# 441 Final Notes

## Contents

# Chapter 3: Transport Layer

## TCP

At the transport layer, packets are delivered as segments. When client-server are connected via TCP connection, the communication is bi-directional (even though there's only one connection)
**TCP Data Segment:**

- *Piggy backing:* a technique used by the **server** when setting up a TCP connection
  - Client opens a TCP connection (request) with a server
  - Server responds with an ACK (successful connection) and ontop of that, can send some data back
- *Sequence numbers:* Byte stream number of first byte in segment's data
- *Acknowledgements:* Sequence number of next byte expected from other-side (cumulative ACK, similar to GBN)
- e.g. Host A and Host B sending data
  1. $A \to \text{Seq} = 92,\ \text{data} = 8 \text{ bytes} \to B$
  2. $B \to \text{ACK} = 100 \to A$
  3. $A \to \text{Seq} = 100,\ \text{data} = 20 \text{ bytes} \to B$
  4. $B \to \text{ACK} = 120 \to A$
- TCP does not handle out-of-order packets, up to the implementors (discarding, buffering, etc)

To set up TCP's timeout value, it should be longer than RTT (even though it varies)

- If t/o is too short $\to$ premature t/o $\to$ unnecessary retransmissions
- If t/o is too long $\to$ slow reaction $\to$ result in segment loss

To estimate RTT, we use **Sample RTT:**

- measured time from segment transmission until ACK reception (ignore retransmissions)
- Will vary (changes from packet to packet)
- Gaussian distribution to set RTT, a moving average

$$\text{EstRTT} = (1 - \alpha) \times \text{CurrEstRTT} + \alpha \times \text{SampleRTT}$$

- Typical $\alpha = 0.125$

- SampleRTT deviation from EstimatedRTT:

$$\text{DevRTT} = (1-\beta) \times \text{ DevRTT} + \beta \times |\text{SampleRTT} - \text{EstRTT}|$$

- Typical $\beta = 0.25$

$$\text{T/oInterval} = \text{EstRTT} + 4 \times \text{DevRTT}$$

- Where EstRTT is the mean, DevRTT is the STD

TCP RDT at the **sending side:**

1. *Data from application layer:*
    - Create a segment with sequence number (goes into window)
    - Start timer if not running already (begins with oldest unACKed segment)
2. *Timeout:*
    - Retransmit segment that cause t/o (not entire window)
        - Restart timer $\rightarrow$ EstRTT $+ 4 \times$ DevRTT
    - Expiration interval: TimeoutInterval (computed using SampleRTT)
3. *ACK received:*
    - If ACK acknowledges previous unACKed segment
        - Update what is known to be ACKed
        - Start timer if there are still unACKed segments

Connection management in TCP: opening and closing a connection (handshake between sender/receiver)

- **Opening:** 3 messages
- **Closing:** 4 messages

# Chapter 4: Network Layer (Data Plane)

In this section, we are concerned with the **data plane (forwarding)** portion of the network layer.
**Data plane**

- Also known as **fowarding**: Function is to move packets from the router input link to the output link
- *Analogy:* The process of getting through a single interchange (turn)
- All information to do forwarding is in the router (localized operations)
- Determines how a datagram is forwarded to the appropriate output link

## Router

As mentioned above, the data plane forwards packets (datagrams at this layer) to the router's input port and out the router's output port. Functionalities of the input and output port are as follows:

- **Input**
  - Bit-level reception (physical layer )$\rightarrow$ Ethernet (link layer) $\rightarrow$ decentralized switching (network layer)
  - At **decentralized switching**, use header field values to look up output port using the **forwarding table** in the input ports memory
  - 2 approaches to **forward** datagrams:
    - **Destination-based forwarding:** Forward based on destination IP address (traditional)
      - Internet tries to match IP using **longest prefix matching**
    - **Generalized forwarding:** Forward based on any set of header field values (implemented in SDN)
  - If datagrams arrive faster than forwarding rate, queue into **switch fabric**
    - Transfers datagrams from input buffer to appropriate output buffer
    - Switching rate is the rate at which packets can be transferred from in to out
    - 3 types: *memory, bus, crossbar*
- **Output**
  - **Buffering** implemented when datagrams arrive from the switch fabric faster than the transmission rate

- **Scheduling discipline** chooses among queued datagrams for transmission (FIFO by default)
- Datagrams could be **lost** due to network **congestion** (lack of buffers)
- **Priority scheduling:** who gets best performance (network neutrality)

Q: So, how much buffering do we need?
A: Minimum amount of buffer because it is costly and consumes a lot of energy

- A good **rule of thumb** with regards to buffering:
  * Average buffer = "typical" $RTT$ ( 250 ms) $\times$ link capacity $C$
  * E.g. $C = 10$ GBps, $RTT = 250$ ms $= 0.25$ s, $w =$ max cwindow

$$w = C \times RTT = 10 \times 0.25 = 2.5 \text{ GBit buffer}$$

## Internet Protocol (IP)

A key function of an IP datagram is **fragmentation** and **reassembly**. These functions exist because datagrams are large in terms of overhead, so links (at the link layer) are able to transmit these packets.

- Links have a **Max Transfer Unit** (MTU), which is the maximum size of a link **frame**
- This means that datagrams>MTU have to be divided, via **fragmentation**
  - One datagram becomes several smaller datagrams
  - Once it reaches the destination, datagram is then **reassembled**
- IP headers are used to identify fragments that are in order

### Subnets

Are equivalent to link layer LANs (a group of networks).

- Use routers to connect networks together
- IP address is split into 2 parts [subnet address|host address] (high order, low order respectively)
- Hosts that are connected together without a router forms a **subnet**
  - They share the same *subnet address*, also known as the prefix
  - e.g. 223.1.1.1, 223.1.1.2, 223.1.1.3, etc.
  - A separate subnet could look like: 223.1.2.1, 223.1.2.2, 223.1.2.3, etc.
  - These subnets can be connected to the same router via different interfaces (of the router)
    - Connected via wired (Ethernet) or wireless (Wifi)
- As mentioned above, each interface of a router could form a subnet
- $2^8 - 2$ subnet addresses could be assigned to hosts, while there are 2 reserves:
  - **Zero address:** host portion all 0's
  - **Broadcast address:** host portion all 1's

A protocol called **CIDR (Classless Inter-Domain Routing)** exists:

- Subnet portion can be of any length
- IPv4 introduced classless to be able to allocate IP addresses more efficiently due to an IP address shortage
- Format: a.b.c.d/x where

  - x: number of bits in the subnet portion
  - Subnet portion: blue, host portion: black
  - e.g. 11001000 00010111 00010000 00000000 → 200.23.16.0/23
  - 23 indicates that there are 9 bits that can be assigned as host

The way an IP address can be obtained is through an ISP. ISPs have a range of addresses that can be purchased

- ISPs condense their block into smaller blocks to give to organizations
  - ISP block: 11001000.00010111.00010000.00000000 (200.23.16.0/20)
  - Organization 1 block: 11001000.00010111.0001**000**0.00000000
  - Organization 2 block: 11001000.00010111.0001**001**0.00000000
  - Organization 3 block: 11001000.00010111.0001**010**0.00000000
  - ...
  - Organization 8 block: 11001000 00010111 0001**111**0 00000000
- In this scenario, each organization will have 9 bits remaining to give to hosts, so $2^9$ IP addresses can be distributed in each condensed (organization) block

ISPs get their block of addresses from an organization called ICANN (Internet Corporation for Assigned Name and Numbers)

- World wide organization in charge of IP address management
- Allocate addresses
- Manage DNS
- Assign domain names, resolve disputes between domain names

**DHCP**

Hosts obtain an IP address either manually (hardcoded by the system admin) or via **DHCP (Dynamic Host Configuration Protocol)**.

- A "plug-and-play" protocol, so we don't need to intervene (only server-side)
- Purpose is to allow hosts to dynamically obtain IP addresses from network servers when they join
  - Since addresses can expire, a host can renew its lease with current address
  - Can lease out addresses (if connected or "on")
  - Once a host disconnects, address goes back into a pool of available IP addresses
- It is an **application layer** protocol that uses **UDP**
- In action:
  1. DHCP server formulates DHCP ACK containing client's IP, first hop router IP (for client), IP and name of DNS server
  2. Encapsulation of DHCP server, forward to client
  3. Client now knows its own IP, IP of first hop router, IP and name of DNS server

When we don't know the IP of a DHCP server, we use a special IP called the **broadcast address** (mentioned earlier).

- Example IP: 10101100.00100000.00000000.00000000 (172.16.0.0/11)
- Broadcast addr: 10101100.00111111.11111111.11111111 (172.31.255.255)
- How it works:
  1. Client **broadcasts** a DHCP request
  2. All DHCP servers discover this request, all respond
  3. Client chooses DHCP server to get IP from then **broadcasts** request
  4. Chosen DHCP server assigns IP, responds with a DHCP ACK

**NAT**

Since IPv4 addresses are scarce (shortage of them), the goal of **NAT (Network Address Translation)** is to be able use **one** IP address for an entire subnet

- That is, one NAT IP source address of all packets leaving a subnet (at router interface)
- Private IP address blocks:
  - Reserved by IANA for private Internets (not routable on global Internet)
    1. IP: 10.0.0.0/8 $\rightarrow$ 10.255.255.255 ($2^2$4 usable IP)
    2. IP: 172.16.0.0/12 $\rightarrow$ 172.31.255.255 ($2^2$0 usable IP)
    3. IP: 192.168.0.0/16 $\rightarrow$ 192.168.255.255 ($2^1$6 usable IP)
- **Advantages:**
  - Do not need a block of addresses from an ISP (1 for all devices)
  - Can change addresses of devices in LAN without notifying the outside world
  - Can change ISP without changing addresses of devices in LAN (private)
  - Devices in LAN not explicitly addressable (not routable, good security)
- How it works:
  1. Sender creates a packet with src IP, port and dest IP, port
  2. Goes through a router. In the router, it:
     - Records the local (private) IP and port number
     - Replaces it with a public IP and random port number, then forward to destination
  3. Receiver responds back with src IP and dest IP (public IP that was used)
  4. Goes through router, router changes dest address back to local (private) IP

### IPv4

The Internet standard (for now).

- *IP:* 32-bit identifier for host and router interfaces
- *Interface (port):* Connection between host/router and physical link
  - Routers have multiple interfaces (router ports)
  - Hosts have one or two (wired Ethernet, wireless Wifi)
  - NIC (hardware component)
- Multiple interfaces means multiple IP addresses
- Represented in dotted decimal notation

### IPv6

Slowly being implemented in the Internet, but not fully because people found ways to keep IPv4 alive (using CIDR, NAT, etc).

- Instead of 32-bit addresses (IPv4), it uses 128-bit addresses
- Changes in the header format helped speed up processing and forwarding of packets
  - Fixed 40-byte header
  - No fragmentation
  - No checksum
- Since IPv4 is deployed everywhere (and IPv6 isn't), it's hard to make the switch immediately
- But, it does coexist with IPv4 through a method called **tunneling** (VPN is tunneling)
  - IPv6 datagram carried as a **payload** in an IPv4 datagram among IPv4 routers
  - IPv4 datagram essentially encapsulates an IPv6 datagram
- How it works:
  - Say we have 2 hosts running the IPv6 protocol, but in between there are strictly IPv4 routers
  - A **dual stack** router exists, where it is able to handle IPv6 and IPv4 datagrams
  - Host is connected to the IPv6 interface, IPv4 interface of the **dual stack** is connected to however many IPv4 routers are in between (creates the **tunnel**)

- On the other end of the IPv4 router interface connects to another **dual stack**, which then connects to the other IPv6 host
- Logically, the **tunnel** acts as a channel that connects the two dual stack routers
- Encapsulation/de-encapsulation of the IPv6 datagram occurs at the IPv4 interfaces of the dual stack

## Generalized Forwarding and Software-Defined Networking

Generally, routers only care about the destination address (Generalized forwarding) of packets, but SDN routers look at a combination of things and decides how it forwards these packets.
**SDN (OpenFlow)**

- **OpenFlow** is an API used by SDN controllers to communicate with routers and applications running on network controllers
- Flow is defined by headers
- Generalized forwarding within SDN: simple packet handling rules
  - Pattern: match values in packet header fields
  - Actions for matched packet: drop, forward, modify, or send to controller
  - This cannot be done in traditional routers
  - Examples of pattern packet must match:
    1. Drop: $src = 1.2.*.*$, $dest = 3.4.5.*$
    2. Forward: $src = *.*.*.*$, $dest = 3.4.*.*$
    3. Send to controller: $src = 10.1.2.3$, $dest = *.*.*.*$

# Chapter 5: Network Layer (Control Plane)

In this section, we are concerned with the **control plane (routing)** portion of the network layer.
**Control Plane**

- Also known as **routing:** To determine the route taken by packets from the source to destination
- *Analogy:* Process of planning a trip from home to the airport
- Has network-wide logic
- Two approaches to this:
  - **Traditional routing algorithms:** Implemented in routers and the most dominant (per-router control)
    - Individual routing algorithms in every router interact with each other (to compute forwarding tables)
  - **Software-Defined Networking:** Implemented in (remote) servers (logically centralized control)
    - Distinct controller interacts with its local Control Agents (CA) in routers (to compute forwarding tables)

## Routing Algorithms

Goal is to determine a *'good' path* from src to dest via routers

- *good:* least cost, least congested, fastest
- *path:* sequence of packets travese via routers from src to dest
- Classification of routing algorithms:
  - Information:
    - **Global (LS)**: All routers have complete topology and link cost information
    - **Decentralized (DV)**: Routers know physically connected neighbors; is an iterative process where they can exchange information
  - Static/Dynamic
    - **Static**: Routes change slowly over time
    - **Dynamic**: Periodic update in response to link cost changes

**Link State: Dijkstra Algorithm**

- Nodes (routers) know the entire topology as well as link costs to every other node
- Algorithm computes least cost paths from source node to all other nodes
- It is an **iterative** process
- Notation:
  - $c(x, y)$: link cost from node $x$ to $y$, $c = \inf$ if not direct neighbors
  - $D(v)$: current value of cost of path from source to dest $v$
  - $p(v)$: predecessor node along path from source to dest $v$
  - $N'$: Set of nodes whose least cost path is definitively known (keeps track of least cost paths)
- Algorithm takes $O(n^2)$ time

**Distance Vector (DV): Bellman-Ford Equation**

- Dynamic programming (constant update)
- Each node has its own (as well as a copy of its neighbors) **distance vector** (DV)
  - Nodes exchange this information periodically
  - When a node receives a **new DV estimate** from its neighbor, updates its own DV via BF equation
  - When enough iterations are done, $Dx(y)$ will eventually become $dx(y)$
- At each node,
  - **Wait** for change in local link cost/message from neighbor
  - **Recompute** estimates
  - If DV to any dest has changed, **notify** neighbors
- Notation:
  - $dx(y)$: cost of least-cost path from $x$ to $y$, where

    $$dx(y) = min\{c(x, v) + dv(y)\}$$

  - $min$: minimum taken over all neighbors $v$ of $x$
  - $c(x, v)$: cost to neighbor $v$
  - $dv(y)$: cost from neighbor $v$ to dest $y$
  - $Dx(y)$ (capital D): estimate of least cost from $x$ to $y$

**Conclusion between LS and DV**

Overall takeaway:

- **Link State**
  - *Message complexity:* Each node broadcasts link info to every other node
  - *Speed of convergence:* Very fast, $O(n \log n)$
  - *Robustness:*
    - Node can advertise incorrect link cost
    - Each node computes its own table
    - More robust compared to DV
- **Distance Vector**
  - *Message complexity:* Exchange between neighbors only over several iterations
  - *Speed of convergence:* Generally slower than LS
  - *Robustness:*
    - Nodes can advertise incorrect link cost
    - Each node's table used by another, error propagates through network

## Intra-AS Routing

Aggregate routers into regions known as "Autonomous Systems" (network), specifically **Intra-AS** routing. Forwarding table configured by intra/inter-AS algorithm

- Determine entries for destinations within AS (and external)
- Routing among hosts and routers in the same AS
- All routers in the AS must run the **same** intra-domain protocol
- Routers in **different** AS can run different intra-domain protocols
- **Gateway router:** at the "edge" of its own AS, links to routers in other AS's

A few protocols to note:

- **RIP (Routing Information Protocol):** one of the oldest, uses DV routing
- **IGRP (Interior Gateway Routing Protocol:)** CISCO, not used that much
- **OSPF (Open Shortest Path First:)** Same as IS-IS protocol, widely used

### Open Shortest Path First (OSPF)

- *Open:* publicly available
- Uses LS algorithm
  - Has all the same features as LS (topology of routers, dijkstra's algorithm to compute)
- *Flooding:* OSPF LS broadcasts to all other routers in AS over IP

## Inter-AS Routing

Aggregate routers into regions known as "Autonomous Systems" (network), specifically **Inter-AS** routing. Forwarding table configured by intra/inter-AS algorithm.

- Determine only external destinations for entries
- Routing among AS's
- **Gateways** perform inter-domain routing (as well as intra-domain)

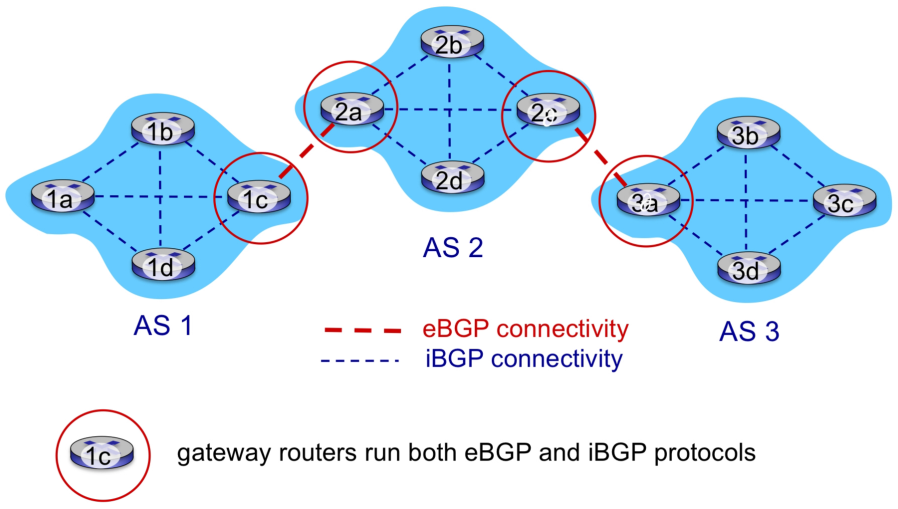
Main protocol used in inter-AS routing:

- **BGP (Border Gateway Protocol):** "Glue that holds the Internet together"

### Border Gateway Protocol (BGP)

Biggest protocol (BGP) that makes the Internet, along side IP

- *eBGP (external):* obtain subnet reachability information from neighboring AS's routers
- *iBGP (internal):* propagate reachability information to all AS-internal routers
- Goal is to determine good routes to other networks based on **reachability information** and **policy**



- *BGP session:* Two BGP routers exchange BGP messages over a TCP connection

- It is a "path vector" protocol - it **advertises** paths to different destination network prefixes
- Important attributes with regards to advertised prefixes (prefix + attribute = "route")
  - **AS-PATH:** list of AS's that has to be traversed to reach destination
  - **NEXT-HOP:** Gateway router that advertised the route
- For a node (from one AS) to reach another node (of a different AS), it has to go through **gateway routers**
  - Nodes learn this through advertised paths (mentioned above), which could present multiple paths
  - A specific path is determined by a gateway router's policy
    - If there are no specifications (by the gateway router), then the node chooses the shortest **AS-PATH**
- Hence, BGP **route selection** priority is as follows:
  1. Local policy
  2. Shortest AS-PATH
  3. Closest NEXT-HOP router locally (not potato routing)
  4. Additional criteria

**Hot Potato Routing**

- Chooses local gateway that has the least intra-domain cost

# Chapter 6: Link Layer

## Overview

The link layer is the second layer of the Internet Protocol stack. Some key words of this section include:

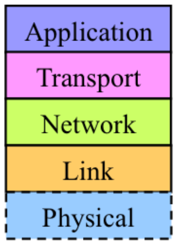


Figure 1: IP Stack

- **Nodes** are what hosts and routers are referred to
- **Links** are communication channels that connect adjacent nodes along a communcation path
  - *Wired* links (e.g. Ethernet)
  - *Wireless* links (e.g. Wifi)
- **Frames** are what packets are called (encapuslates a network **datagram**)

The link layer is responsible for transferring a datagram from one **node** to another adjacent **node** via a **link**. Each link layer protocol (LLP) provides its own services (*wireless wifi link, fiber optic link, copper link, etc*) but are **not guaranteed** to provide RDT/flow control, etc., over the link (upper layers are responsible).
As mentioned earlier, packets sent over to the link layer is called a link **frame**

- It encapsulates a network **datagram** into a **frame**, adding its own header fields
- Allows channel access if the medium is shared (e.g. shared link)
- Identifiers in **frame** headers are **MAC addresses** for source/destination (Completely different from IP!)

Reliable delivery between adjacent nodes were discussed in the transport layer of the stack. Though some links rarely use RDT (fiber optic), *wireless* links have **very high error** rates.

Q: So, if RDT exists (TCP connection), would we still need end-end reliability?
A: Yes, because RDT at links are per hop (over nodes), so it does not guarantee data transfer over hosts.

Some services the link layer provides include:

- **Framing** and **Reliable delivery** (mentioned earlier)
- **Link access**
  - No problems for **point-to-point** links
  - Multiple Access Control (MAC) for shared links
- **Flow control**
  - Pacing between adjacent sending/receiving nodes (to prevent overflowing)
- **Error Detection/Correction (EDC)**
  - Errors caused by signal attenuation (loss of signal strength, e.g. noise)
  - *Receiver* detects presence of errors → signals sender for **retransmission** or **drop frame**
  - **Correction:** Receiver identifies and correct bit errors **without** resorting to retransmission
- **Half/Full duplex**
  - **Half:** Nodes *cannot* simultaneously send/receive transmissions → one at a time
  - **Full:** Nodes *can* simultaneously send/receive transmissions

The link layer is implemented in **every** node (host or router) via

- **Adapter:** Network Interface Card (NIC)
- **Chip:** 5G, Bluetooth, etc.

The above are attached to a system bus, and it is a combination of hardware, software, and firmware. Adapaters communicate with each other, meaning one NIC communicates with another NIC (one sends a frame while the other receives it).

- **Sender:** Encapsulates datagram in frame then adds error checking bits, RDT, flow control, etc.
- **Receiver:** Looks for errors, RDT, flow control, etc. then extracts datagram, passes it to the upper layers at receiving side

## Error Detection and Correction (EDC)

As mentioned earlier, **EDC** is one of the services provided by the link layer. Some variables to note in this protocol:

- **EDC:** Error detection and correction **bits** (AKA redundancy bits → does not carry any information)
- **D:** The data protected by error checking, can also include header fields

While this service exists, it is **not 100%** reliable. The protocol may miss some errors, but rarely. To prevent this, use larger **EDC** fields for better detection and correction. When an error is detected, frame could be dropped.
A couple of **EDC** protocols to note:

- **Single Bit Parity Checking:**
  - Detects single bit errors (cannot detect more than one)
  - Uses the XOR ($\oplus$) operation at the receiver
  - Two types:
    - *Even parity:* odd number of 1's, add 1 parity bit to make even (e.g. 10110**1**←parity bit)
    - *Odd parity:* even number of 1's, add 1 parity bit to make odd
  - **Advantages:**
    - Simple to implemented
    - Low overhead
    - Fast calculation
  - **Disadvantages:**
    - Can only detect one error (hence the name)
    - Cannot correct error
- **2D Parity Checking:**
  - Extended version of **single bit parity**, provides better detection AND does **correction** of the bit error
  - Form of a 2D array (row parity and column parity), in the form of a square matrix
  - e.g. $D = 101101101100$ (12 bits)

$$\begin{array}{cccc|c} 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 1 & 1 \end{array}$$

  - The above matrix shows 3 row parity bits on the right, 4 column parity bits at the bottom, and 1 parity bit at the bottom right corner

- In total: $EDC = 8$ additional bits along with $D = 12$ bits.
- The way it detects an error bit is if a bit is flipped, both the row and column parities will be incorrect, **pinpointing** the incorrect bit.
- **Advantages:**
  - Detects more errors than single bit parity
  - Can locate the error bit
  - Good at dealing with burst errors (confined within row/column)
- **Disadvantages:**
  - Does not detect ALL multi bit errors
  - More overhead

- **Checksum (upper layer):**
  - Adds 16 additional (redundancy) bits, making it stronger than parity
  - **Sender side** caluclation:
    - $D_1 = 0110011010101010$, $D_2 = 1100110000110011$
    - Summation then one's complement (flip sum):

| | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| **1** | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 |
| Carry out | | | | | | | | | | | | | | | | **1** |
| | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 |
| 1's comp. | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |

    - Sender puts this in UDP checksum field
  - **Receiver side** calculation:
    - Compute checksum of received **segment** (transport layer)
    - If computed checksum == checksum field value
      - *Yes:* No error (could potentially have)
      - *No:* Error detected
  - **Advantages:**
    - Simple
    - Fast
  - **Disadvantages:**
    - Not foolproof (even if no errors detected, errors could still exist)
    - Cannot correct errors

- **Cyclic Redundancy Check (CRC):**
  - Powerful error-detection protocol
  - Uses modulo 2 arithmetic (add/subtract in XOR operation)

- **Sender side:**
  - When calculating, we only care about the remainder (not the quotient)
  - *D:* Data bits (binary)
  - *G:* Generator (not an arbitrary number, shared between sender/receiver - hardcoded, given by $r + 1$)
  - *R (or r):* CRC bits ($R = \text{remainder}[\frac{D \times 2^r}{G}]$)
  - Mathematical formula: $D \times 2^r \oplus R$ (gets transmitted to **receiver**)
- **Receiver side:**
  - Computes:

$$\frac{D \times 2^r \oplus R}{G} = \frac{D \times 2^r}{G} \oplus \frac{R}{G} = R \oplus R = 0$$

- Example:
  - $D = 101110$, $G = 1001$ (because $G$ is 4 bits, $R = 000$ is 3 bits)
  - $D \times 2^r \oplus R$, where $D \times 2^r = 101110[000]$

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| D: | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| G: | 1 | 0 | 0 | 1 | | | | | |
| $\oplus$ | 0 | 0 | 1 | 0 | 1 | 0 | | | |
| G: | | | 1 | 0 | 0 | 1 | | | |
| $\oplus$ | | | 0 | 0 | 1 | 1 | 0 | 0 | |
| G: | | | | | 1 | 0 | 0 | 1 | |
| $\oplus$ | | | | | 0 | 1 | 0 | 1 | 0 |
| G: | | | | | | 1 | 0 | 0 | 1 |
| R: | | | | | | 0 | **0** | **1** | **1** |

  - Only take the first **3 bits** because that's all we need
  - For transmission: 101110[011] where $D = 101110$ and $R = 011$
  - At the receiver side, if we divide 101110[011] by $G$, we should get [000]
    - If the remainder comes out to be non-zero, that means an error has been detected
- **Advantages:**
  - Great at detecting errors (specifically burst)
  - Quite efficient
  - Standardized (Wifi, Ethernet)
- **Disadvantages:**
  - A lot more complex
  - Cannot correct errors

To conclude: *CRC, checksum, single bit parity check* **only detect errors** while *2D parity check* **detects errors and corrects them**.

## MAC Protocols

Multiple Access Control (MAC) links and protocols:

- Point-to-point (link between Ethernet switch and host, PPP for dial up access)
- Broadcast (shared wire/medium)

Single-shared broadcast channel. If two or more nodes simultaneously transmit data, it results in an interference (AKA a **collision**, which could result in packet - or frame - loss). To prevent **collisions**, we have MAC protocols:

- An algorithm that determines how nodes share a channel (when nodes can transmit)
- Communcation about channel sharing must **USE** the channel itself

Say we have a broadcast channel of rate $R$ bps. Ideally, we want:

- Active nodes (N) to share equal bandwidth, given of rate $R/N$
- A single node to be able to fully utilize the channel, at rate $R$
- Fully decentralized (no special node to coordinate transmissions, no synchronization of clocks and slots)
- Simple to implement

3 MAC protocol techniques at the link layer:

### Channel Partitioning

- Idea is to **prevent collisions**
- Divide channel into smaller pieces (hence the name)
- 2 procotols within this technique:
  - **TDMA (Time Division Multiple Access)**
    - Example: 6-station LAN
      - Each station gets a fixed length slot, where length = packet transition time
      - Unused slots go idle
      - If only slots 1, 3, 4 have packets to send, we have wasted bandwidth at slots 2, 5, 6 since slots are split equally
    - This technique is **only good** when there are nodes constantly using the channel
  - **FDMA (Frequency Division Multiple Access)**
    - Instead of time slots, stations are divided into frequency bands

    * Same idea as **TDMA**
        - Station gets a fixed frequency band (slot)
        - Unused bands go idle
    * Only good when nodes are in constant use of channels

**Random Access**

- Idea is to **detect and recover from collisions**
- Channel is **not** divided (purposely let collisions occur)
- Channels allows for full transmission rate for nodes
- A few protocols within this technique:
- **ALOHA**
    - One of the oldest and simplest protocols, stemmed from the University of Hawaii by Norman Abramson in the 70s. Idea was to be able to route between islands
    - **How it works:**
        1. User transmits whenever
        2. If two or more transmissions occur simultaneously, result in a **collision**
        3. Sender waits for an $RTT +$ fixed delay (No ACK = collision)
        4. Colliding stations retransmit packet, but stagger attempts randomly (probability) to reduce repeated collisions
    - Within this protocol, we have:
- **Slotted ALOHA**
    - All frames have the same size
    - Time is divided into equal sized slots - time to transmit **one frame**
    - Nodes only transmit at the beginning of time slots
    - Requires a clock synchronization (nodes are in sync, knows beginning and end of time slots)
    - Operation:
        * When a node **receives** a frame, put it in a buffer and transmit at the **beginning of the next** time slot
        * When a **collision occurs**, the node retransmits the frame in each subsequent time slot with a probability $p$ (system parameter) until success
        * When there is **no collision**, the node can proceed to send the frame at the next time slot
        * **Advantages:**

      - One active node can continuously transmit data at full rate of the channel
      - Decentralized (only slots in nodes need to be in sync)
      - Simple
    - **Disadvantages:**
      - Collisions occur, wasted slots (idle slots if node has nothing to transmit)
      - Clock synchronization
      - The time it takes for a node to detect a collision may be less than the time it takes for a frame to transmit
    - **Efficiency:**
      - $N$ nodes, $p$ probability, $R$ channel rate
      - Probability that a given node in a slot suceeds

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

      - Probability that *any* node succeeds

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

      - Probability for optimal efficiency (best case 37%)

$$E = (1 - \frac{1}{N})^{N-1} = \frac{1}{e} = 0.37$$

      - Throughput = $\frac{R}{e}$, Throughput for one node = $\frac{R}{e} \times \frac{1}{N}$

- **Pure ALOHA**
  - Simpler, no synchronization whatsoever
  - When frame first arrives, immediately transmit
  - Probability of collision increases - frame sent in current slot collides with frames sent in current - 1 and current + 1
  - **Advantages:**
    - Simple
    - Decentralized
  - **Disadvantages:**
    - Low throughput and high collisions
    - Wasted bandwidth
    - Delays
  - **Efficiency:**
    - Probability that a given node succeeds

$$E = p(1-p)^{2(N-1)}$$

    * Probability for optimal efficiency (best case 18%)

$$E = \frac{1}{2e} = 0.18$$

  - Main takeaway is that Slotted ALOHA is a lot more efficient than Pure ALOHA

- **CSMA (Carrier Sense Multiple Access)**
  - A way to improve **ALOHA** is to **listen** for packets in transmission
    * If a channel is sensed to be **idle**, transmit entire frame
    * If a channel is busy, defer transmission and wait for transmission of prior frame to finish
  - Although this is the idea, it does not entirely prevent collisions as **two** nodes could sense that a channel is idle (due to propagation delay and distance)
  - Example:
    1. Say nodes *A, B, C, D* are on the same channel (link)
    2. Nodes $B$ and $D$ have data to transmit, sense that the channel is idle
    3. $B$ sends a frame miliseconds before $D$ - because the nodes are of the same channel, these transmissions are broadcasted to one another - but $D$ doesn't know that since the signal from $B$ hasn't been reached to $D$ yet
    4. Though it may result in a partial collision, signals that were collided may not be recoverable
  - Two protocols under CSMA:

- **CSMA/CD (Collision Detection)**
  - Can detect collisions fairly quickly (when detected, colliding transmissions are immediately aborted$\rightarrow$no channel waste)
  - Implemented in **wired LANs** (Ethernet)
    * Measures the signal strength and compares it with the transmission signal
    * If the strengths are substantially different, collision detected
  - Harder to implement in **wirless LANs** (Wifi)
    * Signals going in and out can be overwhelmed by other local transmissions
    * Analogy: Turning on a flashlight in a well-lit room

- **Efficiency:**
  - $t_{prop}$ = max propagation delay between 2 nodes (in a LAN)

  - $t_{trans}$ = time it takes to transmit a frame (max-size)
  - Efficiency equation:

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

  - Efficiency reaches 100% when:
    * $t_{prop}$ goes to 0
    * $t_{trans}$ goes to infinity
  - Performance is a lot better compared to ALOHA

- **Advantages:**
  - Simple
  - Low overhead
  - Decentralized

- **Disadvantages:**
  - Bad under heavy load
  - Half-duplex (cannot do simultaneous transmission)

- **CSMA/CA (Collision Avoidance)**
  - Idea is to allow nodes to *reserve* a channel (avoid collisions of longer data frames)
  - Implemented in **wireless LANs**
  - **Sender and Receiver**
    1. First, **sender** transmits a small *Request-To-Send* (RTS) packet to **receiver** using CSMA
       * RTS's may collide with each other (they're short frames)
    2. **Receiver** broadcasts *Clear-To-Send* (CTS) packet in response to the RTS
       * Broadcast messages mean it's heard by all nodes in the LAN
    3. **Sender** transmits data frame
    4. Other nodes defer any transmissions
  - This is how a node *reserves* a channel - via RTS-CTS message exchange, kind of similar to a handshake that terminates all other actions

**Taking turns**

- Idea is to **implement the above 2**
- Two protocols:
- **Polling**
  - *Master-Slave* relationship between nodes
    1. *Master* node gives permission (sends a request) to a *slave* node
    2. *Slave* node responds with either data or nothing
    3. *Master* then moves onto the next node
  - *Slave* nodes have to wait for their turn - in chronological order
  - Protocol is typically used for "dumb" devices (e.g. Bluetooth devices)
  - **Advantages:**
    * No collisions
    * Fair access
  - **Disadvantages:**
    * Polling overhead (consumes bandwidth even if node has no data to send)
    * Latency (Turn based so nodes have to wait their turn)
    * Single point of failure (If *Master* fails, system fails)
- **Token Passing**
  - Idea was to get rid of the *Master*
  - It follows a ring topology, where nodes know their previous/next nodes
  - A node starts with a *control token* (packet), where they are free to transmit data
  - The token is passed sequentially (in order)
  - Used in industrial control
    * If the current holder of the token has nothing to send, immediately pass the token to the next node
  - **Advantages:**
    * No collisions
    * Fair access
  - **Disadvantages:**
    * Token overhead (consumes bandwidth even if node has no data to send)
    * Latency (Turn based so nodes have to wait their turn)
    * Single point of failure (If a node crashes with *token*)

To conclude:

- Channel Partitioning
  - TDMA: Time division
  - FDMA: Frequency division
- Random Access
  - ALOHA
  - Slotted ALOHA
  - CSMA
  - CSMA/CD (wired, Ethernet)
  - CSMA/CA (wireless, Wifi)
- Taking Turns
  - Polling (Master-Slave)
  - Token Passing

## Local Area Networks (LAN)

A LAN is technically a subnet - a group of hosts, switches, etc. The difference between the two are:

- **Subnets** are a part of the **network** layer
  - Uses IP addresses (32-bit) to forward network datagrams
  - Represented in dotted decimal notation
  - Analogy: Social Insurance Number (does not change no matter where you are)
- **LANs** are a part of the **link** layer
  - Uses MAC addresses (48-bit) to forward link frames
  - Represented in hexadecimal notation
  - Analogy: Postal address (it changes geographically)

Every host has a NIC adapter, and every NIC has a unique MAC address (arbitrary number). MAC addresses are administered by the IEEE, and manufacturers obtain them by buying a portion of MAC address space. MAC addresses do have some sort of life span (20 years), so they can be reused.

Q: Why do we need a MAC address if we already have IP?
A: Not every network layer is an IP network layer

This means that the **link** layer should work independently of the **network** layer. The **link** layer cannot assume what kind of **network** will run above it. We also have scenarios where a host doesn't have an IP address when it first joins a network (DHCP), so it'll need a way to communicate with other nodes (via MAC)

### Addressing

As we know, a network datagram has IP addresses as its *source* and *destination*. When encapsulated by a link frame, since the link layer doesn't deal with any sort of IP, it has MAC addresses in the *source* and *destination* header fields.

- Example: At the network layer, host $A$ wants to send a datagram to host $B$
  1. $A$ has $B$'s IP address, so in the datagram we have *src: IP A* and *dest: IP B*
  2. Once we pass the datagram to the link layer, it gets encapsulated into a frame (header fields go from IP to MAC, so *src: MAC A* and *dest: MAC B*)
  3. Only thing is, $A$ does not have the MAC address for $B$, so what do we do?

We use an addressing protocol:

- **ARP (Address Resolution Protocol)**
  - Every node runs ARP
  - Has a feature called the ARP table (similar to a forwarding table at the network layer)
    - Has parameters: <IP; MAC; TTL> (TTL: if mapping not used within given time, 20min)
  - **Back to the scenario: Nodes communicating in the same LAN**

  1. $A$ wants to send $B$ a packet, has its IP but not its MAC (Not in $A$'s ARP table)
  2. $A$ will broadcast an *ARP query* packet which contains $B$'s IP
     - Destination MAC address will be FF-FF-FF-FF-FF-FF, where all nodes receives this query
  3. Because the IP matches $B$'s IP, $B$ will respond to the sender, $A$, with its MAC address
     - Frame will be sent to $A$'s MAC address (unicast)
  4. $A$ will cache (save) $B$'s IP and MAC into the ARP table <IP B; MAC B; 20>

  - Nice thing about ARP is that it is a "plug and play" protocol, so no intervention needed
  - **Another scenario: Nodes communicating in a different LAN**
    - Same thing, $A$ wants to communicate with $B$, but is separated by $R$ (router), so $A \to R \to B$
    - Assume $A$ knows $B$'s IP
    - Assume $A$ knows $R$'s IP (via DHCP, where it provides information of its local DNS and Gateway address)
    - Assume $A$ knows $R$'s MAC (via ARP query with its Gateway address)

    1. $A$ creates a datagram with *src: IP A* and *dest: IP B*
    2. $A$ encapsulate datagram at link layer with *src: MAC A* and *dest: MAC R* but the payload still includes *IP B* as destination
    3. Frame is received at $R$, de-encapsulates frame at network layer, sees *IP A* as source and *IP B* as destination
    4. $R$ re-encapsulates datagram into frame but instead of *src: MAC A*, the source is the MAC of the output interface that connected to $B$'s LAN (*src: MAC R*, *dest: MAC B*)
       - Output interface from $R$ has $B$'s mapping in its ARP table in this case, otherwise send an ARP query
       - The frame still contains *IP A* as its source in the payload
    5. $B$ successfully receives frame

**Ethernet and Switches**

The most dominant wired LAN (low cost NIC, first widely used LAN technology, fast transmission rates, uses CSMA/CD).
How Ethernet works:

- An active switch exists in the center of connected hosts (via Ethernet)
- Each Ethernet link has its own protocol (nodes do not collide with each other, node per link)
- Hosts cannot see switches, think they are all directly connected to each other
- Ethernet frames (packets) of their own sort of structure:
  - Same idea: except as mentioned earlier, NIC adapters communicate with one another, where it encapsulates an network datagram into an Ethernet frame
  - Frame (at sender) includes the following:
    - *Preamble:* 7 bytes with pattern 10101010 followed by 1 byte with 10101011
      - Used to synchronize the receiver with sender (not the actual time but ticks)
    - *src and dest:* MAC addresses
    - *Type:* Indicates the upper layer protocol, usually IP
    - *Payload:* Data
    - *CRC:* Check if there are any errors (if yes, drop frame)

Although Ethernet is a direct link connection, it is said to be unreliable and connectionless

- **Unreliable:** When a frame is received, no feedback is sent between NICs
  - No ACK when frame is successfully received
  - No NAK when fram is dropped
  - This is where TCP (end-end reliability) comes into play, though errors are not very common
- **Connectionless:** No handshaking done between NICs

Ethernet has its own MAC protocol:

- **Unslotted CSMA/CD with binary backoff**
  - Nodes are able to retransmit frame via binary (exponential) backoff
  - Say we have 3 nodes: *A, B, C*
    1. Collision 1 occurs at $A$, so we want to retransmit frame

        * $A$ is given a contention window of $k = [0, 1]$
            · 0: $A$ will retransmit immediately
            · 1: $A$ will wait $k \times 512$ bits, then retransmit
        * Hence, the probability of retransmission is $\frac{1}{2}$
    2. Collision 2 occurs at $A$, want to retransmit (back to back)
        * $A$'s contention window is doubled, $k = [0, 1, 2, 3]$
            · 0: $A$ will retransmit immediately
            · Any other number: $A$ will wait $k \times 512$ bits, then retransmit
        * Probability of retransmission is $\frac{1}{4}$
    3. Collision 3 occurs at $A$, contention window doubled again, $k = [0, 1, ..., 7]$, probability of retransmission is $\frac{1}{8}$
    4. After the 10th retransmission attempt, contention window will no longer grow so $k = [0, 1, ..., 1023]$ with probability $\frac{1}{1024}$

Along with Ethernet, we have an device called a(n) (Ethernet) **switch**

- Similar functionality as a router (it stores-forwards frames)
    - Similar (if not, same) to the store-forward protocol, it selectively forwards frames given the MAC address
    - Uses CSMA/CD to access a LAN segment (link)
- As mentioned earlier, it is practically invisible to nodes
- Also a plug-and-play device
- Is self-learning
- Hosts have a dedicated and direct connection to a **switch interface** (point-point)
- Since each host has its own dedicated link, there are no collisions amongst hosts connected to a switch
- **Full duplex**, so hosts can simultaneously send/receive data without collisions
- **Switches** have a feature called a **switch table** (similar to a routing table)
    - Parameters include: <MAC address of host; Interface to reach host; TTL>
    - A **switch table** is initially empty, so it records entries every time a host sends a frame to another hosts
    - Example: $A$ (connected via interface 2) wants to send frame to $B$ (connected via interface 5)
        1. $A$ sends frame with destination $B$

2. Table records entry <$A$; 2; TTL>
3. Table does not have an entry for $B$, so it uses a technique called *flooding*
    * Switch forwards the frame to every node (except sender)
4. Table learns that $B$ is reachable via interface 5
    * Now that there are entries for $A$ and $B$, *selective forwarding* can be used (directly send frame without broadcasting)

- Same idea as interconnected switches - they become one big LAN