UNIVERSITY OF GONDAR
INSTITUTE OF TECHNOLOGY
DEPARTMENT OF COMPUTER ENGINEERING
OPERATING SYSTEM GROUP ASSIGNMENT

Group Members

Name	**ID**
1. BEIWKETU ATEK……....	23164/13
2. AGEGNEHU MEQUANINT….....	40177/13
3. HAWARIYAW PAULOS……...	40877/13
4. YEABISRA WORKU.....…	40778/13
5. ERMIAS LEMESA …….	23911/13

TABLE OF CONTENT

1. Network Programming

1.1 TCP/IP AND UDP

Network programming is the process of developing software that allows computers, servers, and other devices to communicate over a network, such as the Internet or a local network (LAN). It involves using specialized programming techniques to send and receive data between machines, ensuring efficient and reliable communication.

Key Aspects of Network Programming

1. Communication Between Devices
    - Network programming enables data exchange between devices using networking protocols like TCP/IP and UDP.
    - This is essential for applications such as web browsing, online gaming, video streaming, and cloud computing.
2. Use of Protocols
    - A protocol is a set of rules that defines how data is formatted and transmitted.
    - Common protocols in network programming:
        - TCP (Transmission Control Protocol) – Reliable, connection-oriented communication.
        - UDP (User Datagram Protocol) – Fast, connectionless communication.
        - HTTP/HTTPS – Used for web-based communication.
        - FTP (File Transfer Protocol) – Used for transferring files between computers.
3. Client-Server Model
    - Most network programs follow a client-server architecture, where:
        - The client sends requests (e.g., a web browser requesting a webpage).
        - The server processes the request and sends a response (e.g., a web server delivering the requested page).
4. Socket Programming
    - Sockets are the endpoints of network communication.
    - Developers use socket programming to create applications that send and receive data over a network.
    - Example:
        - A client socket connects to a server socket using IP address and port number.
        - The two devices can then exchange messages.
5. Common Applications of Network Programming
    - Web applications (browsers, APIs, cloud services).
    - Messaging apps (WhatsApp, Telegram).
    - Online gaming (real-time multiplayer games).
    - File sharing (FTP, P2P networks like BitTorrent).

- Remote access tools (SSH, VPN).

1.2 Importance of Communication Protocols in Network Programming

Communication protocols are essential in network programming as they define the rules and standards for how data is transmitted, received, and processed between devices over a network. Without these protocols, computers and networked devices would not be able to communicate effectively.

Key Reasons Why Communication Protocols Are Important

1. Ensures Reliable Data Transfer
    - Protocols like TCP (Transmission Control Protocol) ensure that data is delivered accurately and in order, preventing data loss or corruption.
    - Example: When sending an email, TCP ensures the entire message is received correctly.
2. Enables Device Interoperability
    - Protocols provide a common language that allows devices from different manufacturers and systems to communicate.
    - Example: A Windows computer can access a website hosted on a Linux server because both follow HTTP/HTTPS protocols.
3. Facilitates Efficient Communication
    - Protocols like UDP (User Datagram Protocol) allow fast and lightweight data transfer for applications that prioritize speed over reliability, such as live streaming and online gaming.
4. Standardizes Network Communication
    - Protocols follow internationally recognized standards (OSI Model, TCP/IP Model), making networking scalable and consistent across the globe.
5. Enables Security and Encryption
    - Secure protocols like HTTPS, SSL/TLS, and SSH ensure encrypted communication, protecting data from cyber threats and unauthorized access.
    - Example: Online banking uses HTTPS to safeguard transactions.
6. Supports Different Network Services
    - Different protocols serve different purposes in network programming:
        - HTTP/HTTPS → Web browsing
        - FTP → File transfer
        - SMTP/IMAP/POP3 → Email communication

- DNS → Domain name resolution

of TCP/IP

1.3 TCP/IP

The TCP/IP (Transmission Control Protocol / Internet Protocol) model is the fundamental framework for internet communication. It defines how data is transmitted, addressed, routed, and received across networks.

TCP:

Reliable Data Transfer – Ensures complete and error-free transmission using acknowledgment (ACK).
Connection-Oriented – Establishes a connection using a three-way handshake before data transfer.
Error Checking & Correction – Retransmits lost or corrupted packets.
Ordered Delivery – Ensures that packets arrive in sequence.
Flow Control – Manages data transmission speed to prevent congestion.

Example Use Cases of TCP:

- Web Browsing (HTTP/HTTPS) – Ensures complete webpage loading.
- Email Services (SMTP, IMAP, POP3) – Prevents message loss.
- File Transfers (FTP, SFTP) – Ensures file integrity.

l (IP) – Addressing and Routing

IP is responsible for addressing and routing data packets across networks. It ensures that data reaches the correct destination using IP addresses.

Key Features of IP:

Addressing – Assigns a unique IP address to each device for identification.
Routing – Determines the best path for data packets to travel from source to destination.
Connectionless Protocol – Does not establish a direct connection before sending data.
Fragmentation & Reassembly – Breaks large packets into smaller ones for transmission.

Types of IP Addresses:

- IPv4 (Internet Protocol Version 4) – Uses 32-bit addresses (e.g., 192.168.1.1).
- IPv6 (Internet Protocol Version 6) – Uses 128-bit addresses for a larger address space.

Example Use Cases of IP:

- Assigning addresses to devices on a local network (LAN/WAN).

- Enabling communication over the internet by routing packets between networks.
- Used in all online services like Google, YouTube, and cloud computing.

or different types of applications.1.2

TCP/IP in Action: How They Work Together

When you visit a website:

1. IP handles addressing & routing → Finds the web server's IP address and routes data.
2. TCP ensures reliable transmission → Establishes a connection and delivers complete, error-free data.

Analogy: TCP is like a postal tracking service, ensuring a package reaches its destination intact, while IP is like the postal system, delivering the package to the correct address.

TCP server and client using Python's socket module:

```
import socket

# Define server parameters

HOST = '127.0.0.1'  # Localhost

PORT = 12345        # Port to listen on


# Create a TCP socket

server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

server_socket.bind((HOST, PORT))  # Bind to the address

server_socket.listen(1)  # Allow 1 connection at a time


print(f"Server listening on {HOST}:{PORT}...")

# Accept a connection

conn, addr = server_socket.accept()

print(f"Connected by {addr}")


# Receive data from client

data = conn.recv(1024).decode()

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


# Send a response
```

```
response = "Hello from server!"
conn.sendall(response.encode())

# Close connection
conn.close()
server_socket.close()
```

TCP Client

This script connects to the server, sends data, and receives a response.

```
import socket

# Define server parameters
HOST = '127.0.0.1'  # Server address (localhost)
PORT = 12345        # Server port

# Create a TCP socket
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client_socket.connect((HOST, PORT))  # Connect to the server

# Send a message
message = "Hello from client!"
client_socket.sendall(message.encode())

# Receive response from server
data = client_socket.recv(1024).decode()
print(f"Received from server: {data}")

# Close connection
client_socket.close()
```

1.4 UDP

The User Datagram Protocol (UDP) is a connectionless, fast, but less reliable transport protocol used in network communication. Unlike TCP, UDP does not establish a connection before sending data and does not guarantee delivery, order, or error correction. This makes it ideal for applications where speed is more important than reliability.

Key Features of UDP

Connectionless Communication

- Does not establish a connection before sending data.
- Each packet (datagram) is sent independently without acknowledgment.

Faster Transmission

- Since UDP does not check for errors or retransmit lost packets, it is much faster than TCP.

No Order Guarantee

- Packets may arrive out of order or be dropped without retransmission.

Minimal Overhead

- UDP has a simple header (only 8 bytes) compared to TCP’s larger, more complex header.

Supports Broadcast & Multicast

- Can send data to multiple devices at once, making it ideal for real-time applications.

Example Use Cases of UDP

Live Streaming & VoIP (Voice over IP)

- Used in video calls (Zoom, Skype, Google Meet) and audio streaming.
- If a packet is lost, it's better to skip it than to delay the conversation.

Online Gaming

- Used in multiplayer games to send real-time position updates.
- Small delays are acceptable; waiting for lost packets would cause lag.

DNS (Domain Name System)

- DNS uses UDP to quickly resolve domain names into IP addresses.
- Speed is critical, and lost packets can be re-requested if needed.

IoT & Sensor Data

- Many Internet of Things (IoT) devices use UDP for lightweight communication.

comparison Between UDP and TCP

A Quick Comparison

Feature	UDP (User Datagram Protocol)	TCP (Transmission Control Protocol)
Connection	Connectionless	Connection-oriented (3-way handshake)
Speed	Faster (low overhead)	Slower (error checking, retransmission)
Reliability	Less reliable (no guarantee of delivery)	Reliable (ensures delivery & order)
Use Cases	Streaming, gaming, DNS, IoT	Web browsing, file transfers, emails
Packet Ordering	No guarantee	Ensures packets arrive in order
Error Handling	No error correction	Checks for errors & retransmits lost packets

TCP and UDP

Both TCP (Transmission Control Protocol) and UDP (User Datagram Protocol) are widely used in network communication, but they serve different purposes based on speed, reliability, and data integrity. Below are some common use cases for each protocol:

Use Cases of TCP (Reliable, Connection-Oriented Protocol)

TCP is used in applications that require accurate, complete, and ordered data transmission.

1. Web Browsing (HTTP/HTTPS)
   - When you load a webpage, TCP ensures that all text, images, and scripts arrive correctly.
   - TCP guarantees that web pages do not load with missing or corrupted content.

2. File Transfers (FTP, SFTP)
   - When downloading or uploading files, TCP ensures that no data is lost or corrupted.
   - Used in File Transfer Protocol (FTP) and Secure File Transfer Protocol (SFTP).

3. Emails (SMTP, IMAP, POP3)
   - TCP ensures email messages are delivered completely and in the correct order.
   - SMTP (Simple Mail Transfer Protocol) is used for sending emails, while IMAP/POP3 is used for receiving.

4. Remote Access (SSH, Telnet, RDP)
   - Used in Secure Shell (SSH) and Remote Desktop Protocol (RDP) to securely connect to remote systems.

- TCP ensures every command and response is transmitted reliably.

5. Online Transactions (E-Commerce, Banking Apps)
   - When making online payments, TCP ensures the transaction is processed correctly.
   - Used in secure payment gateways (SSL/TLS over TCP).

Use Cases of UDP (Fast, Connectionless Protocol)

UDP is used when speed is more important than reliability, such as in real-time applications.

1. Video Streaming (YouTube, Netflix, Twitch, IPTV)
   - UDP allows continuous data flow for streaming videos.
   - Dropped packets are ignored rather than retransmitted to avoid buffering.

2. Online Gaming (Multiplayer Games: PUBG, Call of Duty, Valorant, etc.)
   - UDP ensures low latency for real-time game actions.
   - Dropped packets are not retransmitted to prevent lag or delays.

3. Voice & Video Calls (VoIP, Zoom, Skype, Google Meet)
   - UDP is used in Voice over IP (VoIP) applications where real-time communication is required.
   - Small packet losses are ignored rather than retransmitted to prevent call delays.

4. DNS (Domain Name System) – Fast Website Lookup
   - When you enter a website URL, your system sends a UDP request to a DNS server to resolve the domain name into an IP address.
   - UDP ensures fast response times without the overhead of TCP.

5. IoT (Internet of Things) and Sensor Networks
   - Many IoT devices use UDP because they need to send quick, lightweight data packets without the need for reliability.
   - Example: Smart home devices, weather sensors, and GPS trackers.

TCP vs. UDP – Choosing the Right Protocol

Feature	TCP (Reliable & Connection-Oriented)	UDP (Fast & Connectionless)
Reliability	High (ensures complete data delivery)	Low (packets may be lost)
Speed	Slower due to error checking and retransmission	Faster due to minimal overhead

Feature	TCP (Reliable & Connection-Oriented)	UDP (Fast & Connectionless)
Data Ordering	Ensures packets arrive in order	No guarantee of order
Use Cases	Web browsing, emails, file transfers, banking transactions	Streaming, gaming, VoIP, DNS, IoT
Packet Loss Handling	Retransmits lost packets	Ignores lost packets

integrity and accuracy are paramount, such as web browsing, email, and file transfers.

Key characteristics of TCP/IP include: detect and correct errors in transmitted data.

UDP Server

This script listens for incoming UDP packets, receives data, and sends a response.

```
import socket

# Define server parameters
HOST = "127.0.0.1"  # Localhost
PORT = 12345        # Port to listen on

# Create a UDP socket
server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
server_socket.bind((HOST, PORT))

print(f"UDP Server listening on {HOST}:{PORT}...")

# Receive data from client
data, client_address = server_socket.recvfrom(1024)  # Buffer size is 1024 bytes
print(f"Received from client {client_address}: {data.decode()}")

# Send a response back to the client
response = "Hello from UDP server!"
server_socket.sendto(response.encode(), client_address)
```

```
# Close the socket
server_socket.close()
```

UDP Client

This script sends data to the server and waits for a response.

```
import socket

# Define server parameters
HOST = "127.0.0.1"  # Server address
PORT = 12345        # Server port

# Create a UDP socket
client_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

# Send a message to the server
message = "Hello from UDP client!"
client_socket.sendto(message.encode(), (HOST, PORT))

# Receive response from server
data, server_address = client_socket.recvfrom(1024)
print(f"Received from server: {data.decode()}")

# Close the socketclient_socket.close()
```

y characteristics of UDP include:**Low Latency**: Faster data transmission due to minimal protocol overhead.

No Connection Setup: Data is sent immediately without establishing a connection.

No Guaranteed Delivery: Packets may be lost, duplicated, or arrive out

1.4 Importance in Network Programming

Understanding TCP/IP and UDP is essential for network programmers, as these protocols form the backbone of most networked applications. Whether you're building a web server, a video streaming service, or a multiplayer online game, choosing the right protocol can significantly impact the performance and user experience of your application.