Computer Networks & TCP/IP

Section 1: Protocols & Communication

What we will learn

- What is a protocol
- Why strict rules are needed
- Protocol suites and architecture

What is a Protocol?

A **protocol = rules + format + behavior**

Defines:

- Message format
- Order of messages
- How errors are reported (not necessarily fixed!)

Real-life analogy

Conversation rules:

- Say hello
- Take turns
- Confirm understanding

Protocol Suite & Architecture

A **protocol suite = group of related protocols**

The **architecture** = design that specifies how protocols relate and divide tasks

Example:

- TCP/IP suite implements the **Internet architecture**

Think Deeper 🧠

Why not use just ONE protocol for everything?

Summary

- Protocol = rules + format + behavior
- Protocol suite = multiple cooperating protocols
- Architecture = blueprint for how they fit together

Section 2: Internet vs Web

What we will learn

- Difference between Internet and Web
- `internet` vs `Internet`
- Role of applications

Internet vs Web

- **Internet** = global network infrastructure
- **Web (WWW)** = application using HTTP

Other internet applications:

- Email
- SSH
- FTP

`internet` vs `Internet`

- `internet` (lowercase) = any set of networks connected using a common protocol suite
- `Internet` (uppercase) = the specific global network of hosts using TCP/IP

The Internet is an internet — but not every internet is the Internet

Real-life analogy

Internet = roads

Web = one type of vehicle

`internet` = any road network

`Internet` = the national highway system

Reflect

Can Internet exist without Web?

Can Web exist without Internet?

Summary

- Internet = transport system
- Web = one service on top
- `internet` = generic; `Internet` = the global one

Section 3: Packet Switching

What we will learn

- Circuit vs Packet
- Why packet switching was invented
- Statistical multiplexing
- Datagram model

Circuit Switching

- Dedicated path reserved for entire call
- Predictable performance
- **Inefficient** — reserved bandwidth goes unused even during silence

Example: traditional telephone call

Packet Switching

Data split into **packets**, each routed independently

Why it was invented:

- **Resilience** — network can survive partial failures or physical attack
- **Efficiency** — links are shared and stay busy

Statistical Multiplexing

Packets from many sources are **mixed together** on the same link based on arrival timing

- If there is traffic and capacity, the network stays busy
- No reserved slots — bandwidth is used on demand
- Downside: performance depends on how many others are sharing

Real-life analogy

Circuit switching = renting a dedicated lane on the highway for your whole trip

Statistical multiplexing = highway where all cars share lanes — congestion depends on traffic

Datagram Model

Internet uses:

- **Connectionless delivery**
- Each packet carries its own source and destination address
- Switches don't need to store per-connection state

Real-life analogy

Sending multiple couriers instead of one large truck

Each courier finds their own route

Think About This 🤔

If packets take different routes:

- How do they reassemble correctly?

If many packets share a link:

- What happens when the link gets congested?

Summary

- Packet switching = flexible, resilient, efficient
- Statistical multiplexing = shared links used on demand
- Datagram = each packet carries its own address, no fixed path

Section 4: Internet Design Principles

What we will learn

- End-to-end argument
- Fate sharing
- Error control and flow control
- Design goals

End-to-End Argument

Keep logic at **end systems**, not inside network

Why: correctness can **only** be achieved with the knowledge of the application at the endpoints — the network cannot know what each application truly needs

End-to-End Argument: Consequence

Important functions like:

- Error checking
- Delivery acknowledgment
- Encryption

...should live at the **endpoints**, not be built into the network core

The network may assist, but **cannot replace** endpoint logic

Error Control

Data can be damaged or lost in transit (hardware faults, interference, congestion)

Error control = detecting and recovering from such damage

Per the end-to-end argument:

- Network may detect errors with **checksums**
- But damaged packets are simply **discarded** (best-effort)
- Recovery (retransmission) is handled by **TCP at the endpoints**

Flow Control

A fast sender can overwhelm a slow receiver

Flow control = mechanism to slow down the sender

- Implemented by **TCP** at end hosts — consistent with end-to-end argument
- The network core does not throttle senders
- This also allows the network to survive partial failures (fate sharing)

Fate Sharing

All state needed to maintain a connection is stored at the **endpoints**

- If the network has a temporary failure → TCP can wait and recover
- If an endpoint fails → communication is lost, but that would happen anyway
- Result: **network core stays simple and stateless**

Real-life analogy

Post office does not track your letter's meaning — sender and receiver handle it

If the post office has a delay, you (the endpoint) can resend
If you move away (endpoint gone), the communication is simply over

Design Goals

Original goals for the Internet architecture:

- Survive network/gateway failures
- Support multiple types of communication services
- Accommodate a **variety of networks** (heterogeneity)
- Permit **distributed management** of resources
- Cost-effective
- Easy host attachment
- Scalable

Reflect

What breaks if the network becomes too "smart"?

Where does TCP's reliability fit in the end-to-end argument?

Summary

- End-to-end argument: correctness belongs at endpoints
- Error control + flow control: TCP's job, not the network's
- Fate sharing: state at endpoints → simple network core
- Design goals: survivability, heterogeneity, cost, scale

Section 5: Layered Architecture

What we will learn

- Why layering exists
- Protocol Data Units (PDUs)
- TCP/IP layers and their protocols
- TCP/IP vs OSI

Why Layering?

- Reduces complexity — each layer has one job
- Enables modular design — swap one layer without changing others
- Easier to debug — isolate problems by layer

Protocol Data Units (PDUs)

Each layer has a name for its unit of data:

Layer	PDU name
Application	Message
Transport	Segment (TCP) / Datagram (UDP)
Network	Datagram / Packet
Link	Frame

These names appear in tools like `tcpdump` and Wireshark

TCP/IP Model (with key protocols)

Layer	Protocols
Application	HTTP, SSH, DNS, FTP, SMTP
Transport	TCP, UDP
Network	IP, ICMP
Link	Ethernet, Wi-Fi

OSI vs TCP/IP

- OSI = conceptual reference model (7 layers)
- TCP/IP = practical implementation (4 layers)

OSI is useful for understanding — TCP/IP is what runs the Internet

Real-life analogy

Factory assembly line:

- Each worker has a role
- One worker being replaced doesn't stop the whole line

Think Deeper 🧠

If one layer fails, how does the system recover?

Which layer would you look at if a webpage loads but SSH doesn't work?

Summary

- Layers separate concerns and enable interoperability
- Each layer has a PDU name (frame, datagram, segment, message)
- TCP/IP has 4 practical layers; OSI has 7 conceptual layers

Section 6: Encapsulation & Multiplexing

What we will learn

- Encapsulation
- Multiplexing and demultiplexing
- How identifiers chain across layers

Encapsulation

Each layer **wraps** the data from the layer above by adding a header:

Application message
→ TCP adds header → **Segment**
→ IP adds header → **Datagram**
→ Ethernet adds header → **Frame**

Multiplexing

Multiple applications share:

- Same network
- Same physical connection

Achieved by **statistical multiplexing** — packets from different sources interleaved on demand

Demultiplexing

Each layer uses an **identifier** to decide what to do with incoming data:

Layer	Identifier used
Link (Ethernet)	Ethernet Type field
Network (IP)	Protocol field (e.g. 6=TCP, 17=UDP)
Transport (TCP/UDP)	Port number

Each layer strips its header, checks the identifier, and hands data upward

Real-life analogy

Postal system:

- Country → City → Street → House number

Each level uses its own identifier to route further

Think About This 🤔

How does the system know which app should receive data?

Trace the path: an Ethernet frame arrives → what happens at each layer?

Summary

- Encapsulation = each layer wraps data with its own header
- Multiplexing = sharing links across many sources
- Demultiplexing = chained identifiers (Type → Protocol → Port) route data to the right place

Section 7: Addressing & Ports

What we will learn

- Address types (MAC and IP)
- Port number ranges (corrected)
- DNS — the naming system

Address Types

- **MAC address** — hardware level, 48-bit, identifies a network interface on a local network
- **IP address** — network level, 32-bit (IPv4) or 128-bit (IPv6), identifies a host globally

MAC addresses are used within a local network; IP addresses route across networks

Port Numbers

16-bit numbers (0–65535) that identify which **application** on a host should receive data

Three ranges defined by IANA:

Range	Type	Example
0–1023	Well-known	SSH=22, HTTP=80
1024–49151	Registered	Used by specific apps
49152–65535	Dynamic / Ephemeral	Assigned temporarily to clients

Example

- SSH → 22
- HTTP → 80
- HTTPS → 443
- DNS → 53
- SMTP (email) → 25

Servers bind to well-known ports; clients get a temporary ephemeral port

DNS — Domain Name System

IP addresses are hard for humans to remember

DNS = distributed database that maps **hostnames** → **IP addresses**

- `thejeshgn.com` → `104.21.x.x`
- DNS is itself an application-layer protocol (uses UDP port 53)
- If DNS fails → most Internet access effectively stops

Real-life analogy

IP = building address

Port = room number

DNS = the directory that tells you which building to go to

Think Deeper 🧠

Why separate IP and port instead of one identifier?

What happens when you type a URL in the browser before any TCP connection is made?

Summary

- MAC = hardware; IP = network; both needed for delivery
- Port ranges: well-known (0–1023), registered (1024–49151), ephemeral (49152–65535)
- DNS maps names to IP addresses — failure breaks most Internet use

Section 8: Core Protocols

What we will learn

- IP, TCP, UDP roles
- ICMP — the helper protocol
- Key behavioral differences

IP (Internet Protocol)

- **Best-effort delivery** — no guarantees
- Handles addressing and forwarding
- Damaged packets are discarded (not repaired)
- Does not care about order or reliability — that is TCP's job

ICMP — Internet Control Message Protocol

An adjunct to IP (sometimes called "layer 3.5")

Used for:

- **Error messages** — e.g. "destination unreachable"
- **Diagnostics** — `ping` and `traceroute` both use ICMP

ICMP messages are carried inside IP datagrams

TCP

- Reliable, ordered delivery
- **Does not preserve message boundaries** — it is a byte stream
- Flow control — prevents overwhelming the receiver
- Congestion control — prevents overwhelming the network
- Connection-oriented (establishes session before data)

UDP

- No connection, no reliability
- **Preserves message boundaries** — each send/receive is one unit
- Low overhead, low latency
- Application is responsible for any needed reliability

TCP vs UDP: Key difference

TCP = byte stream (boundaries lost)
UDP = message-based (each datagram is distinct)

This matters in practice — applications using UDP know exactly where each message starts and ends

Real-life analogy

TCP = courier with tracking and delivery confirmation

UDP = dropping a postcard in a mailbox — fast, no receipt

Reflect

Why would video streaming prefer UDP?

Why does `ping` work even when a web server is down?

Summary

- IP = best-effort delivery, addressing
- ICMP = error reporting + diagnostics (ping, traceroute)
- TCP = reliable byte stream, flow/congestion control
- UDP = fast, message-based, no guarantees

Section 9: Devices in Network

What we will learn

- Role of devices
- Forwarding vs switching
- ARP — bridging IP and MAC
- Layer interaction

Devices

- **Host** → full stack (all 4 layers)
- **Switch** → link layer only (forwards frames by MAC)
- **Router** → network layer (forwards packets by IP)

Routers and Forwarding

Forwarding = looking up a packet's destination IP address and sending it to the correct next hop

- Routers maintain **routing tables** for this
- Both hosts and routers forward packets — routers do it far more often
- Routers connect separate networks and hide their internal complexity

ARP — Address Resolution Protocol

When a host knows the **IP address** of a destination but needs the **MAC address** to actually send a frame on the local network, it uses ARP

- Broadcasts: "Who has IP 192.168.1.1?"
- The owner replies with its MAC address
- Bridges the gap between IP addressing and link-layer delivery

Real-life analogy

Router = airport hub — routes between cities

Switch = local roads — routes within a neighbourhood

ARP = asking a neighbour "which house is number 42?"

Summary

- Devices operate at different layers
- Routers forward by IP address (next hop); switches forward by MAC
- ARP resolves IP addresses to MAC addresses on local networks

Section 10: Application Models

What we will learn

- Client-server
- Peer-to-peer
- Intranets and Extranets

Client-Server

- Central server holds resources
- Clients connect on demand
- Controlled, easier to manage
- Scalable with load balancers

Example: web browser (client) → web server

Peer-to-Peer

- Every node can be both client and server
- No central control
- More fault tolerant — no single point of failure

Example: BitTorrent, early Skype

Intranets and Extranets

- **Intranet** = private network using TCP/IP inside an organisation (not public)
- **Extranet** = intranet extended to selected outside partners or users
- **Internet** = the global public network

Same protocols, different scope and access control

APIs (Sockets)

Applications use **sockets** to communicate:

- A socket = IP address + port number + transport protocol
- Same interface whether using TCP or UDP
- Hides the complexity of the layers below

Think About This 🤔

Which model is more fault tolerant — client-server or P2P?

Is your university's internal network an intranet or the Internet?

Summary

- Client-server = centralised, controlled
- Peer-to-peer = distributed, resilient
- Intranet = private TCP/IP network; Extranet = selective external access
- Sockets = application's interface to the network

Final Summary

We covered:

- Protocols, suites, and architecture
- `internet` vs `Internet`
- Packet switching, statistical multiplexing, datagrams
- End-to-end argument, error/flow control, fate sharing
- Layers, PDUs, and key protocols at each layer
- Encapsulation and the demux chain
- Addressing, port ranges (corrected), DNS
- IP, ICMP, TCP, UDP roles and differences
- Review of remaining ARP

Thank you!

Any questions?

Thejesh GN