I. FTP

1. **What is FTP, and how does it work?**
   * FTP (File Transfer Protocol) is a protocol used to transfer files over a network. It operates on the client-server model and uses ports 20 and 21 for communication.
2. **What is vsftpd, and why is it commonly used?**
   * vsftpd stands for Very Secure FTP Daemon. It is a secure and fast FTP server for UNIX-like systems and is known for its stability and security features.
3. **How do you restrict an FTP user to their home directory?**
   * You can use chroot\_local\_user=YES in the vsftpd configuration file (/etc/vsftpd.conf) to restrict users to their home directory.
4. **How do you enable or disable anonymous FTP access in vsftpd?**
   * To disable anonymous FTP access, set anonymous\_enable=NO in the configuration file.
5. **What are passive and active FTP modes, and when would you use each?**
   * Active mode: The client opens a random port and listens, the server connects back.
   * Passive mode: The server opens a port and waits, and the client connects to that port. Passive mode is often used when the client is behind a firewall.
6. **How would you secure FTP on a server?**
   * Common methods include disabling anonymous login, restricting users to their directories, using firewall rules, limiting login attempts, and using SFTP (which runs over SSH) instead of FTP.
7. **What is the difference between FTP and SFTP?**
   * FTP transfers data in plaintext and is less secure. SFTP (SSH File Transfer Protocol) uses SSH for secure data transmission.
8. **How do you start, stop, and restart the vsftpd service?**
   * Use sudo systemctl start vsftpd, sudo systemctl stop vsftpd, and sudo systemctl restart vsftpd to manage the service.
9. **Which port does FTP use by default, and can it be changed?**
   * FTP uses port 21 by default. It can be changed by modifying the listen\_port directive in the vsftpd configuration file.
10. **How can you troubleshoot common FTP errors on Linux?**
    * Common steps include checking firewall rules, verifying configuration settings, checking vsftpd logs (/var/log/vsftpd.log), and ensuring the vsftpd service is active.

II. Telnet

1. **What is Telnet, and how does it work?**
   * Telnet is a network protocol that allows remote communication with servers. It operates over TCP, typically on port 23, and provides a command-line interface for managing servers.
2. **Is Telnet secure? Why or why not?**
   * No, Telnet is not secure because it transmits data, including login credentials, in plaintext. It’s vulnerable to interception and should not be used on untrusted networks.
3. **What is the default port for Telnet, and can it be changed?**
   * The default port for Telnet is 23. It can be changed by modifying the Telnet service configuration files.
4. **What are the main differences between Telnet and SSH?**
   * Telnet is unencrypted and insecure, while SSH (Secure Shell) is encrypted and provides secure remote access.
5. **How can you restrict access to the Telnet server?**
   * Access to the Telnet server can be restricted by using firewall rules to limit access to specific IPs and using TCP wrappers or configuration settings to limit access.
6. **What are the limitations of Telnet compared to modern remote access tools?**
   * Telnet lacks encryption and modern security protocols, making it vulnerable to eavesdropping. Modern tools like SSH provide secure communication with encryption.
7. **Why would you use Telnet in a controlled environment?**
   * Telnet can be used for testing network services, checking open ports, or accessing devices that do not support SSH in a secure, isolated network.
8. **How do you start, stop, and check the status of the Telnet server?**
   * Use sudo systemctl start inetd, sudo systemctl stop inetd, and sudo systemctl status inetd to manage the Telnet server.
9. **What are some practical use cases for Telnet today?**
   * Telnet can be useful for testing network connectivity, troubleshooting network services, and accessing legacy systems in secure environments.

III. Explore Wireshark Tool

Wireshark is a powerful open-source network protocol analyzer used for capturing and analyzing packets on a network in real-time. It’s widely used in network troubleshooting, security analysis, and learning about network protocols.

### **Key Features of Wireshark**

1. **Real-Time Packet Capture**: Capture live network traffic from various interfaces (e.g., Ethernet, Wi-Fi).
2. **Protocol Analysis**: Supports thousands of network protocols (TCP, UDP, HTTP, FTP, DNS, etc.).
3. **Filtering Capabilities**: Apply filters to isolate specific packets of interest.
4. **Packet Inspection**: View detailed information about each packet, including headers and payload.
5. **Data Export and Import**: Export data in multiple formats, including .pcap (Packet Capture) files.
6. **Visualization Tools**: Graph network traffic and create statistics for better insights.
7. **Decryption Support**: Decrypt encrypted traffic for protocols like SSL/TLS (with proper keys).

### **Basic Wireshark Filters**

* **IP Address**: ip.addr == 192.168.1.1
* **Protocol**: tcp, udp, http, dns
* **Port**: tcp.port == 80
* **MAC Address**: eth.addr == 00:0a:95:9d:68:16
* **Range of IP Addresses**: ip.src == 192.168.1.0/24
* **TCP Flags**: tcp.flags.syn == 1 (for SYN packets)

### **Viva Questions for Wireshark**

1. **What is Wireshark, and what is it primarily used for?**
   * Wireshark is a network protocol analyzer used for capturing and analyzing network packets.
2. **What are some common protocols that Wireshark can analyze?**
   * Wireshark supports TCP, UDP, HTTP, FTP, DNS, SSL/TLS, and many other protocols.
3. **How does Wireshark capture packets?**
   * Wireshark uses network interfaces to capture packets as they traverse the network. It requires root or administrator privileges to access interfaces directly.
4. **What is a display filter, and why is it useful in Wireshark?**
   * A display filter allows users to view specific packets by isolating traffic based on certain conditions, like IP addresses or protocols.
5. **What’s the difference between capture filters and display filters?**
   * Capture filters limit the packets that are stored in the capture file, while display filters limit the packets shown in the interface without affecting the capture file.
6. **Can Wireshark decrypt SSL/TLS traffic?**
   * Yes, Wireshark can decrypt SSL/TLS traffic if the encryption keys are available, although this is generally limited to certain use cases.
7. **Why might Wireshark be a security risk on a network?**
   * Wireshark can capture sensitive data (e.g., passwords) in plaintext if used on unsecured protocols, making it risky if misused by unauthorized users.
8. **How can Wireshark be used to detect malicious activities?**
   * Wireshark can help detect unusual traffic patterns, large volumes of traffic from specific IPs, or unexpected protocol usage.
9. **What are I/O graphs in Wireshark, and how are they useful?**
   * I/O graphs plot network traffic over time, helping users visualize spikes, patterns, and potential issues in data flow.
10. **What is packet reassembly in Wireshark, and why is it important?**
    * Packet reassembly helps combine fragmented packets, especially for protocols like TCP, to analyze the complete message.

### IV. Nmap

1. **What is Nmap, and what is it used for?**
   * Nmap is a network scanning tool used for discovering hosts, services, and vulnerabilities on a network.
2. **Explain the difference between TCP Connect Scan (-sT) and SYN Scan (-sS).**
   * TCP Connect Scan (-sT) completes the full TCP handshake, while SYN Scan (-sS) sends SYN packets and waits for SYN-ACK responses, making it faster and stealthier.
3. **What is the default port scanning range in Nmap?**
   * By default, Nmap scans the 1,000 most common ports. The full port range is 1–65535.
4. **How does Nmap identify operating systems on target hosts?**
   * Nmap uses fingerprinting techniques by analyzing responses to TCP/IP packets to identify OS characteristics.
5. **What are some common uses of the Nmap Scripting Engine (NSE)?**
   * NSE scripts can be used for vulnerability detection, malware scanning, service detection, and authentication brute-forcing.
6. **What does the -A option in Nmap do?**
   * The -A option enables aggressive scanning, which includes OS detection, version detection, and traceroute.
7. **What is a ping scan, and when would you use it?**
   * A ping scan (-sn) discovers which hosts are up without performing port scans, useful for basic network discovery.
8. **How can you limit an Nmap scan to specific ports?**
   * Use the -p option followed by the port numbers, such as -p 80,443 or -p 1-1000.
9. **Explain the importance of exporting scan results in multiple formats.**
   * Exporting in multiple formats (e.g., XML, grepable) allows easier integration with analysis tools and simplifies report generation.
10. **What are some limitations of using Nmap?**
    * Nmap can be detected by intrusion detection systems, and it’s limited in scanning well-protected or highly restricted networks.

V. Socket Programming in TCP

Socket programming in TCP (Transmission Control Protocol) is a method for creating applications that communicate over a network. TCP provides a reliable, connection-oriented service, which means it ensures data is delivered accurately and in the correct order.

### **Steps to Implement TCP Socket Programming**

### **1. TCP Server**

The server listens for incoming client connections, accepts them, and communicates over a reliable connection.

python

Copy code

# TCP Server Code

import socket

# Define host and port

host = '127.0.0.1' # Localhost

port = 12345

# Create a TCP socket

server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

# Bind the socket to a specific host and port

server\_socket.bind((host, port))

# Listen for incoming connections (1 client at a time in this case)

server\_socket.listen(1)

print(f"Server listening on {host}:{port}...")

# Accept a connection

conn, addr = server\_socket.accept()

print(f"Connected by {addr}")

# Communication loop

while True:

data = conn.recv(1024) # Receive data from client (buffer size = 1024 bytes)

if not data:

break

print(f"Received from client: {data.decode()}")

response = input("Enter response to client: ")

conn.sendall(response.encode()) # Send response to client

# Close the connection

conn.close()

### **2. TCP Client**

The client connects to the server, sends messages, and receives responses from the server.

python

Copy code

# TCP Client Code

import socket

# Define server address and port

server\_host = '127.0.0.1'

server\_port = 12345

# Create a TCP socket

client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

# Connect to the server

client\_socket.connect((server\_host, server\_port))

print(f"Connected to server at {server\_host}:{server\_port}")

# Communication loop

while True:

message = input("Enter message to send to server: ")

client\_socket.sendall(message.encode()) # Send data to server

# Receive response from server

data = client\_socket.recv(1024)

if not data:

break

print(f"Received from server: {data.decode()}")

# Close the connection

client\_socket.close()

### **Viva Questions for TCP Socket Programming**

1. **What is TCP, and why is it used in socket programming?**
   * TCP (Transmission Control Protocol) is a reliable, connection-oriented protocol that ensures data integrity by establishing a connection and verifying the data order.
2. **What’s the difference between TCP and UDP?**
   * TCP is connection-oriented, ensuring reliable data transfer, while UDP is connectionless, faster, but does not guarantee delivery or order.
3. **Explain the purpose of bind() and listen() functions in a TCP server.**
   * bind() binds the server to an IP address and port, while listen() allows the server to listen for incoming connections.
4. **What is the function of accept() in socket programming?**
   * accept() accepts a connection request from a client, returning a new socket for the connection and the client's address.
5. **What is the significance of the buffer size in the recv() function?**
   * The buffer size determines how much data (in bytes) can be received at once from the network.
6. **What happens if a TCP client tries to connect to a port where no server is listening?**
   * The connection attempt will fail, and an error will be raised, typically a "Connection Refused" error.
7. **How does TCP ensure reliable communication?**
   * TCP establishes a connection, manages data ordering, retransmits lost packets, and performs error checking to ensure reliable communication.
8. **Can you explain the three-way handshake in TCP?**
   * The three-way handshake establishes a TCP connection: the client sends a SYN request, the server responds with SYN-ACK, and the client replies with ACK, establishing the connection.
9. **What is the role of connect() in the TCP client code?**
   * connect() establishes a connection to the server at a specified IP address and port.
10. **Why do we need to close the socket at the end of communication?**
    * Closing the socket frees up system resources and formally ends the communication channel.

VI. Socket programming for UDP

### **1. UDP Server**

The UDP server listens on a specific port and receives datagrams (data packets) from clients. It does not need to establish a connection before receiving data.

python

Copy code

# UDP Server Code

import socket

# Define server host and port

host = '127.0.0.1' # Localhost

port = 12345

# Create a UDP socket

server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

# Bind the socket to the host and port

server\_socket.bind((host, port))

print(f"UDP server listening on {host}:{port}")

# Receive data from clients

while True:

data, client\_address = server\_socket.recvfrom(1024) # Buffer size is 1024 bytes

print(f"Received message from {client\_address}: {data.decode()}")

# Send response to the client

response = input("Enter response to client: ")

server\_socket.sendto(response.encode(), client\_address)

### **2. UDP Client**

The client can send data to the server without establishing a connection. It simply sends datagrams to the server's IP address and port.

python

Copy code

# UDP Client Code

import socket

# Define server address and port

server\_host = '127.0.0.1'

server\_port = 12345

# Create a UDP socket

client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

# Communication loop

while True:

message = input("Enter message to send to server: ")

client\_socket.sendto(message.encode(), (server\_host, server\_port)) # Send data to server

# Receive response from server

data, server\_address = client\_socket.recvfrom(1024) # Buffer size is 1024 bytes

print(f"Received from server: {data.decode()}")

# Close the socket

client\_socket.close()

### **Viva Questions for UDP Socket Programming**

1. **What is UDP, and how is it different from TCP?**
   * UDP (User Datagram Protocol) is a connectionless, unreliable protocol that does not guarantee packet delivery, order, or error checking. It’s faster than TCP but lacks reliability.
2. **Why might someone choose to use UDP over TCP?**
   * UDP is preferred in applications where speed is more important than reliability, such as live video streaming, gaming, and VoIP.
3. **Explain the purpose of recvfrom() in a UDP server.**
   * recvfrom() receives data from a client and returns both the data and the client's address, allowing the server to respond directly.
4. **How does sendto() function in UDP?**
   * sendto() sends data to a specified address (IP and port). It’s used for sending messages without establishing a connection.
5. **What happens if a UDP packet is lost during transmission?**
   * UDP does not retransmit lost packets; the packet is simply dropped. It’s up to the application to handle any lost data if needed.
6. **How does a UDP server handle multiple clients?**
   * Since UDP is connectionless, a single UDP server can receive packets from multiple clients without explicitly handling individual connections.
7. **What is the maximum buffer size that can be set in recvfrom()?**
   * The maximum buffer size is typically 65,535 bytes, but it’s common to use a smaller size, like 1024 bytes, depending on the application requirements.
8. **What is the significance of a “connectionless” protocol in networking?**
   * In a connectionless protocol, data is sent without establishing a persistent connection. Each packet can be sent independently, which reduces overhead but sacrifices reliability.
9. **Can a UDP server send data to a client without first receiving data from it?**
   * Generally, a UDP server responds to a client after receiving data, but it can also send unsolicited data to any client IP and port if known.
10. **In what scenarios would you use UDP instead of TCP?**
    * UDP is ideal for real-time applications like streaming, where low latency is crucial, and minor data loss is tolerable.

VII. Graphically Represent nodes using NS2 for any routing protocol (stop n wait /sliding window)

1. **Install NS2**:

sudo apt-get install ns2

### **Tcl Script (Save as sliding\_window\_simulation.tcl)**

# Define the simulator object

set ns [new Simulator]

# Define output trace file for analysis

set tracefile [open sliding\_window.tr w]

$ns trace-all $tracefile

# Define NAM (Network Animator) output file for visualization

set namfile [open sliding\_window.nam w]

$ns namtrace-all $namfile

# Set parameters for the simulation

set window\_size 4 ;# Sliding window size

set packet\_size 512 ;# Size of each packet in bytes

set num\_packets 10 ;# Number of packets to be sent

set interval 0.05 ;# Time interval between packet sends

# Create nodes

set n0 [$ns node]

set n1 [$ns node]

# Define the link between nodes (link from n0 to n1)

$ns duplex-link $n0 $n1 1Mb 10ms DropTail

# Define a UDP agent for sender (n0)

set udp0 [new Agent/UDP]

$ns attach-agent $n0 $udp0

# Attach a Sliding Window application to the UDP agent

set sliding\_app [new Application/Traffic/Exponential]

$sliding\_app set packetSize\_ $packet\_size

$sliding\_app set burst\_time\_ $interval

$sliding\_app set idle\_time\_ $interval

$udp0 attach-app $sliding\_app

# Define a Null agent for the receiver (n1)

set null0 [new Agent/Null]

$ns attach-agent $n1 $null0

# Connect the sender (UDP) and receiver (Null) agents

$ns connect $udp0 $null0

# Define a procedure to simulate sliding window protocol

proc send\_packets {ns udp num\_packets window\_size interval} {

for {set i 0} {$i < $num\_packets} {incr i} {

# Check if we are within window size

if {$i < $window\_size} {

# Schedule packet sending

$ns at [expr $i \* $interval] "$udp send"

} else {

# Wait for acknowledgment before sending next packet

$ns at [expr $i \* $interval + 0.1] "puts Packet $i waiting for ACK"

}

}

}

# Schedule the packet sending process using Sliding Window

send\_packets $ns $udp0 $num\_packets $window\_size $interval

# Simulation ends at 1 second

$ns at 1.0 "finish"

# Finish procedure to close trace and NAM files

proc finish {} {

global ns tracefile namfile

$ns flush-trace

close $tracefile

close $namfile

exec nam sliding\_window.nam &

exit 0

}

# Run the simulation

$ns run

1. Run the script using NS2  
   ns sliding\_window\_simulation.tcl  
   nam sliding\_window.nam