Chapter 6: Link Layer and LANs

[6.1] introduction

- LINK LAYER is **layer 2** of OSI MODEL
- responsible for moving **datagram(network layer packet)** from one physically adjacent node to another over a **single communication link!**

⭐key terminology:

1. **Nodes:** devices that use a link-layer protocol
    - Hosts(end devices) + routers (forwarding devices)
2. **Links:** physical communication channel between 2 adjacent nodes
    - WiFi link, Ethernet cable, fiber link
3. **Frame:** link layer encapsulates IP datagram into a frame
    - Adds **header** (MAC addr,error checking bits etc)
    - Adds trailer (eg CRC)

🔴 **Important Concept**:
⇒ A **datagram travels hop-by-hop.** At each hop, the link layer moves it from one device to the next.
⇒ link layer's only job is to handle node-to-node delivery
⇒ The **network layer (IP)** chooses the *path*, but the **link layer** handles *hop-to-hop transport*.

Example: Sending Data in a Company Network

- Imagine a wireless laptop sending data to a server.
- The datagram might travel through 6 different links, such as:
    1. WiFi link (host → WiFi AP)
    2. Ethernet (AP → switch)

3. Switch → router
4. Router → router
5. Router → switch
6. Switch → server

- On each link, the data is encapsulated inside a link-layer frame and sent
- Each link may use different link-layer protocol

👋Each link-layer protocol provides different services

Examples:

- **Wi-Fi:** unreliable, wireless, must handle interference, collisions
- **Ethernet:** reliable enough, wired, low error rate
- **Cellular:** high error rate → strong error detection/correction
- **Fiber:** extremely low error rate → minimal link-layer reliability needed

⭐ Link layer: services

1. **Framing**
   - Sender encapsulates datagram inside a frame
   - FRAME contains: **MAC addr srs / dest, type field (ipv4,ipv6,ARP),error detection bits (CRC)**
2. **Link access (Medium Access Control MAC)**
   - Decided who gets to send next on a shared link
   - If it's a **point-to-point** link (just two nodes), **no conflict** → send anytime
   - If **many nodes** share the same link (WiFi, Ethernet LAN), **collisions** can happen
   - MAC protocols prevent chaos by **controlling access.**
3. **Reliable delivery b/w nodes**
   - Wireless links have high error-rate
   - Some link-layer protocols **guarantee error-free delivery** over a link, by sending ACKs and retransmission if needed (like TCP)

- Not used in **wired links** (Ethernet), cuz errors r rare so it would just be unnecessary overhead!

Q. Why do we sometimes use both link-layer and end-to-end reliability?

- Transport-layer reliability (TCP) ensures **end-to-end correctness**.
- Link-layer reliability helps when:
    i. Channel is noisy (wireless)
    ii. Retransmitting a small hop is cheaper than TCP retransmitting a large path

4. **Flow control**
    - Ensures sender does not overwhelm receiver.
5. **Error detection**
    - Detect bits corrupted due to **noise, signal issues,interference**
    - Uses: **CRC (CYCLIC REDUNDANCY CHECK), checksum**
6. **Error correction**
    - Sender includes **ERROR-CHECK BITS (CRC)**
    - Receiver checks them
    - If error:
        - It can detect
        - ORR detect + correct (some protocols)
    - Link layer error detection is **more powerful** than transport layer
    - can **fix certain errors** without retransmission.
7. **half -duplex & full-duplex**

Half Duplex	Full Duplex
Both ends can transmit but not simultaneously	Both can transmit at same time
Wi-Fi	Modern Ethernet

⭐Host Link layer IMPLEMENTATION

Q. Where is link-layer implemented?

⇒ **implemented in both hardware and software**

[hw implementation]

- In every host,router, NIC **Network interface Card**
- NIC handles→framing,link access(MAC),error detection,sending/receiving frames!
- NIC is usually built into the motherboard / separate network card

[sw implementation] sw in OS handles:

- Assigning link layer addr
- Communicates w NIC
- Handles interrupts
- Passing datagrams up to Network layer (IP)

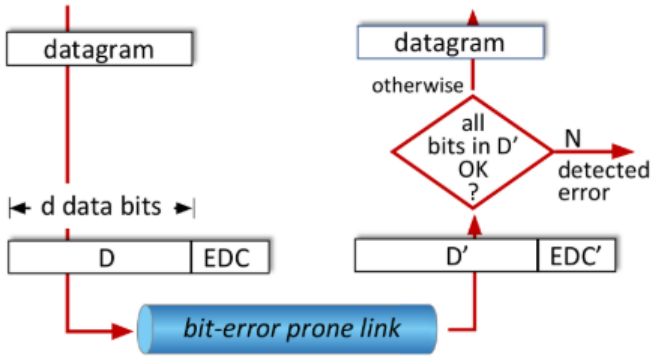
⭐Interfaces Communicating

Sending Side	Receiving Side
1. IP layer passes datagram to link layer. 2. Link layer: ○ Puts datagram into a **frame**. ○ Adds MAC addresses. ○ Adds CRC for error detection. ○ Implements flow control if needed. 3. Frame is sent to **physical layer** to transmit bits on channel.	1. Bits are decoded by a physical layer. 2. Link layer at receiver: ○ Checks for errors (CRC) ○ Performs flow control ○ Removes framing fields 3. extracts datagram and passes it to network layer (IP)

[6.2] Error detection and Correction

⇒When two nodes (like a host and a switch) communicate, the bits they send can become corrupted due to **noise or interference**. The **link layer adds special bits** to help the receiver detect or even correct these errors. These special bits are called **EDC (Error-Detection and Correction bits)**

⇒ **larger EDC fields = better detection n correction**



At the Sender:

- Data = **D** (this includes: actual data + header + addresses)
- Sender adds extra bits = **EDC**
- Sender sends: **D + EDC**

During transmission:

- Bits may flip because of noise (0 → 1 or 1 → 0)

At the Receiver:

- It receives **D + EDC** (maybe changed)
- It calculates whether an error exists
- The question is **"Is an error detected?"**, not "Did an error happen?"

👉 Even with detection bits, some rare errors may go **undetected**.

Trade-off:

- Better detection = more extra bits + more processing

⁉️ Parity checking

1️⃣SINGLE BIT PARITY:

- Detects **single bit** errors
- Can detect ODD no. of bit errors, because flipping an odd number of bits **changes the overall parity**.
- Cant detect even num of bit errors: Because flipping an even number of bits **keeps the overall parity the same**, so the receiver thinks "No error."

  EVEN PARITY:
  - Count num of 1s
  - Add 1 extra bit either 0/1 so total number of 1s becomes **even**

    Example:
    Data: **10101** → number of 1s = 3 (odd)
    To make it even → add parity bit = **1**

    Final bits = **10101 1**

  - **RECEIVER:**

    # counts num of 1s

    # if total is NOT EVEN → ERROR detected

2️⃣2D PARITY:

- Helps **detect and correct** some errors [wo retransmission]

⇒ HOW IT WORKS:

- Arrange data bits into a table with rows and columns
- Calculate parity for each:
  - **Row parity**
  - **Column parity**
- So extra bits = **i row parity + j column parity + 1 overall parity**
- If a single bit flips:
  - One row parity fails
  - One column parity fails
  - Intersection = location of the flipped bit

👉 The receiver knows the exact bit that changed → it can fix it.

Capabilities:

- **Detect + correct 1-bit errors (FEC)**
- **Detect 2-bit errors (but cannot correct)**
- Useful to understand Forward Error Correction

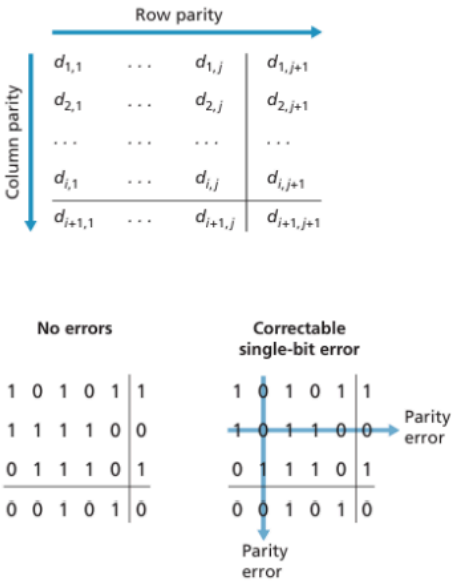


Figure 6.5 ♦ Two-dimensional even parity

🍭 Forward Error Correction (FEC)

- FEC allows the receiver to **fix errors by itself**, without asking sender to resend.
- **Benefits:**
  - No need to wait for retransmission
  - Useful for:
    - Real-time applications (voice, video)
    - Deep-space communication (large delay)

‼️ Internet checksum

- Detects errors(flipped bits) in the transmitted segment!
- Used in transport layer
- Not v strong compared to CRC

✅ Sender Side – Steps to Compute Internet Checksum

1. **Divide** the segment into 16-bit words.
2. **Add** all 16-bit words using **one's-complement addition** (wrap around any carry back into the lower 16 bits).
3. **Take the one's complement** of the final sum.
4. **Insert** this 16-bit value into the UDP checksum field.
5. **Transmit** the segment (data + checksum).

✅ Receiver Side – Steps to Verify Internet Checksum

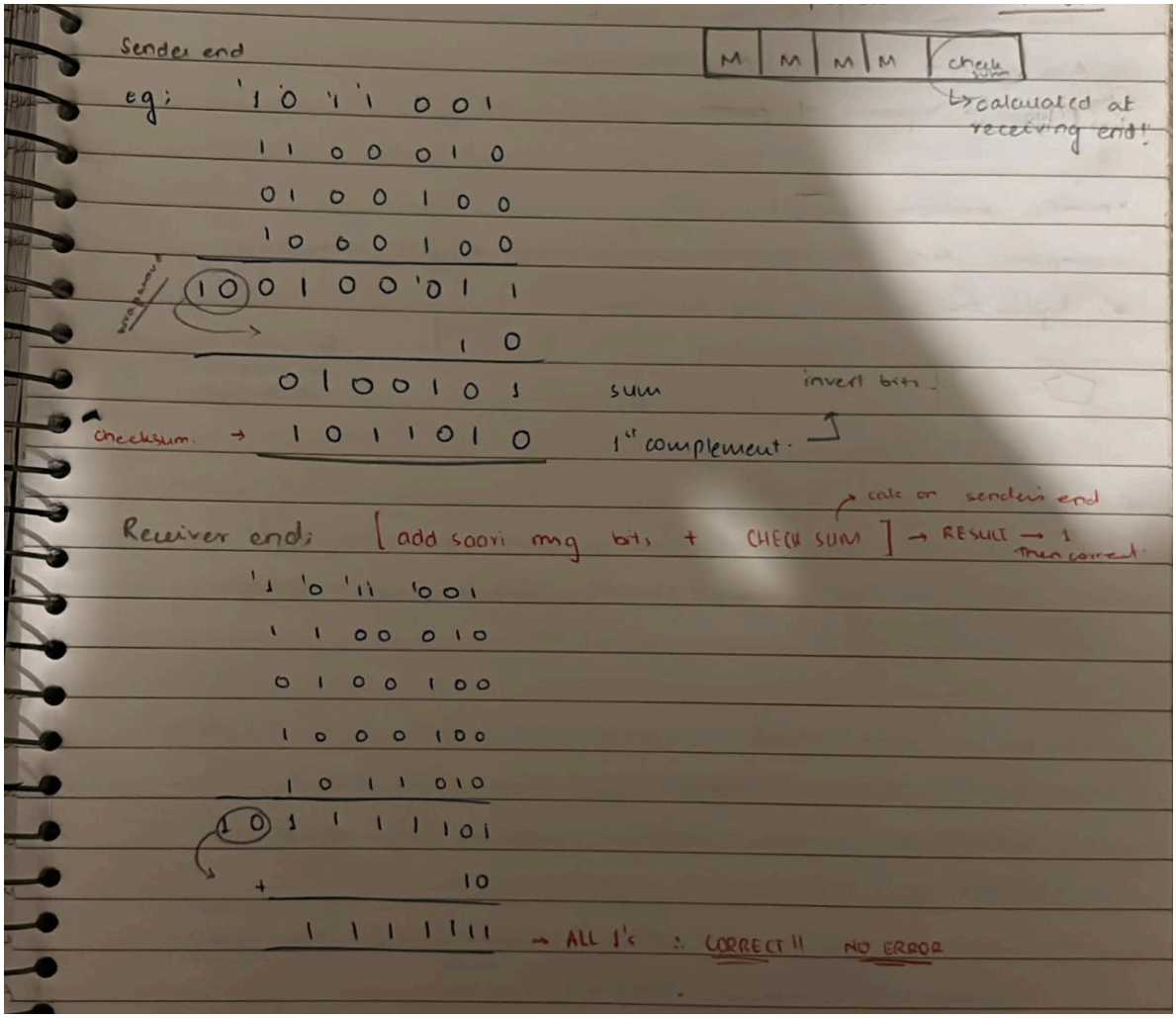
1. **Divide** received segment into 16-bit words (including checksum field).
2. **Add** all 16-bit words using **one's-complement addition**.
3. **Check** whether the final sum = **0xFFFF**.
   - If **0xFFFF → no error detected** → all 1's
   - If **not 0xFFFF → error detected**

Sender end

1. Add how ever many n-bits integers r there
2. If while adding nums, it exceeds n-bits, we **wrap around!!** [a carryout frm the most significant bit needs to be added to the result] **-> check class notes for ref**
3. After sum, we find its **1's complement** [invert bits] to convert the sum into **checksum.**
4. This checksum is placed in the UDP header and sent with the segment.

Receivers end

1. Add all the n-bits integers + the checksum calc on senders end
2. If while adding nums, it exceeds n-bits, we **wrap around!!** [a carryout frm the most significant bit needs to be added to the result] **-> check class notes for ref**
3. If **result == ALL 1's** -> no error
4. If it differs, an error is detected and that segment is discarded



⁉️ Cyclic Redundancy Check (CRC)

- Most powerful error detection method
- Used by: WIFI, Ethernet etc
- Can detect **burst errors of length <= r bits , longer errors, all single bit errors**
- Implemented in hw → therefore its fast

⭐ 1. CRC — Sender Side Steps

Given:

- **D** = data bits
- **G** = generator polynomial (of length **r+1** bits)
- **r** = number of CRC bits = length(G) − 1

$R = \text{remainder}\left[\frac{D \cdot 2^r}{G}\right]$ *algorithm for computing R*

Step 1 — Append r zeros to the data

Compute:

$$D \cdot 2^r$$

which means:

👉 take D and append r zeros at the end.

Step 2 — Divide (D with r zeros) by G using modulo-2 division

- Use XOR instead of subtraction
- No carries or borrows
- Do long division **bit-by-bit**, just like in the slide.

Step 3 — The remainder of this division = R

This remainder has exactly r bits.

Step 4 — Form the final frame

$$\langle D, R \rangle$$

which is **data followed by the CRC bits**. R

Step 5 — Send ⟨D,R⟩

This frame is constructed so that it is **exactly divisible by G**.

⭐ 2. CRC — Receiver Side Steps

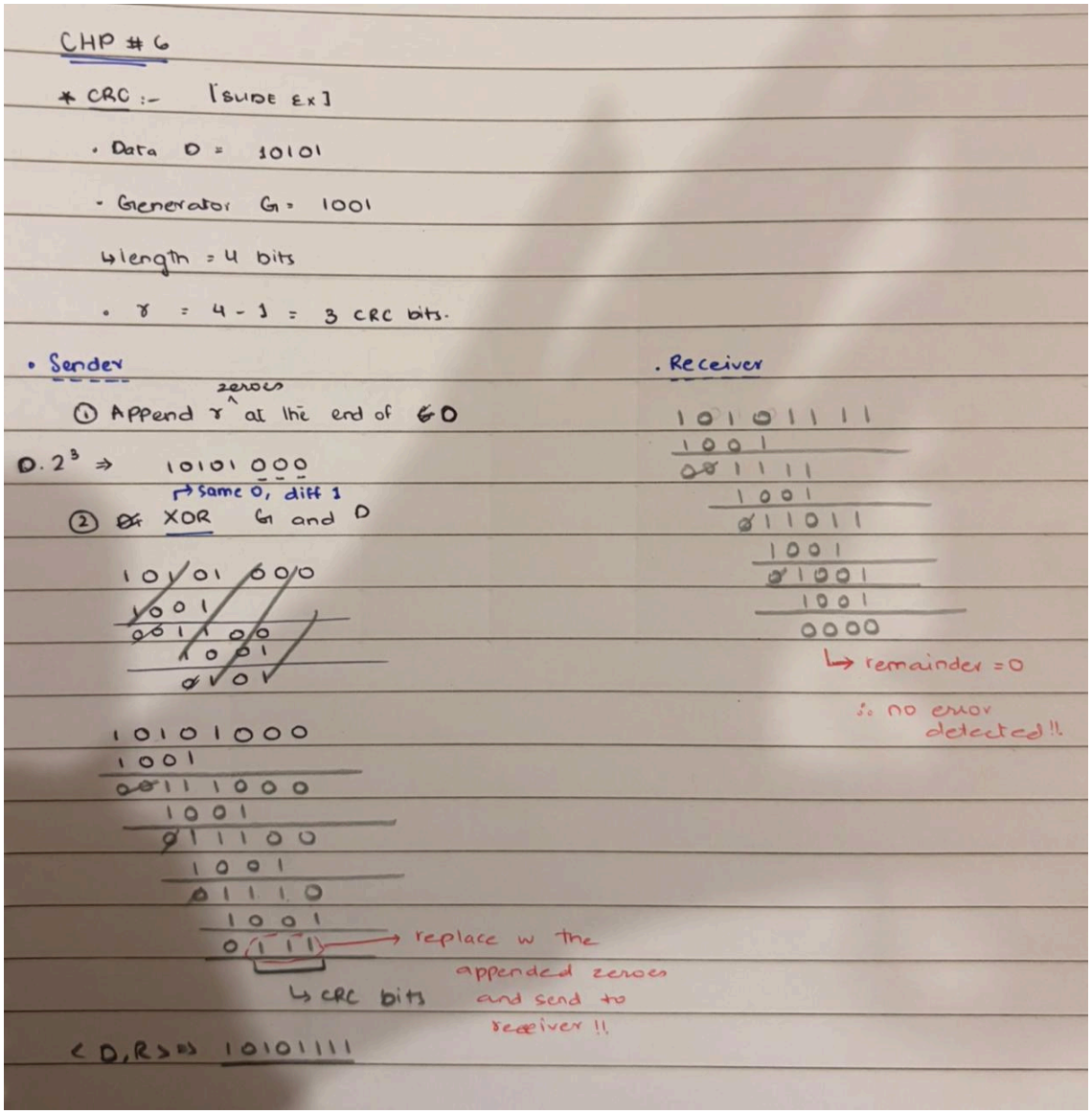
When the receiver gets **⟨D,R⟩**, it does:

Step 1 — Divide received bits by G (same modulo-2 division)

Step 2 — Check the remainder

- If remainder = 0 → **No error detected**
- If remainder ≠ 0 → **Error detected**

⇒ **solved**


CHP # 6
* CRC :- [SLIDE Ex]
• Data D = 10101
• Generator G = 1001
↳length = 4 bits
• r = 4 - 1 = 3 CRC bits.
• Sender
① Append r zeroes at the end of D
D.2^3 ⇒ 10101 000
↳same 0, diff 1
② XOR G and D
10101000
1001
0011000
1001
0111000
1001
0011100
1001
0001010
1001
0000111
replace w the appended zeroes and send to receiver !!
↳ CRC bits
<D,R> ⇒ 10101111
. Receiver
10101111
1001
0011111
1001
0011011
1001
0001001
1001
0000
↳ remainder = 0
∴ no error detected !!

[6.3] multiple access links and protocols

2 types of links:

1. Point-to-point links

- Only 2 devices share the link!
- 1 sender 1 receiver
- Eg: cable connecting host ←→ switch
- No collisions bec of dedicated path + no need for channel sharing

2. Broadcast links (shared medium)

- Many devices share 1 communication channel
- Collisions can occur bec 2 or more devices send at the same time → **MULTIPLE ACCESS PROBLEM**
- Eg: wifi, 4G/5G, old school ethernet
- Requires a **MAC PROTOCOL** to manage access

⭐Multiple Access Protocols

- A broadcast channel means:
  - All nodes transmit into the **same shared medium**
  - If two nodes transmit simultaneously:
    → **signals interfere**
    → **collision occurs**
    → data is corrupted

⇒ **Core Job of a MAC Protocol:**

- distributed algorithm that decides **when each node is allowed to transmit** so the shared channel is used efficiently and collisions are minimized or handled.
- No central controller, no separate coordination channel exists!

⚡An ideal multiple access protocol

- multiple access channel of rate **R bps**
- **Single active node → gets full rate R**
- **M active nodes → each gets R/M**
- Fully **decentralized! No synchronization**
- Easy to implement, low overhead + scalable

⭐MAC protocols: taxonomy

1. Channel partitioning
2. Random access
3. Taking turns

1. Channel partitioning

- Channel divided into **non-overlapping** pieces→ allocation for exclusive use
- Ways to partition:
    - **TDMA** → Time slots
    - **FDMA** → Frequency bands

      CDMA → Code sequences
- No collision → **PROS**
- Bandwidth might be wasted if there's nothing to send → **CONS**
- ✔ **Efficient & fair at high load**

2. Random access

- Channels **NOT DIVIDED**
- **Fully decentralized**
- Highly efficient when few nodes r active
- **CONS:** collisions reduce bandwidth efficiency!
- Allow collisions, cuz nodes can transmit whenever they want
- When a node has to send data, it **transmits immediately at full channel rate R**
- No prior coordination between nodes, so collision occurs if 2 nodes send at the same time
- Recover from collisions via delayed transmission
- Uses protocols to recover from collisions : **ALOHA, slotted ALOHA,CSMA /CD [ethernet] , CSMA /CA[WiFi]**
- ✔ Efficient at **low load** (one node can use full bandwidth)
- ✘ Inefficient at **high load** due to collisions

3. Taking turns

- Nodes take turns transmitting
- Nodes with more data get longer turns
- No collision

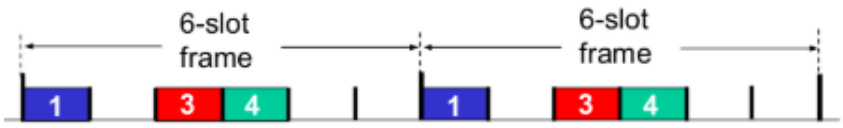
🕑Channel partitioning MAC protocols: TDMA

TDMA ⇒ TIME DIVISION MULTIPLE ACCESS

- Access to channel is divided into **time slots**
- A full set of slots = a **frame**
- Each node gets **one fixed time slot per frame**
- No collisions + fair
- Insufficient if many slots r idle/go wasted

Example:

- 6-station LAN
- Only stations 1, 3, and 4 have data to send, Slots 2, 5, 6 remain idle (wasted)

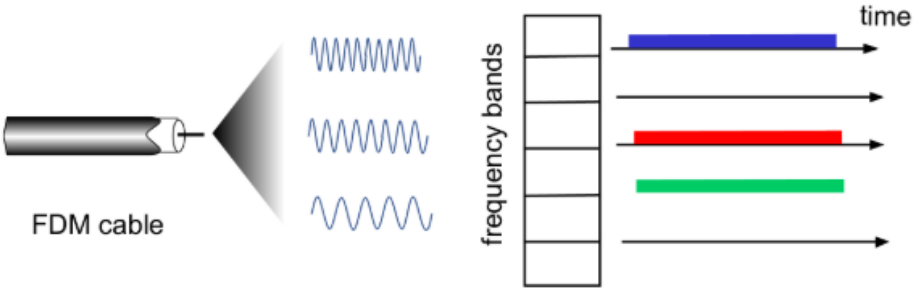


📈Channel partitioning MAC protocols: FDMA

FDMA ⇒ frequency division multiple access

- Channel bandwidth is divided into **multiple frequencies**
- Each node gets a **fixed frequency band**
- unused transmission time in frequency bands go idle → wasted bandwidth
- No collision

- example: 6-station LAN, 1,3,4 have packet to send, frequency bands 2,5,6 idle



⭐Slotted ALOHA

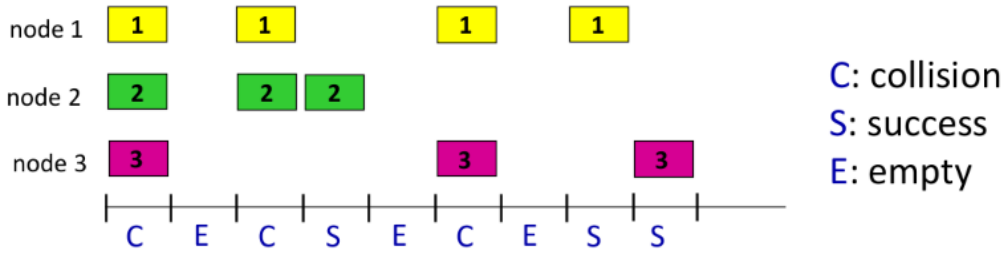
Assumptions:

- All frames have same size
- Time divided into equal sized slots
- 1 slot = time to send 1 frame
- Nodes are time synchronized
- If 2+ nodes transmit in the same time slot → **collision!**

- **node transmits only at the beginning of a slot**

Operation

- Node gets a NEW FRAME? → send it in **next slot**
- If **no collision** → success → node can send again next slot.
- If **collision** → node retransmits in each next slot with **probability p** (***randomization prevents repeated collisions***).



➡️ **A collision occurs → the frame is lost.**
➡️ **Each node must retransmit the SAME frame.**
➡️ **But it must NOT retransmit immediately**, otherwise they would collide again.

How do nodes avoid repeated collisions? They **wait a random amount of time** before trying again.

- After a collision, each node **tries again in each future slot with probability p**.
- That means:
  - With probability **p**, it retransmits in this slot.
  - With probability **1 - p**, it waits until the next slot.

This randomness ensures:

- Nodes do **not** retransmit in the same slot again.
- Eventually one node transmits **alone** → success.

PROS & CONS

PROS	CONS
- 1 node active? Can transmit at full channel rate - Decentralized - V simple protocol	- Collisions → wasted slots - Idle slots → waste capacity - Requires clock synchronization

EFFICIENCY

Efficiency = long-run fraction of slots that contain a **successful** transmission.

⇒ Slotted ALOHA uses channel **37% efficiently**.

⇒ The remaining **63%** of slots are collisions or empty.

Assume:

- N nodes, each transmits in any slot with probability p

Probability a **given node** succeeds:

$$p(1-p)^{N-1}$$

Probability **any node** succeeds:

$$Np(1-p)^{N-1}$$

To find best efficiency, we choose **p** that maximizes this expression.

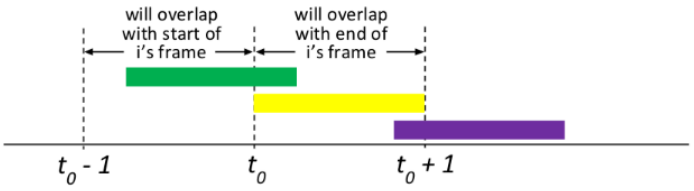
For many nodes (N → ∞):

$$\text{Max efficiency} = \frac{1}{e} = 0.37$$

⭐ Pure ALOHA

- Unslotted! + no synchronization needed
- When frame first arrives, transmit immediately
- More collisions cuz no synchronization
- This **doubles** the collision probability compared to Slotted ALOHA.
- Since collisions are DOUBLED, efficiency is now HALF of that of slotted aloha
- Maximum efficiency = **18%** **[1 / 2e]**
- **82% wasted due to collisions or idle time**

• frame sent at t_0 collides with other frames sent in $[t_0-1, t_0+1]$



⭐ 8. ALOHA Comparison Table (Perfect for Exams)

Feature	Pure ALOHA	Slotted ALOHA
Time slots	✗ No	✓ Yes
Synchronization required	✗ No	✓ Yes
Can send anytime	✓ Yes	✗ Only at slot start
Collision window	2 frame times	1 frame time
Max efficiency	**18%**	**37%**
Simplicity	Very simple	Simple
Performance	Low	Better

⭐CSMA (Carrier Sense Multiple Access)

- Listen bfr transmit
- Check if channel is idle, if yes → send frame, If channel is busy → wait until it becomes idle
- NOO INTERRUPTION!

PROBLEM?

- **COLLISIONS** can still occur
- Reason? **propagation delay**
    - Node A senses idle and starts transmitting.
    - Before A's signal reaches Node B, B senses idle too and starts transmitting.
    - Their signals collide in the middle.
- **Entire packet transmission time is wasted.**
- Longer distances → higher propagation delay → more collisions

⭐CSMA /CD (collision detection)

CSMA/CD = **CSMA + Collision Detection**

- Used in **ETHERNET**
- Reduces amount of time wasted in collisions as it aborts transmission on collision detection FORANNN

✔ What CSMA/CD Adds:

- Node *listens while sending.*
- If it detects another transmission while sending→ **collision detected**.
- It **aborts transmission immediately** (don't waste time sending whole frame).
- Send a **jam signal** to notify others.
- After abort → use **Binary Exponential Backoff** → wait for t time then start listening bfr sending again

⭐ What is Binary Exponential Backoff?

- algorithm used in **Ethernet CSMA/CD** to decide **how long a node should wait before trying again** after a collision.
- 🎯 **Goal:** Reduce repeated collisions when many nodes are trying to send.
- backoff window **doubles every time** → exponential!

⭐ Key Idea (Super Simple Version)

After the **1st collision**, wait a random time in a **small range**.

After the **2nd collision**, wait a random time in a **bigger range**.

After the **3rd collision**, wait in an **even bigger range**.

After each collision, the *waiting window doubles*.

That's why it's called **binary (base-2) exponential (doubling) backoff**.

⭐ The Rule (Formula)

After the **m-th collision**, choose a random integer:

$$K \in \{0, 1, 2, \ldots, 2^m - 1\}$$

Then wait for:

$$\text{Backoff time} = K \times 512 \text{ bit-times}$$

💡 512 bit-times = minimum Ethernet slot time

Why CSMA/CD Cannot Be Used in Wi-Fi:

- Wifi is **half-duplex!!!** Cant listen and transmit simultaneously

📐 Efficiency Formula

$$\text{efficiency} = \frac{1}{1 + 5\frac{t_{\text{prop}}}{t_{\text{trans}}}}$$

t_{prop} = maximum propagation delay between any two nodes in the LAN

t_{trans} = time to transmit a **max-size Ethernet frame**

✔ If propagation delay is small

Signal travels very fast, LAN is small.

$$t_{\text{prop}} \to 0 \Rightarrow \frac{t_{\text{prop}}}{t_{\text{trans}}} \to 0$$

Then:

$$\text{efficiency} \to \frac{1}{1+0} = 1$$

Meaning: **100% efficient**

Collisions almost never happen; medium is used almost perfectly.

✔ If frame transmission time is large

$$t_{\text{trans}} \to \infty \Rightarrow \frac{t_{\text{prop}}}{t_{\text{trans}}} \to 0$$

Again:

$$\text{efficiency} \to 1$$

Meaning:

- If frames are long, overhead of collisions becomes negligible.

This is why Ethernet has a **minimum frame size** — to ensure t_{trans} is not too small.

✔ CSMA/CD is FAR more efficient

✔ Cheaper, simpler, and fully decentralized

✔ Works much better than ALOHA under all LAN conditions

⭐CSMA /CA (collision avoidance)

- Random backoff **before** transmitting
- Optional RTS/CTS handshake
- Acknowledgements (ACK)
- Inter-frame spacing rules

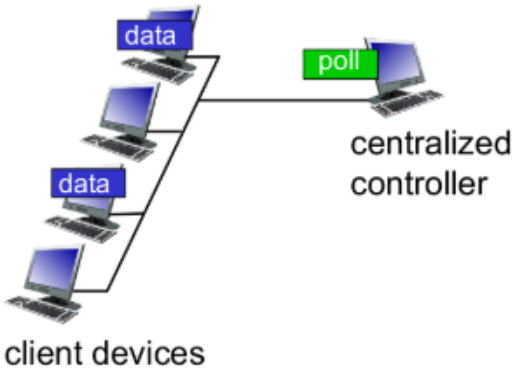
Wi-Fi uses CSMA/CA because **collision detection is impossible in wireless**.

⭐ "Taking Turns" MAC Protocols

- No collisions (like partitioning)
- High utilization even with few active nodes (like random access)
- Two main examples:
    1. **Polling**
    2. **Token Passing**

1️⃣Polling (Centralized "Taking Turns")

- A **central controller** (switch/server) decides who transmits.
- Single point of failure + polling overhead + latency [if many devices, a node might wait long to be polled]

- sequentially **polls** each device: • "Do you have data to send?" • If yes → the device transmits • If no → controller polls next device	

2️⃣Token Passing (Decentralized "Taking Turns")

- a **token** circulates among nodes in a predefined order (often a logical ring).
- A node **may transmit only when holding the token**.
- After transmitting (or if it has nothing to send), it **passes the token** to the next node.
- No collisions+ Fair: every node gets a turn + Efficient at high loads
- Token overhead + latency + single point of failure [token lost? Or node fails]

⭐Cable Access Networks (DOCSIS) — Combination of FDM, TDM, and Random Access

⇒ usaid VM [slide 37]

⭐ Downstream (ISP → You)

Uses **FDM (Frequency Division Multiplexing)**:

- CMTS (Cable Modem Termination System) transmits on **multiple downstream frequencies**
- Each channel = up to **1.6 Gbps**
- Broadcast to all homes

⭐ Upstream (You → ISP)

More complex:

- Limited bandwidth
- Many users contending for few channels

Two methods used:

1. **Random access** → users transmit requests in "minislots"
2. **TDM slots** → CMTS assigns slots to modems

⇒ **FDM ⇒ used for downstream (**broadcast**)**

⇒ **fdm makes multiple channels for upstream → where TDM and random access protocols are used!**

- upstream capacity is always smaller than downstream.
- upstream is a shared, contention-based environment.

🏄 SUMMARY

- channel partitioning, by time, frequency or code
  - Time Division, Frequency Division
- random access (dynamic),
  - ALOHA, S-ALOHA, CSMA, CSMA/CD
  - carrier sensing: easy in some technologies (wire), hard in others (wireless)
  - CSMA/CD used in Ethernet
  - CSMA/CA used in 802.11 ⟶ WiFi
- taking turns
  - polling from central site, token passing
  - Bluetooth, FDDI, token ring

[6.4]LANs

⭐ IP Address

- 32 bit
- Network layer
- used for routing across networks (end-to-end)
- Logical addr, changes when u move to a diff network
- Identifies device location in network

⭐ MAC Address

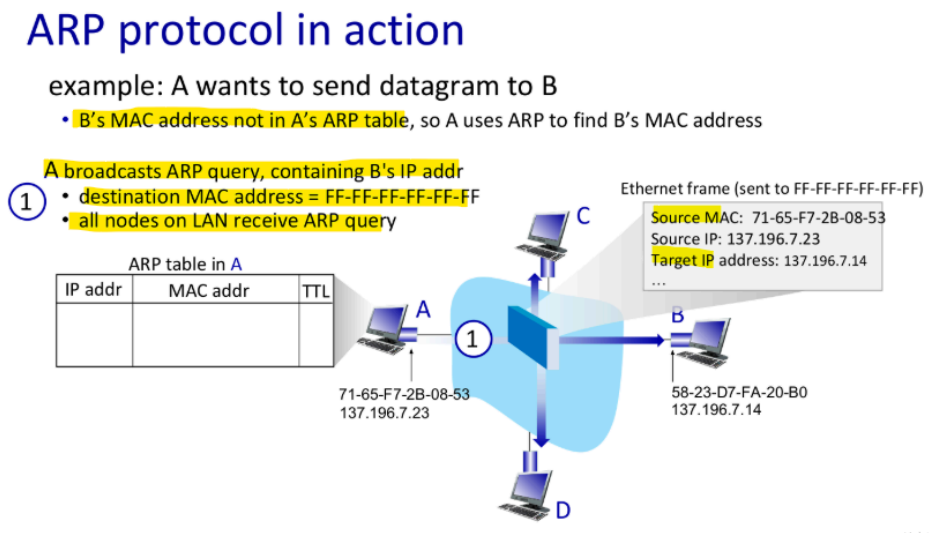
- 48 bits
- **LAN address, Ethernet address, physical address.**

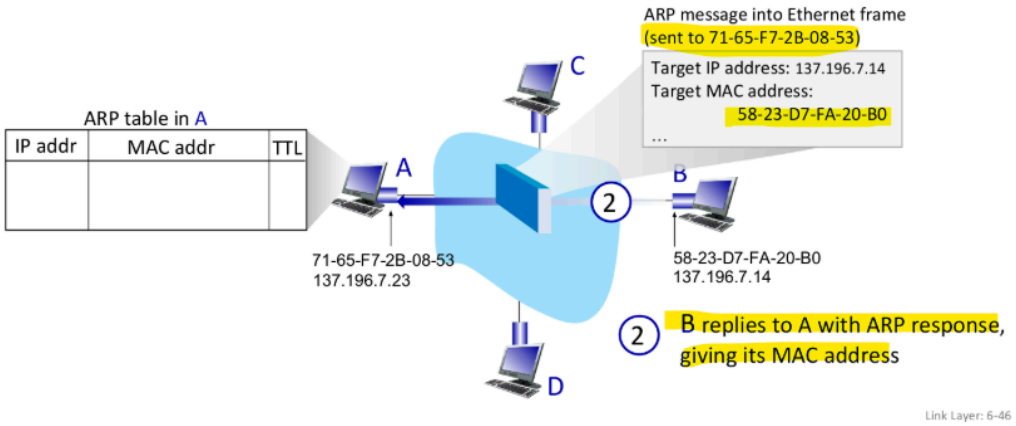
- Used **locally** within the same **LAN** to deliver frames from one interface to another physically connected interface
- 1A-2F-BB-76-09-AD ⇒ Hexadecimal notation (each digit = 4 bits)
- Hard coded into **NIC ROM** (Network Interface Card - Read Only Memory)
- Globally unique!
- **Enables frame delivery within the same subnet (same LAN).**
- MAC addr allocation administered by IEEE
- Manufacturer buys a block of mac addr space →OUI (Organizationally Unique Identifier) → to ensure uniqueness
- **flat address** = no hierarchy (no region, subnet, location info).

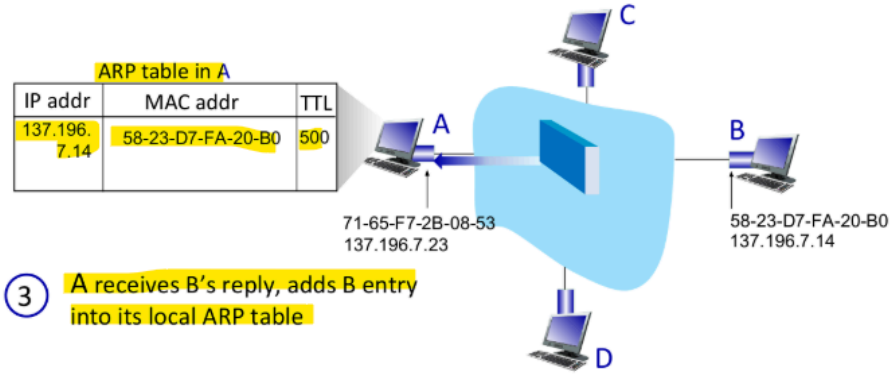
⭐ ARP → ADDRESS RESOLUTION PROTOCOL

- **Given an IP address → find the corresponding MAC address inside the same LAN.**
- Every host/router has an **ARP table:** IP address | MAC address | TTL (time to live) (~ 20mins)

ARP protocol in action







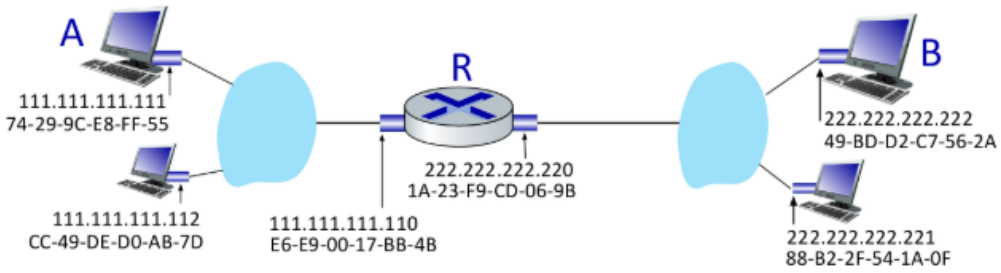
Routing to another subnet: addressing

If A and B are in different subnets, A CANNOT ARP for B's MAC.
Instead, A sends the frame to **its router (default gateway)**.

⭐ **Scenario: A wants to send to B in another network**

A's IP: **111.111.111.111**
B's IP: **222.222.222.222**
They are in **different subnets**.

Router R connects the two networks.



Step 1 — A creates the IP datagram

- Source IP = A's IP
- Destination IP = B's IP

  This IP header will **NOT change** during the entire trip.

Step 2 — A needs to send the frame to R (its default gateway)

A knows:

- B's IP
- R's IP (from configuration or DHCP)
- BUT needs **R's MAC address** → uses ARP.

A sends ARP request:

"Who has IP address of Router R?"

R replies with its MAC.

A now builds an Ethernet frame:

```rust
MAC dest = R's MAC
MAC src  = A's MAC
IP src   = A's IP
IP dest  = B's IP
```

This frame travels from A → R.

Step 3 — Router R receives frame

- R checks the destination MAC (it matches), so it accepts the frame.
- R removes the old Ethernet header. → datagram
- R looks at IP header:
  - IP destination = 222.222.222.222 (B)

R decides output interface.

Step 4 — R must send the datagram toward B

To send to B:

- R checks ARP table for B's MAC.
- If not found → R sends ARP request on B's subnet.
- B responds → R learns B's MAC.

Now R builds a new Ethernet frame:

```java
MAC src  = R's MAC on outgoing interface
MAC dest = B's MAC
IP src   = A's IP            (unchanged!)
IP dest  = B's IP            (unchanged!)
```

R sends the frame to B.

Step 5 — B receives the frame

- Ethernet layer strips MAC header.
- IP layer sees IP destination = B → deliver up the stack.

[slides #48-53]

⭐ Ethernet:

- **dominant wired LAN technology**
- Simple and cheap + high speed **upto 400 Gbps**
- A single Ethernet NIC (e.g., Broadcom BCM5761) can support multiple speeds.

2️⃣ Ethernet Topologies over Time

A. Old days: Bus topology (1980s–1990s)

- All nodes shared **one coaxial cable** (the "bus").
- **Everyone was in the same collision domain** → collisions possible.
- Used with **CSMA/CD** (Carrier Sense Multiple Access with Collision Detection).

B. Modern Ethernet: Switched topology

Today Ethernet uses **LAN switches**, not coaxial bus.

Why switching took over:

- Each device connects to its own **switch port**.
- **No collisions** because each link is separate and full-duplex.
- Faster, more reliable, scalable.

⇒ Ethernet Frame Structure

Ethernet frame wraps the IP packet before transmission.

🔹 Preamble (8 bytes)

- Synchronizes sender and receiver clocks.
- Pattern: `10101010` repeated 7 times + `10101011`.

🔹 Destination MAC address (6 bytes)

- Can be:
  - **Unicast** → specific device
  - **Broadcast** → FF-FF-FF-FF-FF-FF
  - **Multicast**

🔹 Source MAC address (6 bytes)

Always the sender's MAC.

🔹 Type field

Indicates **which network layer protocol** is inside the frame.

Example:

- 0x0800 → IP
- 0x0806 → ARP
- 0x86DD → IPv6

🔹 Data (payload)

- Contains the IP datagram (or other network-layer protocol).
- Minimum 46 bytes, maximum 1500 bytes.

🔹 CRC (Cyclic Redundancy Check)

- Used for error detection.
- If CRC fails → frame dropped (no ACK).

Ethernet is Unreliable and Connectionless

- No handshake b/w sender n receiver
- Frame corrupted? Receiver drops it → No ACK/NAK sent
- TCP handles retransmission if needed

5 Ethernet's MAC Protocol

Originally Ethernet used:

🟥 CSMA/CD (Carrier Sense Multiple Access with Collision Detection)

- Used in **bus topology**
- Devices listen before transmitting
- If collision occurs → binary exponential backoff

⚠️ **Note:**

Modern Ethernet (switched full-duplex) **does NOT use CSMA/CD** because collisions do not occur.

6 Ethernet Standards (IEEE 802.3)

Ethernet has many physical-layer variants, all sharing the **same MAC and frame format**.

✔ Copper (twisted pair)

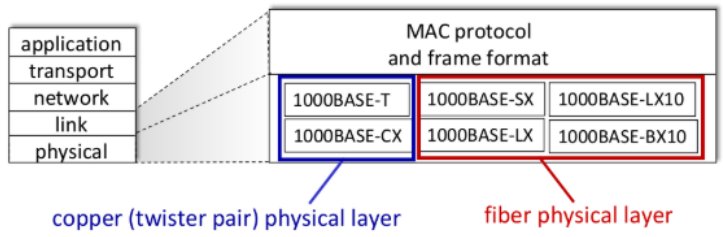
- 10BASE-T
- 100BASE-TX
- 1000BASE-T (gigabit Ethernet)

✔ Fiber

- 1000BASE-SX (short-range fiber)
- 1000BASE-LX (long-range fiber)
- 10GBASE-LR, 40GBASE-SR, 100GBASE-SR4, etc.

✔ Key idea:

Different physical media, SAME Ethernet frame format.

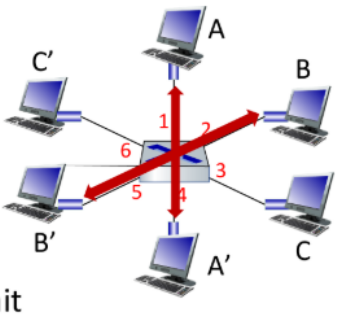


🌟 Ethernet Switch

- **Stores and fwds ethernet frames!**
- Examine the destination MAC in each frame
- Fwd frames out of the correct port
- Use **CSMA/CD only on half-duplex links**
- **Transparent** → hosts don't know switches exist
- **Plug-and-play** → no configuration required
- **Self-learning** → builds its own forwarding table automatically

⇒ Switches allow **multiple pairs of hosts** to communicate **at the same time** without collisions.

- Each host has a **dedicated link** to the switch
- Each link is its **own collision domain**
- Links operate in **full-duplex** → no CSMA/CD needed
- Switch **buffers** packets internally
- Soo, hosts can send simultaneously and no collisions will occur!



nit

switch with six interfaces (1,2,3,4,5,6)

"A-to-A' and B-to-B' can transmit simultaneously without collisions."

- 🔹 A sends data to A'
- 🔹 B sends data to B'

These two transfers use **different input/output port pairs:**

- A → A' (port 1 → port 4)
- B → B' (port 2 → port 5)

➡️ No conflict. No collision.

➡️ Switch can handle both at the same time because each flow uses independent switch paths.

❌ "But A-to-A' and C-to-A' cannot happen simultaneously."

Why? Because both flows need the SAME destination port (port 4):

- A → A' uses port 4
- C → A' also needs port 4

⚠️ **Problem:**

A switch cannot send two frames out through the same port at the same time.

🔹 3. Switch Forwarding Table (a.k.a. MAC Table)

Each switch keeps a table with entries:

MAC Address	Interface	TTL

Example:

If host **A'** is reachable via port 4 → the table will have:

| A' | 4 | 60 |

🔹 4. How Does a Switch Learn? (Self-Learning Mechanism)

When a frame arrives on a port:

Step 1 — Learn the sender's MAC

Switch records:

```csharp
Sender MAC → Incoming interface
```

Step 2 — Look up destination MAC

- If **destination MAC is known** → forward *only* to the correct port
- If **unknown → flood** → send out all ports except the incoming one

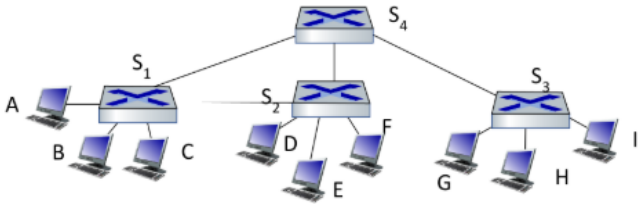
When a frame arrives:

1. **Record** (learn) sender MAC + port
2. **Look up** destination MAC in table
3. **If match found**:
   - If destination is on same port → **drop** (why send back?)
   - Else → **forward** on correct port
4. **If no match → flood** (broadcast on all ports except incoming)

✅ What does "interconnecting switches" mean?

Instead of having one big switch, we can connect **multiple switches together** to form a larger LAN.
This allows many more hosts to communicate without all being on a single physical device.

Switches use **self-learning** and **forwarding tables (MAC address tables)** to figure out where to send frames — even across multiple switches.



⇒ S4 is the **core switch!**

C sends a frame to I.

Step-by-step switch learning:

At S1:

- Learns `C → interface 3`
- Doesn't know I → floods toward S2 and S4

At S4:

- Learns `C → interface toward S1`
- Floods to S3 ↑ S2

At S3:

- Learns `C → interface from S4`
- Destination = I (directly attached) → forwards to I

When I replies to C:

The reverse path becomes known:

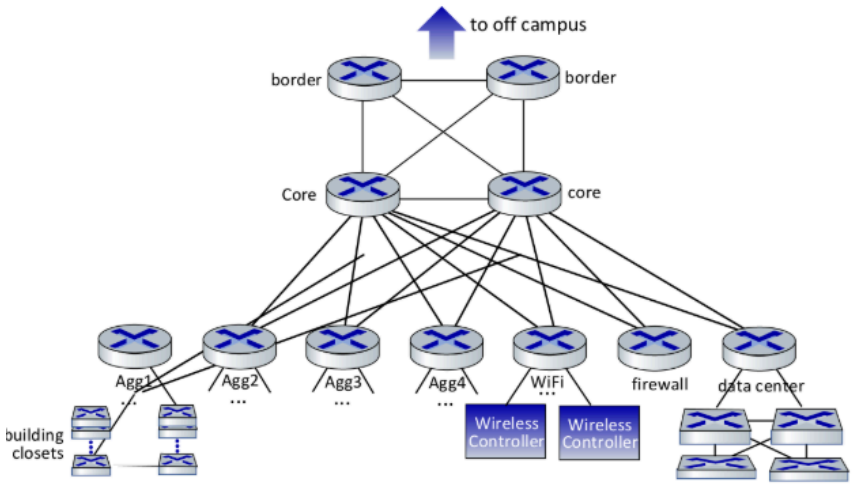
- S3 learns `I → port`
- S4 learns `I → S3 side`
- S2 learns `I → S4 side`
- S1 learns `I → S2 side`

Final state:

All switches now have complete MAC learning.

🏫 UMass Campus Network

⇒ A university campus network must support **tens of thousands** of users—students, staff, faculty—across many buildings. That requires a **large, hierarchical network design**.


to off campus
border
border
Core
core
Agg1
Agg2
Agg3
Agg4
WiFi
firewall
data center
building
closets
Wireless
Controller
Wireless
Controller

✔ Three-Layer Hierarchy

1. **Core Layer (center of network)**
   - Very fast switches
   - Connects all buildings together
   - Provides backbone (40 Gbps – 100 Gbps links)
2. **Aggregation Layer (Agg1, Agg2, Agg3, Agg4…)**
   - Each building or group of buildings has aggregation switches
   - They collect (aggregate) traffic from many access switches
   - Send it upward to core
3. **Access Layer (building closets)**
   - Closest to users
   - Switches where PCs, printers, APs plug in
   - Thousands of access switches across campus

✔ Border Routers (top of diagram)

- Connect the campus to the **Internet**
- Use **eBGP** (external BGP)
- Very high-speed connections (10G → 100G)

✔ Additional Components

- **Firewalls** for security
- **Wireless Controllers** to manage thousands of WiFi APs
- **Data Center** with servers and storage

✔ Scale of UMass Network

- 2000+ switches
- 6000+ wireless access points
- 30000 wired ports
- 55000 active wireless devices

✅ Switches vs Routers

→ **both forward packets**, but at different layers and using different logic.

🔵 1. Both Are Store-and-Forward Devices

They receive a frame/datagram → store it → forward it out the correct interface.

✔ **Switches**

- Work at **Layer 2 (Data Link Layer)**
- Look at **MAC addresses**
- Forward **frames**

✔ **Routers**

- Work at **Layer 3 (Network Layer)**
- Look at **IP addresses**
- Forward **datagrams (packets)**

🔵 2. Both Have Forwarding Tables — But Built Differently

✔ **Switch Forwarding Table (MAC Table)**

- Built **automatically (self-learning)**
- Behavior:
  1. Switch sees a frame from MAC X on port 3 → learns "MAC X is on port 3"
  2. If destination MAC is known → forward only to correct port
  3. If unknown → FLOOD to all ports
- No manual configuration needed

✔ **Router Forwarding Table (Routing Table)**

- **Computed using routing algorithms**
  - OSPF, IS-IS, BGP
- Contains IP prefixes and next hop information
- Carefully configured and controlled
- No flooding

⭐VLAN: Single Broadcast Domain

- All devices belong to **one broadcast domain**.

Problems when LAN grows:

1. **Scaling issue**
   - Broadcast traffic (ARP, DHCP, unknown MAC frames) must travel across the *entire* LAN.
   - More users → more broadcast → **network slows down**.
2. **Security & Privacy issue**
   - Any device can potentially "hear" broadcast traffic from the entire network.

A CS user moves to the EE building, but logically wants to remain part of CS.

Why this is a problem

In a normal LAN:

- Your network membership = **the port you plug into**.
- If you move physically → you change your LAN.

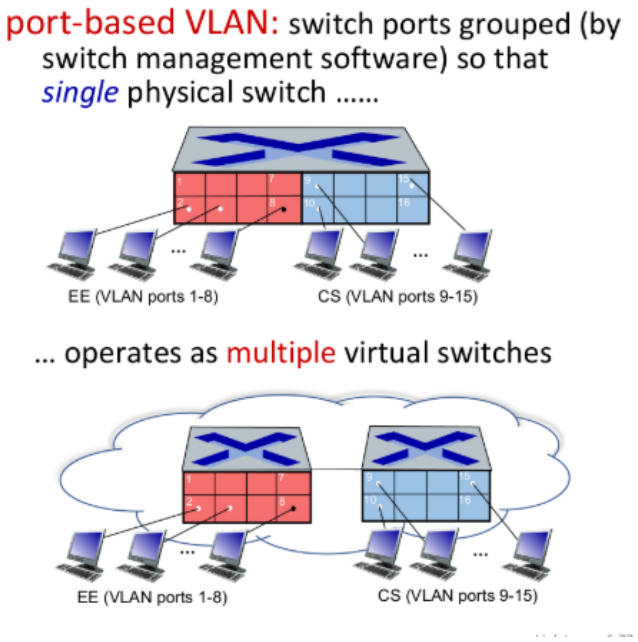
Why VLANs help

VLANs allow you to stay in CS **logically** even if physically connected to a different switch.

⭐Port-Based VLANs

– A **Virtual Local Area Network (VLAN)** divides one physical LAN into **multiple virtual networks**.

- Devices in different VLANs:
- **Cannot communicate** without a router
- Do not receive each other's broadcast traffic.



^^ Even though there is **one physical switch**, it behaves like **two separate switches**.

1. Traffic Isolation

- Ports in VLAN 10 cannot talk to VLAN 20 unless routed.
- Broadcasts stay inside the VLAN → better security & performance.

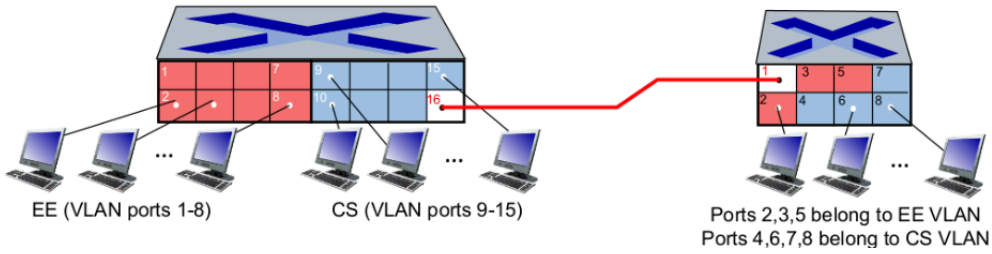
2. Dynamic Membership

- Admin can move devices between VLANs through software—no need to rewire cables.

3. Forwarding Between VLANs

- Must be done using a **router** (Layer 3 switch).
- VLANs behave like separate subnets.

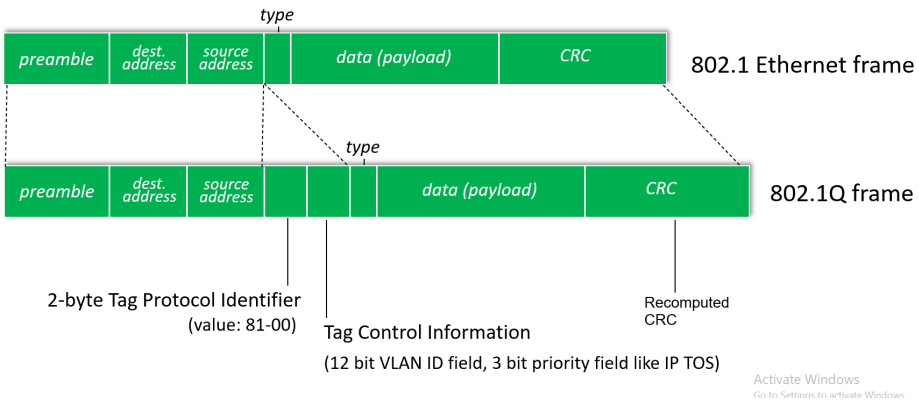
⭐ VLANs Across Multiple Switches (Trunking)



⇒ **trunk port** (red link) carries frames between VLANs defined over multiple switches

- A normal Ethernet frame does **not** contain VLAN ID, so for this a trunk port uses **802.1Q tagging** to add VLAN ID into the frame.
- Trunk port carries frames from multiple VLANs on 1 link
- Frames traveling between switches must carry a **VLAN tag**.

802.1Q VLAN frame format



⭐ EVPN: Ethernet VPNs (aka VXLANs)

Q. What problem does EVPN/VXLAN solve?

- A company may have **two distant data centers**
- They want both sites to behave like **one single Layer-2 network** (same VLANs, same broadcast domain).
- But they are connected through the **Internet (Layer-3)** — NOT a LAN.

➡️ **VXLAN is used to "stretch" Layer-2 across Layer-3 networks.**

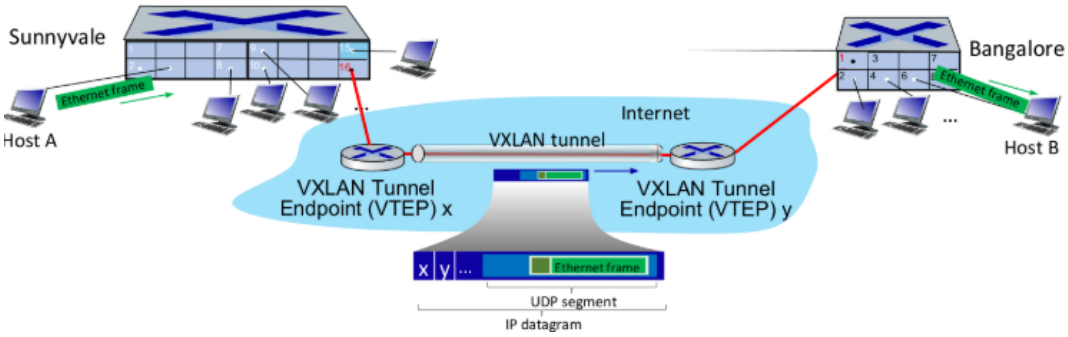
- VLANs don't scale across long distances, solution? VXLAN

It makes two remote Ethernet switches **appear as if they are directly connected.**

1️⃣ Layer-2 switches logically connected using a Layer-3 underlay

- The physical network is **IP-based (Layer-3)**.
- But VXLAN creates a **virtual Layer-2** connection over it.

This is like building a LAN on top of the Internet.



⭐ EVPN in easy words

EVPN is a technology that allows two or more far-away locations—like two data centers in different cities—to act as if they are part of the same local network. Normally, devices can only share a Layer-2 network (same VLAN, same broadcast domain) if they are physically connected. EVPN solves this by creating a *virtual* Layer-2 connection over a Layer-3 IP network such as the Internet. It does this by taking normal Ethernet frames and "tunneling" them inside IP packets, sending them across long distances, and then delivering them as regular Ethernet frames on the other end. Because of this, servers in different data centers can stay in the same VLAN, keep their MAC addresses, move around without changing their IP, and communicate just like they were in the same building. In short: **EVPN stretches a LAN across different locations**, making multiple sites behave like one big network.

[6.6]data center networking

- A **data center** contains **tens of thousands of servers** placed very close to each other. Big companies like **Amazon, Google, YouTube, Apple, Microsoft** use them to run websites, apps, storage, and AI systems. Because these servers handle **massive numbers of users at the same time**, the network inside the data center must be **very fast, reliable, and scalable**.

⭐ 1. What makes data center networking challenging?

✔ Multiple applications

A single data center runs many services (video streaming, search, payments, backups). All generate huge traffic.

✔ Reliability

If one server or link fails, traffic must automatically reroute. Downtime is not acceptable.

✔ Load balancing

Traffic must be spread across thousands of servers to avoid overload and bottlenecks.

⭐ 2. Key network components inside a data center

Border Routers

These connect the data center to the **outside world (Internet, other data centers).**

Core (Tier-1) Switches

Fast backbone switches. They connect all aggregation switches below them.

Aggregation (Tier-2) Switches

Middle level. Each aggregation switch connects to many Top-of-Rack switches.

Top-of-Rack (TOR) Access Switches

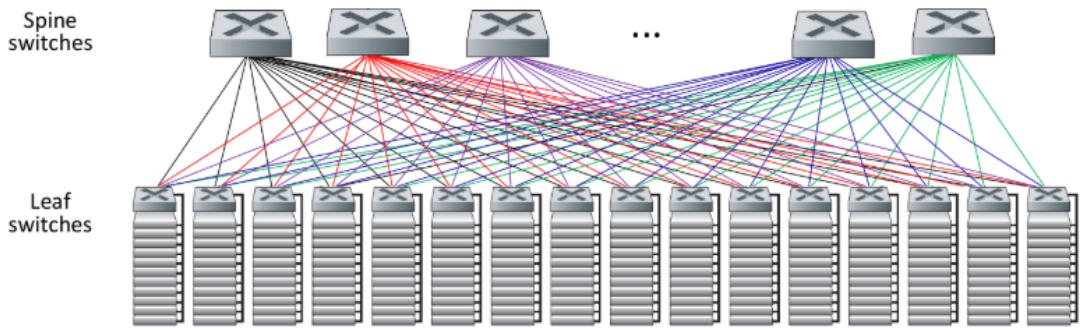
Every server rack has 1 TOR switch.

It connects **20–40 servers** in that rack and uplinks to aggregation switches.

Server Racks

Where physical servers live (blades). → hosts

⭐ Leaf–Spine Architecture (modern data centers)



- Instead of a 3-tier structure, modern data centers use a **2-tier leaf-spine** design

✔ Leaf switches = Access switches

Connect directly to servers.

✔ Spine switches = Core switches

Every leaf connects to *every* spine.

Benefits:

- Equal path length between any two servers (low latency)
- Many parallel paths → high throughput
- If one link fails, others carry traffic → high reliability

🌐 Facebook Datacenter Network (F16 Topology)

- **massively scalable network**
- connects **hundreds of thousands of servers** with **high bandwidth** and **no single point of failure**.

1️⃣Top-of-Rack (TOR) Switches

- Every server rack has one TOR switch.
- *ground floor* where traffic starts.

2️⃣ Fabric Switches

- Middle layer that **interconnects all TOR switches to all spine switches.**
- Provides ***load-balanced paths*** for traffic.
- Ensures that data can travel multiple ways without congestion.

3️⃣ Spine Switches

- Very high-speed switches at the top.
- **Every fabric switch connects to every spine switch.**
- They provide the **core connectivity** of the whole data center.

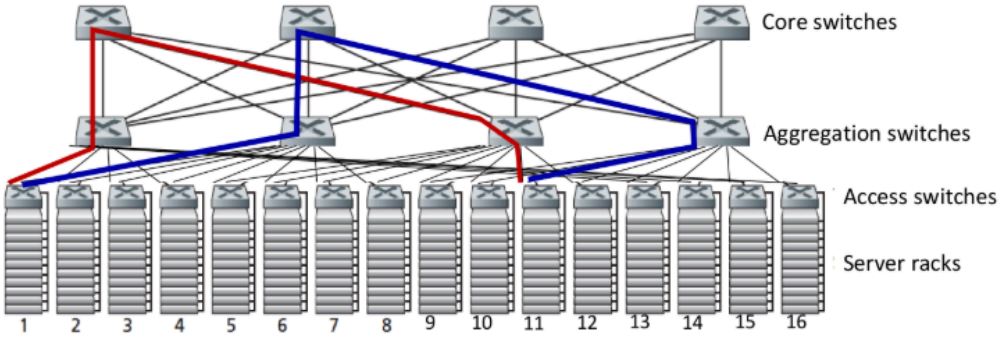
⭐ 4. Multipath Routing (why so many links?)

Data centers use **multiple disjoint paths** between server racks.

Benefits:

- **Higher bandwidth** (traffic spreads across many links)
- **Fault tolerance** (failure of one link doesn't disrupt communication)

Protocols like **ECMP (Equal Cost Multi-Path)** are used.



two disjoint paths highlighted between racks 1 and 11

⭐ 5. Application-Layer Routing (Load Balancers)

Before traffic enters the data center, it often hits a **load balancer**.

The load balancer:

- Receives requests from clients (Internet)
- Chooses which server should handle it
  (based on CPU load, delay, health, location)
- Sends the result back to the client, hiding internal details

This is how services like Google, Netflix, and Amazon serve millions of users.

⭐ 6. Protocol Innovations in Data Centers

Link Layer:

- **RoCE** – RDMA over Converged Ethernet
  Allows servers to access each other's memory directly with very low latency.

Transport Layer:

- **ECN (Explicit Congestion Notification)**
  Helps prevent congestion by marking packets instead of dropping them.
- **DCQCN, DCTCP** – congestion control optimized for data centers.

Routing & Management:

- **SDN (Software Defined Networking)** used heavily
  Allows centralized control of routing, traffic engineering, and policies.
- Data center tries to place related applications **physically close** (same rack or same row) to reduce latency.

⭐ 7. Example: Google's ORION SDN

Google uses an advanced SDN control plane called **ORION** for:

- Internal data center (Jupiter network)
- Global WAN traffic (B4 network)

What ORION does:

- Manages routing inside the data center
- Optimizes traffic flows
- Provides QoS guarantees
- Uses microservices for monitoring and configuration