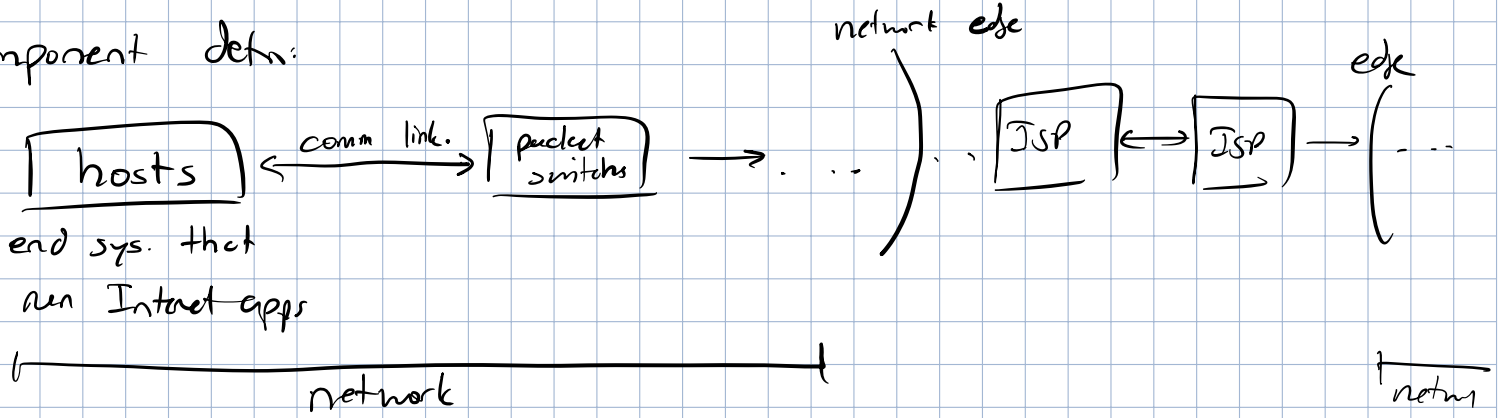
INTRODUCTION

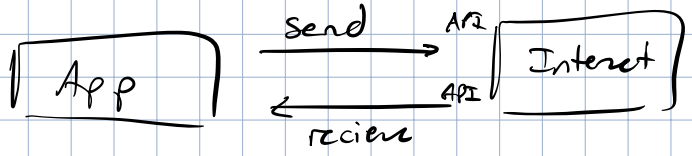
Definition of Internet

Component defn:



Internet: network of networks.

Service-based defn:



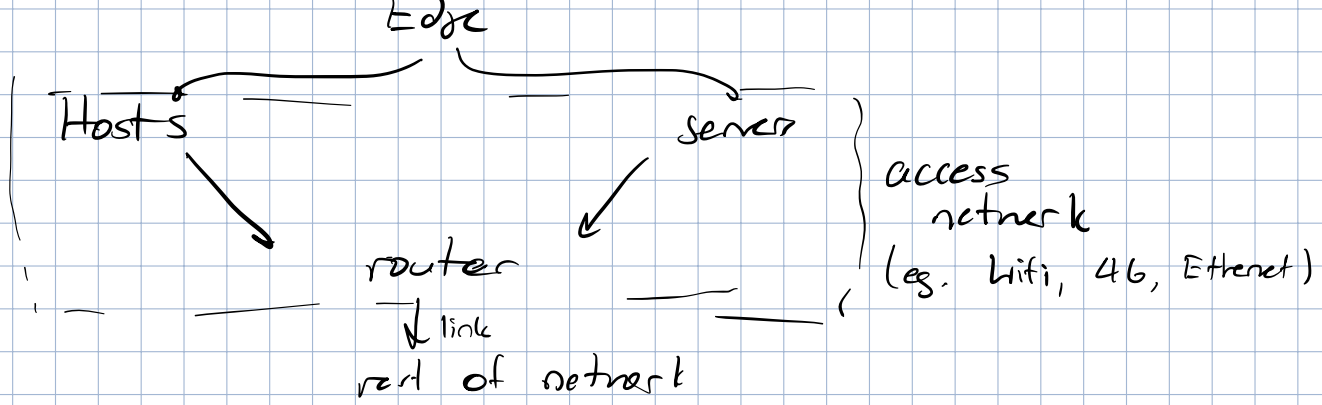
Protocols

Comm. standards that controls sending & recieving messages.

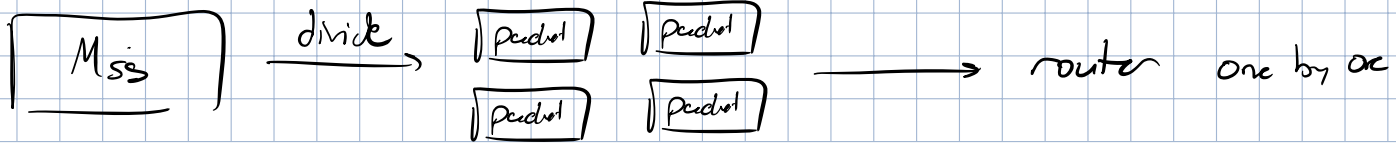
Defined by software

Reason: compatibility

Network Edge



Q: how do we send messages to router



Assume each packet is L bits, bandwidth of router is R bits/sec.
↳ Time to send packet: L/R sec.

Links:

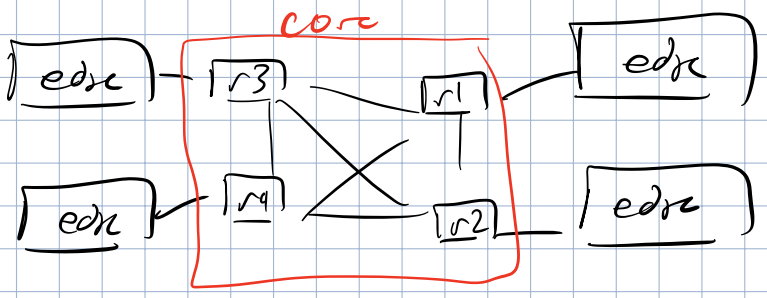
① Guided: transmission in solid media
- Ethernet
- Coaxial: 2 Cu wires → bidirectional recv.
- Fibre optic: 1 pulse → 1 bit. Low error, high bandwidth

② Unguided: propagate freely
- Radio
- Microwave

Cons:
1. Environment can have effect
2. Half-duplex: can only send & recieve for particular spectrum 1 at time.

Network Core

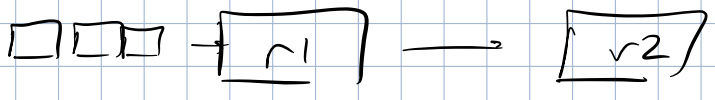
Defn: interconnected routers:



Role of core:
1. Figure out best route from src to dest
2. Forward packets to dest.

2 methods of forwarding.

① Store-and-forward packet switching



entire packet must arrive at r1 b/f moving to r2

Q: How long to move 1 packet from src → dst w/ n

packet switchs w/ same transmission rate:

src •——·——·——·——• dest

# of links = n + 1
transmiss / link = L/R
} total time = (n+1) · L/R

Q: time delta btw packages?

L/R ← wait for package to be uploaded

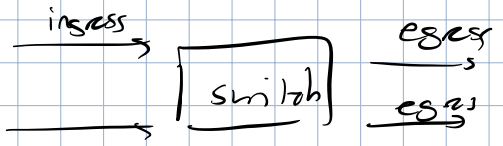
Q: Total time to send p packets?

total time = time to send 1 packet + time to rec (p-1) pack

$$= (n+1) \cdot L/R + \sum_{p-1} L/R$$

$$= \boxed{L/R\ (p+n)}$$

Con: queing

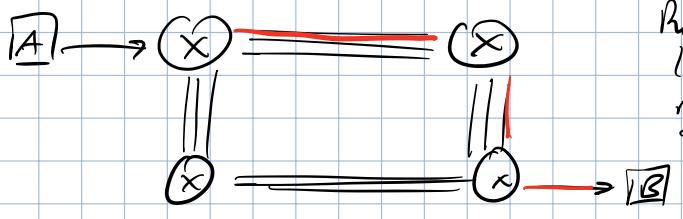


ingress rate > egress rate for some time → queing

Implications:

1. Packet loss ← internal mem. can't store
2. Latency ← switch needs to prioritize packets

② Circuit switching:



Reserve resources just for 1 connection even if not active

How ?

① Frequency div. multiplex. (FDM)
- Reserve freq. / call

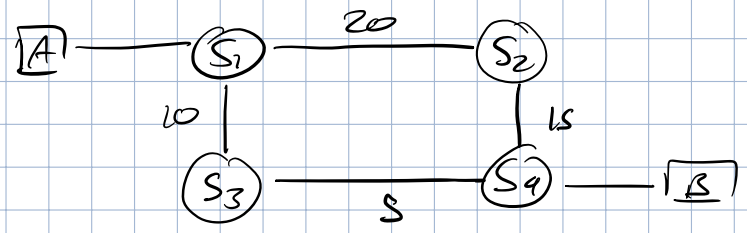
② Time div. mux. (TDM)
- Each call has own tm.

Pros: interference w/ other conn. more, reliable for conn.

Cons:

1. Suboptimal resource alloc.
2. Takes time to alloc. resource
3. Not resilient to network changes.
4. Max # of conn.

Ex:



Q: Total # of calls from A → B?

Procedure: find all paths & take min. # of conn on each path + sum.

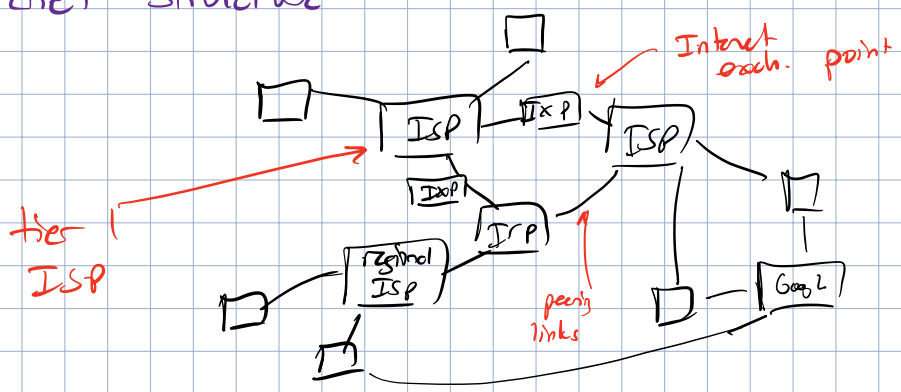
2 paths:

① $S_1 \rightarrow S_2 \rightarrow S_4$: 15 conn. } 20 conn. max,

② $S_1 \rightarrow S_3 \rightarrow S_4$: 5 conn

Reveals pr of packet switching: far more efficient in bursty traffic.

Internet Structure



Alt:

① Pairwise conn: b/w hosts
- Not practical

② Central system directs traffic
- Too centralized

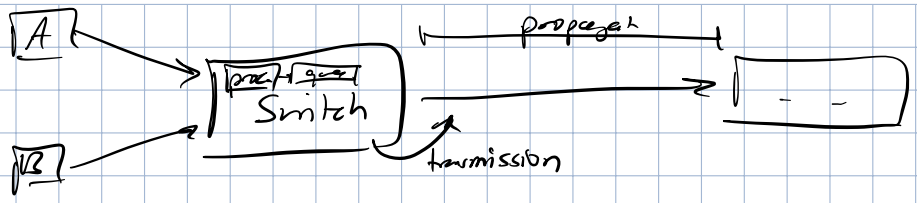
③ Separate ISP.
- Connect ISP together

Companies can run own network.

Performance

① Delay

Cause: queing

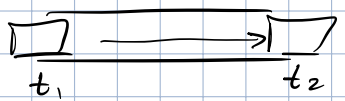


$$d_{nodal} = d_{proc} + d_{queue} + d_{trans} + d_{prop}$$

delay / packet / node

d_{proc}: checking packet & output link
↳ in μsec.

d_{queue}: time at switch waiting for link



$\Rightarrow d_{queue} = t_2 - t_1$

$$\text{Traffic intensity} = \frac{L \cdot a}{R}$$ (a: avg. arrival rate)

Intensity $\begin{cases} \sim 0: & \text{delay small} \\ \to 1: & \text{large delay} \\ > 1: & \approx \text{delay (more work arrives > service)} \end{cases}$

d_{trans}: bandwidth restricted.

$d_{trans} = L/R$ } delay in upload packet

d_{prop}: based off length of link

$d_{prop} = \frac{d}{s}$ → d: length of link, s: prop. speed (2×10^8 m/sec)

Traceroute used to find real delay

① Send n packets
② On each switch, packets sent back
③ Repeat for all switches

② Loss

Packet → full queue ⇒ lose packet

Remediation:

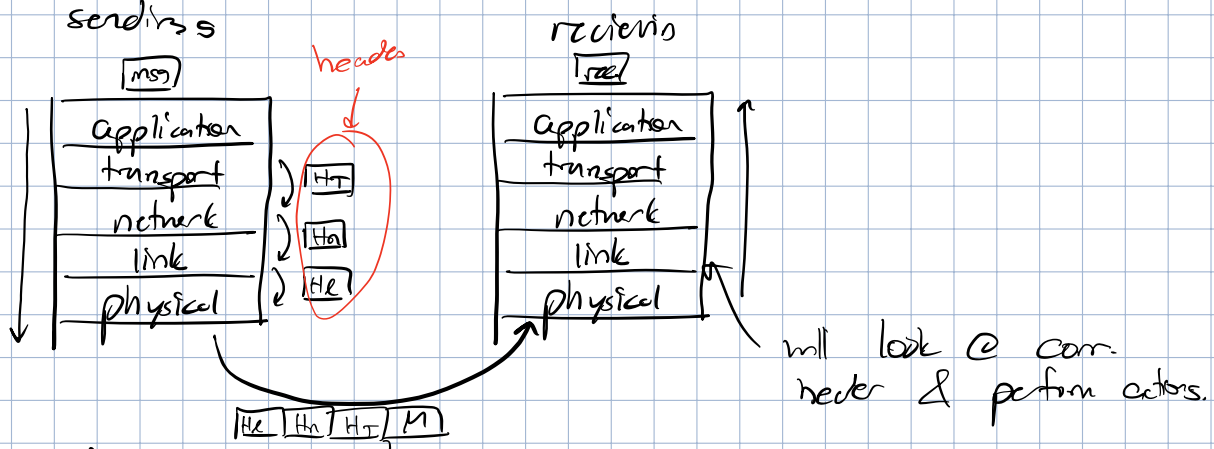
① Retransmit from prev. node ② Don't care } Protocol defines behav.
② " source

③ Throughput

$\frac{\text{bits sent}}{\text{time}}$ → instantaneous: look @ specific time
→ avg.: look @ time range

Throughput of path is limited by slowest link

Layers

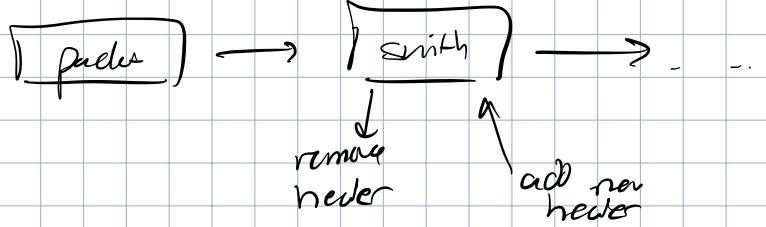


Layer i depends on layer i+1

Layer defn:

- Application: network apps (HTTP, DNS, ...)
- Transport: how to transfer process → process (UDP, TCP)
- Network: routes datagrams from host → host
- Link: route datagram from neighbor
- Physical: physical movement of bits

Not all comp of network implements all layers → only ones that matter



Diff. b/w switch & router: router implements network layer, switch doesn't

Pros: encapsulation

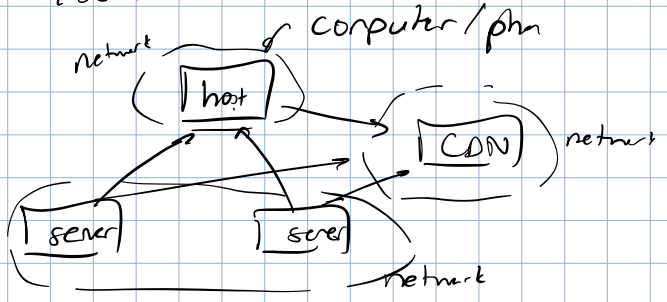
Cons: header overhead

Application Layer

Principles

Network app: runs on diff. machines.

↳ Ex: // YouTube:



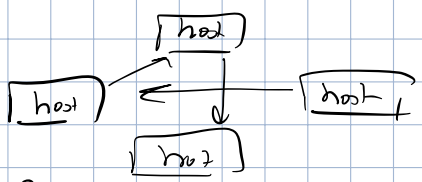
No software needed on apps for network core func. (routing, queuing)

Paradigms

Client-server

- Server is always-on
- Client contacts server for info, no comm. b/w each other
- Protocols: HTTP, IMAP, FTP

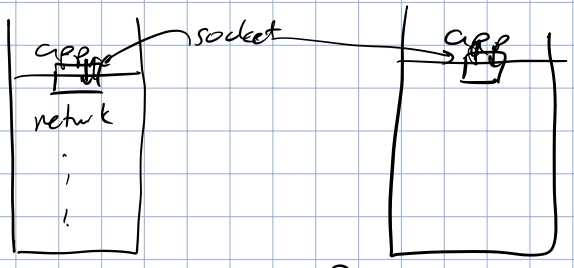
Peer-to-peer



- Peers provide service to each other
- Not always on / connected

Q: How to send messages from proc. in 1 machine → another machine?

A: Sockets.



Q: How to identify proper process?

A: IP addr for machine + port no.
↳ corresponds to proc. (eg. 80 for HTTP server)

App layer protocol defns:

① Type of messages: req./resp.?

② Syntax of messages

③ Semantics of messages

App needs following characteristics of transport servs:

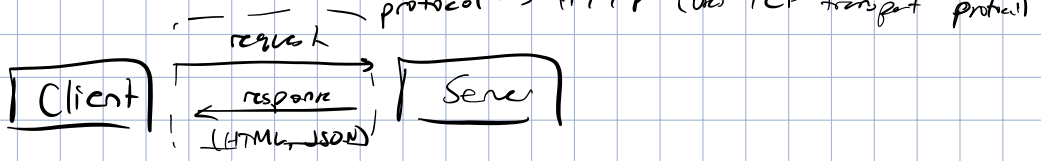
① Data integrity

② Timing

③ Throughput

④ Security

Web & HTTP



HTTP steps:

① Client initiate TCP conn. to serv (port 80)

② Serv accepts conn.

③ HTTP message is exchanged

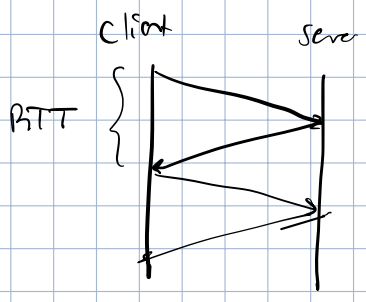
④ Conn. closed

} HTTP is stateless, serv should not know about past requests.

HTTP

Non-persistent

1 packet sent / TCP conn.



∴ Time for getting 1 object:

1 RTT for conn.
+ 1 RTT for request resp.
+ File transmission

Cons:

① OS overhead

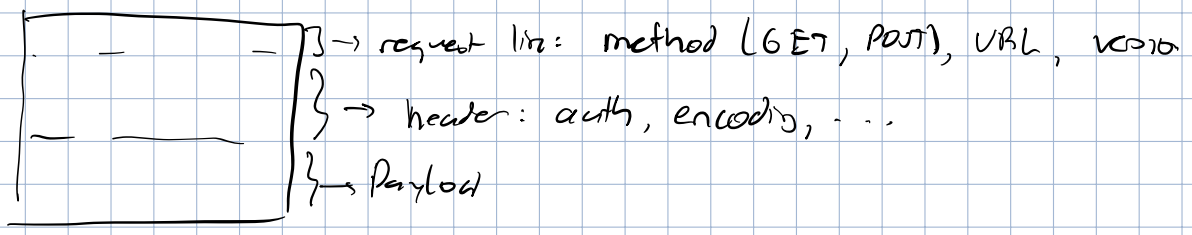
② Not great for parallel conn.

Persistent

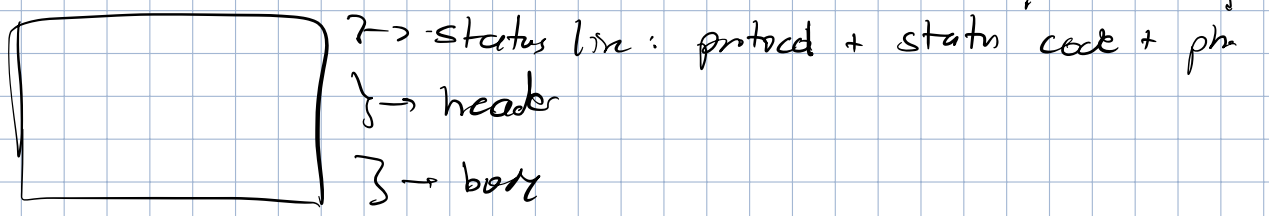
Multiple packets sent / TCP conn.

B/c conn. open, not as much RTT use (as little as 1)

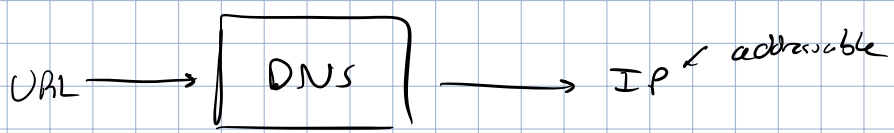
HTTP request:



HTTP response:



DNS



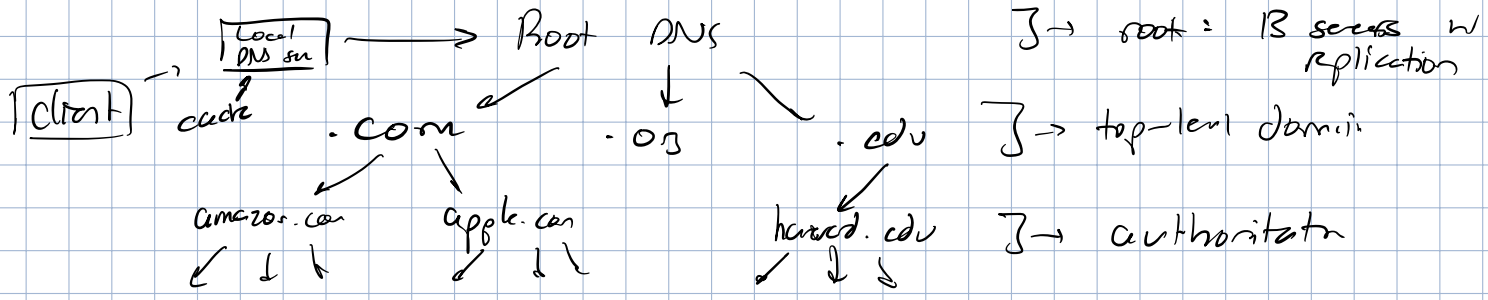
Other func.:

① Host aliasing (diff server w/ same name)

② Mail aliasing (@ uwaterloo.ca)

③ Load balancing

Distributed & hierarchical ← resiliency & performance-sensitive



Traversal start from local DNS server:

① Iterate: go through root → top level → authoritative

② Recurse: let root server recurse & return.

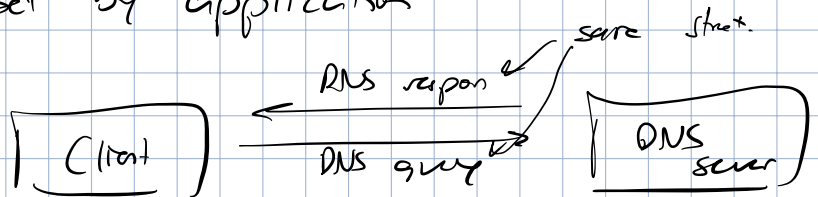
Caching is v. common

DNS record: (name, value, type, ttl)

↳ caching purpose

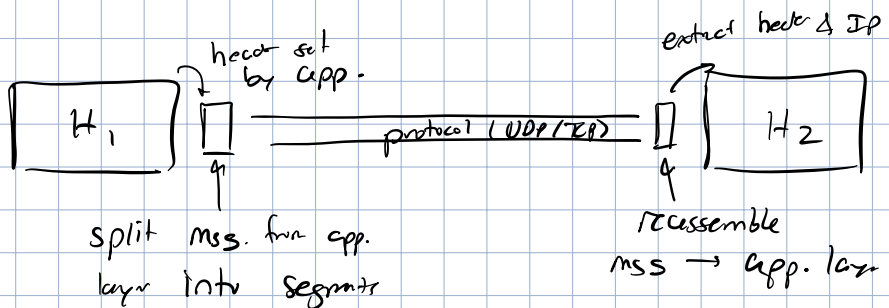
A: (hostname, IP)
NS: (domain, authoritative server)
CNAME: (alias, actual, ...)
MX: (domain, email server)

Set by application



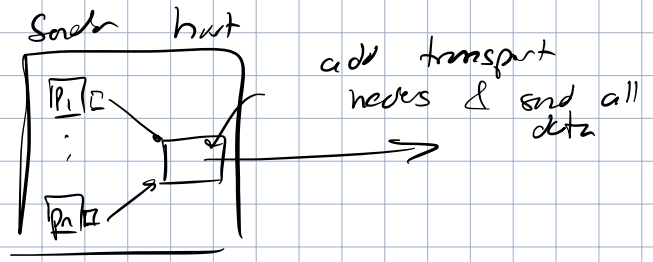
① ID for query
② Flags (eg. recursion)
③ Queries

TRANSPORT LAYER

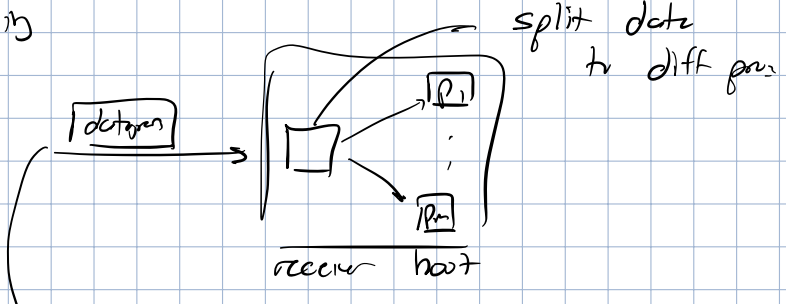


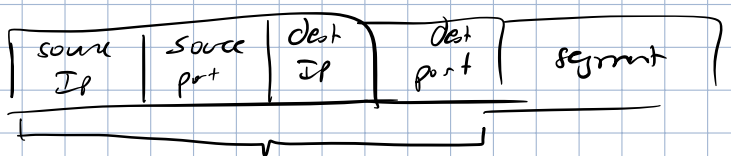
Multiplexing & Demultiplexing

Multiplexing:



Demultiplexing





Use this to figure out who to send msg to.

Ports

Reserved by OS

Free

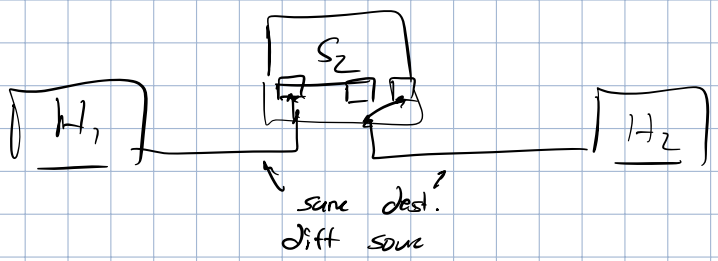
Type of demux:

① Connectionless: only use dest IP & dst port to find correct socket
↳ Used in UDP

If multiple datagrams w/ same dest IP & port, diff. sour IP & port go to same socket

② Connection-oriented: use all 4 data-fields to demux
↳ Used in TCP

Allows multiple TCP sockets b/c not all conn. goes to same socket



UDP: User Datagram Protocol

Bare bones

Best effort: segments can be lost / out-of-order

Connectionless: no handshaking but still 2-way comm.

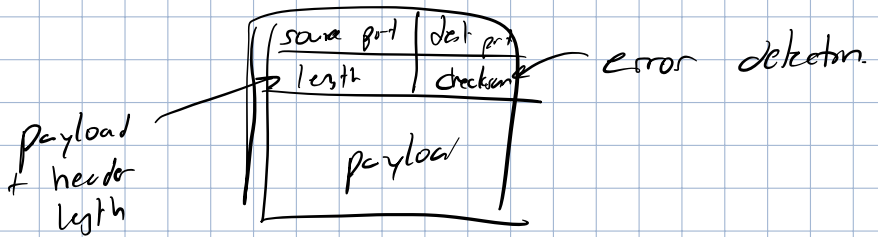
Why?

① Little delay → streaming, video games

② Simple

③ No congestion control

Add anything? Do it in app layer



Principles of reliable data transfer

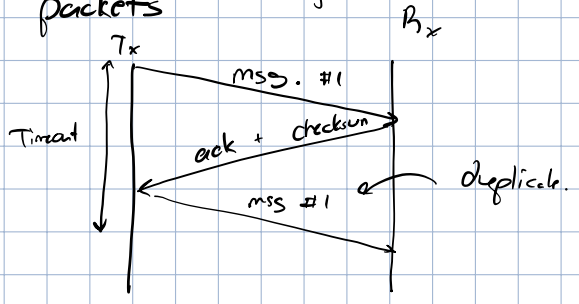
Unreliability of UDP:

① Packet errors ⇒ error det. mechanism needed (ack)

② Packet loss ⇒ feedback to sender & automatic repeat

③ Out-of-order packets } sequence #s for packet

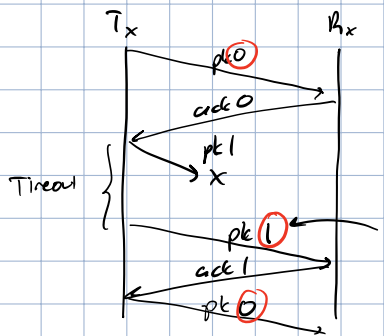
④ Duplicate packets }



Stop-and-wait protocol

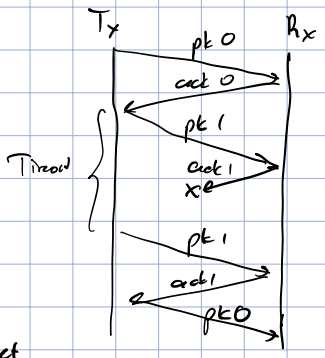
① Send packet w/ sequenc #. If duplicate ⇒ discard

② Wait for ack / timer run out
- i) If ack → next packet
- ii) Else → retransmit

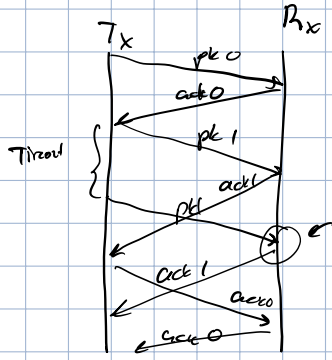


packet loss

sequenc # is 1 bit
⇓
Sends 1 packet at a time, attende is fine

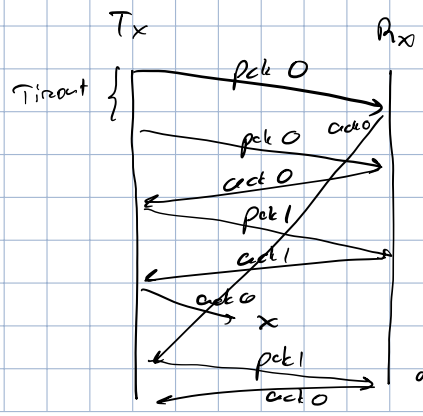


ack loss



duplicate
⇓
sends ack of last received seq

Stuck case:

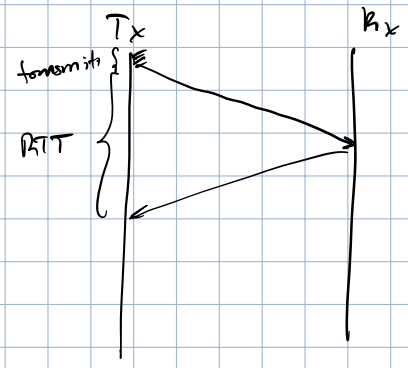


duplicate to receiv

Reason for stuck: timeout too long

Performance of stop-and-wait

$$U_{sender} = \frac{\text{time spent sending}}{\text{total time}}$$



$\Rightarrow$ Total time $= L/R + RTT$

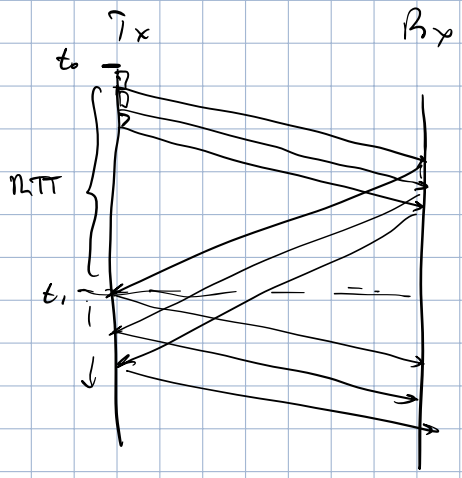
Time transmitting $= L/R$

$$U_{sender} = \frac{L/R}{L/R + RTT}$$

BAD!

Improve? Pipeline!

Def: send multiple unacked packets



$\Rightarrow U_{sender} = \frac{3 \cdot L/R}{RTT + L/R} \Leftarrow t_1 - t_0$ (1st packet ack $\rightarrow$ Tx)

$\downarrow$

3x utilization!

Pipelined protocols

Go-back n

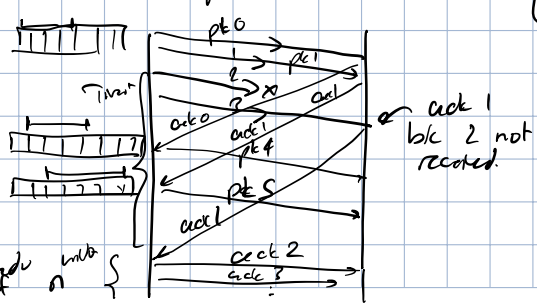
① Window of n packets

② Receiver sends cumulative Ack
- ACK(p) $\Rightarrow$ received all packets up to p (highest in-order seq #)

③ If ACK(p) $\rightarrow$ update cont on Tx & transmit p+1

④ If timeout for oldest (unacked) packet expires $\Rightarrow$ send next n packets starting from unacked packet (move back window)

n=3



Selective repeat

① Packets sent w/ own timer on window n

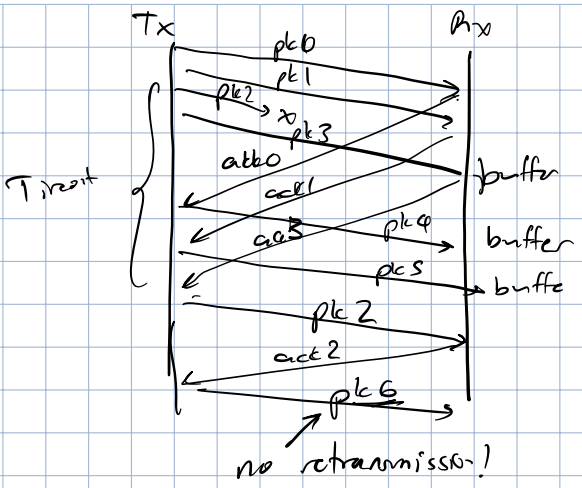
② Sender sends ack/packet

③ If packet/ack lost:
- Sender repeats smallest unacked packet
- No duplicate packets
  - ↳ Out-of-order buffer by receiver

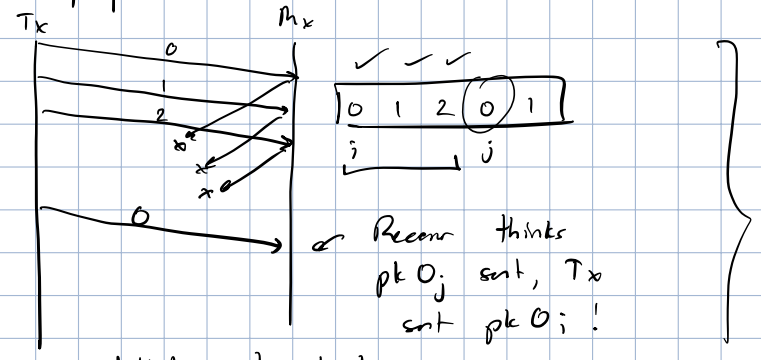
Optimize:

① Buffer packets sent in prev. windows

② Don't wait for time to resend



Selective repeat dilemma:



Window too big!

↳ Overlap possible w/ seq #

Soln: window size $\leq \frac{\text{seq \# space}}{2}$

Pipelined protocol perf.:

Window size W (decided on handshake)

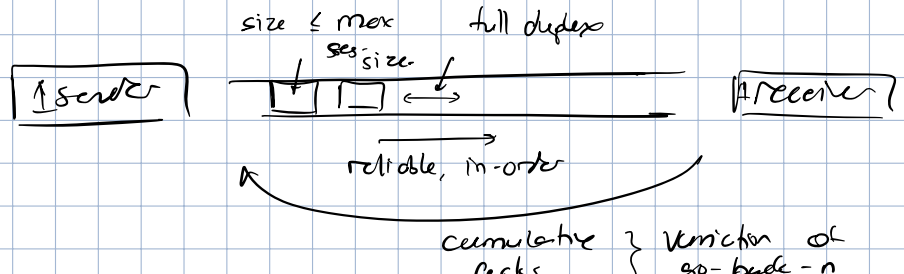
$t_T = L/R + RTT \Rightarrow$ time to send 1 packet & Ack

↳ $\underbrace{W \cdot L/R}_{\text{time to transmit W from}} \leq \underbrace{t_T}_{\text{1st time}} \Rightarrow$ send W packets b/f 1st arrives

Optimal window size $= \frac{\text{time to 1 packet roundtrip}}{\text{time to transmit 1 packet}} = \frac{t_T}{\underbrace{L/R}_{t_i}}$

$\therefore$ Utilization $= \min\left(1, \frac{W \cdot t_i}{t_T}\right)$

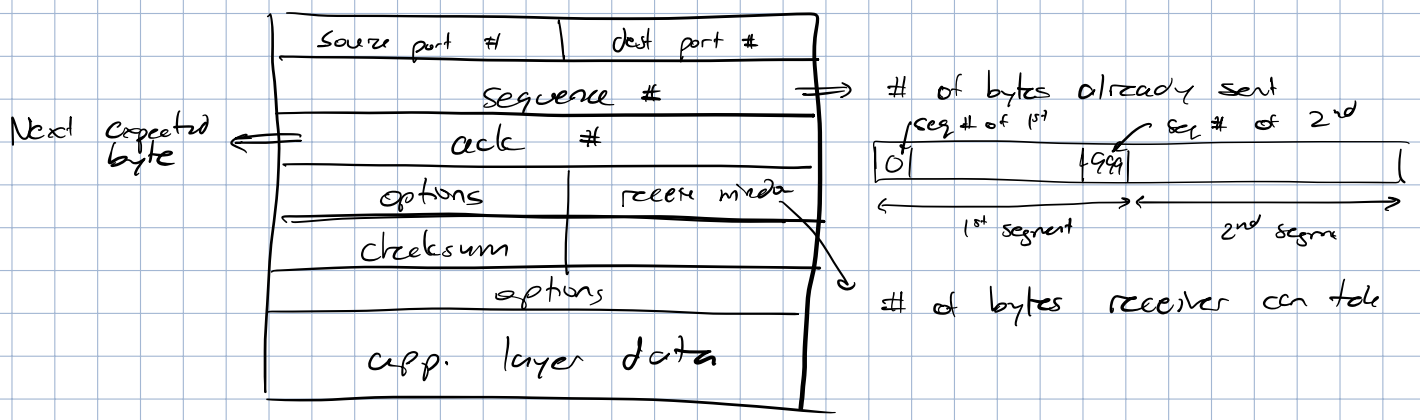
TCP



Other properties:

1. Pipelined
2. Handshaking to esteblish conn.
3. Flow control
4. Congestion control

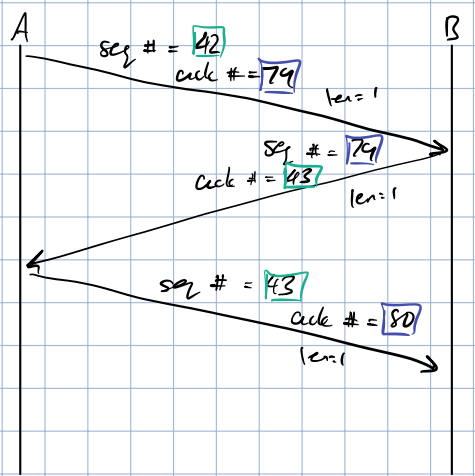
Segment structure:



If out-of-order segments: implementation can decide

- Can discard
- Can buffer (most common)

Full duplex:



TCP sender:

(A) Data from app. layer:

1. Create segment w/ seq # & receive window
2. Start timer if not running (oldest unAcked segment)

(B) Timeout

1. Retransmit segment

2. Timer restart

© ACK receive

1. Update oldest unACKed
2. Update timer

TCP receiver:

(A) In-order segment arrives w/ expected seq. #. Everything up to seq # is ACKed

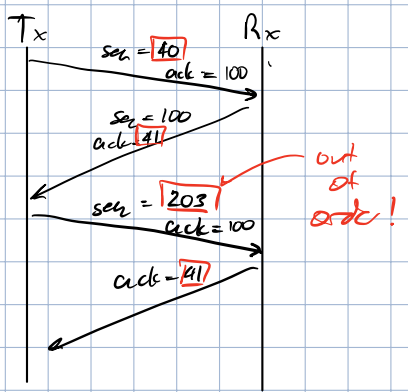
prev. receiver response ACK #

- Ideal
- Delay for 500ms → if no segment, send ACK
  - ↳ Minimizes ACKs b/c cumulative

(B) In-order segment arrives w/ expected seq. #. Some other segment has not sent ACK (A)

Send cumulative ACK ⇒ ACK every 2nd packet

(C) out-of-order segment w/ higher-than-expected seq #



Send duplicate ACK

Buffer

(D) Segment completes gap

Send ACK if segment completely fills gap

Q: What is diff. b/w TCP & go-back-n?

TCP retransmits oldest un-ACKed packet rather than entire window

Even a scenario when it will not retransmit if cumulative ACK covers lost ACK

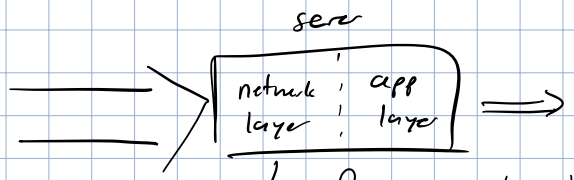
When to resend?

① Timeout

② Triple duplicate ACK (fast retransmit)

↳ Usually faster than timeout

Flow control for TCP



↳ Receiver gets too overloaded, buffer full → drops packets

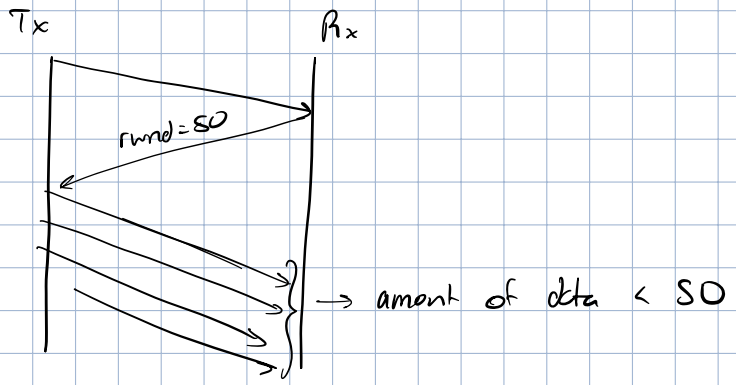
Soln:

① Receiver sets rwnd field in TCP header

↓

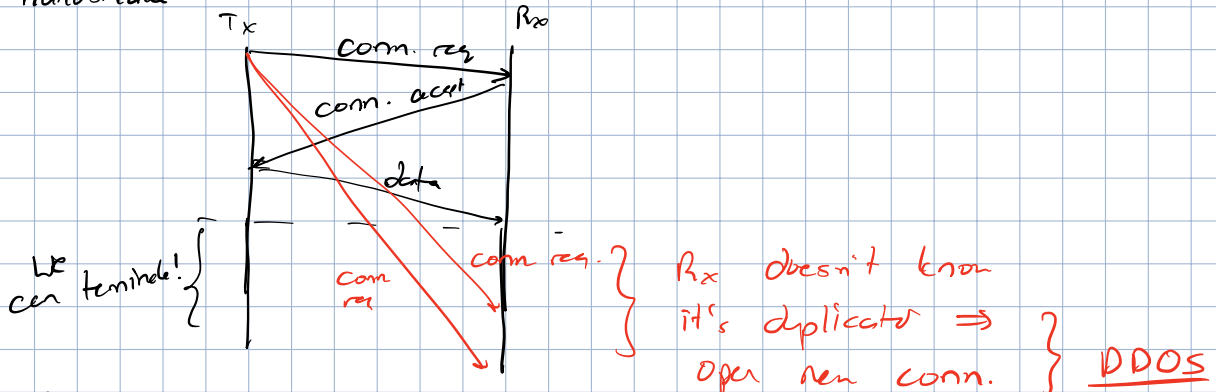
amount of free space in buffer

② Sender sets rwnd ⇒ limit amount of unACKed data to rwnd bytes.

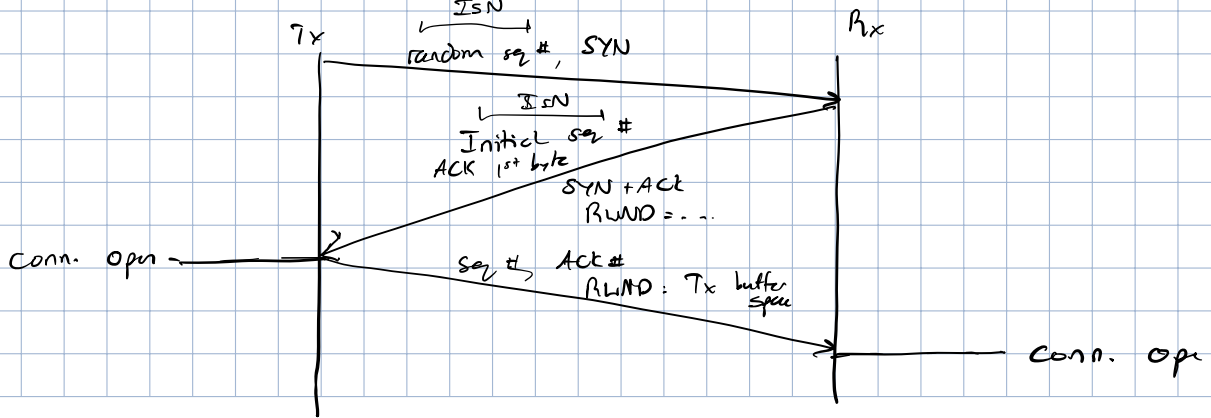


TCP Conn. Establishment

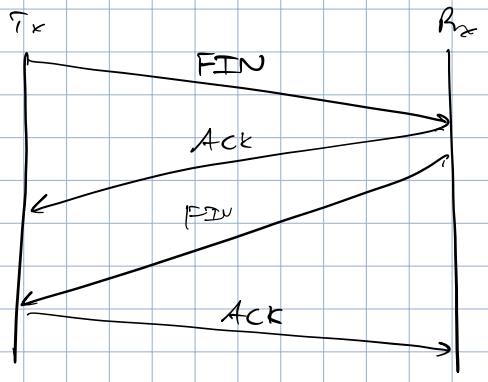
2-way handshake



3-way handshake:



TCP Closing Conn.



Congestion Control Principles

Congestion: too many senders sending traffic → overwhelming
↳ Flow control: 1 sender too fast

Approaches

E2E control

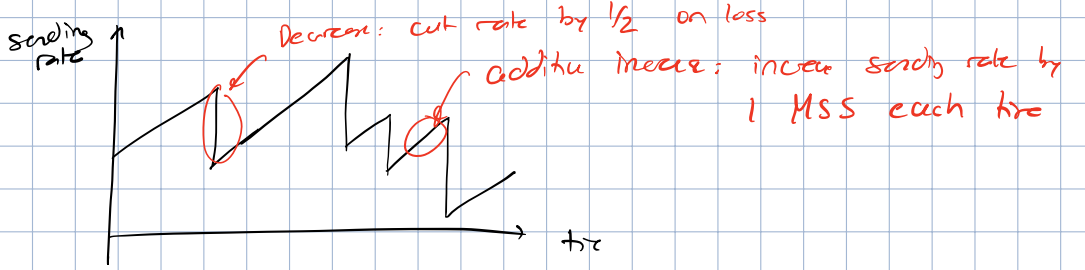
Infer congestion from delays → modify rates

Network-assisted control

Routers comm. flow rate & explicit congestion → senders modify

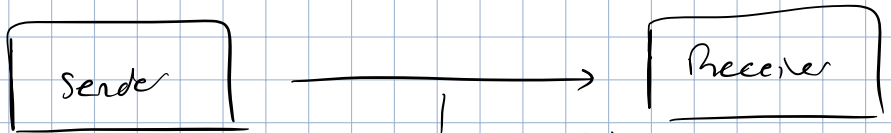
TCP Congestion

Additive increase, multiplicative decrease



Cases:

1) Ideal case



amount of data = min(cwnd, rwnd)

sender window; window set by receiver for flow control

cwnd dynamically adjusted as messages received & acked

2) Timeout loss:

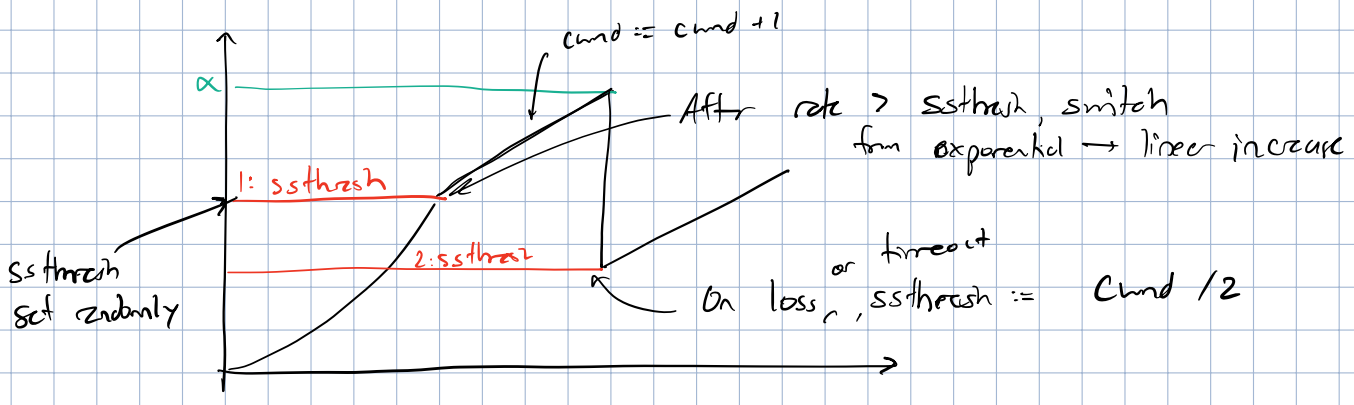
cwnd = 1 MSS

3) Lost packet (3 duplicate ACK):

↳ TCP Reno: cwnd := cwnd / 2

↳ TCP Tahoe: cwnd := 1

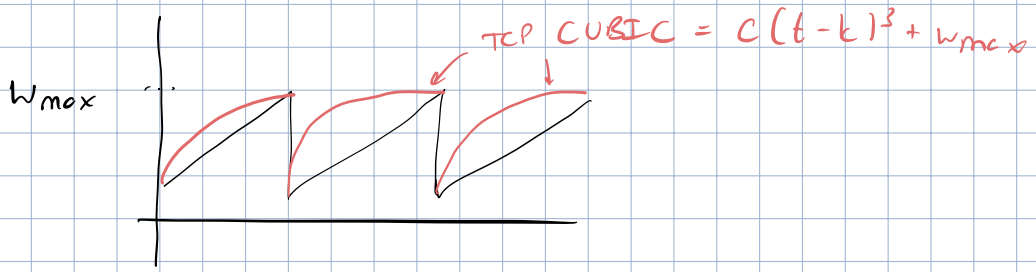
Additive increase details:



TCP throughput: $\frac{\text{avg. window size}}{\text{RTT}} = \frac{3}{4} \cdot \frac{\text{max window}}{\text{RTT}}$

Q: Rather than doing linear increase, better way?

↳ Ans: TCP CUBIC



V. popular TCP variation!

NETWORK LAYER

Overview

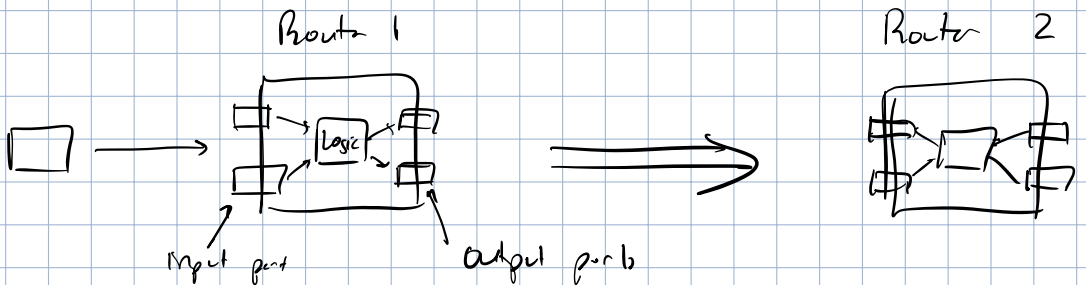
Objective:

① Route packets from host → host (software algos) } Packet delivery across network

② Forward packets from input → output ports (hardware)

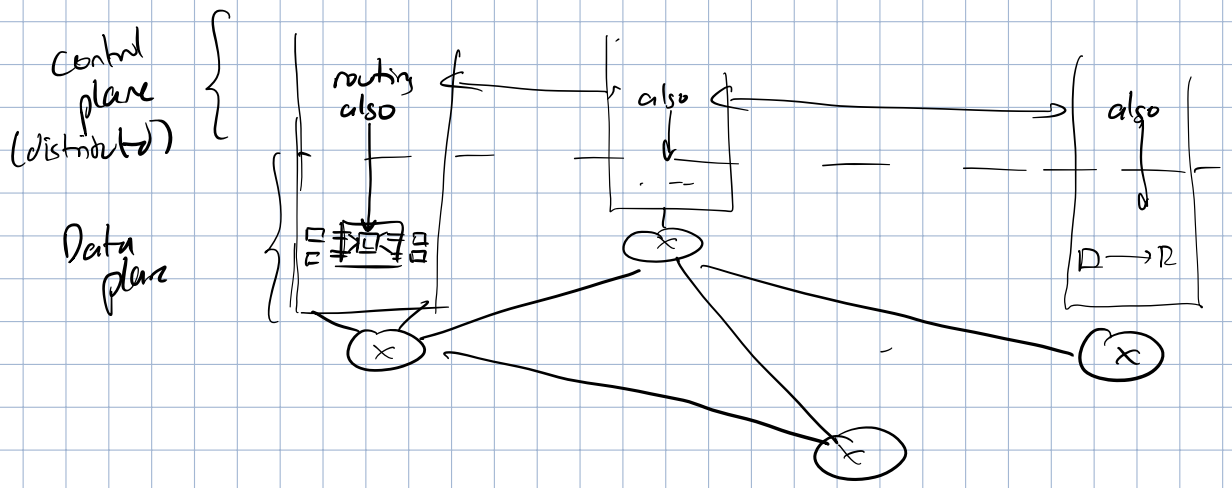
Every Internet-connected device has network layer

Core component: router

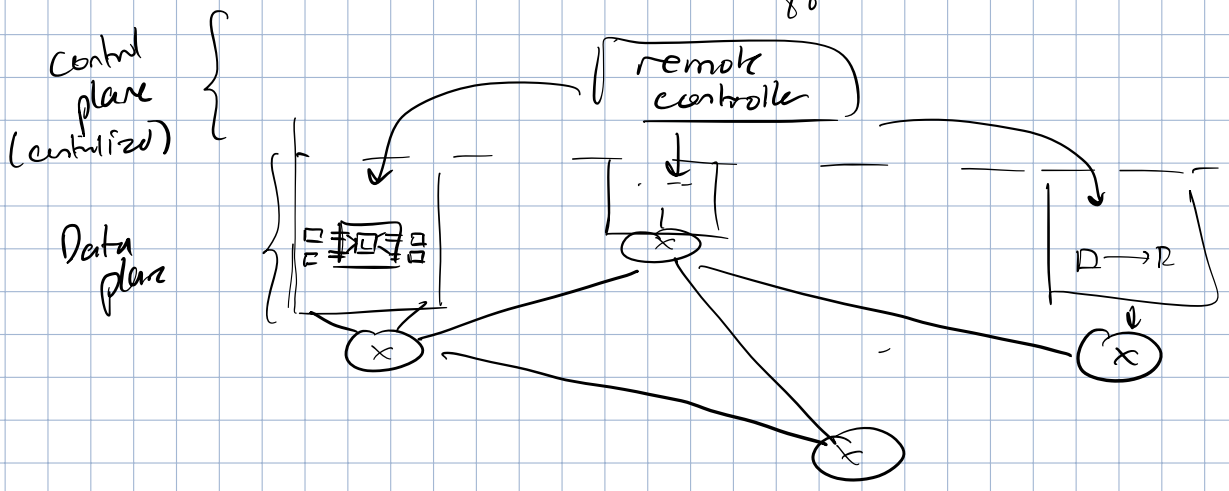


Control plane vs. data plane

① Traditional routing algo approach



② Software defined networking approach:

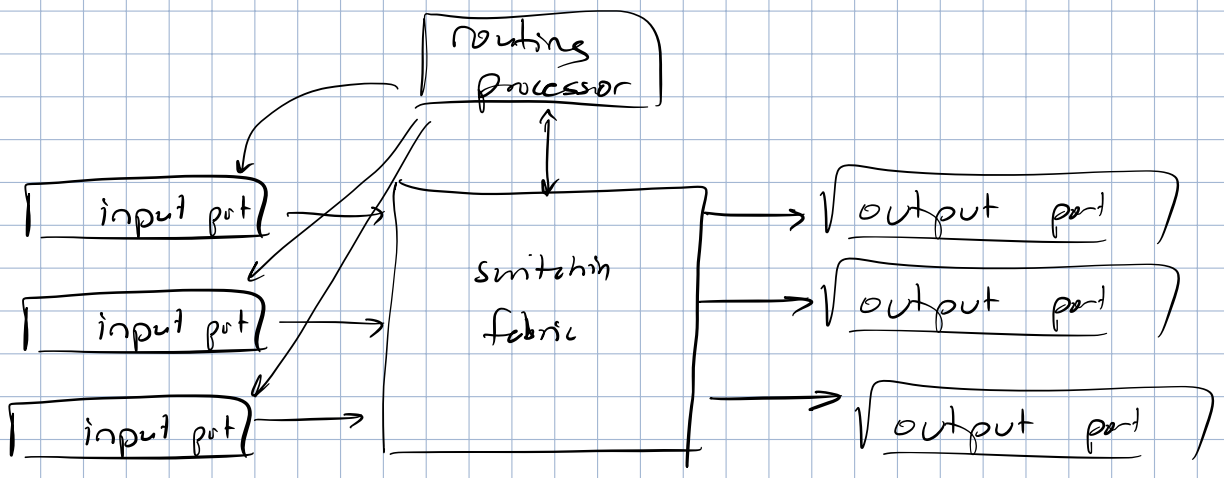


Data plane uses local forwarding table

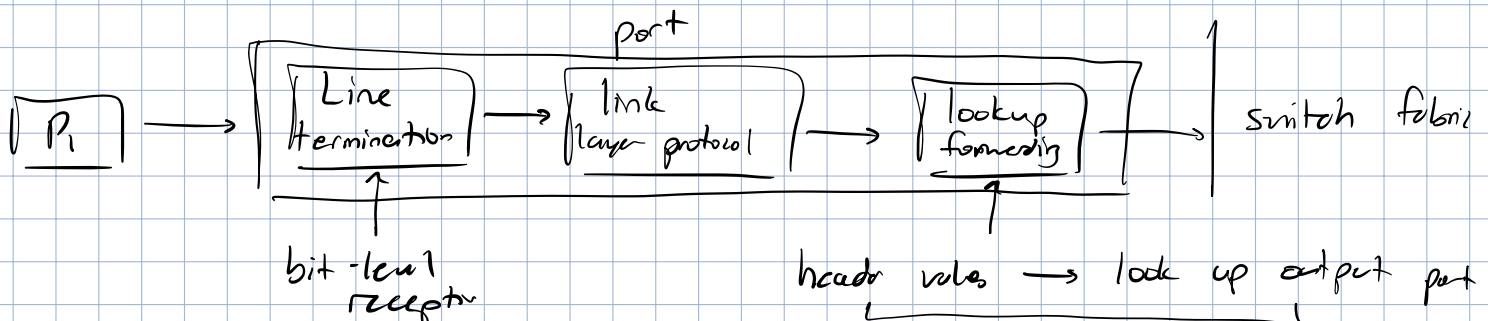
header bits	output
0111	4
1011	3

⟸ determined by control plane

Router



Input port functionality



Goal: rate of comp. should be equal to "line speed" = rate of arrival

If processing time >> arrival rate ⇒ queing

Dest. based forwarding: only use dest. IP addr. for forwarding

Switching fabric

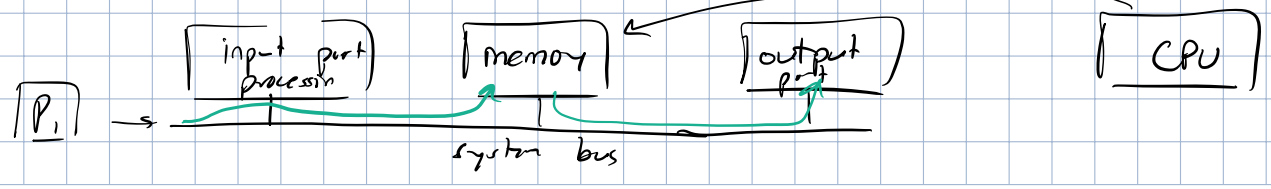
Obj: move packets from input port → output port

Switching rate: rate of packet movement

↳ Ideal case: NR (N: # of input ports, R: arrival rate)
no queing can occur

Types:

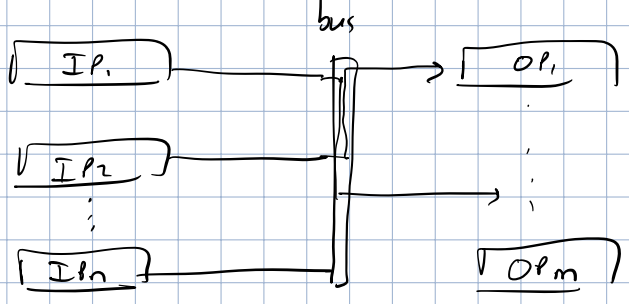
① Memory switching (1st gen routers)



Limitation:

1. Packet moved 1 at time b/c packet blocks sys-bus
2. 2 bus crossings ⇐ limited by memory bandwidth
   ↳ Time to send: $L/B \times 2$

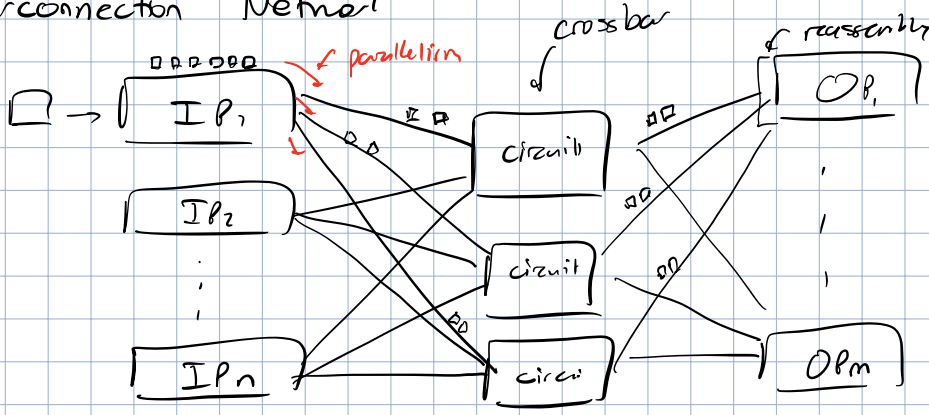
② Bus switching



⇒ Rate: L/B

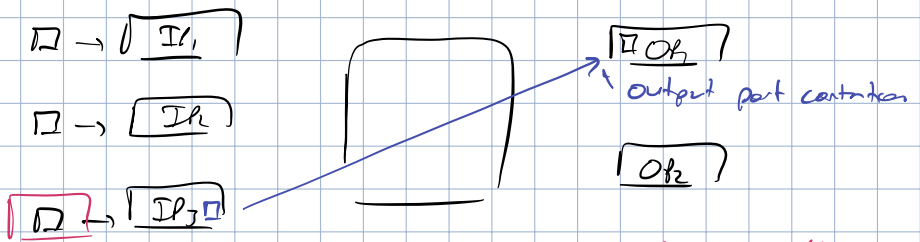
Limitation: bus contention (multiple packets go to same OP_i)

③ Interconnection Network



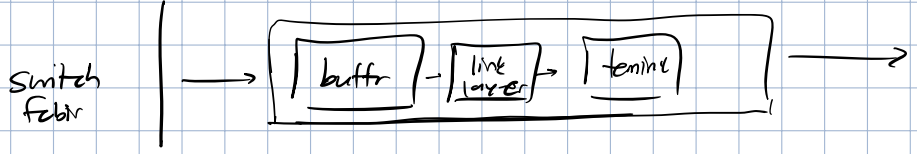
Can add more crossbar circuits to form multistage switch

If not fast enough → queuing



↖ also blocked ⇒ head-of-line blocking (b/c □ blocks □)

Output port



Q: Why do we need buffer in OP?
A: Link layer rate < switch fabric rate

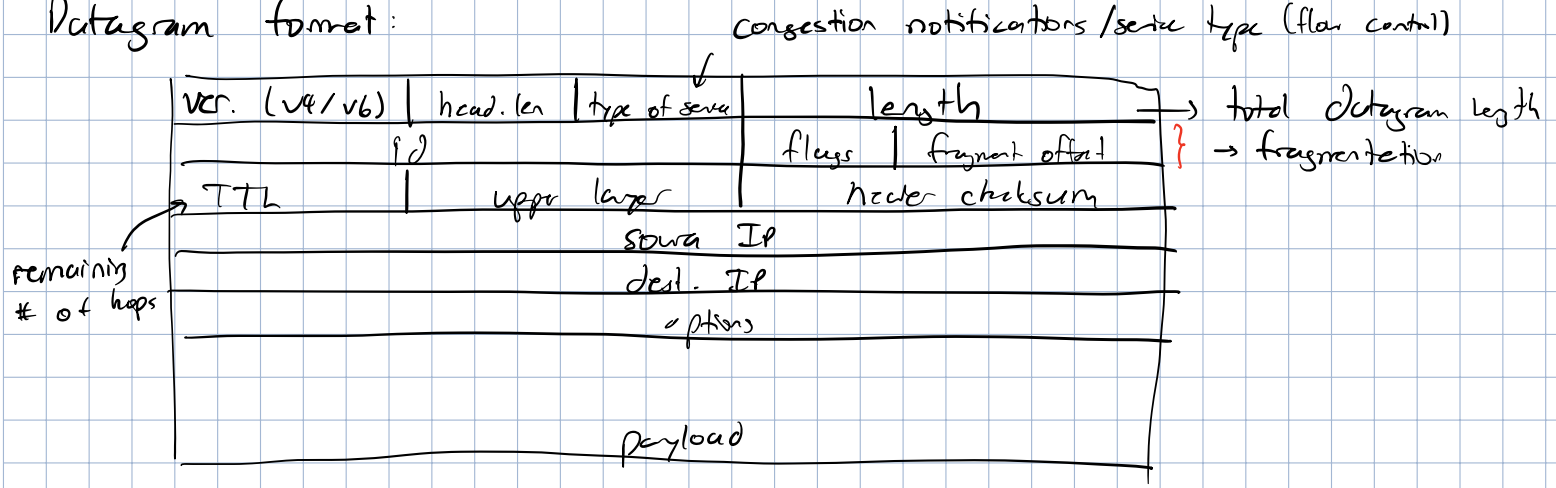
Q: What if too many packets
A: Drop policy

Q: FIFO serve from buffer
A: No! Priority scheduling discipline

Internet Protocol (IP)

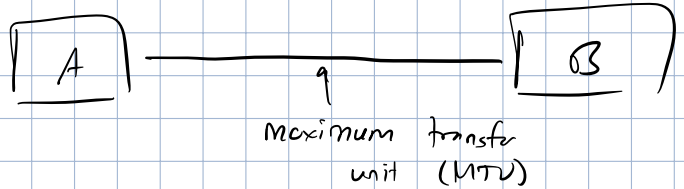
Resp: datagram format, addressing, packet handling

Datagram format:



Length = 20 byte of TCP header + 20 byte of IP header + payload

Fragmentation:



If IP datagram > MTU ⇒ split datagram into fragments

↳ Reassemble datagrams at destination, not at every node

If nofrag flag = 1 ⇒ no fragmentation, send error if ||datagram|| > MTU

How to reassemble fragments? Header.

↳ Each fragment has length field, offset field (8 byte multiples)

Ex:// Datagram: | length = 4000 | ID = x | flag = 0 | offset = 0 |, 20 byte header

MTU: 1500 byte

∴ Payload = 4000 − 20 = 3980 byte

Fragments: (1480 byte of payload, 20 byte of header)

A: length = 1500 ID = x flag = 1 offset = 0

B: length = 1500 ID = x flag = 1 offset = 185 ← 1480 bytes in 1st datagram / 8

C: length = 1040 ID = x flag = 0, offset = 370 ← 185 + 1480/8

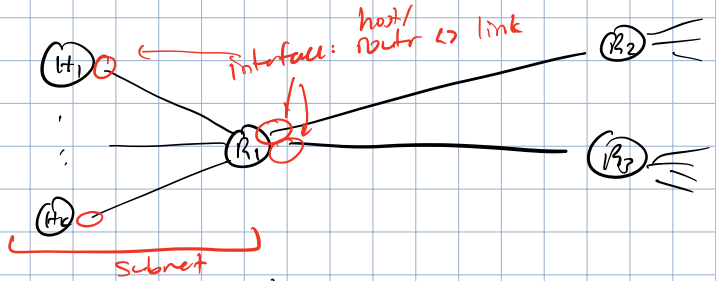
{ length: remaining + 20 byte = (3980 − 2×1480) + 20 = 1040

# of datagrams = $\left\lceil \frac{\text{payload size}}{\text{MTU - header size}} \right\rceil$

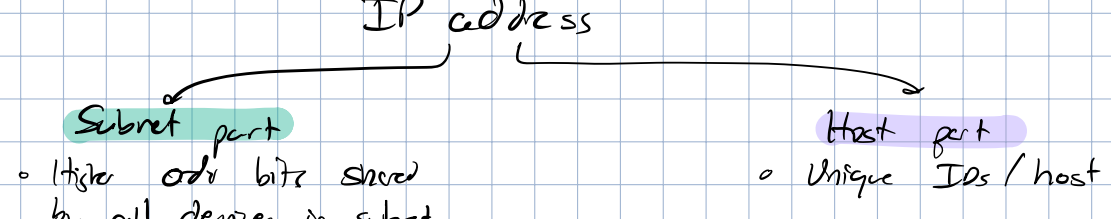
∴ MTU ≥ header size ⇒ if not, non-header data cannot be sent

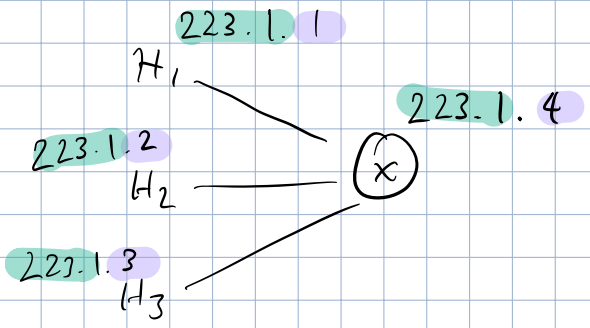
IP Addressing

IP address: 32-bit ID for a host or router interface



Subnet: Collection of devices that can communicate w/out route

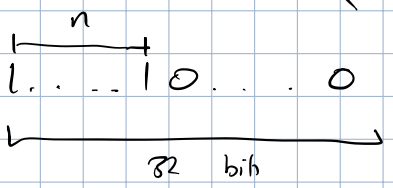




Q: how to know what part is subnet & host?

A: Subnet masking

IP addr /n ↖ # of higher order bits for subnet

Helpful tip: 

AND IP addr = subnet part

Q: How to assign IP addresses to subnet?

① Classful addressing

Fixed length prefixes

A: Prefix of 8 bits, start w/ 0 ⇒ 2^{24} hosts

B: Prefix of 16 bits, start w/ 10 ⇒ 2^{16} hosts

C: Prefix of 24 bits, start w/ 110 ⇒ 2^{8} hosts

Pros: simple

Con: fixed # of addresses

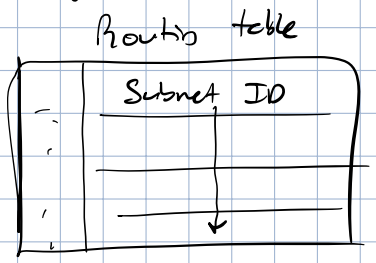
② Classless

CIDR: classless interdomain routing

Subnet masking: a.b.c.d /n

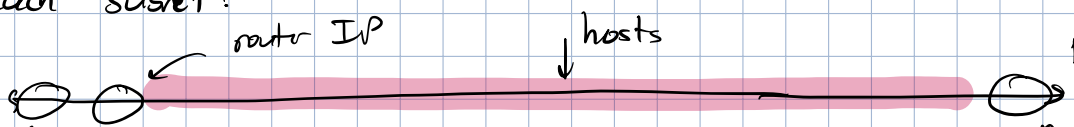
* Important: Subnet mask is consec. 1s follow by consec. 0s *

Q: How to send messages w/ IP addresses & subnet masks



⇒ Does not have host part

For each subnet:



Net ID
subnet part. 0. 0...

IP addr for subnet

Broadcast addr
IP to send msg to all hosts in subnet

Ex:// Network has IP 10.2.5.16 /28. Router IP? Broadcast IP?

① Subnet mask:

11111111. 11111111. 11111111. 11110000

255. 255. 255. 240

128
64
32
16

② Network ID:

10.2.5.16 → 00010000 ← No 1s

& 255.255.255.240 → & 11110000

00010000

10.2.5.16

③ Broadcast IP:

Make host part all 1s

10.2.5 | 00011111 ⇒ 10.2.5.31

↖ 31

Ex:// Given IP addresses 192.168.16.0 /24. We need to create 4 subnets w/ following # of hosts.

- LAN A: 120 hosts
- LAN B: 50 hosts
- LAN C: 12 hosts
- LAN D: 28 hosts

① Find # of addresses needed / LAN

A: 120 + 2 = 122

↖ network + broadcast

B: 52

C: 14

D: 30

② Find least # of bits necessary per network to distinguish hosts

A: $\lfloor \log_2 122 \rfloor + 1 = 7$ bits

B: $\lfloor \log_2 52 \rfloor + 1 = 6$ bits

C: $\lfloor \log_2 14 \rfloor + 1 = 4$ bits

D: $\lfloor \log_2 20 \rfloor + 1 = 5$ bits

③ Subnet mask / network

Formula: 32 − host bit #

A: 32 − 7 = 25 bits

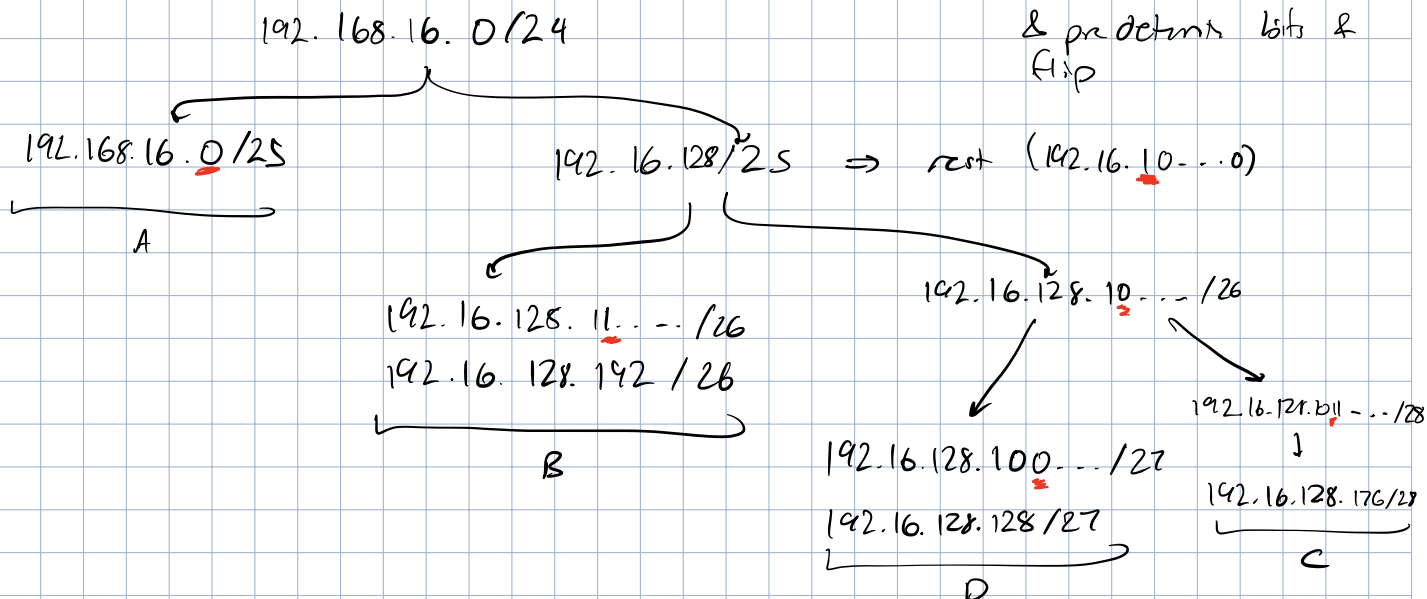
B: 32 − 6 = 26 bits

C: 32 − 4 = 28 bits

D: 32 − 5 = 27 bits

④ Split

Idea: work from large → small & predetermine bits & flip



⑤ *Optional but useful* Calculate network & broadcast IPs to ensure no overlap!

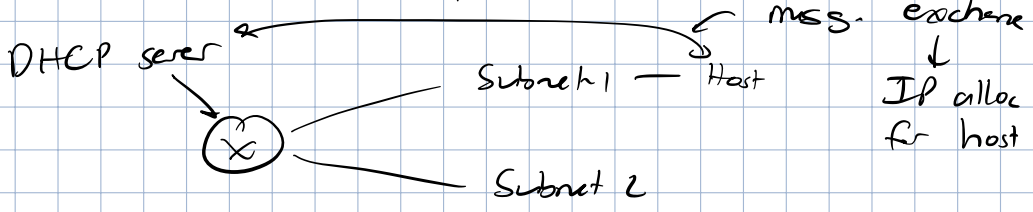
Q: How can host get own IP address?

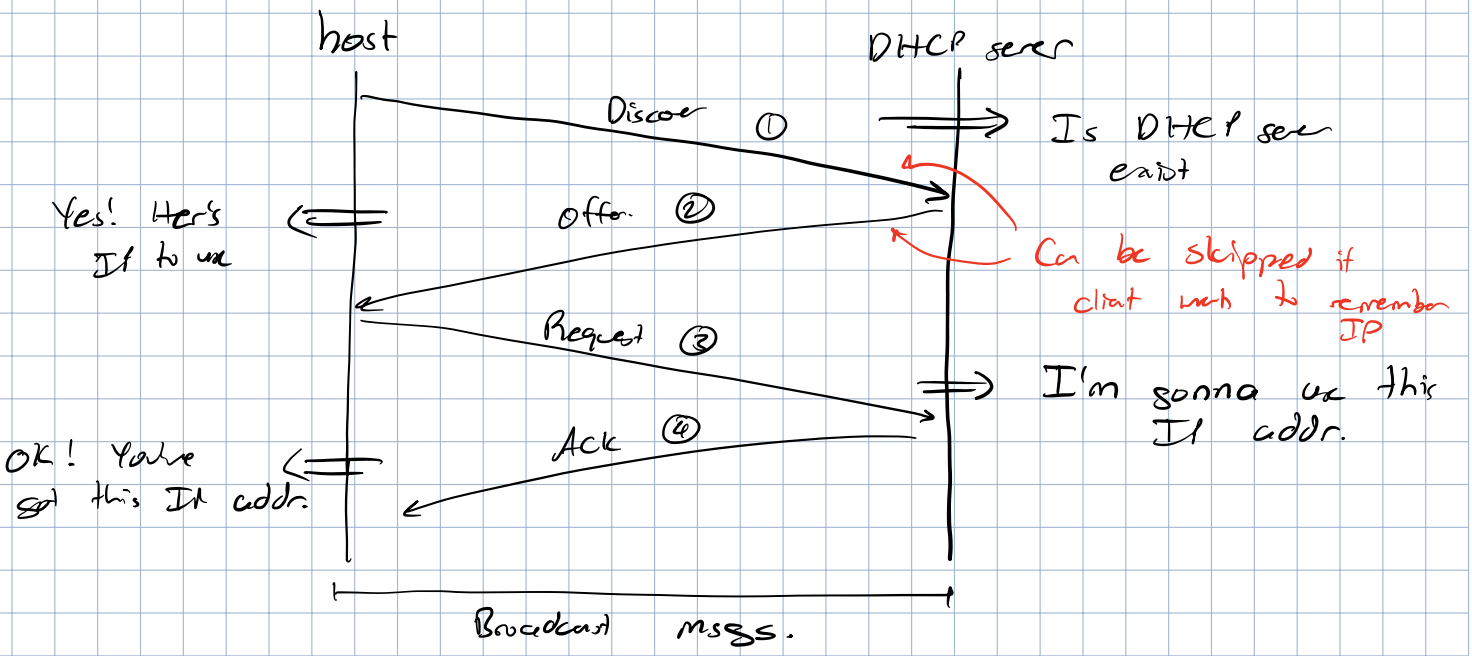
Methods

Static

Dynamic

A: DHCP (Dynamic host config. proto.)





① src: 0.0.0.0, src port #
dest: 255.255.255.255, dest port #
yiaddr: 0.0.0.0
transaction ID: ID for entire process.

② src: DHCP IP addr., port #
dest: dest of ①
yiaddr: host IP addr. it can use
ID: ||
lifetime: TTL for IP addr. ⇒ request for new use.

③ src: src. of ②
dest: broadcast IP
yiadr: host IP addr. in ①
ID: ||
lifeti: ||

④ src: src of ②
dest: dest of ③
yiadr: ||
ID: ||
lifetim: ||

} UDP

DHCP also passes router, DNS IP + network mask.

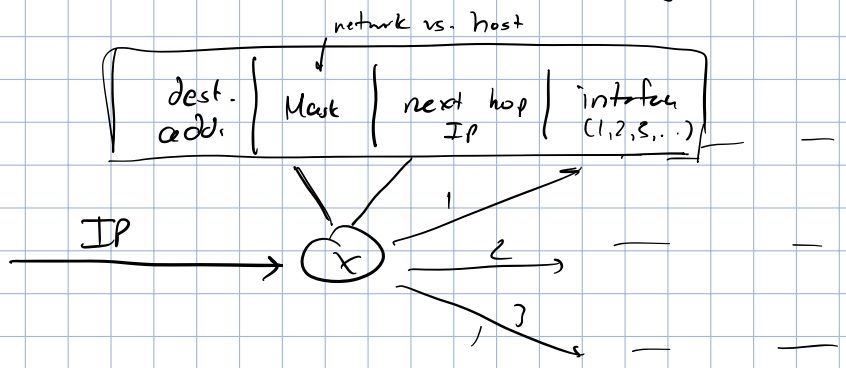
Q: How does network get IP

A: ICANN → IP addresses w/ 5 regional registries

Not enough IP addr in IPv4 (32 bit) → IPv6 (128 bit addr.)

Packet forwarding

Most routers use table-driven routing:

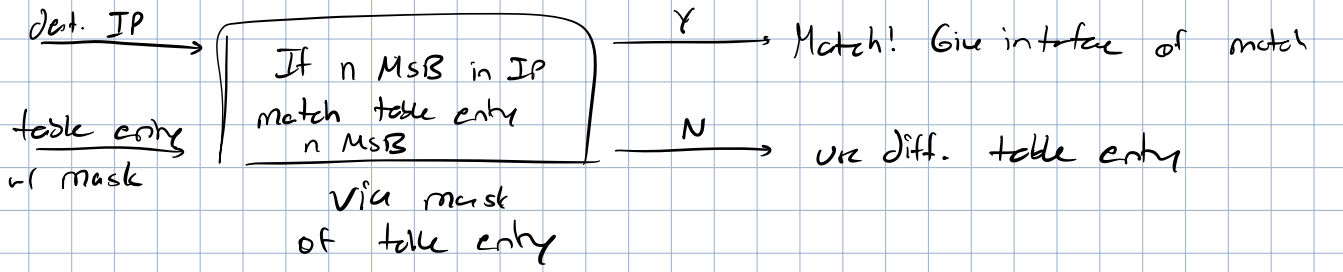


Each table has default entry if no matches

Routing protocols det. table content

Q: How to figure out right interface given dest. IP?

A:



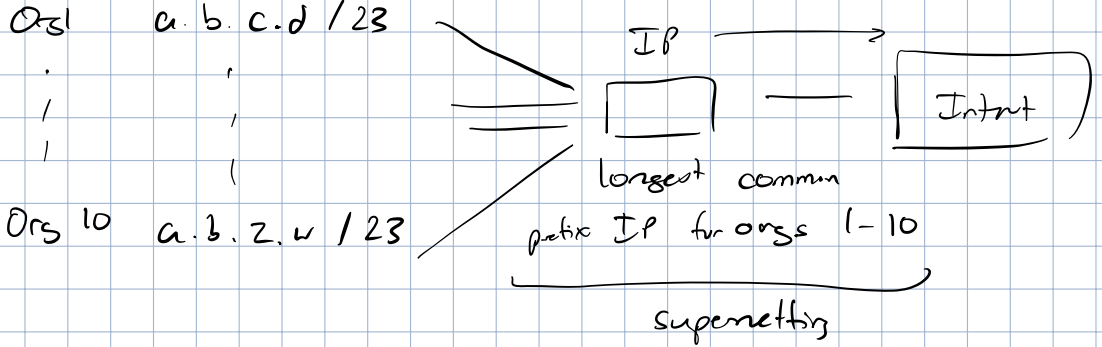
If multiple matches happen, choose entry w/ longest match bits

Ex:// If IP1 matches w/ entry 1 for 5 bits, matches w/ entry 2 for 10 bits, choose entry 2 interface.

Hierarchical addressing

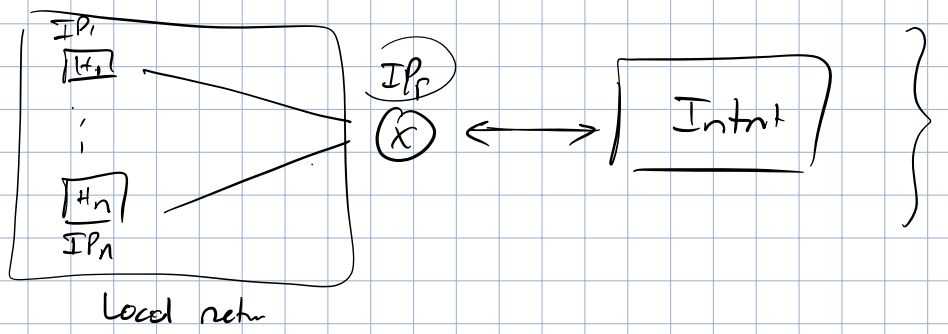
Problem: we don't want to publish all IPs of networks → tables get bloated

Soln: hierarchical addressing



Network address translation (NAT)

Idea: do hierarchical addressing at smaller scale



IP1 ... IPn have same prefix: 10/8, 172.16/12, 192.168/16

↳ For any local network only IPr has to be unique

Pros:

1. Reduces # of unique IP addresses needed
2. Can change IP of host w/out entire Internet knowing } decoupling
3. Can change ISP IP w/out changing host IP
4. Security b/c local hosts not directly addressable

Cons:

1. Routers shouldn't process below layer 3
2. Violates port-to-port agreement ← router is changing port #
3. How to connect to server directly if behind NAT?

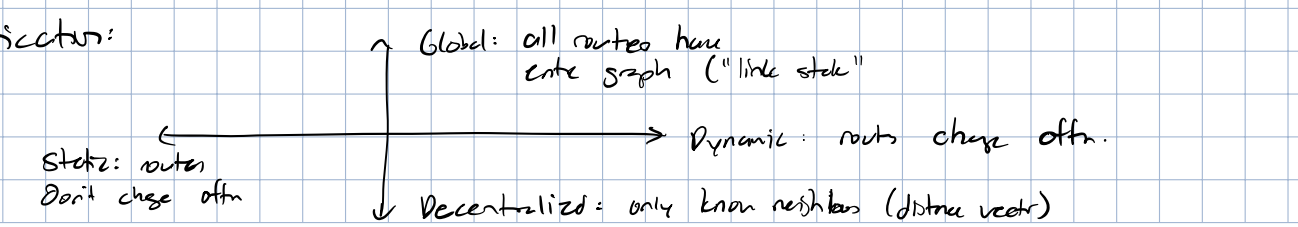
Responsibilities:

1. Outgoing datagrams: replace src IP w/ NAT IP, port #
2. Remember (NAT IP, port #) <> (src IP, port #)
3. Replace NAT IP w/ src IP for incoming datagrams

Routing protocols

Resp.: figure out best path from $H_1 \rightarrow H_2$

Classification:



Link state

Centralized ← entire network known to each host

Finds shortest path from node u to all node in network

↳ After k iterations of algo, we know least cost path up to k dest.

Algorithm (Dijkstra's algo):

Initialization:

$N' = \{u\}$

$\forall$ node v:

if v is adj to u:

$D(v) = c_{u,v}$

else:

$D(v) = \infty$

Loop:

find node w not in N' s.t. $D(w)$ is minimum.

$N' := N' + \{w\}$

for all nodes v adj. to w:

$D(v) = \min(D(v), D(w) + c_{v,w})$

until $|N'| =$ # of total nodes.

Do algo for each node to figure out least cost path ⇒ routing table.

Distance vector

Based off Bellman-Ford DP algo:

Intuition:

① $D_x(y)$: least cost path from $x \to y$ → distance vector

② $D_x(y) = \min_v \{c_{x,v} + D_v(y)\}$

↑ v: all neighbors of x

Whichever node is min is next hop on way to dest.

In practice:

① Node send own distance vect to neighbors occasionally

② Neighbor node update own distance vector via DP algo

③ Should converge to actual least cost path

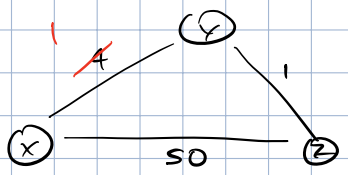
④ Notify neighbors if DV changes

Properties: decentralized, async, self-stopping

Initially: DV of node is only costs to neighbor.

Fast to link cost changes if good, slow if not good

Ex: I



At t_0: y detects change → notify neighbor

$$D_y = \begin{bmatrix} D_x \\ D_z \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

At t_1: z gets update → updates

$$D_{z\ bef} = \begin{bmatrix} D_x \\ D_y \end{bmatrix} = \begin{bmatrix} 5 \\ 1 \end{bmatrix}$$

$$D_{z\ aft} = \begin{bmatrix} D_x \\ D_y \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

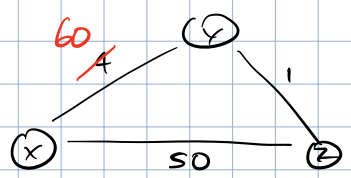
At t_2: y gets z's update:

$D_{y\ before} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ ← Equal

$$D_{y\ aft} = \begin{bmatrix} D_x \\ D_z \end{bmatrix} = \begin{bmatrix} \min(1,\ 1 + D_{z,x} = 3) \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

Stop notif.

Ex: II



t_0: y detects and updates

$$D_{y_b} = \begin{bmatrix} D_x \\ D_z \end{bmatrix} = \begin{bmatrix} 4 \\ 1 \end{bmatrix}$$

$C_{y,z}$

$$D_{y_a} = \begin{bmatrix} D_x \\ D_z \end{bmatrix} = \begin{bmatrix} \min(60,\ D_{zx} + 1 = 5 + 1 = 6) = 6 \\ 1 \end{bmatrix}$$

t_1: z detects & updates

$$D_{z\ b} = \begin{bmatrix} D_x \\ D_y \end{bmatrix} = \begin{bmatrix} 5 \\ 1 \end{bmatrix}$$

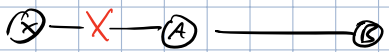
$$D_{z_a} = \begin{bmatrix} \min(50,\ 1 + D_{y,x} = 1 + 6 = 7) = 7 \\ 1 \end{bmatrix}$$

t_2: y detects & updates

$$D_{y_a} = \begin{bmatrix} \min(60,\ 1 + D_{zx} = 1 + 7 = 8) = 8 \\ 1 \end{bmatrix}$$

⋮ Count to infinity problem

2-node instability problem



→ takes long time for both nodes to realize infinite cost to get to X

Soln:

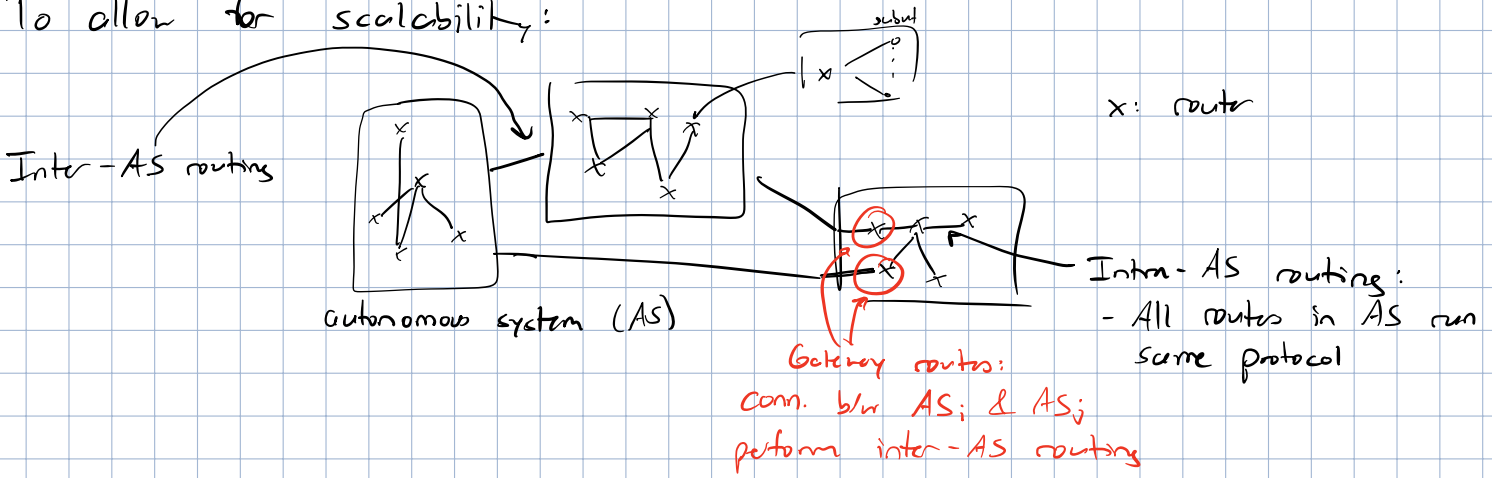
① Split horizon: B doesn't tell A about path $X \leftrightarrow A$, assumes A knows.

② Poisoned reverse: A tells B about infinite cost when getting route to $X \rightarrow B$ stops.

OSPF

We want to scale up soln. to routing & allow for autonomy in networks

To allow for scalability:



These routing mechanisms work together to build forwarding tables

Responsibilities of inter-domain routing:

① Learn which dest. reachable through neighboring AS

② Propagate reachability info to all routers in AS

Intra-domain protocols to route <u>w/in</u> AS:

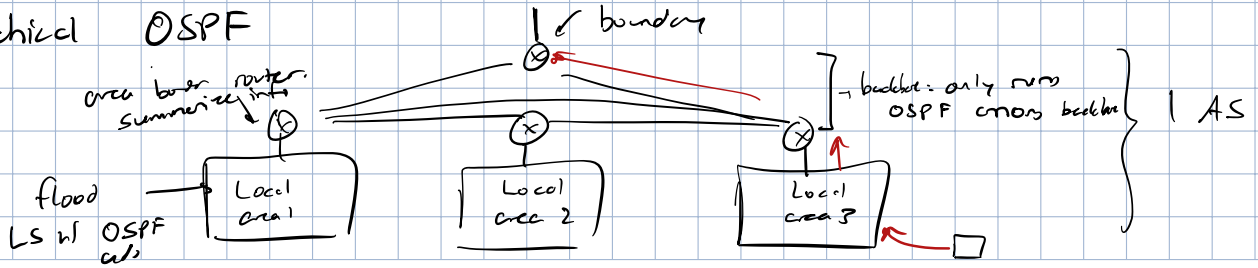
① RIP (routing info. proto.): exchange DVs within AS, not used

② EIGRP (enhanced interior gateway routing proto.): uses DV, used to be Cisco prop.

③ OSPF (open shortest path first): link-state

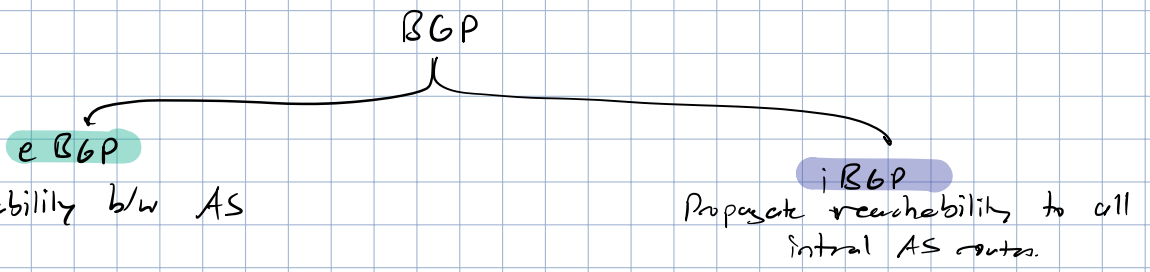
- Each router floods OSPF adverts to all routers in AS w/ costs,
- use Djikstra's to figure out best path to each next. Uses <u>IP</u>, not TCP/UDP
- Authentication → secure.

Hierarchical OSPF

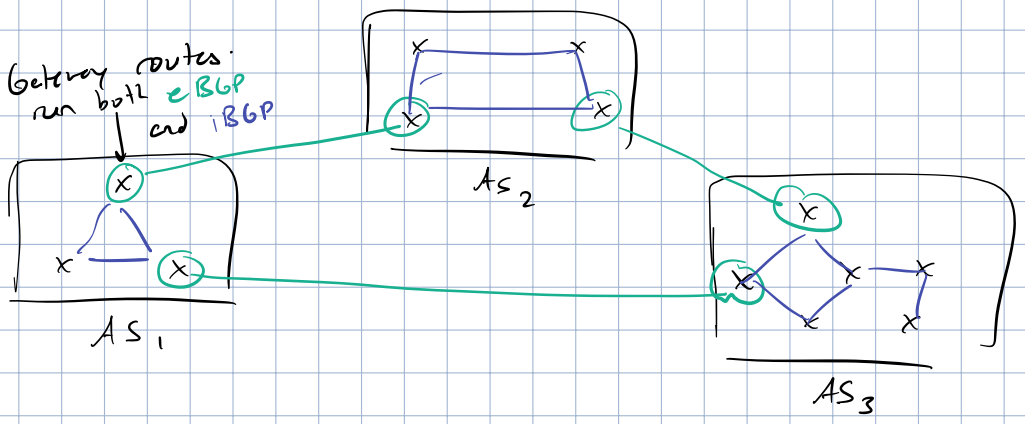


BGP

Border Gateway Protocol: standard inter-AS routing proto.

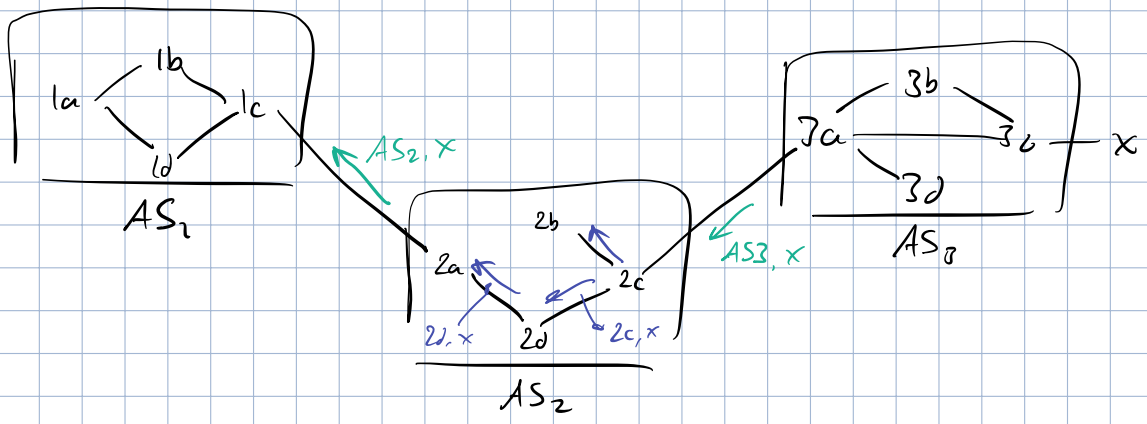


Each AS uses policies to determine best route based off reachability



When new router added to AS, BGP will announce it can reach router to neighboring AS

↳ Across semi-persistent TCP conn.



BGP msg components

Prefix
Destination advertising

Attributes
① AS-PATH: list of AS which ad has passed
② NEXT-HOP: most recent router

BGP msg types

- **OPEN**: Open TCP conn. b/w BGP peers
- **UPDATE**: Advertise new path
- **KEEPALIVE**: Keep conn. alive for faster updates. Also an ACK to OPEN
- **NOTIF.**: Report mistake / close conn.

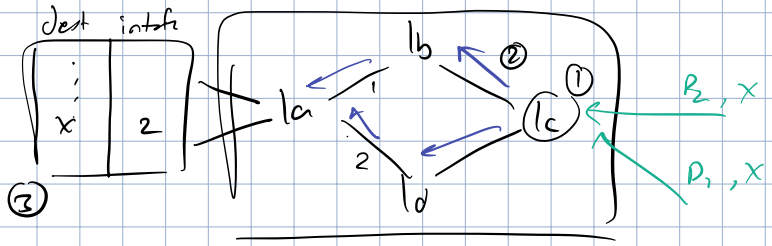
When ad received by gateway, policy in action

① Choose to reject ad (import policy)

② Choose to advertise new path to other AS (eg. don't want other providers to know)

③ Choose to keep existing route (eg. shortest AS-PATH, closest NEXT-HOP, other policy)

Connection w/ intra-AS routing:

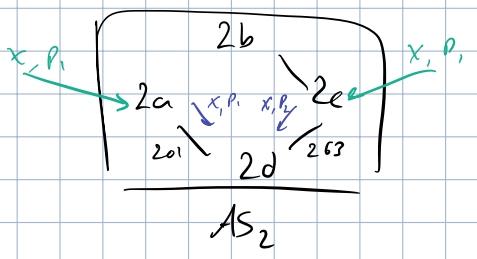


① 1c makes decision to use P_2 via eBGP

② iBGP propagates to all routers about X and to use 1c

③ OSPF/router: tells router which interface to use to get to 1c if X is dest.

What happens if internal router gets 2 iBGP msgs?



2d chooses gateway w/ least OSPF cost. Intra cost > inter-cost

Hot potato routing

Why do we need distinction b/w intra vs. inter routing?

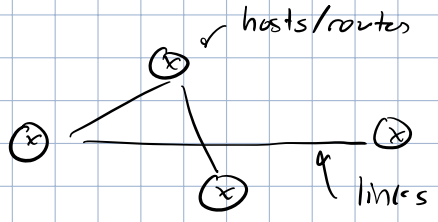
① Policy: inter-domain has more policy considerations

② Scale: hierarchical tables

③ Performance: intra focuses on perf., inter focus on policy

Link Layer

Introduction



Link layer responsible for sending package from Host A → B (adjacent)

frame

Link protocol depend on link type

Link services

Framing
- Encapsulate datagram

Link access
- Channel access esp. if shared
- MAC addr.

Reliable delivery
Used a lot for high-error links

Flow control

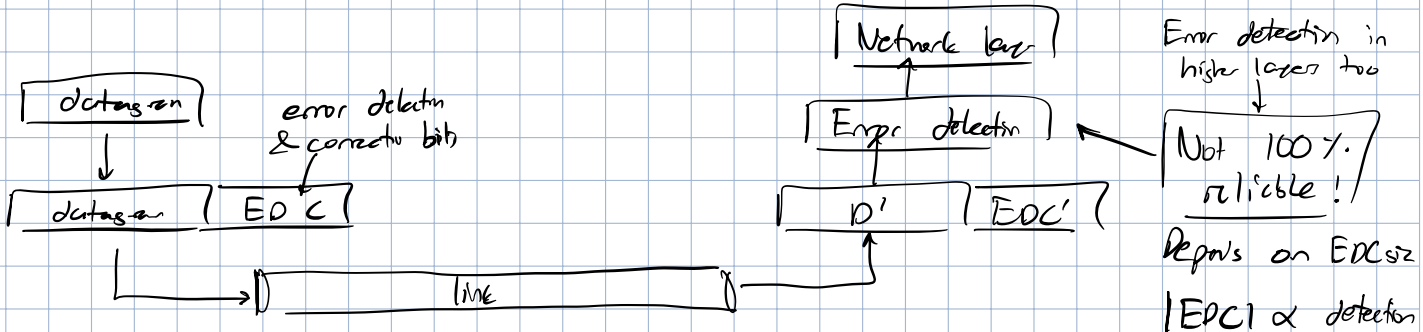
Error detection & correction

Duplex
Full / half

Implementation: network interface card (NIC) on chip in hop

↳ Combo of SW, HW & firmware

Error detection & correction



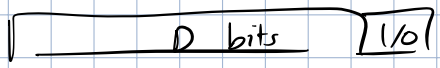
Depends on EDC size

|EDC| ∝ detection

Methods:

① Parity checking

A: Single bit:



} Total # of 1s should be even, set parity bit accordingly

Check on receiving side if parity of 1 bit is same

Con: same parity can be achieved even if error exist

↳ Eg: 0 → 1 & 1 → 0

Hard to correct! Which bit flipped?

B: 2D bit parity

Ex:// Msg: 10101 11110 01110

Sending side

① Table

G_1 [	1	0	1	0	1
G_2 [	1	1	1	1	0
G_3 [	0	1	1	1	0

② Row & col parities

						row
	1	0	1	0	1	1
	1	1	1	1	0	0
	0	1	1	1	0	1
col [	0	0	1	0	1	0

parity of parity, choose either row col

③ Add bits to msg

10101**1** 11110**0** 01110**1** 00101**0**

Add row parity bits to each group

Make row group & col parities.

Receiving side

① Reconstruct table from bits

1	0	1	0	1	1
1	1	1	1	0	0
0	1	1	1	0	1
0	0	1	0	1	0

② Reconstruct table to verify parities → error detection.

Error detection:

Orig:

1	0	1	0	1	1
1	1	1→0	1	0	0→1
0	1	1	1	0	1
0	0	1→0	0	1	0→1

Col & row error detected

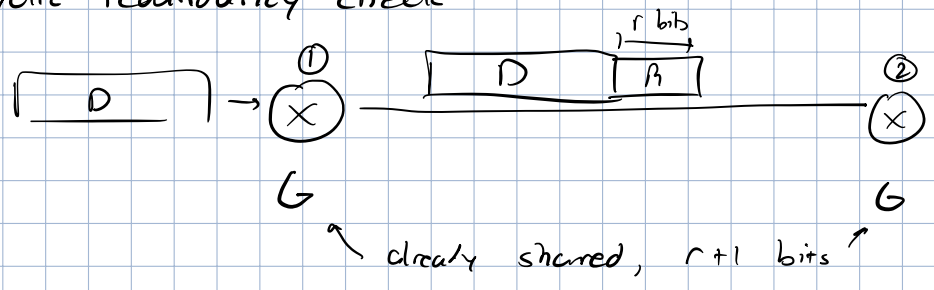
↳ Correct!

1	0	1	0	1	1
1	1	1→0	1→0	0	0
0	1	1	1	0	1
0	0	1→0	0→1	1	0

Error detected but not correctable

Can only detect up to all 3-bit errors.

② Cyclic redundancy check



① $\langle D, R \rangle = D \times 2^r$ XOR R

Choose r CRC bits R s.t. $\langle D, R \rangle \bmod (G \% 2) \equiv 0$

② Create $\langle D, R \rangle$ & check if $\langle D, R \rangle$ divisible by G mod 2. If not → error

} Detect error < r+1 bits

Ex: // $D = 10111O$. $G = 1001$. Find CRC bits

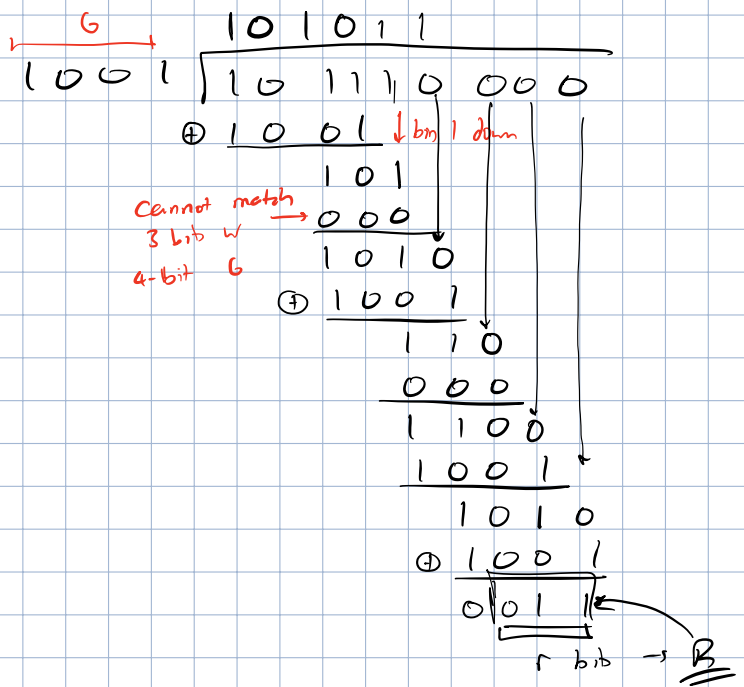
① Find r (# of bit of CRC)

G has 4 bits → $r = 3$ bit

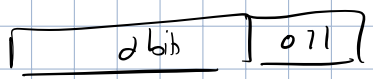
② Bit shift D by r bit

$10\,1110 \longrightarrow 10\,1110\,000$

③ 'Divide' (XOR) $D \times 2^r$ by G → remainder of r bit



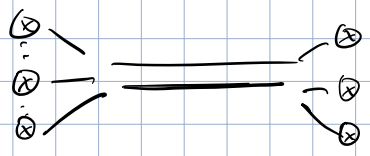
④ Put this in msg



Widely used method! Receiver will do same division of D+R by G ⇒ should be 0 if no error

Multiple access protocols

Q. If we have shared link medium, how do we share link efficiently



A: use multiple access protocol. Def.

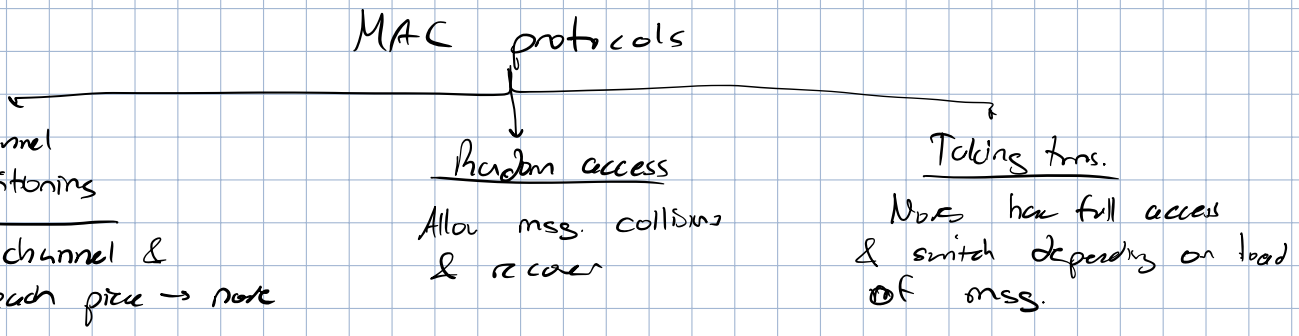
1. When should node transmit
2. Coordination via channel/link only

} distributed algo

Ideally:

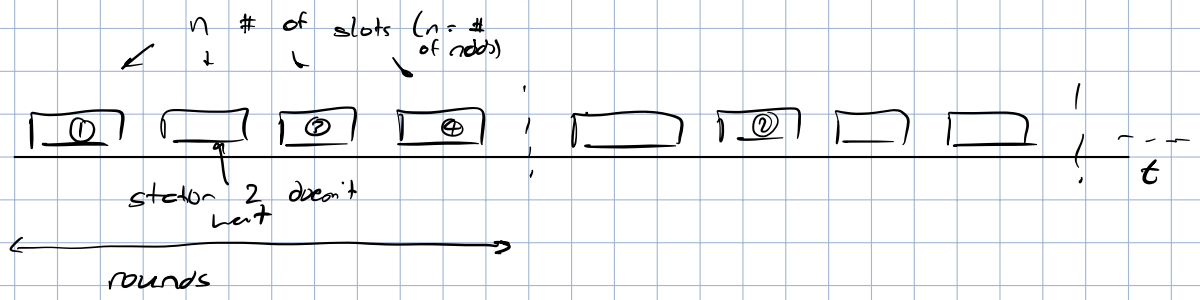
If channel has bandwidth R bps

1. If 1 node wants to transmit → transmit @ R
2. If M nodes → R/M rate per node
3. Fully decentralized (no special coord.)
4. Simple

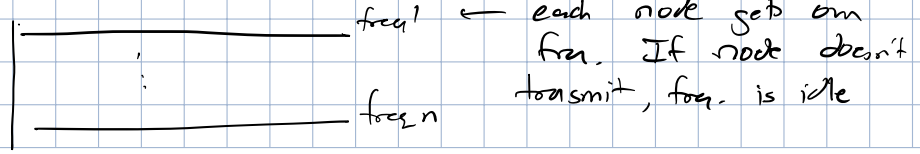


① Channel partitioning

A: TDMA (time division multiple access)



B: FDMA (freq. DMA)



Con: not efficient & not dynamic, needs some coord.

Pro: simple

② Random access protocol

Each node send @ R bits → protocol defines error detection & recovery

A: Slotted ALOHA

Assumptions:

① All frames equal size (L bits)

② Time divided into equal slots (L/R sec.)

③ Nodes synchronized & all detect collisions.

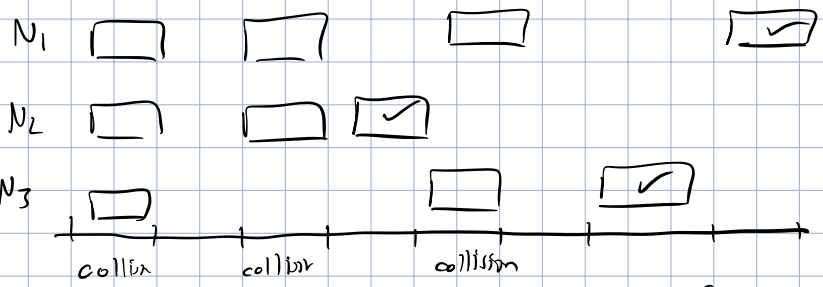
Operation:

① Node sees slot → transmits

② If no collision: send next frame @ next slot

③ If collision: node transmits frame @ next slot w/ prob p ← prob all nodes from retransmitting @ next frm → collision

Ex://



Pros:

① Continuously transmit at R bps

② Decentralized (only slots synced)

③ Simple

Cons:

① Collisions → waste slots

② Accidentally cause more collisions if cannot detect collisions fast enough

③ Synchronization required.

Efficiency: 37% (at best prob. $p^* = 1/N$ (# of nodes))

B: Pure ALOHA

No slotting → more chaos & collisions

Efficiency: 18% (at best prob $p^* = \frac{1}{2N}$)

C: CSMA

Carrier sense multiple access

CSMA

Normal:

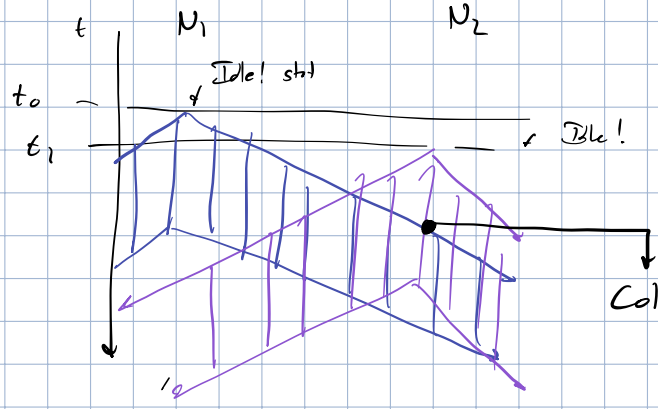
If channel idle → send
busy → wait

Collision detection

Detect CD in short time & abort transmission

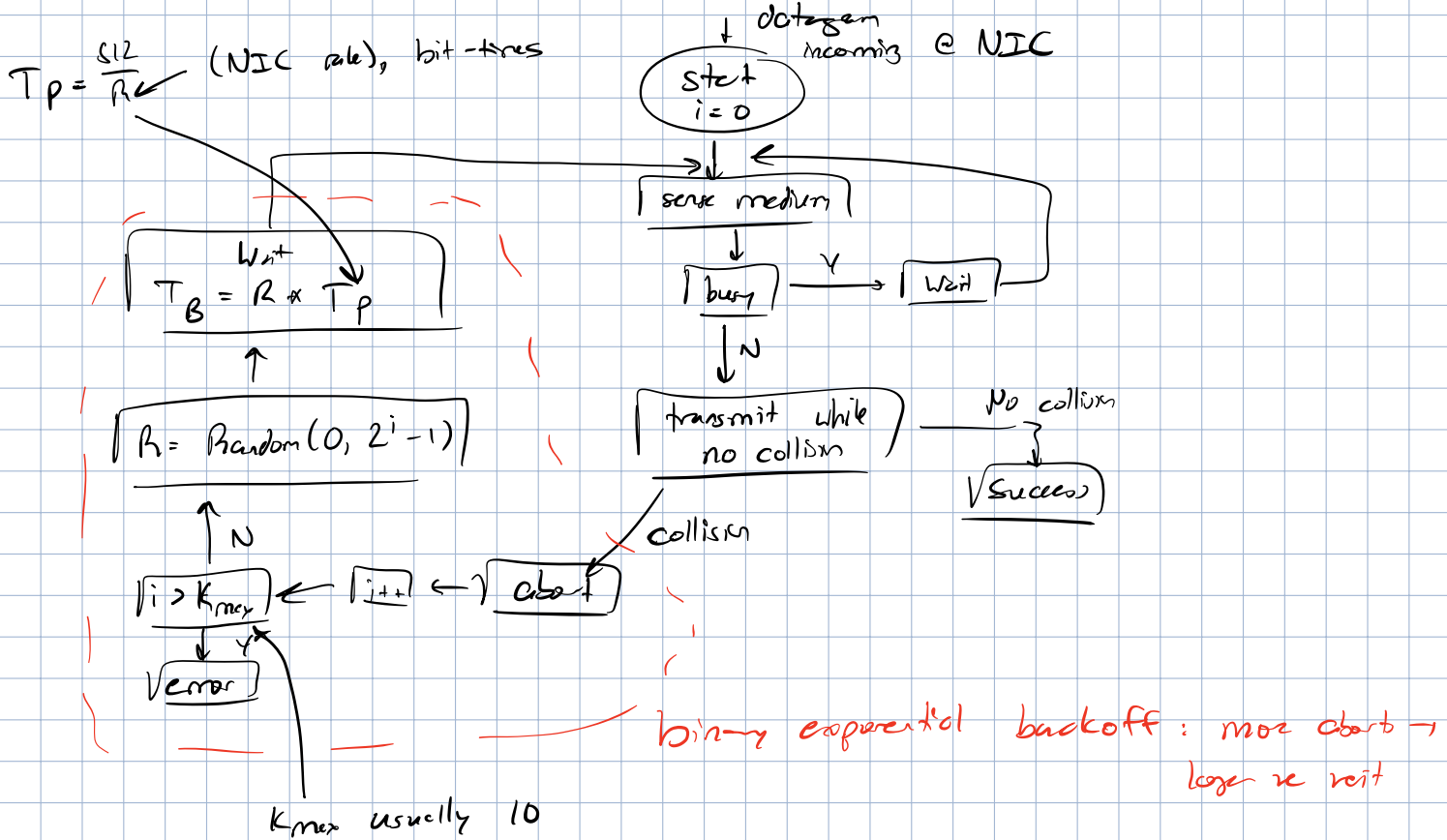
Easy in wired media, not in wireless

You can still have collisions!



Collision! If CSMA/CD, abort transmission

Ethernet CSMA/CD also:



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

t_{prop}: max delay b/w 2 nodes in LAN

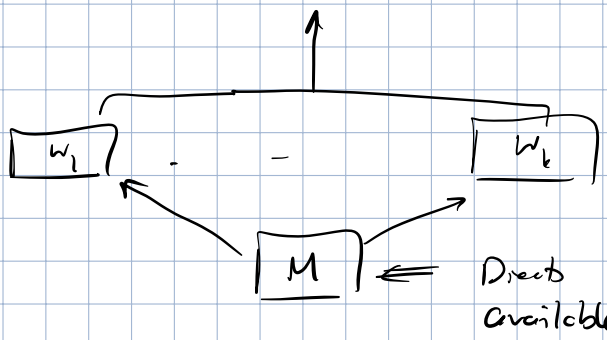
t_{trans}: time to transmit max-size frame.

If $t_{prop} \to 0$ or $t_{trans} \to \infty$ $\Rightarrow$ efficiency $\to 1$

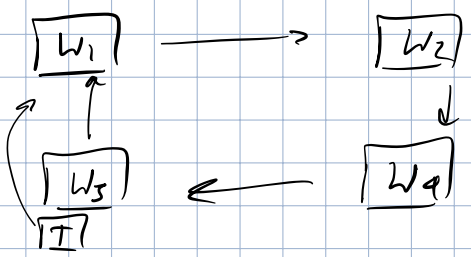
Not the fairest but good perf.

③ Taking turns

A: Polling



B: Token passing



If W_i has T → transmit

Taking turns used more for wireless (Bluetooth)

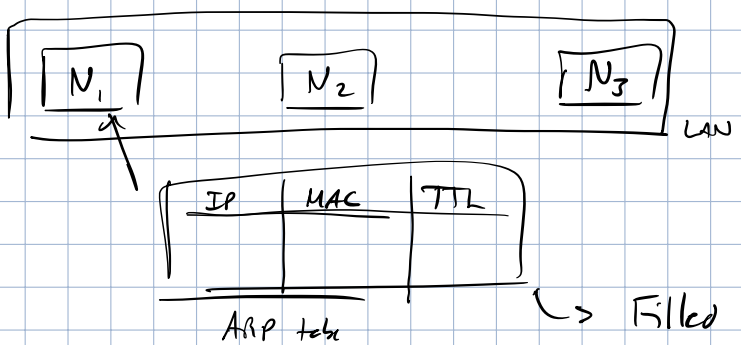
Addressing

Obj: physical address to communicate w/in adjacent nodes

Ans: MAC address (48 bits, in NIC, unique w/in subnet)

MAC addr. allocated by IEEE & portable (unlike IP addr.)

Q: How do we use MAC address in LAN?



↳ Filled in by ARP protocol

ARP protocol: N_1 wants to transmit to N_2

① N_1 broadcasts ARP query w/ N_2's IP

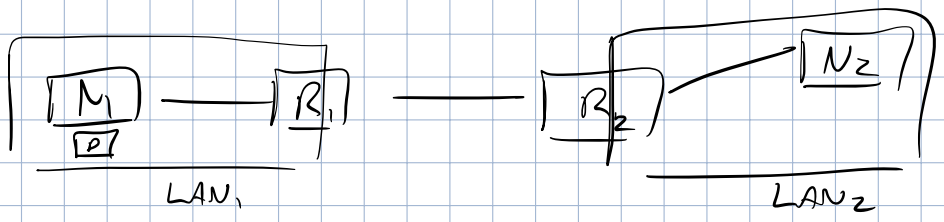
src MAC:	_ _
dest MAC:	FF .. FF
src IP:	_ _
dest IP:	IP N_2

② N_2 receives & sends ARP response

src MAC:	MAC N_2
dst MAC:	MAC N_1
src IP:	IP N_2
dst IP:	IP N_1

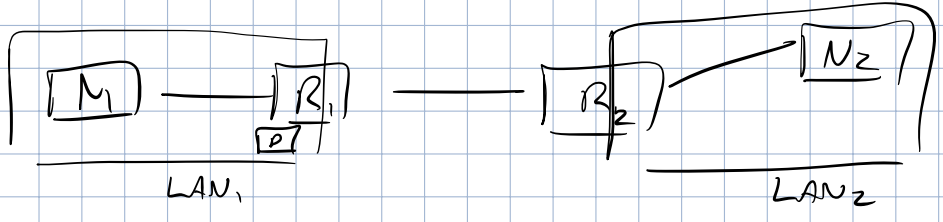
③ N_1 receives & updates ARP table

Q: What if we need to send to diff subnet?



① N_1 will set link frame w/ MAC addr. of R_1, not N_2

② Broadcast frame to LAN_1 devices; if device MAC matches, remove datagram → give to IP (only R_1 responds)



④ IP layer gets interface → link-layer gets MAC to next hop & sends

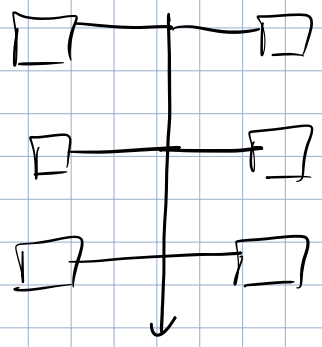
⑤ Repeat until router attached to N_2 & replies MAC w/ MAC of N_2

Ethernet

Dominant wired LAN link type (fast, cheap)

Types of topology:

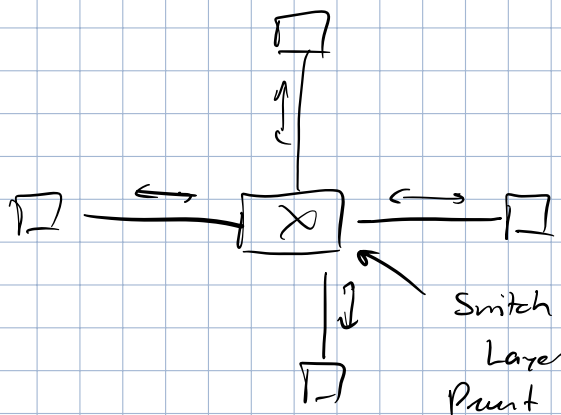
① Bus:



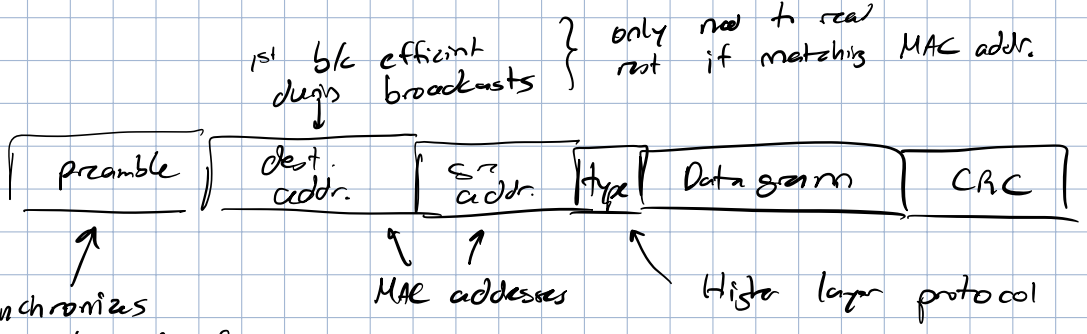
Collisions possible

Popular in 90s

② Switch



Frame structure for Ethernet



Properties:

① Connectionless: no handshaking

② Unreliable: no ACKs, no NACK (if packet lost, GG)

③ MAC protocol: CSMA/CD w/ binary/exp. backoff

Ethernet standards: diff in speed & physical layer

100 Mbps — 400 Gbps

copper / fiber

↳ Distinguish: <rate in Mbps> BASE - <medium type>

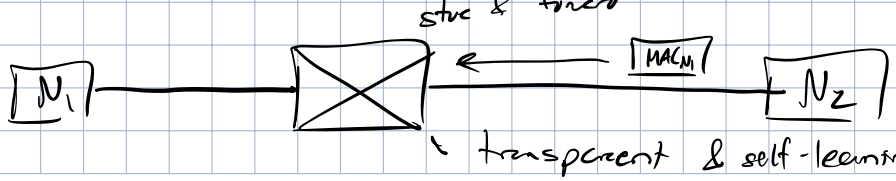
100, 40, 600

if ≥ Mbps → appx w/ 6 (406)

TX ⇒ copper, X = # of mm

else: fiber optic

Switch



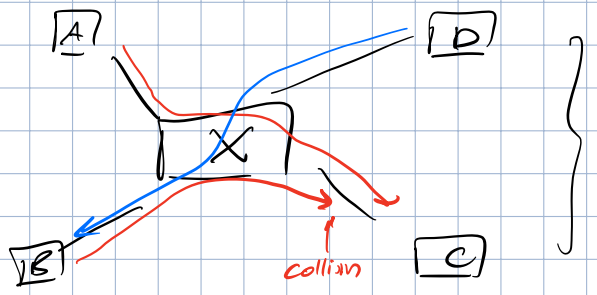
① Look @ packets MAC

② Forward to interface w/ corresponding MAC

Each host has conn. to switch, each run Ethernet protocol (CSMA/CD)

↳ Collisions can still happen if multiple nodes going on same link

↳ Each link is full duplex



Q: How does switch figure out which interface to forward packets?

A: Switch table

MAC addr.	Interface

Learning process:

① Frame comes in → enter MAC addr. & interface

② Check table if dest. MAC in switch table:

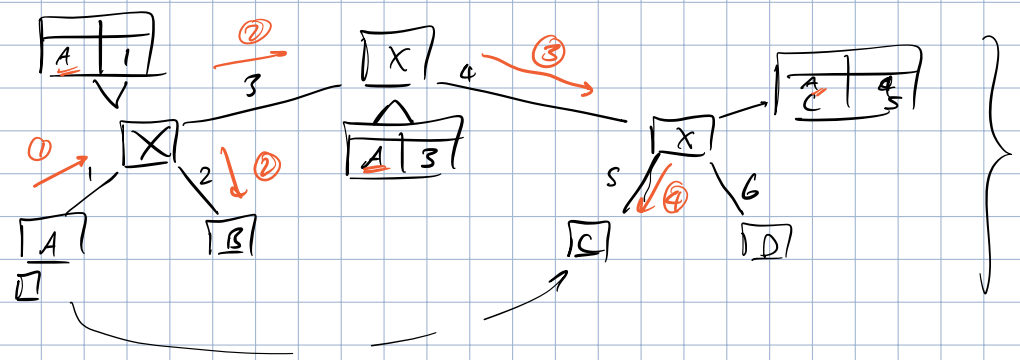
i) If match found:

A: Match is on same interface (going to where it came) ⇒ drop

B: Else: forward

ii) Else: broadcast frame to all hosts

Switches can be put in a hierarchy, works



Cycles are possible → protocols can deal w/ it

Q: Diff. b/w hubs, switches & routers

repeaters, flood network w messages

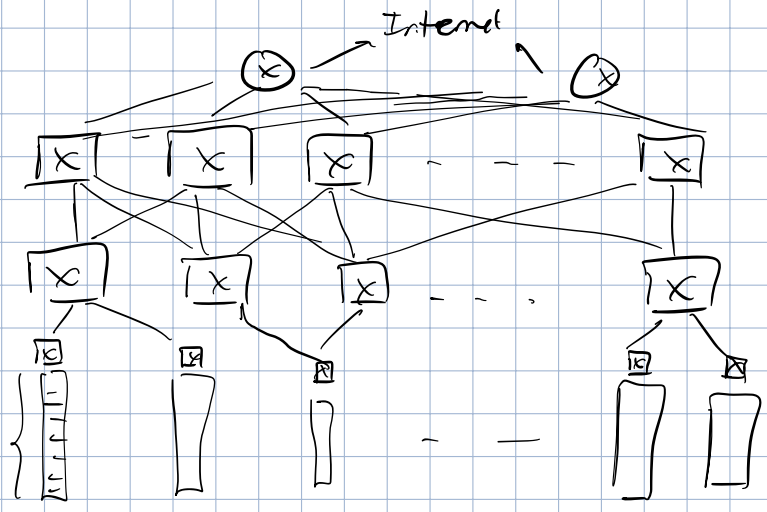
	Hubs	Switches	Routers
Layer	Physical layer	Link	Network
Plug-and-play	✓	✓	X (configs for routing algo, OSPF weights, etc.)
Collisions	✓	X	X
Security (only participants hear packets)	X	✓	✓
Optimal forwarding (always use shortest path)	X ("flooding means we could forward packet in non-optimal path")	X (cycles)	✓

only forward to dest., rarely flood

Datacenter networking

100 000+ hosts working together ⇒ multiple apps, reliable, balance load

General network struct.



} Border routers.

} Tier-1 switches
↳ Each connect to 16 T-2 switches

} Tier-2 switches
↳ Connect to 16 TOR switches

} → Top of rack switch: switch / rack

Multiple interconnections → Higher throughput
→ Reliability

Innovations:

① Load balancer routing (app. layer): directs workload to hosts → load

② Data center TCP (DCTCP):

A: Explicit congestion notif: intermediate nodes send info to routers via packet headers about congestion
↳ no wait for dropped packets

DCQCN ↑ quantized congestion notification

B: Priority flow control

③ Hop-to-hop backpressure detection & rate-limiting

④ RoCE: remote DMA over converged Ethernet
Interact w hosts as if 1 CPU → direct memory access

⑤ Software-defined networking > routing also