DES331 : Computer Network Architectures & Protocols

Midterm Summary

Introduction (L1)

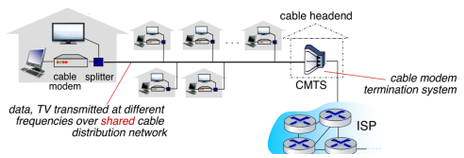
basic view
- internet = network of networks
- protocols = set of communication rules → define format & order of message (sent/received) & actions taken on transmission

service view
- provide programming interface to apps. (allow to send & receive / provide service options)

Internet Structure
- Network edge → clients & servers.
- Access Network → Wired/Wireless communication link
- Network Core → ISP → forward packets from one router to next.

Access Network

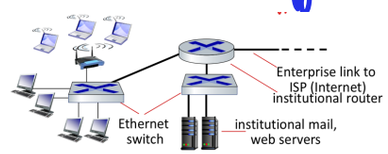
* wired access networks/residential access
- cable-based (netwk of cable, fiber attaches home to ISP Router



- Digital Subscriber Line (DSL)
  use existing telephone line to central office DSLAM

* wireless access network.
- access point → base station to connects end system to router (( )) or (( ))
  - ↳ WLANs: Wireless LAN → within Building, 802.1x standard.
  - ↳ WWANs: Wide Area Cellular Network → mobile network(3G/4G/5G)

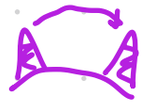
* enterprise network
→ bigger → more Bandwidth



mixed of wired & wireless

* data center → High bandwidth link
Connect servers together & internet.

Links
- Guided Media → Cable!
  - signal propagates in solid media eg. cable
  - → TP: Twisted Pair
  - → Coax: Coaxial Cable
  - → Fiber Optic Cable
- Unguided Media → Wireless
  - signal propagates freely e.g. radio wave
  - → Wifi: WLANs
  - → Mobile: 4G/Cellular
  - → Bluetooth:
  - → Terrestrial Microwave
  - → Satellite



function of Network core

*① Forwarding
- → local action
- move incoming packets to appropriate router output.
- → "forwarding table" in each router

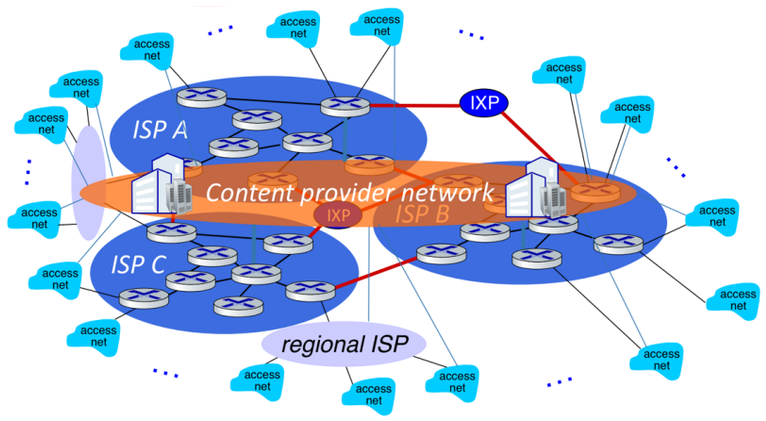
*② Routing
- → global action
- determine paths taken by packets
- → Routing Algorithms

Packet Switching
- Store & Forward: Arrive entirely before transmitted to another link
- Packet transmission Delay: $\frac{L}{R}$ seconds ; L bits per packet, R bps
- Queueing: arrives too much in router, send too slow.
- * Packet Loss: arrival rate > transmission rate for some time,
  ① queue up ② drop if memory buffers filsup.

* Depends on situation
- → packet switching great for bursty data
- → can cause exressive congestion

Alternatively: Circuit Switching
- ↳ end-end resources allocated to between source & destination
- * FDM: multiple user at same time divides into smaller band
- * TDM: more data at same time divides into small time slots

Internet Structure:



tier 1 ISP: huge to connect different part of the world
(CDN: is put on tier 1 country) ↓
national & international coverage

Performance:
- Real Delay
  * traceroute: delay measurement from source to router along the path
- Throughput → "rate" bits sent from sender to receiver
  - instantaneous: at given point in time
  - average: over longer period of time.
  * bottleneck: link on end-end path constrains end-end throughput

Security:

Packet Interception
* Sniffing
  - broadcast media.
  - reads/records all packets passing by

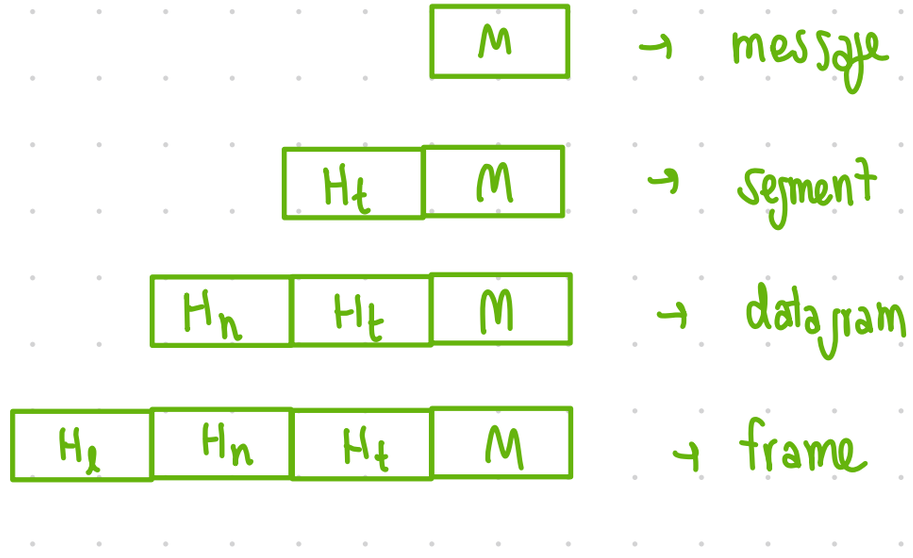
fake identity
* Spoofing
  - injection of packet with false source address

* DDoS
  make resources unavailable to traffic
  by overwhelming resource with massive traffic

* Defense Mechanism
  - Authentication.
  - Confidentiality → encryption
  - Integrity Check
  - Access Restriction
  - Firewall
    → specialized middleboxes
    - detect & react to DDoS attack

Protocol Layer

① Application Layer
→ support network app./exchange message
② Transport Layer
→ process-to-process data transfer
③ Network Access Layer
→ routing datagrams
④ Data Link Layer
→ data transfer b/w neighboring network
⑤ Physical Layer
→ wire/medium



Application Layer (L2)

→ Creating network app.
- Write programs that run on end systems
- Communicate over network

Processes Communication

- Process : Program runs in host.
- process in different hosts communicate by exchange messages.
  - Client : initiates communication
  - server : waits to be connected.

Sockets : process sends/receives message to/from sockets

Client - Server Paradigm

→ Server - always on/fixed IP/often in data center
→ Client → communicate w/ server / Dynamic IP /
not communicate directly w/ each other.

Peer-to-peer architecture

→ no server → Direct communicate
→ request service from one another

Addressing.

identifier : includes both IP address & Port number associated w/ process

ex: HTTP : port 80
HTTPS : port 443
mail server : 25

Defines
- type of message: request, response
- syntax: fields of information
- semantics: meaning of information in fields
- rules: how & when to send & respond
- open protocols: what kinds of header

Transport service does an app need?
- Data Integrity: reliability
- Throughput
- Timing: min. delay
- Security

Transport Protocol Services

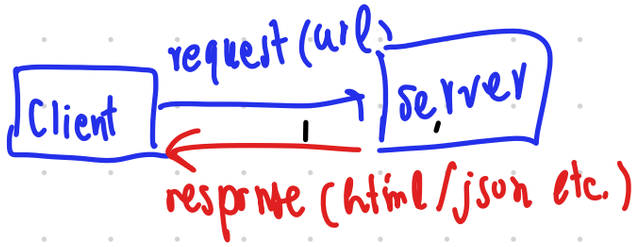
* ① TCP Services.
  - reliable Transport
    - require handshake
  - flow control
  - congestion control
  - connection-oriented

↳ Securing TCP?
- Vanila Socket: no encryption
- TLS: Transport Layer Security
  - provides encrypted TCP connection
  - data integrity, end-point authentication

② UDP: Services
- unreliable data transfer

[speed over accuracy]

Web & HTTP



HTTP: Hypertext transfer protocol.
- web's application layer protocol.

Client = Host's web browser Server = web server.

HTTP: use TCP : port 80.
- accept connection (TCP) from Client.
- ✿ Stateless: server maintain no information about client request.

↳ connection type:

① Non-persistance HTTP
Open TCP → sent 1 Object (max) → Close TCP.

② Persistent HTTP
open TCP → multiple object sent → Close TCP over single connection

RTT: Round Trip time: client → server → client.

Response time:

2 RTT + file transmission time ←

RTT + transmission time

Request message

2 types : Request / Response
↳ from server → status code

- human-readable format (GET/POST etc.)
  - URL? _ = _
  - identity body of POST message

- → 200 OK
- → 404 Not found
- → 400 Bad Request
- → 505 Not Support.

maintain user /server state "COOKIES"

- HTTP is stateless
- all request are independent
- no need for client/server to recover.

4 components → ① Header line: Response message
② next HTTP Request
③ kept on user's host, manged by user's browser
④ back-end database at website

Used for → authorization / shopping carts etc.

How to keep states? → at endpoints or in message

Web Cache

satisfy client request w/o involvig orijin server.

Web caches → Proxy server.
act as both client & server

- if object in cache:
  - return object to client
- if object not in cache:
  - forward request to server.
  - received and store object
  - send object to client

Why? reduce response time
reduce traffic in link

Conditional GET { don't send object if cache has up-to-date cached version

HTTP/2 {
- decrease delay in multi-object HTTP request.
- increase flexibility at server in sending object to client.

objects divided into frames, transmission is interleaved.

HTTP/2 over single TCP connections ⇒ recovery still stalls all object transmission
no security over vanila connection

HTTP/3 { add security, error- & congestion control per object over UDP

Email

3 Major Components

1 user agent
- mail reader
- messages stored on server

2 mail server
- mailbox
- message queue of outgoing

3 SMTP: Simple mail transfer protocol
b/w mail server & client

Use TCP to reliably transfer email message from client / port 25

3 phases of transfer ① Handshake → ② Transfer ③ Closure

In comparison w/ HTTP: client pull but SMTP client push
SMTP is persistent connection, require to use 7-bit ASCII
multiple obj sent in multipart message

Retrieving email: mail access protocols

IMAP (Internet mail access protocol) → provide retrival/delete

HTTP provides web interact on top of SMTP (send) & IMAP/POP3 (retrieve)

DNS

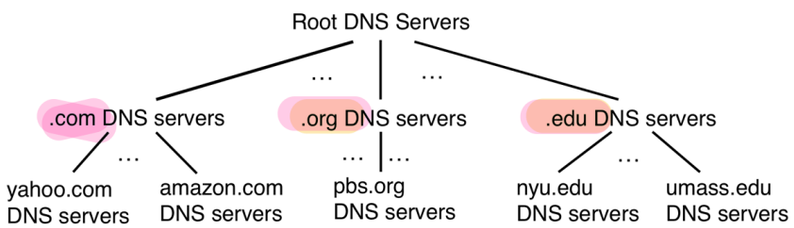
Domain name System

from human understandable address
map to IP address → How? [using DNS!]

- *distributed database* implemented in hierarchy of many *name servers*
  → core internet function implement as application layer protocol

- Services
  address translation & aliasing etc.

DNS → distributed, hierarchical database



Root

Top Level Domain

Authoritative

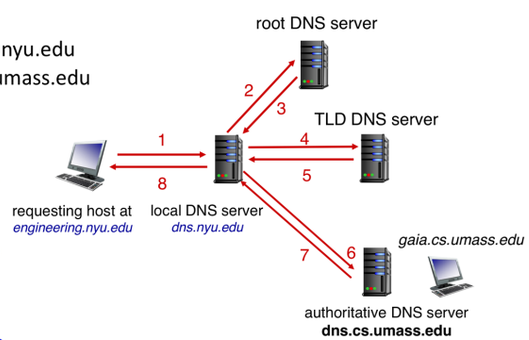
- Top-level Domain
  → responsible .com, .edu, .org, .net & country domain

- authoritative DNS server
  → organization's own DNS server.

DNS name resolution: iterated query

Example: host at engineering.nyu.edu wants IP address for gaia.cs.umass.edu

Iterated query:
- contacted server replies with name of server to contact
- "I don't know this name, but ask this server"

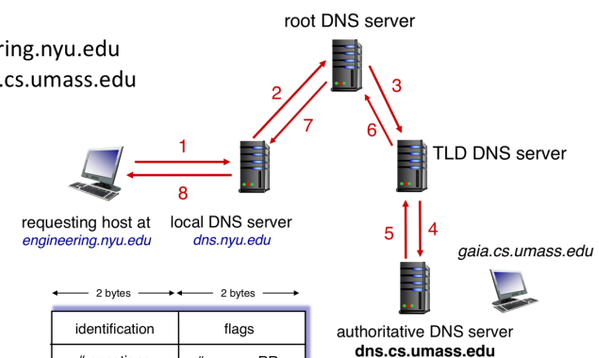


→ stored as RR: resource records
(name, value, type, ttl)
↳ A/NS/CNAME/MX

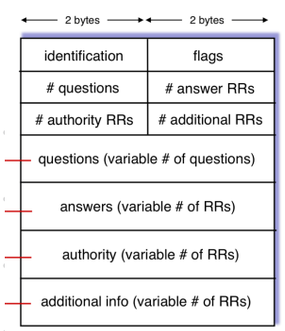
DNS name resolution: recursive query

Example: host at engineering.nyu.edu wants IP address for gaia.cs.umass.edu

Recursive query:
- puts burden of name resolution on contacted name server
- heavy load at upper levels of hierarchy?



format



P2P

arbitrary end systems directly communicate

→ server must upload at least one copy

→ client (each) must download file copy

→ Clients : as aggregate must download NF bits

time to distribute F to N clients using P2P approach

$$D_{P2P} \geq max\{F/u_s, F/d_{min}, NF/(u_s + \Sigma u_i)\}$$

increases linearly in N ...
... but so does this, as each peer brings service capacity

- bit Torrent
  - → divided into 256 Kb Chunks
  - → peers in torrent (group of peers) send/receive chunks / tracker tracks peers participating in torrent
  - → requesting chunks
    - → any time, different peers have different subset of chunks
    - → request for list of chunks w/ peers that they have
    - → request for missing chunks, rarest first
  - → sending chunks (tit-for-tat)
    - → send chunks to peers

CDN

Streaming video → traffic → major consumer of internet bandwidth

→ distributed, application level infrastructure

→ continuous play out constraints → compensate for network delay

→ playout timing must match original one. → "Client-side buffer"

DASH: Dynamic Adaptive Streaming over HTTP.

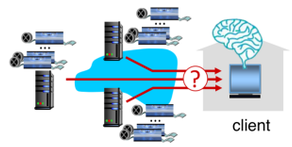
server:

- divides video file into multiple chunks
- each chunk encoded at multiple different rates
- different rate encodings stored in different files
- files replicated in various CDN nodes
- *manifest file:* provides URLs for different chunks

client:

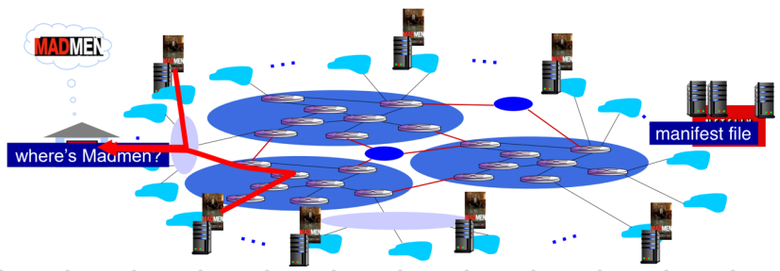
- periodically estimates server-to-client bandwidth
- consulting manifest, requests one chunk at a time

- *"intelligence"* at client: client determines
  - *when* to request chunk (so that buffer starvation, or overflow does not occur)
  - *what encoding rate* to request (higher quality when more bandwidth available)
  - *where* to request chunk (can request from URL server that is "close" to client or has high available bandwidth)



Streaming video = encoding + DASH + playout buffering

- CDN: stores copies of content (e.g. MADMEN) at CDN nodes
  - subscriber requests content, service provider returns manifest
  - using manifest, client retrieves content at highest supportable rate
  - may choose different rate or copy if network path congested



Transport Layer (L3)

Transport Layer Services
- logical communication b/w processes on different host.
- actions → sender : break message into segments
  → receiver ; reassemble segment into message
- 2 Transport protocols: TCP / UDP ⟶ unreliable, unordered
  - ↳ reliable / in-order
  - ↳ congestion + flow control
  - ↳ require handshake

Transport Layer	Network Layer
Logical communication between process (application)	Logical communication between host
Exists only in hosts	Exists in hosts and routers
Ignores network	Routes data through network
Port # used for routing in destination computer	IP addresses used for routing in network

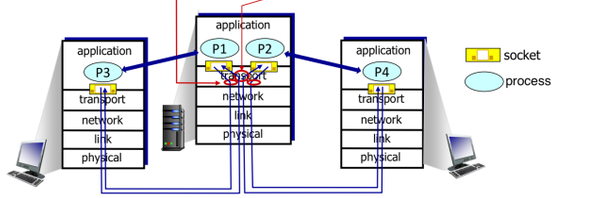
⊗ no delay guarantee + bandwidth guarantee

Mux / Demux

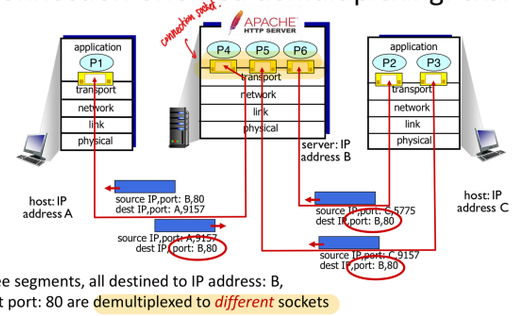
Multiplexing/demultiplexing

multiplexing at sender: handle data from multiple sockets, add transport header (later used for demultiplexing)

demultiplexing at receiver: use header info to deliver received segments to correct socket



Connection-oriented demultiplexing: example



Three segments, all destined to IP address: B, dest port: 80 are demultiplexed to *different* sockets

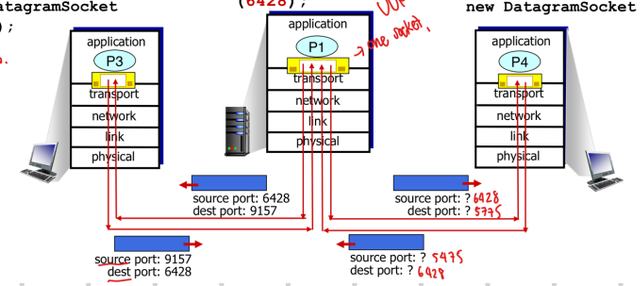
Connectionless demultiplexing: an example

You have to know the port number of the other side opened.

```
DatagramSocket mySocket2 = new DatagramSocket (9157);
DatagramSocket serverSocket = new DatagramSocket (6428);
DatagramSocket mySocket1 = new DatagramSocket (5775);
```

Port no.

UDP: one socket.



UDP: demux using only port number

TCP: demux using source IP/Port dest IP/Port

UDP

"no frills / bare bone" internet protocol.

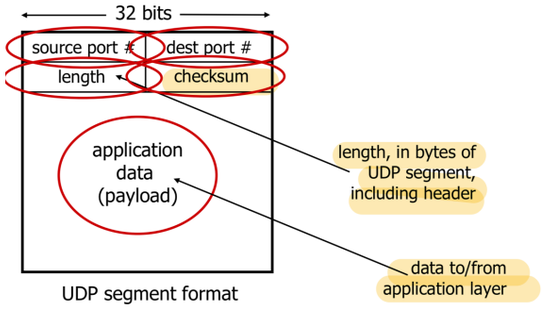
→ speed over accuracy / small header / simple

→ NO HANDSHAKE!

- if reliable transfer needed over UDP (e.g., HTTP/3):
  - add needed reliability at application layer
  - add congestion control at application layer

Check sum ⇒ detect error.

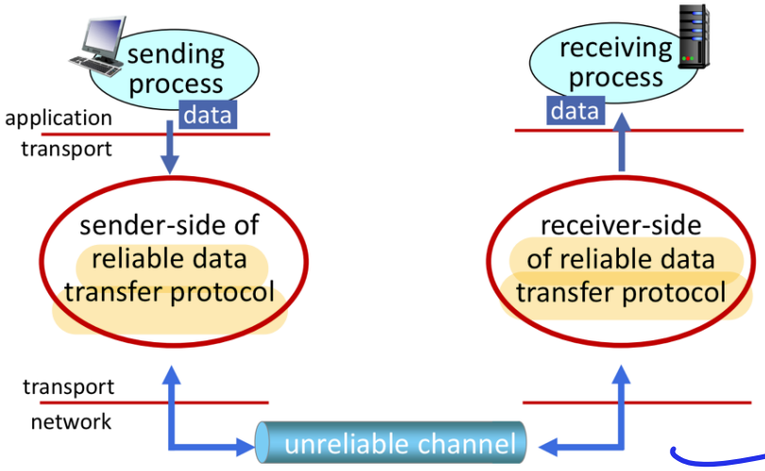
UDP segment header



Summary: UDP

- "no frills" protocol:
  - segments may be lost, delivered out of order
  - best effort service: "send and hope for the best"
- UDP has its plusses:
  - no setup/handshaking needed (no RTT incurred)
  - can function when network service is compromised
  - helps with reliability (checksum)
- build additional functionality on top of UDP in application layer (e.g., HTTP/3)

Principle of reliable data transfer



→ complexity depends on characteristic of unreliable channel!

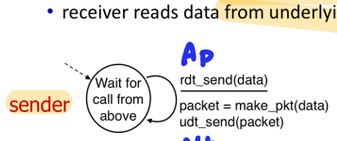
reliable service *implementation*

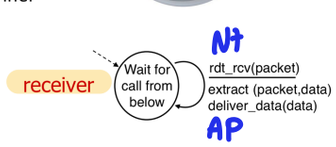
→ sender doesn't know the state of others
→ communicate!

rdt1.0: reliable transfer over a reliable channel

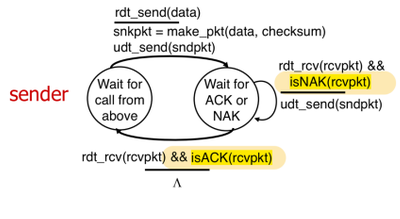
- underlying channel perfectly reliable
  - no bit errors
  - no loss of packets
- *separate* FSMs for sender, receiver:
  - sender sends data into underlying channel
  - receiver reads data from underlying channel

easy

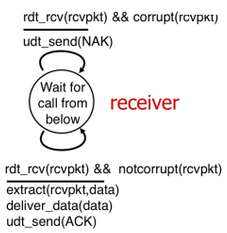




rdt2.0: FSM specification



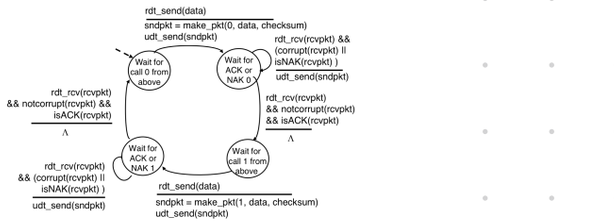
Note: "state" of receiver (did the receiver get my message correctly?) isn't known to sender unless somehow communicated from receiver to sender
- that's why we need a protocol!



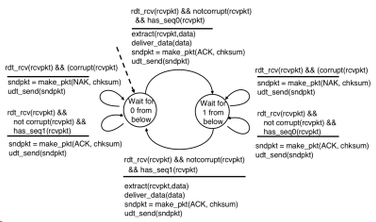
fixed bit error

ACK/NAK corrupt?
Duplicates

rdt2.1: sender, handling garbled ACK/NAKs



rdt2.1: receiver, handling garbled ACK/NAKs



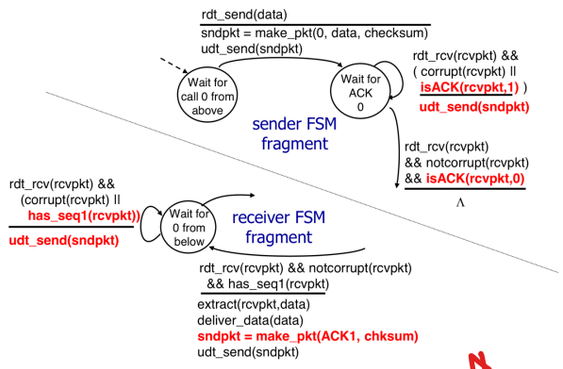
rdt2.2: a NAK-free protocol

- same functionality as rdt2.1, using ACKs only
- instead of NAK, receiver sends ACK for last pkt received OK
  - receiver must *explicitly* include seq # of pkt being ACKed
- duplicate ACK at sender results in same action as NAK: *retransmit current pkt*

Channel error/loss?

As we will see, TCP uses this approach to be NAK-free

rdt2.2: sender, receiver fragments



rdt2.1: discussion

sender:
- seq # added to pkt
- two seq. #s (0,1) will suffice. Why?
- must check if received ACK/NAK corrupted
- twice as many states
  - state must "remember" whether "expected" pkt should have seq # of 0 or 1

receiver:
- must check if received packet is duplicate
  - state indicates whether 0 or 1 is expected pkt seq #
- note: receiver can *not* know if its last ACK/NAK received OK at sender

use only ACKs

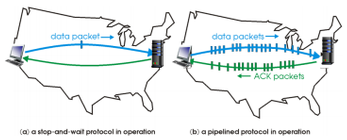
performance too low!

rdt3.0: pipelined protocols operation

pipelining: sender allows multiple, "in-flight", yet-to-be-acknowledged packets

- range of sequence numbers must be increased
- buffering at sender and/or receiver

→ increase utilization

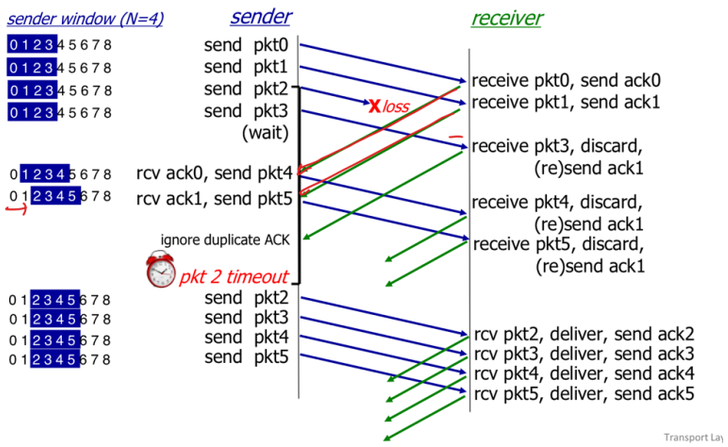


Go-Back-N: sender

- sender: "window" of up to N, consecutive transmitted but unACKed pkts
  - k-bit seq # in pkt header
- *cumulative ACK:* ACK(*n*): ACKs all packets up to, including seq # *n*
  - on receiving ACK(*n*): move window forward to begin at *n+1*
- timer for oldest in-flight packet
- *timeout(n):* retransmit packet n and all higher seq # packets in window

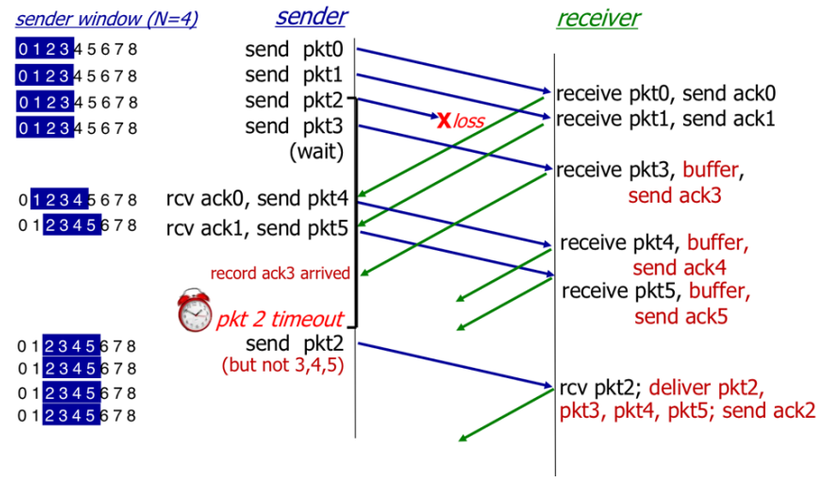
Go-Back-N: receiver

- ACK-only: always send ACK for correctly-received packet so far, with highest *in-order* seq #
  - may generate duplicate ACKs
  - need only remember `rcv_base`
- on receipt of out-of-order packet:
  - can discard (don't buffer) or buffer: an implementation decision
  - re-ACK pkt with highest in-order seq #



Selective repeat

- receiver *individually* acknowledges all correctly received packets
  - buffers packets, as needed, for eventual in-order delivery to upper layer
- sender times-out/retransmits individually for unACKed packets
  - sender maintains timer for each unACKed pkt
- sender window
  - *N* consecutive seq #s
  - limits seq #s of sent, unACKed packets



TCP

- point-to-point:
  - one sender, one receiver
- reliable, in-order *byte steam:*
  - no "message boundaries"
- full duplex data:
  - bi-directional data flow in same connection
  - MSS: maximum segment size
- cumulative ACKs
- pipelining:
  - TCP congestion and flow control set window size
- connection-oriented:
  - handshaking (exchange of control messages) initializes sender, receiver state before data exchange
- flow controlled:
  - sender will not overwhelm receiver

→ flow control

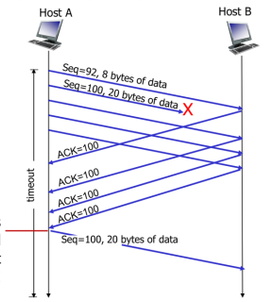
TCP fast retransmit

TCP fast retransmit

if sender receives 3 additional ACKs for same data ("triple duplicate ACKs"), resend unACKed segment with smallest seq #

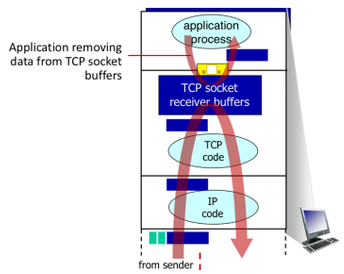
- likely that unACKed segment lost, so don't wait for timeout

Receipt of three duplicate ACKs indicates 3 segments received after a missing segment – lost segment is likely. So retransmit!



flow control

receiver controls sender, so sender won't overflow receiver's buffer by transmitting too much, too fast



TCP Sender (simplified)

event: data received from application

- create segment with seq #
- seq # is byte-stream number of first data byte in segment
- start timer if not already running
  - think of timer as for oldest unACKed segment
  - expiration interval: `TimeOutInterval`

event: timeout

- retransmit segment that caused timeout
- restart timer

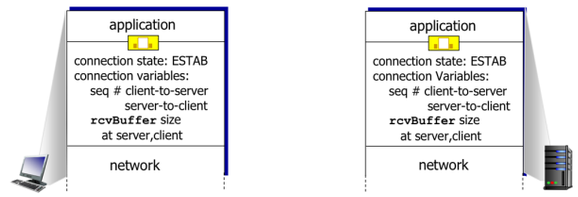
event: ACK received

- if ACK acknowledges previously unACKed segments
  - update what is known to be ACKed
  - start timer if there are still unACKed segments

HANDSHAKE TCP connection management

→ before exchanging data, sender/receiver "handshake":

- agree to establish connection (each knowing the other willing to establish connection)
- agree on connection parameters (e.g., starting seq #s)



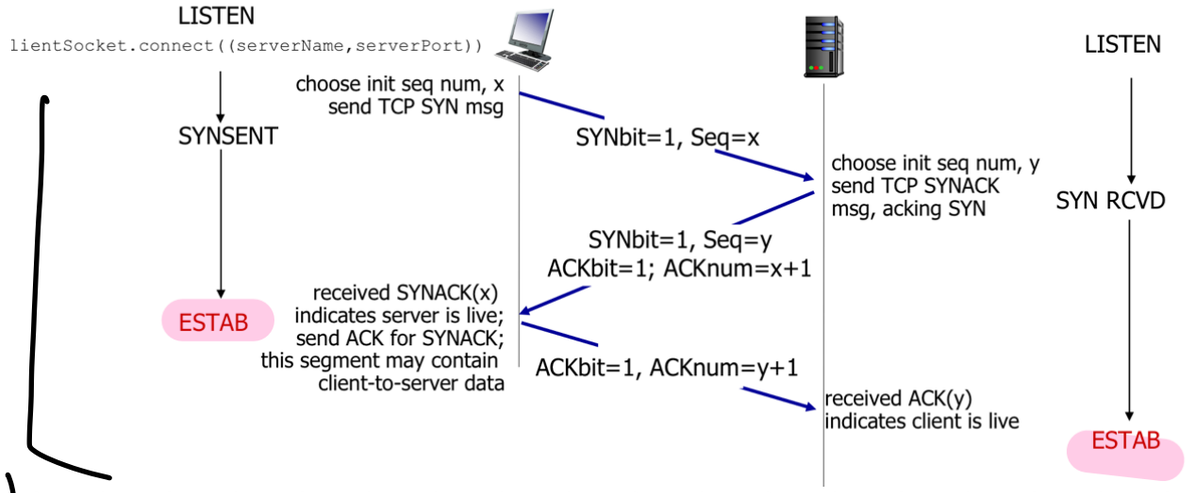

```
Socket clientSocket =
  newSocket("hostname","port number");
```

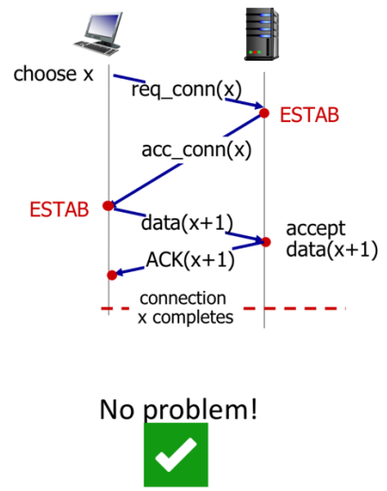
```
Socket connectionSocket =
  welcomeSocket.accept();
```

TCP 3-way handshake

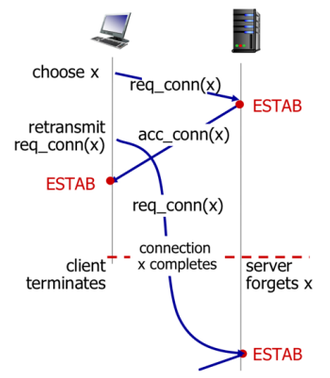
LISTEN → SYCN → ESTAB

b/c 2 ways handshake doesn't always work.





No problem!



Problem: half open connection! (no client)

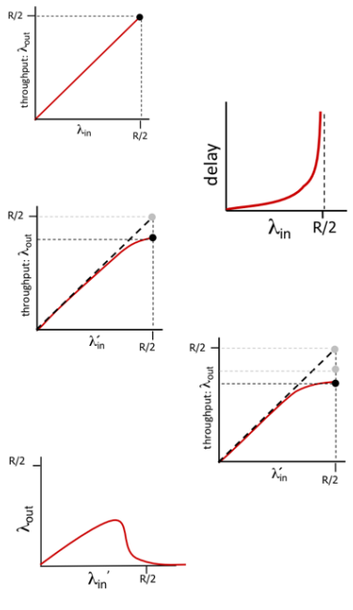
Principle Congestion Control

congestion control: too many sender, too fast

flow control: one sender too fast for one receiver.

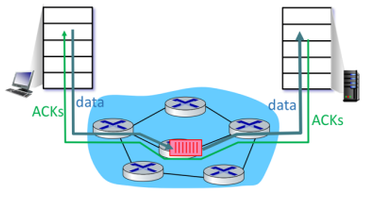
Causes/costs of congestion: insights

- throughput can never exceed capacity
- delay increases as capacity approached
- loss/retransmission decreases effective throughput
- un-needed duplicates further decreases effective throughput
- upstream transmission capacity / buffering wasted for packets lost downstream



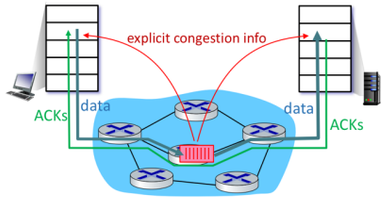
End-end congestion control:

- no explicit feedback from network
- congestion *inferred* from observed loss, delay
- approach taken by TCP



Network-assisted congestion control:

- routers provide *direct* feedback to sending/receiving hosts with flows passing through congested router
- may indicate congestion level or explicitly set sending rate
- TCP ECN, ATM, DECbit protocols



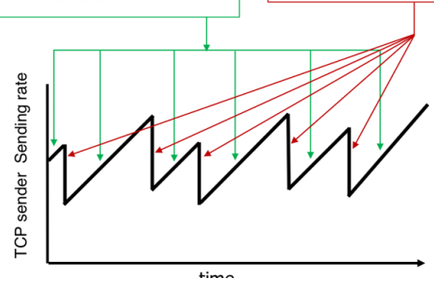
TCP Congestion Control.

TCP congestion control: AIMD

- *approach:* senders can increase sending rate until packet loss (congestion) occurs, then decrease sending rate on loss event

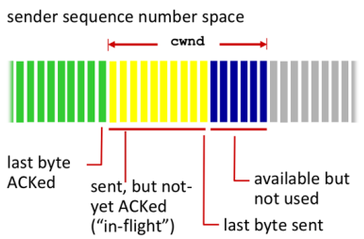
Additive Increase
increase sending rate by 1 maximum segment size every RTT until loss detected

Multiplicative Decrease
cut sending rate in half at each loss event



AIMD sawtooth behavior: *probing* for bandwidth

TCP congestion control: details



TCP sending behavior:

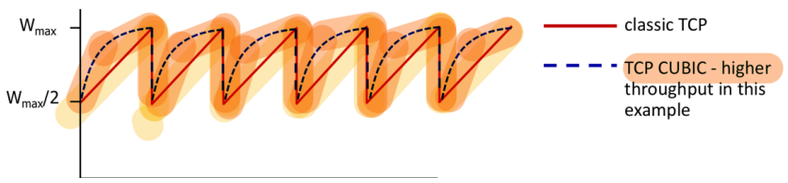
- *roughly:* send `cwnd` bytes, wait RTT for ACKS, then send more bytes

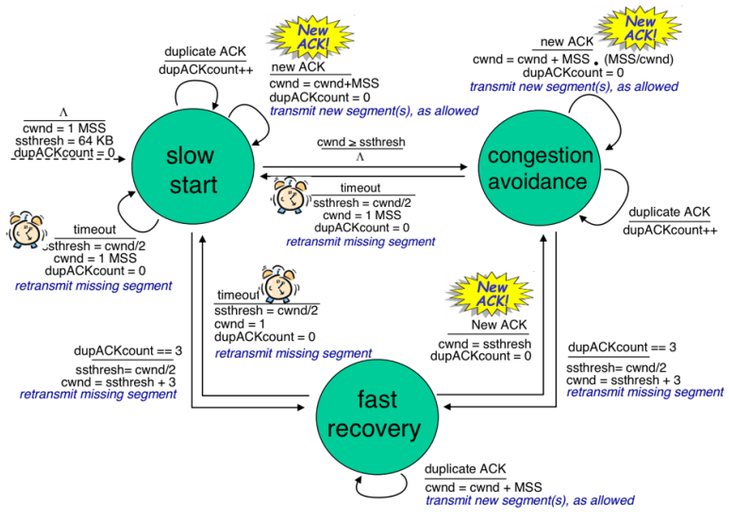
$$\text{TCP rate} \approx \frac{\texttt{cwnd}}{\text{RTT}} \text{ bytes/sec}$$

- TCP sender limits transmission: `LastByteSent- LastByteAcked ≤ cwnd`
- `cwnd` is dynamically adjusted in response to observed network congestion (implementing TCP congestion control)

TCP CUBIC

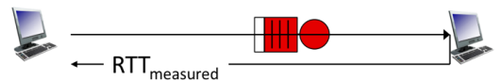
- Is there a better way than AIMD to "probe" for usable bandwidth?
- Insight/intuition:
  - W_{max}: sending rate at which congestion loss was detected
  - congestion state of bottleneck link probably (?) hasn't changed much
  - after cutting rate/window in half on loss, initially ramp to to W_{max} *faster*, but then approach W_{max} more *slowly*





Delay-based TCP congestion control

Keeping sender-to-receiver pipe "just full enough, but no fuller": keep bottleneck link busy transmitting, but avoid high delays/buffering



$$\text{measured throughput} = \frac{\text{\# bytes sent in last RTT interval}}{\text{RTT}_{measured}}$$

Delay-based approach:

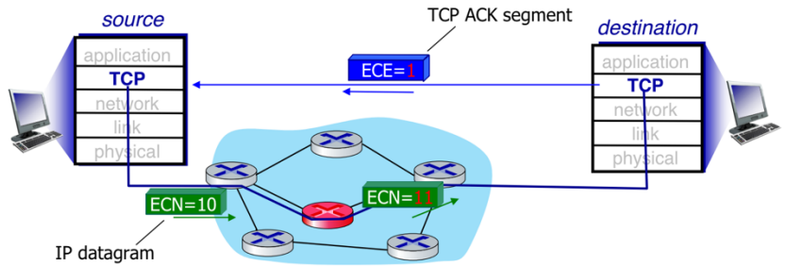
- RTT_{min} - minimum observed RTT (uncongested path)
- uncongested throughput with congestion window `cwnd` is cwnd/RTT_{min}

if measured throughput "very close" to uncongested throughput
  increase `cwnd` linearly /* since path not congested */
else if measured throughput "far below" uncongested throughout
  decrease `cwnd` linearly /* since path is congested */

Explicit congestion notification (ECN)

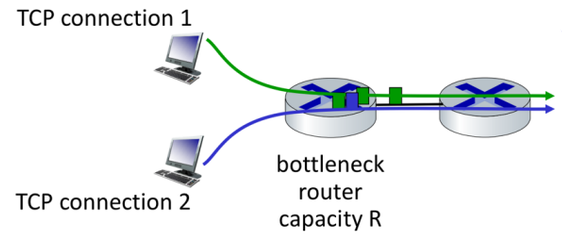
TCP deployments often implement *network-assisted* congestion control:

- two bits in IP header (ToS field) marked *by network router* to indicate congestion
  - *policy* to determine marking chosen by network operator
- congestion indication carried to destination
- destination sets ECE bit on ACK segment to notify sender of congestion
- involves both IP (IP header ECN bit marking) and TCP (TCP header C,E bit marking)



TCP fairness

Fairness goal: if *K* TCP sessions share same bottleneck link of bandwidth *R*, each should have average rate of *R/K*



Is TCP fair?

A: Yes, under idealized assumptions:

- same RTT
- fixed number of sessions only in congestion avoidance

Network Layer (L4): Data Plane

→ transport segment from sending to receiving host

* routing table & algorithm

Network Layer function

Data plane: [Forwarding]

- *local*, per-router function
- determines how datagram arriving on router input port is forwarded to router output port

Control plane [Routing]

- *network-wide* logic
- determines how datagram is routed among routers along end-end path from source host to destination host

Internet "best effort" service model

No guarantees on:

i. successful datagram delivery to destination
ii. timing or order of delivery
iii. bandwidth available to end-end flow

Reflections on best-effort service:

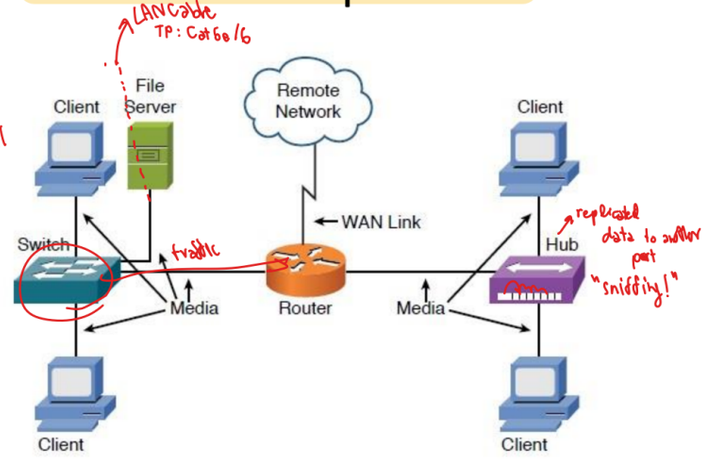
- simplicity of mechanism has allowed Internet to be widely deployed adopted
- sufficient provisioning of bandwidth allows performance of real-time applications (e.g., interactive voice, video) to be "good enough" for "most of the time"
- replicated, application-layer distributed services (datacenters, content distribution networks) connecting close to clients' networks, allow services to be provided from multiple locations
- congestion control of "elastic" services helps

It's hard to argue with success of best-effort service model

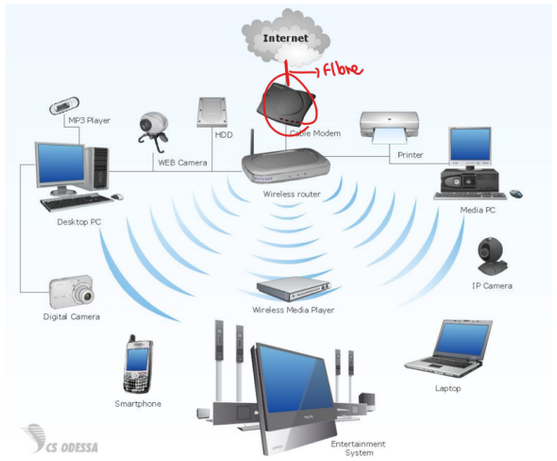
Network Components

24/7

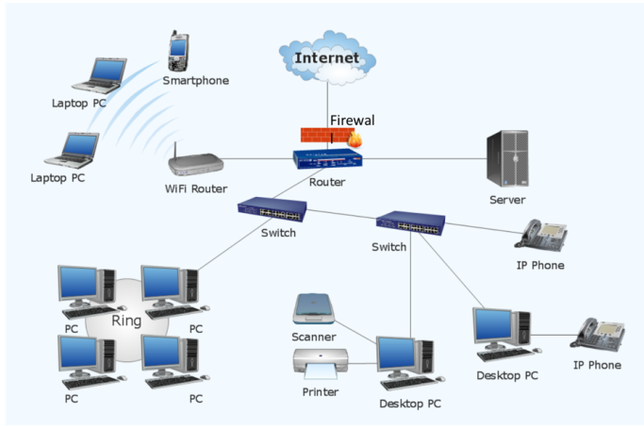
1. client
2. server
3. hub
4. switch
5. router
6. media
7. WAN link



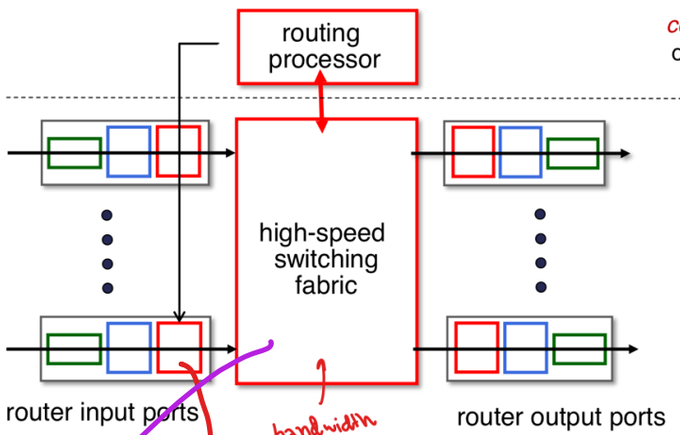
Network Architecture – Home



Network Architecture – Office



Router Architecture



routing, management control plane (software) operates in millisecond time frame

forwarding data plane (hardware) operates in nanosecond timeframe

decentralized switching:

- using header field values, lookup output port using forwarding table in input port memory *("match plus action")*
- destination-based forwarding: forward based only on destination IP address (traditional)
- generalized forwarding: forward based on any set of header field values

Network Layer

Check routing table on PC

> route /print

```
IPv4 Route Table
===========================================================================
Active Routes:
Network Destination        Netmask          Gateway       Interface  Metric
          0.0.0.0          0.0.0.0    192.168.1.1    192.168.1.108     35
        127.0.0.0        255.0.0.0         On-link         127.0.0.1    331
        127.0.0.1  255.255.255.255         On-link         127.0.0.1    331
  127.255.255.255  255.255.255.255         On-link         127.0.0.1    331
      192.168.1.0    255.255.255.0         On-link    192.168.1.108    291
    192.168.1.108  255.255.255.255         On-link    192.168.1.108    291
    192.168.1.255  255.255.255.255         On-link    192.168.1.108    291
        224.0.0.0        240.0.0.0         On-link         127.0.0.1    331
        224.0.0.0        240.0.0.0         On-link    192.168.1.108    291
  255.255.255.255  255.255.255.255         On-link         127.0.0.1    331
  255.255.255.255  255.255.255.255         On-link    192.168.1.108    291
===========================================================================
```

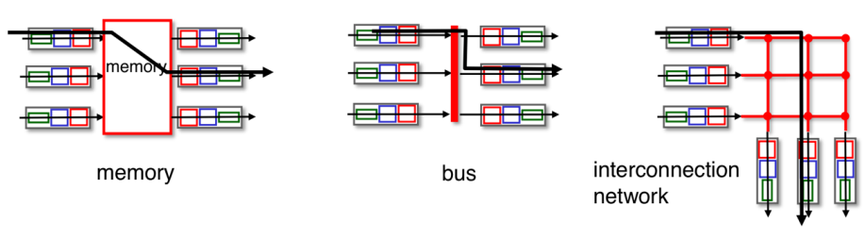
longest prefix match

when looking for forwarding table entry for given destination address, use *longest* address prefix that matches destination address.

- longest prefix matching: often performed using ternary content addressable memories (TCAMs)
  - *content addressable:* present address to TCAM: retrieve address in one clock cycle, regardless of table size

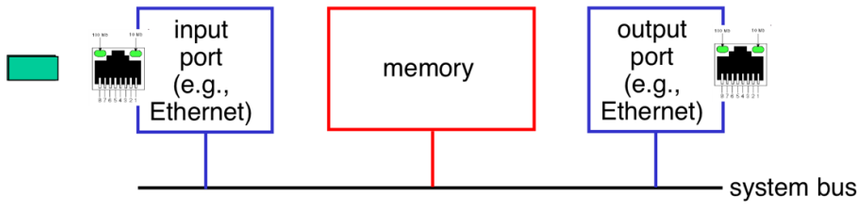
Switching fabrics

- transfer packet from input link to appropriate output link
- switching rate: rate at which packets can be transfer from inputs to outputs
  - often measured as multiple of input/output line rate
  - N inputs: switching rate N times line rate desirable
- three major types of switching fabrics:



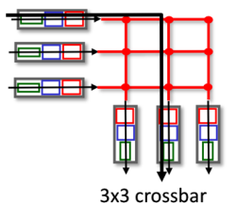
First generation routers:

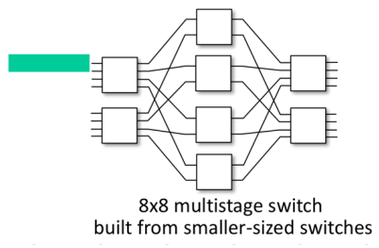
- traditional computers with switching under direct control of CPU
- packet copied to system's memory
- speed limited by memory bandwidth (2 bus crossings per datagram)



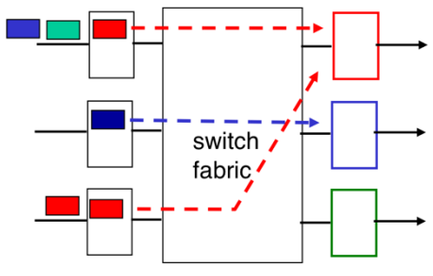
Switching via interconnection network

- Crossbar, Clos networks, other interconnection nets initially developed to connect processors in multiprocessor
- multistage switch: $n \times n$ switch from multiple stages of smaller switches
- exploiting parallelism:
  - fragment datagram into fixed length cells on entry
  - switch cells through the fabric, reassemble datagram at exit

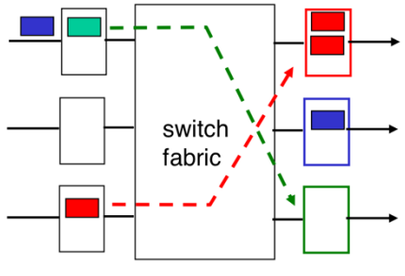




Output port queuing



at *t*, packets more from input to output

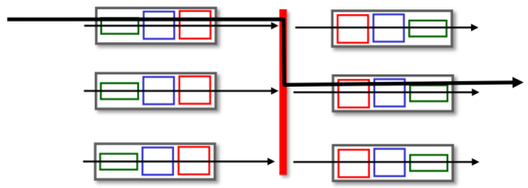


one packet time later

- buffering when arrival rate via switch exceeds output line speed
- *queueing (delay) and loss due to output port buffer overflow!*

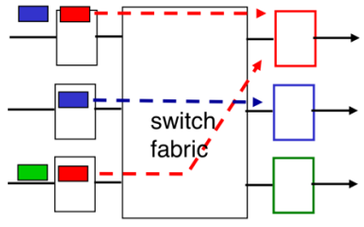
Switching via a bus

- Datagram from input port memory to output port memory via a shared bus
- *Bus contention:* switching speed limited by bus bandwidth
- 32 Gbps bus, Cisco 5600: sufficient speed for access routers

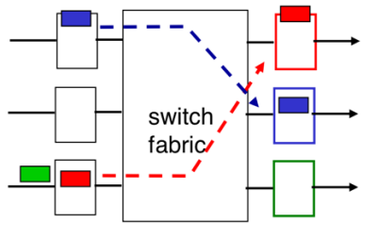


Input port queuing

- If switch fabric slower than input ports combined -> queueing may occur at input queues
  - queueing delay and loss due to input buffer overflow!
- Head-of-the-Line (HOL) blocking: queued datagram at front of queue prevents others in queue from moving forward



output port contention: only one red datagram can be transferred. lower red



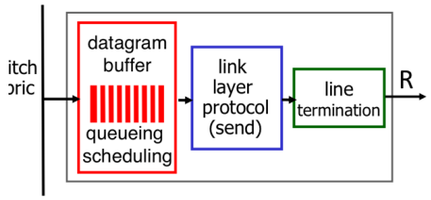
one packet time later: green packet experiences HOL blocking

→ Buffering

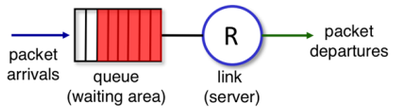
- RFC 3439 rule of thumb: average buffering equal to "typical" RTT (say 250 msec) times link capacity C
  - e.g., C = 10 Gbps link: 2.5 Gbit buffer
- more recent recommendation: with *N* flows, buffering equal to

$$\frac{RTT \cdot C}{\sqrt{N}}$$

Buffer Management



Abstraction: queue



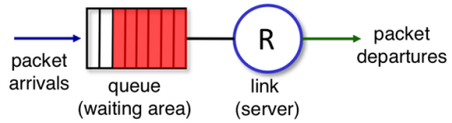
buffer management:

- drop: which packet to add, drop when buffers are full
  - tail drop: drop arriving packet
  - priority: drop/remove on priority basis
- marking: which packets to mark to signal congestion (ECN, RED)

Packet scheduling: deciding which packet to send next on link

- first come, first served (FCFS)
- priority
- round robin
- weighted fair queueing

Abstraction: queue

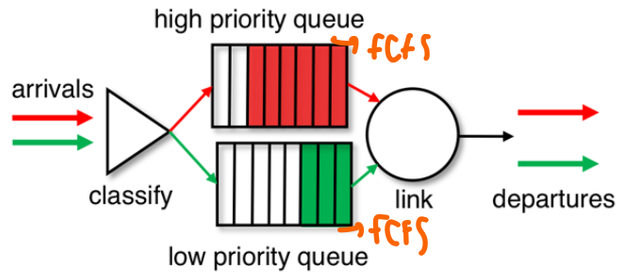


FCFS: packets transmitted in order of arrival to output port

- also known as: First-in-first-out (FIFO)

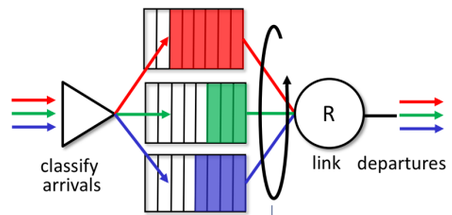
Priority scheduling:

- arriving traffic classified, queued by class
  - any header fields can be used for classification



Round Robin (RR) scheduling:

- arriving traffic classified, queued by class
  - any header fields can be used for classification
- server cyclically, repeatedly scans class queues, sending one complete packet from each class (if available) in turn

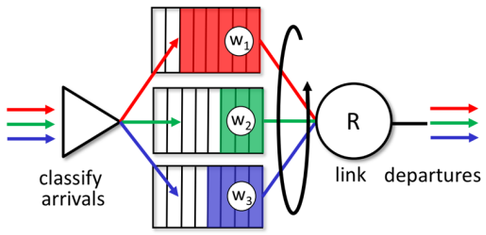


Weighted Fair Queuing (WFQ):

- generalized Round Robin
- each class, i, has weight, w_i, and gets weighted amount of service in each cycle:

$$\frac{w_i}{\sum_j w_j}$$

- Consider bandwidth per class
- minimum bandwidth guarantee (per-traffic-class)



What is network neutrality?

- *technical:* how an ISP should share/allocation its resources
  - packet scheduling, buffer management are the *mechanisms*
- *social, economic* principles
  - protecting free speech
  - encouraging innovation, competition
- enforced *legal* rules and policies

→ no blocking
no throttling
no paid prioritization

ISP: telecommunications or information service

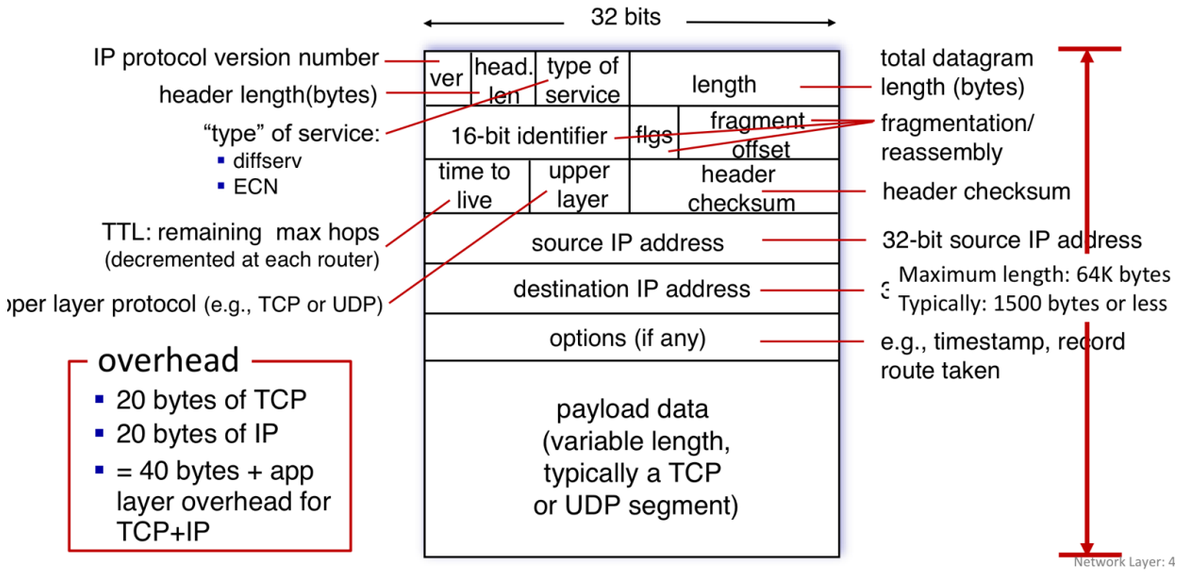
Is an ISP a "telecommunications service" or an "information service" provider?

- the answer *really* matters from a regulatory standpoint!

US Telecommunication Act of 1934 and 1996:

- *Title II:* imposes "common carrier duties" on *telecommunications services*: reasonable rates, non-discrimination and *requires regulation*
- *Title I:* applies to *information services:*
  - no common carrier duties (*not regulated*)
  - but grants FCC authority "… as may be necessary in the execution of its functions".

IP Datagram format

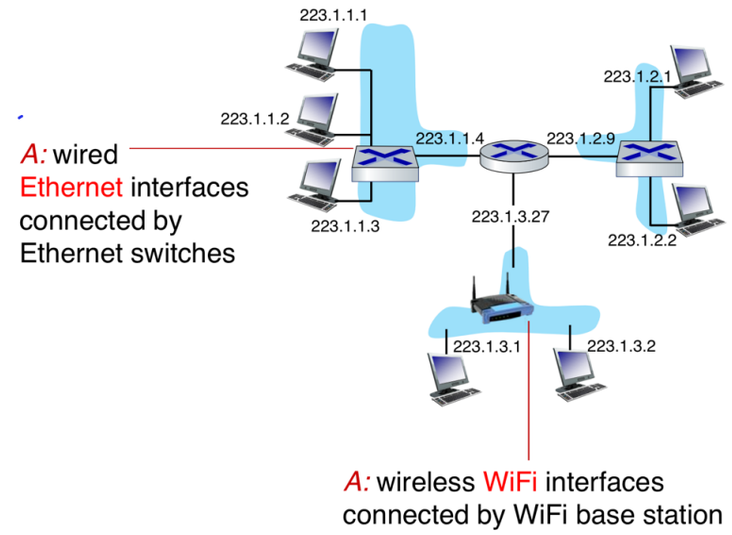


IP Address

Network | Subnet | Host

- IP address: 32-bit identifier associated with each host or router *interface*
- interface: connection between host/router and physical link
  - router's typically have multiple interfaces
  - host typically has one or two interfaces (e.g., wired Ethernet, wireless 802.11)

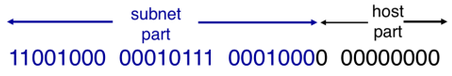
Q: how are interfaces actually connected?



IP addressing: CIDR

- *What's a subnet ?*
  - device interfaces that can physically reach each other without passing through an intervening router

CIDR: Classless InterDomain Routing (pronounced "cider")
- subnet portion of address of arbitrary length
- address format: a.b.c.d/x, where x is # bits in subnet portion of address



/23

IP addresses: how to get one?

That's actually two questions:

1. Q: How does a *host* get IP address within its network (host part of address)?
2. Q: How does a *network* get IP address for itself (network part of address)

How does *host* get IP address?
- hard-coded by sysadmin in config file (e.g., /etc/rc.config in UNIX)
- DHCP: Dynamic Host Configuration Protocol: dynamically get address from as server
  - "plug-and-play"

goal: host *dynamically* obtains IP address from network server when it "joins" network
- can renew its lease on address in use
- allows reuse of addresses (only hold address while connected/on)
- support for mobile users who join/leave network

DHCP overview:
- host broadcasts DHCP discover msg [optional]
- DHCP server responds with DHCP offer msg [c
- host requests IP address: DHCP request msg
- DHCP server sends address: DHCP ack msg

DHCP can return more than just allocated IP address on subnet:
- address of first-hop router for client (Gateway)
- name and IP address of DNS sever (for domain name lookup)
- network mask (indicating network versus host portion of address)

Q: how does *network* get subnet part of IP address?

A: gets allocated portion of its provider ISP's address space

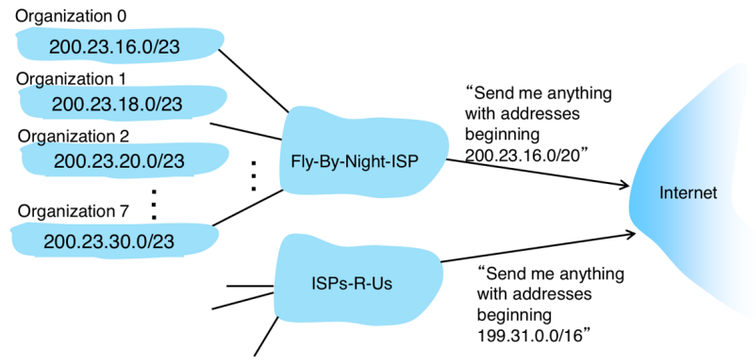
ISP's block	11001000 00010111 00010000	00000000	200.23.16.0/20

ISP can then allocate out its address space in 8 blocks:

Organization 0	11001000 00010111 00010000	00000000	200.23.16.0/23
Organization 1	11001000 00010111 00010010	00000000	200.23.18.0/23
Organization 2	11001000 00010111 00010100	00000000	200.23.20.0/23
...	.....	....	....
Organization 7	11001000 00010111 00011110	00000000	200.23.30.0/23

Hierarchical addressing: route aggregation

hierarchical addressing allows efficient advertisement of routing information:

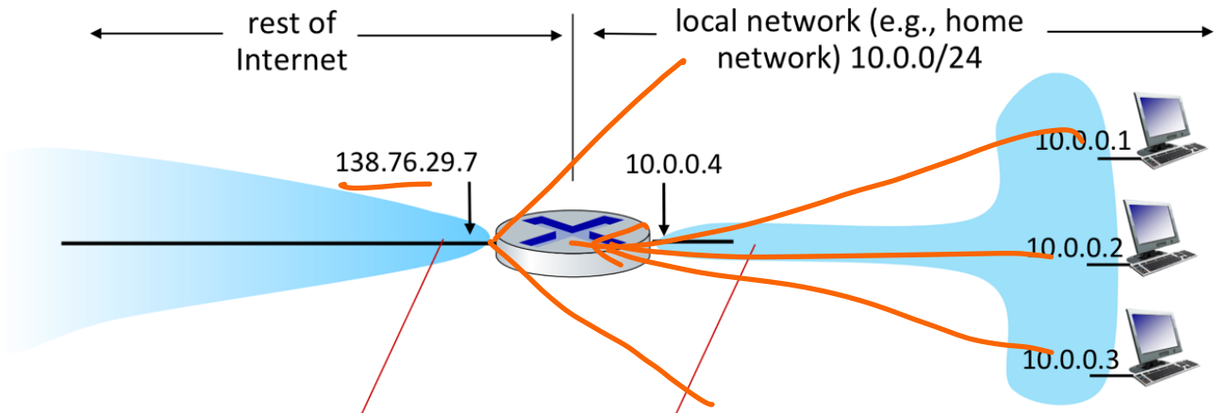


Q: are there enough 32-bit IP addresses?

- ICANN allocated last chunk of IPv4 addresses to RRs in 2011
- NAT (next) helps IPv4 address space exhaustion
- IPv6 has 128-bit address space

Network address translation

NAT: all devices in local network share just one IPv4 address as far as outside world is concerned



all datagrams *leaving* local network have *same* source NAT IP address: 138.76.29.7, but *different* source port numbers

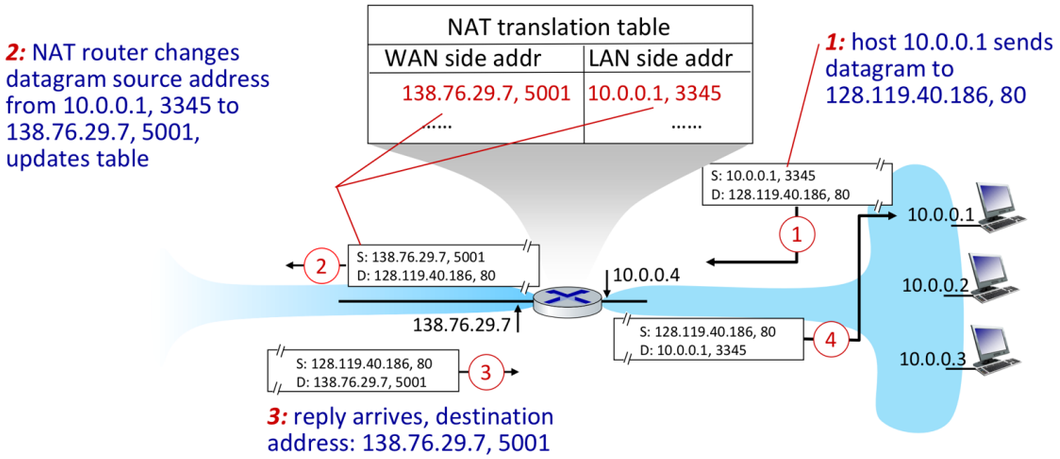
datagrams with source or destination in this network have 10.0.0/24 address for source, destination (as usual)

- All devices in local network have 32-bit addresses in a "private" IP address space (10/8, 172.16/12, 192.168/16 prefixes) that can only be used in local network
- Advantages:
  - just one IP address needed from provider ISP for *all* devices
  - can change addresses of host in local network without notifying outside world
  - can change ISP without changing addresses of devices in local network
  - security: devices inside local net not directly addressable, visible by outside world

implementation: NAT router must (transparently):

- outgoing datagrams: replace (source IP address, port #) of every outgoing datagram to (NAT IP address, new port #)
  - remote clients/servers will respond using (NAT IP address, new port #) as destination address
- remember (in NAT translation table) every (source IP address, port #) to (NAT IP address, new port #) translation pair
- incoming datagrams: replace (NAT IP address, new port #) in destination fields of every incoming datagram with corresponding (source IP address, port #) stored in NAT table

NAT: network address translation



- NAT has been controversial:
  - routers "should" only process up to layer 3
  - address "shortage" should be solved by IPv6
  - violates end-to-end argument (port # manipulation by network-layer device)
  - NAT traversal: what if client wants to connect to server behind NAT?
- but NAT is here to stay:
  - extensively used in home and institutional nets, 4G/5G cellular nets