Vidyalankar Institute of Technology

Wadala(E), Mumbai 400037

CERTIFICATE

This is to certify that this Lab Work (ISA) in the subject of

Computer Network

of semester **V** of **Computer Engineering** course (**Academic Year 2023-24**)

Submitted by

Mr Deep Salunkhe Roll No: 21102A0014

is accepted by the Department

Amit K. Nerurkar

Professor, In Charge

VIT Vidyalankar Institute of Technology
Accredited A+ by NAAC

DEPARTMENT OF COMPUTER ENGINEERING
Computer Network Lab

Sr no	Title of the experiment	Date	Sign
1	Study of historical perspective networks	18/7/2023	
2	Fabrication of cables	25/7/2023	
3	Implementation of error detection mechanism	1/8/2023	
4	Implementation of error detection and correction mechanism	8/8/2023	
5	Networking command	22/8/2023	
6	Implementation of classfull IP addressing	5/9/2023	
7	PBLE 1:Access control	12/9/2023	
8	PBLE 2: Subnetting	26/9/2023	
9	Network analyzers	3/10/2023	
10	Socket programming	10/10/2023	
11	Server room visit	17/10/2023	

VIT Vidyalankar Institute of Technology
Accredited A+ by NAAC

DEPARTMENT OF COMPUTER ENGINEERING
Computer Network Lab

Semester	T.E. Semester V – Computer Engineering
Subject	Computer Network
Subject Professor In-charge	Prof. Amit K. Nerurkar
Assisting Teachers	Prof. Amit K. Nerurkar
Laboratory	M-313-A

Student Name	Deep Salunkhe
Roll Number	21102A0014
TE Division	A

DEPARTMENT OF COMPUTER ENGINEERING
Computer Network Lab

Title: Introduction to Computer Network

Explanation:

➔what is Computer Network?

A computer network is a collection of interconnected computers, devices, and communication. channels that allow the exchange of data and resources among them. The primary purpose of a computer network is to enable efficient and reliable communication, sharing of information, and collaboration between users and devices within the network.

➔How computer network got evolved?

Early Networking Concepts (1960s): In the early days of computing, individual computers operated in isolation, and there was a growing recognition of the potential benefits of connecting them. Researchers began exploring concepts like time-sharing, which allowed multiple users to access a single computer remotely. The development of packet-switching, a method of breaking data into small packets for transmission across a network, laid the foundation for modern data communication.

ARPANET and the Birth of the Internet (1960s-1970s): The Advanced Research Projects Agency Network (ARPANET) was a pioneering project funded by the United States Department of Defense. It went online in 1969 and is considered the precursor to the internet. ARPANET used packet-switching technology and connected computers at various research institutions, enabling the exchange of data and resources.

TCP/IP and Standardization (1970s): The Transmission Control Protocol (TCP) and Internet Protocol (IP) were developed in the 1970s, providing a set of rules and standards for data transmission and communication across networks. TCP/IP became the foundation of the modern internet and allowed different networks to interconnect, forming a global network of networks.

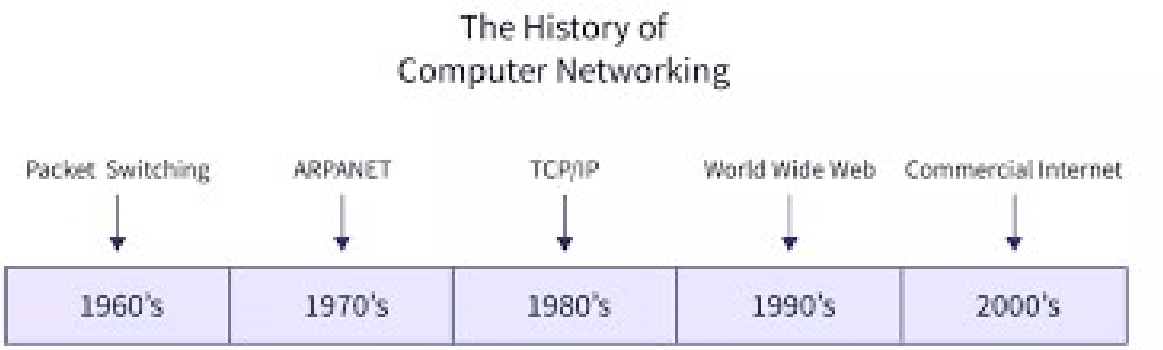
Ethernet and Local Area Networks (LANs) (1970s-1980s): Ethernet, developed by Robert Metcalfe at Xerox PARC in the early 1970s, became the dominant technology for connecting computers within a local area. Ethernet's popularity led to the creation of LANs, allowing devices within a limited geographic area to communicate and share resources.

Commercialization of the Internet (1990s): The 1990s saw the commercialization of the internet, leading to its widespread adoption by businesses and the general public. The introduction of web browsers like Netscape Navigator and Internet Explorer made the internet more user-friendly, and the World Wide Web (WWW) became a crucial aspect of internet usage, allowing easy access to information and services.

Broadband and High-Speed Internet (2000s): Advancements in telecommunications and networking technologies led to the widespread adoption of broadband internet, offering high-speed and always-on connectivity. This allowed for richer multimedia content, online video streaming, and other bandwidth-intensive applications.

Wireless Networks and Mobility (2000s): The proliferation of wireless technologies, such as Wi-Fi, Bluetooth, and cellular networks, brought about the era of mobile computing. Users could now access the internet and connect to networks while on the move, leading to the rapid growth of smartphones, tablets, and other mobile devices.

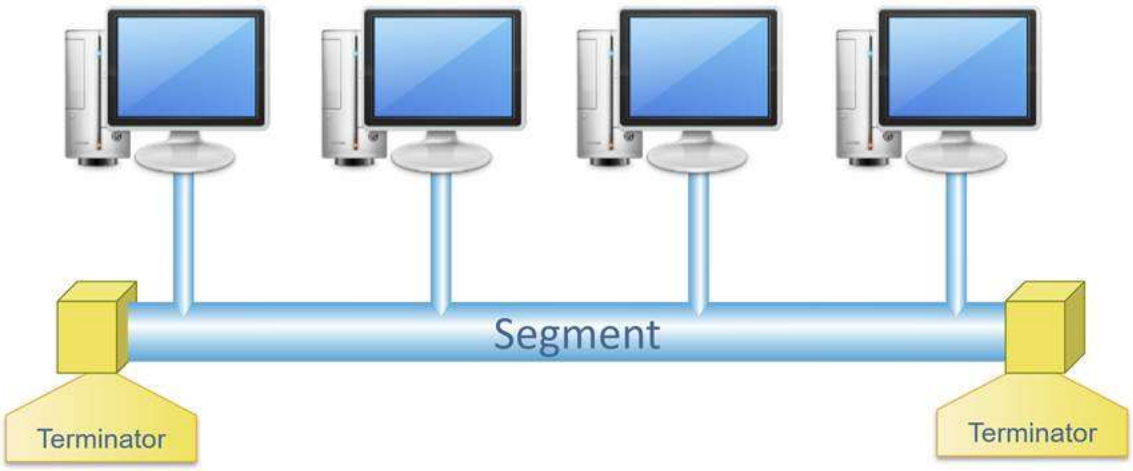
Internet of Things (IoT) (2010s): The concept of the Internet of Things emerged, where everyday objects and devices are equipped with sensors and connectivity to communicate with each other and exchange data over the internet. This has led to smart homes, smart cities, and various IoT applications.



➔what are topologies of computer network?

Bus Topology:

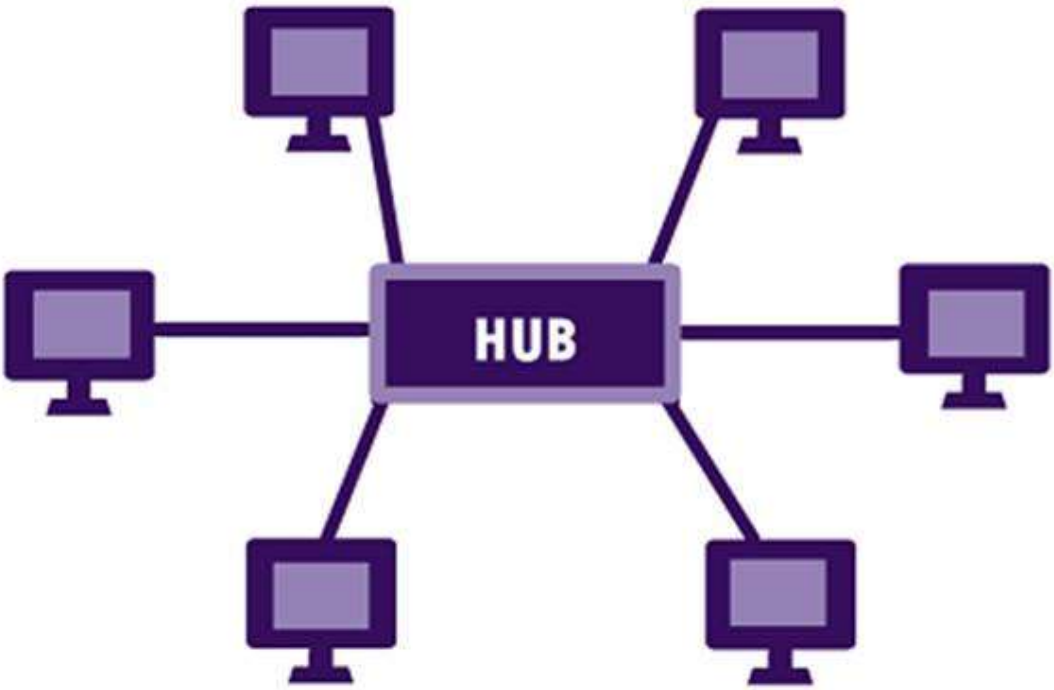
In a bus topology, all devices are connected to a single central cable called the "bus." Each device on the network receives all the data transmitted on the bus but will only process data intended for itself. One drawback of the bus topology is that if the main cable (bus) fails, the entire network may become inoperable.



Star Topology:

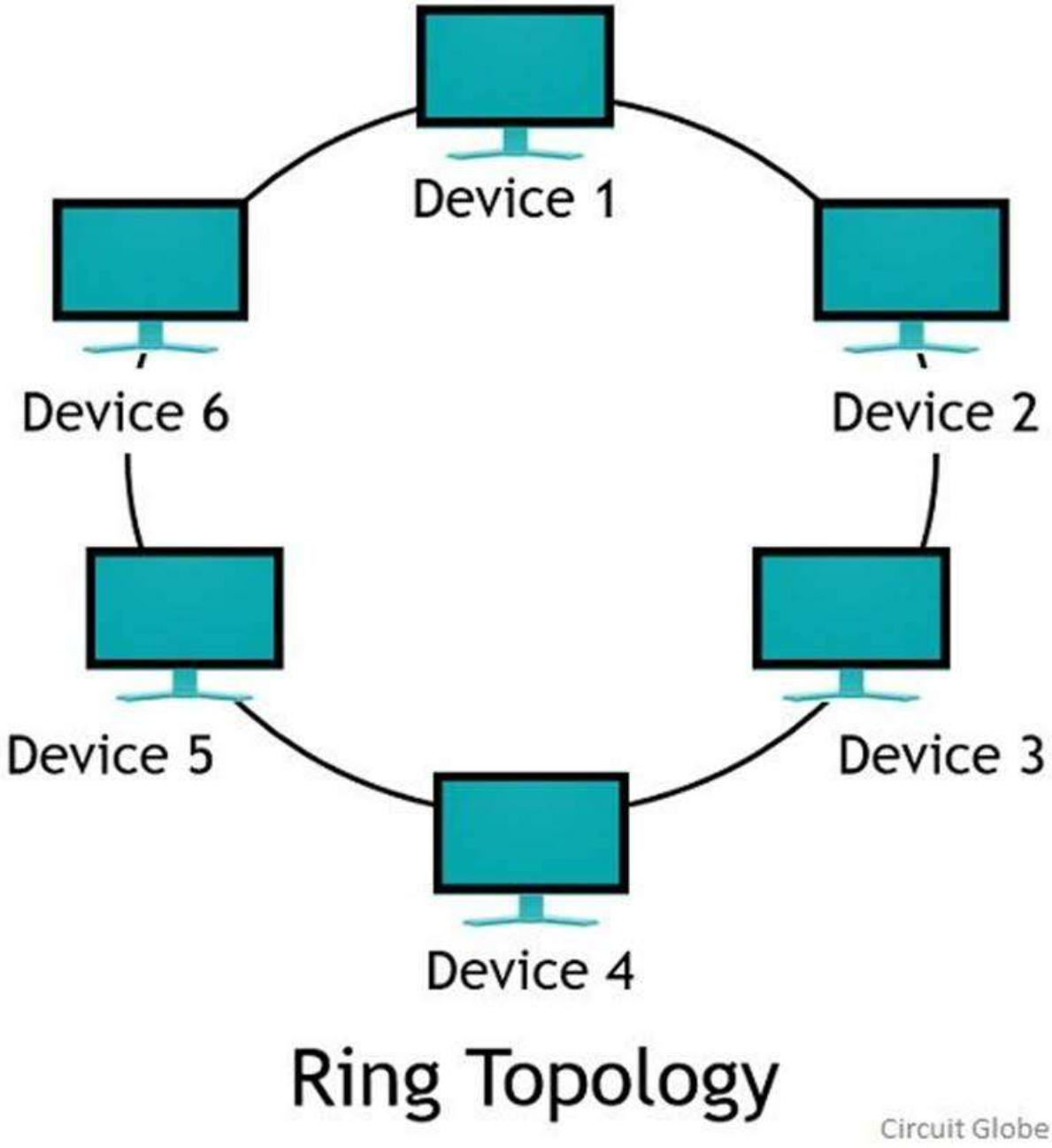
In a star topology, all devices are connected directly to a central hub or switch. The central hub acts as a connection point for all devices, and data transmissions are relayed through the hub. If

one device or cable fails, it will not affect the rest of the network, as all communication goes through the central hub.



Ring Topology:

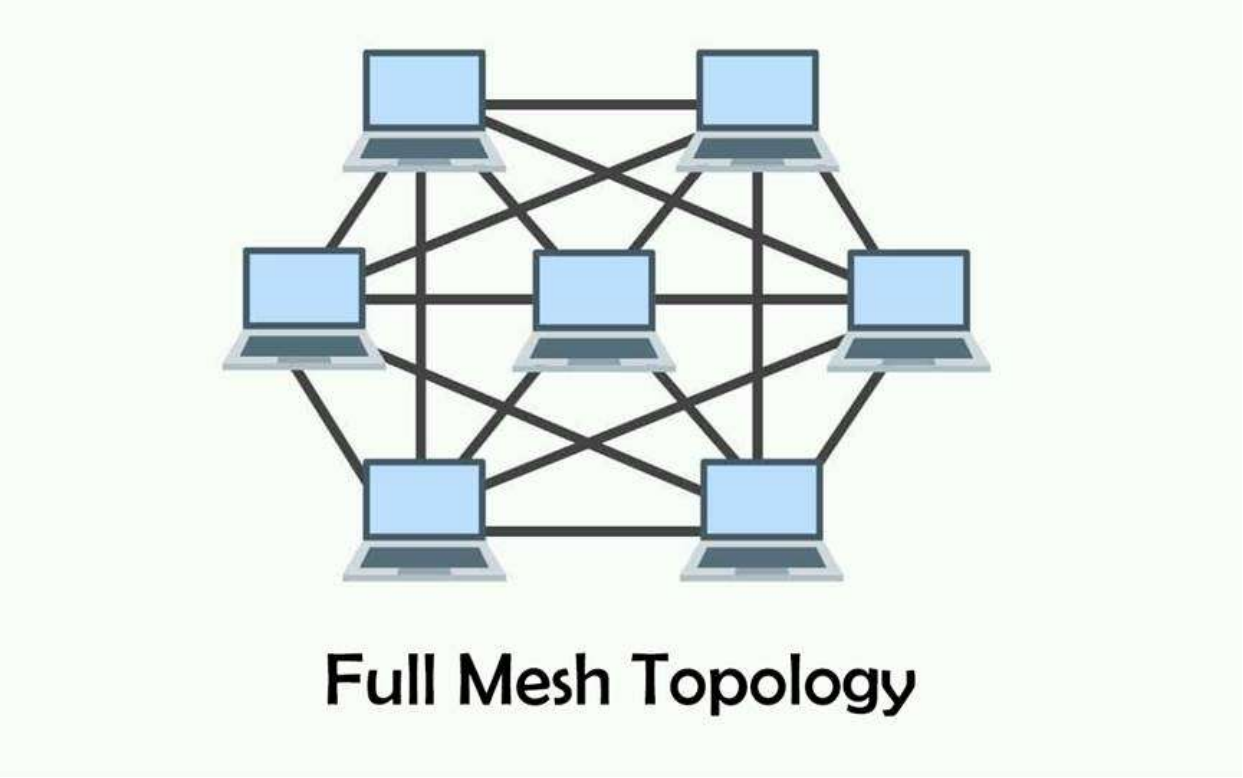
In a ring topology, devices are connected in a closed-loop, forming a ring-like structure. Each device is connected directly to two neighboring devices, and data circulates around the ring until it reaches its intended destination. Ring topologies can suffer from performance issues if a single device or connection fails, disrupting the entire network.



Mesh Topology:

In a mesh topology, every device is connected to every other device in the network, creating a fully interconnected system. Mesh networks are highly fault-tolerant, as data can be rerouted through multiple paths if one link fails. However, this connectivity comes at the cost of increased

complexity and cabling requirements.



Tree Topology (Hierarchical Topology):

A tree topology combines characteristics of the bus and star topologies. Devices are arranged in a hierarchical structure with multiple levels of hubs or switches. The higher-level hubs connect to lower-level hubs, forming a tree-like structure. Tree topologies allow for scalable Tree topology, also known as hierarchical topology, is a network structure that combines characteristics of the bus and star topologies. It forms a tree-like hierarchy with multiple levels of interconnections, typically with a root node at the top and branches extending downward to lower-level nodes. Tree topology is often used in large networks, as it allows for efficient organization and easy expansion.

key features and characteristics of the tree topology:

Root Node: At the top of the hierarchy, there is a central node called the root node or the main hub. The root node acts as the primary connection point for all other nodes in the network. In some cases, the root node could be a central switch or a powerful network device.

Levels: The tree topology is organized into levels or layers. Each level represents a hierarchical tier of interconnected nodes. The levels radiate from the root node, with lower-level nodes branching out from higher-level nodes.

Branches: Each level of the hierarchy has branches that extend downward to lower levels. These branches represent the connections between nodes. Typically, each node at a higher level has multiple child nodes at the next lower level.

Parent and Child Nodes: In a tree topology, nodes at a higher level are considered the parent nodes, and nodes at the next lower level are called child nodes. Each child node is directly connected to its parent node, and data flows from the parent node to its children.

Data Transmission: Data transmission in a tree topology usually follows a path from the root node to the specific destination node. The root node broadcasts data to all its child nodes, and each child node, in turn, broadcasts the data to its children until it reaches the intended destination node.

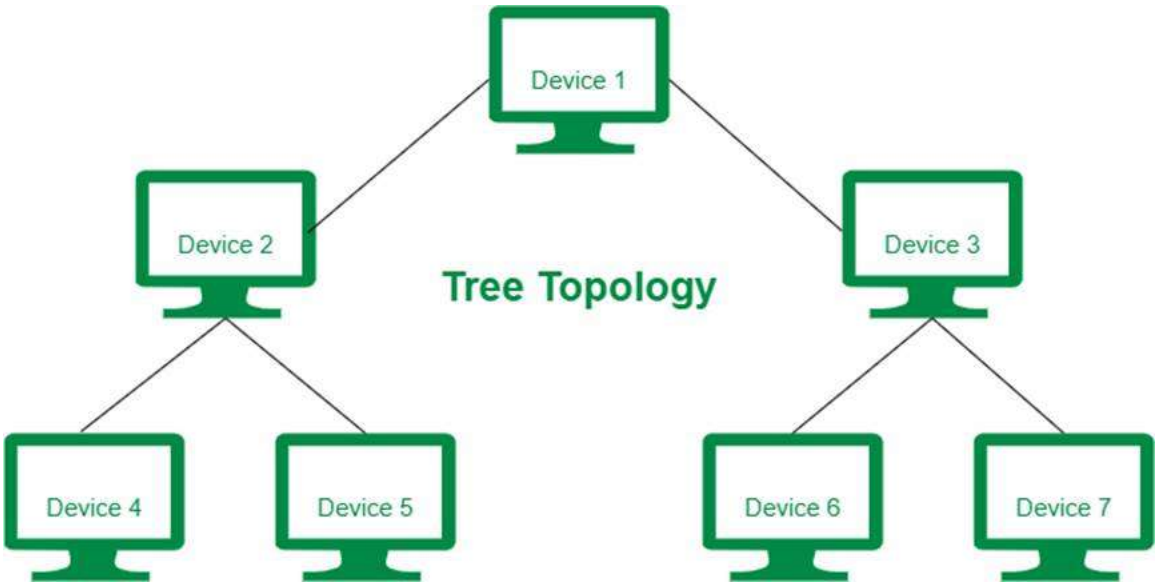
Fault Isolation: One of the advantages of the tree topology is its fault isolation capability. If a node

or connection fails in a particular branch, it only affects the nodes in that branch. The rest of the network remains operational, as the root node and other branches are not affected.

Scalability: Tree topologies are easily scalable. When new devices or nodes need to be added to the network, they can be connected as child nodes to existing parent nodes. This hierarchical expansion makes it relatively straightforward to grow the network as needed.

Centralized Control: Since the root node acts as the central point of connection, network administrators can have better control and management of the network. This centralized control can streamline network administration tasks and facilitate easier troubleshooting.

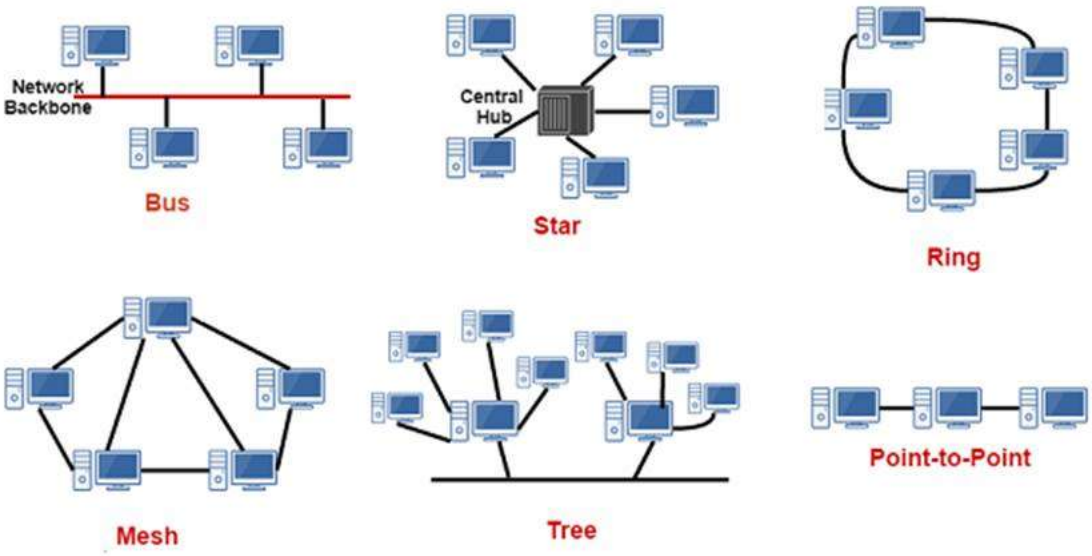
Examples of tree topology in practice include organizational networks in large companies, campus networks connecting multiple buildings in a university, and regional networks linking branches of a bank or retail chain.network expansion and can handle large networks efficiently.



Hybrid Topology:

A hybrid topology is a combination of two or more different topologies. For example, a network may have a combination of star and bus topologies or a mix of ring and mesh topologies. This approach allows network designers to customize the network to meet specific requirements and optimize performance.

Each network topology has its advantages and disadvantages, and the choice of the best topology depends on factors such as network size, fault tolerance requirements, ease of installation, scalability, and cost considerations. Network administrators carefully evaluate these factors to determine the most suitable topology for a particular network implementation.



➔Types of Networks

Personal Area Network (PAN):

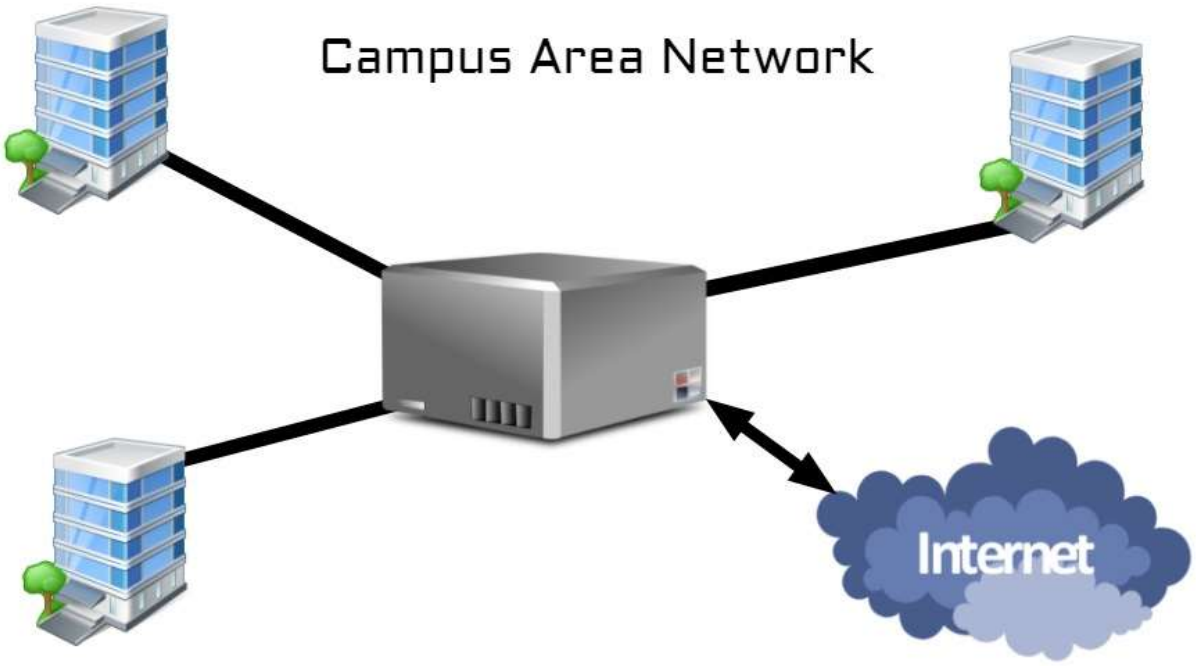
- Range: Up to 10 meters (approximately 33 feet)
- Description: A PAN is a network used for communication between devices in close proximity to an individual, such as a smartphone connecting to a smartwatch or a wireless mouse connecting to a computer.

Local Area Network (LAN):

- Range: Up to a few Kilometres.
- Description: A LAN is a network that covers a small geographic area, such as a single building, office, home, or campus. It is used for connecting devices within close proximity, allowing them to share resources like files, printers, and internet access. Ethernet and Wi-Fi are common technologies used in LANs

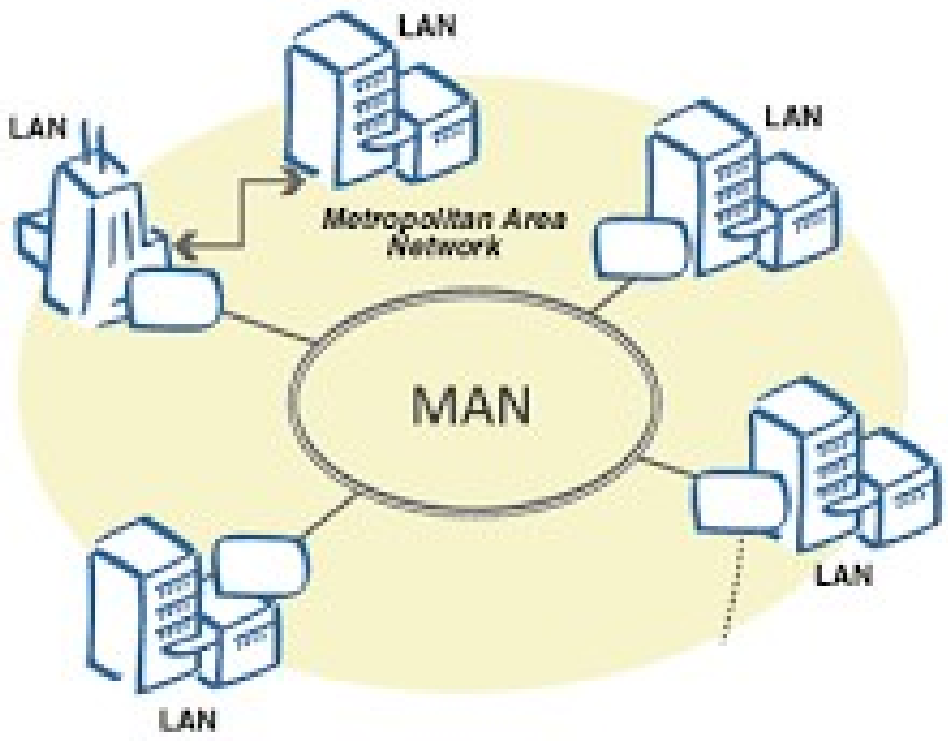
Campus Area Network (CAN):

- Typically covers a university campus or large office complex.
- Description: A CAN is a network that connects multiple buildings within a limited geographic area, such as a college campus or a large corporate office.



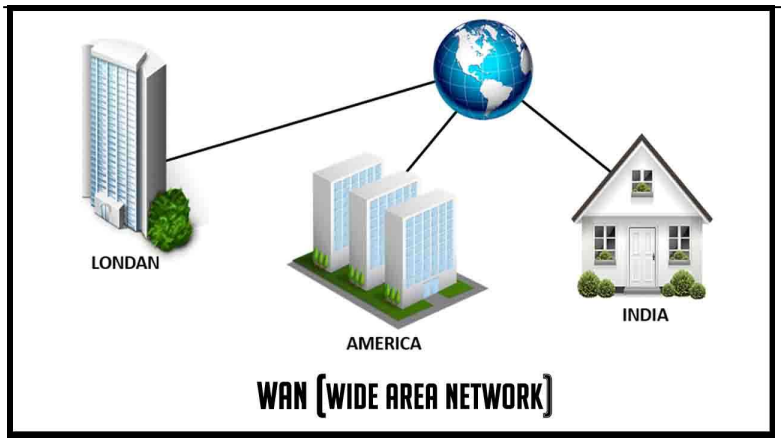
Metropolitan Area Network (MAN):

- Range: Spans across a city or metropolitan area.
- Description: A MAN is a network that covers a larger area than a LAN but smaller than a WAN. It connects multiple LANs together within a city or metropolitan region.



Wide Area Network (WAN:

- Range: Can span across cities, countries, or even continents.
- Description: A WAN is a network that covers a large geographical area and connects multiple LANs and MANs together. The Internet is the most well-known example of a WAN, covering the entire globe.



Conclusion:

Computer networks are interconnected systems that allow the exchange of data and resources among devices and users. They evolved from early isolated computers to the birth of the internet with ARPANET. TCP/IP standardization enabled global networking, and Ethernet popularized Local Area Networks (LANs). The commercialization of the internet in the 1990s led to widespread adoption, and advancements in technology brought high-speed internet and wireless networks.

Topologies in computer networks define their structure. Bus connects devices to a central cable; Star links devices to a central hub; Ring forms a closed-loop; Mesh interconnects all devices; and Tree is hierarchically arranged Network. Hybrid combines multiple topologies for customized networks.

Network types include PAN for personal devices, LAN for small areas, CAN for campus, MAN for metropolitan regions, and WAN for large geographical areas. The Internet represents a global WAN. The evolution of computer networks has revolutionized communication, collaboration, and access to information worldwide.

VIT Vidyalankar Institute of Technology
Accredited A+ by NAAC

DEPARTMENT OF COMPUTER ENGINEERING
Computer Network Lab

Semester	T.E. Semester V – Computer Engineering
Subject	Computer Network
Subject Professor In-charge	Prof. Amit K. Nerurkar
Assisting Teachers	Prof. Amit K. Nerurkar
Laboratory	M-313-A

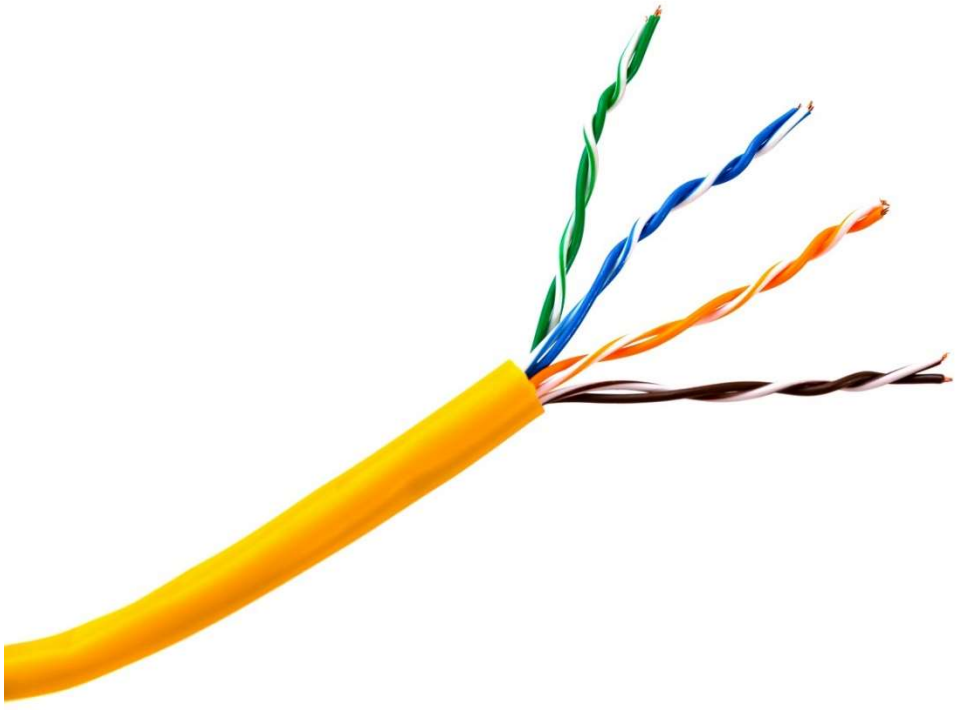
Student Name	Deep Salunkhe
Roll Number	21102A0014
TE Division	A

Title: Fabrication of cables using virtual lab.

Explanation:

Guided Media: Guided media, also known as bounded media, refers to the transmission of signals in a wired or physically confined environment. In guided media, electromagnetic signals are guided along a physical path, ensuring secure and efficient data transmission. The main types of guided media are twisted wire, coaxial cable, and optical fiber.

1. **Twisted Wire(UTP):**

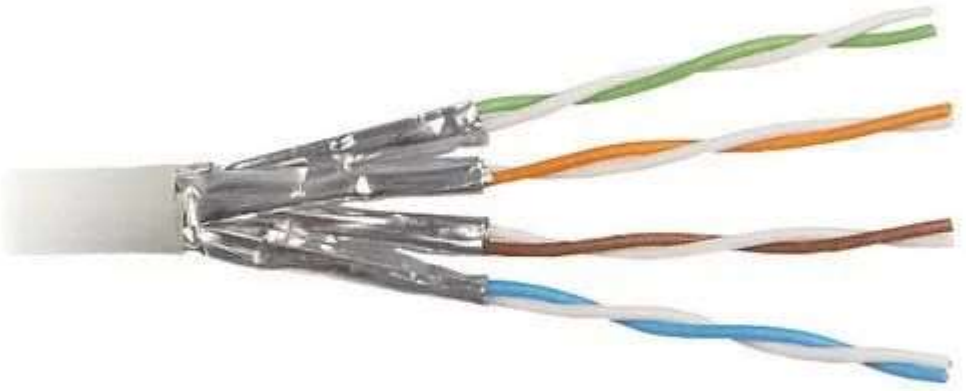


- Working Principle: Twisted wire is a type of guided media that consists of two insulated copper wires twisted together in a helical form. The twisting helps reduce electromagnetic interference and crosstalk between adjacent pairs of wires.
- Range: Twisted wire can support data transmission rates up to several Mbps for short distances, typically up to a few kilometers.
- Applications: Twisted wire is commonly used in telephone systems, local area networks (LANs), and some broadband connections.
- Advantages:
    - Cost-effective and easy to install.

- Suitable for short-distance communication needs.
- Provides reasonable data transmission rates for basic applications.

- Disadvantages:
  - Prone to electromagnetic interference and noise, affecting data quality.
  - Limited bandwidth and distance compared to other guided media.

2. **Shielded Twisted Pair (STP**):



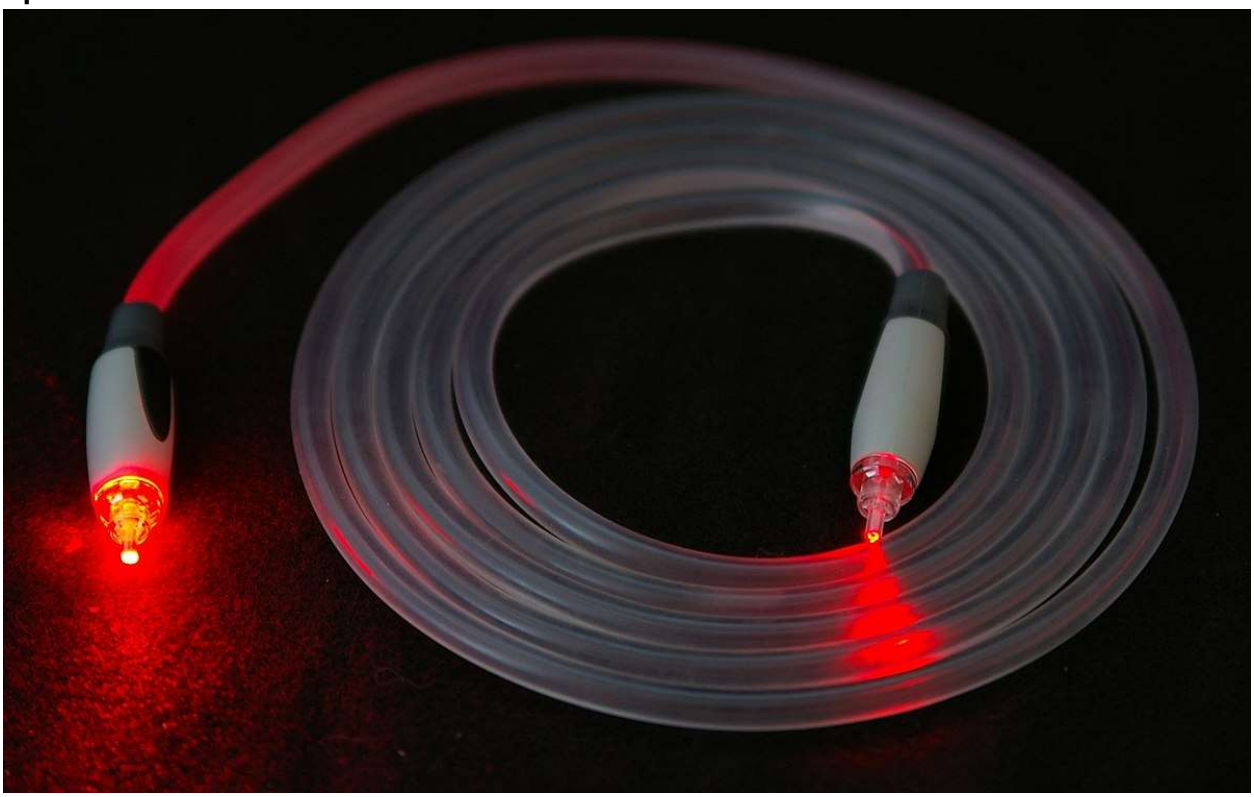
- Working Principle: STP is a variation of twisted wire with additional shielding (usually metallic foil or braided mesh) around each twisted pair, providing better protection against external interference.

- Range: STP offers improved performance and can support higher data rates compared to unshielded twisted pair (UTP).
- Applications: STP is used in environments with higher electromagnetic interference, such as industrial settings and data centers.
- Advantages:
    - Improved protection against external interference.
    - Higher data transmission rates and longer distance capabilities compared to UTP.
- Disadvantages:
    - More expensive and bulkier than UTP.
    - Installation and maintenance can be more complex.

3. **Coaxial Cable**:

- Working Principle: Coaxial cable consists of a copper conductor at the center, surrounded by a dielectric insulating layer, and an outer metallic shield. The shield helps prevent signal loss due to external interference.
- Range: Coaxial cables can support data transmission rates up to several Gbps over relatively long distances, typically up to 10 kilometers or more.
- Applications: Coaxial cables are used in cable television (CATV) networks, broadband internet, and CCTV systems.
- Advantages:
    - High data transmission rates and longer reach compared to twisted pair cables.
    - Less susceptible to electromagnetic interference.
- Disadvantages:
    - Thicker and less flexible than twisted pair cables.
    - Costlier and more challenging to install.

4. **Optical Fibe**r:



- Working Principle: Optical fiber uses thin strands of glass or plastic (fiber) to transmit data as pulses of light, utilizing the principle of total internal reflection.
- Range: Optical fiber offers very high data transmission rates, ranging from Mbps to Tbps, and can cover long distances, spanning hundreds of kilometers without significant signal loss.
- Applications: Optical fiber is extensively used in long-distance telecommunication networks, high-speed internet connections, and high-bandwidth data transmission.
- Advantages:
    - Immunity to electromagnetic interference, making it ideal for high-noise environments.
    - Extremely high data transmission rates and long-distance capabilities.
    - Lighter and more durable than copper-based cables.
- Disadvantages:
    - Higher installation and equipment costs compared to other guided media.
    - More delicate and susceptible to damage during installation and maintenance.

Conclusion:

In conclusion, guided media plays a vital role in the transmission of signals in wired or physically confined environments. The three main types of guided media, namely twisted wire, coaxial cable, and optical fiber, each offer unique advantages and disadvantages. Twisted wire is a cost-effective option suitable for short-distance communication needs but is prone to electromagnetic interference. Shielded twisted pair (STP) provides improved protection against external interference, making it preferable for industrial settings and data centers. Coaxial cable offers high data transmission rates and is less susceptible to electromagnetic interference, making it a popular choice for cable television networks and broadband internet. On the other hand, optical fiber excels in providing extremely high data transmission rates and immunity to electromagnetic interference, making it the preferred choice for long-distance telecommunication networks and high-bandwidth data transmission. While each guided medium has its strengths and limitations, the selection of the appropriate medium depends on specific requirements, data transmission needs, distance, and budget considerations.

VIT Vidyalankar Institute of Technology
Accredited A+ by NAAC

DEPARTMENT OF COMPUTER ENGINEERING
Computer Network Lab

Semester	T.E. Semester V – Computer Engineering
Subject	Computer Network
Subject Professor In-charge	Prof. Amit K. Nerurkar
Assisting Teachers	Prof. Amit K. Nerurkar
Laboratory	M-313-A

Student Name	Deep Salunkhe
Roll Number	21102A0014
TE Division	A

Title : Error Handling

Theory:

Parity Check:

Parity is a simple error-checking method used in digital communication to detect errors in data transmission. It involves adding an extra bit, called a **parity bit**, to a group of data bits.

- **Even Parity:** In even parity, the total number of 1s in the data and parity bit combined should be even. If there's an odd number of 1s, an error is detected.
- **Odd Parity:** In odd parity, the total number of 1s in the data and parity bit combined should be odd. If there's an even number of 1s, an error is detected.

Parity check is useful for detecting single-bit errors. If a single bit is flipped during transmission, the parity check will detect it. However, it cannot correct errors, only identify them.

Checksum:

A **checksum** is a value calculated from a data set for the purpose of error checking. It is generated by summing up the binary values of data in a specific way and then appending the checksum to the data. When the data is received, the recipient can calculate the checksum again and compare it to the received checksum to check for errors.

Checksums are commonly used in networking protocols, such as the Internet Control Message Protocol (ICMP) and the Transmission Control Protocol (TCP), to ensure data integrity during transmission.

One of the most common checksum algorithms is the **Internet Checksum**, which calculates a 16-bit checksum for data. If the calculated checksum at the receiver's end doesn't match the received checksum, an error is detected.

CRC (Cyclic Redundancy Check):

A **Cyclic Redundancy Check (CRC)** is an advanced error-checking method used for data transmission. CRCs use polynomial division to detect errors. A specific polynomial, called the **generator polynomial**, is used for CRC calculations.

Here's how CRC works:

1. The sender and receiver agree on a generator polynomial (e.g., x^3 + x + 1).
2. The sender appends a certain number of zeros (equal to the degree of the generator polynomial) to the data.
3. The sender performs polynomial division on the combined data to create a **remainder**, which is appended to the data.
4. The data with the remainder is transmitted.
5. The receiver performs the same polynomial division using the received data, including the remainder.
6. If the remainder at the receiver's end is not zero, an error is detected.

CRCs are powerful error-detection methods and are widely used in network communication and storage systems. They can detect a wide range of errors, including burst errors, and are more robust than simple parity checks or checksums.

Implementation:

```cpp
#include <iostream>
#include <vector>
#include <cstdlib>
using namespace std;

// Parity check functions
void sender_code_parity_check(vector<int>& data) {
    cout<<"enter the data to transmit........"<<endl;
    int n;
    cout<<"enter the number of bits in data:";
    cin>>n;
    //taking input and calculating parity bit
    int n_one=0;
    cout<<"start entering data bitwise"<<endl;
    for(int i=0;i<n;i++)
    {
        int temp;
        cin>>temp;
        if(temp==1)n_one++;
        data.push_back(temp);
    }
    //now adding parity bit to last
    int parity_bit;
    if(n_one%2==0)
    {
        //if even
        parity_bit=0;
    }else{
        //if odd
        parity_bit=1;
    }
    //appending parity bit to last
    data.push_back(parity_bit);
}

// Transmitter function
void transmit_P(vector<int>& data) {
    int n=data.size();

    cout<<"want to transmit with error(1) or not(0):";
```

```
    int check=0;
    cin>>check;

    //processing according to instruction
    if(check==0)
    {
        //without error

        //not updating the data
    }else{
        //with error

        //now converting data with someerror
        //as this is parity bit it can handle only 1 bit error
        int posi=rand()%n;
        //fliping the bit (to generate error)
        if(data[posi]==1)
        {
            data[posi]=0;
        }else{
            data[posi]=1;
        }
    }
}

void reciever_code_parity_check(vector<int>& data) {
    //calculating no_of_ones
    int n_one=0;
    int n=data.size();

    for(int i=0;i<n;i++)
    {
        if(data[i]==1)n_one++;
    }
    //finding parity bit for reciever
    int parity_bit;
    if(n_one%2==0)
    {
        //if even
        parity_bit=0;
    }else{
        //if odd
        parity_bit=1;
```

```
    }
    if(parity_bit==0)
    {
        cout<<"data send.........(without error)";
    }else{
        cout<<"data send..........(with error)";
    }
}

//-------------------------------------------------------------------------------
-------------------------
// Check Sum functions
void accept_Bits(vector<int>& f) {
    cout<<"\nEnter the 8 bit binary: ";
    for(int i = 0; i<8; i++){
        cin>>f[i];
    }
    cout << "\nf: ";
    for(int i = 0; i<8; i++){
        cout<<f[i];
    }

}

void divide(vector<int>& f, vector<int>& f1, vector<int>& f2) {
    for (int i = 0; i < 4; i++) {
        f1.push_back(f[i]);
    }
    for (int i = 4; i < 8; i++) {
        f2.push_back(f[i]);
    }
    cout << "\nf1: ";
    for (int i = 0; i < f1.size(); i++) {
        cout << f1[i];
    }
    cout << "\nf2: ";
    for (int i = 0; i < f2.size(); i++) {
        cout << f2[i];
    }
}

void binaryAddition(vector<int>& f1, vector<int>& f2, vector<int>& result) {
    int carry = 0;
```

```
    for (int i = 3; i >= 0; i--) {
        int sum = f1[i] + f2[i] + carry;
        if (sum == 0) {
            result[i] = 0;
            carry = 0;
        } else if (sum == 1) {
            result[i] = 1;
            carry = 0;
        } else if (sum == 2) {
            result[i] = 0;
            carry = 1;
        } else { // sum == 3
            result[i] = 1;
            carry = 1;
        }
    }
}

void binaryComplement(vector<int>& Bin_result, vector<int>& Comp) {
    for(int i = 0; i<Bin_result.size(); i++){
        if(Bin_result[i] == 1){
            Bin_result[i] = 0;
        }
        else if(Bin_result[i] == 0){
            Bin_result[i] = 1;
        }
        Comp.push_back(Bin_result[i]);
    }
    cout << "\nComplement of Binary Addition: ";
    for (int i = 0; i < Comp.size(); i++) {
        cout << Comp[i];
    }
}

void CheckSumRecivers(vector<int>& f, vector<int>& Bin_result) {
    vector<int> Rf1;
    vector<int> Rf2;
    vector<int> Rf3;
    vector<int> RBin_result(4, 0);
    for (int i = 0; i < 4; i++) {
        Rf1.push_back(f[i]);
    }
    for (int i = 4; i < 8; i++) {
```

```
        Rf2.push_back(f[i]);
    }
    for (int i = 8; i < 12; i++) {
        Rf3.push_back(f[i]);
    }
    cout << "\nRf1: ";
    for (int i = 0; i < Rf1.size(); i++) {
        cout << Rf1[i];
    }
    cout << "\nRf2: ";
    for (int i = 0; i < Rf2.size(); i++) {
        cout << Rf2[i];
    }
    cout << "\nRf3: ";
    for (int i = 0; i < Rf3.size(); i++) {
        cout << Rf3[i];
    }
    binaryAddition(Rf1, Rf2, Bin_result);
    binaryAddition(Bin_result, Rf3, RBin_result);
    cout << "\nBinaryAddition: ";
    for (int i = 0; i < RBin_result.size(); i++) {
        cout << RBin_result[i];
    }
    int count = 0;
    int i = 0;
    for( ; i< RBin_result.size(); i++){
        if(RBin_result[i] == 1){
            count++;
        }
    }
    cout<<"\n\nFinal Output: ";
    if((count%2) == 0){
        cout<<"No error\n";
    }
    else{
        cout<<"Error\n";
    }
}

// Check Sum main function
void checkSum(vector<int>& data) {
     cout<<"Check Sum technique\n";
    vector<int> f(8);
```

```
    vector<int> f1;
    vector<int> f2;
    vector<int> Comp;
    accept_Bits(f); //accepts the 8 bits binary message
    divide(f, f1, f2); //Divides main binary message to 4 bits two frames
    vector<int> Bin_result(4, 0);
    cout<<"\n\nSender's Side calculations =>\n";
    binaryAddition(f1, f2, Bin_result);
    cout << "\nBinaryAddition: ";
    for (int i = 0; i < Bin_result.size(); i++) {
        cout << Bin_result[i];
    }
    binaryComplement(Bin_result, Comp);
    cout<<"\n";
    f.insert(f.end(), Comp.begin(), Comp.end());
    cout<<"Data + Complement: ";
    for(int i = 0; i<f.size(); i++){
        cout<<f[i];
    }
    cout<<"\n\nReciever's Side calculations =>\n";
    CheckSumRecivers(f, Bin_result);
}
//--------------------------------------------------------------------------------
------------------------
// CRC function
std::vector<int> xor_division(const std::vector<int> &dividend, const
std::vector<int> &divisor) {
    std::vector<int> remainder = dividend;

    for (int i = 0; i <= remainder.size() - divisor.size(); ++i) {
        if (remainder[i] == 1) {
            for (int j = 0; j < divisor.size(); ++j) {
                remainder[i + j] ^= divisor[j];
            }
        }
    }

    // Return the remainder after division
    return std::vector<int>(remainder.begin() + remainder.size() -
divisor.size(), remainder.end());
}

void printdata(vector<int>data)
```

```
{
    int n=data.size();
    for(int i=0;i<n;i++)
    {
        cout<<data[i];
    }
    cout<<endl;
}
void sender_crc(vector<int>&data,vector<int>&gx)
{
    //accepting data in binary form only
    cout<<"enter the data to transmit........"<<endl;
    int n;
    cout<<"enter the number of bits in data:";
    cin>>n;
    //taking input
    cout<<"start entering data bitwise"<<endl;
    for(int i=0;i<n;i++)
    {
        int temp;
        cin>>temp;
        data.push_back(temp);
    }

    //taking g(x) as input
    int n2;
    cout<<"enter number of bits in g(x):";
    cin>>n2;
    //taking input
    cout<<"start entering data bitwise"<<endl;
    for(int i=0;i<n2;i++)
    {
        int temp;
        cin>>temp;
        gx.push_back(temp);
    }

    //calculating number of zeros to be appended
    int nu_zero=n2;
    nu_zero--;

    //appending n zero's to the data(bits of gx)
    vector<int>datacopy;
```

```
    datacopy=data;
    for(int i=0;i<nu_zero;i++)
    {
        datacopy.push_back(0);
    }

    //xor division datacopy/gx
    vector<int> remainder = xor_division(datacopy, gx);
    //append remainder to data and then send that data
    for(int i=0;i<remainder.size();i++)
    {
        data.push_back(remainder[i]);
    }
}

void reciever_crc(vector<int>&data,vector<int>&gx)
{
    //xor division datacopy/gx
    vector<int> remainder = xor_division(data,gx);

    //calculating that remainder is zero or not
    bool iszero=true;

    for(int i=0;i<remainder.size();i++)
    {
        if(remainder[i]==1)
        {
            iszero=false;
        }
    }
    cout<<"printing remainder........."<<endl;
    printdata(remainder);

    if(iszero)
    {
        cout<<"data send.........(without error)";
    }else{
        cout<<"data send.........(with error)";
    }
}


int main() {
```

```
    int C;
    while (true) {
        cout << "\nMenu\n1. Parity check\n2. CheckSum\n3. CRC\n";
        cout << "\nEnter the choice: ";
        cin >> C;

        vector<int> data;
        vector<int> gx;

        switch (C) {
            case 1:
                sender_code_parity_check(data);
                transmit_P(data);
                reciever_code_parity_check(data);
                break;
            case 2:
                checkSum(data);
                break;
            case 3:
                sender_crc(data, gx);
                transmit_P(data);
                reciever_crc(data, gx);
                break;
            default:
                break;
        }
    }

    return 0;
}
```

Output:

```
Menu
1. Parity check
2. CheckSum
3. CRC

Enter the choice: 2
Check Sum technique

Enter the 8 bit binary: 1 0 1 1 1 0 0 1

f: 10111001
f1: 1011
f2: 1001

Sender's Side calculations =>

BinaryAddition: 0100
Complement of Binary Addition: 1011
Data + Complement: 101110011011

Reciever's Side calculations =>

Rf1: 1011
Rf2: 1001
Rf3: 1011
BinaryAddition: 1111

Final Output: No error

Menu
1. Parity check
2. CheckSum
3. CRC
```

Conclusion:

In this lab, we explored three important methods for error detection in digital communication: Parity Check, Checksum, and Cyclic Redundancy Check (CRC). Each of these methods plays a crucial role in ensuring data integrity during transmission. Here are the key takeaways from this lab:

- **Parity Check** is a straightforward method that involves adding an extra bit to data to make the total number of 1s (either even or odd) in the data and parity bit even or odd. It can detect single-bit errors but cannot correct them.
- **Checksums** are values calculated from data sets to verify data integrity. A checksum is appended to the data, and the recipient can recalculate it to check for errors. Checksums are commonly used in networking protocols to detect errors during data transmission.
- **Cyclic Redundancy Check (CRC)** is an advanced error-checking method that uses polynomial division. A generator polynomial is agreed upon by both sender and receiver. The sender appends a remainder obtained through polynomial division to the data. The receiver performs the same division and checks if the remainder is zero. CRCs are powerful and can detect a wide range of errors, including burst errors

VIT Vidyalankar Institute of Technology
Accredited A+ by NAAC

DEPARTMENT OF COMPUTER ENGINEERING
Computer Network Lab

Semester	T.E. Semester V – Computer Engineering
Subject	Computer Network
Subject Professor In-charge	Prof. Amit K. Nerurkar
Assisting Teachers	Prof. Amit K. Nerurkar
Laboratory	M-313-A

Student Name	Deep Salunkhe
Roll Number	21102A0014
TE Division	A

Title : Hamming Code

Theory:

Hamming codes are designed to detect and correct single-bit errors. The receiver checks the received code word for errors using the parity bits. If an error is detected, the receiver can pinpoint the erroneous bit and correct it. If multiple errors occur, Hamming codes may not be able to correct them.

Implementation:

```
#include<iostream>
#include<cmath>
#include<vector>
#include<cstdlib>
#include<ctime>

using namespace std;

// Function to find the number of redundant bits required (r)
void findr(int &r)
{
    for (int i = 0; i < 7; i++)
    {
        if (pow(2, i) >= 7 + i + 1)
        {
            r = i;
            break;
        }
    }
}

// Function to calculate the parity bits (R1, R2, R4, R8)
void fparity(int &R1, int &R2, int &R4, int &R8, vector<int> frame)
{
    // R1 family
    int p1 = 0;
    for (int i = 0; i <= 11; i = i + 2)
    {
        if (frame[i] == 1)
            p1++;
    }
```

```
    R1 = 1;
    if (p1 % 2 == 0)
        R1 = 0;

    // R2 Family
    int p2 = 0;
    if (frame[9] == 1)
        p2++;
    if (frame[8] == 1)
        p2++;
    if (frame[5] == 1)
        p2++;
    if (frame[4] == 1)
        p2++;
    if (frame[1] == 1)
        p2++;
    if (frame[0] == 1)
        p2++;
    R2 = 1;
    if (p2 % 2 == 0)
        R2 = 0;

    // R4 family
    int p4 = 0;
    for (int i = 4; i <= 7; i++)
    {
        if (frame[i] == 1)
            p4++;
    }
    R4 = 1;
    if (p4 % 2 == 0)
        R4 = 0;

    // R8 family
    int p8 = 0;
    for (int i = 0; i <= 3; i++)
    {
        if (frame[i] == 1)
            p8++;
    }
    R8 = 1;
    if (p8 % 2 == 0)
        R8 = 0;
```

```
    cout << "R1 R2 R4 R8" << endl;
    cout << R8 << " " << R4 << " " << R2 << " " << R1 << endl;
}

// Function to display received data without errors
void Noerror(vector<int> frame)
{
    cout << "Received Data: ";
    for (int i = 0; i < frame.size(); i++)
    {

        cout << frame[i] << " ";
    }
    cout << "\nData is error-free (OK)" << endl;

    int R1, R2, R4, R8;

    fparity(R1, R2, R4, R8, frame);
    frame[10] = R1; //R1
    frame[9] = R2;  //R2
    frame[7] = R4;  //R4
    frame[3] = R8;  //R8
    //parity bits

    cout << "As all the parities are 0" << endl;
}

// Function to simulate received data with errors
void Witherror(vector<int> frame)
{
    vector<int> itf = {0, 1, 2, 4, 5, 6, 8};

    int randn = rand() % 7;
    if (frame[itf[randn]] == 1)
        frame[itf[randn]] = 0;
    else
        frame[itf[randn]] = 1;

    cout << "Received Data with Errors: ";
    cout<< frame.size()<<endl;
    for (int i = 0; i < frame.size(); i++)
    {
```

```
        cout << frame[i] << " ";
    }
    cout<<endl;

    int R1, R2, R4, R8;
    fparity(R1, R2, R4, R8, frame);
    frame[10] = R1; //R1
    frame[9] = R2;  //R2
    frame[7] = R4;  //R4
    frame[3] = R8;  //R8




    // Finding and displaying the error position
    int error = 0;
    if (R8 == 1)
        error += 8;
    if (R4 == 1)
        error += 4;
    if (R2 == 1)
        error += 2;
    if (R1 == 1)
        error += 1;

    cout << "Error at bit position " << error <<"th bit  from end in frame "<<
endl;
}

// Function to send the data
void sender(vector<int> &data, int &r)
{
    cout << "Enter 7-bit data: ";
    for (int i = 0; i < 7; i++)
        cin >> data[i];
    findr(r);
    int fsize = 7 + r; // m + r
    vector<int> frame(fsize, 0);

    // Initialize parity bits to -1
    frame[10] = -1; //R1
    frame[9] = -1;  //R2
    frame[7] = -1;  //R4
```

```
    frame[3] = -1;  //R8

    // Fill the frame with data
    int id = 0; // Trace the data
    for (int i = 0; i < fsize; i++)
    {
        if (frame[i] == -1)
            continue;
        frame[i] = data[id];
        id++;
    }

    // Calculate and set the parity bits
    int R1, R2, R4, R8;
    fparity(R1, R2, R4, R8, frame);
    frame[10] = R1; //R1
    frame[9] = R2;  //R2
    frame[7] = R4;  //R4
    frame[3] = R8;  //R8

    cout << "Sending Data to user...." << endl;
    cout << "Sent Data: ";
    for (int i = 0; i < fsize; i++)
    {
        cout << frame[i] << " ";
    }




while(1)
{
    cout << "\nChoose an option:\n";
cout << "1. Send without error\n";
cout << "2. Send with error\n";
cout << "Enter your choice: ";
    int choice;
  cin >> choice;
    switch (choice) {
    case 1:
        Noerror(frame);
        break;
    case 2:
```

```
        Witherror(frame);
        break;
    default:
        cout << "Invalid choice." << endl;
        break;
}
}


}

int main()
{
    srand(time(0)); // Seed the random number generator
    vector<int> data(7);
    int r; // Number of redundant bits

    sender(data, r);

    return 0;
}
```

Output:

```
PS E:\GIt> cd "e:\GIt\SEM-5\CN\" ; if ($?) { g++ Hamming.cpp -o Hamm
Enter 7-bit data: 1 0 1 0 1 1 1
R1 R2 R4 R8
0 0 1 0
Sending Data to user....
Sent Data: 1 0 1 0 0 1 1 0 1 1 0
Choose an option:
1. Send without error
2. Send with error
Enter your choice: 2
Received Data with Errors: 11
1 0 1 0 0 0 1 0 1 1 0
R1 R2 R4 R8
0 1 1 0
Error at bit position 6th bit  from end in frame

Choose an option:
1. Send without error
2. Send with error
Enter your choice:
```

VIT Vidyalankar Institute of Technology
Accredited A+ by NAAC

DEPARTMENT OF COMPUTER ENGINEERING
Computer Network Lab

Semester	T.E. Semester V – Computer Engineering
Subject	Computer Network
Subject Professor In-charge	Prof. Amit K. Nerurkar
Assisting Teachers	Prof. Amit K. Nerurkar
Laboratory	Lab number

Student Name	Deep Salunkhe
Roll Number	21102A0014
TE Division	A

Title: Study of Networking commands

Implementation:

1. **Netstat** (Network Statistics):
   - Purpose: Displays network-related information, including active network connections, routing tables, interface statistics, and more.
   - Example: Display all listening ports on a Linux system.

```
C:\Users\student>netstat

Active Connections

  Proto  Local Address          Foreign Address        State
  TCP    172.16.136.91:51414    20.198.119.84:https    ESTABLISHED
  TCP    172.16.136.91:51464    104.17.187.189:https   CLOSE_WAIT
  TCP    172.16.136.91:51591    sa-in-f188:5228        ESTABLISHED
  TCP    172.16.136.91:51676    a23-212-254-41:https   LAST_ACK
  TCP    172.16.136.91:51677    a23-212-254-74:https   LAST_ACK
  TCP    172.16.136.91:51678    52.98.59.18:https      LAST_ACK
  TCP    172.16.136.91:51681    13.107.6.254:https     LAST_ACK
  TCP    172.16.136.91:51682    13.107.3.254:https     LAST_ACK
  TCP    172.16.136.91:51683    13.107.237.254:https   LAST_ACK
  TCP    172.16.136.91:51684    204.79.197.222:https   LAST_ACK
  TCP    172.16.136.91:51792    104.17.187.189:https   CLOSE_WAIT
  TCP    172.16.136.91:51805    104.17.187.189:https   TIME_WAIT
  TCP    172.16.136.91:51900    172.16.136.84:ms-do    ESTABLISHED
  TCP    172.16.136.91:51904    104.17.187.189:https   ESTABLISHED
  TCP    172.16.136.91:51910    104.17.187.189:https   ESTABLISHED
  TCP    172.16.136.91:51911    172.16.116.98:ms-do    ESTABLISHED
  TCP    172.16.136.91:51915    172.16.141.18:ms-do    ESTABLISHED
  TCP    172.16.136.91:51920    bom07s32-in-f14:https  TIME_WAIT
  TCP    172.16.136.91:51921    hkg12s10-in-f46:https  TIME_WAIT
  TCP    172.16.136.91:51923    20.44.229.112:https    ESTABLISHED
  TCP    172.16.136.91:51924    52.139.250.209:https   TIME_WAIT
  TCP    172.16.136.91:51927    1:https                TIME_WAIT
  TCP    172.16.136.91:51928    bom12s18-in-f2:https   TIME_WAIT
  TCP    172.16.136.91:51929    125.99.88.205:https    TIME_WAIT
  TCP    172.16.136.91:51930    125.99.88.205:https    TIME_WAIT
  TCP    172.16.136.91:51931    bom12s17-in-f1:https   TIME_WAIT
  TCP    172.16.136.91:51932    bom07s25-in-f22:https  TIME_WAIT
  TCP    172.16.136.91:51933    bom07s25-in-f22:https  TIME_WAIT
  TCP    172.16.136.91:51935    bom12s20-in-f4:https   TIME_WAIT
  TCP    172.16.136.91:51936    a23-212-254-74:https   ESTABLISHED
  TCP    172.16.136.91:51937    a23-212-254-74:https   ESTABLISHED
  TCP    172.16.136.91:51938    a23-212-254-74:https   ESTABLISHED
  TCP    172.16.136.91:51939    52.98.59.18:https      ESTABLISHED
  TCP    172.16.136.91:51940    a23-212-254-41:https   ESTABLISHED
```

Command: netstat -tuln

2. **TraceRoute** (Traceroute) :
- Purpose: Traces the route that packets take from one host to another, showing the IP addresses and response times of each hop.
- Example: Trace the route to google.com.

```
C:\Users\student>tracert google.com

Tracing route to google.com [142.250.192.46]
over a maximum of 30 hops:

  1    <1 ms    25 ms     4 ms  172.16.136.1
  2    <1 ms    <1 ms    <1 ms  172.16.0.1
  3     1 ms     2 ms     8 ms  125.99.106.137
  4     2 ms     1 ms     1 ms  192.168.210.129
  5     2 ms     2 ms     2 ms  192.168.27.34
  6     2 ms     *        1 ms  125.99.55.254
  7     1 ms     2 ms     1 ms  125.99.55.253
  8     3 ms     *        3 ms  125.99.55.163
  9     *        4 ms     4 ms  125.99.55.165
 10     3 ms     2 ms     3 ms  142.251.225.77
 11     2 ms     2 ms     2 ms  142.250.212.171
 12     1 ms     1 ms     2 ms  bom12s15-in-f14.1e100.net [142.250.192.46]

Trace complete.
```

Command: traceroute google.com

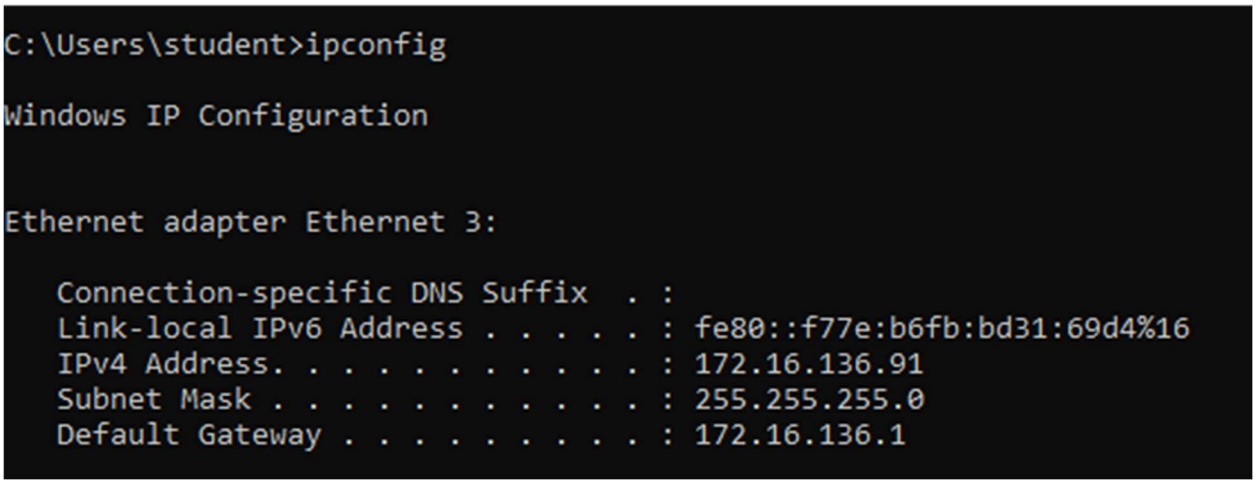
3. **Ifconfig**:
- Purpose: Used to configure and display information about network interfaces on Unix-like systems (Linux, macOS).
- Example: Display network information for the "eth0" interface on a Linux system.

Command: ifconfig

4. **IpConfig**:
- Purpose: Displays or configures IP-related information on Windows systems.
- Example: Display IP configuration information for all interfaces on a Windows system.

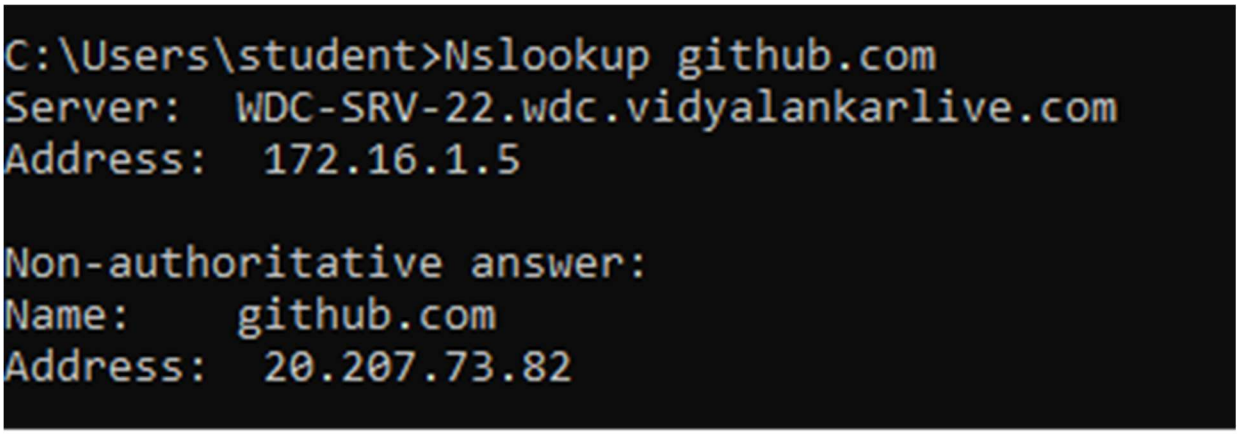
Command: ipconfig /all

5. **IpAddr**:
   - Purpose: Similar to `ifconfig`, used to show or manipulate IP addresses on Linux systems.
   - Example: Display IP address information for all interfaces on a Linux system.



Command: ip addr show

6. **Dig** (Domain Information Groper):
   - Purpose: Performs DNS queries to look up DNS records (A, MX, NS, etc.) for domain names.
   - Example: Look up the IP address of example.com using Dig.



Command: dig example.com

7. **Host**:
   - Purpose: Resolves domain names to IP addresses and vice versa.
   - Example: Resolve the IP address of google.com using the host command.

   Command: host google.com

8. **ARP** (Address Resolution Protocol):
   - Purpose: Maps an IP address to a physical MAC address on a local network.
   - Example: Display the ARP cache on a Windows system.

   Command: arp -a

```
C:\Users\student>arp -a

Interface: 172.16.136.91 --- 0x10
  Internet Address      Physical Address      Type
  172.16.136.1          64-9e-f3-64-4a-71     dynamic
  172.16.136.84         f4-4d-30-ab-fc-3b     dynamic
  172.16.136.90         f4-4d-30-ac-74-3e     dynamic
  172.16.136.99         f4-4d-30-ab-f7-ec     dynamic
  172.16.136.255        ff-ff-ff-ff-ff-ff     static
  224.0.0.22            01-00-5e-00-00-16     static
  224.0.0.251           01-00-5e-00-00-fb     static
  224.0.0.252           01-00-5e-00-00-fc     static
  239.255.255.250       01-00-5e-7f-ff-fa     static
```

9. **FTP** (File Transfer Protocol):
   - Purpose: A protocol for transferring files between a client and a server over a network.
   - Example: Connect to an FTP server and upload a file using FTP.

   Command: ftp ftp.example.com

10. **TelNet**:
   - Purpose: Allows remote terminal access to a device or server over a network.
   - Example: Connect to a remote server with IP address 192.168.1.100 on port 22 (SSH).

   Command: telnet 192.168.1.100 22

Differences between ifconfig , ipconfig, and ip addr:

- **ifconfig**(Unix/Linux):
  - Primarily used on Unix-like systems (Linux, macOS).
  - Displays and configures network interfaces.
- **ipconfig** (Windows):
  - Specific to Windows.
  - Displays and configures IP-related information.
- **ip addr** (Unix/Linux):
  - Similar to ipconfig on Windows.
  - Provides detailed IP address information on Linux systems.

End Result:

Conclusion: In summary, these commands are used for various networking tasks such as network configuration, diagnostics, DNS queries, and file transfer. The choice of command depends on your operating system and specific networking needs.

VIT Vidyalankar Institute of Technology
Accredited A+ by NAAC

DEPARTMENT OF COMPUTER ENGINEERING
Computer Network Lab

Semester	T.E. Semester V – Computer Engineering
Subject	Computer Network
Subject Professor In-charge	Prof. Amit K. Nerurkar
Assisting Teachers	Prof. Amit K. Nerurkar
Laboratory	M-313-A

Student Name	Deep Salunkhe
Roll Number	21102A0014
TE Division	A

Title : IP Addressing

Theory:

IP (Internet Protocol) addresses are numerical labels assigned to devices on a computer network that uses the Internet Protocol for communication. These addresses serve two primary functions:

1. **Host Identification**: IP addresses uniquely identify devices (hosts) on a network. Each device connected to a network, whether it's a computer, smartphone, or server, is assigned a distinct IP address.
2. **Location Addressing**: IP addresses also provide information about the device's location on the network. This allows routers and switches to determine how to forward data packets from the source to the destination.

IP Address Classes

IPv4 addresses are divided into different classes, each with its own range and purpose. The classes are:

1. **Class A**: IP addresses in the range 1.0.0.0 to 126.0.0.0 are designated as Class A addresses. They are typically used by large organizations and corporations.
2. **Class B**: IP addresses in the range 128.0.0.0 to 191.255.0.0 belong to Class B. These addresses are often assigned to medium-sized networks.
3. **Class C**: IP addresses in the range 192.0.0.0 to 223.255.255.0 fall into Class C. These addresses are commonly used for smaller networks.
4. **Class D**: Class D addresses (224.0.0.0 to 239.255.255.255) are reserved for multicast groups, where data is sent to multiple recipients simultaneously.
5. **Class E**: Class E addresses (240.0.0.0 to 255.255.255.255) are reserved for experimental and research purposes.

Validating IP Addresses

In your lab, you are tasked with creating a program to validate IP addresses. Valid IP addresses must adhere to the following rules:

- The address must consist of four numerical segments separated by periods (e.g., "192.168.0.1").
- Each segment must be between 0 and 255.
- Leading zeros in each segment are not allowed.
- The first segment must not be 0.
- The last segment must not be 0.

Determining IP Address Class

In addition to validation, your program will determine the class of the IP address based on its first segment:

- Class A: 1 to 126
- Class B: 128 to 191
- Class C: 192 to 223
- Class D: 224 to 239
- Class E: 240 to 255

Implementation:

```cpp
#include <iostream>
#include <bits/stdc++.h>
using namespace std;

void takeip(vector<int> &ip)
{
    cout << "enter ip without . :";
    int n;
    cout << "enter the number of group's':";
    cin >> n;
    cout << "start entering the input's......." << endl;
    for (int i = 0; i < n; i++)
    {
        int t;
        cin >> t;
        ip.push_back(t);
    }
}
bool isvalid(vector<int> &ip)
{
    // size
    int sz = ip.size();
    if (sz != 4)
    {
        return false;
    }
    // every group should [0,255]
    for (int i = 0; i < 4; i++)
    {
        int value = ip[i];
        if (!(value >= 0 && value < 256))
        {
            return false;
        }
    }
    // hex and binary
    return true;
}
```

```
int findgroup(vector<int> &ip)
{
    if (ip[0] >= 0 && ip[0] <= 127)
    {
        cout << "Belong to group A" << endl;
        return 0;
    }
    else if (ip[0] >= 128 && ip[0] <= 191)
    {
        cout << "Belong to group B" << endl;
        return 1;
    }
    else if (ip[0] >= 192 && ip[0] <= 223)
    {
        cout << "Belong to group C" << endl;
        return 2;
    }
    else if (ip[0] >= 224 && ip[0] <= 239)
    {
        cout << "Belong to group D" << endl;
        return 3;
    }
    else if (ip[0] >= 240 && ip[0] <= 255)
    {
        cout << "Belong to group E" << endl;
        return 4;
    }

    return 0;
}
void firstip(vector<int> &ip, int gid, vector<vector<int>> &cm)
{
    vector<int> fip;
    cout << "first ip is:" << endl;
    for (int i = 0; i < 4; i++)
    {
        int temp = ip[i] & cm[gid][i];
        fip.push_back(temp);
        cout << fip[i] << ".";
    }
}
void lastip(vector<int> &ip, int gid, vector<vector<int>> &cm)
{
```

```
    vector<int> lip;
    cout << "last ip is:" << endl;
    for (int i = 0; i < 4; i++)
    {
        int ones;
        if (cm[gid][i] == 255)
            ones = 0;
        else
            ones = 255;
        int temp = ip[i] | ones;
        lip.push_back(temp);
        cout << lip[i] << ".";
    }
}
int main()
{
    vector<int> ip;
    takeip(ip);
    vector<vector<int>> cm = {{255, 0, 0, 0}, {255, 255, 0, 0}, {255, 255, 255,
0}};
    if (isvalid(ip))
    {
        cout << "ip is valid......" << endl;
        // finding to which group it belong's
        int gid = findgroup(ip);
        if ((gid == 3 || gid == 4))
        {
            cout << "this is reversed class......" << endl;
        }
        else
        {
            firstip(ip, gid, cm);
            lastip(ip, gid, cm);
        }
    }
    else
    {
        cout << "ip is not valid......" << endl;
    }

    return 0;
}
```

Output:

```
PS E:\GIt> cd "e:\GIt\SEM-5\CN\" ; if ($?) { g++ Ipaddresing.cpp -o Ipaddresing } ; if ($?) { .\
enter ip without . :enter the number of group's':5
start entering the input's.......
12 12 12 12 12
ip is not valid......
PS E:\GIt\SEM-5\CN> cd "e:\GIt\SEM-5\CN\" ; if ($?) { g++ Ipaddresing.cpp -o Ipaddresing } ; if
enter ip without . :enter the number of group's':4
start entering the input's.......
12 12 12 12
ip is valid......
Belong to group A
first ip is:
12.0.0.0.last ip is:
12.255.255.255.
PS E:\GIt\SEM-5\CN>
```

Team members
(i) Deep Salunkhe (21102A0014)
(ii) Pranav Redij (21102A0005)
(iii) Omkar Patil (21102A0003)

PBLE-1

Batch ⇒ 1 Div ⇒ TE ENPN A

- Problem

→ Social media distraction.
→ unguided use of internet.
→ Investing time in playing games.
→ Addiction to quick dopamine due to comsumption of short content.

- Solution

→ Focus mode application/setting ⇒ Deavice is set to a certain duration usage of Application like social media, Netflix, etc. After due interval, warning popup occurs and Application closes.
→ Kids mode on browsing webpages ⇒ Browers ~~is~~ sets certain limitations to access unwanted webpages These access limitations can be controlled by user.
→ Blocker extension keeps watch on internet usage of particular user and this extension has parameter to set limitations on internet access. After due limit, internet ~~to~~ usage of that device closes.

- Tools

→ Qstudio ⇒ web filter / time managment tool.
→ Bark ⇒ social media monitering tool.
→ Mobicip ⇒ web filter / screen time tool.
→ NetBalancer ⇒ Internet usage Moniter.

PBLE-2

Names : Deep Salunkhe 21102A0014
Pranav Redij 21102A0005
Omkar Patil 21102A0003
DIV : TE CMPN A

ISP has ⇒ 245.248.128.0/20
Mask ⇒ 11111111 11111111 11110000 00000000

Total available address = 2^{12} = 4096

We need to provide 2048 to A
1024 to B
1024 to Mr A.

∴ For A ⇒

2^{11} = 2048
32 - 11 = 21
∴ Subnet mask ⇒ 11111111 11111111 11111000 00000000
255 255 248 0

First IP ⇒ 245 . 248 . 128 . 0
& 255 . 255 . 248 . 0
245 . 248 . 128 . 0

last IP ⇒ 245 . 248 . 128 . 0
OR 0 . 0 . 7 . 255 ← complement of mask
245 . 248 . 135 . 255

A ⇒ 245.248.128.0 to 245.248.135.255

For B ⇒
2^{10} = 1024
32 - 10 = 22
subnet mask ⇒ 11111111 11111111 11111100 00000000
255 255 252 0

Teacher's Signature:.....

PAGE No.
DATE / /

As the A ended on ~~255~~ 135.

First IP ⇒ 245 . 248 . 136 . 0
& 255 . 255 . 252 . 0
245 . 248 . 136 . 0

Ending IP ⇒ 245 . 248 . 138 . 0
or 0 . 0 . 3 . 255
245 . 248 . 139 . 255

∴ B ⇒ 245 . 248 . 136 . 0 to 245 . 248 . 139 . 255

For Mr A

Subnet mask = 255 . 255 . 252 . 0

First IP ⇒ 245 . 248 . 140 . 0
& 255 . 255 . 252 . 0
245 . 248 . 140 . 0

Ending IP ⇒ 245 . 248 . 140 . 0
or 0 . 0 . 3 . 255
245 . 248 . 143 . 255

∴ Mr A ⇒ 245 . 248 . 140 . 0 to 245 . 248 . 143 . 255

Teacher's Signature:

VIT Vidyalankar Institute of Technology
Accredited A+ by NAAC

DEPARTMENT OF COMPUTER ENGINEERING
Computer Network Lab

Semester	T.E. Semester V – Computer Engineering
Subject	Computer Network
Subject Professor In-charge	Prof. Amit K. Nerurkar
Assisting Teachers	Prof. Amit Alyani
Laboratory	313-A

Student Name	Deep Salunkhe
Roll Number	21102A0014
TE Division	A

Study Of Traffic Analysis tool's

1.WireShark

2.Liveactionomnipeek

3.etherape

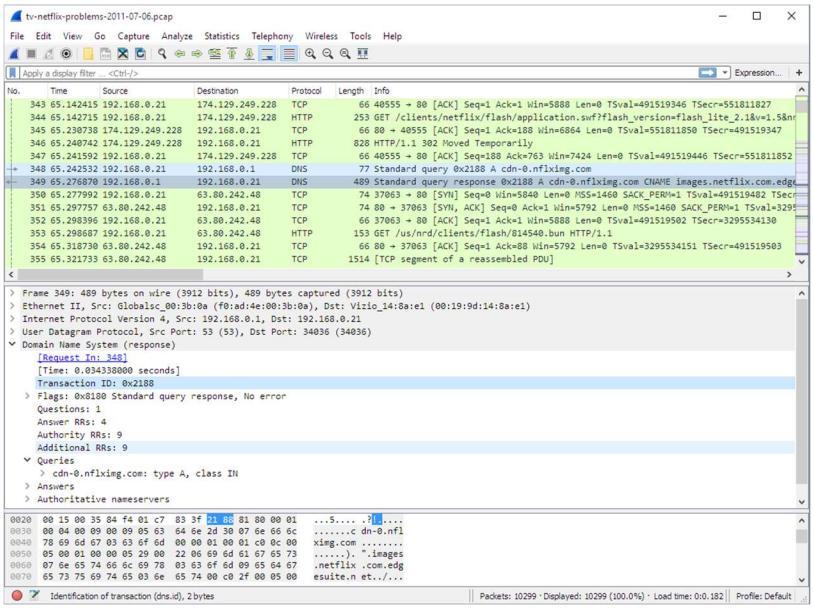
1.WireShark

1.1. What is Wireshark?

Wireshark is a network packet analyzer. A network packet analyzer presents captured packet data in as much detail as possible.

You could think of a network packet analyzer as a measuring device for examining what's happening inside a network cable, just like an electrician uses a voltmeter for examining what's happening inside an electric cable (but at a higher level, of course).

In the past, such tools were either very expensive, proprietary, or both. However, with



the advent of Wireshark, that has changed. Wireshark is available for free, is open source, and is one of the best packet analyzers available today.

2.LiveAction OmniPeek

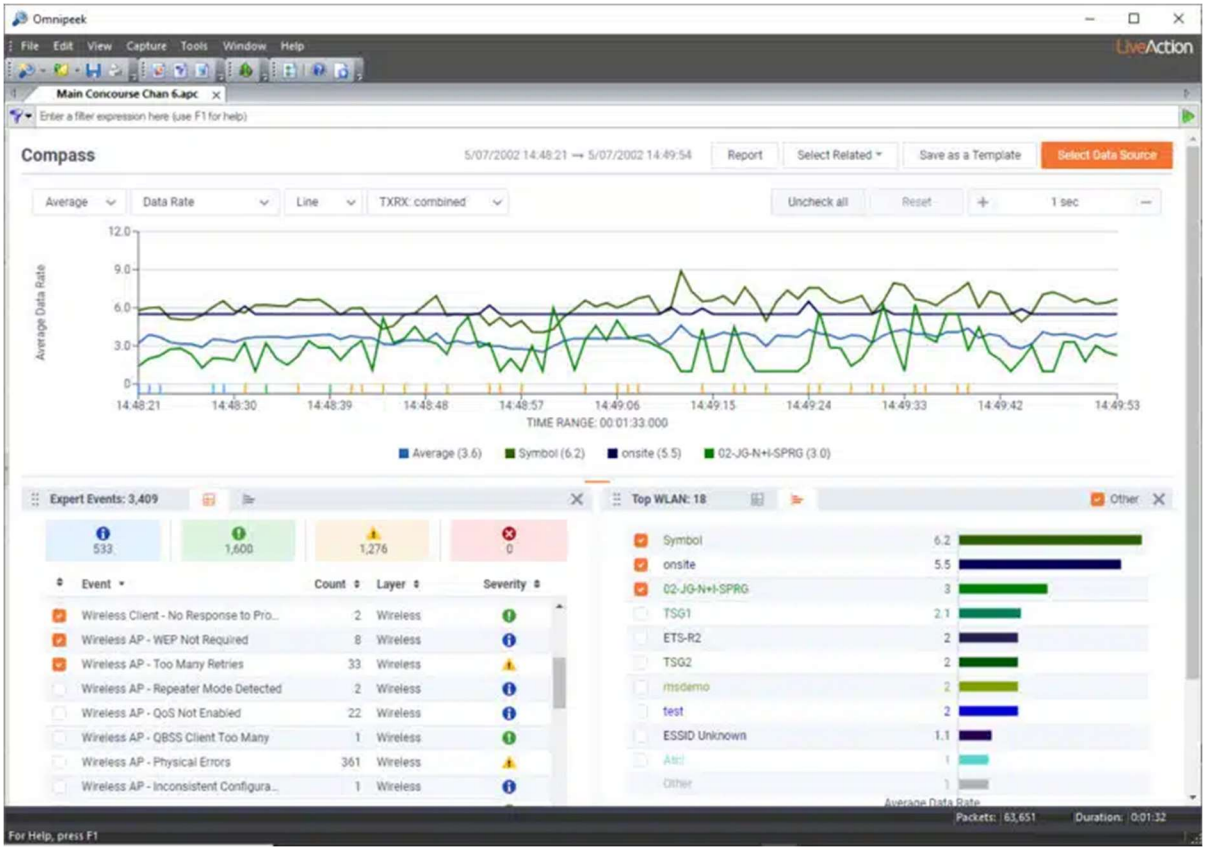
LiveAction OmniPeek is a sophisticated network analysis and monitoring tool used for in-depth examination of network traffic and performance. It allows network administrators and IT professionals to gain a comprehensive understanding of their network infrastructure through the following key features:

1. **Packet Capture and Analysis:** OmniPeek can capture data packets as they traverse the network. This includes Ethernet, Wi-Fi, and other protocols. It then provides a detailed breakdown of each packet, allowing users to inspect the contents, headers, and metadata.
2. **Real-Time Monitoring:** It offers real-time monitoring capabilities, providing live statistics on network utilization, bandwidth consumption, and other performance metrics. This helps in promptly identifying and addressing network bottlenecks and anomalies.
3. **Protocol Analysis:** OmniPeek supports a wide range of network protocols, including TCP/IP, HTTP, FTP, DNS, and more. Users can analyze the behavior of these protocols to pinpoint issues or security threats.
4. **Network Visualization:** The tool often provides visual representations of network traffic flows and patterns, making it easier to understand complex networks and spot irregularities.
5. **Traffic Filtering and Search:** Users can apply filters to isolate specific types of traffic or criteria, which is particularly useful for investigating issues or monitoring specific network activities.
6. **Expert Analysis:** OmniPeek includes expert analysis tools that automatically detect and flag potential network problems or security risks. It can also provide recommendations for remediation.

7. **Historical Analysis:** Beyond real-time monitoring, the tool allows users to review historical data, making it possible to trace network issues back in time to identify trends or recurring problems.

8. **Remote Packet Capture:** OmniPeek can capture network packets remotely from various locations, which is beneficial for analyzing distributed networks or troubleshooting issues in different parts of an organization.

9. **Reporting:** It offers reporting capabilities to document network performance and issues for further analysis or compliance purposes.



Overall, LiveAction OmniPeek is a robust network analysis tool that provides deep insights into network behavior and helps organizations maintain, optimize, and secure their networks effectively. It is a valuable asset for those responsible for managing and ensuring the reliability of complex computer networks.

3. Etherape

EtherApe is a graphical network monitoring tool used to visualize network traffic activity in real-time. Here are some key details about EtherApe:

1. **Real-Time Network Visualization:** EtherApe displays a dynamic and animated graph that represents network traffic flows between different hosts on a network. Each host is represented by a node, and the connections between them are depicted as lines. The size and color of nodes and lines correspond to the volume and type of traffic, allowing users to quickly identify the most active hosts and communication patterns.

2. **Protocols and Traffic Types:** EtherApe provides information about the types of traffic being transmitted on the network, such as TCP, UDP, ICMP, and more. It can also differentiate between various protocols and services, making it useful for identifying the sources and destinations of network traffic.

3. **Ease of Use:** This tool is designed with a user-friendly interface that makes it accessible to both novice and experienced users. The graphical representation simplifies the process of understanding network activity, making it easier to spot potential issues or anomalies.

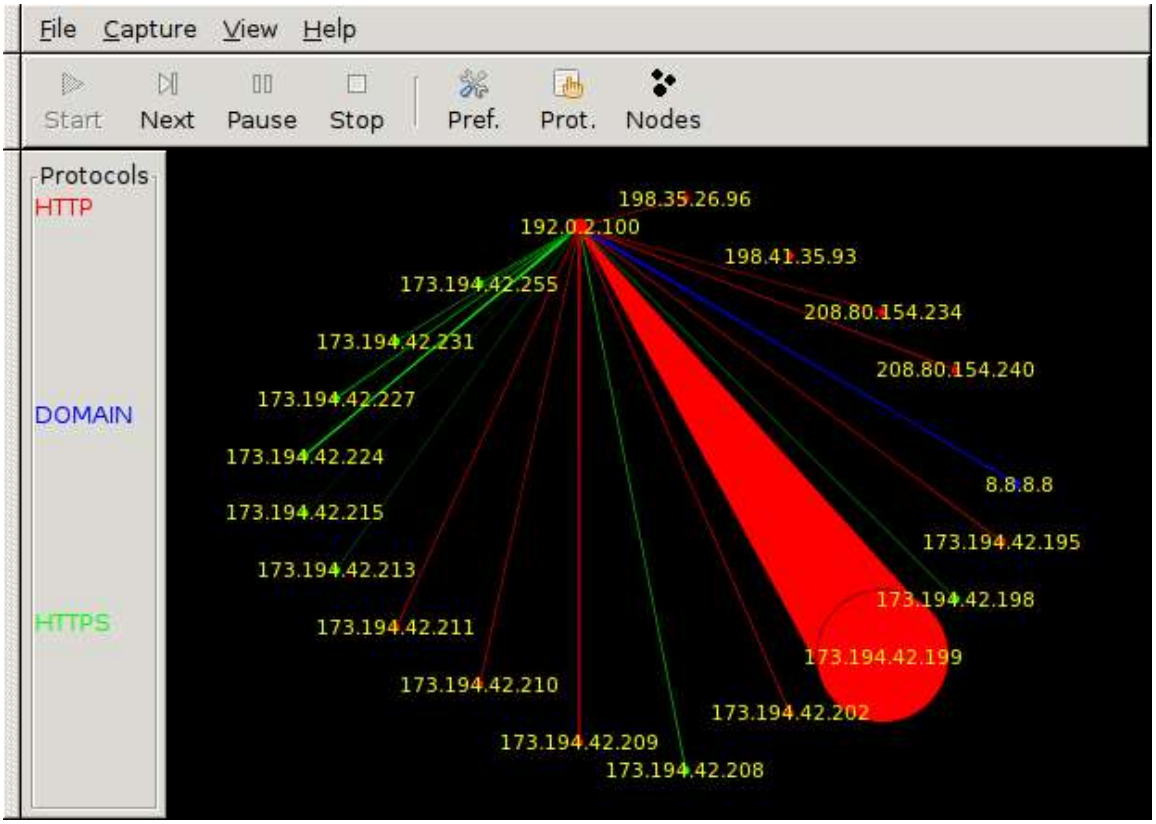
4. **Packet Capture:** While EtherApe primarily focuses on real-time network visualization, it also has the capability to capture network packets for more in-depth analysis when needed. Users can export captured packets for further examination using other tools.

5. **Cross-Platform Compatibility:** EtherApe is available for various operating systems, including Linux, Unix, and macOS. This cross-platform support makes it versatile and widely accessible.

6. **Open Source:** EtherApe is an open-source tool, which means it is freely available for anyone to use and modify. This fosters a community of developers who can contribute to its ongoing development and improvement.

7. **Network Troubleshooting:** Network administrators and IT professionals often use EtherApe for troubleshooting network performance issues, identifying

bandwidth hogs, and assessing network health. It can help pinpoint the source of congestion or unusual network behavior.



In summary, EtherApe is a real-time network monitoring and visualization tool that provides a visual representation of network traffic patterns. It is particularly valuable for network administrators and analysts who want to quickly understand the state of their networks, identify potential problems, and optimize network performance. Its user-friendly interface and open-source nature make it a popular choice among network professionals.

VIT Vidyalankar Institute of Technology
Accredited A+ by NAAC

DEPARTMENT OF COMPUTER ENGINEERING
Computer Network Lab

Semester	T.E. Semester V – Computer Engineering
Subject	Computer Network
Subject Professor In-charge	Prof. Amit K. Nerurkar
Assisting Teachers	Prof. Amit K. Nerurkar
Laboratory	Lab number

Student Name	Deep Salunkhe
Roll Number	21102A00014
TE Division	A

DEPARTMENT OF COMPUTER ENGINEERING
Computer Network Lab

Title: Two-Way Chat Application with TCP and UDP

Explanation:

Server-side:

- The server offers the user a choice between TCP and UDP.
- For TCP, it establishes a server socket, accepts client connections, and handles two-way communication.
- For UDP, it creates a datagram socket, receives messages from clients, and sends responses back.

Client-side:

- The client prompts the user to choose between TCP and UDP.
- For TCP, it connects to the server, spawns a thread to continuously receive messages, and allows the user to send messages.
- For UDP, it creates a datagram socket, spawns a thread to continuously receive messages, and allows the user to send messages.

The main differences between TCP (Transmission Control Protocol) and UDP (User Datagram Protocol) chats in the provided code lie in the characteristics of these protocols:

- **Connection-oriented vs. Connectionless:**
- **TCP:** It is a connection-oriented protocol. The server and client establish a connection before exchanging data. This ensures reliable, ordered, and error-checked delivery of information. The **ServerSocket** and **Socket** classes are used in Java for TCP communication.
- **UDP:** It is a connectionless protocol. Communication is achieved by sending independent packets, known as datagrams, to each other. UDP is faster but doesn't guarantee delivery, order, or error checking. The **DatagramSocket** and **DatagramPacket** classes are used in Java for UDP communication.
- **Reliability:**
- **TCP:** Reliable and ensures that data is received in the order it was sent. It also handles retransmission of lost packets and error detection.
- **UDP:** Unreliable, as it doesn't guarantee delivery, order, or error checking. It's often used in scenarios where a small amount of data loss is acceptable, such as real-time applications.
- **Overhead:**
- **TCP:** Higher overhead due to its reliability features and the need to establish and maintain a connection.
- **UDP:** Lower overhead since it's connectionless and doesn't include mechanisms for reliability.
- **Usage:**
- **TCP:** Suitable for applications where accurate and ordered delivery of data is crucial, such as file transfers, email, and web browsing.
- **UDP:** Used in scenarios where low latency and high-speed data transmission are more critical, such as video streaming, online gaming, and real-time communication.

Implementation:

Server-side:-

```
import java.io.*;
import java.net.*;

public class ChatServer {
    private static final int TCP_PORT = 12345;
    private static final int UDP_PORT = 12346;

    public static void main(String[] args) {
        System.out.println("Chat Server");

        try {
            BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));

            System.out.println("Choose the server type:");
            System.out.println("1. TCP Server");
            System.out.println("2. UDP Server");
            System.out.print("Enter your choice: ");

            int choice = Integer.parseInt(reader.readLine());

            switch (choice) {
                case 1:
                    startTCPServer();
                    break;
                case 2:
                    startUDPServer();
                    break;
                default:
                    System.out.println("Invalid choice. Please enter 1 or 2.");
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

```
    private static void startTCPServer() {
        try {
            ServerSocket serverSocket = new ServerSocket(TCP_PORT);
            System.out.println("TCP Server listening on port " + TCP_PORT);

            while (true) {
                Socket clientSocket = serverSocket.accept();
                System.out.println("TCP Client connected: " + clientSocket.getInetAddress());

                Thread clientThread = new Thread(() -> handleTCPClient(clientSocket));
                clientThread.start();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    private static void handleTCPClient(Socket clientSocket) {
        try {
            BufferedReader reader = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
            PrintWriter writer = new PrintWriter(clientSocket.getOutputStream(), true);

            BufferedReader consoleReader = new BufferedReader(new InputStreamReader(System.in));

            while (true) {
                String message = reader.readLine();
                if (message == null || message.equals("exit")) {
                    System.out.println("TCP Client disconnected: " + clientSocket.getInetAddress());
                    break;
                }

                System.out.println("TCP Received from " + clientSocket.getInetAddress() + ": " + message);

                System.out.print("Enter your response: ");
                String response = consoleReader.readLine();

                writer.println("Server: " + response);
            }

            clientSocket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

```
    private static void startUDPServer() {
        try {
            DatagramSocket serverSocket = new DatagramSocket(UDP_PORT);
            System.out.println("UDP Server listening on port " + UDP_PORT);

            while (true) {
                byte[] receiveData = new byte[1024];
                DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);
                serverSocket.receive(receivePacket);

                InetAddress clientAddress = receivePacket.getAddress();
                int clientPort = receivePacket.getPort();

                String message = new String(receivePacket.getData(), 0, receivePacket.getLength());
                System.out.println("UDP Received from " + clientAddress + ":" + clientPort + ": " + message);

                if (message.equals("exit")) {
                    System.out.println("UDP Client disconnected: " + clientAddress + ":" + clientPort);
                    continue;
                }

                String replyMessage = "Server: " + message;
                byte[] sendData = replyMessage.getBytes();
                DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length, clientAddress,
clientPort);
                serverSocket.send(sendPacket);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Client-side:-

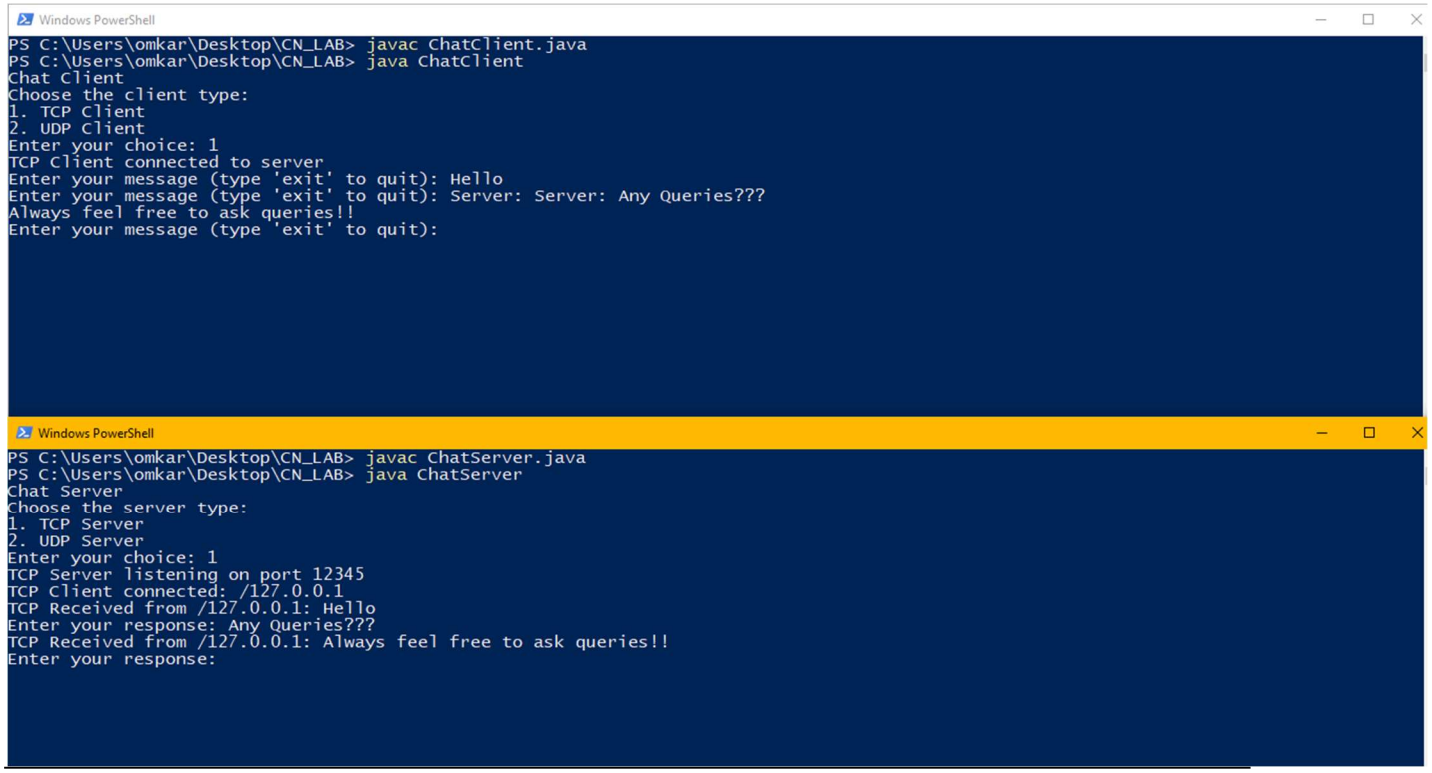
```
import java.io.*;
import java.net.*;

public class ChatClient {
    private static final int TCP_PORT = 12345;
    private static final int UDP_PORT = 12346;

    public static void main(String[] args) {
        System.out.println("Chat Client");

        try {
```

```
        BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));

        System.out.println("Choose the client type:");
        System.out.println("1. TCP Client");
        System.out.println("2. UDP Client");
        System.out.print("Enter your choice: ");

        int choice = Integer.parseInt(reader.readLine());

        switch (choice) {
            case 1:
                startTCPClient();
                break;
            case 2:
                startUDPClient();
                break;
            default:
                System.out.println("Invalid choice. Please enter 1 or 2.");
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}

private static void startTCPClient() {
    try {
        Socket socket = new Socket("localhost", TCP_PORT);
        System.out.println("TCP Client connected to server");

        BufferedReader serverReader = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
        PrintWriter writer = new PrintWriter(socket.getOutputStream(), true);
        BufferedReader consoleReader = new BufferedReader(new InputStreamReader(System.in));

        new Thread(() -> {
            try {
                while (true) {
                    String response = serverReader.readLine();
                    System.out.println("Server: " + response);
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        }).start();

        while (true) {
```

```
            System.out.print("Enter your message (type 'exit' to quit): ");
            String message = consoleReader.readLine();

            writer.println(message);

            if (message.equals("exit")) {
                break;
            }
        }

        socket.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}

private static void startUDPClient() {
    try {
        DatagramSocket socket = new DatagramSocket();
        InetAddress serverAddress = InetAddress.getByName("localhost");

        BufferedReader consoleReader = new BufferedReader(new InputStreamReader(System.in));

        new Thread(() -> {
            try {
                while (true) {
                    byte[] receiveData = new byte[1024];
                    DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);
                    socket.receive(receivePacket);

                    String response = new String(receivePacket.getData(), 0, receivePacket.getLength());
                    System.out.println("Server: " + response);
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        }).start();

        while (true) {
            System.out.print("Enter your message (type 'exit' to quit): ");
            String message = consoleReader.readLine();

            byte[] sendData = message.getBytes();
            DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length, serverAddress,
UDP_PORT);
            socket.send(sendPacket);
```

```
            if (message.equals("exit")) {
                break;
            }
        }

        socket.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
  }
}
```

End Result:



Conclusion:

The TCP chat implementation in the provided code showcases a reliable and connection-oriented communication model, ensuring ordered and error-checked message exchange. This makes it suitable for applications prioritizing data integrity, such as file transfers or text-based communication. In contrast, the UDP chat leverages a connectionless, low-overhead approach, offering faster data transmission but without guarantees of reliability or ordered delivery. The choice between TCP and UDP in a chat application depends on the specific requirements, balancing factors like message integrity and real-time responsiveness. The code provides a practical illustration of these fundamental differences in socket-based communication.