Advanced Programming
CSE 201

Instructor: Sambuddho

(Semester: Monsoon 2025)

Week 11 - Networking and Network Programming

What is a Computer Network?

- Interconnected devices sharing data/resources
  - Types: LAN, MAN, WAN, Internet
  - Wired vs Wireless networks

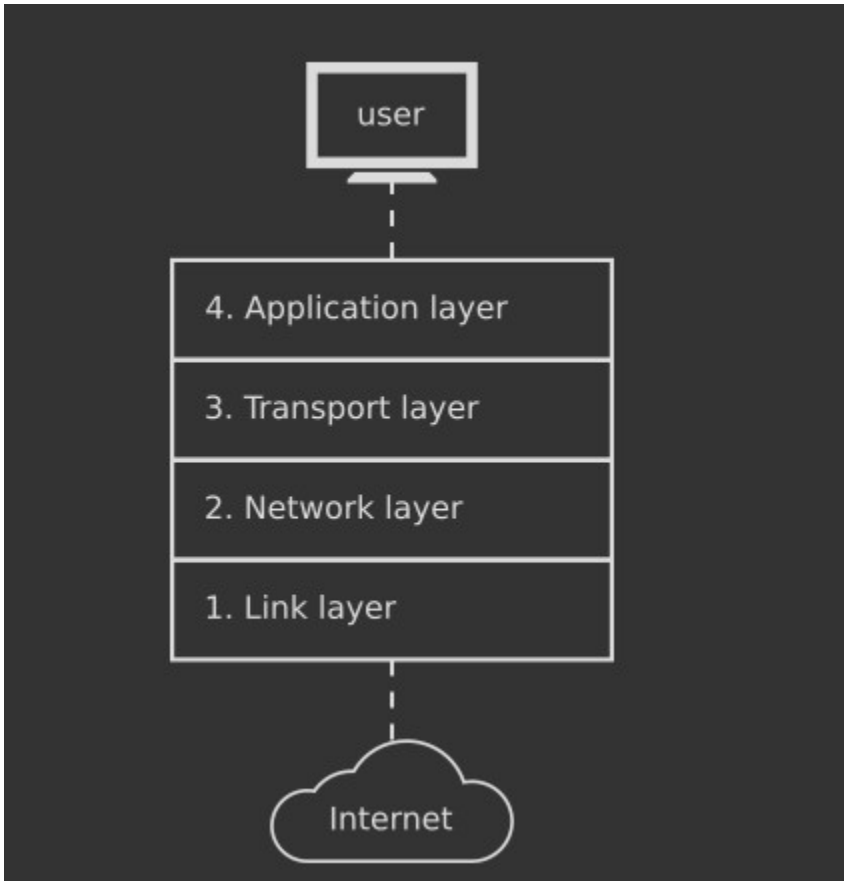
TCP/IP Model - 5 Layers

- Physical, Data Link, Network, Transport, Application
  - OSI vs TCP/IP (TCP/IP is used in practice)
  - Security issues can occur at each layer

Data Encapsulation & Decapsulation

- Data moves through layers and headers get added
  - Application → Transport → Network → Data Link → Physical
  - Reverse happens at the receiver side

Layered Architecture (TCP/IP)



TCP/IP model	Protocols and services
Application	HTTP, FTTP, Telnet, NTP, DHCP, PING
Transport	TCP, UDP
Network	IP, ARP, ICMP, IGMP
Network Interface	Ethernet

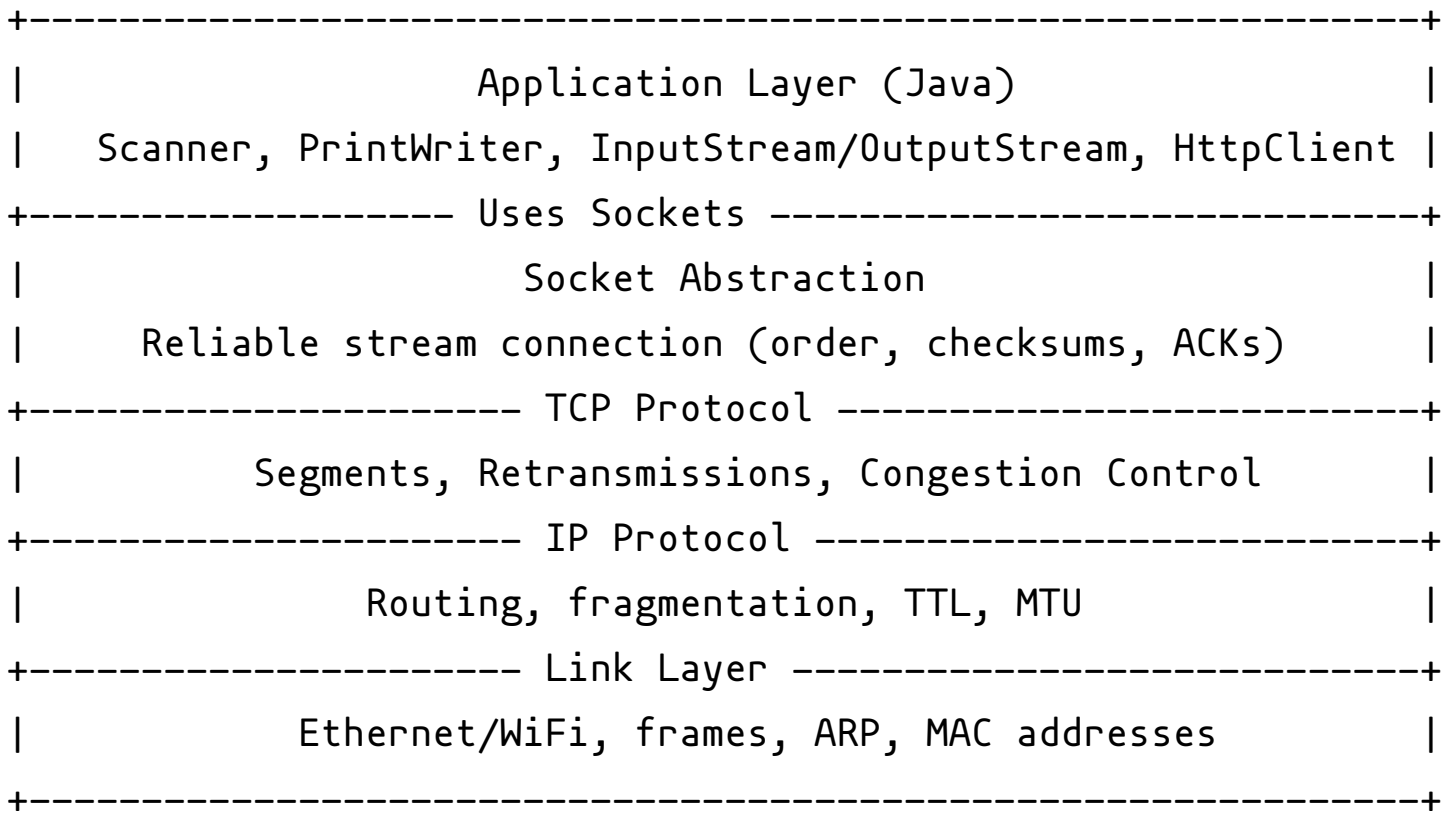
Layered Architecture (TCP/IP): L1 (MAC / Phy-MAC layer)

- Medium Access Control (*Not* Message Authentication Code!).
- 48-bit address of the network card
- (e.g. 00:1A:2B:3C:4D:5E).
- Used for physical connectivity between machines.
- Often uses broadcast(Wifi) or semi-broadcast medium(Ethernet).

Layered Architecture (TCP/IP): L1 (MAC / Phy-MAC layer)

- Only local; not Global.
- Broadcast MAC: FF:FF:FF:FF:FF:FF (used for ARP Requests to broadcast requests, otherwise often blocked).
- Medium contention detection and avoidance: CSMA/CD and CSMA/CA
- (Carrier sense multiple access with collision detection / carrier sense multiple access with collision avoidance)

Layered Architecture (TCP/IP): L1 (MAC / Phy–MAC layer)



IP Addressing Basics

- IPv4: 32-bit dotted decimal notation
  - Private IP ranges: 10.x.x.x, 172.16-31.x.x, 192.168.x.x
  - Public vs Private IP + role of Subnet & Gateway
  - Unique 32-bit number that is used to reach end-points on the Internet.
  - A lot like PIN/ZIP code for postal delivery.
- IPv6: 128-bit unique number now pervasive across ISP networks (End hosts though often still rely on IPv4).
  - 2001:0db8:85a3:0000:0000:8a2e:0370:7334 <=>

    2001:0db8:85a3::8a2e:0370:7334 (compressed form)

Transport Layer: TCP vs UDP

- TCP - reliable, connection-oriented (Web, Email, SSH)
  - UDP - fast, no reliability (DNS, VoIP, Streaming)
  - ICMP used for diagnostics (e.g., ping)

Ports & Services

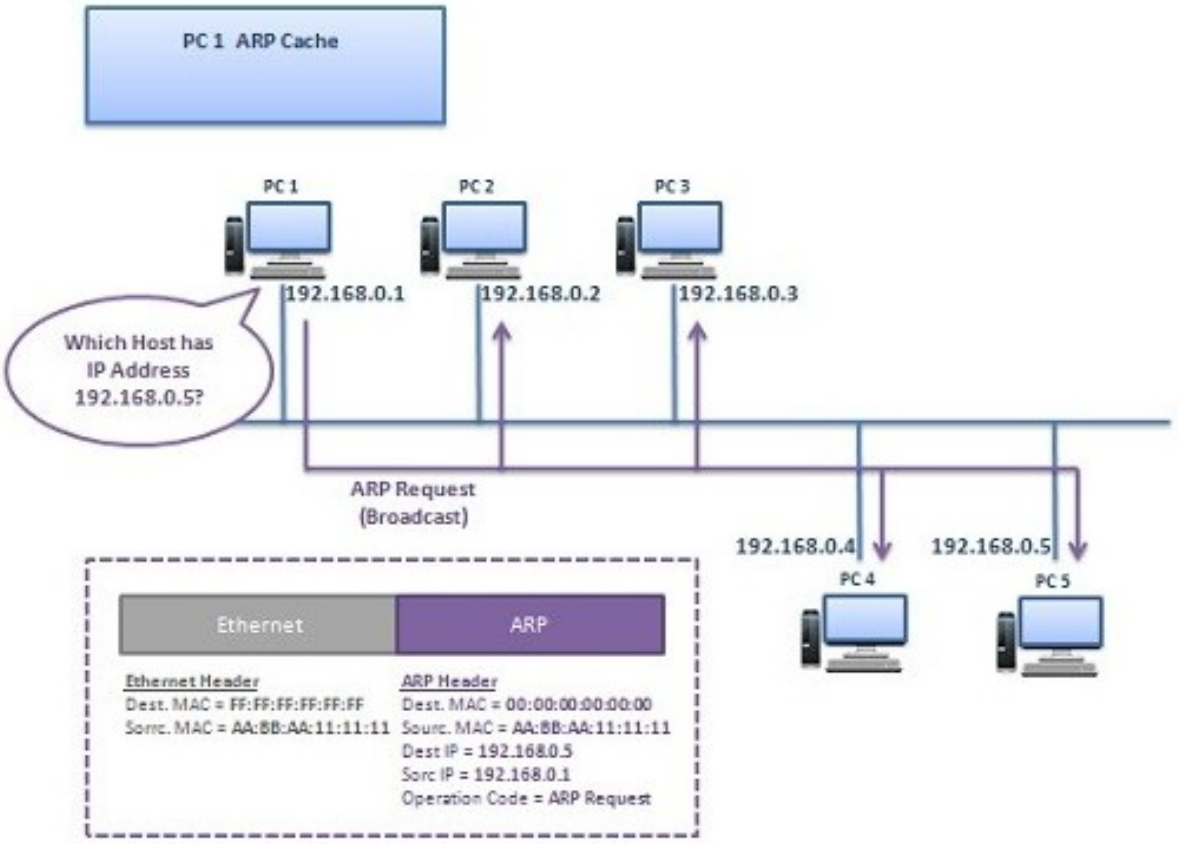
- Ports identify services on a host
  - Examples: 22 SSH, 80 HTTP, 443 HTTPS, 53 DNS
  - Security requires understanding port usage

Switching vs Routing

- Switching (Layer 2) uses MAC addresses
  - Routing (Layer 3) uses IP addresses
  - Example: PC → Switch → Router → Internet

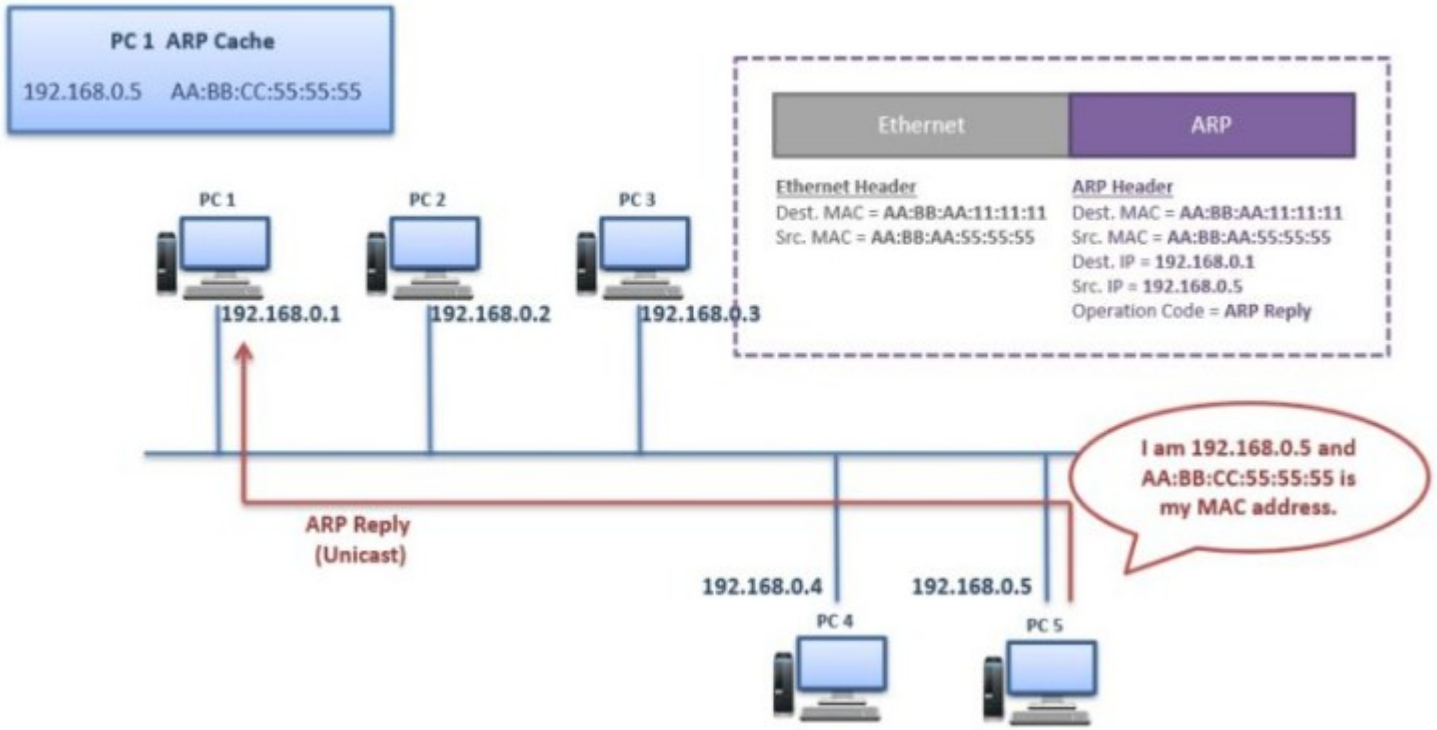
Packet Journey to a Website

- Step 0: ARP Lookup (IP to MAC Mapping; used always while sending a packet)



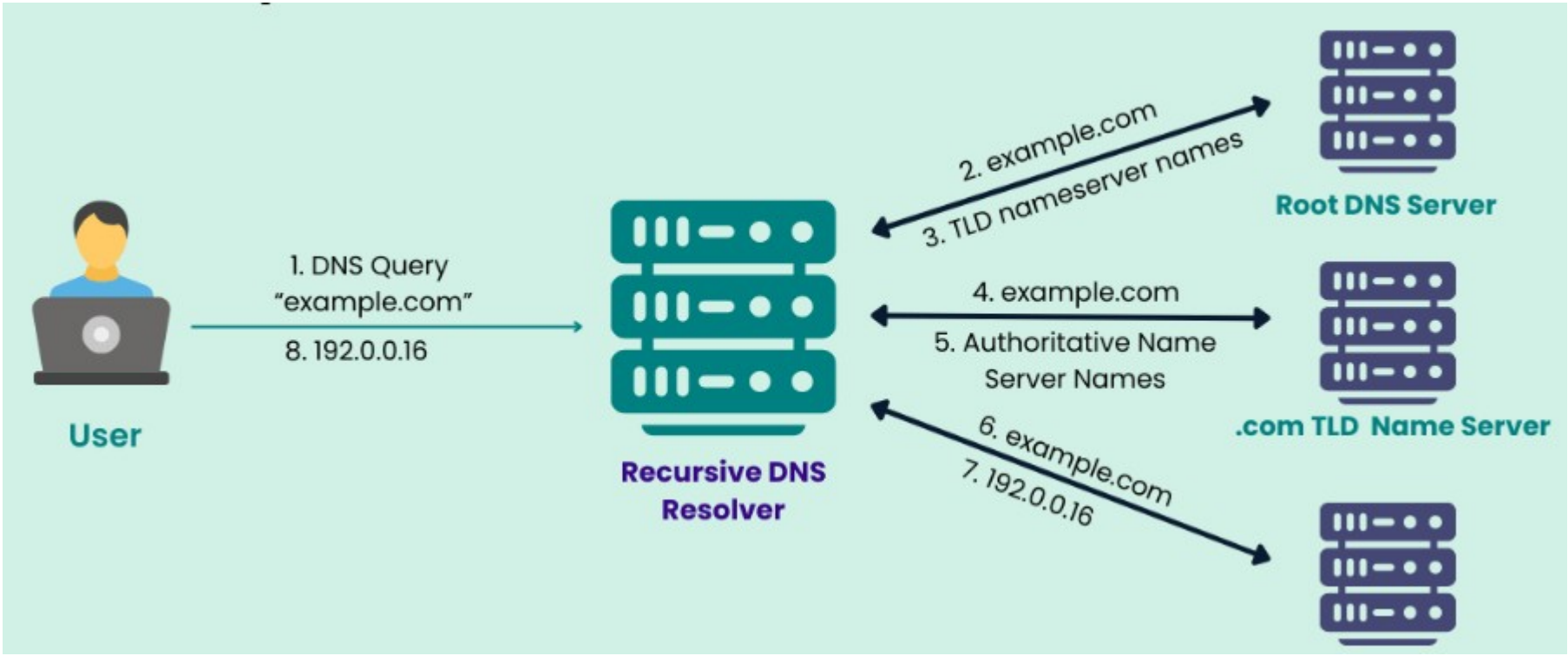
Packet Journey to a Website

- Step 0: ARP Response.



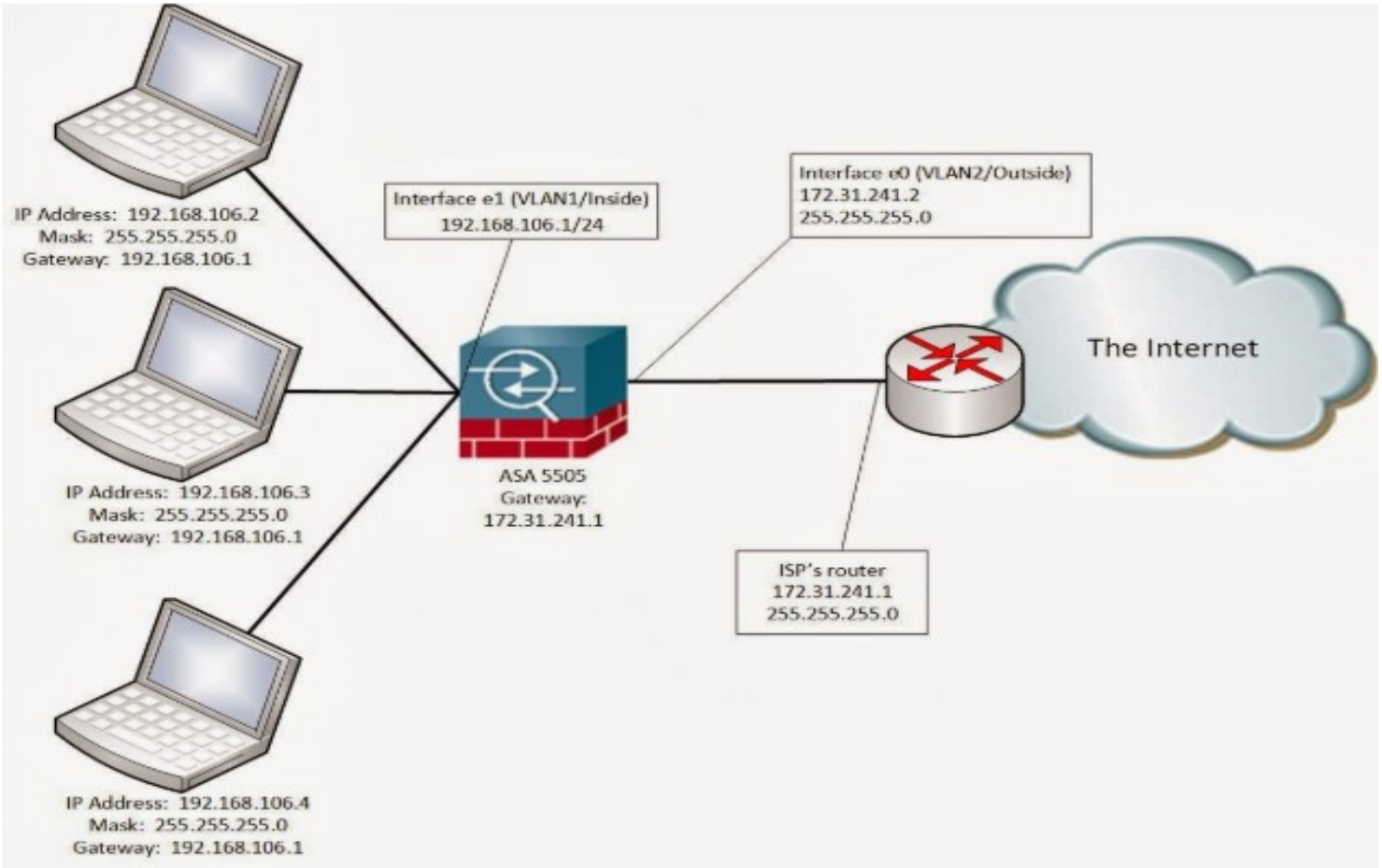
Packet Journey to a Website

Step 1: DNS Lookup



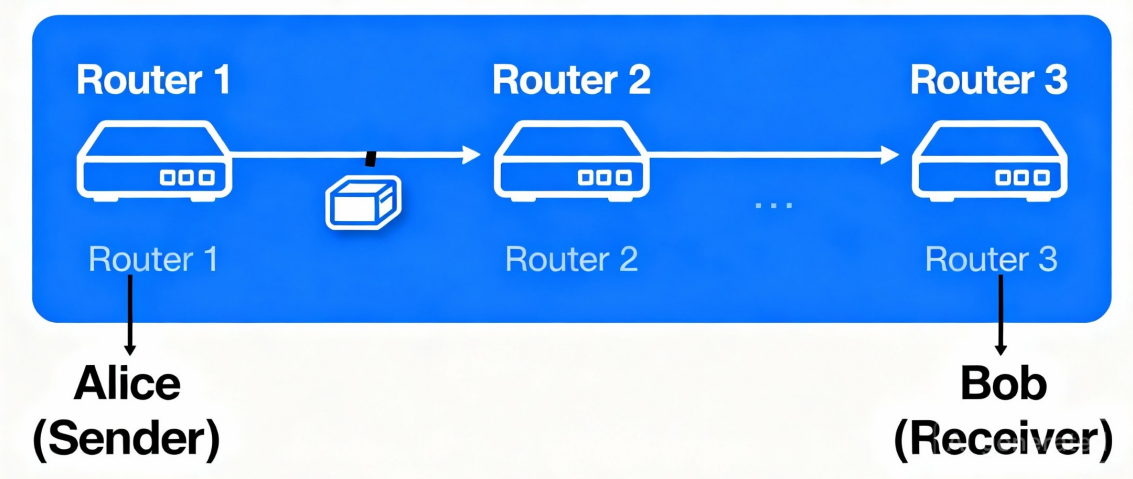
Packet Journey to a Website

Step 2: Routing Table Lookup/FWD to Default Gateway



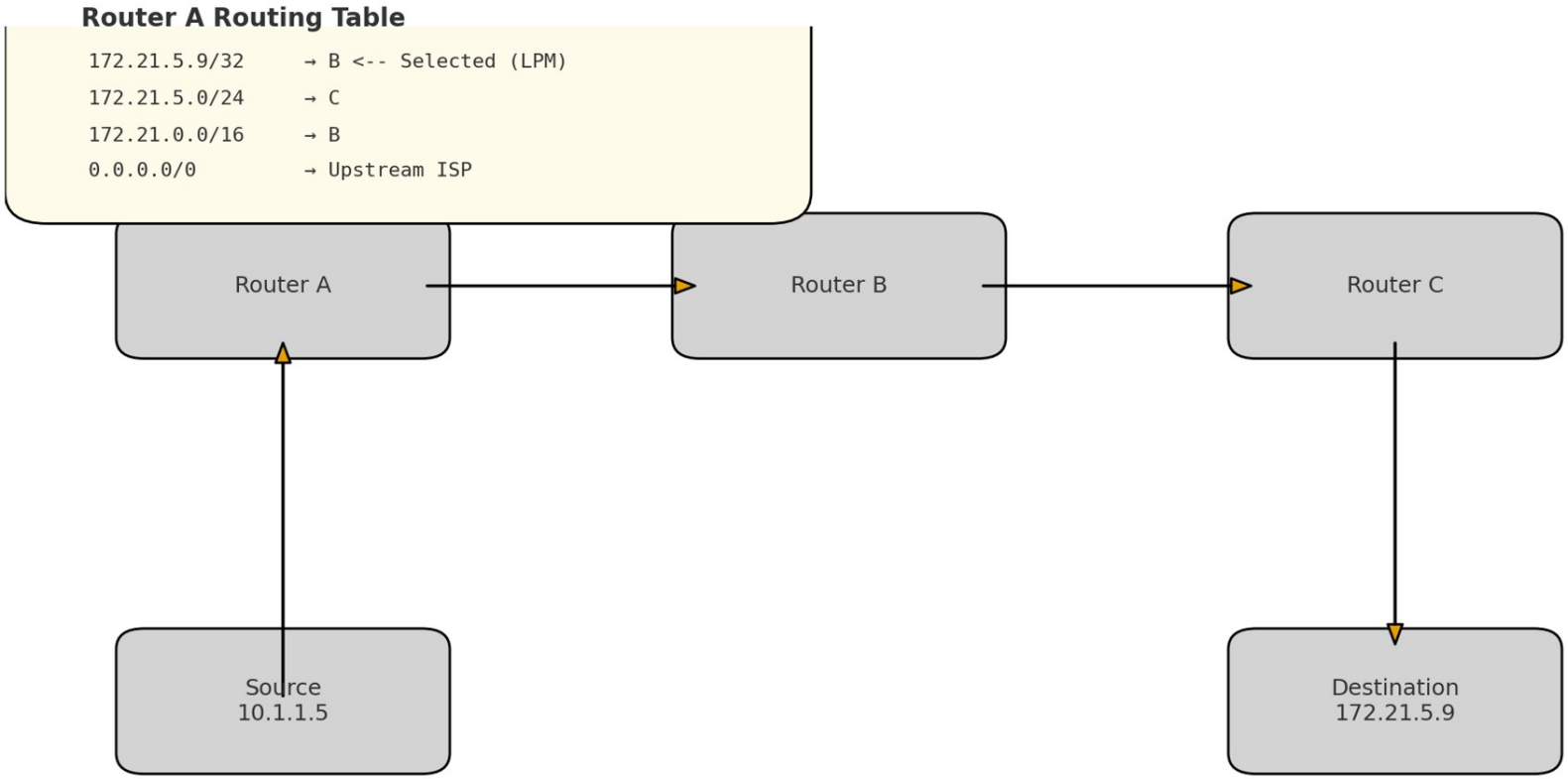
Packet Journey to a Website

- Step 3: Packet Travelling Between Routers Over the Internet. At each hop routing table is looked up and the longest prefix match is selected.



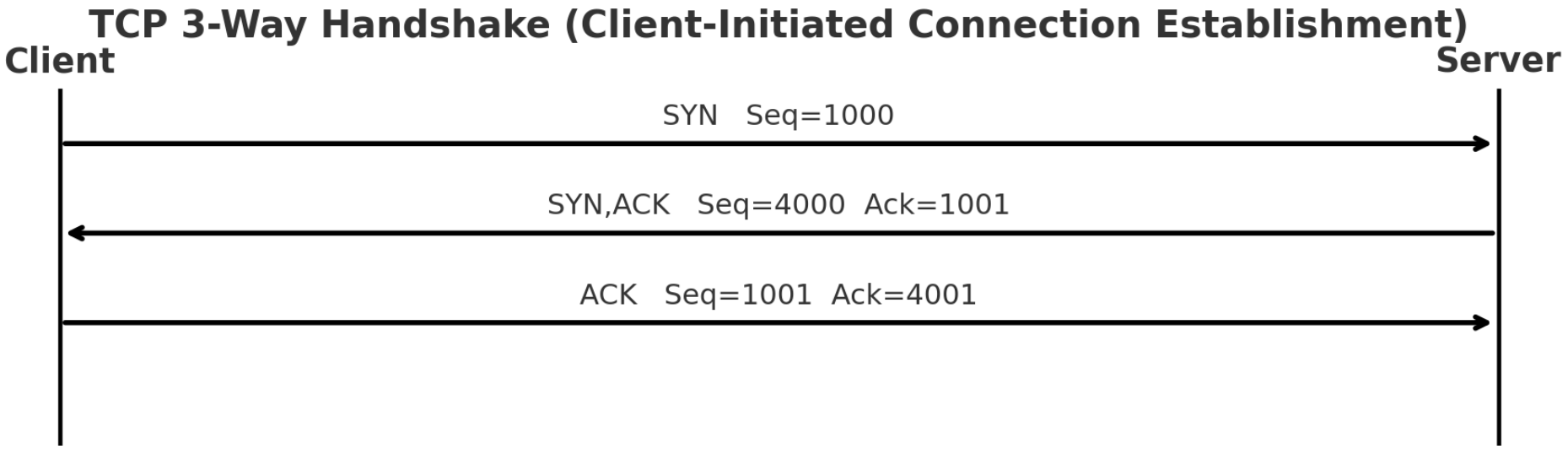
Packet Journey to a Website

- Step 3: Packet Travelling Between Routers Over the Internet. At each hop routing table is looked up and the longest prefix match is selected.



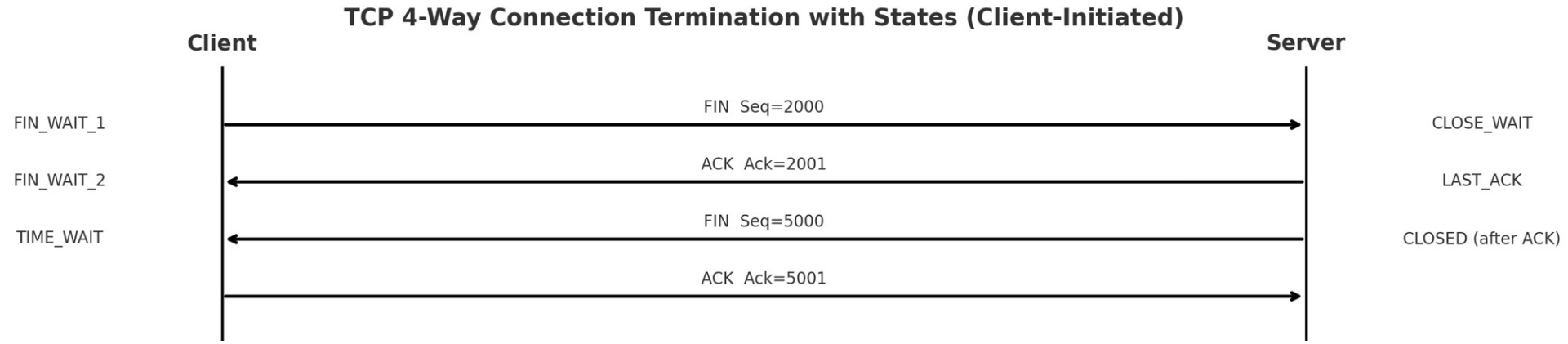
Packet Journey to a Website

Step 4: End point communication/ e.g. 3-way handshake for TCP



Packet Journey to a Website

Step 5: Connection Termination



What is a Socket

Internet hosts work on a client server model, especially TCP/IP

A socket is a bidirectional communication endpoint created when two computers communicate over a network.

Your Program ----- (Socket) ------ Internet ----- (Socket) ---- Server Program

Internally (conceptual):

- OS maintains send buffer & receive buffer
- TCP ensures every byte sent arrives (or connection fails)
- Application reads/writes using I/O streams
- The socket abstracts packet fragmentation and reassembly

What is a Socket

Java’s abstraction:

```
Socket s = new Socket("example.com", 80);
```

This line triggers:

1)DNS resolution

2)TCP three-way handshake

3)OS creates local ephemeral port

4)Stream objects are created (InputStream & OutputStream)

Why sockets matter:

- Everything — HTTP, HTTPS, SSH, FTP, SMTP — sits on top of sockets
- Understanding the socket layer is essential for debugging real systems

Using Telnet to Probe Servers

Before writing Java code, always understand what the protocol looks like raw.

```
telnet www.horstmann.com 80
GET / HTTP/1.0
Host: www.horstmann.com
```

What happens behind the scenes:

1)You open a TCP connection

(port 80 is HTTP)

2)You manually send an HTTP request

3)Server responds with:

- Status line (HTTP/1.1 200 OK)
- Headers (Content-Type, Content-Length,

  etc.)
- Blank line
- Body (HTML)

First Java Client: SocketTest (Deep Dive)

```
try (var s = new Socket("time-a.nist.gov", 13);
     var in = new Scanner(s.getInputStream()))
{
    while (in.hasNextLine()) {
        String line = in.nextLine();
        System.out.println(line);
    }
}
```

Detailed Analysis:

- The constructor triggers:
  - DNS lookup
  - TCP handshake (SYN → SYN/ACK → ACK)
- Port 13 = “Daytime Protocol” (very old, but excellent teaching example)
- `getInputStream()` returns a raw stream of bytes
- `Scanner` turns the raw byte stream into human-readable text lines
- The connection closes when:
- Server closes its end

Client implicitly closes via try-with-resources
Real-world note: Port 13 is usually blocked by firewalls; this is why the example uses NIST.

Common Issues with Basic Socket Clients

- UnknownHostException

→ DNS name cannot be resolved (typo, offline, etc.).

- ConnectException: Connection refused

→ No process is listening on that host+port.

- Hanging forever while reading

→ Server never sends data; read() blocks indefinitely.

- Firewall issues

→ Corporate/college network may block certain ports.

Why Timeouts Matter (Read Timeout)

- Default behavior
- – read() on a socket blocks until data arrives or the socket is closed.
- – This can mean blocking forever if the server is slow or unresponsive.

Setting a read timeout

```
Socket s = new Socket(host, port);
s.setSoTimeout(10_000); // 10 seconds
   try (Scanner in = new Scanner(s.getInputStream())) {
    while (in.hasNextLine()) {
        String line = in.nextLine();
        // process line
    }
} catch (SocketTimeoutException e) {
    System.err.println("Read timed out!");
}
```

Takeaway

– Use read timeouts in user-facing or interactive apps to avoid freezing the UI or threads.

Connect Timeout (vs. Read Timeout)

Two kinds of blocking

1. Connecting to the server.
2. Reading from an already connected socket.

No connect timeout)

```
Socket s = new Socket(host, port); // may block for a long time
```

With connect timeout)

```
Socket s = new Socket();
s.connect(new InetSocketAddress(host, port), 5000); // 5 seconds
```

Differences

- setSoTimeout(...) affects only read operations.
- connect(..., timeout) affects connection establishment

Practical point

- GUIs and servers should never let connect attempts block indefinitely.

InetAddress: Java's DNS Wrapper

Purpose

- Represents an IP address and optionally its hostname.
- Wraps DNS resolution for Java programs.

Key methods

```
    InetAddress addr =
InetAddress.getByName("www.google.com");
System.out.println(addr.getHostAddress());
    InetAddress[] all =
InetAddress.getAllByName("www.google.com");
InetAddress local = InetAddress.getLocalHost();
```

getAllByName may return multiple IPs (load-balanced services).

getLocalHost shows how the local machine identifies itself.

Example: InetAddressTest Program

Utility example

```
public class InetAddressTest {
    public static void main(String[] args) throws IOException {
        if (args.length == 0) {
            System.out.println("Usage: java InetAddressTest
<hostname>");
            return;
        }
        String host = args[0];
        InetAddress[] addresses =
InetAddress.getAllByName(host);
        System.out.println("All addresses for " + host + ":");
        for (InetAddress a : addresses)
           System.out.println("  " + a);
        System.out.println("Local host:");
        System.out.println("  " + InetAddress.getLocalHost());
    }
}
```

Transition: From Clients to Servers

So far

- We wrote clients that connect to existing servers.
- The server was always “someone else’s program”.

Now we flip roles

- Our Java program becomes the server.
- It listens on a port and waits for clients to connect.

Client

- Initiates the connection to a known server and port.

Server

- Binds a `ServerSocket` to a port.
- Waits (blocks) in `accept()`.
- For each accepted connection, uses a Socket to talk to that client.

Key class: `java.net.ServerSocket`

ServerSocket Basics

Core API

```
ServerSocket server = new ServerSocket(8189);
Socket incoming = server.accept(); // blocks until a client connects
```

What's happening

- new ServerSocket(8189):
  - Reserves TCP port 8189 on the local host.
- accept():
  - Blocks until a client connects.
  - Returns a new Socket for that particular client.

Important distinction

ServerSocket is just for listening and accepting.

Socket is used to communicate with the connected client (streams, etc.).

Simple Echo Server: High-Level Design

Goal

Build a server that:

1. Accepts a client connection.
2. Sends a greeting.
3. Reads lines from the client.
4. Echoes each line back with a prefix.
5. Terminates when the client sends BYE.

Echo Server Code (Single Client)

Example implementation

```
public class EchoServer {
    public static void main(String[] args) throws IOException {
        int port = 8189;
        try (ServerSocket server = new ServerSocket(port)) {
            System.out.println("Echo server running on port " + port);
            Socket incoming = server.accept(); // wait for a client
            handleClient(incoming);
        }
    }
  private static void handleClient(Socket incoming) throws IOException {
        try (Scanner in = new Scanner(incoming.getInputStream());
             PrintWriter out = new PrintWriter(incoming.getOutputStream(), true)) {
            out.println("Hello! Enter BYE to exit.");
            while (in.hasNextLine()) {
                String line = in.nextLine();
                out.println("Echo: " + line);
                if (line.trim().equalsIgnoreCase("BYE"))
                    break;
            }
        }


    }
```

Trying the Echo Server with Telnet

Steps

1. Start the Java echo server:
   $java EchoServer

2. In another terminal:
   $telnet localhost 8189

3. Example interaction
   Server: Hello! Enter BYE to exit.
   Client: Hello
   Server: Echo: Hello
   Client: Something
   Server: Echo: Something
   Client: BYE
   Server: Echo: BYE
   (connection closes)

Limitations of a Single-Client Server

Current design

- The server accepts exactly one client and then exits.

Problems

- Only one client can use the server.
- While this client is connected, no other client can connect.
- Real servers must handle many clients concurrently and stay running indefinitely.

Next step

- Turn the server into a multi-client server by using one thread per client connection.