ICMP: Internet Control Message Protocol

- used by hosts and routers to communicate network-level information
  - error reporting: unreachable host, network, port, protocol
  - echo request/reply (used by ping)
- network-layer "above" IP:
  - ICMP messages carried in IP datagrams

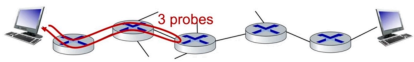
We will send n probes and after receiving the response (reach destination or not), we can send n probes with TTL +1

This will show what path the message took, to get to dest.

It's network layer (IP), meaning if the source ip is blocked, we can't complete the traceroute.

Traceroute and ICMP

UDP segments



- source sends sets of UDP segments to destination
  - 1st set has TTL =1, 2nd set has TTL=2, etc.
- datagram in *n*th set arrives to nth router:
  - router discards datagram and sends source ICMP message (type 11, code 0)
  - ICMP message possibly includes name of router & IP address
- when ICMP message arrives at source: record RTTs

* ICMP also gets loaded to the payload of a datagram like TCP or UDP, but its not Transport layer protocol

Link layer: services

- framing, link access:
  - encapsulate datagram into frame, adding header, trailer
  - channel access if shared medium
  - "MAC" addresses in frame headers identify source, destination (different from IP address!)
- reliable delivery between adjacent nodes
  - we already know how to do this!
  - seldom used on low bit-error links
  - wireless links: high error rates
    - Q: why both link-level and end-end reliability?

We usually want to check for errors locally, as well as end-end because → it better to detect errors right after they occur, instead of sending the frame to the destination and then finding that out.

part of the link layer is in the software, and parts of it is in hardware.

Link layer: services (mo

- flow control:
  - pacing between adjacent sending and receiving nodes
- error detection:
  - errors caused by signal attenuation, noise.
  - receiver detects errors, signals retransmission, or drops frame
- error correction:
  - receiver identifies *and corrects* bit error(s) without retransmission
- half-duplex and full-duplex:
  - with half duplex, nodes at both ends of link can transmit, but not at same time

Parity checking

single bit parity:

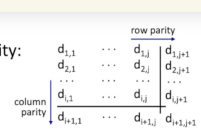
- detect single bit errors

Even parity: set parity bit so there is an even number of 1's

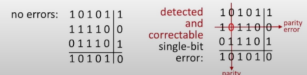
At receiver:

- compute parity of *d+1* received bits, if not even, then error detected
- can detect odd number of bit flips

Two-D parity:

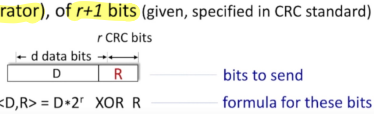


- detect two-bit errors
- detect *and correct* single bit errors without retransmission!



Cyclic Redundancy Check (CRC)

- more powerful error-detection coding
- D: data bits (given, think of these as a binary number)
- G: bit pattern (generator), of *r+1* bits (given, specified in CRC standard)



sender: compute *r* CRC bits, R, such that <D,R> *exactly* divisible by G (mod 2)

- receiver knows G, divides <D,R> by G. If non-zero remainder: error detected!
- can detect all burst errors less than r+1 bits
- widely used in practice (Ethernet, 802.11 WiFi)

Multiple access links, protocols

two types of "links":

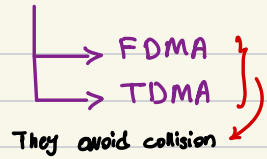
- point-to-point
  - point-to-point link between Ethernet switch, host
  - PPP for dial-up access
- broadcast (shared wire or medium)
  - old-school Ethernet
  - upstream HFC in cable-based access network
  - 802.11 wireless LAN, 4G/4G. satellite

MAC protocols: taxonomy

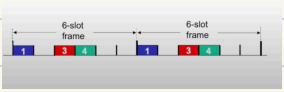
three broad classes:

- channel partitioning
  - divide channel into smaller "pieces" (time slots, frequency, code)
  - allocate piece to node for exclusive use
- random access
  - channel not divided, allow collisions
  - "recover" from collisions
- "taking turns"
  - nodes take turns, but nodes with more to send can take longer turns

Channel Partitioning

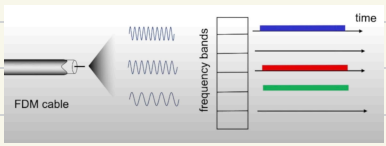


They avoid collision



Random Access Protocols: Allow channels to transmit data

if two or more transmit at the same time, collision happens.



Slotted ALOHA

the setting:

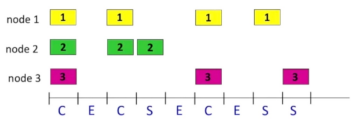
- all frames same size
- time divided into equal size slots (time to transmit 1 frame)
- nodes are synchronized
- nodes begin transmissions (if any) at slot start times
- if 2 or more nodes transmit in slot, collision detected by senders

operation:

- when node has new frame to send, transmit in next slot
  - *if no collision:* success!
  - *if collision:* node retransmits frame in each subsequent slot with probability p until success

randomization – *why?*

Slotted ALOHA



C: collision
S: success
E: empty

Pros:

- single active node can continuously transmit at full rate of channel
- highly decentralized: only slots in nodes need to be in sync
- simple

Cons:

- synchronization
- collisions, "wasting" slots
- idle slots, "wasting" slots

CSMA (carrier sense multiple access)

simple CSMA: listen before transmit:

- if channel sensed idle: transmit entire frame
- if channel sensed busy: defer transmission

- human analogy: don't interrupt others!

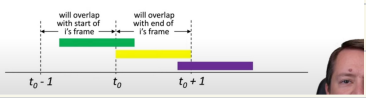
CSMA/CD: CSMA with *collision detection*

- collisions *detected* within short time
- colliding transmissions aborted, reducing channel wastage
- collision detection easy in wired, difficult with wireless

Listening before speaking

Pure ALOHA

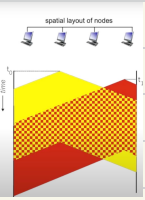
- unslotted Aloha: simpler, no synchronization
  - when frame first arrives: transmit immediately
- collision probability increases with no synchronization:
  - frame sent at t_0 collides with other frames sent in $[t_0-1, t_0+1]$



less efficient

CSMA: collisions

- collisions can *still* occur with carrier sensing:
  - propagation delay means two nodes may not hear each other's just-started transmission
- collision: entire packet transmission time wasted
  - distance & propagation delay play role in in determining collision probability



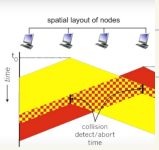
Ethernet CSMA/CD algorithm

1. Ethernet receives datagram from network layer, creates frame
2. If Ethernet senses channel:
   if idle: start frame transmission.
   if busy: wait until channel idle, then transmit
3. If entire frame transmitted without collision - done!
4. If another transmission detected while sending: abort, send jam signal
5. After aborting, enter *binary (exponential) backoff:*
   - after mth collision, chooses K at random from $\{0,1,2, \ldots, 2^m-1\}$. NIC waits $K \cdot 512$ bit times, returns to Step 2
   - more collisions: longer backoff interval

as collisions increase, the backoff time also increases.

CSMA/CD:

- CSMA/CD reduces the amount of time wasted in collisions
  - Collision Detection (CD): transmission aborted on collision detection



"Taking turns" MAC protocols

channel partitioning MAC protocols:

- share channel *efficiently* and *fairly* at high load
- inefficient at low load: delay in channel access, 1/N bandwidth allocated even if only 1 active node!

random access MAC protocols

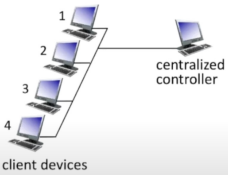
- efficient at low load: single node can fully utilize channel
- high load: collision overhead

"taking turns" protocols: try to get best of both worlds

- channel allocated explicitly (no collisions)
- nodes won't hold channel for long if nothing to send
- two approaches: polling, token passing

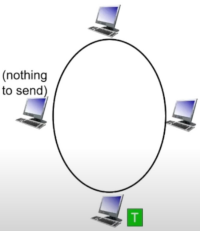
polling:

- centralized controller uses polling messages to "invite" client nodes to transmit in turn
- protocol needed for client devices to join/leave network
- concerns:
  - polling overhead
  - access latency (wait for turn)
  - single point of failure
- Bluetooth uses polling



token passing:

- control *token* message explicitly passed from one node to next, sequentially
  - transmit while holding token
- concerns: similar to polling
  - join/leave protocol needed
  - control overhead
  - access latency
  - single point of failure (token)



↳: if the token gets lost, they cannot communicate. Only the holder of the token can transmit

Summary of MAC protocols

- 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
- taking turns
  - polling from central site, token passing
  - Bluetooth, FDDI, token ring

MAC Address: (or LAN or physical or Ethernet address) is used "locally" to get frame from one interface to another physically-connected interface (same subnet, in IP address)

→ 48bit MAC address burned into ROM (hard to change) → written in pairs of hexadecimal digits

IP → used for layer 3 (network layer) forwarding

→ 32 bit address, network-layer address for interface

Each interface on LAN has → locally unique 32-bit IP address

→ unique 48-bit MAC address → we can't do prefix matching since the number is not related to the network

* MAC address assigned by manufacturer

How to find the destination hosts' MAC addr.?

ARP protocol → similar to how DNS works for IP

ARP protocol in action

example: A wants to send datagram to B

- B's MAC address not in A's ARP table, so A uses ARP to find B's MAC address

A broadcasts ARP query, containing B's IP addr

1. • destination MAC address = FF-FF-FF-FF-FF-FF
   • all nodes on LAN receive ARP query

ARP table in A

IP addr	MAC addr	TTL

Ethernet frame (sent to FF-FF-FF-FF-FF-F

Source MAC: 71-65-F7-2B-08-53
Source IP: 137.196.7.23
Target IP address: 137.196.7.14
...

A
71-65-F7-2B-08-53
137.196.7.23

C

B
58-23-D7-FA-20-B0
137.196.7.14

A → broadcasts the target IP

B → responds with its MAC address

Why do we need MAC addr?

MAC address is built on the link layer which does not have access to IP(network)

↳ only valid within a Local Network (LAN)

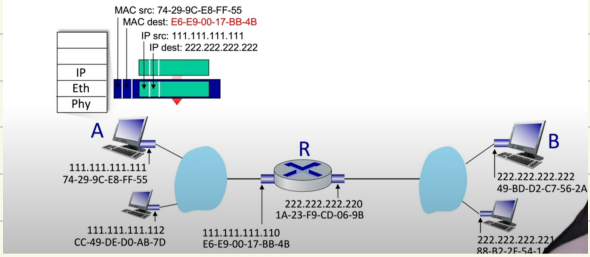
Routing to another subnet: addressing

walkthrough: sending a datagram from *A* to *B* via *R*

- focus on addressing – at IP (datagram) and MAC layer (frame) levels
- assume that:
  - A knows B's IP address
  - A knows IP address of first hop router, R (how?)
  - A knows R's MAC address (how?)

Since A and B are not in the same local area network, A creates a frame with dest MAC set to R. When the frame arrives at R, the frame is removed and a new frame is used for the route from R to B

- A creates IP datagram with IP source A, destination B
- A creates link-layer frame containing A-to-B IP datagram
  - R's MAC address is frame's destination



- R determines outgoing interface, passes datagram with IP source A, destination B to link layer
- R creates link-layer frame containing A-to-B IP datagram. Frame destination address: B's MAC address

Ethernet: physical topology

- bus: popular through mid 90s
  - all nodes in same collision domain (can collide with each other)
- switched: prevails today
  - active link-layer 2 *switch* in center
  - each "spoke" runs a (separate) Ethernet protocol (nodes do not collide with each other)

Ethernet: unreliable, connectionless

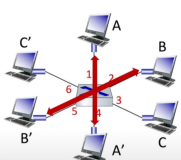
- connectionless: no handshaking between sending and receiving NICs
- unreliable: receiving NIC doesn't send ACKs or NAKs to sending NIC
  - data in dropped frames recovered only if initial sender uses higher layer rdt (e.g., TCP), otherwise dropped data lost
- Ethernet's MAC protocol: unslotted CSMA/CD with binary backoff

Ethernet switch

- Switch is a link-layer device: takes an *active* role
  - store, forward Ethernet frames
  - examine incoming frame's MAC address, *selectively* forward frame to one-or-more outgoing links when frame is to be forwarded on segment, uses CSMA/CD to access segment
- transparent: hosts *unaware* of presence of switches
- plug-and-play, self-learning
  - switches do not need to be configured

Switch: multiple simultaneous transmissions

- hosts have dedicated, direct connection to switch
- switches buffer packets
- Ethernet protocol used on *each* incoming link, so:
  - no collisions; full duplex
  - each link is its own collision domain
- switching: A-to-A' and B-to-B' can transmit simultaneously, without collisions

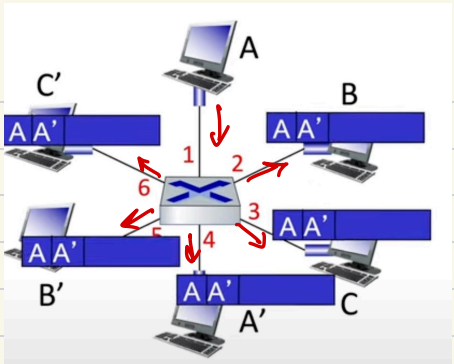


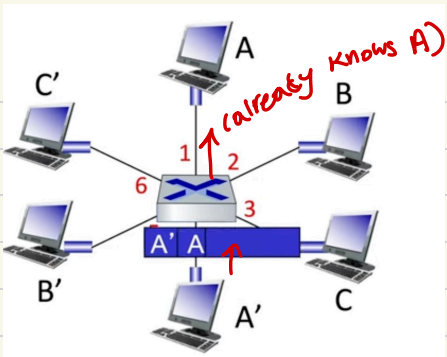
switch with six interfaces (1,2,3,4,5,6)

↳ two host transmitting to one host will cause collision and thus, can't happen simultaneously

How does a switch know which interface corresponds to which? each switch has a switch table (similar to a routing table) → with MAC, corresponding interface

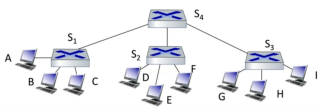
How is the table created? no routing protocol → switch learns which host can be reached using which interface. when a packet is sent to the switch, it will learn the sender's interface. However, if it doesn't know which interface is the receiver, it will floods the interfaces → we don't have a lot of interfaces (small compared to layer 3)





Interconnecting switches

self-learning switches can be connected together:



Q: sending from A to G - how does S_1 know to forward frame destined to G via S_4 and S_3?

- A: self learning! (works exactly the same as in single-switch case!)

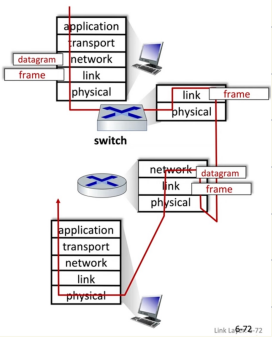
Switches vs. routers

both are store-and-forward:

- *routers*: network-layer devices (examine network-layer headers)
- *switches*: link-layer devices (examine link-layer headers)

both have forwarding tables:

- *routers:* compute tables using routing algorithms, IP addresses
- *switches:* learn forwarding table using flooding, learning, MAC addresses

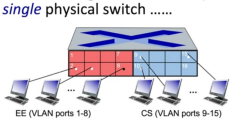


Port-based VLANs

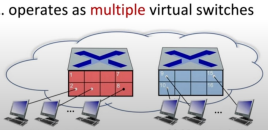
Virtual Local Area Network (VLAN)

switch(es) supporting VLAN capabilities can be configured to define multiple *virtual* LANS over single physical LAN infrastructure.

port-based VLAN: switch ports grouped (by switch management software) so that *single* physical switch .....

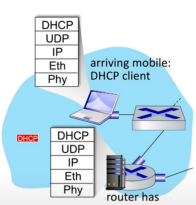


... operates as multiple virtual switches



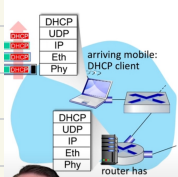
The broadcasting aspect is not efficient for scaling thus we would like to make big LANs into smaller subLAN → VLANs

A day in the life: connecting to the Internet

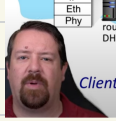


- connecting laptop needs to get its own IP address, addr of first-hop router, addr of DNS server: use DHCP
- DHCP request encapsulated in UDP, encapsulated in IP, encapsulated in 802.3 Ethernet
- Ethernet frame broadcast (dest: FFFFFFFFFFFF) on LAN, received at router running DHCP server
- Ethernet demuxed to IP demuxed, UDP demuxed to DHCP

A day in the life: connecting to the Internet

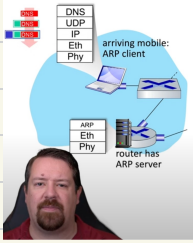


- DHCP server formulates DHCP ACK containing client's IP address, IP address of first-hop router for client, name & IP address of DNS server
- encapsulation at DHCP server, frame forwarded (switch learning) through LAN, demultiplexing at client
- DHCP client receives DHCP ACK reply



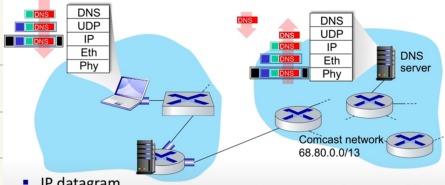
Client now has IP address, knows name & addr of DNS server, IP address of its first-hop router

A day in the life... ARP (before DNS, before HTTP)



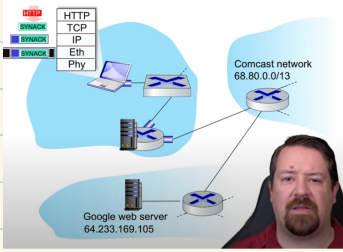
- before sending HTTP request, need IP address of www.google.com: DNS
- DNS query created, encapsulated in UDP, encapsulated in IP, encapsulated in Eth. To send frame to router, need MAC address of router interface: ARP
- ARP query broadcast, received by router, which replies with ARP reply giving MAC address of router interface
- client now knows MAC address of first hop router, so can now send frame containing DNS query

A day in the life... using DNS



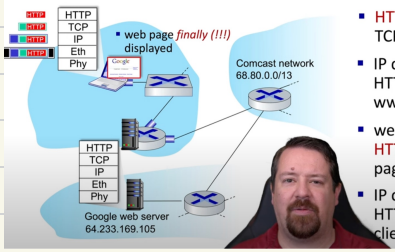
- demuxed to DNS
- DNS replies to client with IP address of www.google.com
- IP datagram containing DNS query forwarded via LAN switch from client to 1st hop router
- IP datagram forwarded from campus network into Comcast network, routed (tables created by RIP, OSPF, IS-IS and/or BGP routing protocols) to DNS server

A day in the life...TCP connection carrying HTTP



- to send HTTP request, client first opens TCP socket to web server
- TCP SYN segment (step 1 in TCP 3-way handshake) inter-domain routed to web server
- web server responds with TCP SYNACK (step 2 in TCP 3-way handshake)
- TCP connection established!

A day in the life... HTTP request/reply



- HTTP request sent into TCP socket
- IP datagram containing HTTP request routed to www.google.com
- web server responds with HTTP reply (containing web page)
- IP datagram containing HTTP reply routed back to client

Wireless networks → Infrastructure mode: Base station connects mobiles into wired network
→ Ad hoc networks: no base network → mobile changes base station (hand off)
nodes only transmit to other nodes within link coverage
nodes organize themselves in the network

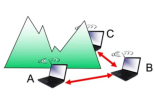
Wireless network taxonomy

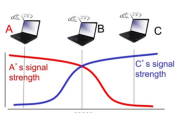
	single hop	multiple hops
infrastructure (e.g., APs)	host connects to base station (WiFi, cellular) which connects to larger Internet	host may have to relay through several wireless nodes to connect to larger Internet: *mesh net*
no infrastructure	no base station, no connection to larger Internet (Bluetooth, ad hoc nets)	no base station, no connection to larger Internet. May have to relay to reach other a given wireless node MANET, VANET

WiFi repeaters

Wireless link characteristics (3)

Multiple wireless senders, receivers create additional problems (beyond multiple access):





Hidden terminal problem
- B, A hear each other
- B, C hear each other
- A, C can not hear each other means A, C unaware of their interference at B

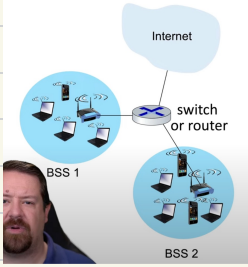
Signal attenuation:
- B, A hear each other
- B, C hear each other
- A, C can not hear each other interfering at B

Code Division Multiple Access (CDMA)

- unique "code" assigned to each user; i.e., code set partitioning
  - all users share same frequency, but each user has own "chipping" sequence (i.e., code) to encode data
  - allows multiple users to "coexist" and transmit simultaneously with minimal interference (if codes are "orthogonal")
- encoding: inner product: (original data) X (chipping sequence)
- decoding: summed inner-product: (encoded data) X (chipping sequence)

CDMA: two-sender interference

802.11 LAN architecture



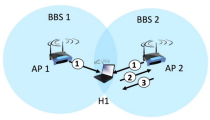
- wireless host communicates with base station
  - base station = access point (AP)
- Basic Service Set (BSS) (aka "cell") in infrastructure mode contains:
  - wireless hosts
  - access point (AP): base station
  - ad hoc mode: hosts only

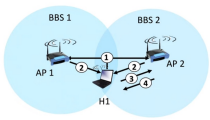
802.11: Channels, association

- spectrum divided into channels at different frequencies
  - AP admin chooses frequency for AP
  - interference possible: channel can be same as that chosen by neighboring AP!
- arriving host: must associate with an AP
  - scans channels, listening for *beacon frames* containing AP's name (SSID) and MAC address
  - selects AP to associate with
  - then may perform authentication [Chapter 8]

if two networks have the same SSID, the host will distinguish using their unique MAC addr

802.11: passive/active scanning



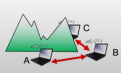


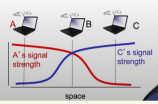
passive scanning:
(1) beacon frames sent from APs
(2) association Request frame sent: H1 to selected AP
(3) association Response frame sent from selected AP to H1

active scanning:
(1) Probe Request frame broadcast from H1
(2) Probe Response frames sent from APs
(3) Association Request frame sent: H1 to selected AP
(4) Association Response frame sent from selected AP to H1

IEEE 802.11: multiple access

- avoid collisions: 2+ nodes transmitting at same time
- 802.11: CSMA - sense before transmitting
  - don't collide with detected ongoing transmission by another node
- 802.11: *no* collision detection!
  - difficult to sense collisions: high transmitting signal, weak received signal due to fading
  - can't sense all collisions in any case: hidden terminal, fading
  - goal: *avoid collisions:* CSMA/CollisionAvoidance





in case of on terminal, we might have collisions

Avoiding collisions (more)

idea: sender "reserves" channel use for data frames using small reservation packets

- sender first transmits *small* request-to-send (RTS) packet to BS using CSMA
  - RTSs may still collide with each other (but they're short)
- BS broadcasts clear-to-send CTS in response to RTS
- CTS heard by all nodes
  - sender transmits data frame
  - other stations defer transmissions

Collision Avoidance: RTS-CTS exchange