General Knowledge

- **Structure**: End points ↔ Comm Links ↔ Switch ↔ End points

End-points (Network Edge)	PC, Servers, Phones, ...
Comm Links (Access Net / Phys Media)	Copper, Fiber, WiFi, 5G, ...
Packet Switchs (Network Core)	Router, Switches, ...

- **High-level Structure**:

Network of Networks	Interconnected ISP
Network Protocols	Rules that govern how data exchange
Internet Standards	Ensure diff. HW & SW work together

Access Network

Digital Subscriber Line (DSL)	Use **existing** telephone lines. Direct link to telecom company's central office (**dedicated access**)
Cable Networks	Fiber coax cable: homes ↔ ISP (**shared access**)
Ethernet	Fiber optic cables. Used in institutions / company
Wireless Access Network	**Wireless LAN**: short range. E.g., WiFi **Widearea Wireless Access** E.g., Cellular

Physical Media

Coaxial Cable — (**Guided Media**): 2 concentric copper conductors
- Bidirectional, broadband (multiple channel)

Fiber Optical Cable — (**Guided Media**): Glass fiber, light pulses. High Speed
- Low err. rate. Immune electromagnetic noise

Radio Signal — (**Unguided Media**): electromagnetic signals
- Stability: reflection / obstruction / interference
- Wireless, Bidirectional

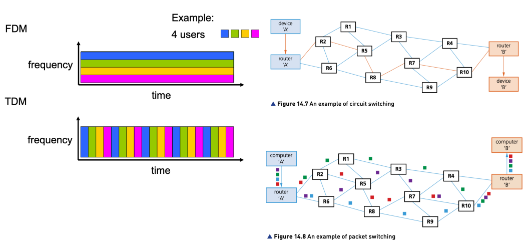
Network Core

Packet Switching:
1. Host break applicaiton-layer message into *packets*.
2. Host send packets to edge router
3. Router forward one to the next **(wait for entire packet arrive)**

Key Functions
- Routing: determine source-destination route
- Forwarding: move packet to next router (input → output)

Switching Modes

Frequency Division Multiplexing (FDM) & Time Division ... (TDM)



Transmission Loss & Delay

Delay at 1 Node: : $d_{\text{nodal}} = d_{\text{proc}} + d_{\text{queue}} + d_{\text{trans}} + d_{\text{prop}}$

d_{proc} **processing delay**: check bit error, determine output link
d_{queue} **queueing delay**: time waiting at output link (congesetion)
d_{trans} **transmission delay**: $d_{\text{trans}} = \sum_i \frac{\text{packet size (bits)}}{\text{transmission rate at hop } i \text{ (bits/sec)}}$
d_{prop} **propagation delay**: $d_{\text{prop}} = \sum_i \frac{\text{length of link } i}{\text{propagation speed in medium}}$

More on Queueing Delay:
L: packet length (bits); R: link bandwidth (bps); a: average arrival rate

Avg service time	$T = \frac{L}{R}$	Notice,
Network Intensity	$\rho = \frac{La}{R} = Ta$	$\rho \to 0, d_{\text{queue}} \to 0,$
Avg queuing delay	$d_{\text{queue}} = \frac{T\rho}{1-\rho} = \frac{L\rho}{R(1-\rho)}$	$\rho \to 1, d_{\text{queue}} \to \infty$

Packet Loss
When $La > R$: Packet queue at router. Packet lost if memory full.

Throughput
Rate (**bits/time unit**) which bits transferred between sender/receiver
- instantaneous: rate at given point in time
- average: rate over longer period of time

Theoretical Upperbound: **max-flow min-cut** of the network

Protocol Layers & Service Model

TCP/IP

Application	Application-specific functionality (**Data**) FTP, HTTP
Transport	Process ↔ process w/ports (**Segments**) TCP, UDP
Network	Host ↔ host w/IP (**Packets**) IP, routing protocols.
Link	Node ↔ node w/MAC (**Frame**) Ethernet, WiFi, PPP
Physical	**Binary Bits** "in" physical medium

ISO/OSI Reference Model

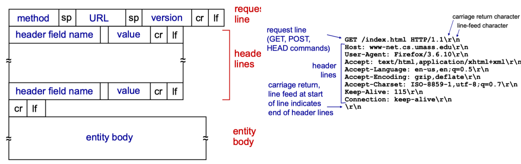
Seven Layers Application → Presentation → Session → Transport → Network → Data Link → Physical

Two more than TCP/IP:
- *Presentation*: data format, encryption, compression
- *Session*: syncrhonization, checkpointing, recovery of data

Application Layer

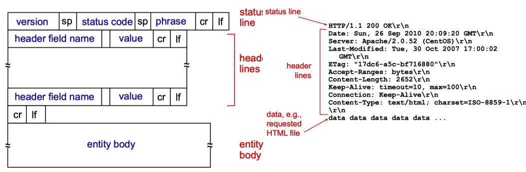
- Non-persistent HTTP:
  - Each object request → new TCP connection
  - **2 RTT** to fetch every object
- Persistent HTTP
  - Server leaves TCP connection open after init
  - Object sent over same TCP connection
  - Minimum of **1 RTT** to fetch all objects

HTTP Request



- cr: carriage return; lf: line feed
- Method types:
  - `HTTP/1.0`: GET, POST, HEAD
  - `HTTP/1.1`: GET, POST, HEAD, PUT (uploads file in entity body to path psecified in URL field) DELETE (deletes file specified in URL field)
    - `POST` method: data in request message body. High security.
    - `URL` (or `GET`) method: data in URL field. Low security.

HTTP Response



- Status code:
  - 200: OK, 301: Moved Permanently, 400: Bad Request, 404: Not Found, 505: HTTP Version Not Supported

Caching

- Total delay = Internet delay + access delay + LAN delay
- HTTP request: `If-modified-since: <date>`
- HTTP response: `304 Not Modified` or `200 OK`

Email

- Send: SMTP (Simple Mail Transfer Protocol)
- Recv: POP3 (Post Office Prot) / IMAP (Internet Mail Access Prot)
- User Agent → Mail Server A → Mail Server B → User Agent
- `CRLF.CRLF` to end message

POP3 (Post Officee Protocol): Deletes message, stateless
- Authorization phase:
  - Client command to server: `user <username>`, `pass <password>`
  - Server response: `+OK`, `-ERR`
- Transaction phase:
  - `list`: list message nuumbers
  - `retr <msg_number>`: retrieve message by number
  - `dele <msg_number>`: delete message by number
  - `quit`: end session

IMAP (Internet Mail Access Protocol): Keep state
- Keep all messages on server
- Allows user to organize messages in folders

DNS (Domain Name System)

- **Root Name Servers**: 13 root servers. return IP of TLD server
- **TLD Name Servers**: return IP of authoritative name server
  - Responsible for com, org, net, edu, ... uk, fr, ca, ...
- **Authoritative Name Servers**: return IP of host
  - Usually organization's own DNS server (e.g., Cloudflare)
- **Local DNS**: each ISP has one. not part of hierarchy.
  - If have cache, return (might outdate). Else, forward into hierarchy.

Resource Records (RR): format `(name, value, type, ttl)`
- **A**: hostname → IP. `name` = hostname, `value` = IP
- **NS**: name → auth DNS. `name` = domain, `value` = auth DNS hostname
- **MX**: mail server for domain. `name` = domain, `value` = server name
- **CNAME**: canonical name. `name` = alias, `value` = real name

Transport Layer

- **Multiplexing**: assign app data to different ports, add these port headers to outgoing packets. Let multiple apps to use same network.
- **Demultiplexing**: based on incoming packet port numbers, direct data to correct app.

Reliable Data Transfer (RDT)

- **RDT 1.0**: do nothing, assume no error
- **RDT 2.0**: solve packet corruption.
  - Adds checksum to detect error.
  - Receiver send ACK/NAK to acknowledge sender.
- **RDT 2.1**: solve ACK/NAK packet corruption
  - Important: assumes stop-and-wait protocol (1 packet at a time)
  - Sender Rule:
    - Send packet 0, wait for ACK
    - If `NAK` or `ACK/NAK` corrupt, resend packet 0
    - If `ACK`, send packet 1, wait for ACK
  - Receiver Rule:
    - If packet corrupt, send NAK
    - If packet 0 received, when prev is 1, send ACK, flip 0/1
    - If packet 1 received, when prev is 0, send ACK, flip 0/1
    - If packet 0 received two times, delete duplicate, send ACK again
- **RDT 2.2**: NAK-free protocol - Send `ACK 0/1` instead of `NAK`
  - Receiver ACK just the last correct packet if new packet corrupt
  - Sender retransmit packet if duplicate ACK received
- **RDT 3.0**: Add timeout to sender (Solves ACK packet loss)

Performance of RDT 3.0:
- Transmission Time = $D_{\text{trans}} = \frac{\text{Length}}{\text{Rate}} + \text{RTT}$
- Utilization (sender) = $U_{\text{sender}} = \frac{L/R}{\text{RTT} + L/R}$

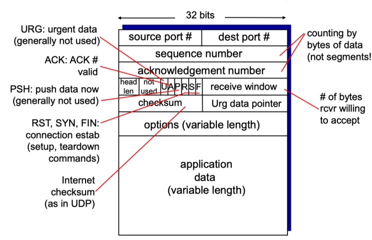
Pipelining: Utilization (sender): $U_{\text{sender}} = \frac{N \times L/R}{\text{RTT} + L/R}$
- where N is the number of packet transmitted at once (parallization)

Go-Back-N (GBN):
- Key Idea: ensure <u>in-order delivery</u>.
  - Receiver discard out-of-order pkt, only ACK packet $i-1$
  - **Cumulative ACK** sender get any ACK i, mark all pkts $\leq i$ ACK
- Sender:
  - Send N (window size) packets, wait for ACK
  - Receive any valid ACK i, move window to $[i+1, i+N]$
  - If 1st pkt in window timeout, resend entire window
- Receiver:
  - Only accept next in-order pkt (window size = 1, conceptually)
  - Only accept pkt i **if and only if** i is the next #seq
  - If got out-of-order pkt $j > i$, discard and send ACK $i-1$
  - If got in-order pkt i, send ACK i

Prevent Corruption: $N \leq K - 1$, where K is the #seq range

Selective Repeat (SR):
Prevent Corruption: $N \leq \lfloor K/2 \rfloor$, where K is the #seq range
- Sender:
  - Send N (window size) packets, wait for ACK
  - Track timeout of **every** single packet. Resend whichever timeouts
- Receiver:
  - ACK (not cumulative) every single valid pkt (even out-of-order)
  - Buffer out-of-order packets within its window (window size = N)
  - Send buffered pkt to application layer once prior pkts arrive.

TCP (Transmission Control Protocol)



- **Sequence Num**:
  - Byte number of 1st byte in the current segment
  - <u>NOT the same</u> as #seq in GBN and SR
  - Example: `SEQ=1000` data length=500 bytes (Sent bytes 1000~1499)
- **Acknowledgement Num**:
  - Sequence number of next byte expected from the other side
  - Example (Con't): `RECV ACK=1500` (i.e., next byte expected is 1500)
- 3-way-handshake w/ `SYN` and `ACK` flags
  - $A \to B$: `SYN, seq = x` A initiate connection
  - $B \to A$: `SYN-ACK, seq = y, ack = x+1` B acknowledge A's SYN
  - $A \to B$: `ACK, ack = y+1`. A acknowledge B's SYN
  - A and B maintain randomly initialized unique seq #

TCP Timeout Control

$$\text{EstimateRTT}_n = (1-\alpha) \cdot \text{EstimateRTT}_{n-1} + \alpha \cdot \text{SampleRTT}_n$$
$$\text{DevRTT}_n = (1-\beta) \cdot \text{DevRTT}_{n-1} + \beta \cdot |\text{SampleRTT}_n - \text{EstimateRTT}_n|$$
$$\text{TimeoutInterval}_n = \text{EstimateRTT}_n + 4 \cdot \text{DevRTT}_n$$

- where $\alpha = 0.125, \beta = 0.25$ typically, EMA
- Timeout = avg RTT + 4 * deviation (for safety margin)
  - too short: premature timenout, unnecessary retransmission
  - too long: slow reaction to segment loss

TCP Receiver Actions:
- Recv in-order seg, previous received all already ACK'ed
  - Action: Wait 500ms for new pkt, if no, send ACK of current pkt.
- Recv in-order seg, previous received pkt not yet ACK'ed
  - Action: immediately send **single** accumulative ACK (all previous)
- Recv out-of-order seg $j > i$ (Gap Detected)
  - Action: immediately send **duplicate** ACK $i-1$, with seq# = i
- Recv seg that fills **lower-end** gap caused in the above case.
  - Action: immediately send ACK.
- If sender recv 3 duplicate ACK of same data, resend un-ACK'ed segment with smallest seq # immediately.
- Receiver "advertise" free buffer space via **`rwnd`** value,
  - sender limit in-flight data in response to this value

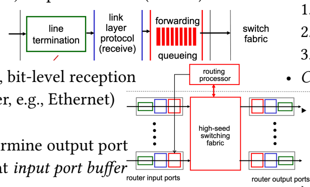
TCP Congestion Control
- Additive Increase, Multiplicative Decrease (AIMD):
  - Increase `cwnd` by 1 MSS (Maximum Segment Size) **every RTT**.
  - Cut `cwnd` by half when loss detected.
  - rate ≈ cwnd ÷ RTT (bytes/sec)
- *TCP Reno*:
  - timeout loss: `cwnd = 1 MSS`, exp grow to `ssthresh`, then lin grow

- 3 duplicate ACK loss: `ssthresh` and `cwnd` $\div 2$, then linearly grow
- *TCP Tahoe*: always set `cwnd` to 1 MSS when loss detected
- **avg TCP throughput**: $\frac{3}{4} \cdot \frac{\text{Window Size}}{\text{RTT}}$ bytes/sec

Network Layer

- ***forwarding (data plane)***: move packets from a router's input to output
- ***routing (control plane)***: determine route from source to destination
  - *Traditional routing algorithms*: implemented in routers (local decision)
  - *Software-defined networking (SDN)*: implemented in (remote) servers

Router



Input Port (3 components):

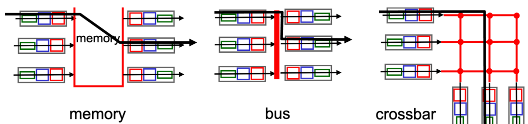
1. Line termination (physical layer), bit-level reception
2. link layer protocol (data link layer, e.g., Ethernet)
3. Forwarding:
   - Use header field values $\rightarrow$ determine output port
   - If datagram arrive fast, queue at *input port buffer*

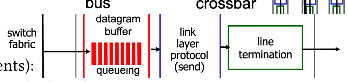
Input Port Queuing

- *Head-of-the-line (HOL) blocking*: in same input port, packet infront of line blocking packets behind (e.g., when head packet's output port is busy)

Switching Fabric (3 types of methods):

1. *Memory*: CPU direct control. packet copied to RAM, speed bottlenecked.
2. *Bus*: datagram move thru. shared bus. Limited by bus bandwidth.
3. *Crossbar*: high parallelism, n^2 crosspoints.

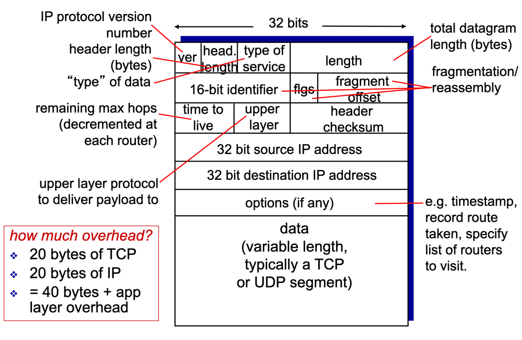




Output Port (3 components):

1. Datagram buffer. Required when datagram arrives > transmission rate
2. Link layer protocol (send) 3. Line termination

Internet Protocol (IP)



IPv4 Address: 32-bits, 2 levels, 5 classes (A, B, C, D, E)

- A: begin w/ $(0)_2$ 0-127, 8/24bit, $\frac{1}{2}$ addr space, mask: 255.0.0.0
- B: begin w/ $(10)_2$ 128-191, 16/16bit, $\frac{1}{4}$ addr space, mask: 255.255.0.0
- C: begin w/ $(110)_2$ 192-223, 24/8bit, $\frac{1}{8}$ addr space, mask: 255.255.255.0
- D: begin w/ $(1110)_2$ 224-239, multicast, $\frac{1}{16}$ addr space
- E: begin w/ $(1111)_2$ 240-255, reserved (future proof), $\frac{1}{16}$ addr space

Subnets: devices / interfaces with same netid (network part of IP address), physically reach each other without router.

Classless Inter-Domain Routing (CIDR):

- subnet part (netid) is arbitrary length
- format: **`a.b.c.d/x`**, where **`x`** is the # bits in subnet portion of address

IP Fragmentation:

Purpose:

- Network links have ***MTU*** (*Max Transmission Unit*)
- Need to fragment datagrams, then reassemble at destination.

Header:

- `length` field: total length of datagram (header + data)
- `offset` field: data length $\div$ 8
- `fragflag`: 0 for last fragment, 1 for not last

Destination-based Forwarding:

- Longest prefix matching: forward table contains list of prefixes, find longest prefix match

DHCP (Dynamic Host Configuration Protocol):

<u>*Purpose*: let host dynamically get IP addr. from network server when join</u>

- Host: boradcast **DHCP discover** msg to find DHCP server
- DHCP server: respond **DHCP offer** msg
- Host: requests IP address via **DHCP request** msg
- DHCP server: send **DHCP ack** msg, which returns:
  - allocatd IP, first-hop router addr, network mask, name and IP of DNS

ICANN (Internet Corporation for Assigned Names and Numbers):

- Assign blocks of IP addresses to the RIRs (Regional Internet Registries); RIRs assign to ISPs or large organizations

Routing Protocols

Bellman-Ford Algorithm:

- Same as Dijkstra's algorithm: $d(x) = \min(d(x), d(y) + l(y, x))$
- Except, don't choose min node each iteration.
- **for** $n - 1$ iterations (n = number of nodes),
  - **for** each m edge (y, x):
    - $d(x) = \min(d(x), d(y) + l(y, x))$ **if** $y \rightarrow x$
    - $d(y) = \min(d(y), d(x) + l(x, y))$ **if** $x \rightarrow y$

Distance Vector Algorithm:

- Same as Bellman-Ford, except asynchronous, distributed, iterative
- Maintains a 多源最短路矩阵
- Each node:
  1. **wait** for change in local link cost, or msg from neighbor
  2. **re-compute** distance vector estimate using Bellman-Ford
  3. if DV to **any** dest changed, notify **all** neighbors
- *Characteristics*:
  - **Good news travel fast** (adj edge cost $\downarrow$): node cost unchange / improve. Advertise to all neighbors. Update ripple out one hop at a time (fast).
  - **Bad news travel slow** (adj edge cost $\uparrow$): $\boldsymbol{A}$ drop routes that use the bad edge, seek alternative from neighbor $\boldsymbol{B}$. But neighbor $\boldsymbol{B}$ might route through $\boldsymbol{A}$, its shorter only because it is oudated. End up "bouncing" distance updates back and forth (slow, *count-to-infinity problem*).
    - **Solution**: if $\boldsymbol{B}$ routes through $\boldsymbol{A}$, set $\boldsymbol{B}$'s distance to ∞.

Link-State Routing (Dijkstra) **vs. Distance Vector** (Bellman-Ford)

	LS (Dijkstra)	DV (Bellman-Ford)
Message Complexity	$O(nm)$	exchange msg between adj nodes only
Convergence Speed	$O(n^2)$	vary. loops / count-to-infinity

AS: Autonomous System, group of routers under same admin control

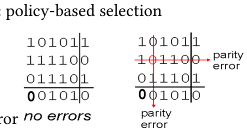
Intra-AS Routing:

- *Interior Gateway Protocols* (IGPs):
  - *RIP (Routing Information Protocol)*:
    - distance metric: <u>number of hops</u> (max 15 hops)
    - update interval: DV advertise every 30 seconds
    - routing table managed by application-level process route-d (daemon)
- *Open Shortest Path First* (OSPF):
  - LS-based, use Dijkstra's algorithm
  - OSPF message directly over IP (**not** TCP/UDP). Use `protocol` field 89
  - All OSPF messages authenticated; Multiple same-cost paths allowed
  - Each link, multiple cost metric for different services
  - Supports multicast (MOSPF) and unicast
  - **Hierarchical OSPF** in large domains

Inter-AS Routing:

- Forwarding table configured by both intra- and inter-AS routing algorithm
- *Hot Potato Routing*:
  - When destination reachable via multiple AS, greedily choose gateway w/ least cost. Don't care about cost of inter-AS routes.
- *BGP* (Border Gateway Protocol):
  - Types:
    - *eBGP*: obtain subnet reachability from neighboring AS
    - *iBGP*: propagate reachability to AS-internal routers
  - BGP Messages:
    - `OPEN`: open semi-persistent TCP connection to remote BGP peers
    - `UPDATE`: advertises new paths (or withdraw old)
    - `KEEPALIVE`: keep connection alive in absence of `UPDATE`, and ack `OPEN`
    - `NOTIFICATION`: report errors in previous msg; also for closing conn.
  - BGP Route Selection:
    1. Shortest AS-path
    2. Closest next-hop router
    3. local preference value attribute: policy-based selection
    4. Aadditional criterias

Link Layer



Parity Checking:

- *Single Bit Parity*: **detect** single bit error
- *Two-Dimensional Parity*: **detect** and **correct** single bit error

Cyclic Redundancy Check (CRC):

1. Choose divisor G (**must** be $r + 1$ bits long)
2. Left shift data bits D by r bits $\Rightarrow D \cdot 2^r$
3. Obtain remainder: $R = (D \cdot 2^r) \bmod G$ (r bits)
4. Form final code: concatenate D with R (equivalently, $D \cdot 2^r$ xor R)

$$(D \cdot 2^r \text{ xor } R) \bmod G = 0$$
$$D \cdot 2^r \text{ xor } R = nG$$
$$D \cdot 2^r = nG \text{ xor } R$$
$$R = (D \cdot 2^r) \bmod G$$

- **IMPORTANT**: both addition in "nG" and subtraction in "$\div G$" is **XOR**
- ***Intuition***:
  - Normally: $\mathbf{A} * \mathbf{B} = (A_0 \cdot 2^0 \cdot \mathbf{B}) + (A_1 \cdot 2^1 \cdot \mathbf{B}) + \ldots + (A_n \cdot 2^n \cdot \mathbf{B})$
  - But in CRC: $\mathbf{A} * \mathbf{B} = (A_0 \cdot 2^0 \cdot \mathbf{B})$ xor ... xor $(A_n \cdot 2^n \cdot \mathbf{B})$
  - Normally in long division, subtract each $(A_i \cdot 2^i \cdot \mathbf{B})$ from dividend
  - But in CRC, we xor each $(A_i \cdot 2^i \cdot \mathbf{B})$ from dividend
  - Intuitively, both $+$ and $-$ replaced with **xor**.
  - Crucially, this works because <u>**xor is its own inverse**</u>

Multiple Access Protocols:

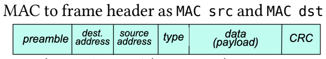
- Idealy goals: **1)** When 1 node, send at rate R (channel max rate) **2)** When M nodes, each send at average rate R/M **3)** Fully decentralized: no central controller / clock for synchronization **4)** Simple
- **MAC (medium access control) protocols**:
  1. ***Channel Partitioning MAC protocols***:
     1. TDMA (Time Division Multiple Access):
        - Channel time divided into fixed length slots
        - Each station gets a slot; unused slots go idle
     2. FDMA (Frequency Division Multiple Access):
        - Channel spectrum divided into frequency bands
        - Each station assign a fixed frequency. Unused freq bands go idle
  2. ***Random Access MAC protocols***:
     1. Slotted ALOHA:
        - Each node: attempt to transmit as soon as possible
        - When no collision: transmit at full channel rate
        - When collision: no node can send (slot wasted)
          - each node retry in subsequent slot with prob. p until success
        - *Assumptions*: all frame same size; time divided into equal-sized slots; nodes are synchronized, send only at time slot boundary;
        - **Pros**: single node full rate; highly decentralized; simple
        - **Cons**: collision waste slots; clock synchronization needed
        - **Low efficiency**: $\lim_{N \to \infty} Np(1-p)^{N-1} = \frac{1}{e} \approx 37\%$, for N nodes each transmit in a slot with probability p
     2. Pure (unslotted) ALOHA:
        - Each node send ASAP (no slot, no boundary, thus no sync needed)
        - Collision when ≥ 2 nodes overlap. **Worse efficiency**:
          - $\lim_{N \to \infty} Np(1-p)^{2(N-1)} = \frac{1}{2}e \approx 18\%$
     3. CSMA (Carrier Sense Multiple Access):
        - Listen before transmit. Send if idle; defer if busy.
        - Collision still happen (two nodes think channel is idle)
     4. CSMA/CD (Carrier Sense Multiple Access with Collision Detection):
        - Added collision detection. Better performance than ALOHA
  3. ***"Taking turns" MAC protocols***:
     1. Polling: master node polls one by one from "slave" devices
        - Polling overhead, latency, single point of failure
     2. Token ring: token passed around ring. Only token-holder can send
        - Token overhead, latency, single point of failure (token)

Local Area Networks (LANs)

MAC Address (aka LAN Address)

- used "locally" to send frame between interfaces (same network)
- 48 bit address, burned in NIC ROM, hex (e.g., `1A-2F-BB-76-09-AD`)
- Alloc. administered by IEEE, manufacturer obtain portion of addr space
- **ARP** (addr. resolution protocol): ip$\rightarrow$mac; **RARP** (reverse ARP): mac$\rightarrow$ip
  - each node maintain a ARP cache (ARP table: `<IP, MAC>`), with TTL
  - A *boradcast* ARP requesst containing B's IP address
  - B receives ARP packet and *unicast* reply packet to A
- **Addressing**:
  1. datagram created with IP src/dst, unchanged throughout transmission
  2. At each node: lookup next-hop IP $\rightarrow$ lookup MAC in ARP table$\rightarrow$ add self MAC and next-hop MAC to frame header as `MAC src` and `MAC dst`

Ethernet



- `preamble`: **7 bytes** of $(10101010)_2$ + **1 byte** of $(10101011)_2$
  - $(10101010)_2$ creates fixed wave pattern, let receiver sync w/ sender freq.
  - $(10101011)_2$ known as *Start Frame Delimiter* (SFD), tells receiver to start
- `dest address` and `src address`: **6 bytes** each (contains MAC address)
  - If interface receives a frame w/ `dest addr` not matching its own, discard
- `type`: **2 bytes** upper layer protocol (e.g., mostly IP)
- `data` (payload): **46-1500 bytes**
- `CRC`: **4 bytes**. If error detected, frame dropped

Ethernet Characteristics:

- Connectionless & unreliable: no handshake, no ack. It's upper layer's job
- Uses **Unslotted CSMA/CD w/ binary backoff**
  1. NIC receives datagram from network layer $\Rightarrow$ create frame
  2. If sense channel idle, transmit frame. Else, wait until idle
  3. If transmit w/o collision, done; Otherwise, abort.
  4. After abort, ***exponential backoff***: for m-th collision, wait $K \times 512$ bit time (K sample from $\{0, 1, \ldots, 2^{m-1}\}$). Then return to **step 2** above

Switches

1. *Hub*: bits come in on one port $\Rightarrow$ broadcast to **all** other ports at same rate
   - All connecting nodes can collide w/ one another
   - No frame buffering; no CSMA/CD at hub: host NICs detect collision
2. *Switch*: examine incoming frame's `dest address` $\Rightarrow$ selective forward
   - Buffer incoming frames at switch. CSMA/CD before forwarding.
   - Transparent (invisible to host); Plug-and-play (no config needed)
   - **Switch table**:
     - `<MAC, Port, TTL>`: records which port reach the host with the MAC
     - Learning: when recv frame, record sender location in switch table

Switch Update & Forwarding

Assume: frame F received at switch S from port p
Given: switch table at S.table
src, dst $\leftarrow$ unpack(F) // src and dst MAC addr in F
S.table[src] $= p$
if dst **in** S.swith_table **then**:
  if S.table[dst] == S.table[src] **then** drop F
  else forward F to interface at S.table[dst]
else flood // forward F to all interface, except arriving interface