {
         count++;
         stuffedData[j++] = data[i];
      } else {
         stuffedData[j++] = data[i];
         count = 0; // Reset count if '0' is encountered
```

```
        }

        // If 5 consecutive 1s are found, stuff a '0'
        if (count == 5) {
            stuffedData[j++] = '0';
            count = 0; // Reset counter
        }
        i++;
    }
    stuffedData[j] = '\0'; // Null terminate the new string
}

void performBitDestuffing(char* stuffedData, char* destuffedData) {
    int i = 0, j = 0, count = 0;
    int len = strlen(stuffedData);

    while (i < len) {
        if (stuffedData[i] == '1') {
            count++;
            destuffedData[j++] = stuffedData[i];
        } else {
            destuffedData[j++] = stuffedData[i];
            count = 0; // Reset count if '0' is encountered
        }

        // If 5 consecutive 1s are followed by a 0, skip the 0
        if (count == 5) {
            if (i + 1 < len && stuffedData[i + 1] == '0') {
                i++; // Skip the stuffed bit
                count = 0; // Reset counter
            }
        }
        i++;
    }
    destuffedData[j] = '\0';
}
```

```
int main() {
    char data[100];
    char stuffedData[200];
    char destuffedData[100];

    printf("Enter the data bit stream (0s and 1s): ");
    scanf("%s", data);

    // 1. Bit Stuffing
    performBitStuffing(data, stuffedData);
    printf("\nStuffed Data: %s", stuffedData);

    // 2. Bit Destuffing
    performBitDestuffing(stuffedData, destuffedData);
    printf("\nDestuffed Data: %s", destuffedData);

    printf("\n");
    return 0;
}
```

(2) Write a program to implement character stuffing & De-stuffing.

Resources: Turbo C, C++.

Character Stuffing and Destuffing

- Include <iostream.h>,<conio.h>,<io.h> files both in transmitter & receiver programs.
- This is type of Framing Method.
- During the transmission attach a ASCII Code pattern DLE STX at the start & DLE ETX end of data Unit.
- If transmitter sees DLE stuff another DLE text in data.
- At the receiving end, whenever the data it finds five consecutive DLE then receiver will destuff One DLE.

C PROGRAM

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

```
#include <stdlib.h>

void performStuffing(char *data, char *stuffedData) {
    char *dle = "DLE";
    char *stx = "STX";
    char *etx = "ETX";

    int j = 0;
    // Add Start of Frame
    strcpy(stuffedData, "DLESTX");
    j = 6;

    for (int i = 0; i < strlen(data); i++) {
        // If 'D', 'L', 'E' is found, stuff another 'DLE'
        if (data[i] == 'D' && data[i+1] == 'L' && data[i+2] == 'E') {
            strcat(stuffedData, "DLEDLE");
            j += 6;
            i += 2; // Skip 'DLE'
        } else {
            stuffedData[j++] = data[i];
        }
    }
    // Add End of Frame
    strcat(stuffedData, "DLEETX");
}

void performDestuffing(char *stuffedData, char *destuffedData) {
    int j = 0;
    // Skip the first 6 characters (DLESTX) and last 6 characters (DLEETX)
    for (int i = 6; i < strlen(stuffedData) - 6; i++) {
        if (stuffedData[i] == 'D' && stuffedData[i+1] == 'L' && stuffedData[i+2] == 'E' &&
            stuffedData[i+3] == 'D' && stuffedData[i+4] == 'L' && stuffedData[i+5] == 'E') {
            // Remove one stuffed DLE
            destuffedData[j++] = 'D';
            destuffedData[j++] = 'L';
            destuffedData[j++] = 'E';
            i += 5; // Skip extra DLE characters
```

```
        } else {
            destuffedData[j++] = stuffedData[i];
        }
    }
    destuffedData[j] = '\0';
}

int main() {
    char data[100];
    char stuffedData[200];
    char destuffedData[100];

    printf("Enter the data (e.g., ABcDLEEF): ");
    scanf("%s", data);

    // 1. Stuffing
    performStuffing(data, stuffedData);
    printf("\nData after stuffing: %s\n", stuffedData);

    // 2. De-stuffing
    performDestuffing(stuffedData, destuffedData);
    printf("Data after destuffing: %s\n", destuffedData);

    return 0;
}
```

Program No. 7

Objective: Program to implement Cyclic Redundancy Check CRC.

CRC or Cyclic Redundancy Check is a method of detecting accidental changes/errors in communication channel.

CRC uses **Generator Polynomial** which is available on both sender and receiver side. An example generator polynomial is of the form like $x^3 + x + 1$. This generator polynomial represents key 1011. Another example is $x^2 + 1$ that represents key 101.

Sender Side (Generation of Encoded Data from Data and Generator Polynomial (or Key)):

1. The binary data is first augmented by adding k-1 zeros in the end of the data
2. Use ***modulo-2 binary division*** to divide binary data by the key and store remainder of division.
3. Append the remainder at the end of the data to form the encoded data and send the same

.

Receiver Side (Check if there are errors introduced in transmission) Perform modulo-2 division again and if remainder is 0, then there are no errors. In this we will focus only on finding the remainder i.e. check word and the code word.

Modulo 2 Division:

The process of modulo-2 binary division is the same as the familiar division process we use for decimal numbers. Just that instead of subtraction, we use XOR here.

- In each step, a copy of the divisor (or data) is XORed with the k bits of the dividend (or key).
- The result of the XOR operation (remainder) is (n-1) bits, which is used for the next step after 1 extra bit is pulled down to make it n bits long.
- When there are no bits left to pull down, we have a result. The (n-1)-bit remainder which is appended at the sender side.

Illustration:

Example 1 (No error in transmission):

Data word to be sent - 100100

Key - 1101 [Or generator polynomial x3 + x2 + 1]

Sender Side:

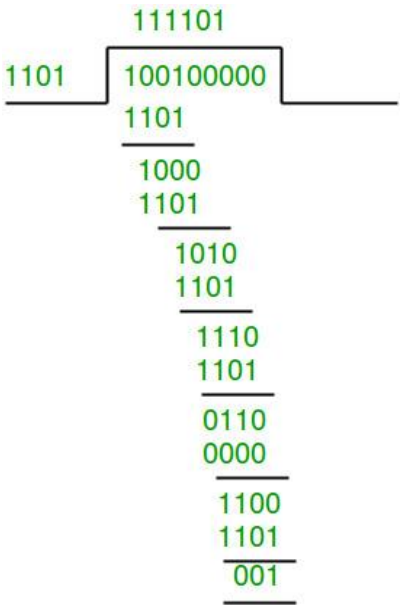
Therefore, the remainder is 001 and hence the encoded data sent is 100100001.

Receiver Side:

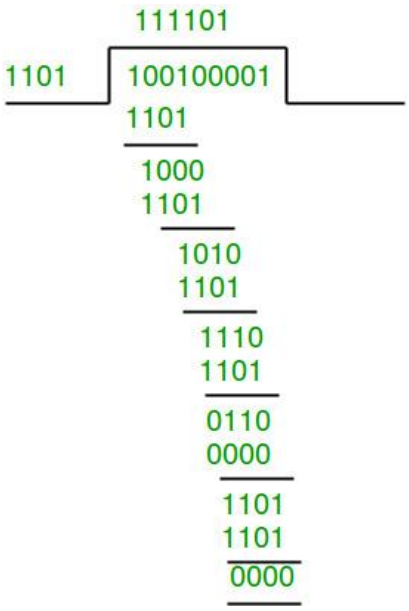
Code word received at the receiver side 100100001

Therefore, the remainder is all zeros. Hence, the data received has no error.

Sender Side:



Receiver Side:



Example 2: (Error in transmission)

Data word to be sent - 100100

Key - 1101

Sender Side:

Therefore, the remainder is 001 and hence the

code word sent is 100100001.

Receiver Side

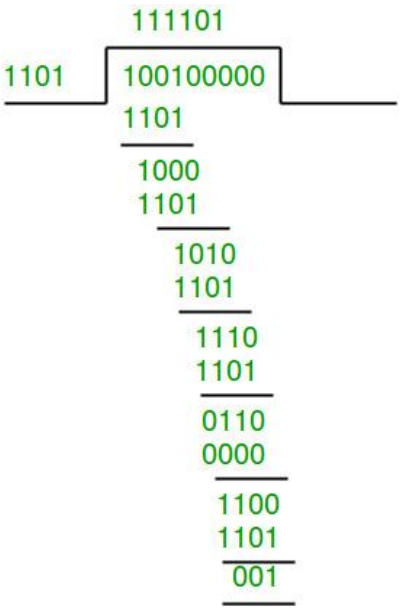
Let there be error in transmission media

Code word received at the receiver side - 100000001

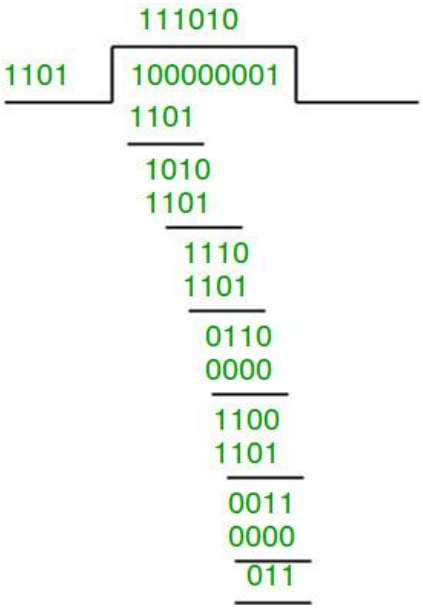
Since the remainder is not all zeroes, the error

is detected at the receiver side.

Sender Side:



Receiver Side:



C PROGRAM

```
# Function to perform XOR operation on two binary strings
def xor(a, b):
    result = []
    for i in range(1, len(b)):
        if a[i] == b[i]:
            result.append('0')
        else:
```

```
            result.append('1')
    return ''.join(result)

# Function to perform Modulo-2 division
def mod2div(dividend, divisor):
    pick = len(divisor)
    tmp = dividend[0:pick]
    while pick < len(dividend):
        if tmp[0] == '1':
            # XOR with the divisor and append the next bit
            tmp = xor(divisor, tmp) + dividend[pick]
        else:
            # XOR with a string of zeros and append the next bit
            tmp = xor('0'*pick, tmp) + dividend[pick]
        pick += 1

    # Perform the final XOR operation
    if tmp[0] == '1':
        tmp = xor(divisor, tmp)
    else:
        tmp = xor('0'*pick, tmp)

    checkword = tmp
    return checkword

# Function to encode the data with CRC
def encodeData(data, key):
    l_key = len(key)
    # Append n-1 zeros to the data, where n is the key length
    appended_data = data + '0' * (l_key - 1)
    remainder = mod2div(appended_data, key)
    # The codeword is the original data + remainder (CRC bits)
    codeword = data + remainder

    print(f"Remainder (CRC bits): {remainder}")
    print(f"Encoded Data (Data + Remainder): {codeword}")
    return codeword
```

```
# Function to check for errors at the receiver's end
def check_data(received_data, key):
    # Perform modulo-2 division on the entire received data
    remainder = mod2div(received_data, key)
    # If the remainder is all zeros, no error is detected
    if '1' not in remainder:
        print("No error in data transmission has occurred")
        return True
    else:
        print("Error in data transmission has occurred")
        return False

# --- Example Usage ---
data = input("Enter the input message in binary: ")
key = input("Enter the generator polynomial in binary: ")

print("\n--- Sender Side ---")
encoded_message = encodeData(data, key)

print("\n--- Receiver Side ---")
# Simulate reception (can introduce an error here for testing)
# Example of introducing an error: change a bit
# received_message = list(encoded_message)
# received_message[2] = '0' if received_message[2] == '1' else '1'
# received_message = "".join(received_message)
received_message = encoded_message # No error for this example

print(f"Received message: {received_message}")
check_data(received_message, key)
```

Program No. 8

Objective: C code to implement RSA Algorithm (Encryption and Decryption)

RSA Algorithm in Cryptography

RSA algorithm is asymmetric cryptography algorithm. Asymmetric actually means that it works on two different keys i.e. **Public Key** and **Private Key.** As the name describes that the Public Key is given to everyone and Private key is kept private.

An example of asymmetric cryptography:

1. A client (for example browser) sends its public key to the server and requests for some data.
2. The server encrypts the data using client's public key and sends the encrypted data.
3. Client receives this data and decrypts it.

Since this is asymmetric, nobody else except browser can decrypt the data even if a third party has public key of browser.

The idea! The idea of RSA is based on the fact that it is difficult to factorize a large integer. The public key consists of two numbers where one number is multiplication of two large prime numbers. And private key is also derived from the same two prime numbers. So if somebody can factorize the large number, the private key is compromised. Therefore encryption strength totally lies on the key size and if we double or triple the key size, the strength of encryption increases exponentially. RSA keys can be typically 1024 or 2048 bits long, but experts believe that 1024 bit keys could be broken in the near future. But till now it seems to be an infeasible task.

Key Generation

Select p, q	p and q both prime, $p \neq q$
Calculate $n = p \times q$	
Calculate $\phi(n) = (p-1)(q-1)$	
Select integer e	$\gcd(\phi(n), e) = 1;\ 1 < e < \phi(n)$
Calculate d	$d = e^{-1} \pmod{\phi(n)}$
Public key	$PU = \{e, n\}$
Private key	$PR = \{d, n\}$

Encryption	
Plaintext:	$M < n$
Ciphertext:	$C = M^e \bmod n$

Decryption	
Ciphertext:	C
Plaintext:	$M = C^d \bmod n$

RSA Example - Key Setup

1. Select primes: $p = 17$; $q = 11$
2. Calculate $n = pq = 17 \times 11 = 187$
3. Calculate $\phi(n) = (p-1)(q-1) = 16 \times 10 = 160$
4. Select e: $GCD(e, 160) = 1$; choose $e = 7$
5. Derive d: $de = 1 \bmod 160$ and $d < 160$
   Get $d = 23$ since $23 \times 7 = 161 = 10 \times 160 + 1$
6. Publish public key: $PU = \{7, 187\}$
7. Keep private key secret: $PR = \{23, 187\}$

RSA Example - En/Decryption

- sample RSA encryption/decryption is:
- given message $M = 88$ (nb. $88 < 187$)
- encryption:
  $C = 88^7 \bmod 187 = 11$
- decryption:
  $M = 11^{23} \bmod 187 = 88$

C PROGRAM

```
#include <math.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>


// Helper functions for prime checking, GCD, and modular exponentiation

int isPrime(int n) { /* ... */ }
```

```
int gcd(int a, int b) { /* ... */ }
long pomod(long a, long b, long m) { /* ... */ }

// Finds private key d, where (d * e) % lambda_n == 1
long private_key(long e, long lambda_n) { /* ... */ }

// Encryption: C = M^e mod n
char *encrypt(char *message, long e, long n) { /* ... */ }

// Decryption: M = C^d mod n
char *decrypt(char *cipher, long d, long n) { /* ... */ }

int main() {
    int p = 61, q = 53; // Example small primes
    long n = p * q;
    long lambda_n = (p - 1) * (q - 1);
    long e = 17; // Commonly used public exponent
    long d = private_key(e, lambda_n);
    char message[] = "HELLO";
    char *cipher = encrypt(message, e, n);
    char *decrypted = decrypt(cipher, d, n);
    printf("Encrypted: %s\n", cipher);
    printf("Decrypted: %s\n", decrypted);
      free(cipher); free(decrypted);
    return 0;
}
```

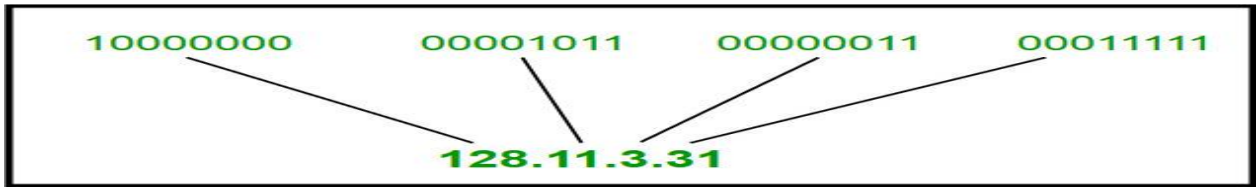
Program No.9

Objective:
Write a C program for IPV4, Implementation of decimal to binary, and Implementation of binary to decimal.

Resources: Turbo C, C++.

IP Addressing :- Introduction and Classful Addressing

IP address is an address having information about how to reach a specific host, especially outside the LAN. An IP address is a 32 bit unique address having an address space of 2^{32}. Generally, there are two notations in which IP address is written, dotted decimal notation and hexadecimal notation. Dotted Decimal Notation



Hexadecimal Notation

Some points to be noted about dotted decimal notation:

1. The value of any segment (byte) is between 0 and 255 (both included).
2. There are no zeroes preceding the value in any segment (054 is wrong, 54 is correct).

Classful Addressing

The 32 bit IP address is divided into five sub-classes which is depicted in figure 6. These are:

- Class A
- Class B
- Class C
- Class D
- Class E

Each of these classes has a valid range of IP addresses. Classes D and E are reserved for multicast and experimental purposes respectively. The order of bits in the first octet determine the classes of IP address.

IPv4 address is divided into two parts as shown in figure 7:

- **Network ID**
- **Host ID**

The class of IP address is used to determine the bits used for network ID and host ID and the number of total networks and hosts possible in that particular class. Each ISP or network administrator assigns IP address to each device that is connected to its network.

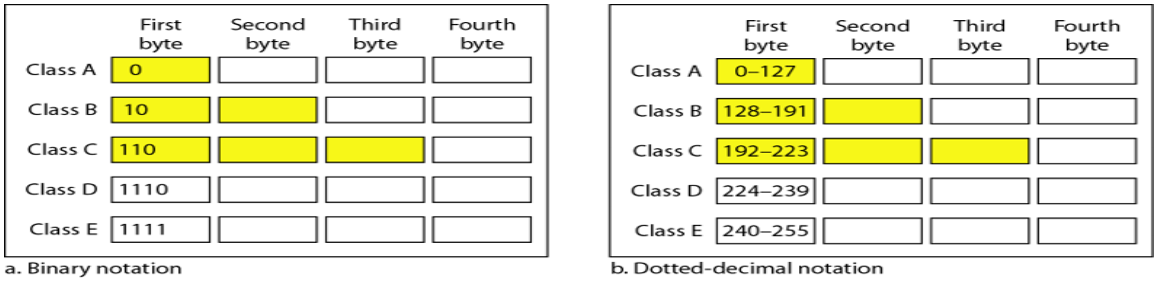


Fig. 6. Different classes of IPv4 binary and decimal notation.

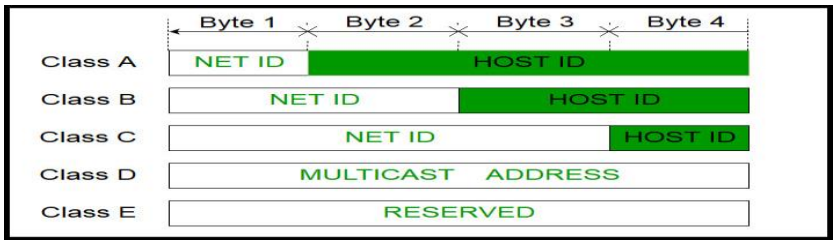


Fig. 7. Net ID and Host ID of IPV6

Program No.10

Leaky bucket algorithm

In the network layer, before the network can make Quality of service guarantees, it must know what traffic is being guaranteed. One of the main causes of congestion is that traffic is often bursty.

To understand this concept first we have to know little about traffic shaping. **Traffic Shaping** is a mechanism to control the amount and the rate of the traffic sent to the network. Approach of congestion management is called Traffic shaping. Traffic shaping helps to regulate rate of data transmission and reduces congestion.

There are 2 types of traffic shaping algorithms:

1. Leaky Bucket
2. Token Bucket

Suppose we have a bucket in which we are pouring water in a random order but we have to get water in a fixed rate, for this we will make a hole at the bottom of the bucket. It will ensure that water coming out is in a some fixed rate, and also if bucket will full we will stop pouring in it.

The input rate can vary, but the output rate remains constant. Similarly, in networking, a technique called leaky bucket can smooth out bursty traffic. Bursty chunks are stored in the bucket and sent out at an average rate.

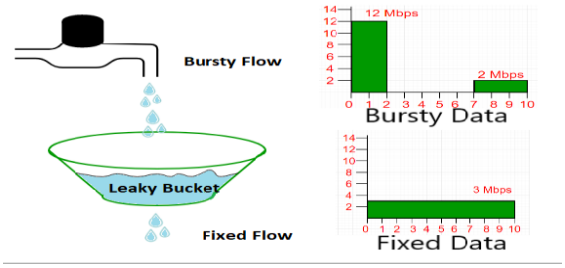


Fig. 7. Leaky bucket for shaping the traffic

In the figure, we assume that the network has committed a bandwidth of 3 Mbps for a host. The use of the leaky bucket shapes the input traffic to make it conform to this commitment. In figure 7 the host sends a burst of data at a rate of 12 Mbps for 2 s, for a total of 24 Mbits of data. The host is silent for 5 s and then sends data at a rate of 2 Mbps for 3 s, for a total of 6 Mbits of data. In all, the host has sent 30 Mbits of data in 10 s. The leaky bucket smooths the traffic by sending out data at a rate of 3 Mbps during the same 10 s.

Without the leaky bucket, the beginning burst may have hurt the network by consuming more bandwidth than is set aside for this host. We can also see that the leaky bucket may prevent congestion.

A simple leaky bucket algorithm can be implemented using FIFO queue. A FIFO queue holds the packets. If the traffic consists of fixed-size packets (e.g., cells in ATM networks), the process removes a fixed number of packets from the queue at each tick of the clock. If the traffic consists of variable-length packets, the fixed output rate must be based on the number of bytes or bits.

The following is an algorithm for variable-length packets:

1. Initialize a counter to n at the tick of the clock.
2. If n is greater than the size of the packet, send the packet and decrement the counter by the packet size. Repeat this step until n is smaller than the packet size.
3. Reset the counter and go to step 1.

Example – Let n=1000

Packet=

200	700	500	450	400	200

Since n> front of Queue i.e. n>200

Therefore, n=1000-200=800

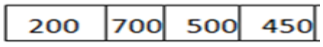
Packet size of 200 is sent to the network.

200	700	500	450	400

Now Again n>front of the queue i.e. n > 400

Therefore, n=800-400=400

Packet size of 400 is sent to the network.



Since n< front of queue

Therefore, the procedure is stop.

Initialize n=1000 on another tick of clock.

This procedure is repeated until all the packets are sent to the network.

Program No.11

OBJECTIVE: To implement a network using Cisco Packet Tracer.

THEORY: Packet Tracer (PT) is a powerful and dynamic tool that displays the various protocols used in networking, in either Real Time or Simulation mode. This includes layer 2 protocols such as Ethernet and PPP, layer 3 protocols such as IP, ICMP, and ARP, and layer 4 protocols such as TCP and UDP. Routing protocols can also be traced.

Steps to simulate a network:

Step 1: Start Packet Tracer You will see the start screen as shown below.

Step 2: Choose "Hub" and then select "Generic"

Step 3: After selecting "Generic" click on the main area. You will see a Hub.

Step 4: Select "End Devices" and then click at "Generic" Choosing Devices and Connections We will begin building our network topology by selecting devices and the media in which to connect them. Several types of devices and network connections can be used.

Step 6: Select "Connections" from Power Cycle Devices and click on "Automatically choose Connection Type"

Step 7: Draw connections from Hub to PCs

Step 8: Double click on a PC, a box will appear. Click on the "Desktop" tab.

Step 9: Then select "IP configuration"

Step 10: Write the IP address of your network and click at the Subnet mask filed. Subnet Mask will appear automatically.

Step11: Repeat Step 10 to set the IPs for all the PCs.

Step 12: Select "Add simple message"

Step 13: Drag and Drop the message to the source device and then to the Destination device In this case my source device is PC1 and destination device is PC4.

Step 14: Select the Simulation Mode at the bottom right corner.

Step15: Click at “Auto Capture / Play” Conclusion: Connection established successfully between Source and Destination.

Step 16: Observe the path of the Message from source to Hub, then to all devices. And then from Destination to Hub then back to the source.

Step 17: Finally observe the marks. If the source PC is marked correct it means you have successfully established the connection.

Some of the steps are shown in figures 11.1 to 11.15

Screenshots:

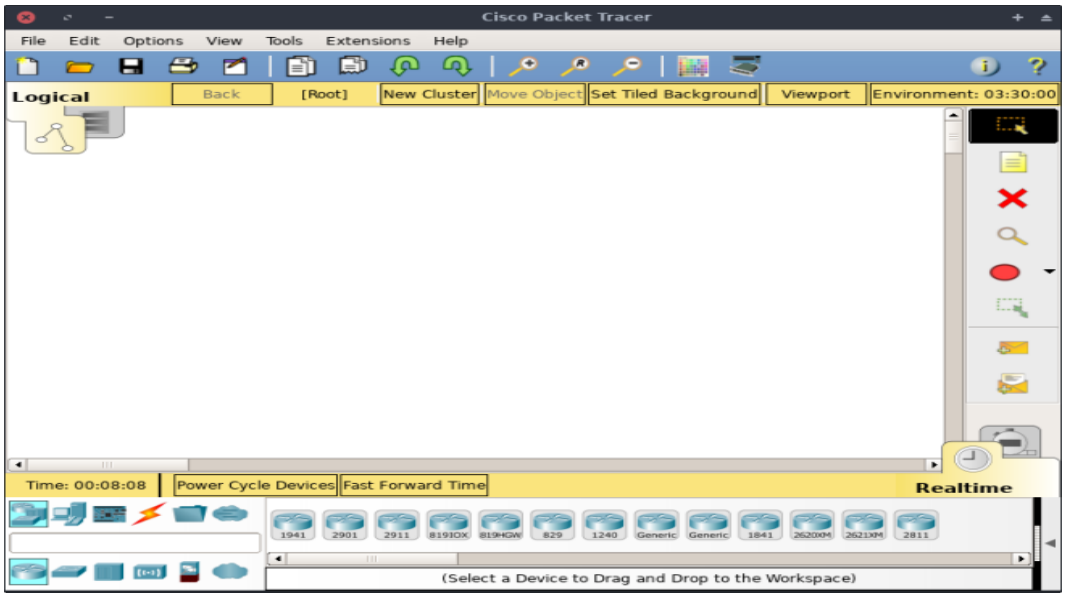


Fig. 11.1 Step 1

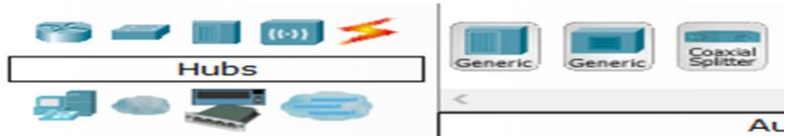


Fig. 11.2. Step 2

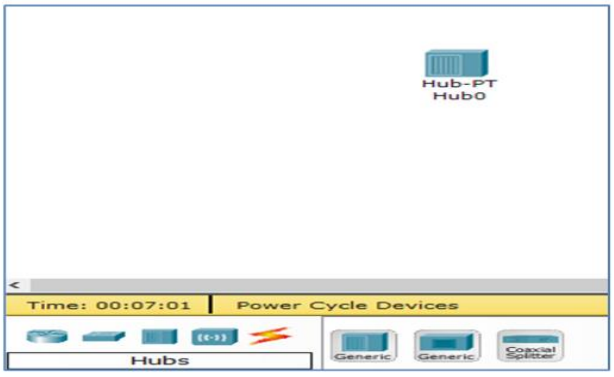


Fig. 11.3. Step 3

Fig. 11.4. Step 4

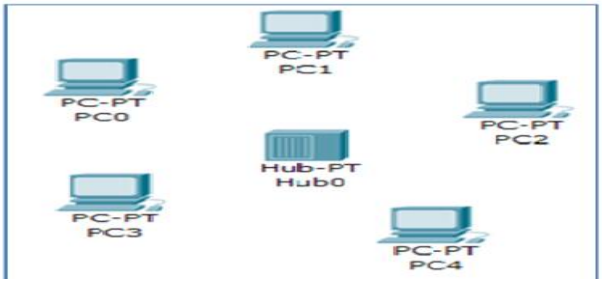


Fig. 11.5. Step 5

Fig. 11.6. Step 6

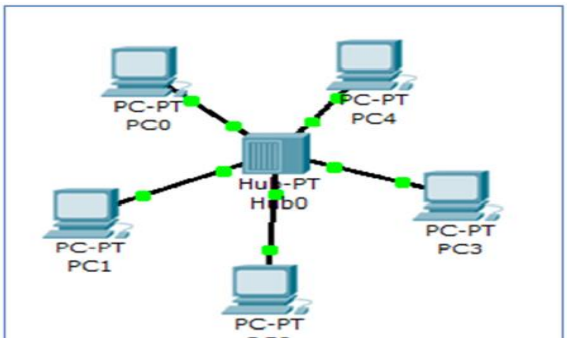


Fig. 11.7. Step 7

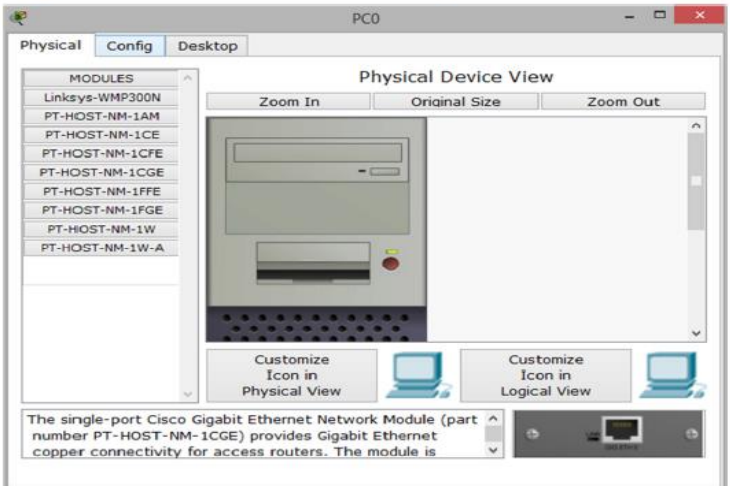


Fig. 11.8. Step 8

Fig. 11.9. Step 9

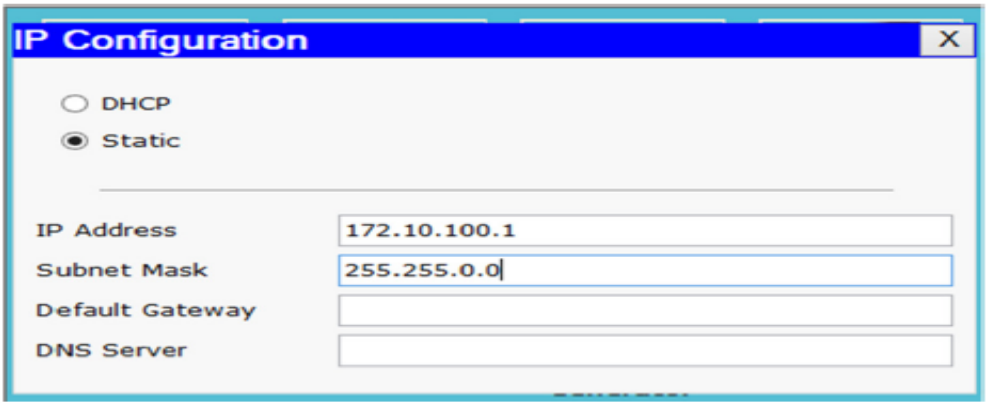


Fig. 11.10. Step 10

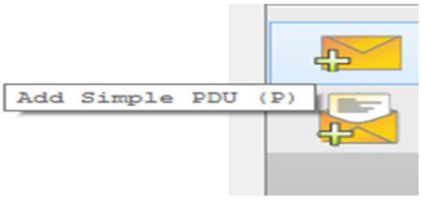


Fig. 11.11. Step 11

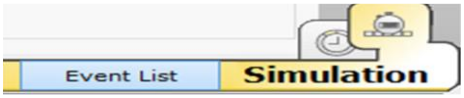


Fig. 11.12. Step 12

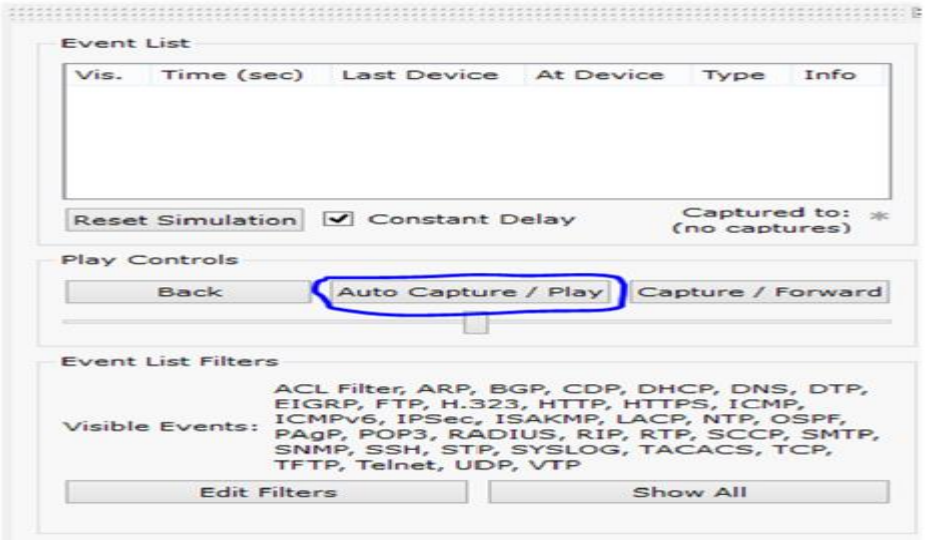


Fig. 11.13. Step 13

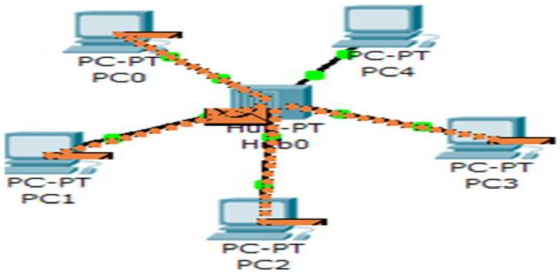


Fig. 11.14. Step 14

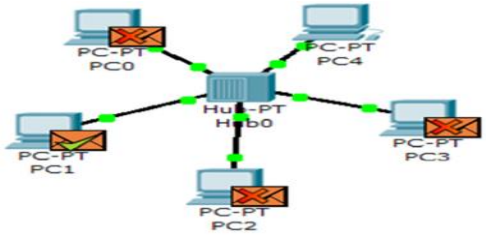


Fig. 11.15. Step 15

Program No.12

OBJECTIVE: To Study packet's information through Wire shark Simulator.

THEORY

Wireshark is a tool that allows packet traces to be sniffed, captured and analyzed. Before Wireshark (or in general, any packet capture tool) is used, careful consideration should be given to where in the network packets are to be captured. Refer to the capture setup pages in the wireshark.org wiki for technical details on various deployment scenarios. If it is unclear which deployment scenario should be used to capture traces for a particular problem, consider opening a service request with Novell Technical Services for assistance.

Obtain appropriate Wireshark package

Obtain a Wireshark package or installer for the operating system running on the system which is to be used for packet capture.

Wireshark is included in Novell's SUSE Linux products (for some products, under its old name, Ethereal). For other platforms, download a binary or installer from http://www.wireshark.org. With installers, ensure all product components are selected for installation.

Start Wire shark

Start Wire shark. On a Linux or Unix environment, select the Wireshark or Ethereal entry in the desktop environment's menu, or run "wire shark" (or "ethereal") from a root shell in a terminal emulator. In a Microsoft Windows environment, launch wireshark.exe from C:\Program Files\Wire shark.

Note that on Un*x systems, a non-GUI version of Wire shark called "tshark" (or "tethereal") may be available as well, but its use is beyond the scope of this document.

Configure Wire shark

After starting Wire shark, do the following:

1. Select **Capture | Interfaces**
2. Select the interface on which packets need to be captured.
3. If capture options need to be configured, click the **Options** button for the chosen interface. Note the following recommendations for traces that are to be analysed by Novell Technical Services:

- **Capture packet in promiscuous mode:** This option allows the adapter to capture all traffic not just traffic destined for this workstation. It should be enabled.
- **Limit each packet to:** Leave this option unset. Novell Support will always want to see full frames.
- **Filters:** Generally, Novell Support prefers an unfiltered trace. For documentation on filters, please refer to TID 10084702 - How to configure a capture filter for Ethereal.
- **Capture file(s):** This allows a file to be specified to be used for the packet capture. By default Wireshark will use temporary files and memory to capture traffic. Specify a file for reliability.
- **Use multiple files, Ring buffer with:** These options should be used when Wireshark needs to be left running capturing data data for a long period of time. The number of files is configurable. When a file fills up, it it will wrap to the next file. The file name should be specified if the ring buffer is to be used.
- **Stop capture after xxx packet(s) captured:** Novell Technical Support would most likely never use this option. Leave disabled.
- **Stop capture after xxx kilobyte(s) captured:** Novell Technical Support would most likely never use this option. Leave disabled.
- **Stop capture after xxx second(s):** Novell Technical Support would most likely never use this option. Leave disabled.
- **Update list of packets in real time:** Disable this option if the problem that's being investigated is occuring on the same workstation as where Wireshark is running.
- **Automatic scrolling in live capture:** Wireshark will scroll the window so that the most current packet is displayed.

- **Hide capture info dialog:** Disable this option so that you can view the count of packets being captured for each protocol.
- **Enable MAC name resolution:** Wireshark contains a table to resolve MAC addresses to vendors. Leave enabled.
- **Enable network name resolution:** Wireshark will issue DNS queries to resolve IP host names. Also will attempt to resolve network names for other protocols. Leave disabled.
- **Enable transport name resolution:** Wireshark will attempt to resolve transport names. Leave disabled.

4. Now click the **Start** button to start the capture.
5. Recreate the problem. The capture dialog should show the number of packets increasing. If not, then stop the capture. Examine the interface list and pick the one that is not associated with the WANIP. It will probably be a long alpha-numeric string. If packets are still not being captured, try removing any filters that have been defined.
6. Once the problem which is to be analyzed has been reproduced, click on **Stop**. It might take a few seconds for Wireshark to display the packets captured. If the destination address is always displayed as FFFFFFFF (IPX) or always ends in .255 (IP) then all that has been captured is broadcast traffic. **This is a useless trace.** This usually occurs when another machine is being traced (to start the trace while the target machine is powered off, in order to capture the bootup process). The capture setup needs to be reconsidered - port mirroring on the switch may need to be set up, or a dumb hub may need to be used to make the traffic reach the sniffing system. (Some devices advertised as "hubs" are in fact switches that may have the intelligence to prevent the workstations from seeing each other's packets; with these, getting a good trace may not be possible)

   The Wireshark website has a good FAQ on this subject. Please refer to http://www.wireshark.org/faq.html#q7.1

7. Save the packet trace in any supported format. Just click on the **File** menu option and select **Save As**. By default Wireshark will save the packet trace in libpcap

format. This is a filename with a.pcap extension. Use this default for files sent to Novell.

8. Create a trace_info.txt file with the IP and MAC address of the machines that are being traced as well as any pertinent information, such as:

- What is the problem? (when did it start? steps to reproduce? any other pertinent information)
- What steps were traced?
- Give names of the servers and files being accessed.
- If analysis of the trace has already been attempted, please provide Novell Support with analysis notes.

  For example: Packets 1-30 are boot. Packets 31-500 are login. Packets 501 to 1,000 is my application loading. Packet 1,001 to 1,500 is me saving my file. The error occurred at approximately packet 1,480.
- Give the MAC addresses of hardware involved? (Workstation, servers, printers ...)
- What is the workstation OS and configuration?
- What version of client software is running?
- If it works with one version of the client (or a particular server patch), then get a trace of it working, and a trace of it not working.
- For Novell Client issues: Are there any client patches loaded?
- For Novell servers: What version of NetWare/OES (and other relevant products i.e. ZEN or NDPS) are running on the server?
- What patches have been applied?
- What is the configuration of the network? Are there routers involved? If so, what kind of routers?

Assignment Questions:

Part 1

1. Is your browser running HTTP version 1.0 or 1.1? What version of HTTP is the server running?
2. What languages (if any) does your browser indicate that it can accept to the server?
3. What is the IP address of your computer?
4. What is the status code returned from the server to your browser?
5. When was the HTML file that you are retrieving last modified at the server?
6. How many bytes of content are being returned to your browser?
7. By inspecting the raw data in the packet content window, do you see any headers within the data that are not displayed in the packet-listing window? If so, name one.

Part 2

8. Inspect the contents of the first HTTP GET request from your browser to the server. Do you see an "IF-MODIFIED-SINCE" line in the HTTP GET?
9. Inspect the contents of the server response. Did the server explicitly return the Contents of the file? How can you tell?
10. Now inspect the contents of the second HTTP GET request from your browser to the server. Do you see an "IF-MODIFIED-SINCE:" line in the HTTP GET? If so, what information follows the "IF-MODIFIED-SINCE:" header?
11. What is the HTTP status code and phrase returned from the server in response to this second HTTP GET? Did the server explicitly return the contents of the file? Explain.

Part 3

12. How many HTTP GET request messages did your browser send? Which packet number in the trace contains the GET message for the Bill or Rights?
13. Which packet number in the trace contains the status code and phrase associated with the response to the HTTP GET request?
14. What is the status code and Phrase in the response?
15. How many data-containing TCP segments were needed to carry the single HTTP response and the text of the Bill of Rights?

Part 4

16. How many HTTP GET request messages did your browser send? To which Internet addresses were these GET requests sent?

17. Can you tell whether your browser downloaded the two images serially, or whether they were downloaded from the two web sites in parallel? Explain.

Part 5

Let's try visiting a web site that is password-protected and examine the sequence of HTTP message exchanged for such a site. The URL http://gaia.cs.umass.edu/wireshark-labs/protected_pages/HTTP-wireshark-file5.html is password protected. The username is "wireshark-students" (without the quotes), and the password is "network" (again, without the quotes). So let's access this "secure"

Password-protected site. Do the following:

•Make sure your browser's cache is cleared, as discussed above, and close down your browser. Then, start up your browser

•Start up the Wireshark packet sniffer

•Enter the following URL into your browser

http://gaia.cs.umass.edu/wireshark-labs/protected_pages/HTTP-wiresharkfile5. Html Type the requested user name and password into the pop up box.

•Stop Wire shark packet capture, and enter "http" in the display-filter-specification window, so that only captured HTTP messages will be displayed later in the packet-listing window.

•(Note: If you are unable to run Wire shark on a live network connection, you can use the http-ethereal-trace-5 packet trace to answer the questions below; see footnote 2. This trace file was gathered while performing the steps above on one of the author's computers.)

Now let's examine the Wire shark output. You might want to first read up on HTTP authentication by reviewing the easy-to-read material on "HTTP Access Authentication Framework" at http://frontier.userland.com/stories/storyReader$2159

18. What is the server's response (status code and phrase) in response to the initial HTTP GET message from your browser?

19. When your browser's sends the HTTP GET message for the second time, what new field is included in the HTTP GET message?

OUTPUT

Answers:

1. Version 1.1

2. Languages supported en-us and en

3. 192.168.1.102

4.200 Ok

5.73 bytes

6. Last-Modified: Tue, 23 Sep 2003 05:29:00 GMT

7.No

8. NO

9. Yes, because it return's text/html on the webpage

10. yes, it tells the last modification date and time

11. Status code: 304 No it does not return any information explicitly as we cannot see any line based text data or any other return type.

12. one, packet no. 8

13.Packet no: 14

14. status code : 200 phrase: OK

15. 4 TCP segments

16. 3 HTTP GET request

IP1: 128.119.245.12 IP2: 165.193.1.102 IP3: 134.241.6.82

17. The browser downloaded the images serially as the arrival times of both the images are different and they are in separate tcp packet.

18. STATUS CODE: 401 PHRASES: Authorization Required

19. Authorization field

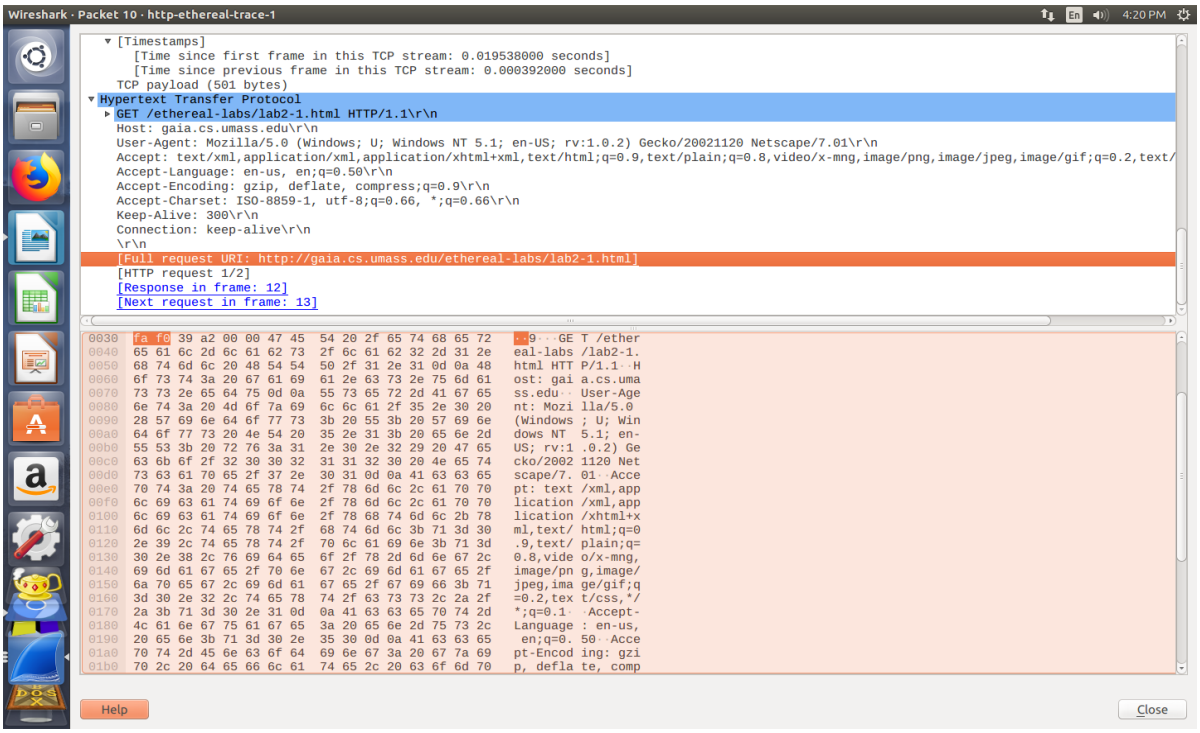


Fig. 12.1. Snapshot of question 1,2

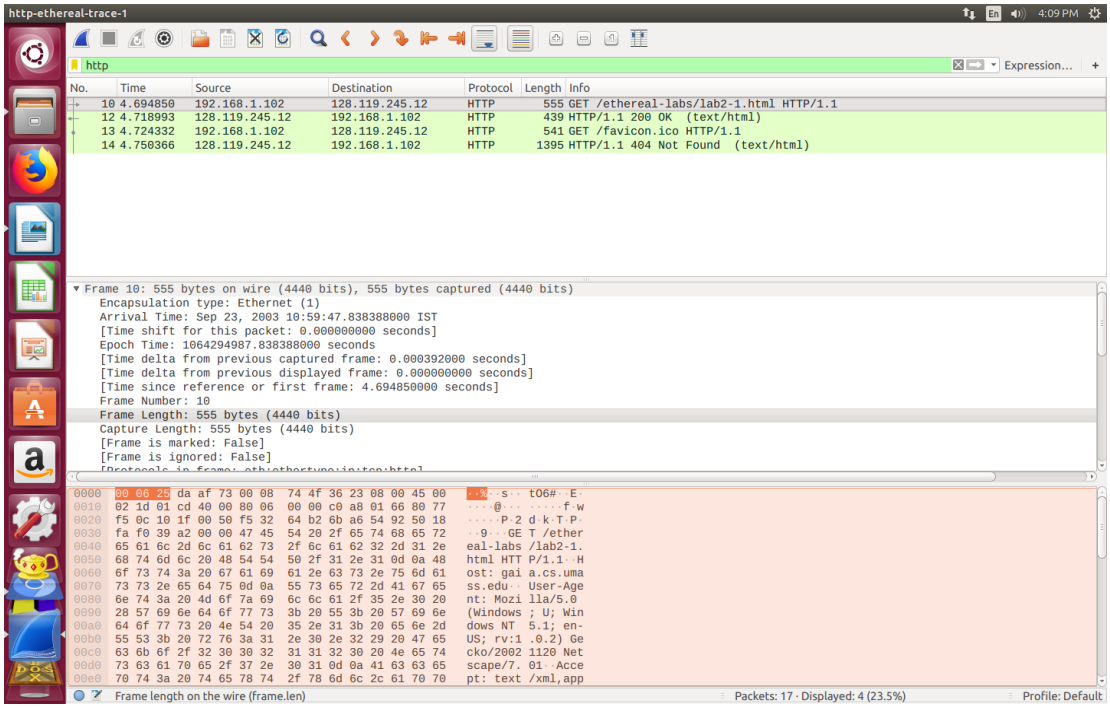


Fig. 12.2. Snapshot of question 3,4,16

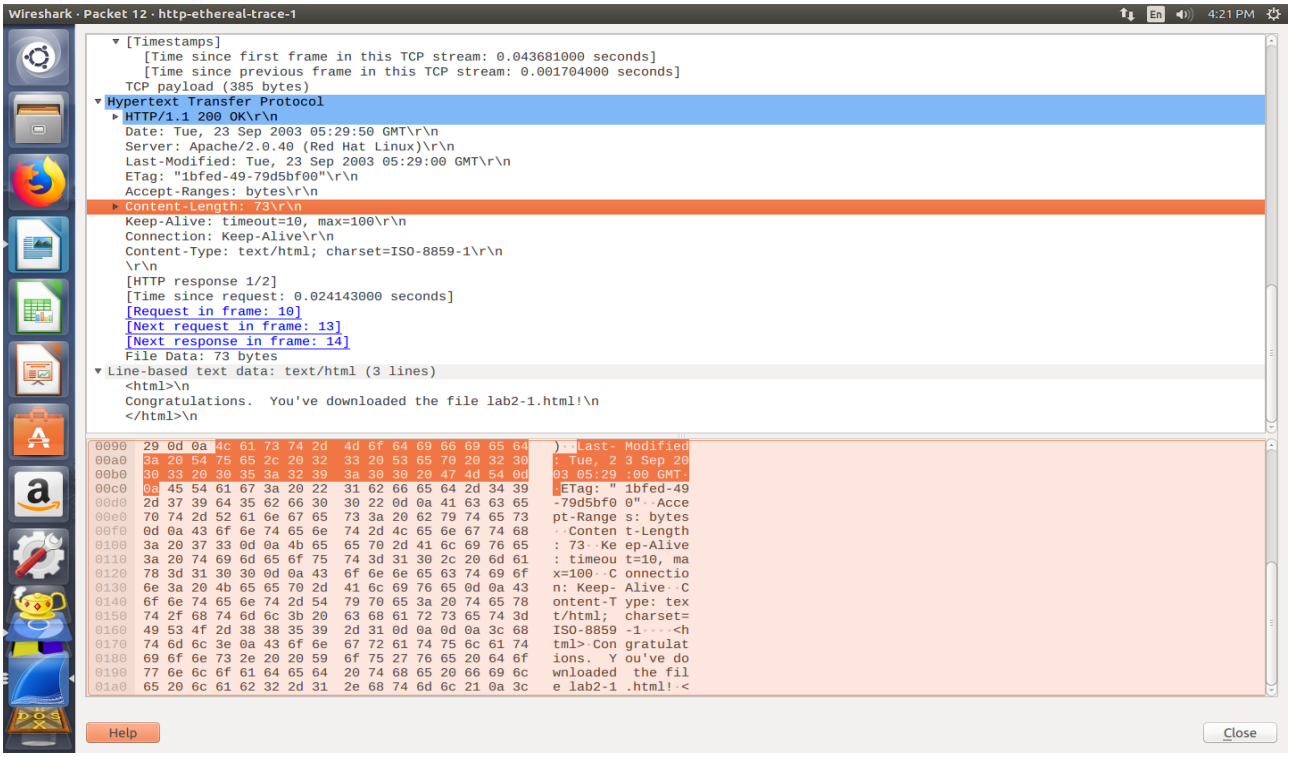


Fig. 12.3. Snapshot of question 5,6,7,8,12,13,14

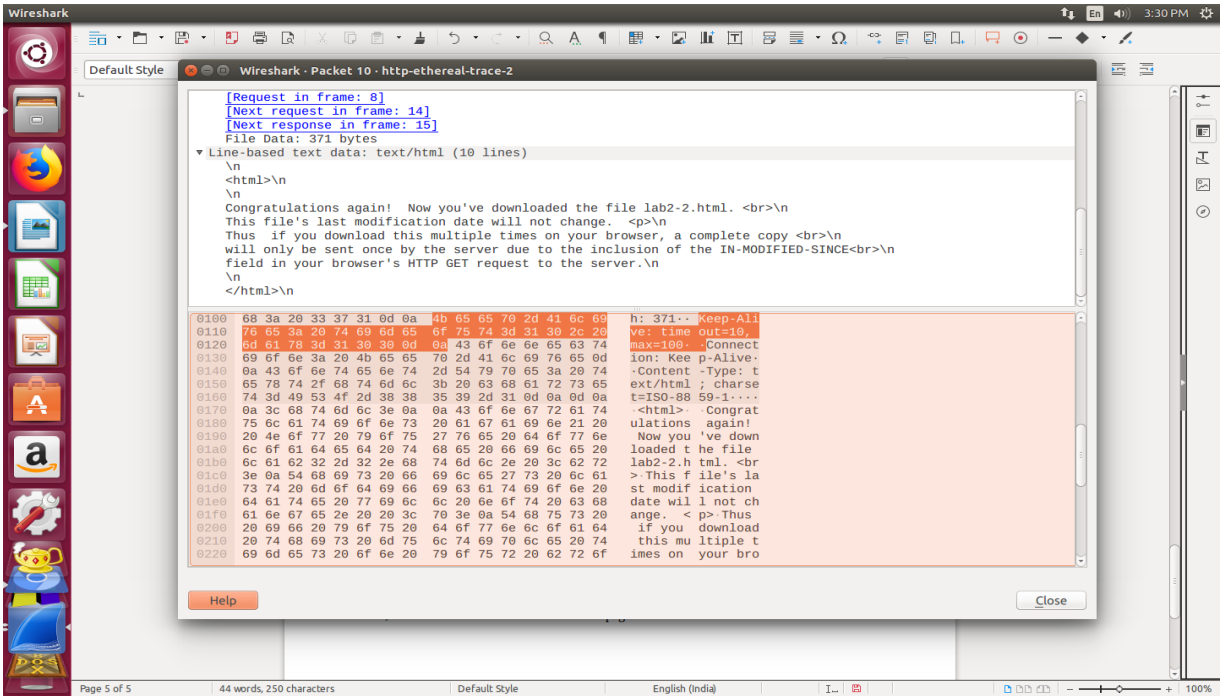


Fig. 12.4. Snapshot of question 9,10,11

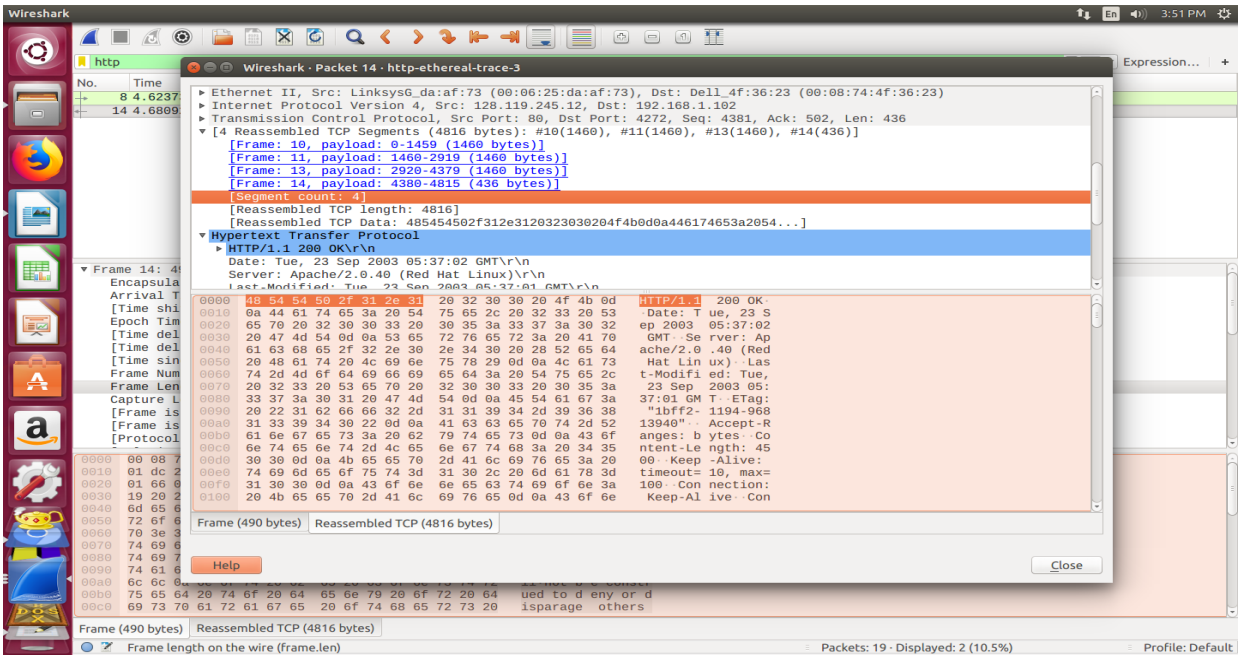


Fig. 12.5. Snapshot of question 15

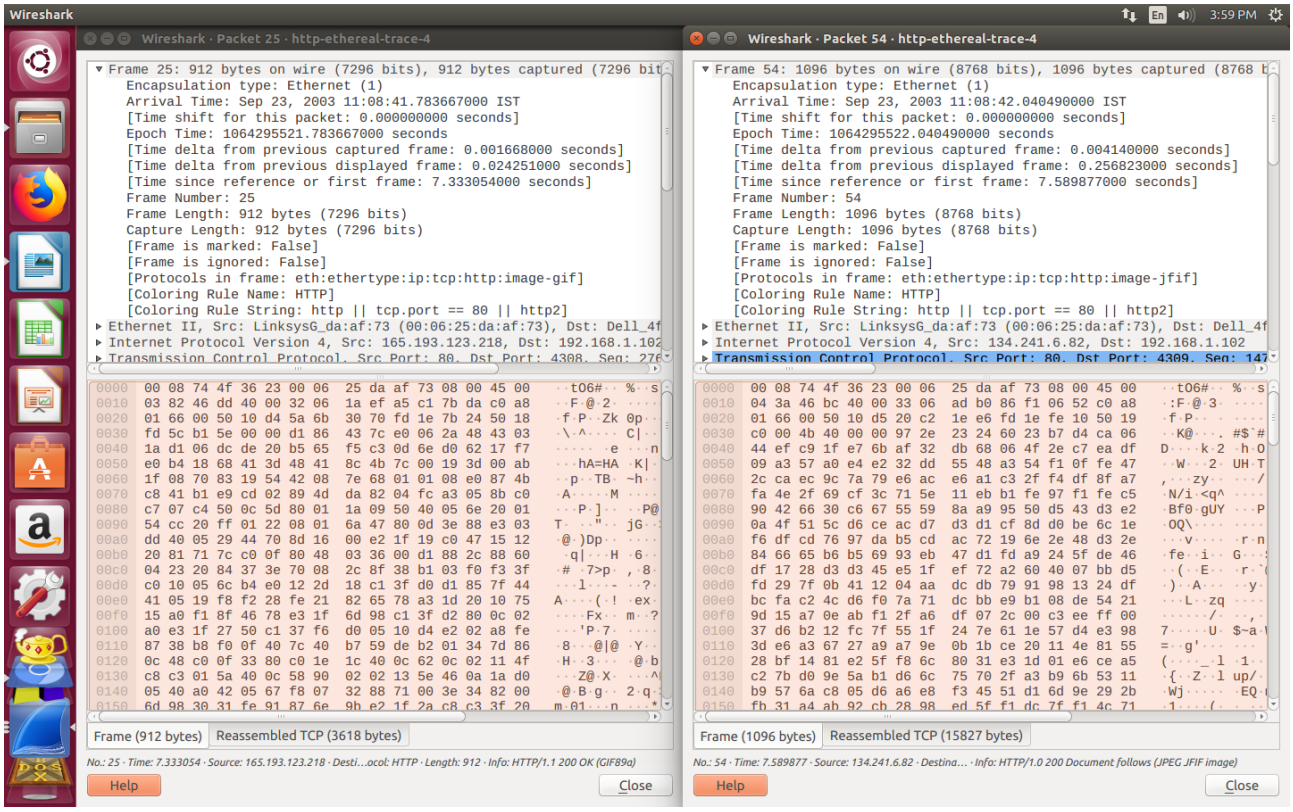


Fig. 12.6. Snapshot of question 17

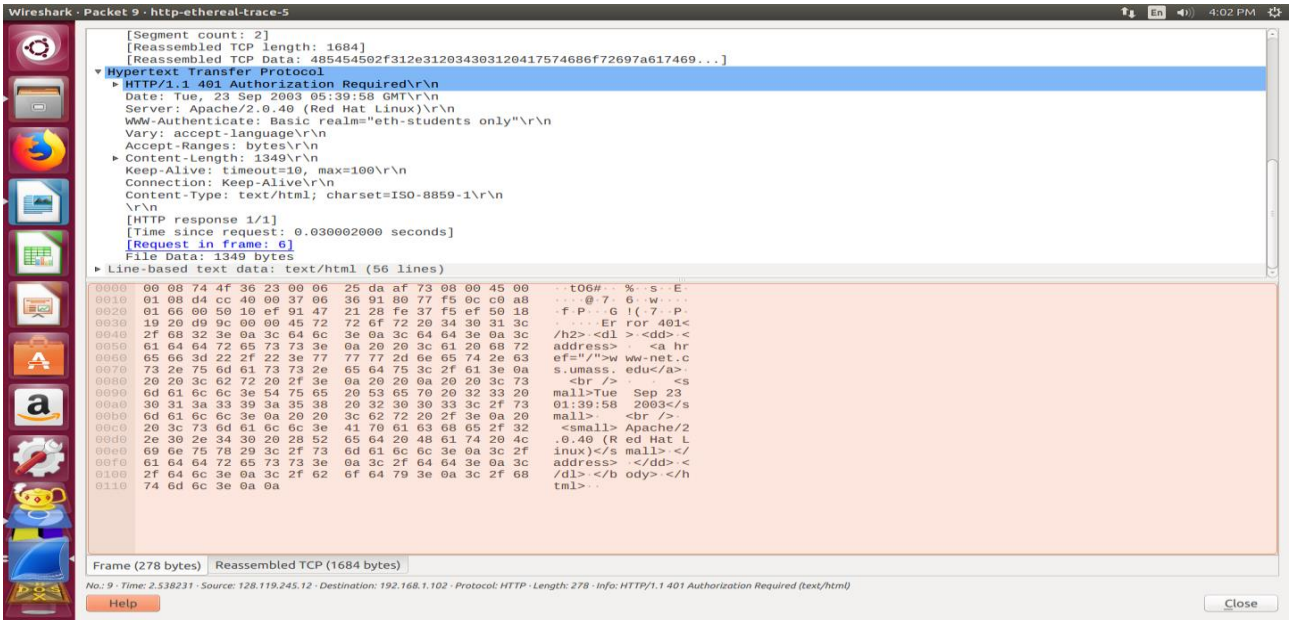


Fig. 12.7. Snapshot of question 18

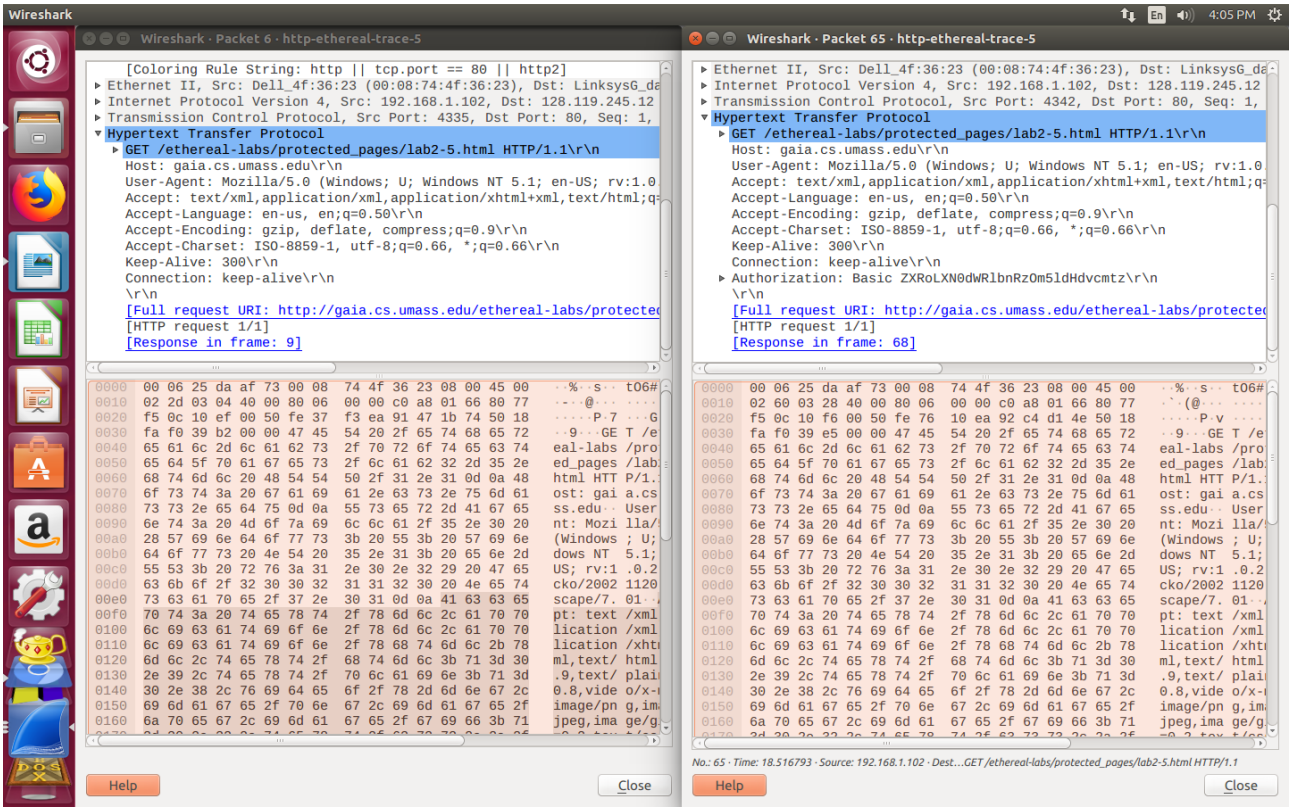


Fig. 12.8. Snapshot of question 19

Program No. 13

OBJECTIVE: To implement socket programming using TCP.

THEORY:

Socket programming is a way of connecting two nodes on a network to communicate with each other. One socket (node) listens on a particular port at an IP, while other socket reaches out to the other to form a connection. Server forms the listener socket while client reaches out to the server.

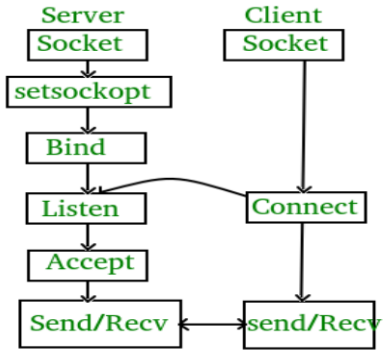


Fig. 12.1. System call used in client-server model

Stages for server

Socket creation:

int sockfd = socket(domain, type, protocol)

sockfd: socket descriptor, an integer (like a file-handle)

domain: integer, communication domain e.g., AF_INET (IPv4 protocol) , AF_INET6 (IPv6 protocol)

type: communication type

SOCK_STREAM: TCP(reliable, connection oriented)

SOCK_DGRAM: UDP(unreliable, connectionless)

protocol: Protocol value for Internet Protocol(IP), which is 0. This is the same number which appears on protocol field in the IP header of a packet.(man protocols for more details)

Setsockopt:

int setsockopt(int sockfd, int level, int optname,

const void *optval, socklen_t optlen);

This helps in manipulating options for the socket referred by the file descriptor sockfd. This is completely optional, but it helps in reuse of address and port. Prevents error such as: “address already in use”.

Bind:

int bind(int sockfd, const struct sockaddr *addr,

socklen_t addrlen);

After creation of the socket, bind function binds the socket to the address and port number specified in addr(custom data structure). In the example code, we bind the server to the localhost, hence we use INADDR_ANY to specify the IP address.

Listen:

int listen(int sockfd, int backlog);

It puts the server socket in a passive mode, where it waits for the client to approach the server to make a connection. The backlog, defines the maximum length to which the queue of pending connections for sockfd may grow. If a connection request arrives when the queue is full, the client may receive an error with an indication of ECONNREFUSED.

Accept:

int new_socket= accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);

It extracts the first connection request on the queue of pending connections for the listening socket, sockfd, creates a new connected socket, and returns a new file descriptor referring to that socket. At this point, connection is established between client and server, and they are ready to transfer data.

Stages for Client

Socket connection: Exactly same as that of server's socket creation

Connect:

int connect(int sockfd, const struct sockaddr *addr,

socklen_t addrlen);

- The connect () system call connects the socket referred to by the file descriptor sockfd to the address specified by addr. Server's address and port is specified in addr.

The steps involved in establishing a socket on the client side are as follows:

1.Create a socket with the socket() system call.

2.Connect the socket to the address of the server using the connect() system call

3.Send and receive data. There are a number of ways to do this, but the simplest is to use the read() and write() system calls.

The steps involved in establishing a socket on the server side are as follows:

1. Create a socket with the socket() system call

2. Bind the socket to an address using the bind() system call. For a server socket on the Internet, an address consists of a port number on the host machine.

3. Listen for connections with the listen() system call

4. Accept a connection with the accept() system call. This call typically blocks until a client connects with the server.

5. Send and receive data

The interactions among the system calls is shown in figure 12.1