1. What is a Network?

- A **network** is a group of computers or devices connected to share **data** and **resources**.
- Example:
  - **Internet**: A global network.
  - **Local Area Network (LAN)**: A network within a home or office.

---

2. Types of Networks

- **LAN (Local Area Network)**: Small networks like in homes, schools, or offices.
- **WAN (Wide Area Network)**: Large networks spread over long distances (e.g., Internet).
- **MAN (Metropolitan Area Network)**: Covers a city or large campus.

---

3. What are Protocols?

- **Definition**: Protocols are **rules** that govern how data is sent and received over a network.
- Examples:
  - **HTTP**: For web browsing.
  - **FTP**: For file transfers.
  - **SMTP**: For sending emails.
  - **TCP/IP**: Core protocols of the Internet.

---

1. OSI and TCP/IP Models

OSI Reference Model (7 Layers)

- **Definition**: OSI (Open Systems Interconnection) is a standardized network architecture model (ISO 7498). It has **7 layers**, dividing communication tasks into manageable pieces.

7 Layers of OSI Model

Layer	Function
Layer 7: Application	Provides services directly to user applications (e.g., file transfer, email).
Layer 6: Presentation	Translates, compresses, and encrypts data into a standard format.
Layer 5: Session	Manages and controls connections between applications (e.g., dialog control).
Layer 4: Transport	Ensures reliable data delivery; handles packet segmentation and error recovery.
Layer 3: Network	Routes data packets using logical addressing (e.g., IP). Handles congestion.
Layer 2: Data Link	Converts bits into frames and ensures reliable transmission using

Layer	Function
	acknowledgments.
Layer 1: Physical	Transfers raw bits over the physical medium (e.g., cables, voltage).

Key Points:

- **Upper Layers (5-7)**: Focus on user application interaction.
- **Lower Layers (1-4)**: Focus on transmission and networking.

TCP/IP Model

- A simplified model used for the **Internet**.
- Combines OSI layers into **4 layers**:

TCP/IP Layer	OSI Equivalent	Example Protocols
Application	Application, Presentation, Session	HTTP, FTP, SMTP, DNS
Transport	Transport	TCP, UDP
Internet	Network	IP, ICMP, ARP
Link (Network Access)	Data Link, Physical	Ethernet, WiFi, PPP

Encapsulation: Data moves through layers; each layer adds its own **header**.

- **Order**: Message → Segment → Datagram → Frame → Bits.

---

2. Socket Programming

What is Socket Programming?

- **Socket**: Acts as a **door** between an application process and the transport layer.
- **Definition**: A socket allows two processes (on the same or different systems) to communicate.

Types of Sockets

1. **TCP Socket** (Connection-Oriented):
    - Reliable, ordered, and error-checked data transfer.
    - Requires **connection establishment** (handshake).
    - Example Protocol: **HTTP**.
2. **UDP Socket** (Connectionless):
    - Unreliable, faster, no connection setup.
    - Data may be lost or arrive out of order.
    - Example Protocol: **DNS**.

Socket Programming Example (Python)

UDP Client

```
python

from socket import *
serverName = 'hostname'
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_DGRAM)
message = input('Input message:')
clientSocket.sendto(message.encode(), (serverName, serverPort))
modifiedMessage, serverAddress = clientSocket.recvfrom(2048)
print(modifiedMessage.decode())
clientSocket.close()
```

UDP Server

```
python

from socket import *
serverPort = 12000
serverSocket = socket(AF_INET, SOCK_DGRAM)
serverSocket.bind(('', serverPort))
print("The server is ready to receive")
while True:
    message, clientAddress = serverSocket.recvfrom(2048)
    modifiedMessage = message.decode().upper()
    serverSocket.sendto(modifiedMessage.encode(), clientAddress)
```

TCP Client

```
python

from socket import *
serverName = 'hostname'
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName, serverPort))
sentence = input('Input message:')
clientSocket.send(sentence.encode())
modifiedSentence = clientSocket.recv(1024)
print('From Server:', modifiedSentence.decode())
clientSocket.close()
```

TCP Server

```
python

from socket import *
serverPort = 12000
serverSocket = socket(AF_INET, SOCK_STREAM)
serverSocket.bind(('', serverPort))
serverSocket.listen(1)
print("The server is ready to receive")
while True:
    connectionSocket, addr = serverSocket.accept()
    sentence = connectionSocket.recv(1024).decode()
    capitalizedSentence = sentence.upper()
    connectionSocket.send(capitalizedSentence.encode())
    connectionSocket.close()
```

3. Services and Connections

1. **Connection-Oriented vs Connectionless**
    - **Connection-Oriented** (e.g., TCP): Reliable, like a telephone call.
    - **Connectionless** (e.g., UDP): Unreliable, like sending letters in a postal system.
2. **Reliable vs Unreliable Services**
    - **Reliable**: Guarantees delivery without errors (e.g., file transfer).
    - **Unreliable**: Tolerates some loss (e.g., voice calls).

4. Network Applications

Examples of network applications include:

- **HTTP**: Web services.
- **SMTP**: Email.
- **FTP**: File transfer.
- **DNS**: Domain name resolution.
- **P2P File Sharing**: Direct file transfer between peers (e.g., BitTorrent).

Key Points to Remember

1. **OSI vs TCP/IP**: OSI is theoretical; TCP/IP is practical and used for the Internet.
2. **Encapsulation**: Data passes through layers, gaining headers at each step.
3. **Sockets**: Enable communication between processes via TCP (reliable) or UDP (faster but unreliable).
4. **Python Sockets**: Understand TCP/UDP client-server code for basic programs.

1. Why Layering?

- **Purpose**: Breaks down complex network systems into modular, manageable pieces.
- **Benefits**:
    - Makes **maintenance and updates** easier.
    - Changes in one layer do **not affect** others (modularity).
    - Provides a clear structure for discussion and understanding.

2. Addressing Processes

- To identify a process uniquely:
    - Use a combination of **IP Address** (identifies the host) and **Port Number** (identifies the process).

- Example:
    - **HTTP Server** uses Port **80**.
    - **Mail Server (SMTP)** uses Port **25**.

---

3. Client-Server vs Peer-to-Peer Architecture

- **Client-Server Model**:
    - A central server provides services to multiple clients.
    - Examples: **HTTP, IMAP, FTP**.
- **Peer-to-Peer (P2P)**:
    - Peers (end systems) both request and provide services.
    - Examples: **BitTorrent** for file sharing.

---

4. Transport Layer Protocols (TCP vs UDP)

Feature	TCP	UDP
Reliability	Reliable, ensures error-free data	Unreliable, data loss possible
Connection	Connection-oriented	Connectionless
Speed	Slower due to error checking	Faster, minimal overhead
Usage	File transfer, emails, web pages	Streaming, DNS, video calls