# Assignment :1

**Title:** Study of different types of Networking Commands used for analyzing and troubleshooting.

**Introduction:** In the realm of networking, understanding and mastering various networking commands are essential for analyzing and troubleshooting network-related issues effectively. These commands provide insights into network configurations, connectivity, performance, and security, enabling network administrators and engineers to diagnose and resolve problems efficiently. From basic tools like Ping and Traceroute to advanced utilities such as Netstat and SSH, each command serves a specific purpose in dissecting network behavior and identifying potential issues.

**Objective:** The objective is to equip individuals with the knowledge and skills to efficiently diagnose and resolve network problems. By mastering these commands, professionals can identify connectivity issues, analyze configurations, monitor performance, investigate security incidents, and securely access remote systems.

1. **Ping**: Ping is a fundamental tool used to test the reachability of a host on a network, assess network connectivity, and measure the round-trip time for packets sent from the source to the destination.

A screenshot of a computer program

Description automatically generated

1. **Traceroute (Tracert):** Traceroute, also known as Tracert in Windows, is employed to determine the route packets take to reach a destination, identify network hops, and diagnose potential points of failure or latency issues along the path.

A screenshot of a computer program

Description automatically generated

1. **ifconfig / ipconfig:** Ifconfig (on Unix-like systems) or ipconfig (on Windows) allows users to inspect the IP configuration of network interfaces, revealing crucial details such as IP addresses, subnet masks, and gateways, aiding in troubleshooting network connectivity problems.

A screenshot of a computer

Description automatically generated

1. **Netstat:** Netstat facilitates the display of active network connections, routing tables, and interface statistics, enabling the identification of network services and associated ports for analysis and troubleshooting purposes.

A screen shot of a computer

Description automatically generated

1. **Nslookup:** Nslookup serves as a tool for querying DNS servers to resolve hostnames to IP addresses, troubleshoot DNS resolution issues, and retrieve DNS-related information.

A black background with white text

Description automatically generated

1. **Route:** Route displays and modifies the IP routing table, assisting in determining routing paths for packets and troubleshooting routing and connectivity problems.

A screenshot of a computer program

Description automatically generated

1. **Arp:** Arp enables the display and modification of the ARP cache, facilitating the mapping of IP addresses to MAC addresses and troubleshooting ARP-related issues.

A screenshot of a computer program

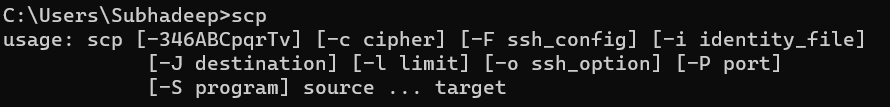
Description automatically generated

1. **SSH (Secure Shell):** SSH (Secure Shell) provides secure access to remote systems, enabling users to execute commands securely and transfer files securely between hosts.

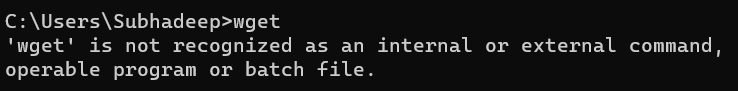
A screen shot of a computer program

Description automatically generated

1. **SCP (Secure Copy):** SCP (Secure Copy) allows for secure file transfers between local and remote hosts over SSH, ensuring data integrity and confidentiality.



1. **Wget (or Curl):** Wget or Curl enables the downloading of files from web servers over HTTP/HTTPS, testing web server availability, and retrieving content from URLs for scripting or automation tasks.



1. **Hostname:** The hostname command, used in computing, displays or sets the system's name, crucial for network identification and communication settings, facilitating efficient network operations and system configuration.

A black background with white text

Description automatically generated

**Conclusion:** Mastering networking commands is essential for effective network management. With these tools, professionals can swiftly diagnose issues, optimize performance, and ensure the security of network infrastructure, contributing to smoother operations and enhanced efficiency.

# Assignment : 2

**Title:** Write the steps of Network packet tracing using Wireshark.

**Introduction:** Wireshark is a powerful network protocol analyzer that enables real-time examination and

troubleshooting of network traffic. This guide will help you download Wireshark and utilize its capabilities for packet tracing, providing insights into network behavior.

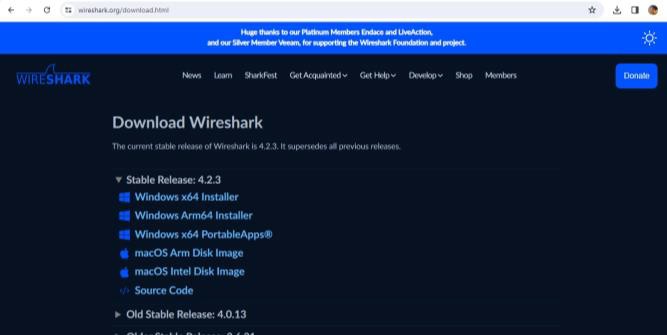
**Objective:** This guide aims to provide you with the knowledge and skills to efficiently download Wireshark, configure it to capture network traffic from different interfaces, analyze captured packets to understand communication

patterns, apply filters for focused analysis, and save captured data for future reference. By mastering these skills, you'll gain the ability to troubleshoot network issues, optimize performance, and enhance network security

effectively.

# Download Wireshark:

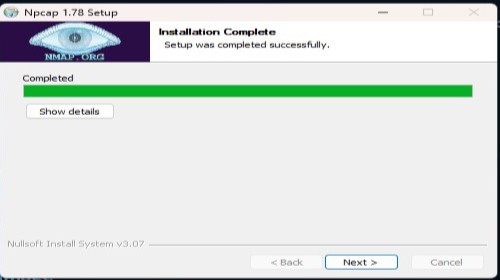
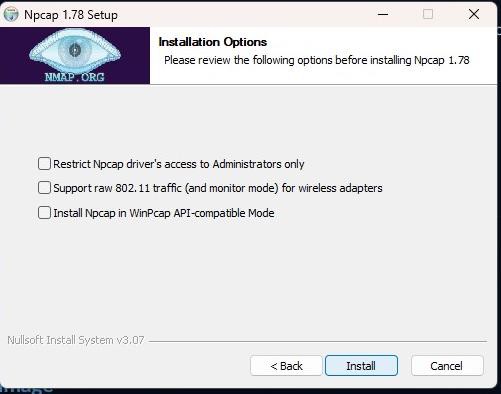
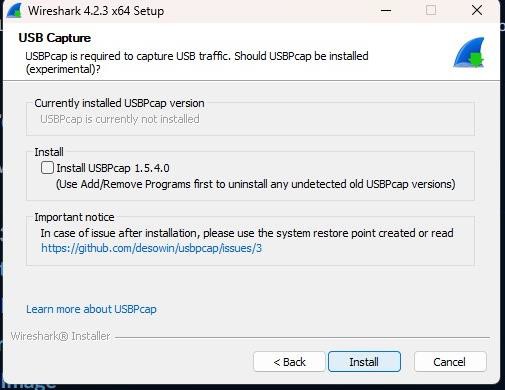
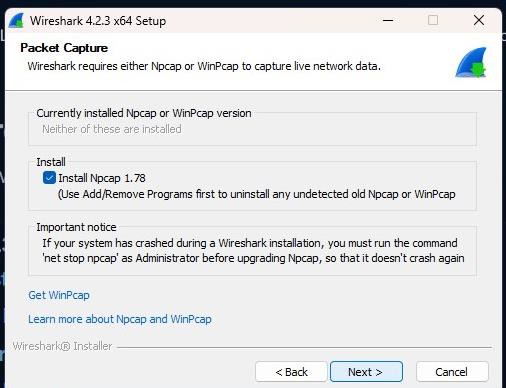
**Visit the Wireshark website:** Go to the official Wireshark website by typing the following URL into your web browser's address bar: [https://www.wireshark.org/.](https://www.wireshark.org/)



**Navigate to the Download section:** Once you're on the Wireshark website's homepage, look for the "Download" section. This is usually prominently displayed on the main page.

**Choose your operating system:** Wireshark is available for various operating systems including Windows, macOS, and Linux. Click on the appropriate download link for your operating system.

**Download the installer:** Click on the download link provided for your operating system. This will start the download of the Wireshark installer file. The file size may vary depending on your operating system and the version of

Wireshark you are downloading

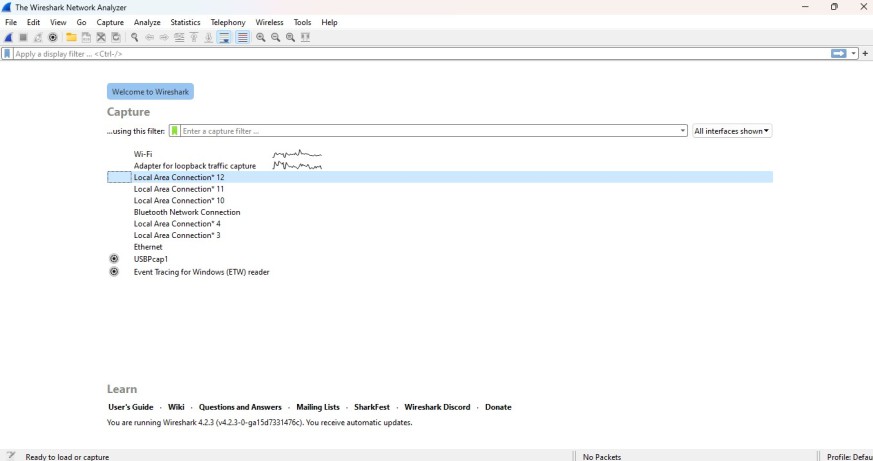
**Verify the download (optional):** It's a good practice to verify the integrity of the downloaded file, especially for security reasons. Some websites provide checksums or digital signatures for this purpose. Check the Wireshark website for any provided verification instructions.

**Run the installer:** Once the download is complete, locate the downloaded installer file on your system (usually in your Downloads folder) and double-click on it to run the installer.

**Follow the installation wizard:** The Wireshark installation wizard will guide you through the installation process. Follow the on-screen instructions, such as accepting the license agreement, choosing the installation directory, and selecting additional components if prompted.

**Complete the installation:** After you've configured the installation settings, proceed with the installation process by clicking on the "Install" or "Next" button. The installer will then proceed to install Wireshark on your system.

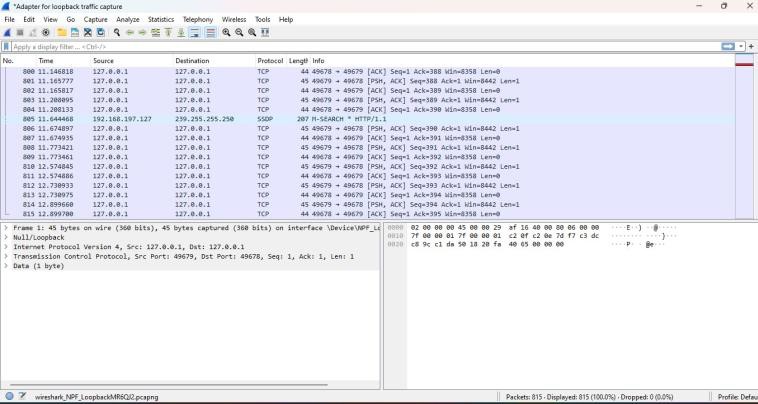
**Launch Wireshark:** After installation, launch Wireshark from the desktop shortcut or application menu.



# Select Network Interface:

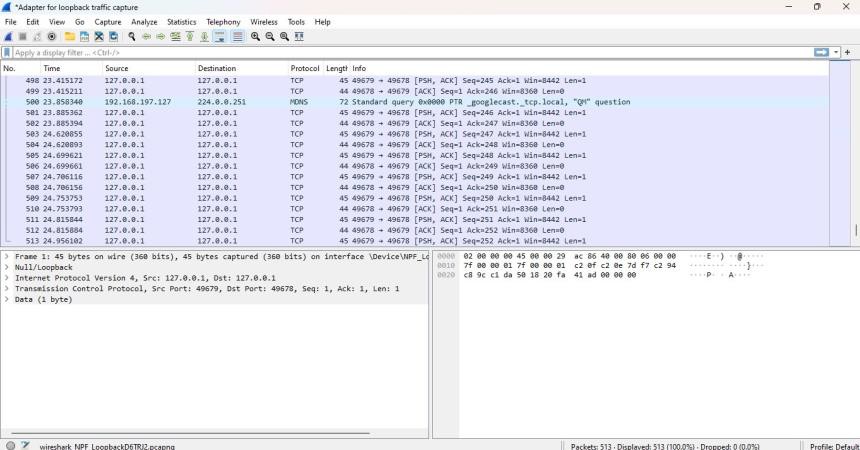
Upon launching, Wireshark will display a list of available network interfaces.

Choose the network interface through which you want to capture packets. This could be your Wi-Fi adapter, Ethernet adapter, etc.

**Start Capturing Packets:** Once the desired interface is selected, click on the "Start" or "Capture" button to begin capturing packets. Wireshark will start displaying captured packets in real-time.

**Analyze Packets:** You can analyze captured packets by examining various details such as source and destination IP addresses, protocols, packet size, etc. Use Wireshark's filter options to focus on specific types of packets or communication.

**Stop Capturing Packets:** When you've captured enough data or want to stop capturing, click on the "Stop" or "Capture" button.



**Save Captured Packets:** You can save the captured packets for later analysis by going to "File" > "Save As" and choosing a file format (e.g., pcap).

**Analyze Saved Packets:** If you've saved the captured packets, you can reopen them in Wireshark later for further analysis.

**Additional Features:** Explore Wireshark's various features such as protocol analysis, statistics generation, and packet decryption for encrypted traffic.

**Learn and Experiment:** Wireshark is a powerful tool with many features. Take some time to explore its capabilities, experiment with different settings, and learn how to interpret packet data effectively.

**Conclusion:** Wireshark is an essential tool for network analysis, offering valuable insights into network communication. By mastering its features, you can effectively troubleshoot issues, optimize performance, and enhance security. Embrace Wireshark to unlock the full potential of your network infrastructure.

# Assignment: 3

**Title:** Write a programming to implement Single server Socket Programming in Java.

**Introduction:** Socket programming is a fundamental aspect of network communication, allowing applications to communicate across a network using the Internet Protocol (IP). In this paradigm, sockets serve as the endpoints for communication, facilitating the exchange of data between different processes, either on the same machine or across the network. By utilizing sockets, developers can create versatile and powerful networked applications, ranging from simple client-server interactions to complex distributed systems.

**Objective:** The primary objective of socket programming is to enable communication between processes over a

network. This involves establishing connections, sending and receiving data, and handling various network protocols. By mastering socket programming, developers aim to create efficient, reliable, and scalable network applications that can fulfill a wide range of requirements, including real-time data exchange, remote procedure calls, and distributed computing tasks.

# Code:

**//Creating Server**

import java.io.\*;

import java.net.\*;

public class MyServer1 {

public static void main(String[] args){ try{

ServerSocket ss=new ServerSocket(6666); Socket s=ss.accept();//establishes connection

DataInputStream dis=new DataInputStream(s.getInputStream()); String str=(String)dis.readUTF();

System.out.println("message= "+str); ss.close();

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

} }

# //Creating Client

import java.io.\*;

import java.net.\*;

public class MyClient1 {

public static void main(String[] args) {

try{

Socket s=new Socket("localhost",6666);

DataOutputStream dout=new DataOutputStream(s.getOutputStream()); dout.writeUTF("Hello Server");

dout.flush();

dout.close();

s.close();

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

} }

# Output:

Server Output

A screen shot of a computer code

Description automatically generated

Client Output

A screen shot of a computer code

Description automatically generated

**Conclusion:** Socket programming plays a crucial role in modern networked applications, empowering developers to create dynamic and interconnected systems. By leveraging sockets, applications can communicate seamlessly across networks, enabling functionalities such as file transfer, remote command execution, and real-time data streaming.

Understanding socket programming principles equips developers with the tools to build robust and scalable networked solutions, facilitating efficient data exchange and collaboration in today's interconnected world.

# Assignment: 4

**Title:** Write a programming to implement Multithreading Socket programming Java

# Introduction:

Multithreaded socket programming is an advanced technique in network communication that utilizes multiple

threads to handle concurrent connections between clients and servers. In this approach, each client connection is managed by a separate thread, allowing the server to handle multiple clients simultaneously. Multithreaded socket programming enhances the scalability and responsiveness of networked applications by efficiently utilizing system resources and enabling parallel processing of client requests.

# Objective:

The primary objective of multithreaded socket programming is to improve the performance and scalability of networked applications by leveraging the concurrency capabilities of modern operating systems. By employing

multiple threads, developers aim to maximize the utilization of system resources and enhance the responsiveness of servers to client requests. The objective is to design efficient and scalable networked systems that can handle a large number of concurrent connections while maintaining high performance and reliability.

# Code:

**// Server class**

import java.io.\*;

import java.net.\*; class Server1 {

public static void main(String[] args) { ServerSocket server = null;

try {

server = new ServerSocket(1234); server.setReuseAddress(true);

while (true) {

Socket client = server.accept();

System.out.println("New client connected"+ client.getInetAddress() .getHostAddress()); ClientHandler clientSock = new ClientHandler(client);

new Thread(clientSock).start();

} }

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

finally {

if (server != null) { try {

server.close(); } catch (IOException e) {

e.printStackTrace();

} } } }

private static class ClientHandler implements Runnable { private final Socket clientSocket;

public ClientHandler(Socket socket) { this.clientSocket = socket; }

public void run() { PrintWriter out = null;

BufferedReader in = null; try {

out = new PrintWriter(clientSocket.getOutputStream(), true);

in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream())); String line;

while ((line = in.readLine()) != null) {

System.out.printf(" Sent from the client: %s\n",line); out.println(line);

} }

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

finally { try {

if (out != null) { out.close(); }

if (in != null) {

in.close();

clientSocket.close();

} }

catch (IOException e) { e.printStackTrace();

} } } } }

# // Client class

import java.io.\*;

import java.net.\*; import java.util.\*; class Client1 {

public static void main(String[] args) {

try (Socket socket = new Socket("localhost", 1234)) {

PrintWriter out = new PrintWriter(socket.getOutputStream(), true);

BufferedReader in= new BufferedReader(new InputStreamReader(socket.getInputStream())); Scanner sc = new Scanner(System.in);

String line = null;

while (!"exit".equalsIgnoreCase(line)) { line = sc.nextLine();

out.println(line); out.flush();

System.out.println("Server replied "+ in.readLine()); } sc.close(); }

catch (IOException e) { e.printStackTrace();

} } }

# Output:

Server Output

A screen shot of a computer

Description automatically generated

Client Output

A black screen with white text

Description automatically generated

A black screen with white text

Description automatically generated

# Conclusion:

Multithreaded socket programming offers a powerful solution for building high-performance networked applications capable of handling concurrent connections efficiently. By utilizing multiple threads, developers can create

responsive and scalable server architectures that can accommodate a large number of clients simultaneously. Multithreaded socket programming enables the development of robust and scalable networked systems that can meet the demands of modern applications, ranging from web servers to real-time communication platforms.