COMP3331 NOTES

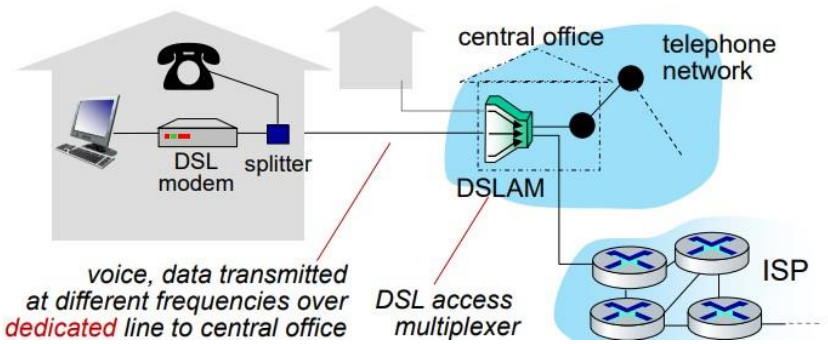
Access Networks:

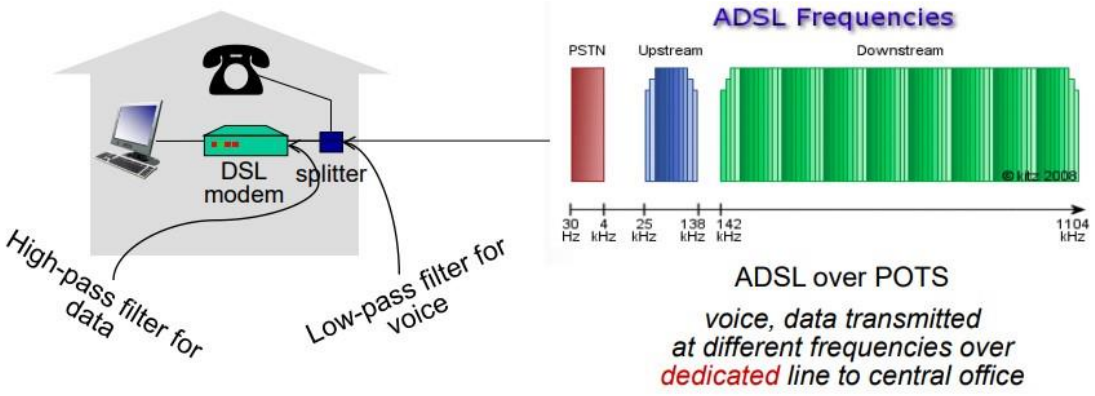
Digital Subscriber Line (DSL):

Uses existing telephone lines to central office *DSLAM.*

- Data over DSL phone line goes to internet
- Voice over DSL phone line goes to telephone net

DSLAM: *Digital Service Line Access Multiplexer*





Cable-Based Access:

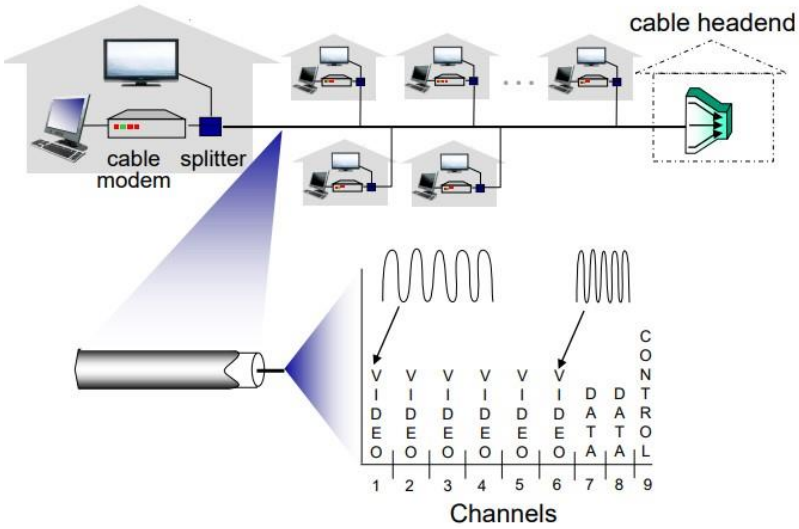
Uses *frequency division multiplexing (FDM)*

- Different channels transmitted in different frequency bands.
- Shared cable to transmit data, TV, etc.

Hybrid Fiber Coax (HFC):

- asymmetric: up to 40Mbps – 1.2Gbs downstream transmission rate, 30-100 Mbps upstream transmission rate.

Network of cable, fiber attaches home to ISP router:



- Homes *share access network* to cable headend
- In contrast to DSL, which has *dedicated* access to central office.

Fiber to the home/premise/curb:

- Fully optical fiber path all the way to destination.
- E.g. NBN, Google, Verizon FIOS
- ~30 Mbps to 1Gbps

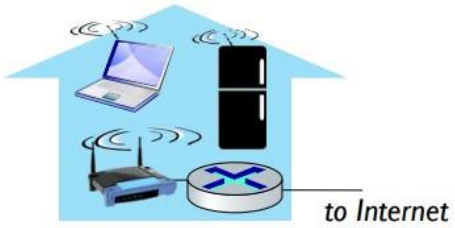
Wireless Access Networks:

Shared *wireless* access network connects end system to router

- Via base station (access point).

Wireless local area networks (WLANs)
Typically within or around building (~100ft)

802.11b/g/n (Wi-Fi): 11, 54, 450 Mbps transmission rate.

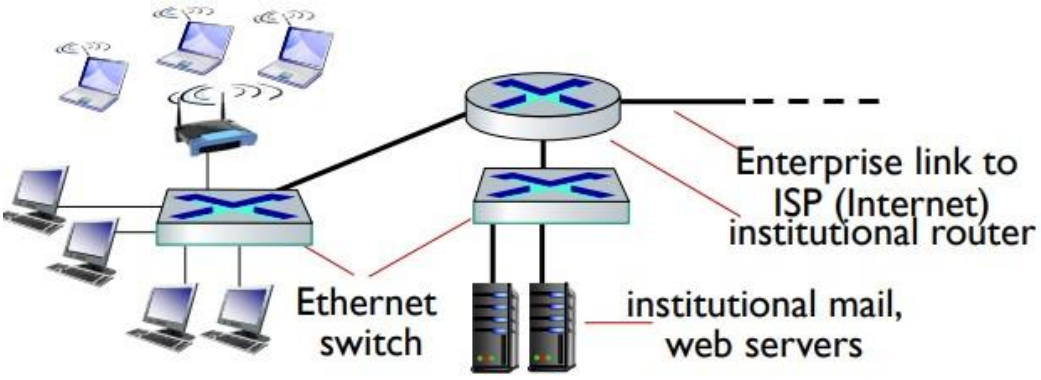


Wide-area cellular access networks
Provided by cellular network operators (10's km)

10's Mbps

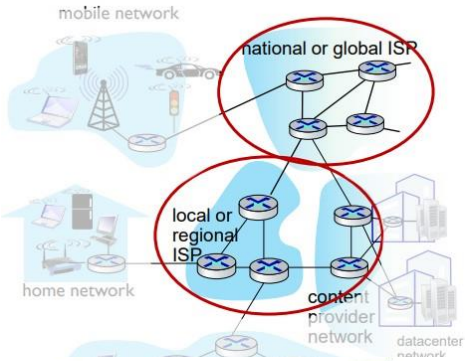
4G, 5G networks.

Access Networks: enterprise networks:



- Companies, universities, etc.
- Mix of wired, wireless link technologies, connecting a mix of switchers and routers
    - *Ethernet*: Wired access at 100 Mbps, 1Gbps, 10Gbps
    - *Wi-Fi*: wireless access points at 11, 54, 450 Mbps

The Network Core

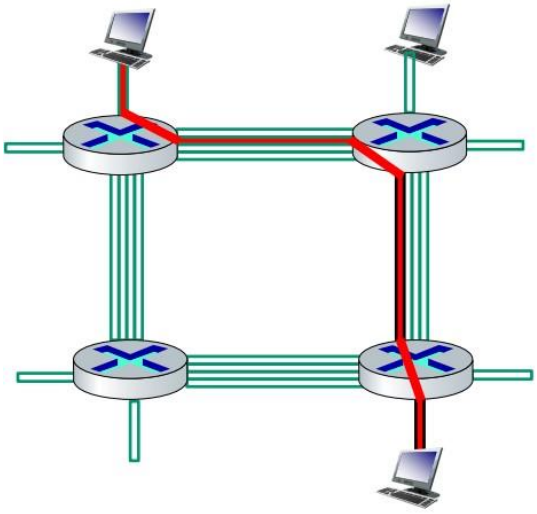


Mesh of interconnected routers

Packet-switching: hosts break application layer into *packets:*

- Forward packets from one router to the next, across links on path from source to destination
- Each packet transmitted at full link capacity.

Circuit-switching: An alternative used in *legacy telephone networks* which was considered during the design of the internet

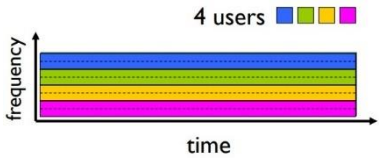


- Resources allocated and *reserved* for "call" between source and destination.
- In diagram, each link has 4 circuits.
  - Call gets 2nd circuit in top link and 1st circuit in right link,
- *Dedicated resources*: no sharing.
  - Guaranteed performance
- Circuit segment idle if not being used (wasteful)

Circuit Switching: FDM and TDM:

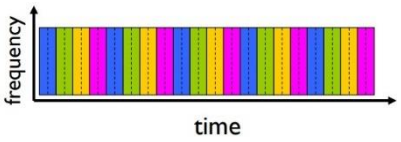
Frequency Division Multiplexing (FDM):

- Optical, electromagnetic frequencies divided into (narrow) frequency bands.
- Each call allocated its own band, can transmit at max rate of that narrow band.



Time Division Multiplexing (TDM):

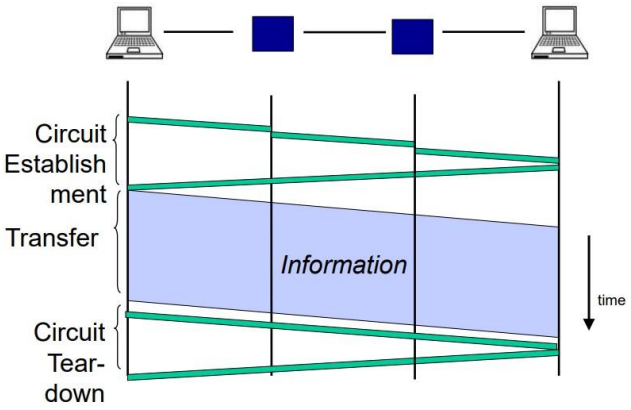
- Time divided into slots
- Each call allocated periodic slot(s), can transmit at maximum rate of (wider) frequency band, but only during its time slot(s).



Why not circuit switching?

Inefficient:

- Internet communication tends to be very bursty (High usage at once, e.g. load web pages).
- Waste of capacity after burst.
- No adaption to network dynamics



Fixed data rate:

- Computers communicate at very diverse rates. E.g. video streaming / telnet / web browsing.

Connection state maintenance

- Requires per communication state to be maintained (considerable overhead).
- Not scalable.

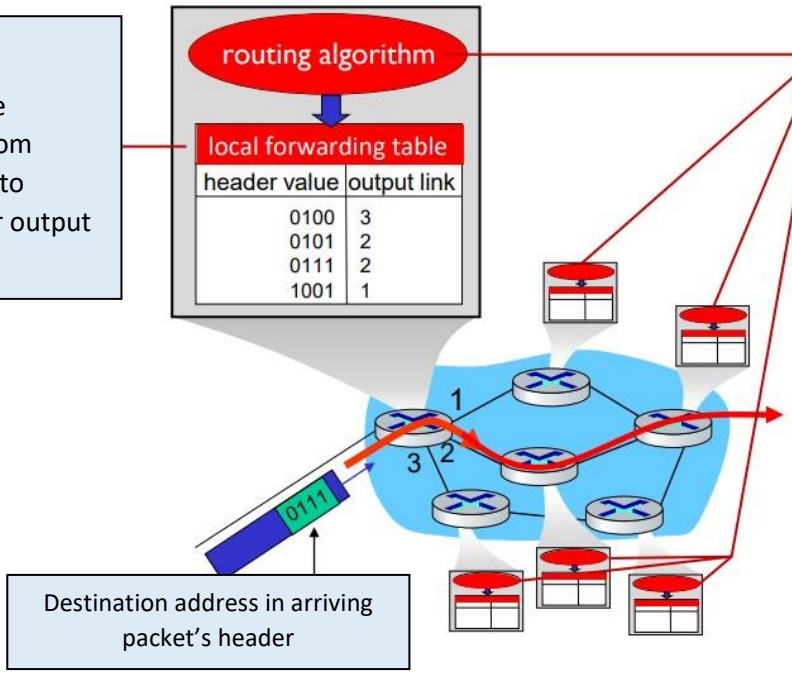
Packet Switching:

- Data is sent as chunks of formatted bits (*packets)*
- Packet consists of *header* and *payload*.
    - Payload: data being carried
    - Header: Instructions on how to handle packet.
- Switches 'forward' packets based on their headers
- Each packet travels independently (no notion of packets belonging to a circuit).
- No link resources are reserved.
    - Packet switching instead leverages statistical multiplexing.
    - Relies on assumption that not all flows burst at the same time
-

Peak ahead: Two key network-core functions

Forwarding:

***Local* action:** move arriving packets from router's input link to appropriate router output link.



Routing:

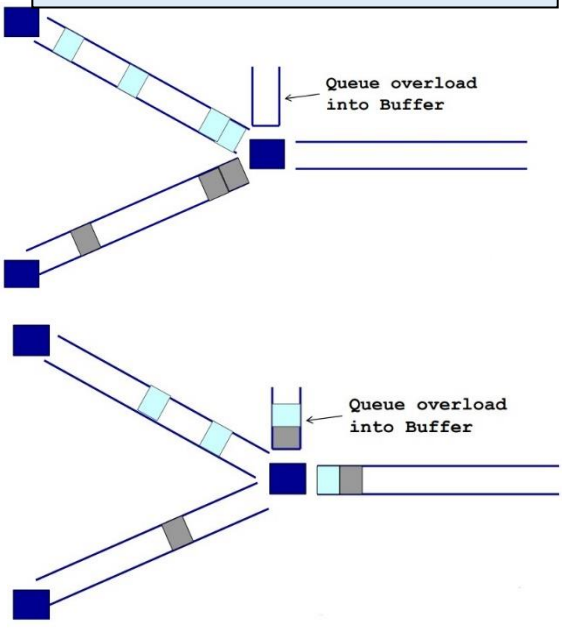
Global action: determine source destination paths taken by packets.

Uses *routing algorithms*.

Timing in Packet Switching:

- Time to process packet at switch is relatively negligible.
- We will always assume a switch processes/forwards a packet *after it has received it entirely.*
    - This is called "store and forward" switching.
- Switch can start transmitting as soon as it has processed the header (as opposed to processing whole packet).
    - This is called a "cut through" switch.

Image: buffer absorbing transient bursts during statistical multiplexing.



Statistical Multiplexing:

How to deal with capacity overload?

- When the queue is overloaded, store the overload into a buffer.
- If overload is *persistent*, eventually packets will be dropped.

Packet Switching vs Circuit Switching

Packet switching allows more users to use network!

E.g.
1 Mb/s link
Each user:
- 100kb/s when 'active'
- Active 10% of the time

Circuit switching:
10 users

Packet Switching:
With 35 users, probability > 10 active at same time is less than 0.004.

Binomial Probability Distribution

A fixed number of observations (trials), n.
- E.g. 5 tosses of a coin

Binary random variable
- E.g. head/tail in coin toss
- Often called success/failure
- Prob of success is *p*
- Prob of failure is (*1-p*)

Constant probability for each observation

Example: Q: What is the probability of observing exactly 3 heads in a sequence of 5 coin tosses?

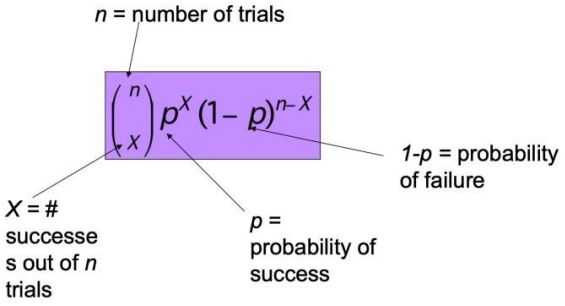
- One way to get exactly 3 heads is: *HHHTT*
- Probability of this sequence occurring: $\left(\frac{1}{2}\right) \times \left(\frac{1}{2}\right) \times \left(\frac{1}{2}\right) \times \left(1-\frac{1}{2}\right) \times \left(1-\frac{1}{2}\right) = \left(\frac{1}{2}\right)^5$
- Another way to get exactly 3 heads is: THHHT
- Probability of this sequence occurring: $\left(1-\frac{1}{2}\right) \times \left(\frac{1}{2}\right) \times \left(\frac{1}{2}\right) \times \left(\frac{1}{2}\right) \times \left(1-\frac{1}{2}\right) = \left(\frac{1}{2}\right)^5$

How many such combinations exist?

$$\binom{5}{3} = 5C3 = \frac{5!}{3!\,2!\,1!} = 10$$

P(3 heads and 2 tails) = $10 \times \left(\frac{1}{2}\right)^5 = 0.3125$

Note the general pattern emerging → if you have only two possible outcomes (call them 1/0 or yes/no or success/failure) in *n* independent trials, then the probability of exactly *X* "successes"=



Packet Switching vs Circuit Switching continued…

N = 35 *Prob (# active users > 10) =*	1 – Prob(# active = 10) – Prob(# active = 9) … – Prob(# active = 0)	*Where Prob (# active = 10) =* $C(35,10) \times 0.1^{10} \times 0.9^{25}$ *Prob (# active users > 10) = 0.0004 (approx.).*

With Packet switching, excessive congestion is possible: packet delay and loss due to buffer overflow.
- Protocols are required to ensure reliable data transfer, congestion control.

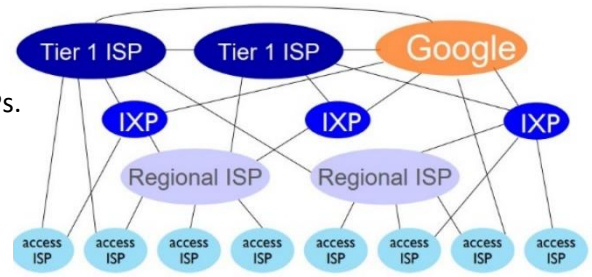
Q: How to provide circuit-like behaviour in Packet Switching?
- Bandwidth guarantees traditionally used for audio/video applications.

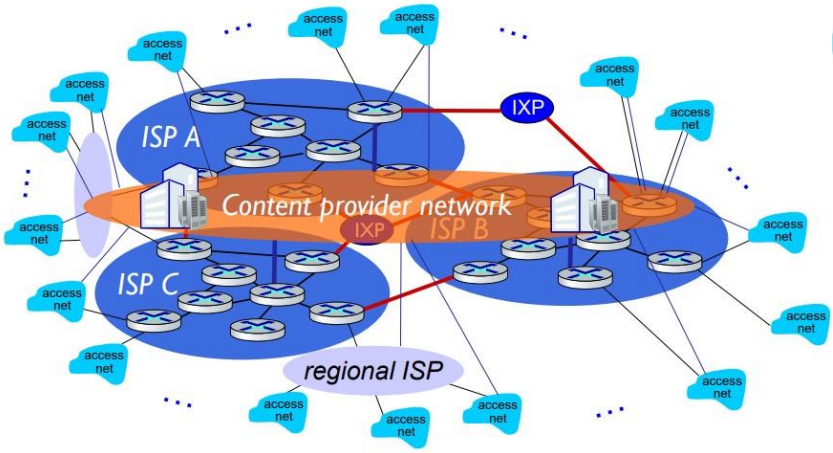
Internet Structure: A 'Network of Networks':

Hosts connect to Internet via access Internet Service Providers (ISPs)

- Residential, enterprise (company, university, commercial) ISPs.

Access ISPs in turn must be interconnected.





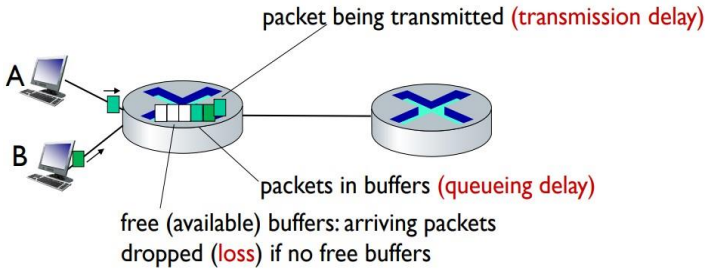
IXP: Internet Exchange Point

Regional networks arise to connect access nets to ISPs.

Content provider networks (e.g. Google, Microsoft, Akamai) may run their own network to bring services/content closer to end users.

- "Tier 1" commercial ISPs (e.g. Level 3, Sprint, AT&T, NTT): national and international coverage.
- Content provider networks (e.g. Google, Facebook): private network that connects its data centers to internet, often bypassing tier-1, regional ISPs

Performance: Loss, Delay, Throughput:

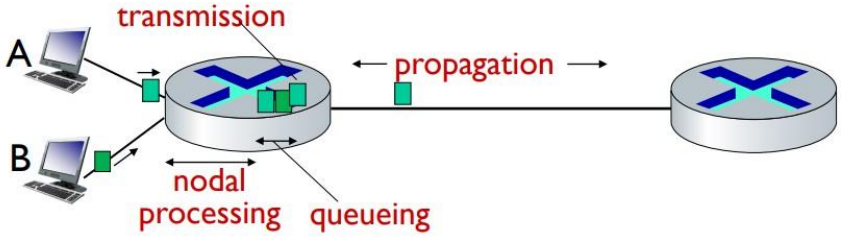


How does packet loss and delay occur?

Packets queue in router buffers

- Packets queue, wait for turn
- Arrival rate to link (temporarily) exceeds output link capacity: packet loss

Packet delay: four sources



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

d_{proc}: nodal processings

- Check bit errors
- Determine output link
- Typically < 1ms

d_{queue}: queueing delay

- Time waiting at output link for transmission
- Depends on congestion level of router

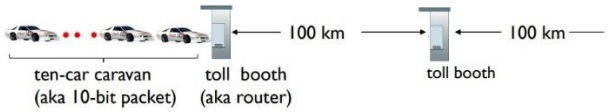
d_{trans}: transmission delay

- L: packet length (bits)
- R: link transmission rate (bps)
- $d_{trans} = \frac{L}{R}$

d_{prop}: propagation delay

- d: length of physical link
- s: propagation speed ($\sim 2 \cdot 10^8$ m/sec)
- $d_{prop} = \frac{d}{s}$

Caravan / Toll-booth Analogy:



- cars "propagate" at 100 km/hr
- toll booth takes 12 sec to service car (bit transmission time)
- car ~ bit; caravan ~ packet
- *Q:* How long until caravan is lined up before 2nd toll booth?

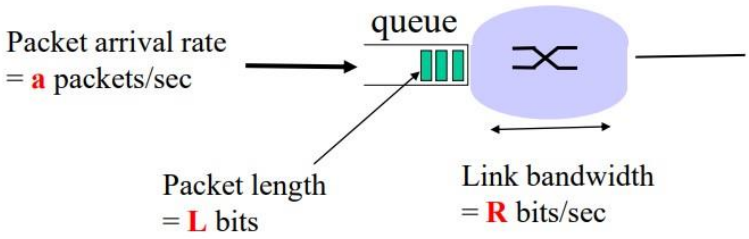
- time to "push" entire caravan through toll booth onto highway = 12*10 = 120 sec
- time for last car to propagate from 1st to 2nd toll both: 100km/(100km/hr) = 1 hr
- *A:* 62 minutes

- suppose cars now "propagate" at 1000 km/hr
- and suppose toll booth now takes one min to service a car
- *Q:* Will cars arrive to 2nd booth before all cars serviced at first booth?

A:Yes! after 7 min, first car arrives at second booth; three cars still at first booth

---

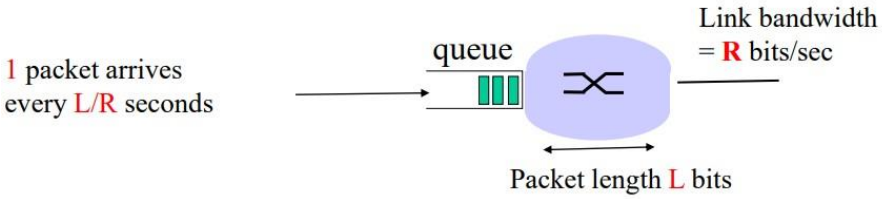
Queuing Delay (More Insight)



- Every second: aL bits arrive to queue
- Every second: R bits leave the router
- Question: what happens if $aL > R$?
- Answer: queue will fill up, and packets will get *dropped*.

aL/R is called traffic intensity

Typical behaviour:



Arrival rate: a = 1/(L/R) = R/L (packet/second)

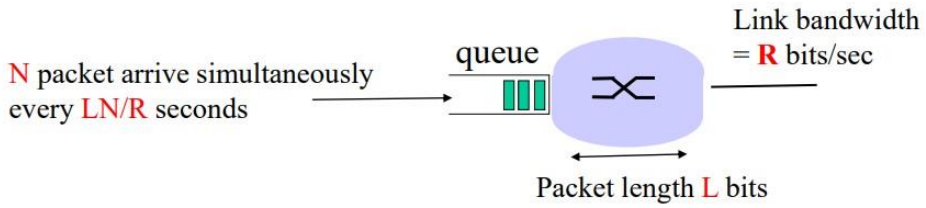
Traffic intensity = aL/R = (R/L) (L/R) = 1

Average queueing delay = 0
(queue is initially empty)

This diagram depicts *typical* behaviour (average 0/low delay). Outcomes can include:

- $\frac{La}{R} \sim 0$: average queueing delay is small
- $\frac{La}{R} \to 1$: delays become large
- $\frac{La}{R} > 1$: more 'work' than can be serviced, average delay *infinite*.

Congested queue scenario:



Arrival rate: a = N/(LN/R) = R/L packet/second

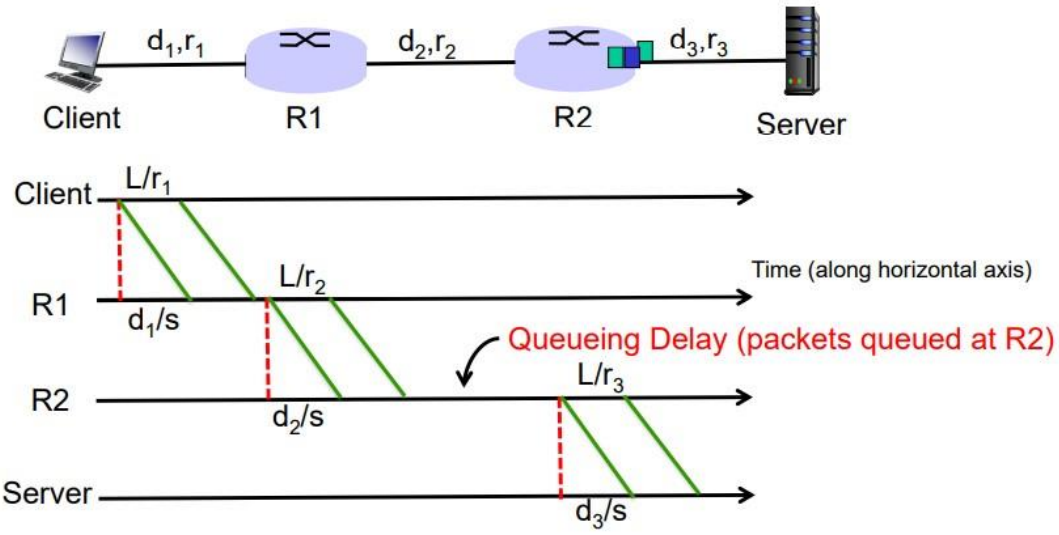
Traffic intensity = aL/R = (R/L) (L/R) = 1

Average queueing delay (queue is empty at time 0) ?

{0 + L/R + 2L/R + … + (N-1)L/R}/N = L/(RN){1+2+…+(N-1)} =L(N-1)/(2R)

Note: traffic intensity is same as previous scenario, but queueing delay is different

Example of Client -> R1 -> R2 -> Server, with queuing delay occurring at R2



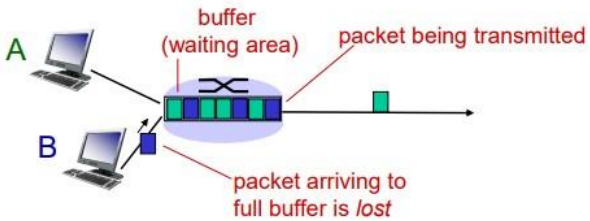
"Real" delay variations:

$d_{nodal} = d_{proc} + d_{queue} + d_{trans} + d_{prop}$ *(In this order!)*

End-to-end delay = sum of d_{nodal} along the path

Packet Loss:

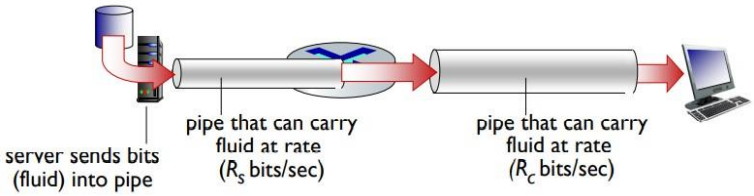
- Queue (aka buffer) preceding link has *finite capacity.*
- Packet arriving to full queue is dropped (aka lost)
- Lost packet may be retransmitted by previous node, source end system, or not at all

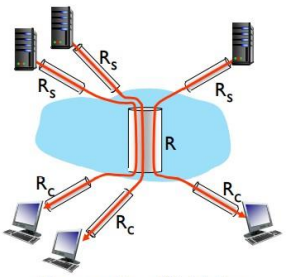


Throughput:

Throughput: rate (bits/time unit) at which bits are being sent from sender to receiver.

- Instantaneous: rate at given point in time
- Average: Rate over longer period of time.

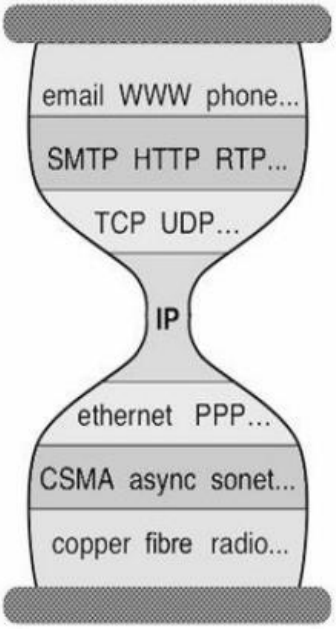




- per-connection end-end throughput: $min(R_c,R_s,R/10)$
- in practice: R_c or R_s is often bottleneck

Links which constrain overall throughput are considered to be *bottleneck links*.

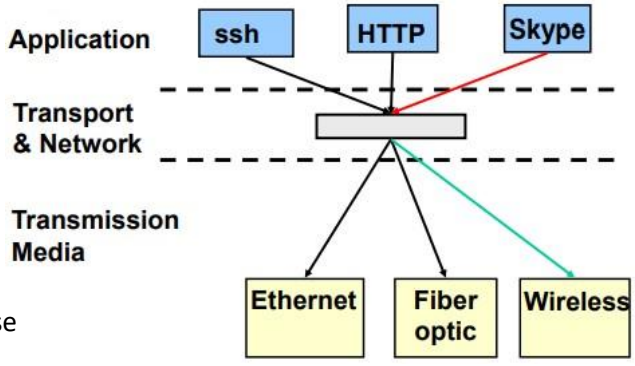
Protocol Layers / Service Models



What are the tasks in networking?

- *Application* - Prepare data, support network applications
  - FTP, **SMTP**, **HTTP**, Skype, etc
- *Transport* - Ensure that packets get to the destination process
  - **TCP**, **UDP**
- *Network* – Deliver packets across global network
  - **IP**, routing protocols
- *Datalink* – Delivery packets within local network to next hop
  - **Ethernet**, 802.11 (**WiFi**), PPP
- *Physical* – Bits / Packets on wire

This is known as *decomposition.*

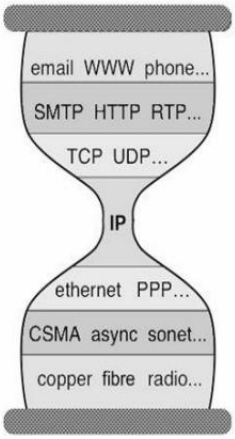


Three observations:

- Each layer:
  - *Depends on* layer below
  - *Supports* layer above
  - *Independent* of others
- Multiple versions in layer
  - Interfaces differ somewhat
  - Components pick which lower-level protocol to use
- Only one IP layer
  - Unifying protocol.

Why layering?

- Intermediate layers provides a *common abstraction* for various network technologies.



Downsides of layering:

- Layer N may duplicate lower-level functionality
  - i.e. Error recovery to retransmit lost data
- Information hiding may hurt performance
  - i.e. Packet loss due to corruption vs congestion
- Headers start to get large
  - i.e. TCP + IP + Ethernet headers typically add up to *54 bytes*
- Layer violations when the gains too great to resost
  - E.g. Network Address Translation (NAT)
- Layer violations when network doesn't trust ends
  - Security, Firewalls

Distributing Layers Across The Network

Layers become increasingly more complex when they begin to interact with multiple machines.

- Hosts
- Routers
- Switches
- *What gets implemented where?*

What gets implemented on Host?

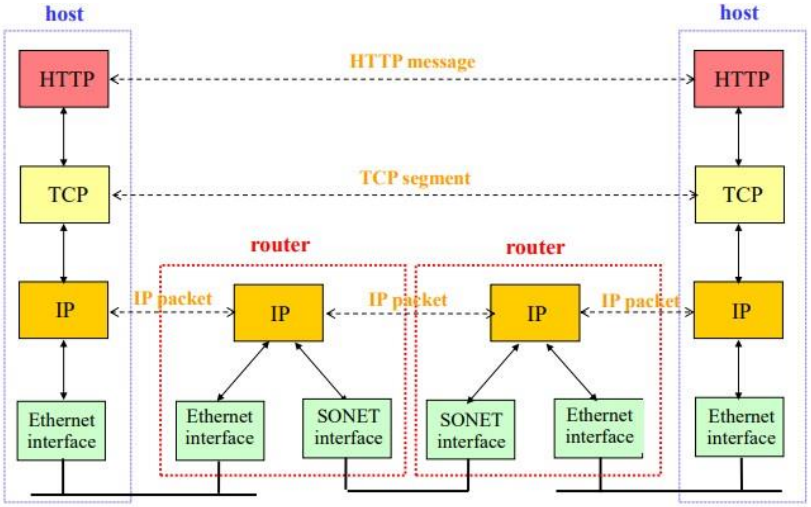
- Hosts have applications that generate data/messages that are eventually put on wire.
- At receiver host bits arrive on wire and make it up to application.
- Therefore, all layers must exist at host.

What gets implemented on Router?

- Physical - Bits arrive on wire
- Datalink – Packets must be delivered to next-hop
- Network – Routers participate in global delivery

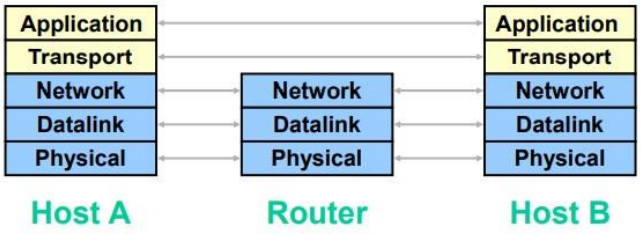
Routers do NOT support reliable delivery

- Transport layer (and above) **not** supported.



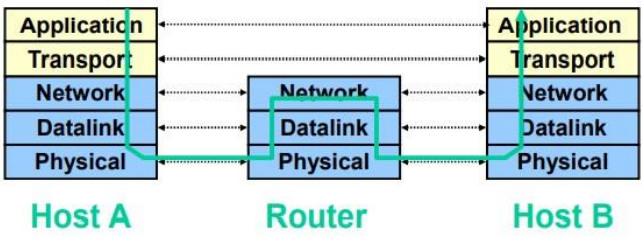
Logical Communication:

Layers interact with peer's corresponding layer



Physical Communication:

- Communication goes dow not physical network
- Then from network peer to peer
- Then up to relevant layer



Application layer

Purpose?

- Conceptual, implementation aspects of network application protocols
  - Transport-layer service models
  - Client-server paradigm
  - Peer-to-peer paradigm
- Learn about protocols by examining popular application-level protocols
  - HTTP
  - SMTP, IMAP
  - DNS
- Programming network applications
  - Socket API

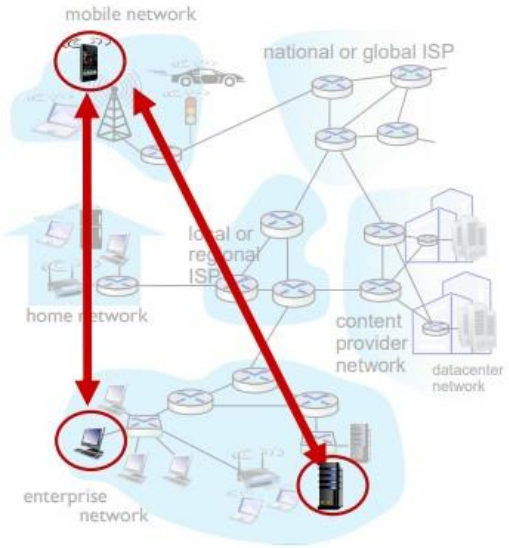
Creating a network app:

Write programs that:

- Run on *different* end systems
- Communicate over network
- E.g. Web server software communicates with browser software

No need to write software for network-core devices:

- Network-core devices do not run user applications
- Applications on end systems allows for rapid app development, propagation.



Client-server paradigm

Server:

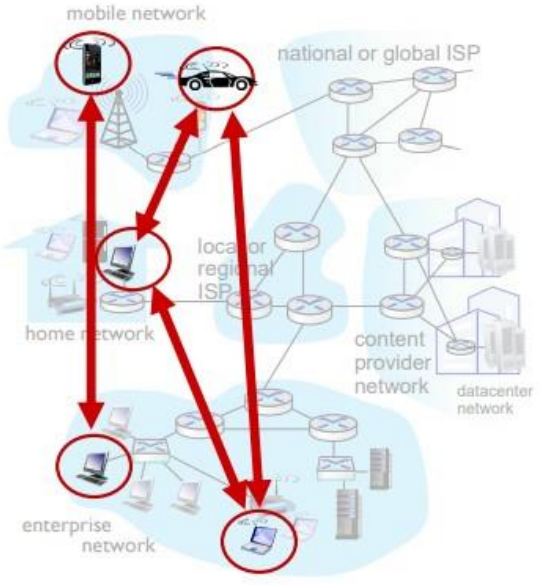
- Always-on host
- Permanent IP address
- Often in data centres (for scaling)

Clients:

- Contact/communicate with server
- *May* be intermittently connected
- *May* have dynamic IP addresses
- Do not communicate directly with each other
  - This would be P2P
- E.g. HTTP, IMAP, FTP

Peer-to-Peer architecture (P2P)

- No always-on server
- Arbitrary end systems directly communicate
- Peers request service from other peers, provide service in return to other peers
  - Self scalability – new peers bring new service capacity, *as well as new service demands*
- Peers are intermittently connected and change IP address
  - Complex management
- Example: P2P file sharing, blockchain



Processes communicated

Process: program running within a host
- Within same host, two processes communicate using inter-process communication (defined by OS)
- Processes in different hosts communicate by exchanging messages

Note: Applications with P2P architectures have client processes and server processes

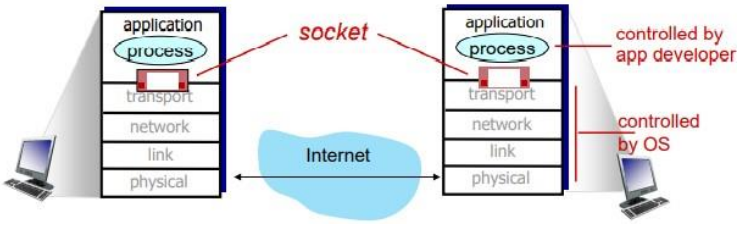
Clients / Servers

Client Process: process that initiates communication

Server Process: process that waits to be contacted

Sockets

Process sends/receives messages to/from its socket
A socket is analogous to a *door*:
- Sending process shoves message out the door
- Sending process relies on transport infrastructure on other side of door to deliver message to
  socket at receiving process
- Two sockets involved: one on each side



Addressing processes

- To receive messages, process must have identifier
- Host device has unique 32-bit IP address
- Q: does IP address of host on which process runs suffice for identifying the process?
- A: No, *many* processes can be running on same host
- Identifier includes both IP address and port numbers associated with process on host
- Example port numbers:
  - HTTP server: 80
  - Mail server: 25
- To send HTTP message to gaia.cs.unmass.edu web server:
  - IP address: 128.119.245.12
  - Port number: 80

An application-layer protocol defines:

Types of messages exchanged:
- E.g. request, response

Message syntax:
- What fields in messages & how fields are delineated

Message semantics:
- Meaning of information in fields

Rules for when and how processes send and respond to messages

Open protocols:
- Defined in RFCs, everyone has access to protocol definition
- Allows for interoperability
- E.g. HTTP, SMTP, WebRTC

Proprietary protocols:
- E.g. Skype, Zoom, Teams.

What transport service does an app need?

Data integrity
- Some apps (e.g. file transfer, web transactions) require 100% reliable data transfer
- Other apps (e.g. audio) can tolerate some loss

Timing
- Some apps (e.g. Internet telephony, interactive games) require low delay to be 'effetive'

Throughput
- Some apps (e.g. multimedia) require minimum amount of throughput to be 'effective'
- Other apps ('elastic' apps) make use of whatever throughput they can get

Security
- Encryption, data integrity, etc

Transport service requirements: common apps

Application	Data loss	Throughput	Time sensitivity
File transfer/download	No loss	Elastic	N/A
E-mail	No loss	Elastic	N/A
Web Documents	No loss	Elastic	N/A
Real-time audio/video	Loss-tolerant	*Audio*: 5Kbps-1Mbps *Video*: 10Kbps-5Mbps	10's of ms
Streaming audio/video	Loss-tolerant	*Audio*: 5Kbps-1Mbps *Video*: 10Kbps-5Mbps	Few seconds
Interactive games	Loss-tolerant	Kbps+	10's of ms
Text messaging	No loss	Elastic	Context dependent (Sometimes?)

Internet Transport Protocol Services

TCP service

- Reliable transport between sending and receiving process
- Flow control: sender won't overwhelm receiver
- Congestion control: throttle sender when network overloaded
- *Does not provide*: timing, minimum throughput guarantee, security
- Connection-oriented: setup required between client and server process

UDP Service:

- Unreliable data transfer between sending and receiving process
- *Does not provide:* reliability, flow control, congestion control, timing, throughput guarantee, security, or connection setup.

Common apps

Application	Application Layer Protocol	Transport Protocol
File transfer/download	FTP [RFC 959]	TCP
E-mail	SMTP [RFC 5321]	TCP
Web documents	HTTP 1.1 [RFC 7320]	TCP
Internet Telephony	SIP [RFC 3261], RTP [RFC 3550], or proprietary	TCP or UDP
Streaming audio/video	HTTP [RFC 7320], DASH	TCP
Interactive games	WOW, FPS (proprietary)	UDP or TCP

Securing TCP

Vanilla TCP & UDP sockets:
- No encryption
- Cleartext passwords sent into socket traverse Internet in cleartext (!)

Transport Layer Security (*TLS*):
- Provides encrypted TCP connections
- Data integrity
- End-point authentication

TLS implemented in application layer:
- Apps use TLS libraries, that use TCP in turn

TLS socket API:
- Cleartext sent into socket traverse Internet *encrypted.*

Application Layer: Web and HTTP

First HTTP impl. (1990)	Created by Tim Berners at CERN
HTTP/0.9 (1991)	Simple GET command for the Web
HTTP/1.0 (1992)	Client/Server information, simple caching
HTTP/1.1 (1996)	Messages are secured using Secure Sockets Layer (SSL) or its successor Transport Layer Security (TLS) HTTPS is used *automatically* at start of any URL
HTTP/2.0 (2015)	Performance improvements from HTTP/1.1. Less TCP requests required to process a web page.

Web and HTTP: Review

- Web page consists of *objects*, each of which can be stored on different Web servers
- Object can be HTML file, JPEG image, Java applet, audio file, etc
- Web page consists of *base HTML-file* which includes *several referenced objects, each* addressable by a *URL*, e.g.

Host Name **Path Name**
`www.someschool.edu / someDept/pic.gif`

Uniform Resource Locator (URL)

`protocol://host-name[:port]/directory-path/resource`	
Protocol	HTTP, FTP, HTTPS, SMTP, etc
Hostname	DNS name, IP address
Port	Defaults to protocol's standard port - E.g. HTTP: 80, HTTP: 443, SMTP: 25
Directory path	Hierarchical, reflecting file system
Resource	Identifies the desired resource

HTTP Overview

HTTP: Hypertext Transfer Protocol

- Web's application layer protocol
- Client/Server model:
  - *Client:* browser that requests, receives, (using HTTP protocol) and 'displays' Web objects
  - *Server:* Web server sends (using HTTP protocol) objects in response to requests

HTTP uses TCP

- Client initiates TCP connection (creates socket) to server, port 80
- Server accepts TCP connection from client
- HTTP messages (application-layer protocol messages) exchanged between browser (HTTP client) and Web server (HTTP server)
- TCP connection closed

HTTP is 'stateless'

- Server maintains *no* information about past client requests

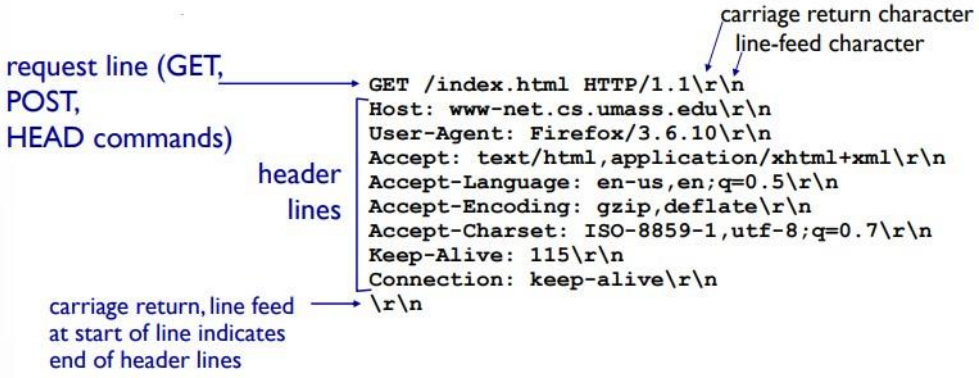
Note: Protocols that maintain state are complex!

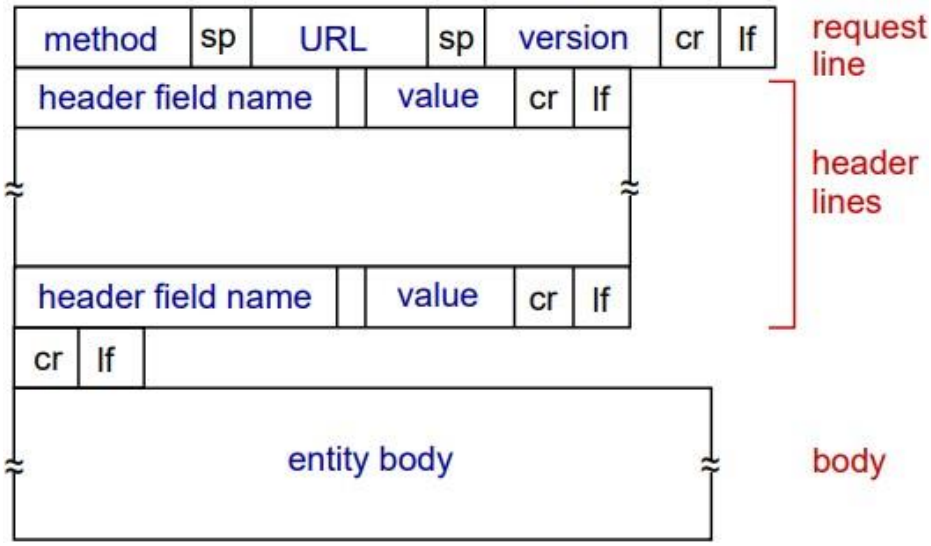
- State history must be maintained
- If server/client crashes, their views may be inconsistent and have to backtrack

HTTP Request Message

Two types of HTTP messages:
- *Request*
- *Response*

HTTP request message:
- ASCII (human-readable format)





Other HTTP Request Messages

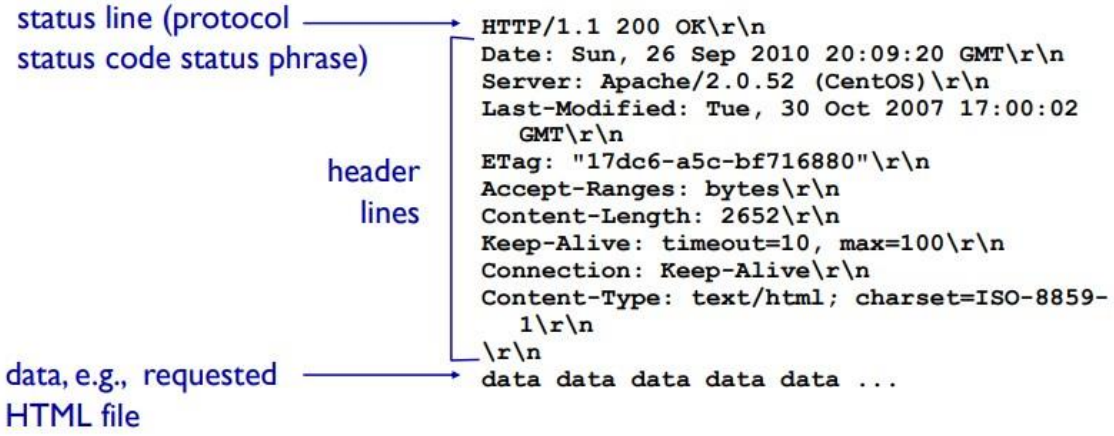
POST Method:
- Web page often includes form input
- User input sent from client to server in entity body of HTTP POST request message

GET Method: (for sending data to server)
- Include user data in URL field of a HTTP GET request message (following a '?'):

HEAD Method:
- Requests headers (only) that would be returned *if* specified URL were requested with a HTTP GET method.

PUT Method:
- Uploads new file (object) to server
- Completely replaces file that exists at specified URL with content in entity body of PUT HTTP request message

HTTP Response Message Format:



HTTP Response Status Codes:

- Status code appears in 1st line in Server-to-Client response message.
- Common codes:
  - **200 OK**
    - Request succeeded, requested object later in this message
  - **301 Moved Permanently**
    - Requested object moved, new location in `Location` field.
  - **400 Bad Request**
    - Request message not understood by server
  - **404 Not Found**
    - Requested document not found on this server
  - **505 HTTP Version Not Supported**

HTTP Is All Text *(Non text content needs to be encoded!)*

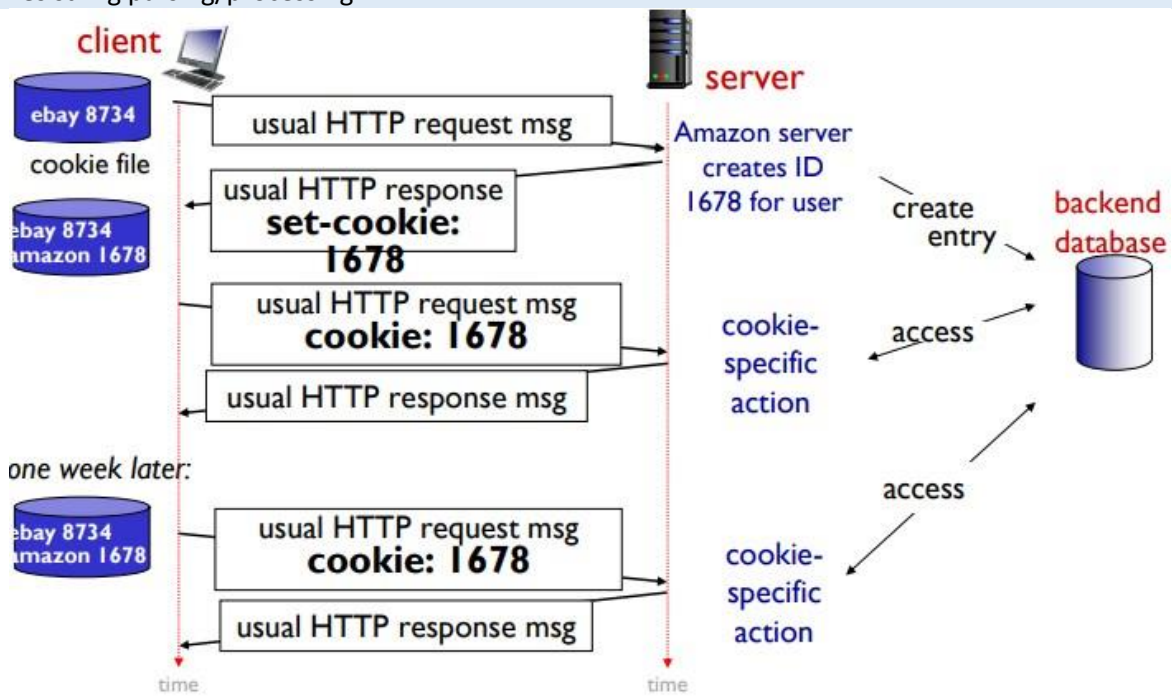
Makes protocol simple
- Easy to delineate messages (\r\n)
- Human readable
- No issues with encoding/formatting data
- Variable length data

However, not the most efficient 😟
- Many protocols use binary fields
- Sends these binary fields as integers, strings, etc
- Headers may come in any order
- Requires string parsing/processing

Maintaining User/Server State: *Cookies*

Recall: HTTP GET/response interaction is *stateless*

No notion of multi-step exchanges of HTTP messages to complete a Web 'transaction'
- No need for client/server to track 'state' of multi-step exchange
- All HTTP requests are independent of each other
- No need for client/server to 'recover' from a partially-completed-but-never-finished transaction.



What are Cookies?

Maintains some state between transactions

Four components:
1. Cookie header line of HTTP response message
2. Cookie header line in next HTTP request message
3. Cookie file kept on user's host, managed by user's browser
4. Back-end database at Web site

Example:
- Susan uses browser on laptop, visits specific e-commerce site for first time
- When initial HTTP requests arrives at site, the site creates:
  - Unique ID (aka 'cookie')
  - Entry in backend database for ID
- Subsequent HTTP requests from Susan to this site will contain cookie ID value, allowing site to 'identify' Susan.

HTTP Cookies: Comments

What cookies can be used for:
- Authorisation
- Shopping carts
- Recommendations
- User session state (Web e-mail)

Challenge- How to keep state:
- Protocol endpoints: maintain state at sender/receiver over multiple transactions
- Cookies: HTTP messages carry state

The Downside of Cookies:
- Cookies permit sites to learn a lot about you
- You may supply name and e-mail to sites (and more)
- 3rd party cookies (from ad networks, etc) can follow you across multiple sites
  - Example is visiting a site and all subsequent ads are from them.
- Far too entrenched in ecosystem of the web to outright disable.

Cookies and privacy:
- Cookies permit sites to learn a lot about you on their site.
- Third party persistent cookies (tracking cookies) allow common identity (cookie value) to be tracked across multiple web sites

Performance of HTTP

Page Load Time (PLT) is a key web performance metric.
Measured from click (or typing URL) until user sees page

Depends on many factors such as:
- Page content/structure
- Protocols involved
- Network bandwidth and RTT

Performance Goals

User:
- Fast downloads
- High availability

Content Provider:
- Happy users (hence, above)
- Cost-effective infrastructure

Network:
- Avoid overload

Solutions to improve PLT
- Improve HTTP (persistent connection, pipelining)
  - Faster downloads
  - Avoid repeated transfer of same content (ties into caching)
- Reduce content size for transfer (smaller images, *compression*)
- Caching and Replication (Improves availability and download speed)
- Webhosting, CDNs, data centers (Exploit economies of scale)
  - Move content closer to the client

HTTP Performance

Most Web pages have multiple objects
- E.g. HTML file and a bunch of embedded images

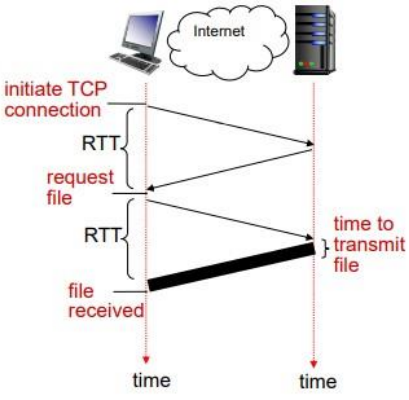
How do you retrieve those objects (naively)?
- *One item at a time*
- *New TCP connection per object!*

This is known as *Non-persistent HTTP*
- At most one object sent over TCP connection
- Connection then closed
- Downloading multiple objects requires multiple connections (huge overhead)

Non-persistent HTTP: Response time

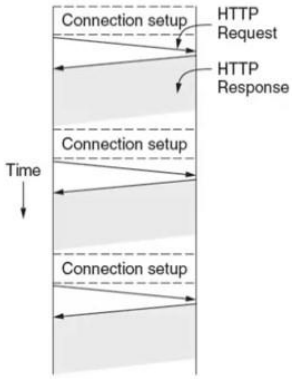
RTT (Return Trip Time) - Time taken for a packet to travel from client to server and back

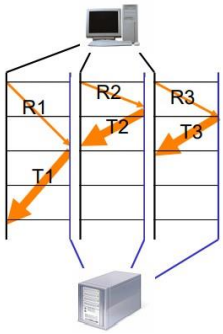
HTTP Response time:
- One RTT to init. TCP connection. (Approx. 3 way handshake)
- One RTT for HTTP request and first few bytes of response to return.
- File transmission time
- Non-persistent HTTP response time: $2 \cdot RTT + fileTransTime$



HTTP/1.0

- Non-Persistent: One TCP connection to fetch one web resource
- Fairly poor PLT
- Two scenarios:
  - Multiple TCP connections setups to the *same server*
  - Sequential request/responses even when resources are located on *different servers.*
- Multiple TCP slow-start phases.





Improving HTTP Performance:

Concurrent Requests & Responses

- Use multiple connections in *parallel*
- Does not necessarily maintain order of responses

Persistent HTTP (HTTP/1.1)

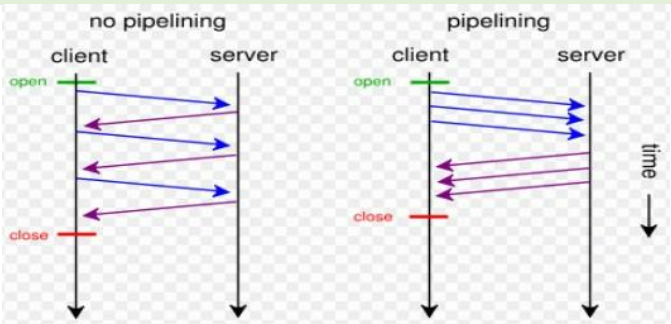
- Server leaves TCP connection open after sending response
- Subsequent HTTP messages between same client/server are sent over the same TCP connection
- Allow TCP to learn more accurate RTT estimate Allow TCP congestion window to increase
- Leverage previously discovered bandwidth

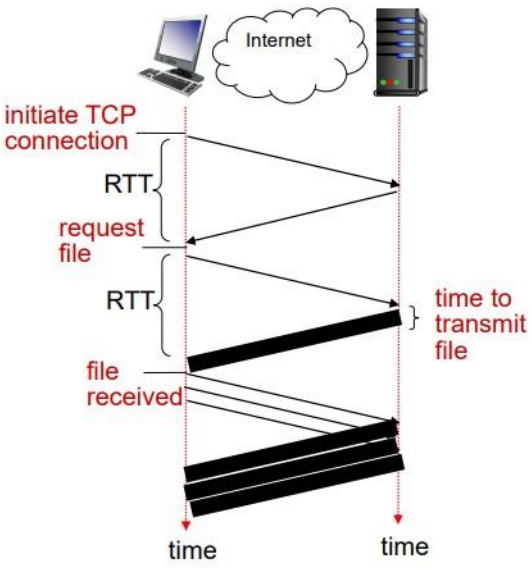
Persistent with pipelining:

- Introduced in HTTP/1.1
- Client sends requests as soon as it encounters a referenced object
- As little as one RTT for all the referenced objects

Persistent without pipelining:

- Client issues new request only when previous response has been received
- One RTT for each referenced object



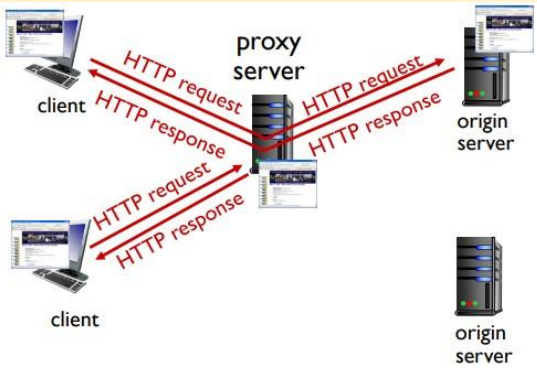


Improving HTTP Performance: Caching

Why? To exploit *locality of reference*
- Reduce traffic on access link
- Reduce response time for client request

Works well in many contexts… to an extent.

Trend: Increase in dynamic content
- E.g. customisation of web pages
- Reduces benefit of caching



Web Caches (Proxy Servers)

Goal: Satisfy client request without involving origin server

User configures browser to point to a Web cache
Browser sends all HTTP requests to cache
- **If** object in cache: cache returns object to client
- **Else** cache *requests* object from origin server, caches received object, then returns object to client.

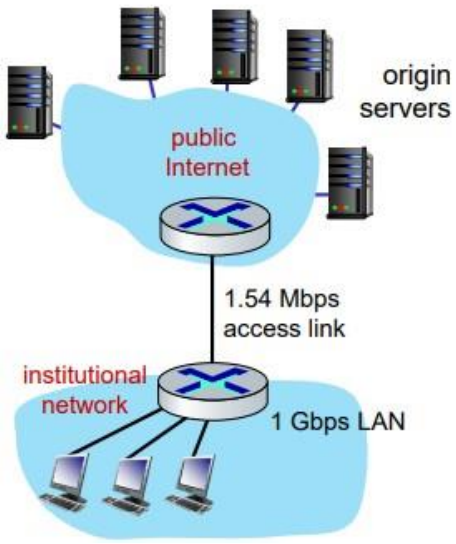
Web cache acts as both client and server
- Server for original requesting client
- Client to origin server

Typically, cache is installed by ISP (university, company, residential)

Caching example

Scenario:
- Access link rate: 1.54Mbps
- RTT from institutional router to server: 2 sec
- Web object size: 100k bits
- Average request rate from browsers to origin servers: 15/sec
- Average data rate to browsers: 1.50 Mbps

Performance:
- LAN utilization: 0.0015
- Access link utilisation = 0.97 *(large delays at high utilisation!)*
- End-to-end delay
  $= del_{Internet} + del_{accessLink} + del_{LAN}$
  = 2 sec + minutes + usecs

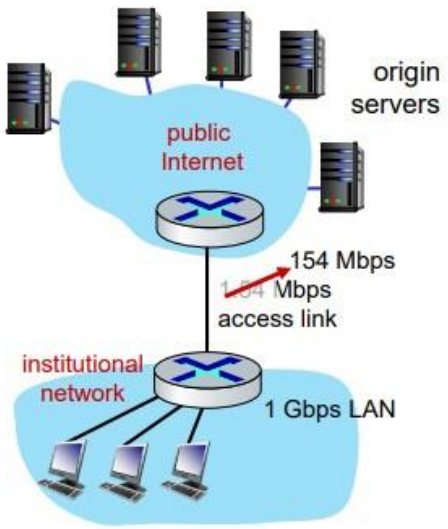


Caching example: buy a faster access link

Scenario:
- Access link rate: ~~1.54Mbps~~ 154Mbps
- RTT from institutional router to server: 2 sec
- Web object size: 100k bits
- Average request rate from browsers to origin servers: 15/sec
- Average data rate to browsers: 1.50 Mbps

Performance:
- LAN utilization: 0.0015
- Access link utilisation = ~~0.97~~ 0.0097
- End-to-end delay
  $= del_{Internet} + \sout{del_{accessLink}}\ msecs + del_{LAN}$
  = 2 sec + ~~minutes~~ msecs + usecs

Cost: faster access link (expensive!)



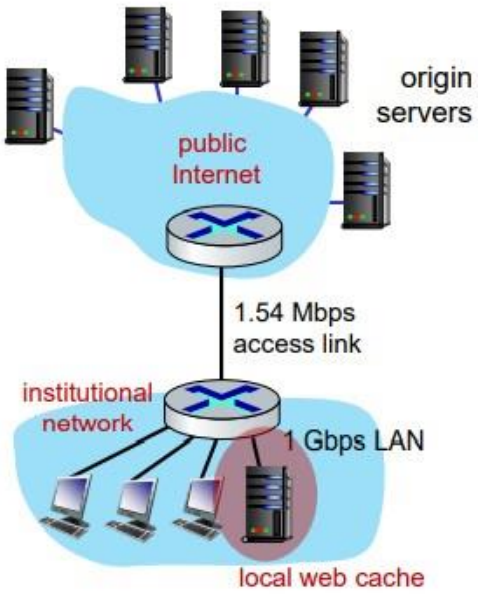
Caching example: Install a Web Cache

Scenario:
- Access link rate: 1.54Mbps
- RTT from institutional router to server: 2 sec
- Web object size: 100k bits
- Average request rate from browsers to origin servers: 15/sec
- Average data rate to browsers: 1.50 Mbps

Performance:
- LAN utilization: ?
- Access link utilisation = ?
- End-to-end delay = ?

Assuming cache hit rate of 0.4 (40% requests satisfied by cache):
- Access link used for 60% of requests
- Data rate over access link = 0.6 * 1.50Mbps = 0.9Mbps
- Utilisation = 0.9/1.54 = 0.58
- Average end-to-end delay
  = 0.6 * (delay from origin serers)
  + 0.4 *(delay when satisfied at cache)
  = 0.6 (2.01) + 0.4 (~msecs) = ~1.2sec

Result?
Lower average end-to-end delay than with 154 Mbps link – also Cheaper!



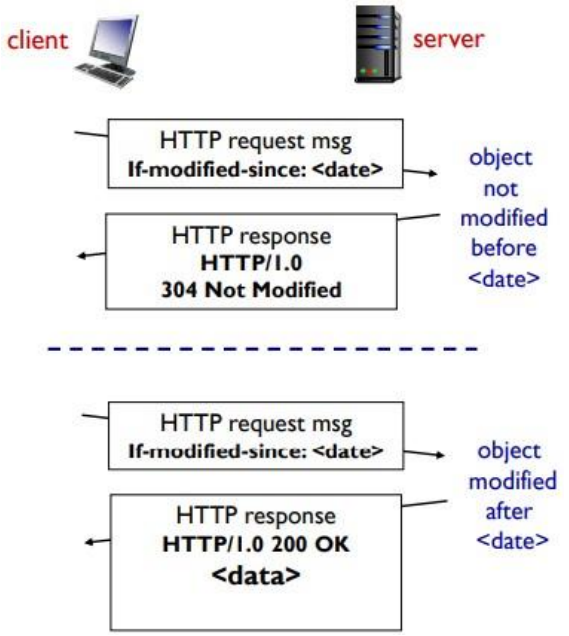
Conditional GET

Goal: Don't sent object if cache has up-tdo-date cached version
- No object transmission delay
- Lower link utilization

Cache: Specify date of cached copy in HTTP request.
`If-modified-since:<date>`

Server: Response contains no object if cached copy is up-to-date:
`HTTP/1.0 304 Not Modified`

Below: Cache check request(left) and response(right)




```
GET / HTTP/1.1
Accept: */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
If-Modified-Since: Mon, 29 Jan 2001 17:54:18 GMT
If-None-Match: "7a11f-10ed-3a75ae4a"
User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows NT
  5.0)
Host: www.intel-iris.net
Connection: Keep-Alive
```

```
HTTP/1.1 304 Not Modified
Date: Tue, 27 Mar 2001 03:50:51 GMT
Server: Apache/1.3.14 (Unix)  (Red-Hat/Linux) mod_ssl/2.7.1
  OpenSSL/0.9.5a DAV/1.0.2 PHP/4.0.1pl2 mod_perl/1.24
Connection: Keep-Alive
Keep-Alive: timeout=15, max=100
ETag: "7a11f-10ed-3a75ae4a"
```

Etag: Usually used for dynamic content. The value is often a cryptographic hash of the content.

Strategies for Improving HTTP Performance

Replication:

Replicate popular Web site across many machines
- Spreads load on servers
- Places content closer to clients
- Helps when content isn't cacheable

Problem:
- Want to direct client to a particular replica
  - Balance load across server replicas
  - Pair clients with nearby servers
- Expensive

Common solution:
- DNS returns different addresses based on client's geo-location, server load, etc.

CDN:

Caching and replication as a *service.*

Large-scale distributed storage infrastructure (usually) administered by one entity
- E.g. Akamai has servers in 20,000+ locations

Combination of (pull) caching and (push) replication
- **Pull:** Direct result of client requests
- **Push**: Expectation of high access rate

Also do some processing
- Handle dynamic web pages
- Transcoding

FCFS: First-come-first-serve (queuing strategy)

HOL: Head-Of-Line (leading queue)

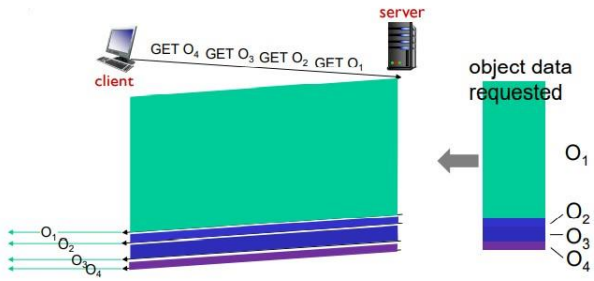
What about HTTPS?

- HTTP is *insecure*
- HTTP basic authentication: password sent using base64 encoding
  - Can be readily converted to plaintext
- **HTTPS**: HTTP over connection encrypted by Transport Layer Security (TLS)
  - Provides *authentication* and *bidirectional encryption*
- Widely used in place of plain vanilla HTTP

Problem with HTTP/1.1? Mitigating HOL blocking

Key goal: decreased delay in multi-object HTTP requests

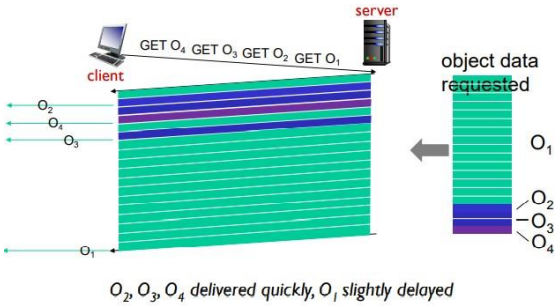
HTTP/1.1: Introduced multiple, pipelined GETs over single TCP connection
- Server responds in-order (FCFS scheduling) to GET requests
- With FCFS, small object may have to wait for transmission (HOL blocking) behind large object(s)
- Loss recovery (retransmitting lost TCP segments) stalls object transmission



HTTP/2

Key goal: decreased delay in multi-object HTTP requests

HTTP/2: [RFC 7540, 2015] increased flexibility at *server* in sending objects to client:
- Methods, status codes, most header fields *unchanged* from HTTP/1.1
- Transmission order of requested objects based on client-specified object priority (*not necessarily FCFS*)
- Push unrequested objects to client
- Divide objects into frames, schedule frames to mitigate HOL blocking

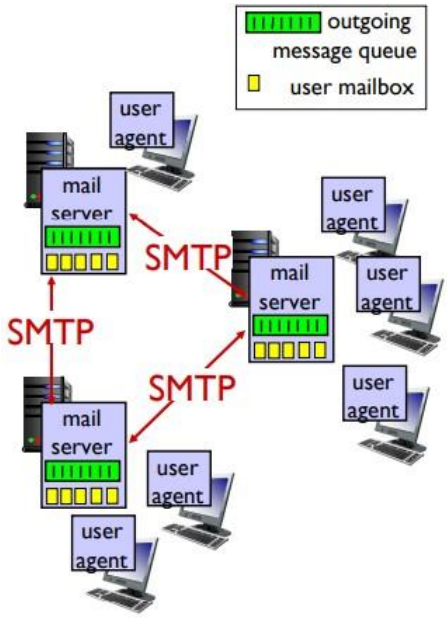


O_2, O_3, O_4 delivered quickly, O_1 slightly delayed

Application Layer: Electronic Mail

Three major components:

- User Agents
- Mail servers
- Simple Mail Transfer Protocol (SMPT)



User Agent

- A.k.a "mail reader"
- Composing, editing, reading mail messages
- E.g. Outlook, iPhone mail client, Gmail app
- Outgoing, incoming messages stored on server

Mail Servers:

- *Mailbox* contains incoming messages for user
- *Message queue* of outgoing (to be sent) mail messages
- *SMTP protocol* between mail servers to send email messages ('client/server' often refers to sender/receiver)

E-mail: the RFC (5321)

- Uses TCP to reliably transfer email message from client (mail server initiating server) to server, port 25
- Direct transfer: sending server (acting like client) to receiving server)
- Command/response interaction (like HTTP)
  - Commands: ASCII text
  - Response: status code and phrase
- Messages must be in 7-bit ASCI
- Three phases of transfer
  - Handshaking (greeting)
  - Transfer of messages
  - Closure

SMTP: Closing Observations

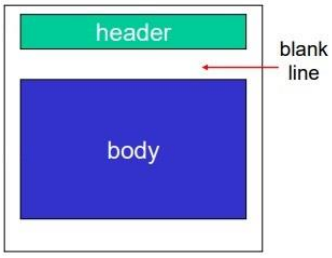
Comparison with HTTP:

- *HTTP*: pull
- *SMTP*: push
- Both have ASCII command/response interaction, status codes
- *HTTP:* each object encapsulated in its own response message
- *SMTP*: multiple objects sent in multipart message
- *SMTP* uses persistent connections
- *SMTP* requires message (header / body) to be in 7-bit ASCII
- *SMTP* server uses CRLF.CRLF to determine end of message

Mail message format

Syntax for e-mail message:

- Header lines (`To:`, `From:`, `Subject:`, etc)
  - These lines differ to `SMTP MAIL FROM:`, `RCPT TO:` commands
- Blank line
- Body (Message of email, ASCII characters only)



HTTP/2 to HTTP/3

Key goal: decreased delay in multi-object HTTP requests

HTTP/2 over a single TCP connection means:

- Recovery from packet loss still stalls all object transmissions
- No security over vanilla TCP connection

HTTP/3: adds security, per object error and congestion control (more pipelining) over UDP

Domain Name System (DNS)

People: many identifiers:
- TFN, name, passport #, etc
-

Internet hosts, routers:
- IP address (32 bit) – used for addressing datagrams

"Name", e.g. cs.unmass.edu – used by humans

Domain name system

Distributed database implemented in many *name servers*
Application-layer protocol: hosts, name servers communicate to *resolve* names (address/name translation)

Complexity at network's "edge" (adds scalability)
– host-address mappings used to be maintained by a single department at Stanford Research Institute, changes sent via email!

DNS: Services, Structure

DNS Services:
- Hostname to IP address translation
- Host aliasing (canonical, alias names)
- Mail server aliasing
- Load distribution
  - Replication Web servers - many IP addresses correspond to one name

Q: Why not centralise DNS?
- Single point of failure
- Traffic volume
- Distance centralised databsae
- Maintenance

A: Huge scaling issues
- Comcast DNS servers alone process *600 billion* DNS queries per day

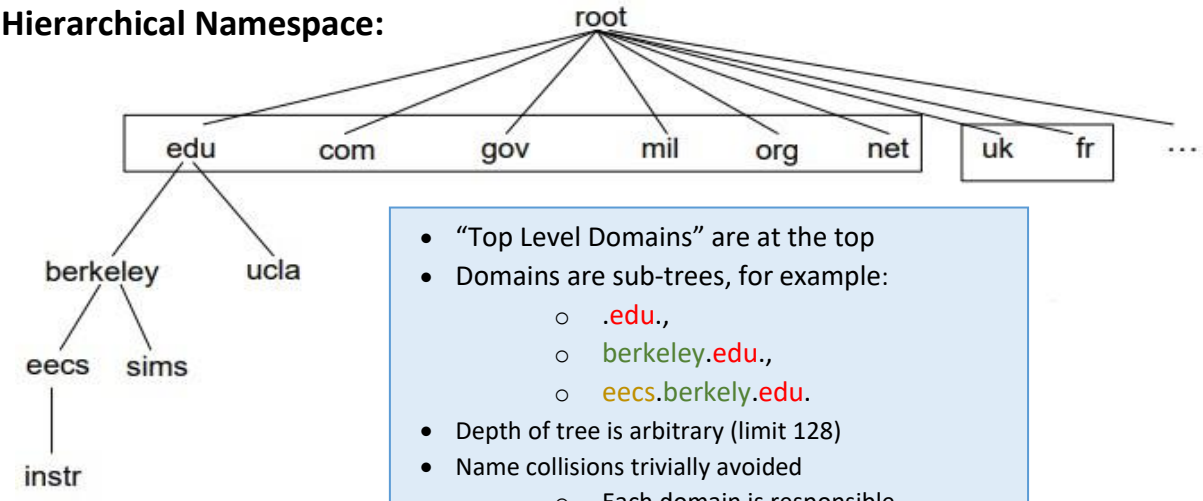
Goals:
- No naming conflicts (uniqueness)
- Scalable
  - Many names, frequent updates
- Distributed, *autonomous* administration
  - Ability to update my own domains' names
  - Don't have to track everybody's updates
- Highly available
- *Fast*

Key Idea - Hierarchy

Three intertwined hierarchies

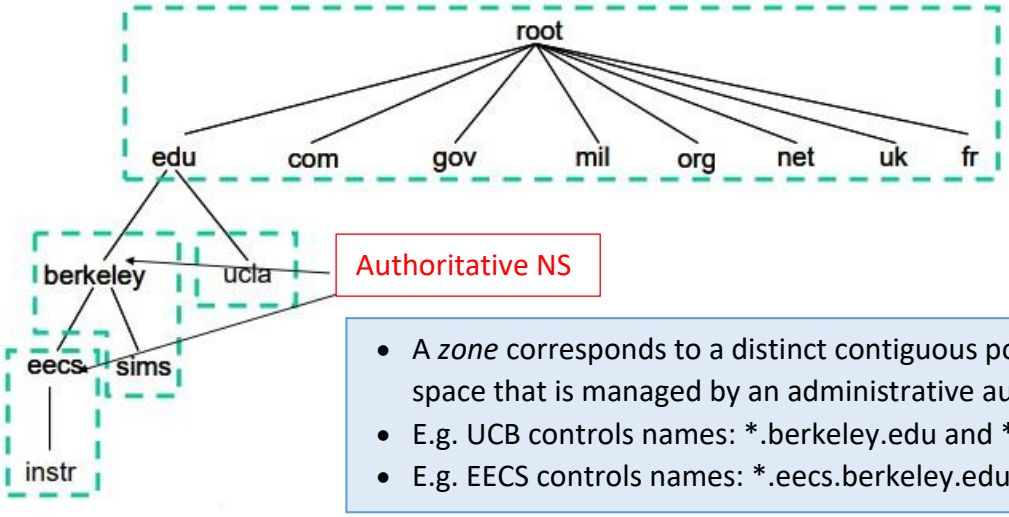
1. Hierarchical namespace
   - As opposed to original flat namespace
2. Hierarchically administered
   - As opposed to centralised
3. Distributed hiearchy of servers
   - As opposed to centralised storage

Hierarchical Namespace:



- "Top Level Domains" are at the top
- Domains are sub-trees, for example:
  - .edu.,
  - berkeley.edu.,
  - eecs.berkely.edu.
- Depth of tree is arbitrary (limit 128)
- Name collisions trivially avoided
  - Each domain is responsible

Hierarchical Administration



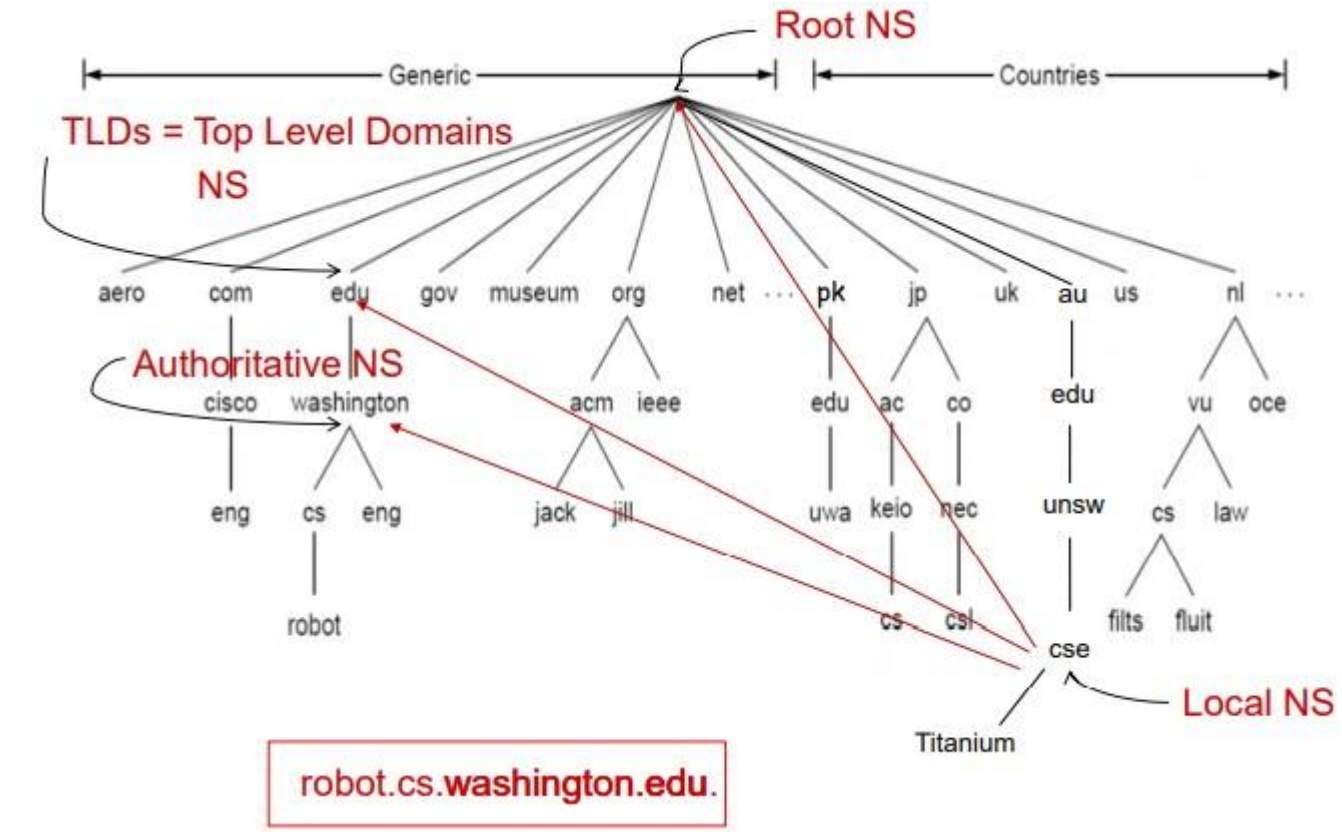
- A *zone* corresponds to a distinct contiguous portion of the DNS name space that is managed by an administrative authority
- E.g. UCB controls names: *.berkeley.edu and *.sims.berkeley.edu
- E.g. EECS controls names: *.eecs.berkeley.edu

Server Hierarchy

- Top of hierarchy: Root servers
  - Location hardwired into other servers
- Next Level: Top-level domain (TLD) servers
  - .com, .edu, etc (several new TLDs introduced recently)
  - Managed professionally
- Bottom Level: *Authoritatative* DNS servers
  - Store the name-to-address mapping
  - Maintained by the corresponding administrative authority

- Each server stores a (small) subset of the total DNS database
- An authoritative DNS server stores "resource records" for all DNS names in the domain that it has authority for
- Each server can discover the server(s) that are responsible for the other portions of the hierarchy
  - Every server knows the root server(s)
  - Root server(s) know about all the top-level domains

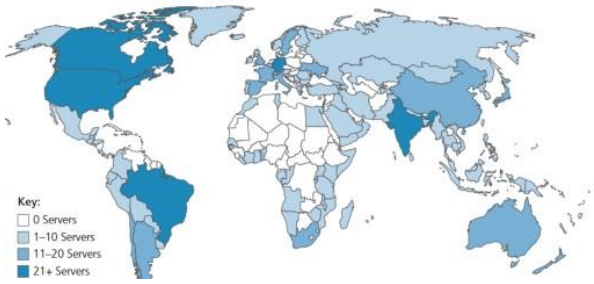
DNS: A distributed, hierarchical database



DNS: Root name servers

- Official, contact-of-last-resort by name servers that can not resolve name
- *Incredibly important* Internet function
- Internet couldn't function without root servers
- DNSSEC – provides security (authentication and message integrity)
- ICANN (Internet Corporation for Assigned Names and Numbers)
- Manages root DNS domain

13 logical root name "servers" worldwide each "server" replicated many times (~200 servers in US)



TLD: Authoritative Servers

Top-Level Domain (TLD) servers:

- Responsible for .com, .org, .net, .edu, .aero, .jobs, .museums, and all top-level country domains, e.g. .cn, .uk, .fr, .ca, .jp
- Network Solutions: authoritiative registry for .com, .net TLD
- Educause: .edu TLD

Authoritative DNS servers:

- Organisations' own DNS server(s), providing authoritative hostname to IP mappings for organisation's named hosts
- Can be maintained by organisation or service provider

Local DNS name Servers

- Each ISP (residential, company, university) has one (also called "default name server"
- Hosts learn about the local DNS server via a host configuration protocol (e.g. DHCP)
- Client application
  - Obtain hostname (e.g. from URL)
  - `gethostbyname()` to trigger DNS request to its local DNS server
- When the host makes DNS query, query is sent to its local DNS server.
- Has local cache of recent name-to-address translation pairs *(but may be out of date!)*
- Acts as proxy, forwards query into hierarchy
- *Does not strictly belong to heirarchy*

DNS Name resolution: Iterated Query vs Recursive Query

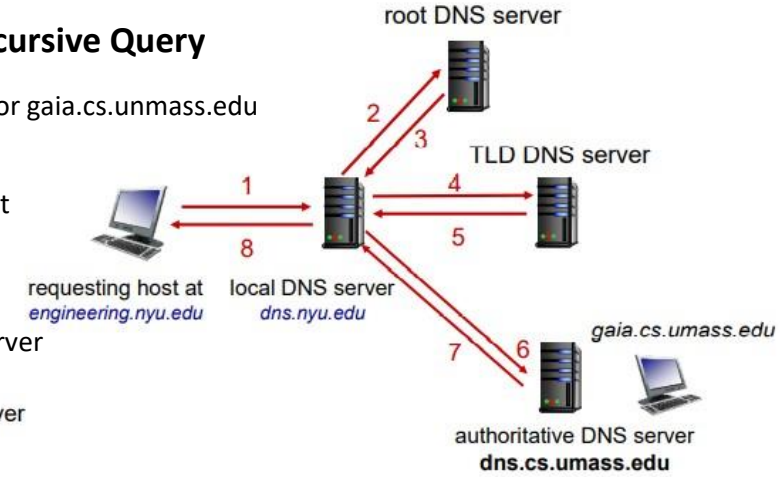
Example: host at engineering.nyu.edu wants IP address for gaia.cs.unmass.edu

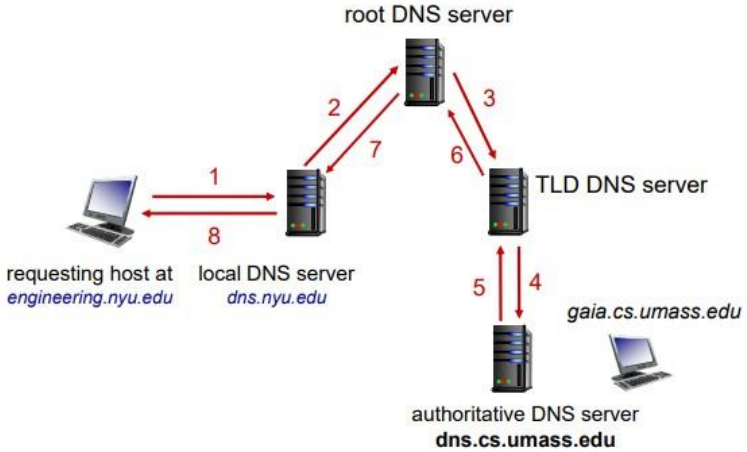
Iterated Query (Right):

- Contacted server replies with name of server to contact
- "I don't know this name, but ask this server"

Recursive Query (below):

- Puts burden of name resolution on contacted name server
- Heavy load at uper levels of heirarchy





Caching, Updating DNS Records

- Once (any) name server learns mapping, it *caches* mapping
  - Cache entries timeout (disappear) after some time (TTL)
  - TLD servers typically cached in local name servers
    - Thus root name servers not often visited
- Cached entries may be out-of-date (best-effort name-to-address translation!)
  - If name host changes IP address, may not be known Internet-wide until all TTLs expire
- Update/notify mechanisms proposed IETF standard *[RFC 2136]*
- Negative caching (optional)
  - Remember things that don't work
  - E.g. Misspellings like www.cnn.comm and www.cnnn.com

DNS Records

DNS: Distributed database storing resource resource records (**RR**)

RR format: (`name, value, type, ttl`)

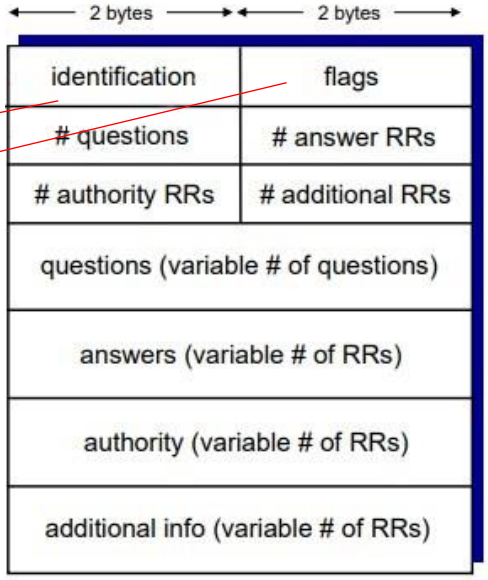
Type=A - `name` is hostname - `value` is IP address	**Type=CNAME** - `name` is alias name for some "canonical" (the real) name. - e.g. www.ibm.com is really servereast.backup2.ibm.com - `value` is canonical name
Type=NS - `name` is domain (e.g. foo.com) - `value` is hostname of authoritative name server for this domain	**Type=MX** - `value` is name of mailserver associated with `name`

DNS Protocol messages:

DNS *query* and *reply* messages, both have same *format:*

Message header:

- **Identification**: 16 bit # for query, reply to query uses same #
- **Flags**:
  - query or reply
  - recursion desired
  - recursion available
  - reply is authoritative



Inserting Records Into DNS

Example: new startup "Network Utopia"

- Register name networkutopia.com at *DNS registrar* (e.g. Network Solutions)
  - Provide names, IP addresses of authoritative name server (primary and secondary)
  - Registrar inserts NS, A RRs into .com TLD server:
    (`networkutopia.com, dns1.networkutopia.com, NS`)
    (`dns1.networkutopia.com, 212.212.212.1, A`)
- Create authoritative server locally with IP address `212.212.212.1`
  - Containing type `A` record for www.networkutopia.com
  - Containing type `MX` record for www.networkutopia.com

Updating DNS records

Remember that old records may be cached in other DNS servers (for up to TTL)

General guidelines:

- Record the current TTL value of the record
- Lower the TTL of the record to a low value (e.g. 30 sec)
- Wait the length of the previous TTL
- Update the record
- Wait for some time (e.g. 1 hour)
- Change the TTL back to your previous time

Many well known sites are hosted by CDNs
See '`ANSWER SECTION`' of dig command below:

Reliability

DNS servers are *replicated* (primary/secondary)

- Name service available if at least one replica is up
- Queries can be load-balanced between replicas

Usually, UDP used for queries

- Need reliability: must implement *on top* of UDP
- Spec supports TCP too, but not always implemented

DNS uses port *53.*

On timeout, try alternate servers

- Exponential backoff when retrying same server

Same identifer for all queries

- Don't care which server responds.

```
bash-3.2$ dig www.mit.edu

; <<>> DiG 9.10.6 <<>> www.mit.edu
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 17913
;; flags: qr rd ra; QUERY: 1, ANSWER: 3, AUTHORITY: 8, ADDITIONAL: 8

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;www.mit.edu.                   IN      A

;; ANSWER SECTION:
www.mit.edu.            924     IN      CNAME   www.mit.edu.edgekey.net.
www.mit.edu.edgekey.net. 54     IN      CNAME   e9566.dscb.akamaiedge.net.
e9566.dscb.akamaiedge.net. 14   IN      A       23.77.154.132

;; AUTHORITY SECTION:
dscb.akamaiedge.net.    623     IN      NS      n0dscb.akamaiedge.net.
dscb.akamaiedge.net.    623     IN      NS      n2dscb.akamaiedge.net.
dscb.akamaiedge.net.    623     IN      NS      n7dscb.akamaiedge.net.
dscb.akamaiedge.net.    623     IN      NS      n6dscb.akamaiedge.net.
dscb.akamaiedge.net.    623     IN      NS      n1dscb.akamaiedge.net.
dscb.akamaiedge.net.    623     IN      NS      n3dscb.akamaiedge.net.
dscb.akamaiedge.net.    623     IN      NS      n5dscb.akamaiedge.net.
dscb.akamaiedge.net.    623     IN      NS      n4dscb.akamaiedge.net.

;; ADDITIONAL SECTION:
n0dscb.akamaiedge.net.  1241    IN      A       88.221.81.192
n0dscb.akamaiedge.net.  1124    IN      AAAA    2600:1480:e800::c0
n1dscb.akamaiedge.net.  842     IN      A       23.32.5.76
n2dscb.akamaiedge.net.  749     IN      A       23.32.5.84
n4dscb.akamaiedge.net.  1399    IN      A       23.32.5.177
n6dscb.akamaiedge.net.  702     IN      A       23.32.5.98
n7dscb.akamaiedge.net.  1208    IN      A       23.206.243.54

;; Query time: 46 msec
;; SERVER: 129.94.172.11#53(129.94.172.11)
;; WHEN: Mon Sep 28 13:15:28 AEST 2020
;; MSG SIZE  rcvd: 421
```

DNS Security

DDOS Attacks

Bombard root servers with traffic:

- Not successful to date
- Traffic filtering
- Local DNS servers cache Ips of TLD servers, allowing root server to bypass

Bombard TLD servers:

- Potentially more dangerous

Redirect attacks

Man-in-the-middle:

- Intercept DNS queries

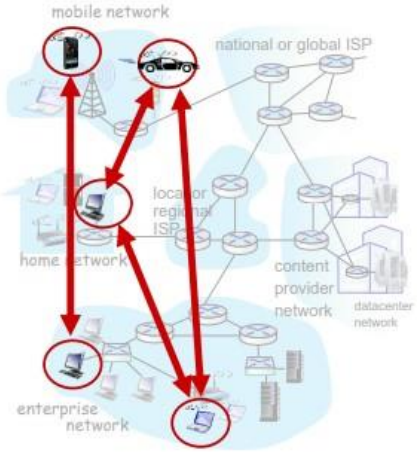
DNS poisoning:

- Send bogus replies to DNS server, which *caches*
- Solution: Only allow authoritative name servers to cache IP address mappings

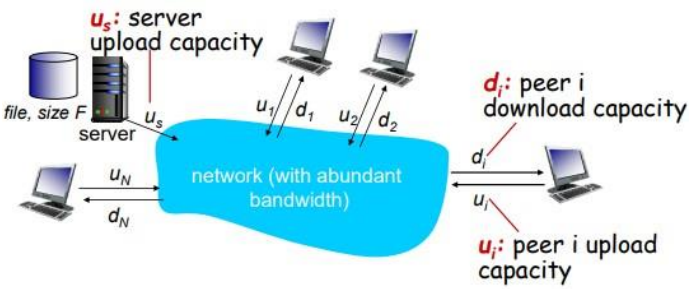
Exploit DNS for DDOS

- Send queries with spoofed source address: target IP
- Requires amplification

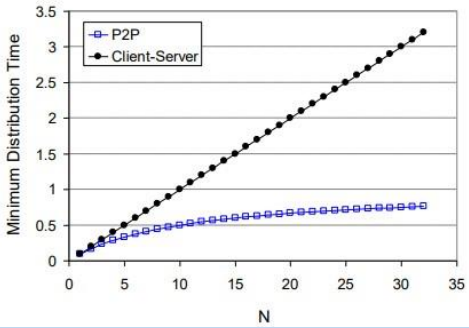
Peer-to-Peer (P2P) architecture



- No always-on server
- Arbitrary end systems directly communicate
- Peers request service from other peers, provide service in return to other peers
  - Self scalability – new peers bring new service capcity, and new service demands
- Peers are intermittently connected and change IP addresses
- Examples: P2P file sharing (BitTorrent), Streaming (KanKan), VoIP (Skype), Cryptocurrency (Bitcoin)



client upload rate = u, F/u = 1 hour, $u_s = 10u$



File Distribution Time: Client-Server

Server transmission: must sequentially send (upload) N file copies

- Time to send one copy: $\frac{F}{u_s}$
- Time to send N copies: $\frac{NF}{U_s}$

Client: each client must download file copy

- d_{min} = min client download rate
- Slowest client download time: F/d_{min}

Time to distribute F to N clients using client-server approach:

$$D_{c-s} \geq \max\left(\frac{NF}{U_s}, \frac{F}{D_{min}}\right)$$

Increases *linearly* in N

File Distribution Time: P2P

Server transmission: must upload at least *one* copy

- Time to send one copy: $\frac{F}{u_s}$

Client: each client must download file copy

- Slowest client download time: F/d_{min}

Clients: as aggregate must download NF bits

- Max upload rate (limiting max download rate):

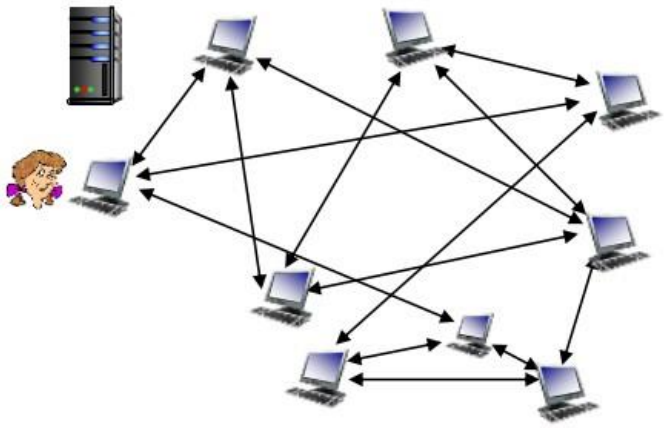
$$u_s + \sum u_i$$

Time to distribute F to N clients using client-server approach:

$$D_{P2P} \geq \max\left(\frac{F}{U_s}, \frac{F}{D_{min}}, \frac{NF}{u_s + \sum u_i}\right)$$

Increases linearly in N, but each peer brings capacity

P2P File Distribution: BitTorrent



- File divided into 256Kb chunks
- Peers in torrents send/receive file chunks
- *Tracker:* tracks peers participating in torrent
- *Torrent*: group of peers exchanging chunks of a file

Order of events:

1. User arrives
2. Obtains list of peers from tracker
3. Connect to subset of peers('neighbours')
4. Accumulates chunks from peers, subsequently uploading these chunks to other peers who require.

Peer may change peers with whom it exchanges chunks.
Once a peer has an entire file, it may (selfishly) leave or (altruistically) remain in torrent.
This notion of peers coming and going is known as *churn*.

Requesting, sending file chunks

Requesting chunks:

- Any any time, different peers have different subset of file chunks
- Periodically, *user* asks each peer for a list of chunks that they have
- *User* requests missing chunks from peers, rarest first. This is in recognition that peers may abruptly leave, resulting in no active subset of users to collectively hold the entire file.

Sending chunks: tit-for-tat

- *User* sends chunks to those four peers currently sending her chunks at *highest* rate.
- Other peers are 'choked' – do not receive chunks
- Re-evaluate top 4 periodically (every 10 seconds)
- Every 30 seconds, randomly select ('unchoke') another peer.
- New peer may join top 4.

General idea: incentivise high upload rates, avoid stagnation by shuffling neighbour peers.

Distributed Hash Table (DHT)

- DHT: a *distributed P2P database*
- Database has (`key, value`) pairs: example:
  - `key`: TFN number; `value`: human name
  - `key`: file name; `value`: IP addresses of peers (BitTorrent Tracker)
- Distribute the (`key, value`) pairs over many peers
- A peer *queries* DHT with key
  - DHT returns values that match the key
- Peers can also *insert* (`key, value`) pairs

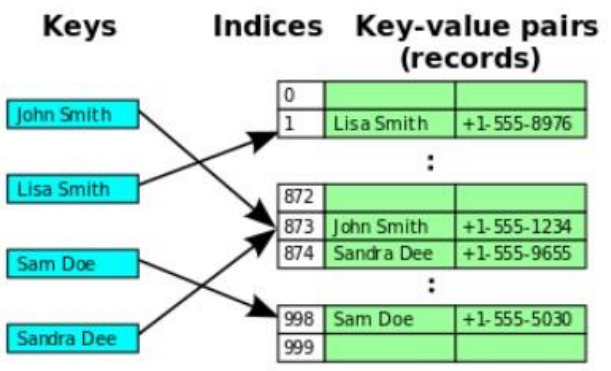
Q: How to assign keys to peers?

Basic idea:

- Convert each key to an integer
- Assign integer value to each peer
- Put (`key, value`) pair in the peer that is *closest* to the key

DHT Identifiers: Consistent Hashing

- Assign integer identifer to each peer in range $[0, 2^n - 1]$ for some *n*-bit hash function
  - E.g. node ID is hash of its IP address
- Require each key to be an integer in **same range**
- To get integer key, hash original key
  - E.g. `key = hash("The Boys Season 2")`
  - Therefore, it is referred to as a *distributed "hash" table*



Assign Keys to Peers

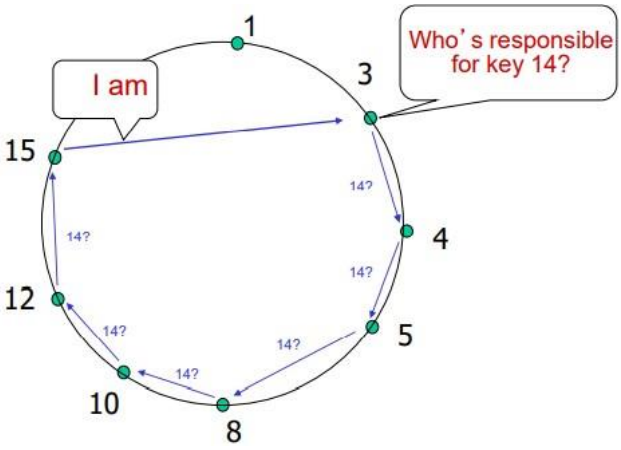
- Rule: assign key to the peer that has the *closest* ID.
- Common convention: closest is the *immediate successor* of the key.
- E.g. `n = 4`; all peers & key identifiers are in the range `[0-15]`, peers: `1,3,4,5,8,10,12,14`;
  - `key = 13`, then `successorPeer = 14`
  - `key = 15`, then `successorPeer = 1`

Q: How is the P2P network organised?

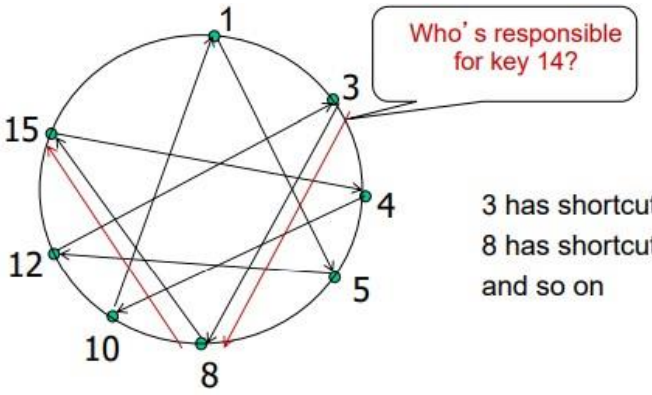
One way would be to require each peer to be aware of every other peer, but this would not scale

Circular DHT

- Each peer only aware of immediate succesor and predecessor
  - Each peer maintains 2 neighbours
- "Overlay network"
- Queries typically propagate in clockwise direction
  - Worse case: N messages, Average: $\frac{N}{2}$ messages
- Example: 6 query messages are sent



Circular DHT with shortcuts

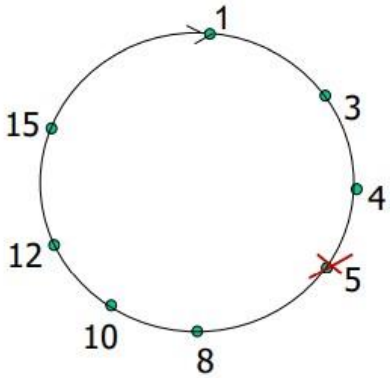


3 has shortcut to 8
8 has shortcut to 15
and so on

Each peer keeps track of IP addresses of predecessors, successor, and *shortcuts*

Possible to design shortcuts so $O(\log N)$ neighbours, $O(logN)$ messages in query

Peer Churn (Peers coming and going)



example: peer 5 abruptly leaves

Handling peer churn:

- Each peer knows address of its two successors
- Each peer periodically pings its two successors to check aliveness
- If immediate successor leaves, choose next successor as new immediate

Example:

- Peer 4 detects peer 5 departure
- Makes 8 immediate successor
- Asks 8 who its immediate successor is
- Makes 8's immediate succesor its second successor.

Application Layer: Video Streaming and Content Distribution Networks (CDNs)

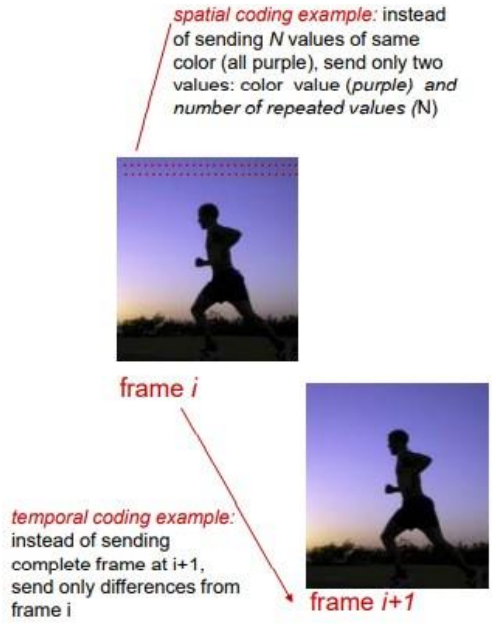
Stream video traffic is a major consumer of Internet bandwidth

- Netflix, Youtube, Amazon Prime account for 80% of residential ISP traffic (2020)

Distributed, application-level infrastructure is the preferable scalable solution!

Multimedia: Video

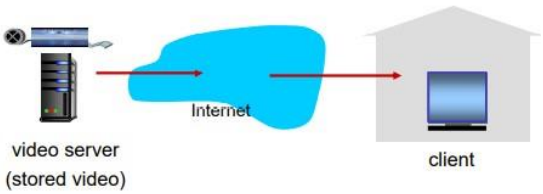
- Video: sequence of images displayed at constant rate
- Digital image: array of pixels (each represented by bits)
- Coding: use redundancy *within* and *between* images to decrease #bits used to encode image
  - Spatial (within image)
  - Temporal (between neighbouring images)
- CBR (Constant Bit Rate): video encoding rate fixed
- VBR (Variable Bit Rate): video encoding rate changes as amount of spatial, temporal coding changes
- Examples:
  - MPEG 1 (CD-ROM) 1.5Mbps
  - MPEG2 (DVD) 3-6Mbps
  - MPEG4 (often used in Internet) 64Kbps-12Mbps



Streaming stored video

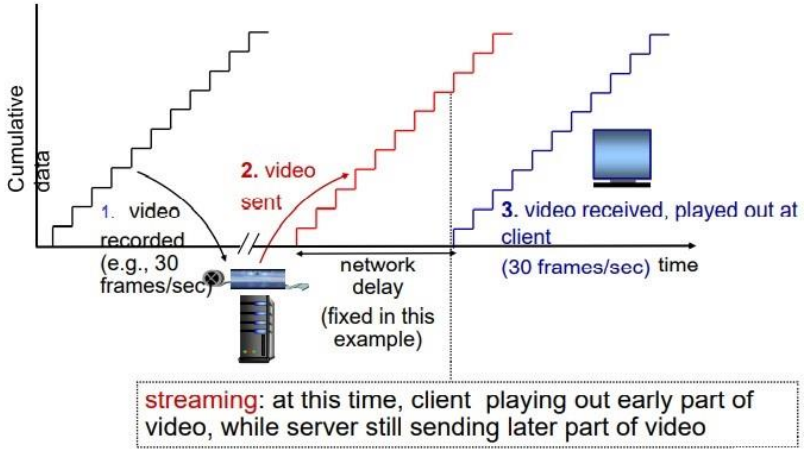
Main challenge - Packet loss and delay due to congestion:

- Server-to-client bandwidth will vary over time, with changing network congestion levels (in house, in access network, in network core, at video server, etc)
- Will delay playout, or result in poor video quality



Solution – *client-side buffer*

Compensates for network-added delay and delay jitter (variable network delay)



Other challenges faced?

Client Interactivity

- Pause, fast-forward, rewind, jump

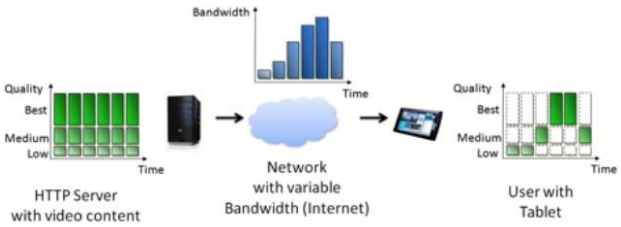
Video packets may be lost, retransmitted.

Streaming multi-media: DASH (Dynamic, Adaptive, Streaming over HTTP)

Server	**Client**
Divides video file into multiple chunks Each chunk stored, encoded at different rates *Manifest file*: provides URLs for different chunks	Periodically measures server-to-client bandwidth Consulting manifest, requests one chunk at a time - Chooses maximum coding rate sustainable given current bandwidth - Can choose different coding rates at different points in time (bandwidth permitting)

"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 bandwidth available)
- *Where* to request chunk (can request from URL server that is "close" to client or has high available bandwidth

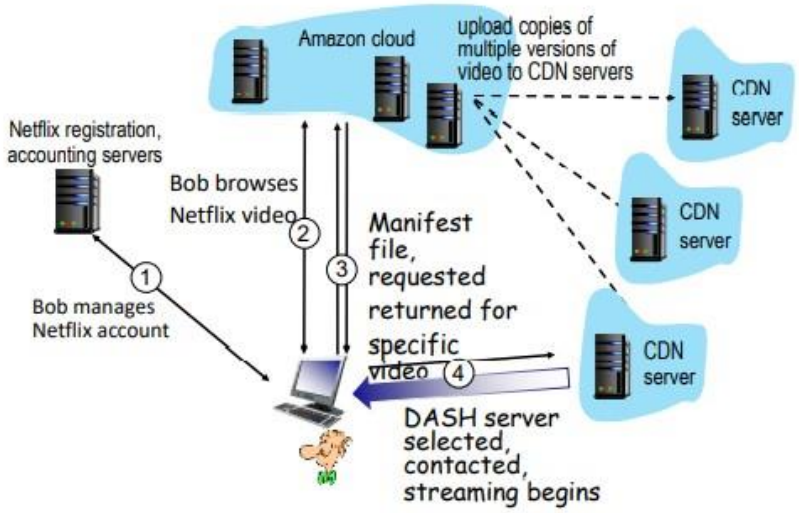
Streaming video = encoding + DASH + playout buffering



Content Distribution Networks (CDNs)

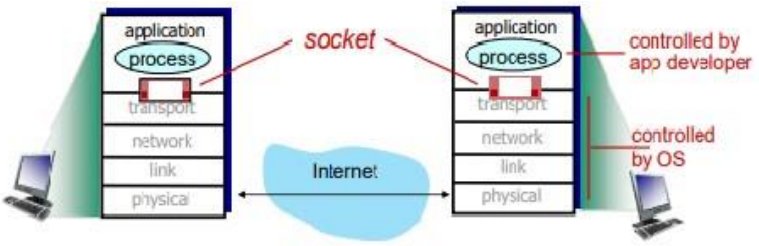
In a nutshell: Geographically distributed sites to store/serve multiple copies of popular content.
- Close to users, reduces network delays when accessing content
- When a user requests content, they are redirected to a CDN
    - They may choose a different CDN if path is congested.

Case study: Netflix



Socket Programming

Socket: Door between application process and end-to-end transport protocol



Socket programming *with UDP*

UDP: no "connection" between client & server

- No handshaking before sending data
- Sender explicitly attaches IP destination address and port # to each packet
- Receiver extracts sender IP address port# and received packet
- *Transmitted data may be lost or received out-of-order*

Application viewpoint:

- UDP provides *unreliable* transfer of groups of bytes ("segments") between client and server.

Pseudo code for UDP Client	Pseudo code for UDP Server

- Create socket
- Loop
  ○ Send UDP segment to known port/IP of server
  ○ Receive UDP segment as response from server
- Close socket
	- Create socket
	- Bind socket to designated port so clients can contact
	- Loop
	  ○ (*) Receive UDP segment from client X
	  ○ Send UDP segment as reply to client X
	- Close socket
	
	*Note: IP and port of client must be extracted from client message.

Socket programming *with TCP*

Client must contact server

- Server process must first be running
- Server must be created socket (door) that welcomes client's contact

Client contacts server by:

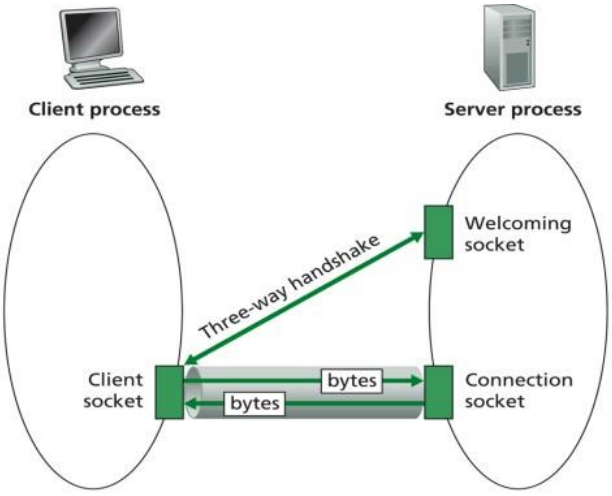
- Creating TCP socket, specifying IP address, port number of server process
- *When client creates socket:* client TCP establishes connection to server TCP

When contacted by client, TCP server creates new socket for server process to communicate with that client

- Allows server to talk with multiple clients
- Source port numbers used to distinguish clients

Application viewpoint:

TCP provides reliable, in-order byte-stream transfer ("pipe") between client and server



Pseudo code for TCP Client	Pseudo code for TCP Server

- Create socket (ConnectionSocket)
- Do an active connect specifying the IP address and port number of server
- Read and write data into ConnectionSocket to communicate with client
- Close socket
	- Create socket (WelcomingSocket)
	- Bind socket to designated port so clients can contact
	- Register with OS willingness to listen on that socket.
	- Loop
	  ○ Accept new connection (ConnectionSocket)
	  ○ Read and write data into ConnectionSocket to communicate with client
	  ○ Close ConnectionSocket
	- Close WelcomingSocket

Queues

- While the server socket is busy, incoming connection requests are stored in a queue
- Once the queue fills up, further incoming connections are refused
- This is clearly a problem (e.g. HTTP servers)
- *Solution* – Concurrency

Concurrent TCP Servers

- Benefit comes in ability to hand off interaction with a client to another process
- Parent process creates the WelcomingSocket and waits for clients to request connection
- When a connection request is received, fork off a child process to handle that connection so that the parent process can return to waiting for connections as soon as possible
- Multithreaded server: same idea, just spawn off another thread rather than a process.

Summary – Application Layer

- Application architectures
  - Client-Server
  - P2P
- Application service requirements:
  - Reliability, bandwidth, delay
- Internet transport service model
  - Connection-oriented, reliable (TCP)
  - Unreliable, datagrams (UDP)
- Specific protocols:
  - HTTP
  - SMTP, IMAP
  - DNS
  - P2P: BitTorrent, DHT
- Video streaming, CDNs
- Socket programming:
  - TCP, UDP sockets

DNS provides indirection

Addresses can change underneath:

- Move www.cnn.com to 4.125.91.21
- Humans/apps should be unaffected

Name could map to *multiple* IP addresses

- Enables load balancing, reducing latency by picking nearby servers

Multiple names for the same address

- E.g. many services (mail, www, ftp) on same machine
- E.g. aliases like www.cnn.com and cnn.com

But this flexibility applies only within domain!

Reverse DNS

- IP address -> domain name
- Special PTR record type to store reverse DNS entries

Where is reverse DNS used? (dig -x …)

- Troubleshooting tools (traceroute, ping)
- 'Received' trace header field in SMTP
- SMTP servers for validating IP addresses of originating servers
- Internet forums for tracking users
- System logging or monitoring tools
- Load balancing servers/CDNs, determine location of requester

Transport layer: Overview

Basic concepts

Understand principles behind transport layer services:
- Multiplexing, demultiplexing
- Reliable data transfer
- Flow control
- Congestion control

Learn about internet transport layer protocols:
- UDP: connectionless transport
- TCP: connection-oriented reliable transport
- TCP congestion control

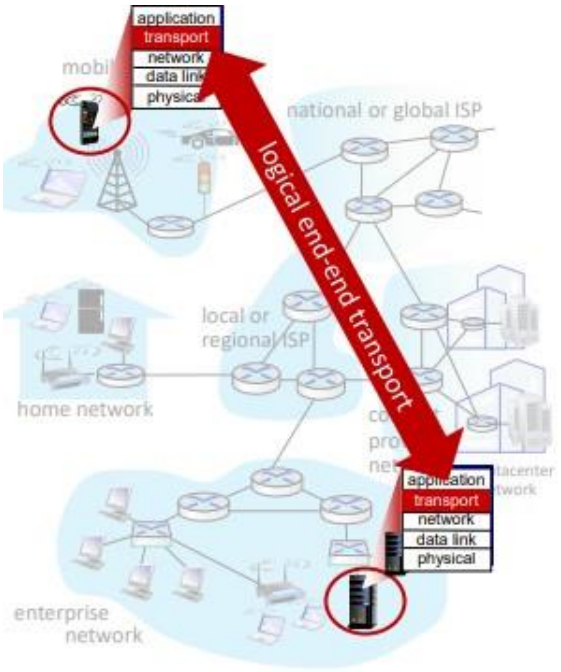
Network layer (for context…)

What it does: finds path through network
- Routing from one end host to another

What it doesn't:
- Reliable transfer: "best effort delivery"
- Guarantee paths
- Arbitrate transmission rates

Transport services and protocols

- Provide logical communication between application processes running on different hosts
- Transport protocols actions in end systems:
  - Sender: breaks application messages into *segments*, passes to network layer
  - Receiver: reassembles segments into messages, passes to application layer
- Two transport protocols available to Internet applications
  - TCP, UDP



Transport Layer actions

Sender:
- Is passed an application layer message
- Determines segment header fields values
- Creates segments
- Passes segment to IP

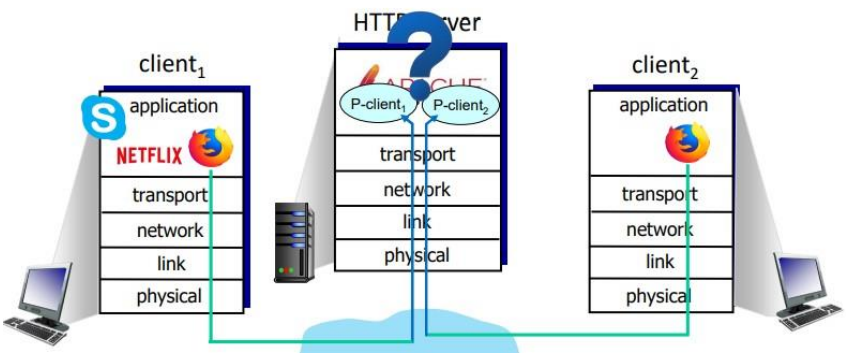
Receiver:
- Received segment from IP
- Checks header values
- Extracts application-layer message
- Demultiplexes message up to application via socket

Two principal Internet transport protocols

TCP: Transmission Control Protocol
- Reliable, in-order delivery
- Congestion control
- Flow control
- Connection setup

UDP: User Datagram Protocol
- Unreliable, unordered delivery
- No-frills extension of "best-effort" IP

Services not available - Delay and bandwidth guarantees

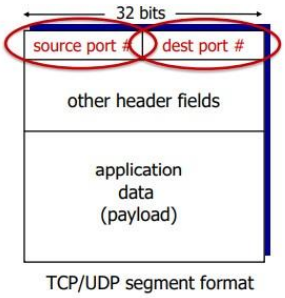


Multiplexing	Demultiplexing
- Handling data from multiple sockets - Add transport header (for later demultiplexing)	- Use header information to deliver received segments to correct socket

Multiplexing/Demultiplexing happen at *all* layers

How demultiplexing works

- Host received IP datagrams
  - Each datagram has source IP address, destination IP address
  - Each datagram carries one transport-layer segment
  - Each segment has source, destination port number
- Host uses *IP addresses & port numbers* to direct segment to socket.



TCP/UDP segment format

Connectionless Demultiplexing (UDP)

Recall:

- When creating socket, either specify *host-local* port# (or let OS pick random available port):

```
DatagramSocket mySocket1 = new
DatagramSocket(12534);
```

- When creating datagram to send into UDP socket, must specify destination IP address and port #

When receiping host receives *UDP* segment:

- Checks destination port# in segment
- Directs UDP segment to socket with that port#

IP/UDP datagrams with *same dest. port#,* but different source IP addresses and/or source port numbers will be directed to *same socket* at receiving host.

Connection-Oriented Demultiplexing (TCP)

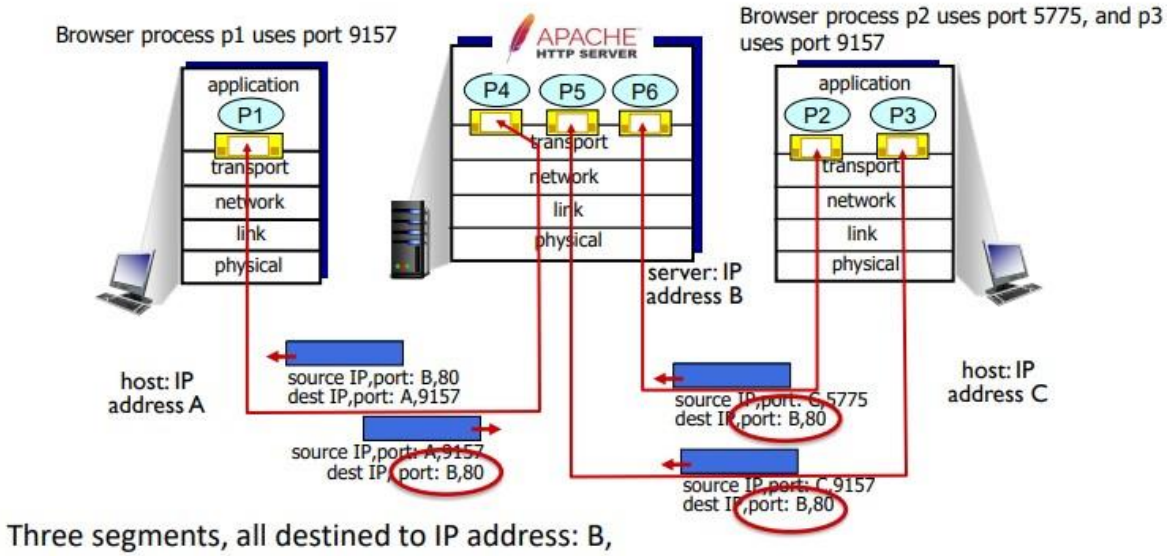
TCP socket identified by 4-tuple:

- Source IP address
- Source port number
- Dest IP address
- Dest port number

Server may support many simultaneous TCP sockets:

- Each socket identified by its own 4-tuple
- Each socket associated with a different connecting client

Demultiplexing: Receiver uses all four values (4-tuple) to direct segment to appropriate socket



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

Port Security

- Servers wait at open ports for client requests
- Hackers often perform *port scans* to determine open, closed and unreachable ports on candidate victims.
- Several ports are well-known
    - < 1024 are reserved for well-known apps
    - Other apps also use known ports
        - MS SQL server uses 1434 (UDP)
        - Sun Network File System (NFS) uses 2049 (TCP/UDP)
- Hackers can exploit known flaws with these apps
    - E.g. Slammer worm exploited buffer overflow flaw in the SQL server
- How do you scan ports?
    - Nmap, Superscan, etc

UDP: User Datagram Protocol

- "No frills", "bare bones" transport protocol
- "Best effort" service, UDP segments may be lost, delivered out of order.
- *Connectionless:*
    - No handshaking between sender and receiver
    - Each UDP segment handled independently

Applications that use UDP:
- Streaming multimedia apps (loss tolerant, rate sensitive)
- DNS
- SNMP
- HTTP/3

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

Why UDP?

- No connection establishment (can avoid RTT delay)
- Simple, no connection state at sender, receiver
- Small header size

No congestion control (don't need it!)

UDP: Transport Layer Actions

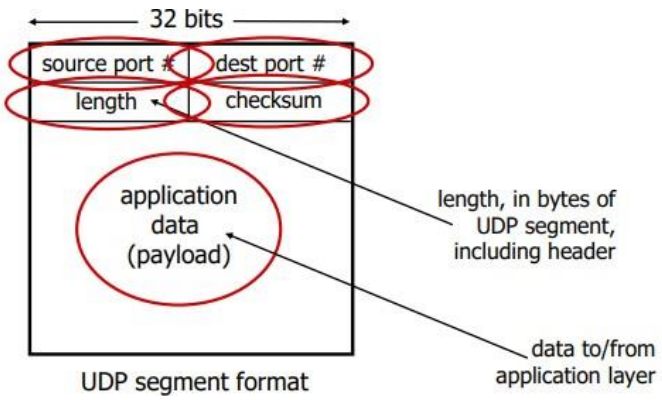
UDP Sender Actions	UDP Receiver Actions
- Is passed an application layer message - Determines UDP segment header fields values - Creates UDP segment - Passes segment to IP	- Receives segment from IP - Checks UDP checksum header value - Extracts application-layer message - Demultiplexes message up to application via socket

UDP Checksum

Goal: detect errors (i.e., flipped bits) in transmitted segment

	1st num	2nd num	checksum
Transmitted	5	6	11
Received	4	6	10

Checksum has detected an error in the segment (10 ≠ 11)



Internet checksum: Example

example: add two 16-bit integers

```
  1 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0
  1 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
wraparound (1) 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 1
sum            1 0 1 1 1 0 1 1 1 0 1 1 1 1 0 0
checksum       0 1 0 0 0 1 0 0 0 1 0 0 0 0 1 1
```

Note: when adding numbers, a carryout from the most significant bit needs to be added to the result

example: add two 16-bit integers

```
  1 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0   (0 1)
  1 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1   (1 0)
wraparound (1) 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 1
sum            1 0 1 1 1 0 1 1 1 0 1 1 1 1 0 0
checksum       0 1 0 0 0 1 0 0 0 1 0 0 0 0 1 1
```

Even though numbers have changed (bit flips), *no* change in checksum!

UDP Checksum in Practice

Checksum is the 16-bit one's complement of the one's complement sum of a pseudo header of information from the IP header, the UDP header, and the data, padded with zero octets at the end (if necessary) to make a multiple of two octets

Checksum header, data and pre-pended IP pseudo-header (some fields from the IP header)

But the header contains the checksum itself?

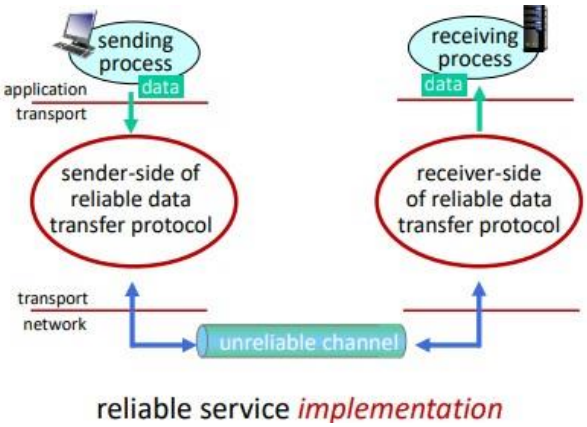
Principles of Reliable Data Transfer (RDT)

Sender, receiver do *not* know the "state" of each other, e.g. was a message received?

- Unless communicated via a message.

Goal:

- Incrementally develop sender, receiver sides of `RDT` protocol
- Consider only unidirectional data transfer (however, control info will flow bidirectionally)



reliable service *implementation*

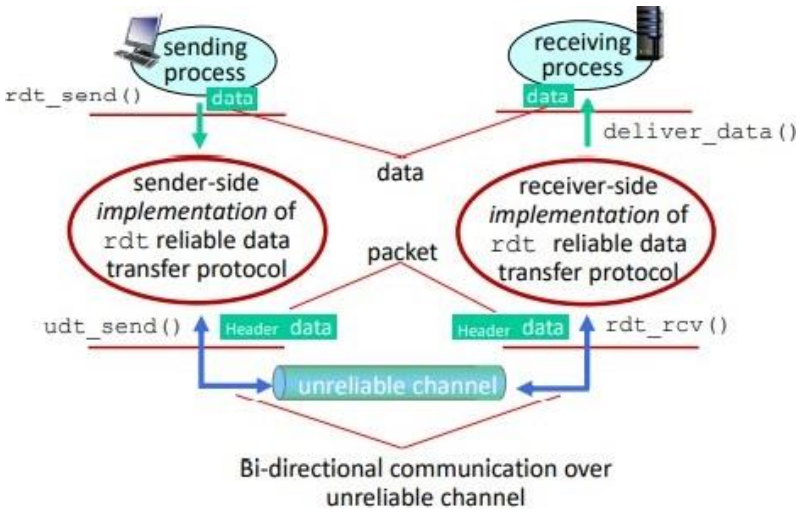
RDT Interface:

`rdt_send()`: called from above (e.g. by app). Passed data to deliver to receiver upper layer

`deliver_data()`: called by `rdt` to deliver data to upper layer

`udt_send()`: called by `rdt` to transfer packet over unreliable channel to receiver

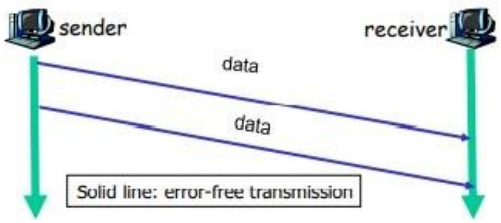
`rdt_rcv()`: called when packet arrives on receiver side of channel



Bi-directional communication over unreliable channel

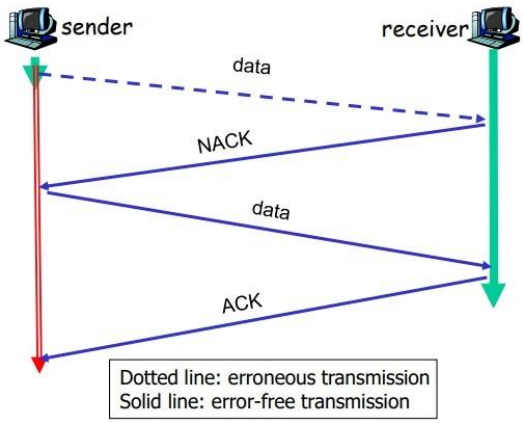
RDT 1.0: Reliable Transfer over a Reliable Channel

- Underlying channel perfectly reliable
  - No bit errors or packet loss



RDT 2.0: Channel with Bit Errors

- Underlying channel may flip bits in packet
  - Checksum (e.g. Internet checksum) to detect bit errors
- The question: how to recover from errors?
  - Acknowledgements (ACKs): Receiver explicitly tells sender that packet received OK
  - Negative Acknowledgement (NAKs): Receiver explicitly tells sender that packet had errors
  - Sender *retransmits* packet on receipt of NAK
- New mechanisms in **RDT2.0** (beyond **RDT1.0**):
  - Error detection
  - Feedback: Control messages (ACK, NAK) from receiver to sender
  - Retransmission



Stop and wait: Sender sends one packet, then waits for receiver response

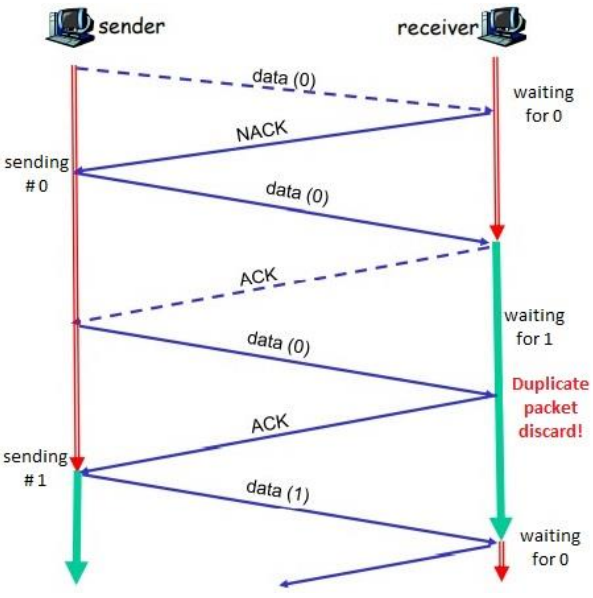
RDT 2.0 has a fatal flaw!	
What happens if ACK/NAK corrupted? - Sender doesn't know what happened at receiver! - Can't just retransmit: possible duplicate	Handling duplicates: - Sender retransmits current packet if ACK/NAK corrupted - Sender adds *sequence number* to each packet - Receiver discards (doesn't deliver) duplicate packets

RDT 2.1: Discussion

Sender:
- Sequence # added to packet
- Two sequence #s (0,1) will suffice
- Must check if received ACK/NAK corrupted
- Twice as many states
  - State must "remember" whether "expected" packet should have sequence # of 0 or 1

Receiver:
Must check if received packet is duplicate
- State indicates whether 0 or 1 is expected sequence #

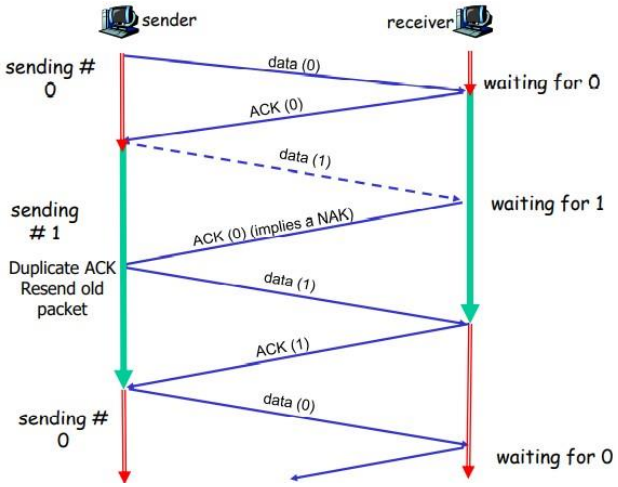
Downsides of NAK:
- Receiver can *not* know if its last ACK/NAK received OK at sender
- Error is only detected after subsequent packet if sequence number is off. This means infrequent packet exchanging could theoretically not be detected indefinitely.



RDT 2.2: A NAK-Free Protocol

- Same functionality as RDT 2.1, *using ACKs only*
- Instead of NAK, receiver sends ACK for last packet received OK
  - Receiver must explicitly include sequence # being ACK'd
- Duplicate ACK at sender results in same action as NAK – Retransmit current packet

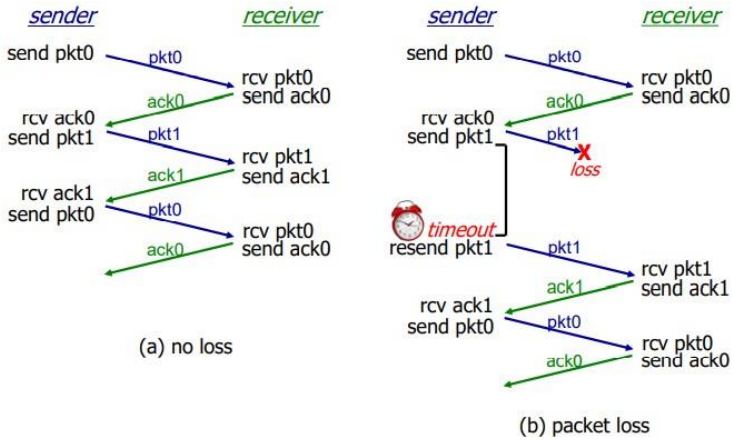
TCP uses this approach to be NAK-free



RDT 3.0 Channels with Errors *and* Loss

Approach: Sender waits "reasonable" amount of time for ACK

- Retransmits if no ACK received in this time
- If packet (or ACK) just delayed (not lost):
  - Retransmission will be duplicate, but sequence #s already handles this
  - Receiver must specify sequence # of packet being ACK'd
- Use countdown timer to interrupt after "reasonable" amount of time
- No retransmission on duplicate ACKs



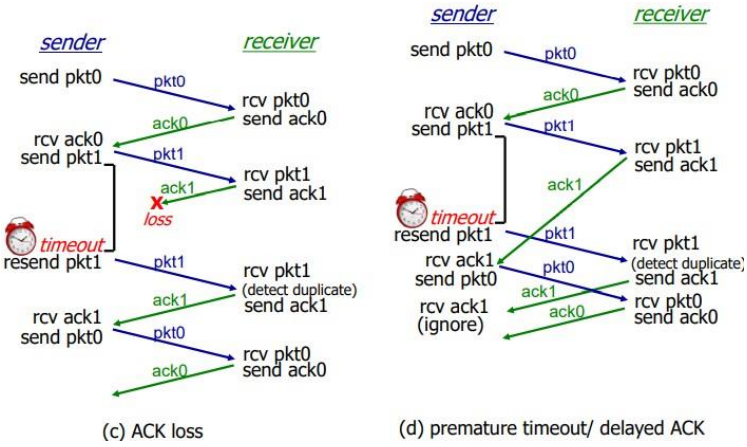
Performance of RDT 3.0 (Stop/Wait)

U_{sender}: *utilisation* – fraction of time sender busy sending

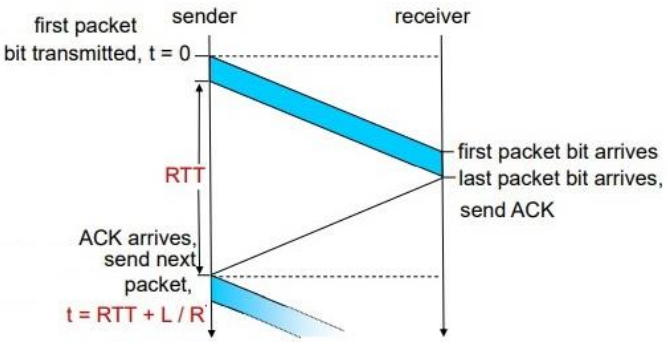
Example: 1Gbps link, 15ms d_{prop}, 8kb packet

Time to transmit packet into channel:

$$d_{trans} = \frac{L}{R} = \frac{8000}{10^9 bits\ per\ sec} = 8us$$

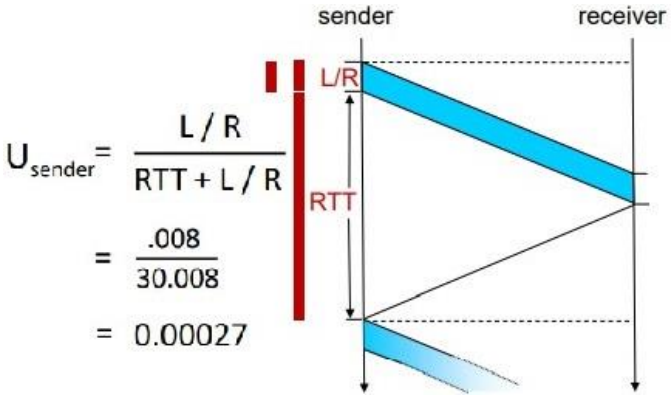


RDT 3.0 Stop-and-Wait Operation



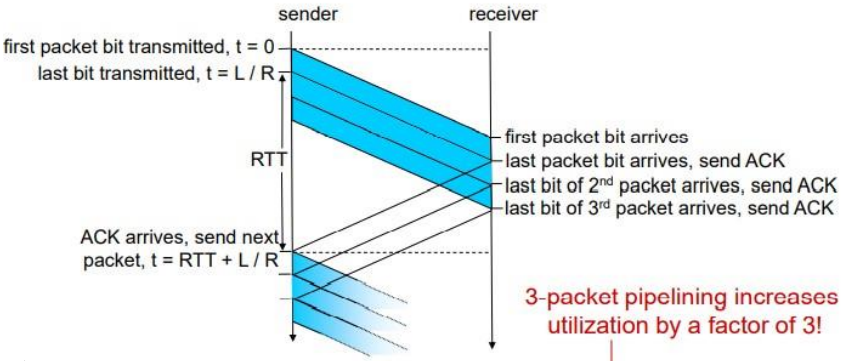
RDT 3.0 Performance

RDT 3.0 protocol performance is *very slow*
Protocol limits performance of underlying infrastructure (channel)



RDT 3.0: 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
- Go-Back-N (GBN), Selective Repeat



Pipelining: Increased Utilisation

$$U_{sender} = \frac{3L/R}{RTT + L/R} = \frac{.0024}{30.008} = 0.00081$$

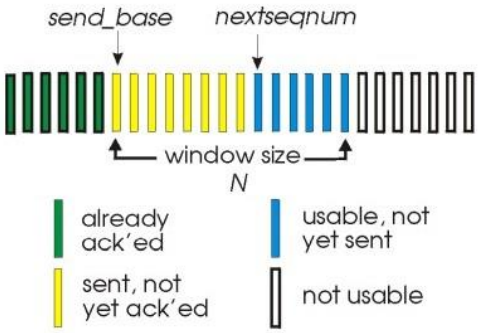
Go-Back-N: Sender

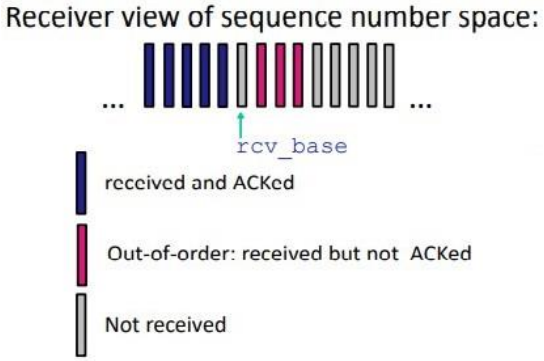
Sender: "window" of up to N, consecutive transmitted but unACKed packets
- K-bit sequence # in packet header

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

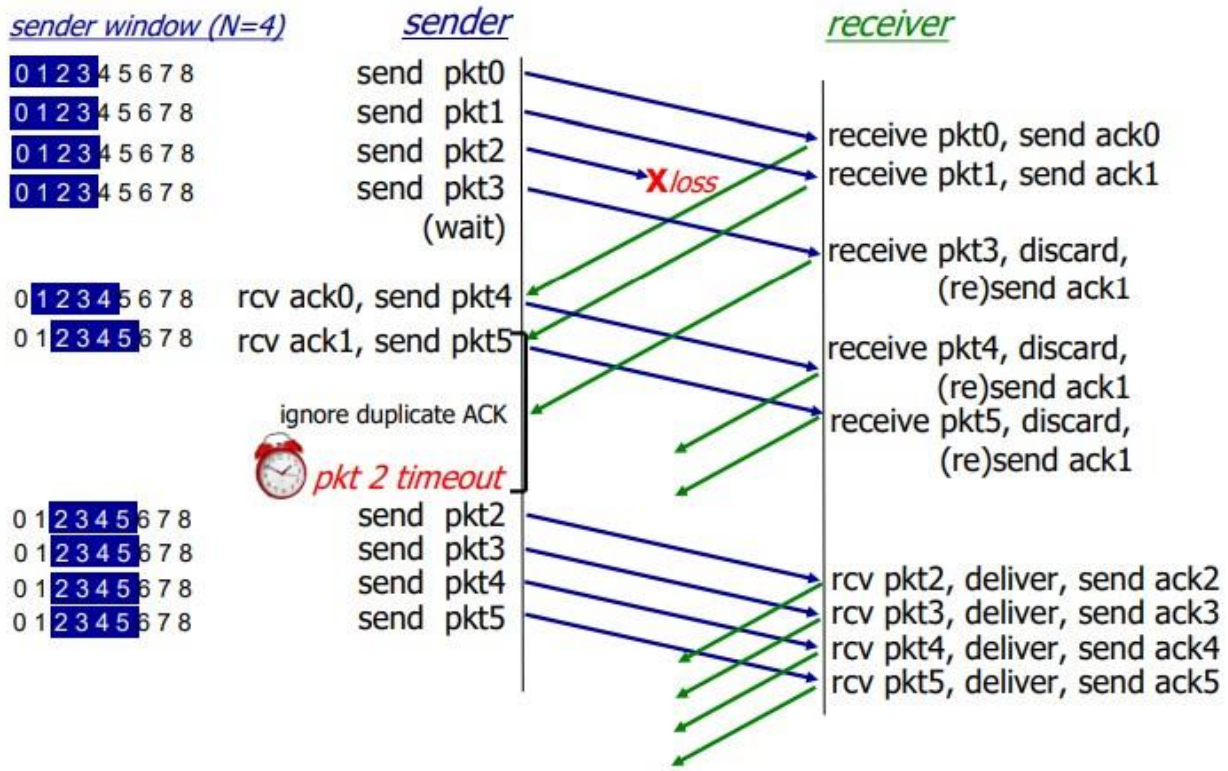




Go-Back-N: Receiver

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 packet with highest in-order sequence #

Go-Back-N in Action



Selective Repeat (ss)

Receiver *individually* acknowledges all correctly received packets

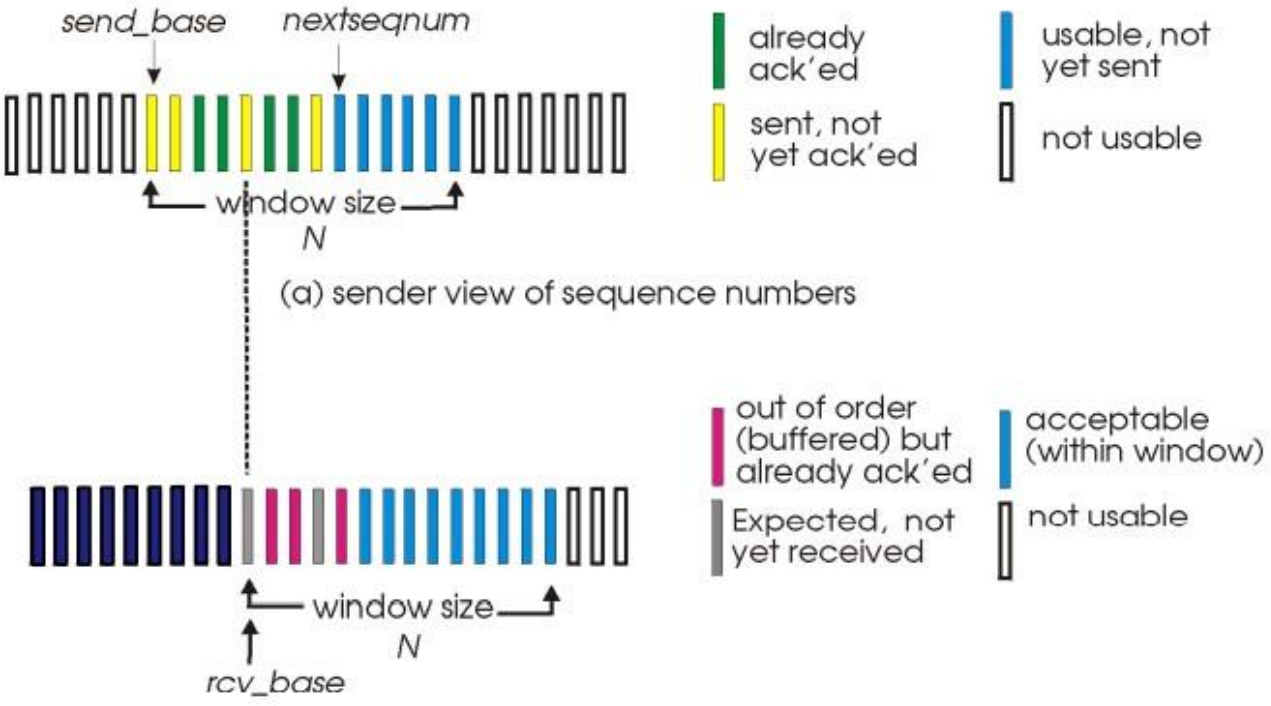
- Buffers packets, as needed, for eventual in-order delivery to upper layer

Sender times-out/retransmits individually for unACKed packets

- Sender maintains timers for each unACKed packet

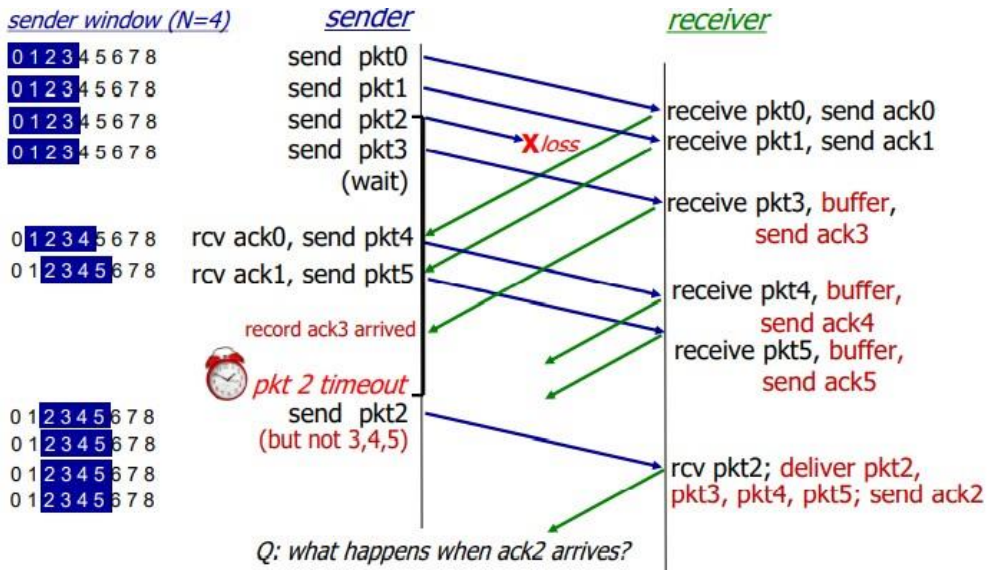
Sender window

- N consecutive seq #s
- Limits seq #s



Selective Repeat (SR)	
Sender	**Receiver**
Data from above: - If next window available seq # in window, send packet Timeout(n): - Resent packet *n*, restart timer ACK(n) in [sendbase, sendbase+N]: - Mark packet *n* as received - If n smallest unACKed packet, advance window base to next unACKed sequence #	Packet *n* in [rcvbase, rcvbase+N-1] - Send ACK(n) - Out-of-order: buffer - In-order: deliver (also deliver buffered, in order packet), advance window to next not-yet-received packet Packet *n* in [rcvbase-N,rcvbase-1] - ACK(n) Otherwise: - ignore

In action:



Selective repeat: A Dilemma!

Example:

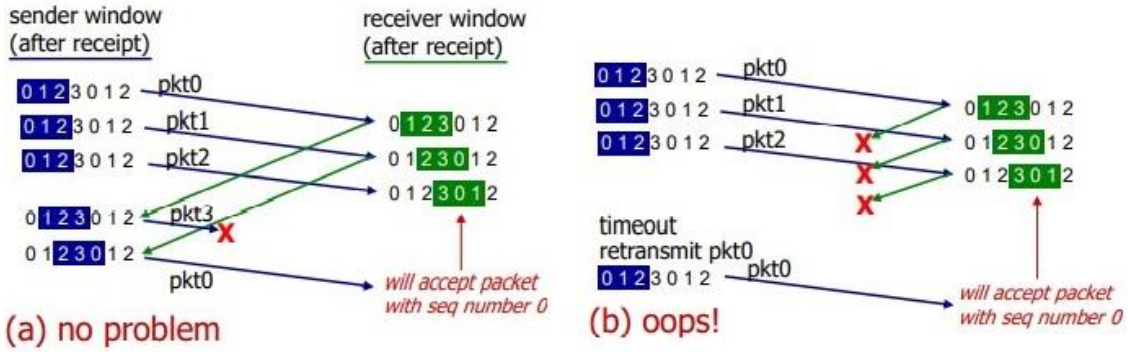
Seq #s:

0, 1, 2, 3

(base 4 counting)

Window size:

3



Q: What relationship is needed between sequence # size and window size to avoid problem in scenario (b)?

A: Sender window size ≤ ½ of sequence number space

Recap - Components of a Solution:

- Checksums (for error detection)
- Timers (for loss detection)
- Acknowledgements
  - Cumulative
  - Selective
- Sequence numbers (duplicates, windows)
- Sliding Windows (for efficiency)

- Reliability protocols use the above to decide when and what to retransmit or acknowledge

---

TCP: Overview [RFCs: 793, 1122, 2018, 5681, 7323]

Point-to-point:
- One sender, one receiver

Reliable, in-order *byte stream*:
- No "message boundaries"

Full duplex data:
- Bi-directional data flow in same connection
- MSS: Maximum segment size

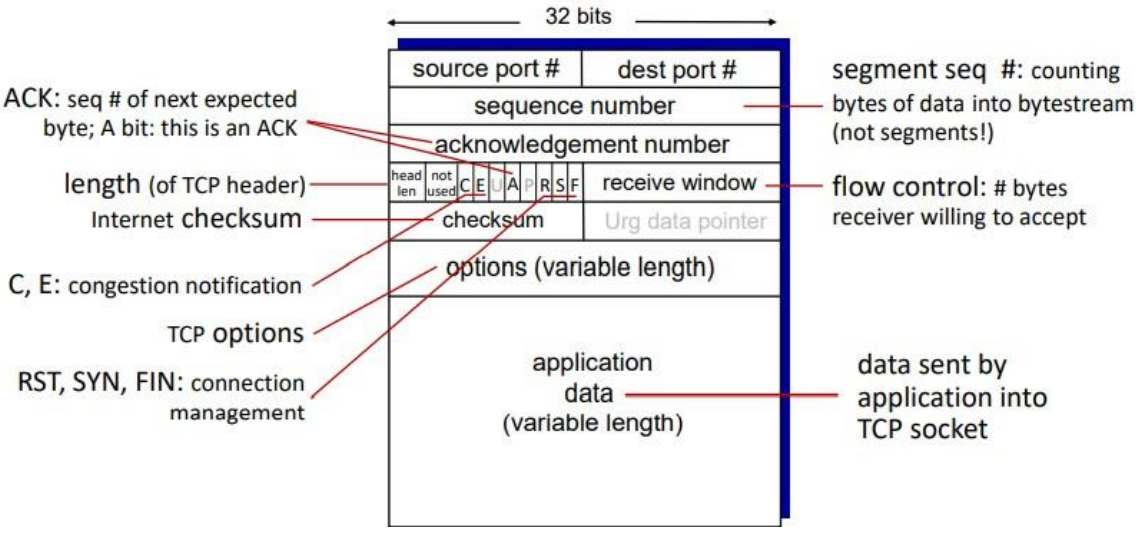
Cumulative ACKs

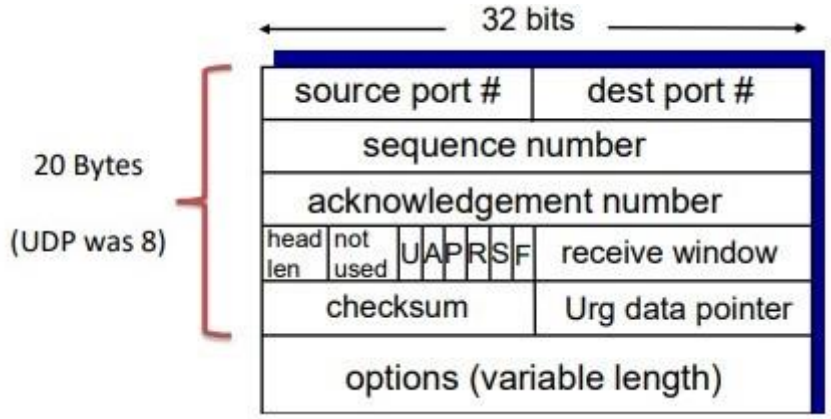
Pipelining:
- TCP congestion and flow control set window size

Connection-oriented:
- Handshaking (exchange of control messages) initialises sender, receiver state before data exchange

Flow controlled:
- Sender will not overwhelm receiver

TCP Segment structure:

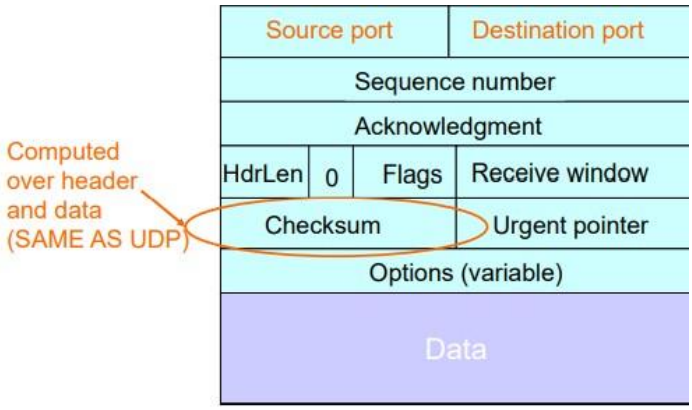




What does TCP Do?

Many of our previous ideas, but some key differences

- Checksum
- Sequence numbers are byte offsets
- Receiver sends cumulative acknowledgements (like GBN)
- Receivers **can** buffer out-of-sequent packets (like SR)
- Sender maintains a single retransmission timer (like GBN) and retransmits on timeout.



TCP Maximum Segment Size

IP Packet

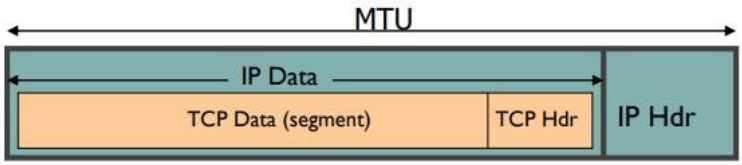
- No bigger than Maximum Transmission Unit (MTU) of link layer
- E.g. up to 1500 bytes with Ethernet

TCP Packet

- IP packet with a TCP header and data inside
- TCP header ≥ 20 bytes long

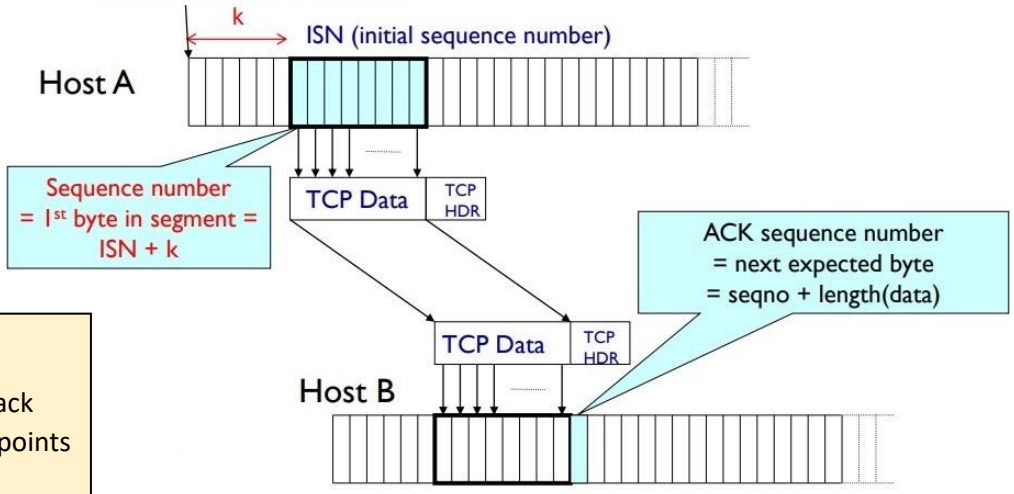
TCP segment

- No more than Maximum Segment Size (MSS) bytes
- E.g. up to 1460 consecutive bytes from the stream
- $MSS = MTU - \min(IP_{header}) - \min(TCP_{header})$
- $MSS = MTU - 20 - 20$



Sequence Numbers:

Byte stream "number" of first byte in segment's data



Why choose random ISN?

Avoids ambiguity with back-to-back connections between same end-points

Potential security issue is ISN is known

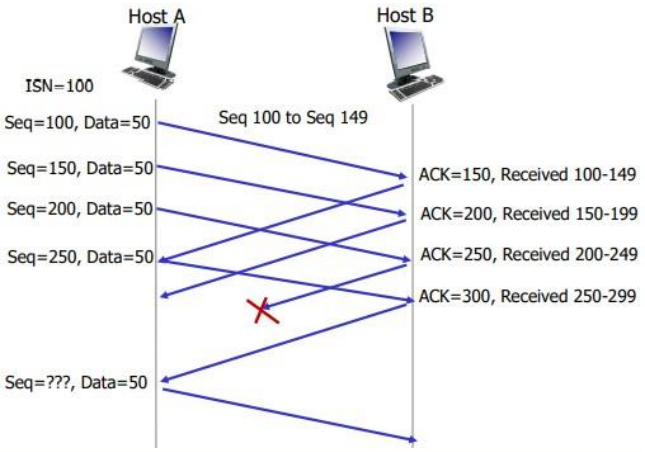
ACKing and Sequence Numbers

Sender sends packet

- Data starts with sequence number X
- Packet contains B bytes [X, X+1, X+2, …, X+B-1]

Upon receipt of packet, receiver sends an ACK

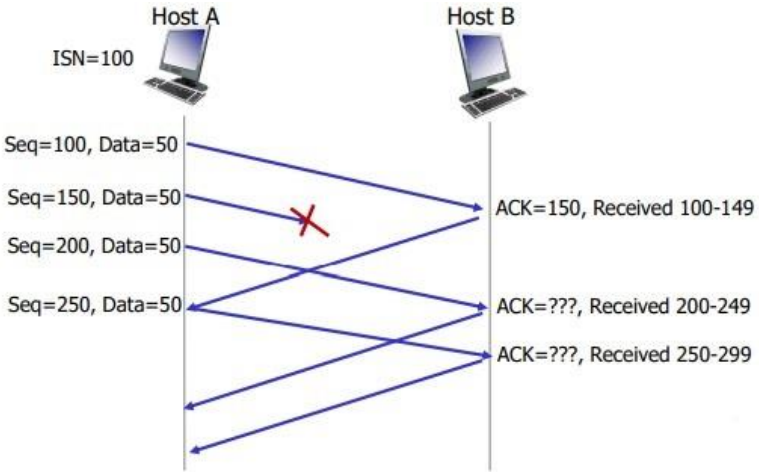
- If all data prior to X already received:
  - Ack acknowledges *X + B* (next expected byte)
- If highest in-order byte received is Y such that (Y+1) < X
  - Ack acknowledges *Y + 1*
  - Even if this has been ACKed before



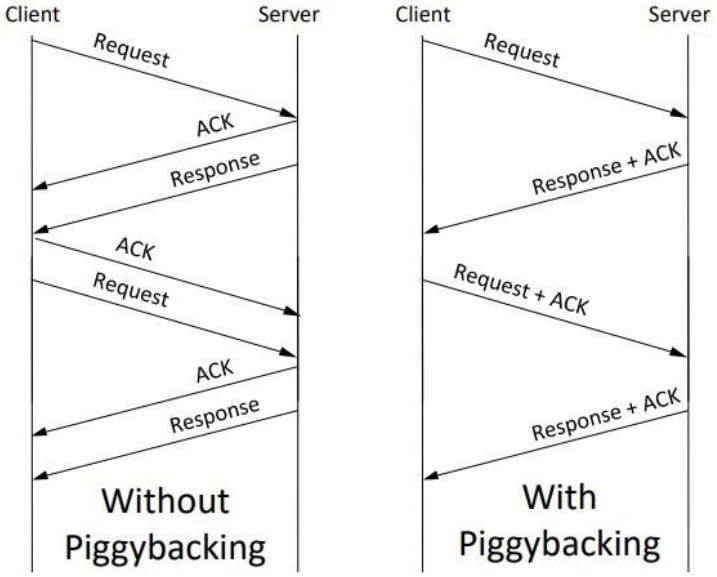
Seq = 300 (new segment)

Since TCP uses cumulative ACKs, the receipt of ACK 300 before a timeout (for seg with sequence number 200) implies the receiver has received all 4 segments sent above

Another example:



Both ACKs will be 150 as the receiver has received everything up to 149 in correct sequence. The two out of order segments are buffered



Piggybacking

So far, we've assumed distinct "sender" and "receiver" roles

Usually both sides of a connection (i.e. the application processes) send some data

Example - Loss with cumulative ACKs

Sender sends packets with 100 bytes and sequence #s:

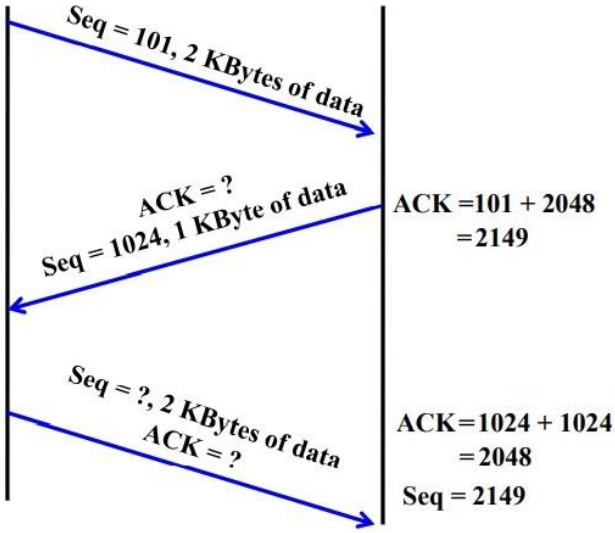
- 100, 200, 300, 400, 500, 600, 700, 800, 900, …

Assume (only) the fifth packet (seq. no 500) is lost.

6th packet onwards are buffered

Stream of ACKs will be:

- 200, 300, 400, 500, 500, 500, 500, …



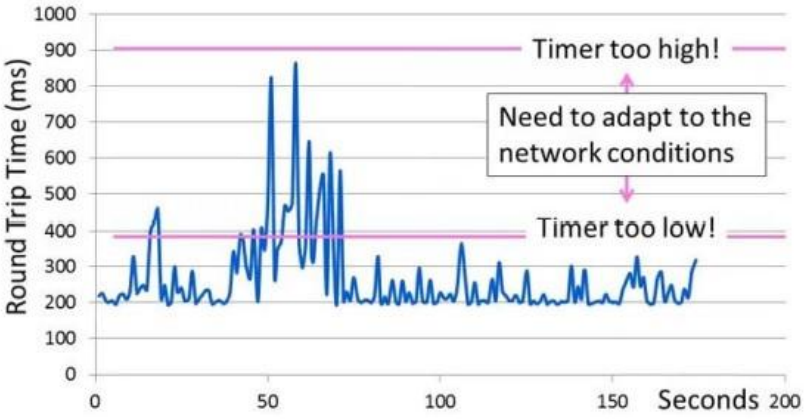
TCP Round Trip Time, Timeout

Q: How to set TCP timeout value?
- Longer than RTT, but RTT *varies*
- *Too short:* unnecessary retransmissions
- *Too long:* slow reaction to segment loss

Q: How to estimate RTT?

`SampleRTT`: measured time from segment transmission until ACK receipt (ignore retransmissions)
- Will vary, want estimated RTT "smoother"

Average several recent measurements, not just `SampleRTT`

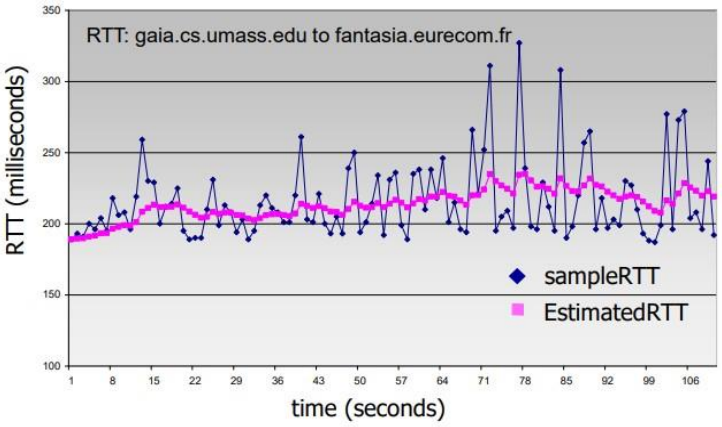



```
EstimatedRTT =
(1 - α)*EstimatedRTT + α*SampleRTT
```

Exponential **W**eighted **M**oving **A**verage (EWMA)

Influence of past sample decreases exponentially fast.

Typical value $\alpha = 0.125$

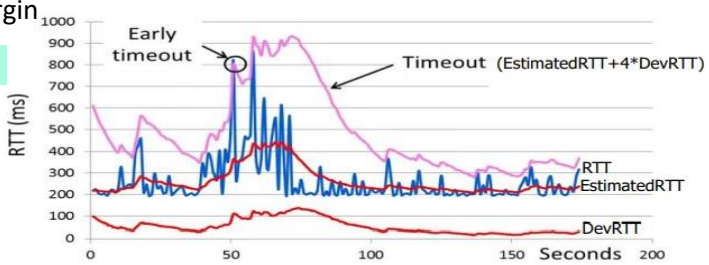


Timeout interval: `EstimatedRTT` + "safety margin"

Large variation in `EstimatedRTT`? Want larger safety margin

```
TimeoutInterval = EstimatedRTT + 4*DevRTT
```

"safety margin"



`TimeoutInterval = EstimatedRTT + 4*DevRTT`

estimated RTT

"safety margin"

`DevRTT`: EWMA of `SampleRTT` deviation from `EstimatedRTT`:

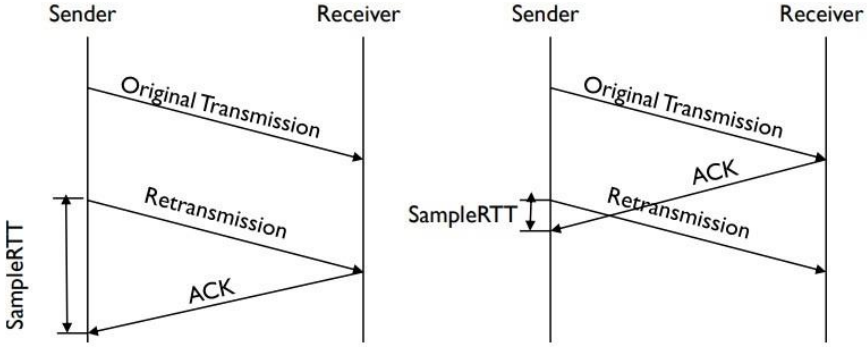
```
DevRTT = (1-β)*DevRTT β*|SampleRTT - EstimatedRTT
```

(Typically, β = 0.25)

Why exclude retransmissions in RTT computation?

How do we differentiate between the real ACK, and ACK of retransmitted packet?

Sender cannot differentiate between the two scenarios shown below



TCP Sender (Simplified)

Event: *data received from application*

- Create segment with sequence #
- Sequence # is byte-stream number of first data byte in segment
- Start timer if not already running (for oldest unACK'd segment)
- Expiration interval = `TimeOutInterval`

Event: *timeout*

- Retransmit segment that caused timeout
- Restart timer

Event: *ACK received*

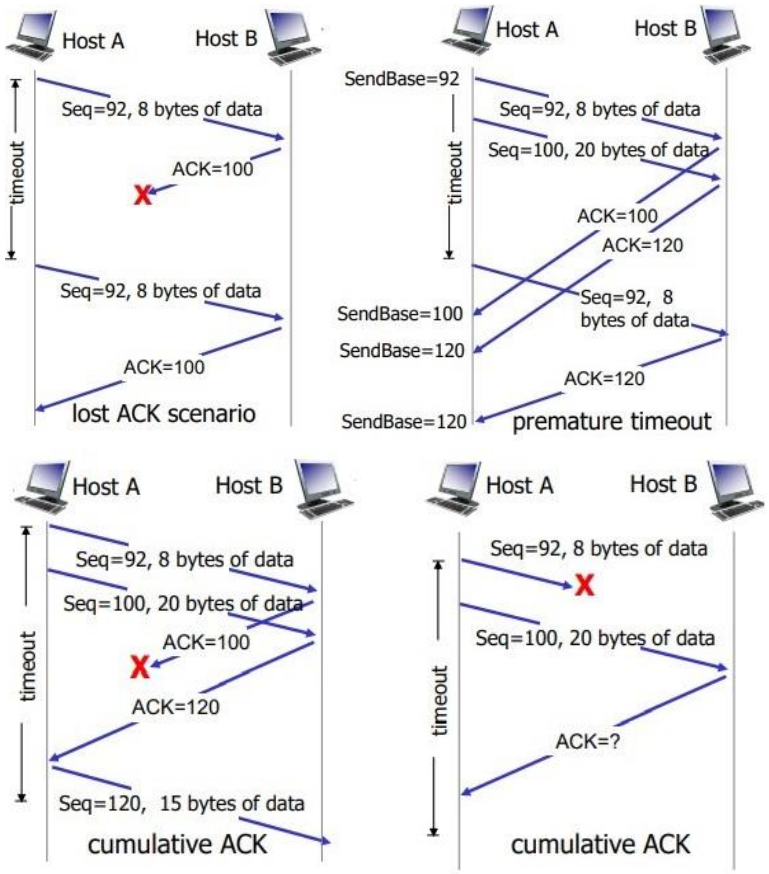
- If ACK acknowledges previously unACKed segments
- Update what is known to be ACKd
- Start timer if there are still unACKd segments

TCP ACK Generation

Note: You may neglect delayed ACKs in exam unless explicitly told to consider it

Event At Receiver	TCP Receiver Action
Arrival of inorder segment with expected sequence #. All data up to expected sequence # already ACKd	Delayed ACK. Wait up to 500ms for next segment. If no next segment, send ACK
Arrival of in-order segment with expected sequence #. One other segment has ACK pending	Immediately send single cumulative ACK, ACKing both in-order segments
Arrival of out-of-order segment higher-than-expected sequence #, gap detected.	Immediately send *duplicate ACK*, indicating sequence # of next expected byte
Arrival of segment that partially or completely fills gap	Immediately send ACK, provided that segment starts at lower end of gap

TCP: Retransmission Scenarios



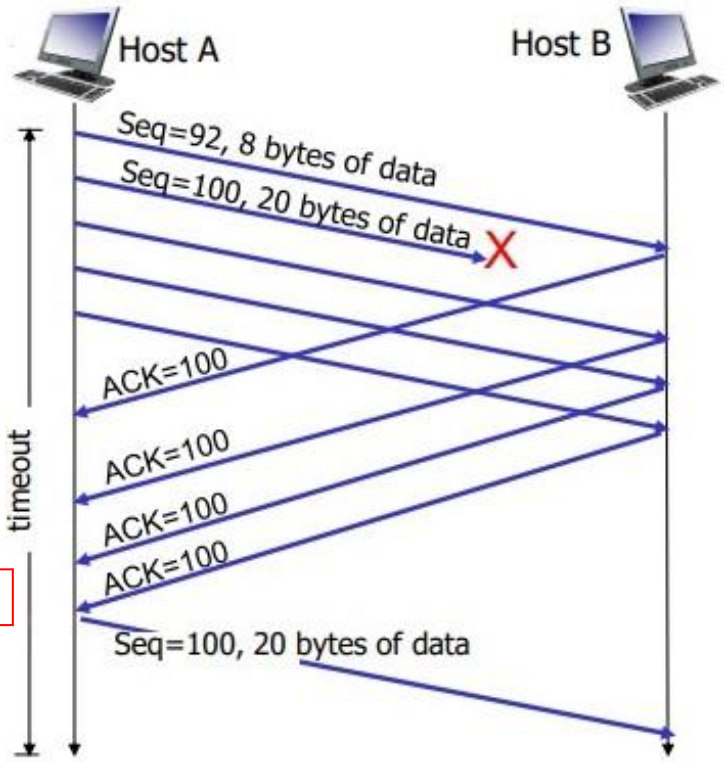
TCP Retransmit (cont.)

TCP Fast Retransmit

If sender receives 3 additional ACKs for same data ("triple duplicate ACKs"), resent unACKed segment with smallest sequence #

Likely that unACKed segment lost, so don't wait for timeout.

Receipt of 3 duplicate ACKs = lost segment likely!

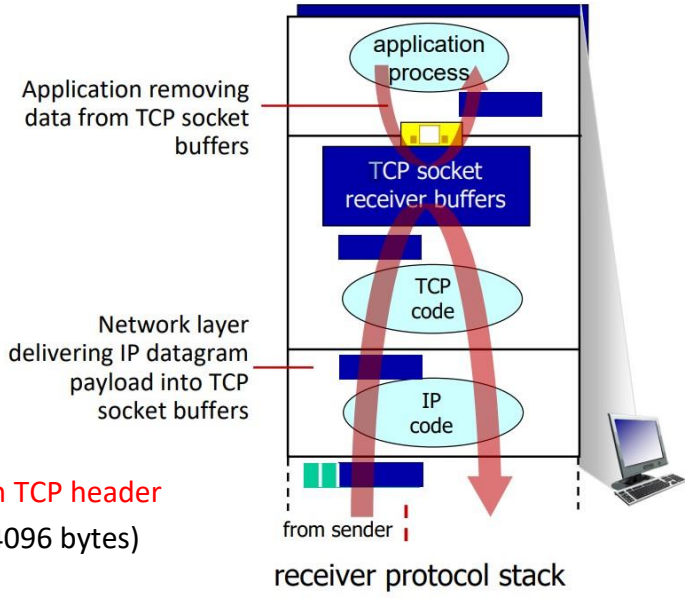


TCP Flow Control

Q: What happens if network layer delivers data faster than application layer removes data from socket buffers?

Flow Control

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

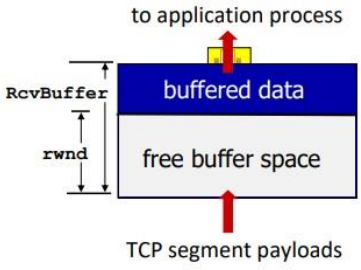


TCP receiver "advertises" free buffer space in **`rwnd`** field in TCP header

- **`RcvBuffer`** size set via socket options (typical default 4096 bytes)
- Many operating systems auto adjust **`RcvBuffer`**

Sender limits amount of unACKed ("in-flight") data to received **`rwnd`**

Guarantees receive buffer will not overflow



TCP receiver-side buffering

What if `rwnd = 0`?

- Sender would stop sending data
- Eventually the receive buffer would have space when the application process reads some bytes
- But how does the receiver advertise the new `rwnd` to the sender?

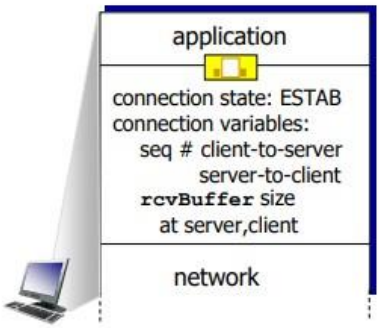
Sender keeps sending TCP segments with one data byte to the sender

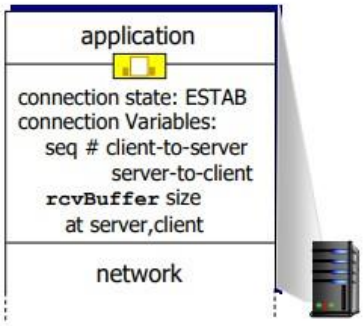
These segments are dropped but acknowledge by the receiver with a zero-window size

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 sequence #s)

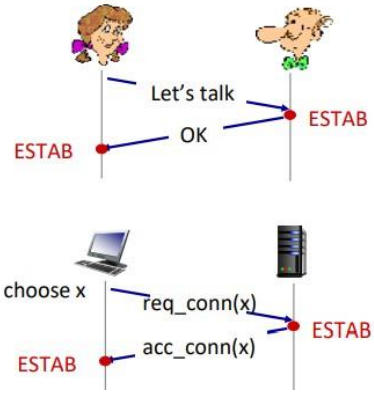





```
Socket clientSocket =
  newSocket("hostname","port number");
```

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

2-way handshake:

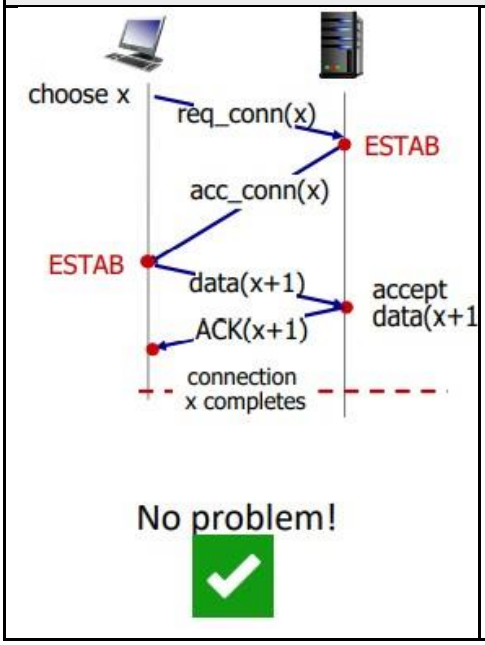


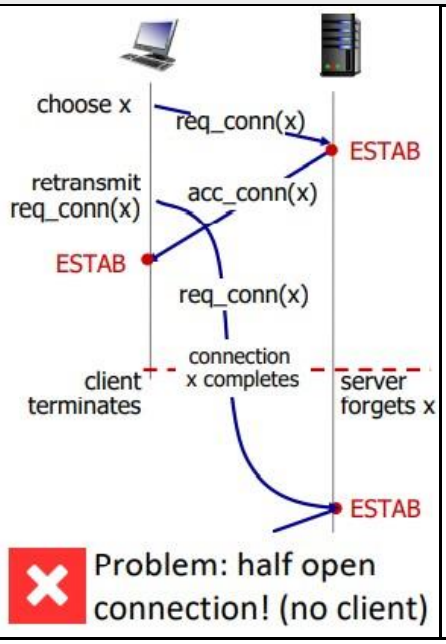
Agreeing to Establish a Connection

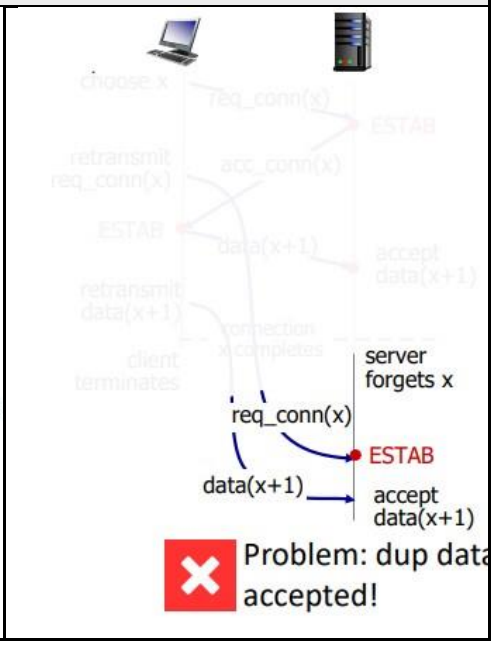
Q: Will 2-way handshake always work in network?

- Variable delays
- Retransmitted messages (e.g. req_conn(x)) due to message loss
- Message ordering
- Can't "see" other side

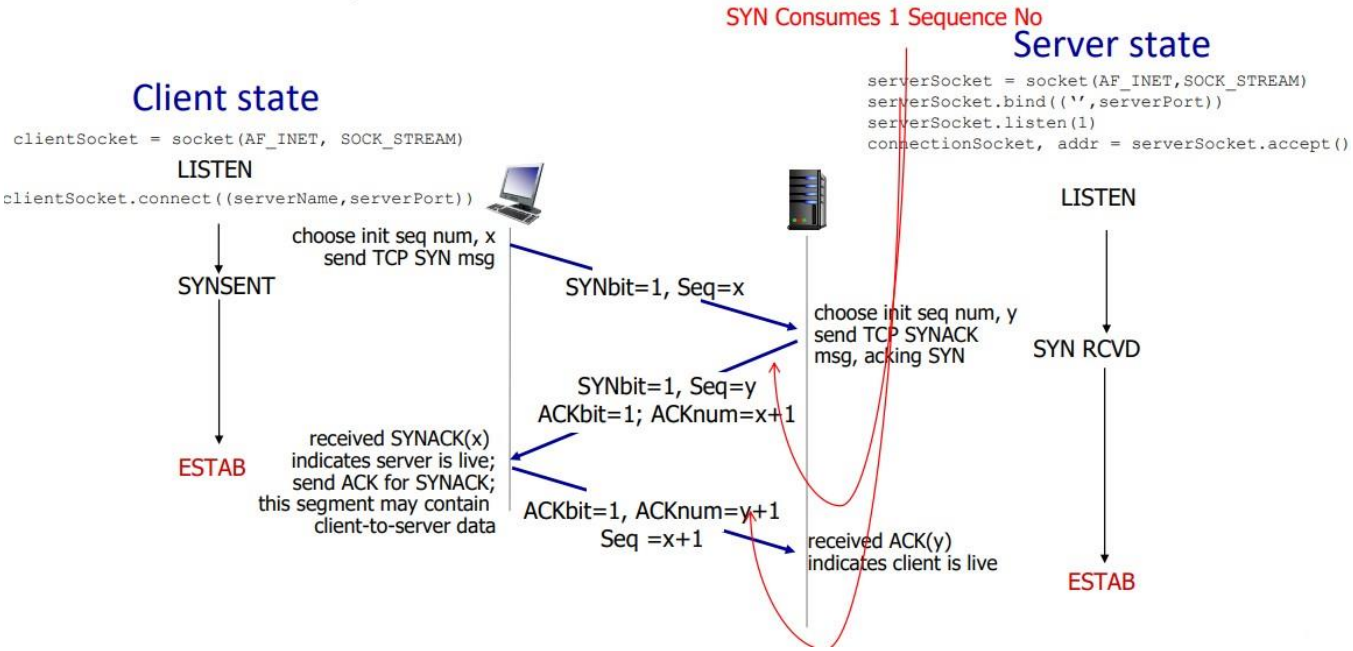
2-Way Handshake Scenarios

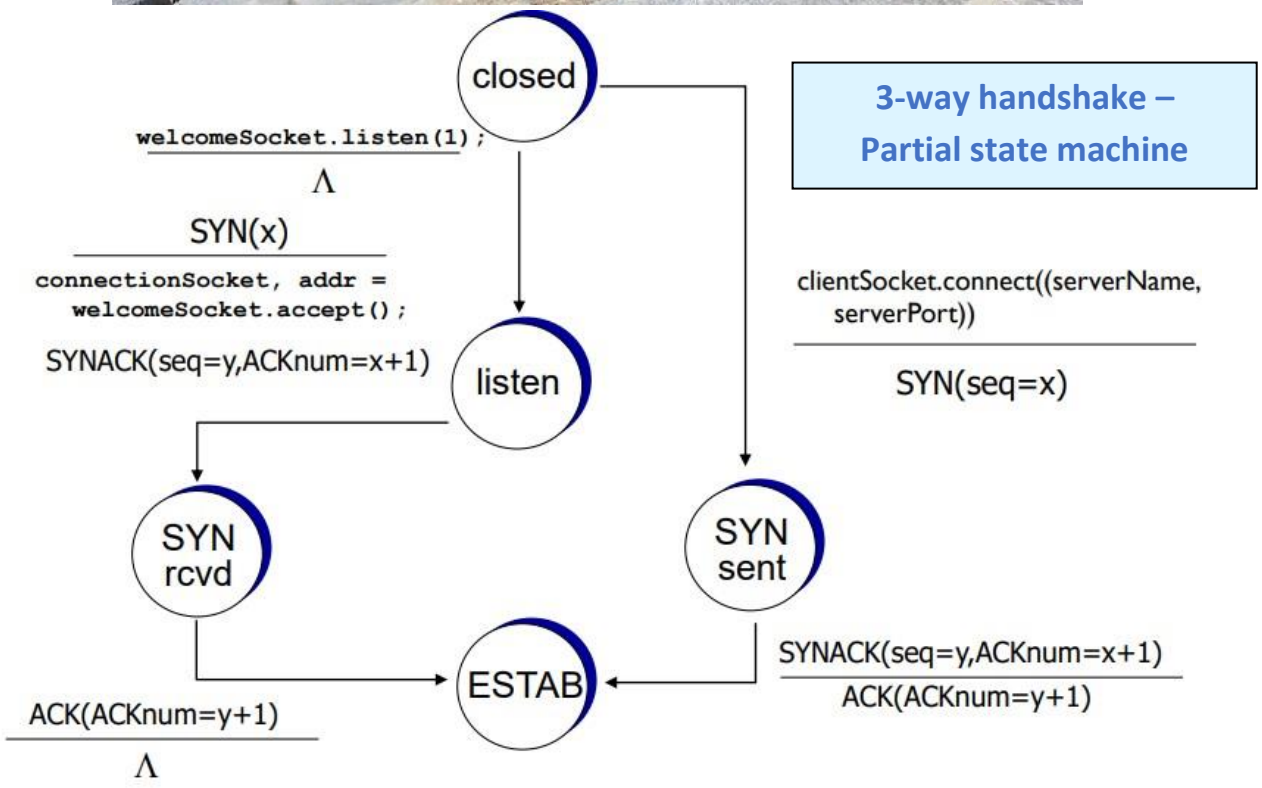






TCP 3-Way Handshake (Partial State Machine)





3-way handshake –
Partial state machine

What if the SYN Packet Gets Lost?

Suppose the SYN packet gets lost - Packet is lost inside the network, OR - Server *discards* the packet (e.g. its too busy) Eventually, no SYN-ACK arrives - Sender sets a *timer* and *waits* for the SYN-ACK - Retransmit if needed (timer expires)	How should the TCP sender set the timer? - Sender has no idea how far away the receiver is - Hard to guess a reasonable length of time to wait - **SHOULD** use default of 3 second (RFCs 1122,2988) RFC 6298 use default of 1 second

TCP: Closing a Connection

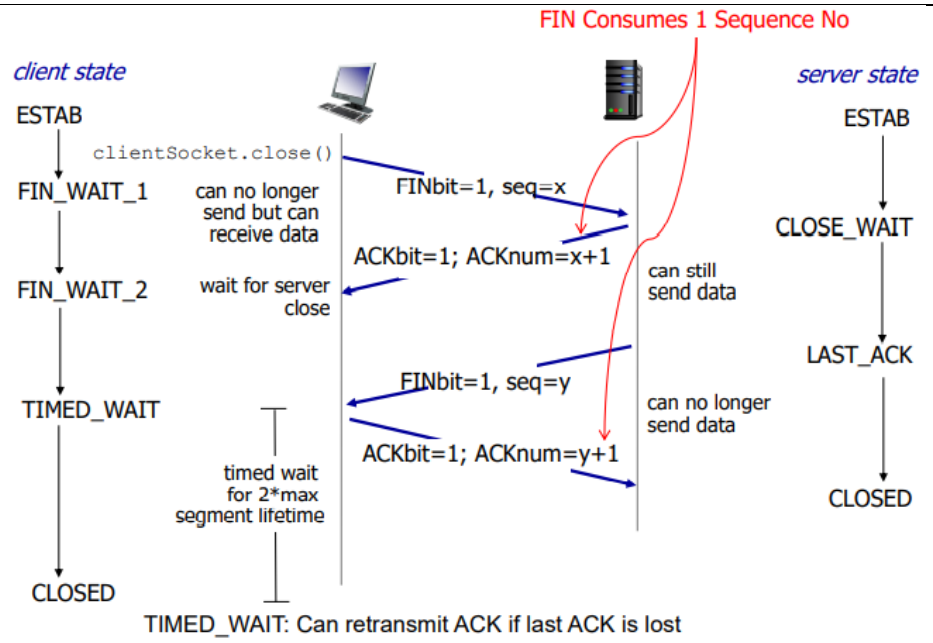
Client, server each close their side of connection

- Send TCP segment with **`FIN` bit = 1**

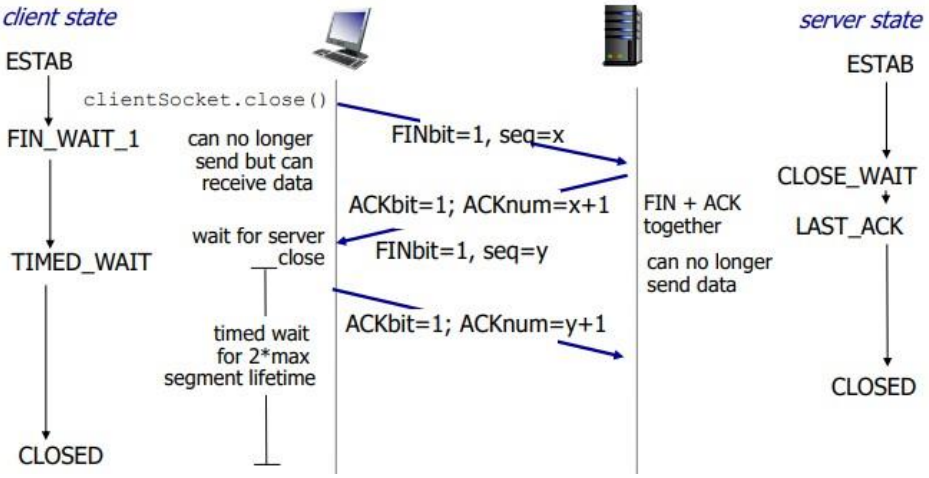
Respond to received `FIN` with ACK

- On receiving `FIN`, ACK can be combined with own `FIN`

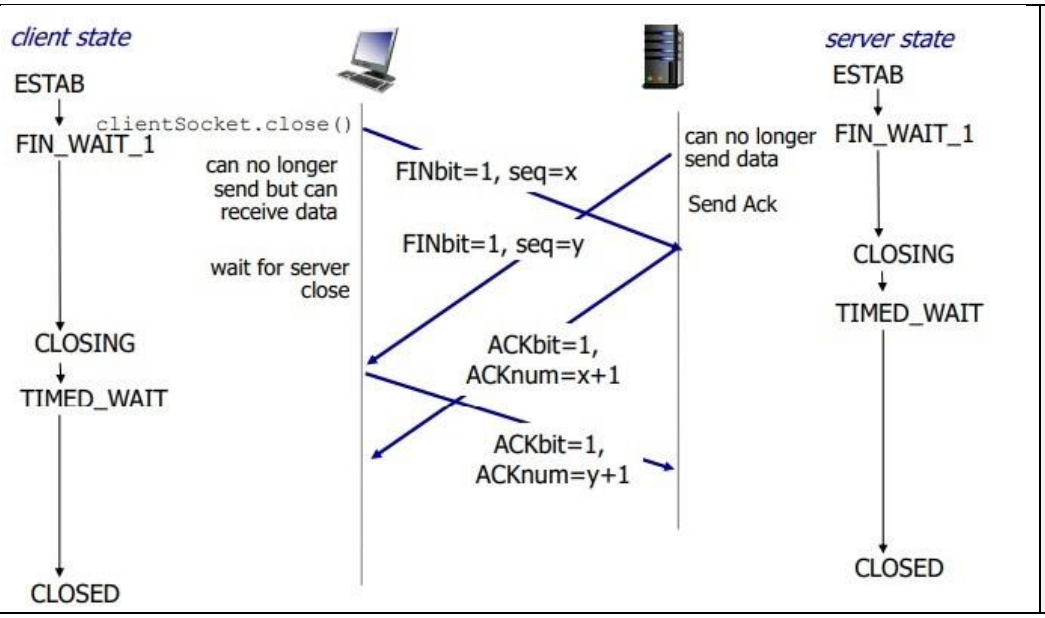
Simultaneous `FIN` exhanges can be handled



Normal Termination: *One at a time*

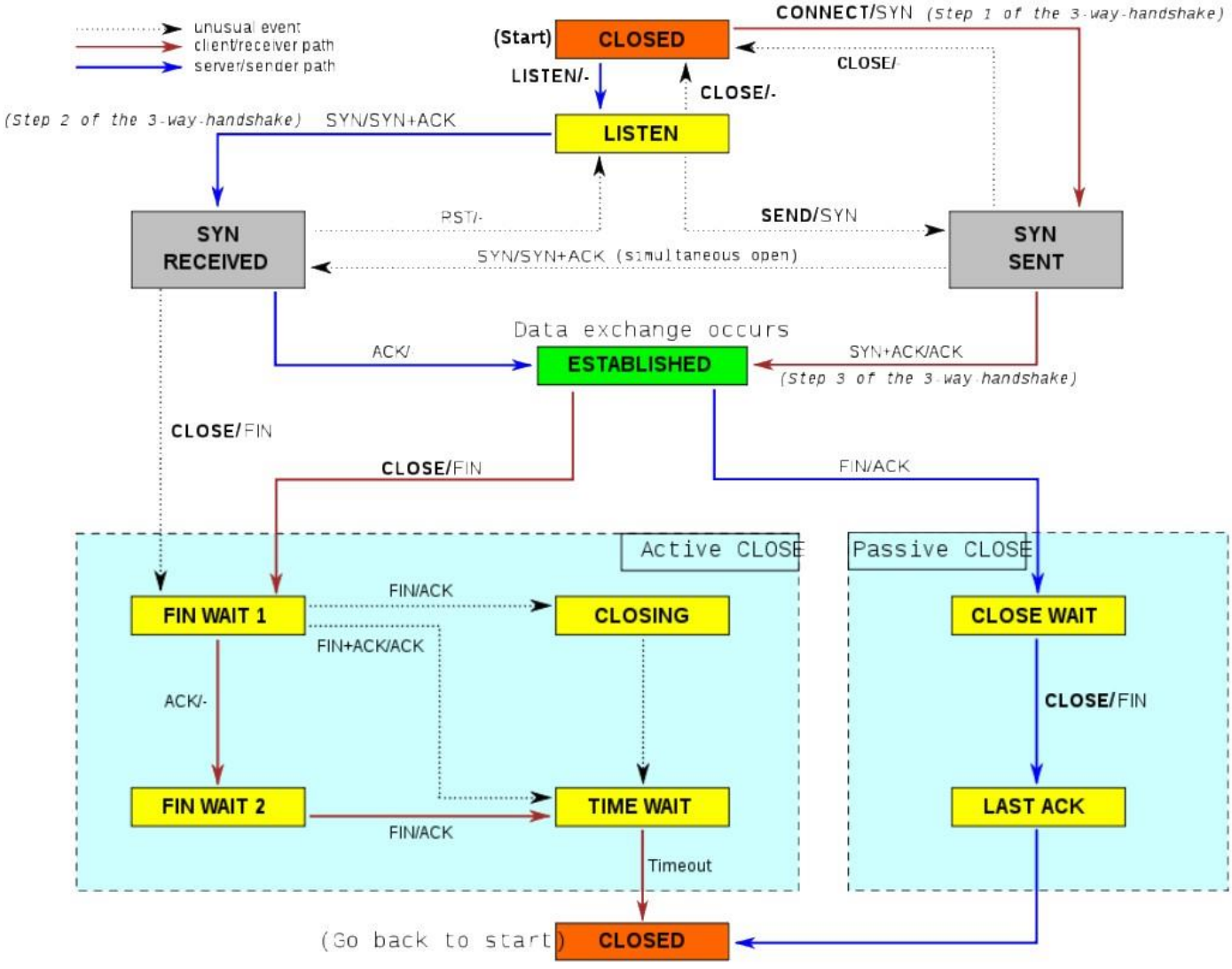


Normal Termination: Both at once

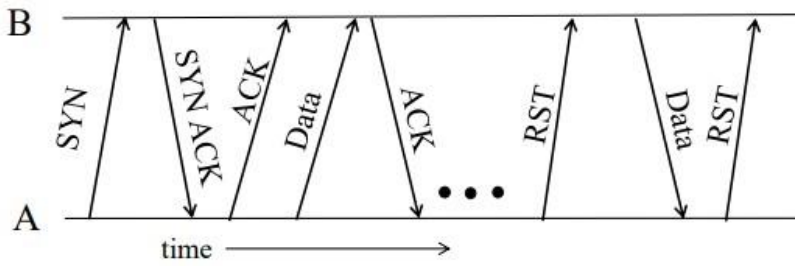


Simultaneous Closure

TCP Finite State Machine



Abrupt Termination



A sends a RESET (**RST**) to B

- Because application on process A has crashed

That's it!

- B does *not* ack the RST
- Thus, RST is *not delivered reliably*
- Any data in flight is *lost*
- But if B sends anything more, will elicit another **RST**

TCP SYN Attack (SYN Flooding)

- Miscreant creates a fake SYN packet
  - Destination is IP address of victim host (usually some server)
  - Source is some spoofed IP address
- Victim host on receiving creates a TCP connection state *i.e. allocates buffers, creates variables, etc*, and sends SYN ACK to the spoofed address (half-open connection)
- ACK never comes back
- After a timeout connection state is freed
- However for this duration the connection state is unnecessarily created
- Further miscreant sends large number of fake SYNs
  - Can easily overwhelm the victim
- Solutions:
  - Increase size of connection queue
  - Decrease timeout wait for the 3-way handshake
  - Firewalls: list of known bad source IP addresses
  - TCP SYN Cookies

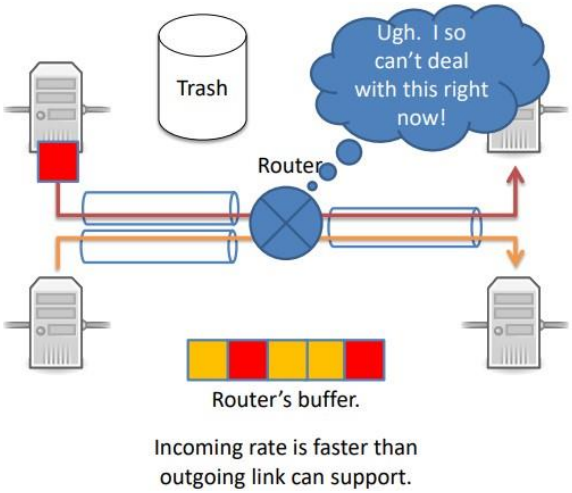
TCP Cookies

- On receipt of SYN, server does not create connection state
- It creates an initial sequence number (*init_seq*) that is a hash of source & dest IP address and port number of SYN packet (secret key used for hash)
  - Replies back with SYN ACK containing *init_seq*
  - Server does not need to store this sequence number
- If original SYN is genuine, an ACK will come back
  - Same hash function run on the same header fields to get the initial sequence number (*init_seq*)
  - Checks if the ACK is equal to (*init_seq*+1)
  - Only create connection state if above is true
- If fake SYN, no harm done since no state was created

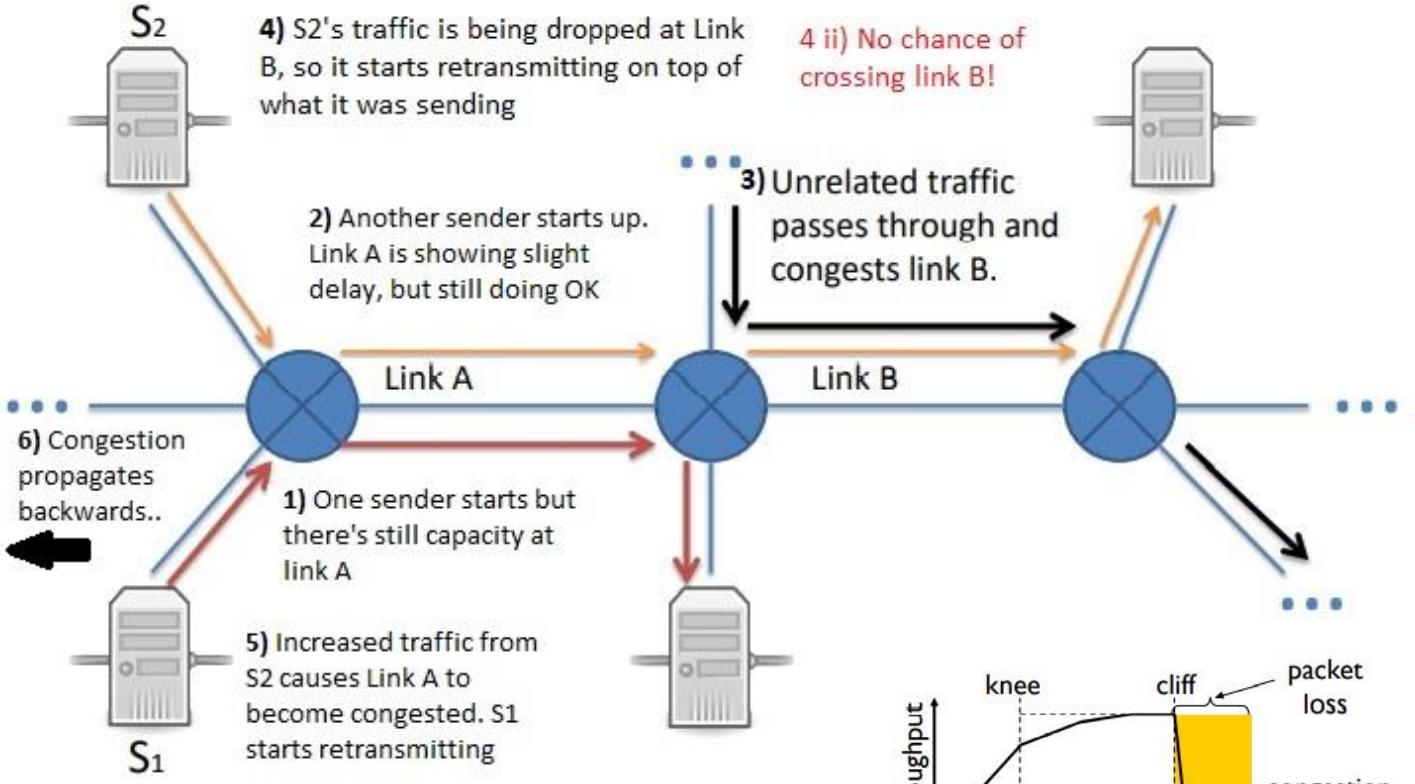
Principles of Congestion Control

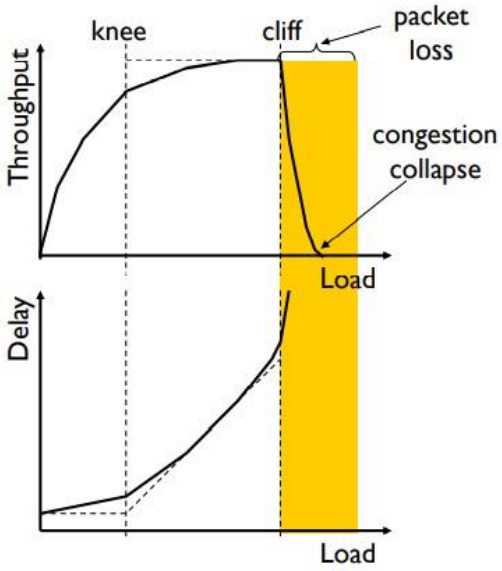
Congestion:

- Informally: "too many sources sending too much data too fast for network to handle"
- *Different from flow control!*
- Manifestations:
  - Lost packets (buffer overflow at routers)
  - Long delays (queueing in router buffers)
- A Top-10 problem!



Example: Congestion Collapse:





Without Congestion Control – Congestion…

- Increases delays
  - If delays > RTO, sender retransmits
- Increases loss rate
  - Dropped packets also retransmitted
- Increases retransmissions, many unnecessary
  - Wastes capacity of traffic that is never delivered
  - Increase in load results in decrease in useful work done
- Increases congestion, cycle continues

Knee – point after which throughput increases slowly, delay increases fast

Cliff – Congestion collapse. Point after which throughput starts to drop to zero. Delay approaches infinity

Approaches Towards Congestion Control

End-To-End	Network Assisted
- No explicit feedback from network - Congestion inferred from end-system observed loss, delay - Approach taken by TCP	- Routers provide feedback to end systems - Single bit indicating congestion (SNA, DECbit, TPC/IP ECN, ATM) - Explicit rate for sender to send at

---

TCP's Approach in a Nutshell

CWND Measured in units of Maximum Segment Size (MSS)

MSS – amount of payload data in a TCP packet

TCP connection maintains a window
- Controls number of packets in flight

TCP sending rate:
- *Roughly*: send cwnd bytes, wait RTT for ACKs, then send more bytes

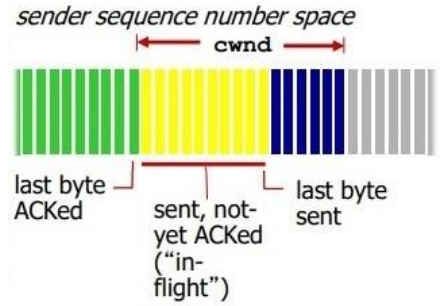
Vary window size to control sending rate

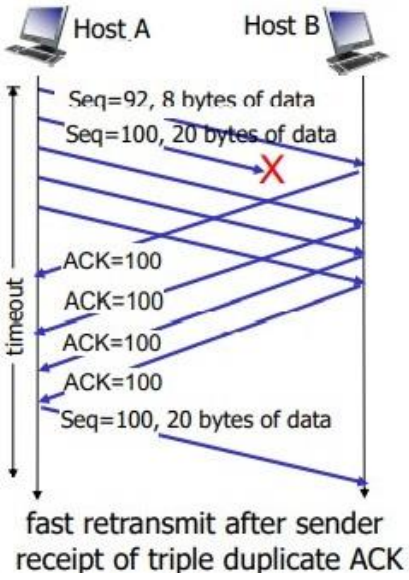
Congestion Window: CWND
- How many bytes can be sent without overflowing routers
- Computers by the sender using congestion control algorithm

Flow Control Window: Advertised / Receive Window (RWND)
- How many bytes can be sent without overflowing receiver's buffers
- Determined by the receiver and reported to the sender

Sender-side window = MIN(CWND, RWND)
- Assuming RWND >> CWND

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

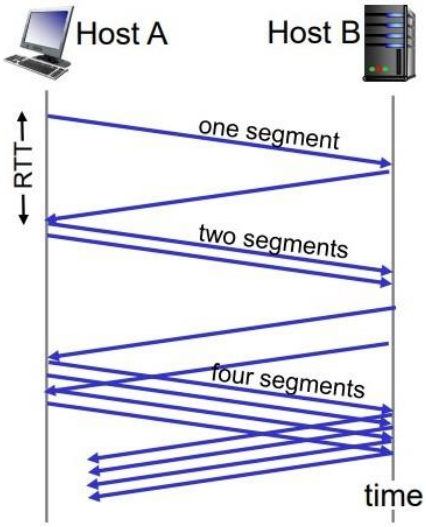




fast retransmit after sender receipt of triple duplicate ACK

How does the sender detect congestion?	How does the sender adjust its sending rate?
Duplicate ACKs: isolated loss - Dup ACKs indicate network capable of delivering some segments Timeout: *much more serious* - Not enough dup ACKs - Must have suffered *several losses* How to adjust depends on which case	Basic idea: - Receipt of ACK (new data) = *increased rate* - Detection of loss = *decrease rate* How we increase/decrease rate dpends on phase of congestion control we're in: - Discovering available bottleneck bandwidth vs adjusting to bandwidth variations

TCP Slow Start (Bandwidth Discovery)



When connection begins, increase rate exponentially until first loss event:
- Initially `cwnd` = 1 MSS
- Double `cwnd` every RTT (all ACKs)
- Simpler implementation achieved by incrementing `cwnd` for every ACK received
    - `cwnd` += for each ACK

Summary: initial rate is slow but ramps up expoentially fast

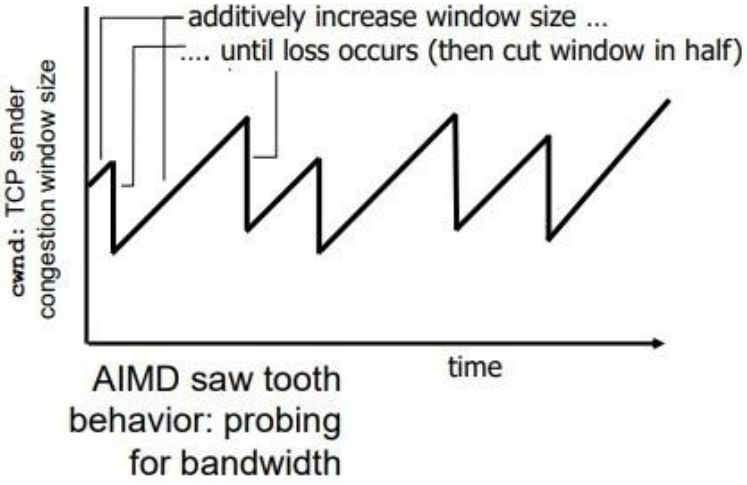
Adjusting to Varying Bandwidth

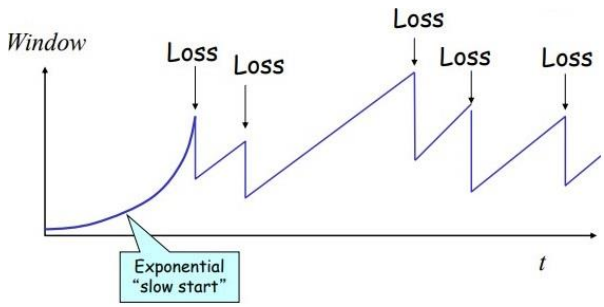
- Slow start gave an estimate of available bandwidth
- Now, want to track variations in this available bandwidth, oscillating around its current value
    - Repeated probing (rate increase) and back-off (rate decrease)
    - Known as Congestion Avoidance (CA)
- TCP uses: "Additive Increase Multiplactive Decrease" (*AIMD*)

AIMD

Approach: Sender increases transmission rate (window size), probing for usable bandwidth, until another congestion event occurs

- Additive increase: increase `cwnd` by 1 MSS every RTT until loss detected
    - For each successful RTT (all ACKs), `cwnd += 1` (in multiples of MSS)
    - Simple implementation: for each ACK, `cwnd += 1/cwnd` (since there are `cwnd/MSS` packets in window
- Multiplicative decrease: cut `cwnd` in half after loss





Slow-Start vs AIMD

- When does a sender stop Slow-Start and start Congestion Avoidance?
- Introduce a "slow start threshold" (`ssthresh`), initialised to a large value
- Convert to CA when cwnd = `ssthresh`, sender switches from slow-start to AIMD-style increase
    - On loss, ssthresh = `cwnd`/2

Implementation

State at sender
- cwnd (initialised to a small constant)
  - The slides use multiples of MSS
- Ssthresh (initialised to a large constant)
- [Also dupACKcount and timer, as before]

Events
- ACK (new data)
- dupACK (duplicate ACK for old data)
- Timeout

Event: ACK (new data)

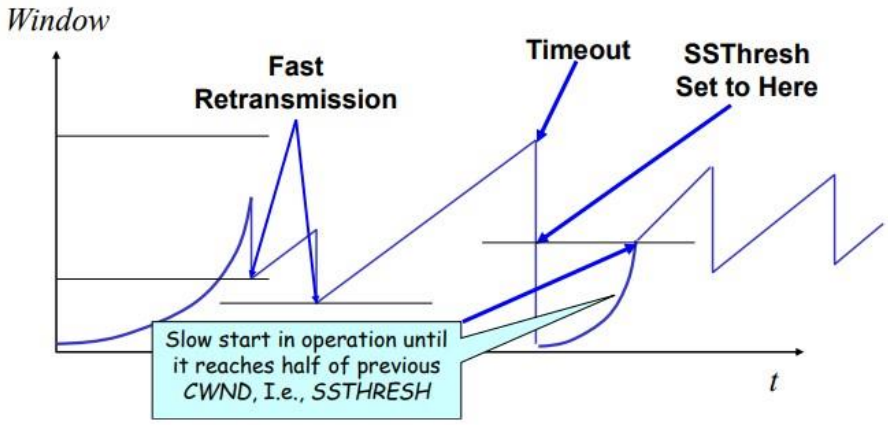
If `cwnd` < `ssthresh`: - cwnd += 1	Slow start phase
Else: - `cwnd` += 1/`cwnd` *(Hence after one RTT (all ACKs with no drops):* `cwnd` += 1	"Congestion Avoidance" phase (Additive increase)

Event: dupACK

- dupACKcount += 1
- If dupACKcount = 3 (*fast retransmit*)
  - ssthresh = `cwnd`/2
  - `cwnd` /= 2

Event: TimeOut

- On Timeout
  - ssthresh ← `cwnd`/2
  - `cwnd` ← 1



Slow-start restart: Go back to CWND = 1 MSS, but take advantage of knowing the previous value of CWND

TCP Flavours (Excluding those not on exam)

TCP-Tahoe

- `cwnd = 1` on triple dup ACK & timeout

TCP-Reno

- `cwnd = 1` on timeout
- `cwnd /= 2` on triple dup ACK

Final Phase: Fast Recovery

Context: Fast retransmission is slow in recovering from an isolated loss

The dilemma:

Consider a TCP connection with:

- cwnd = 10 packets (of size MSS, which is 100 bytes)
- Last ACK was for byte #101
    - (So, receiver is expecting next packet to have sequence # 101)

10 packets [101, 201, 301, …, 1001] are in flight

- Packet 101 is dropped
- What ACKS do they generate?
- How does the sender respond?

- ACK 101 (due to 201) cwnd=10 dupACK#1 (no xmit)
- ACK 101 (due to 301) cwnd=10 dupACK#2 (no xmit)
- ACK 101 (due to 401) cwnd=10 dupACK#3 (no xmit)
- RETRANSMIT 101 ssthresh=5 cwnd= 5
- ACK 101 (due to 501) cwnd=5 + 1/5 (no xmit)
- ACK 101 (due to 601) cwnd=5 + 2/5 (no xmit)
- ACK 101 (due to 701) cwnd=5 + 3/5 (no xmit)
- ACK 101 (due to 801) cwnd=5 + 4/5 (no xmit)
- ACK 101 (due to 901) cwnd=5 + 5/5 (no xmit)
- ACK 101 (due to 1001) cwnd=6 + 1/6 (no xmit)
- ACK 1101 (due to 101) ← only now can we transmit new packets
- Plus no packets in flight so ACK "clocking" (to increase CWND) stalls for another RTT

Solution: Fast Recovery

Concept: Grant the sender temporary 'credit' for each dupACK so as to keep packets in flight

If dupACKcount = 3

- `ssthresh = cwnd/2`
- `Cwnd = ssthresh + 3`

While in fast recovery

- `Cwnd += 1` for each additional duplicate ACK

Exit fast recovery after receiving new ACK

- Set `cwnd = ssthresh`

Example:

Picture the same example as above - TCP connection with:

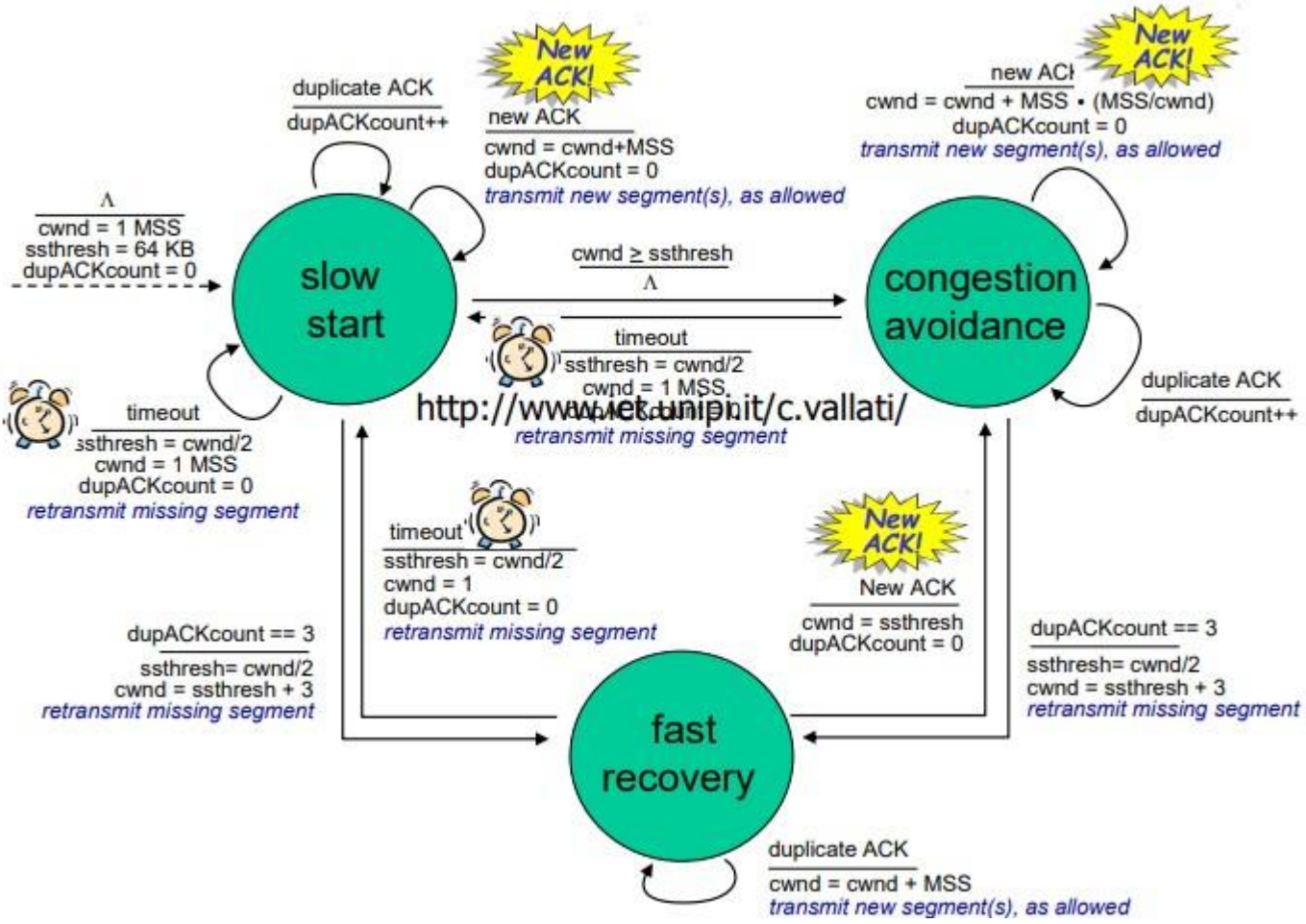
- cwnd = 10 packets (of size MSS, which is 100 bytes)
- Last ACK was for byte #101
    - (So, receiver is expecting next packet to have sequence # 101)

10 packets [101, 201, 301, …, 1001] are in flight

- Packet 101 is dropped

- ACK 101 (due to 201) cwnd=10 dup#1
- ACK 101 (due to 301) cwnd=10 dup#2
- ACK 101 (due to 401) cwnd=10 dup#3
- REXMIT 101 ssthresh=5 cwnd= 8 (5+3)
- ACK 101 (due to 501) cwnd= 9 (no xmit)
- ACK 101 (due to 601) cwnd=10 (no xmit)
- ACK 101 (due to 701) cwnd=11 (xmit 1101)
- ACK 101 (due to 801) cwnd=12 (xmit 1201)
- ACK 101 (due to 901) cwnd=13 (xmit 1301)
- ACK 101 (due to 1001) cwnd=14 (xmit 1401)
- ACK 1101 (due to 101) cwnd = 5 (xmit 1501) ← exiting fast recovery
- Packets 1101-1401 already in flight
- ACK 1201 (due to 1101) cwnd = 5 + 1/5 ← back in congestion avoidance

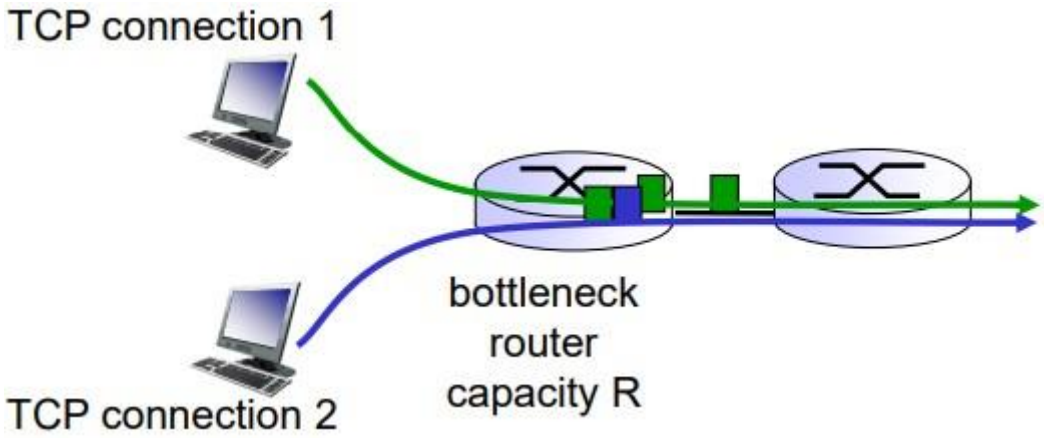
Summary: TCP Congestion Control



TCP Fairness

Fairness goal:

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



Why AIMD?

Some rate adjustment options: Every RTT, we can:

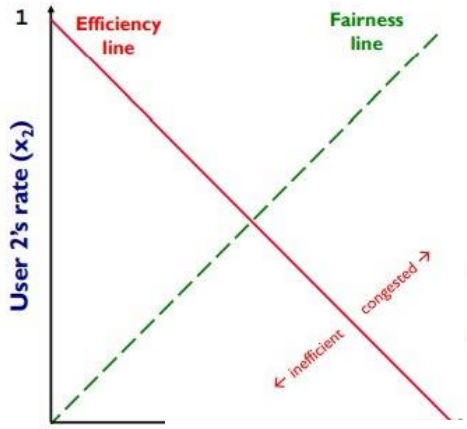
- Multiplicative increase/decrease: cwnd → a*cwnd
- Additive increase/decrease: cwnd → cwnd + b

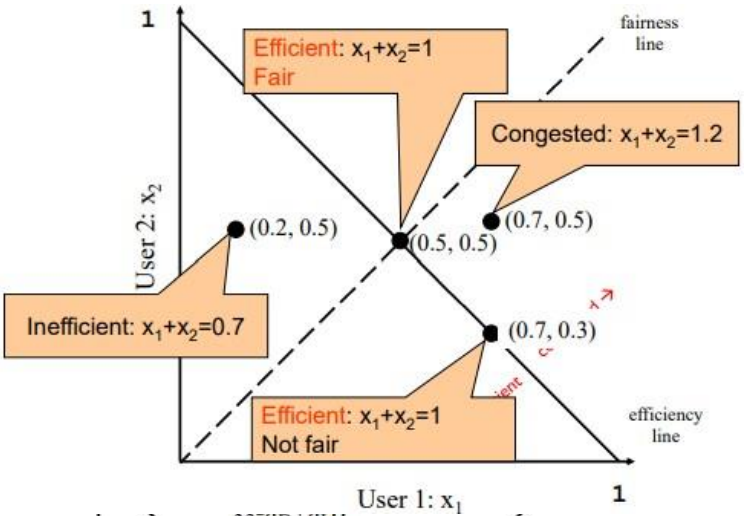
Four alternatives:
- AIAD: gentle increase, gentle decrease
- AIMD: gentle increase, drastic decrease
- MIAD: drastic increase, gentle decrease
- MIMD: drastic increase, drastic decrease

Simple model of Congestion Control

Example:

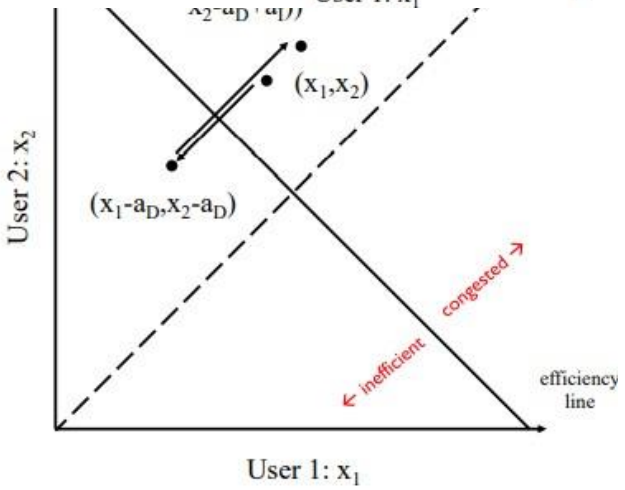
- Two users
  - rates x_1 and x_2
- Congestion when $x_1+x_2 > 1$
- Unused capacity when $x_1+x_2 < 1$
- Fair when $x_1 = x_2$



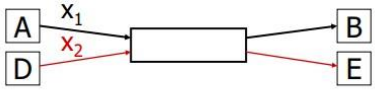


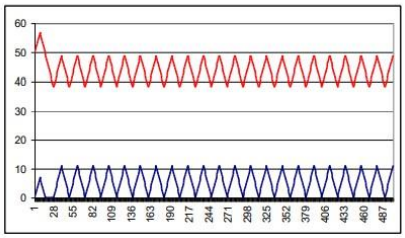
AIAD (Gentle increase, gentle decrease)

- Increase: $x + a_I$
- Decrease: $x - a_D$
- Does not converge to fairness



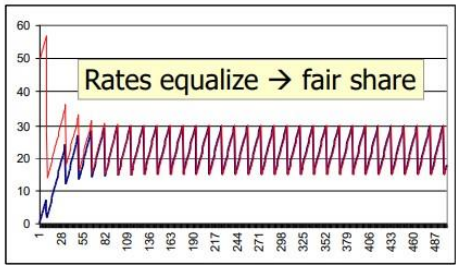
Sharing dynamics:



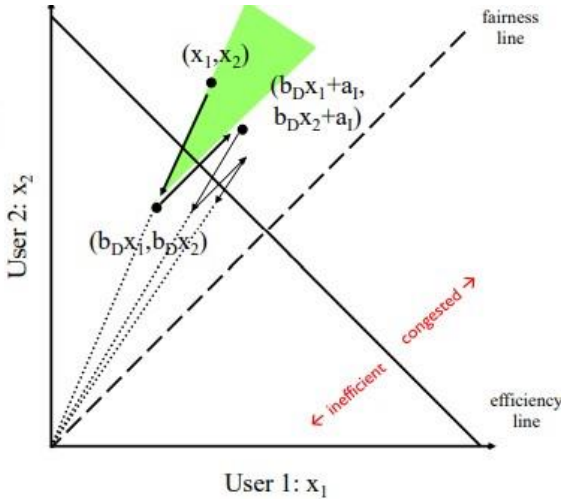


AIMD (gentle increase, drastic decrease)

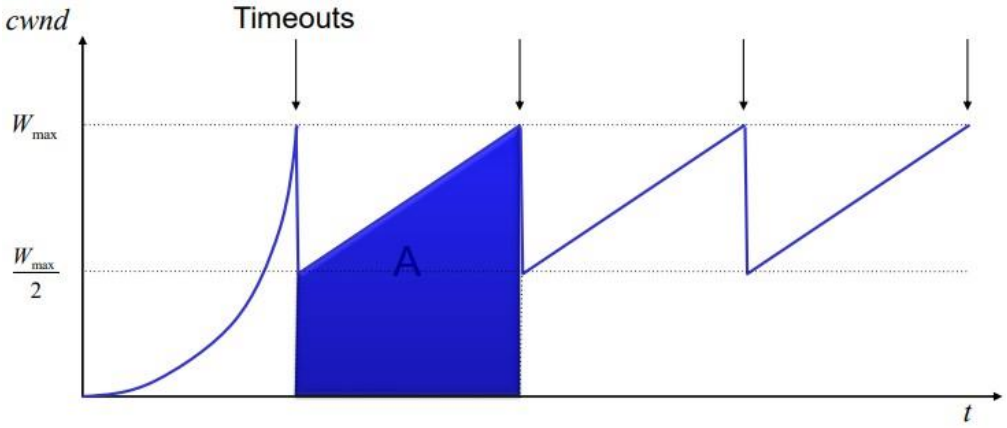
Sharing dynamics:



- Increase: $x+a_I$
- Decrease: $x*b_D$
- Converges to fairness

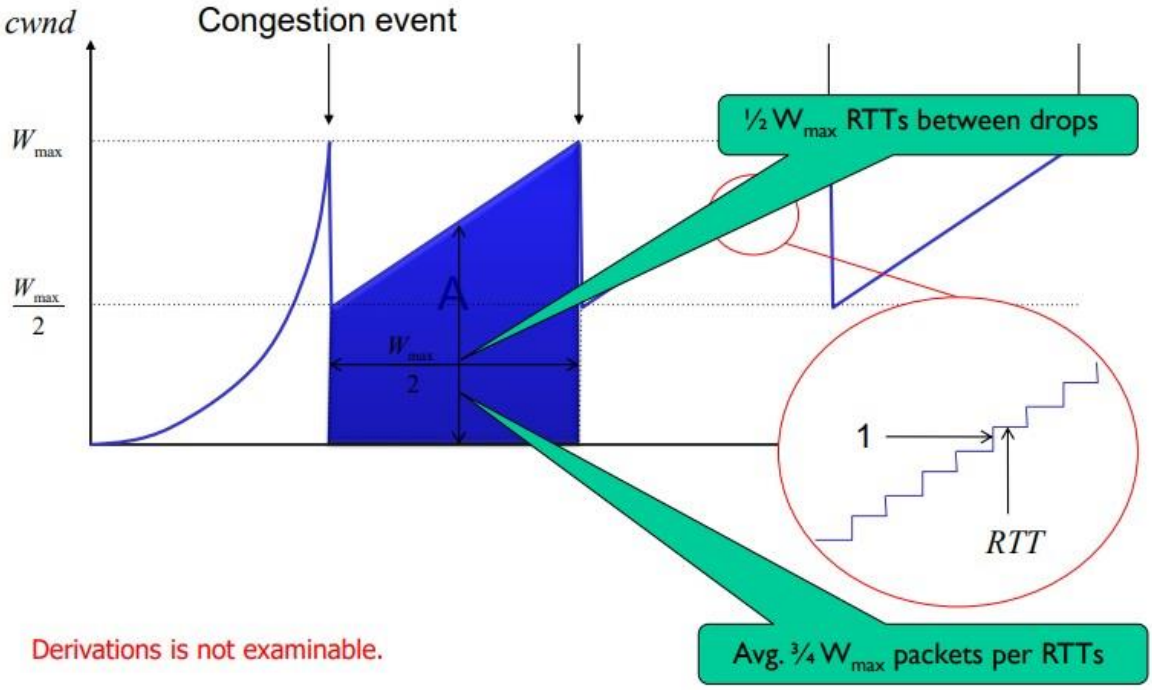


Simple model for TCP Throughput:



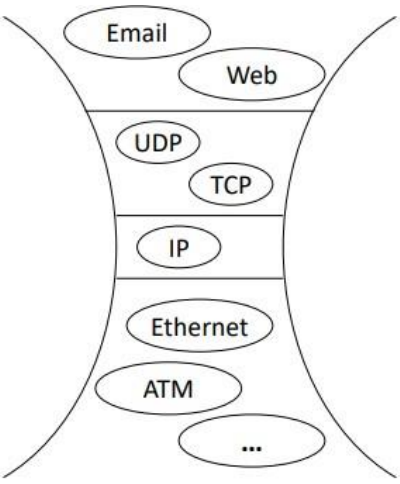
Packet drop rate, $p = 1/A$, where $A = \frac{3}{8}W_{max}^2$

Throughput, $B = \frac{A}{\frac{W_{max}}{2}RTT} = \sqrt{\frac{3}{2}}\frac{1}{RTT\sqrt{p}}$



Derivations is not examinable.

Network Layer: Data Plane



Outline – our goal:

- Understand principles behind network layer services, focusing on data plane:
  - Network layer service models
  - Forwarding versus routing
  - Addressing
- Instantiation, implementation in the internet
  - IP, NAT, ICMP

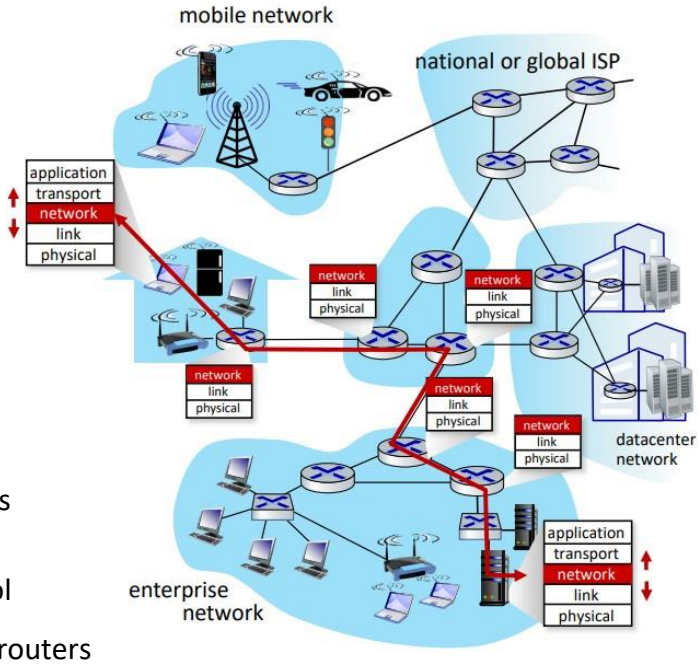
Internetworking

Routers forward packets from source to destination

- May cross many separate networks along the way

All packets use a common *Internet Protocol*

- Any underlying data link protocol
- Any higher layer transport protocol



Network Layer Services and Protocols

Transport segment from sending to receiving host

- Sender: encapsulates segments into datagrams, passes to link layer.
- Receiver: delivers segments to transport layer protocol

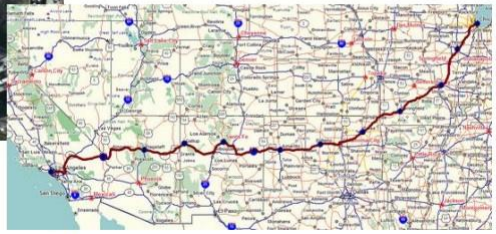
Network layer protocols in *every Internet device*: hosts, routers

Routers:

- Examines header fields in all IP datagrams passing through it
- Moves datagrams from input ports to output ports to transfer datagrams along end-end path.

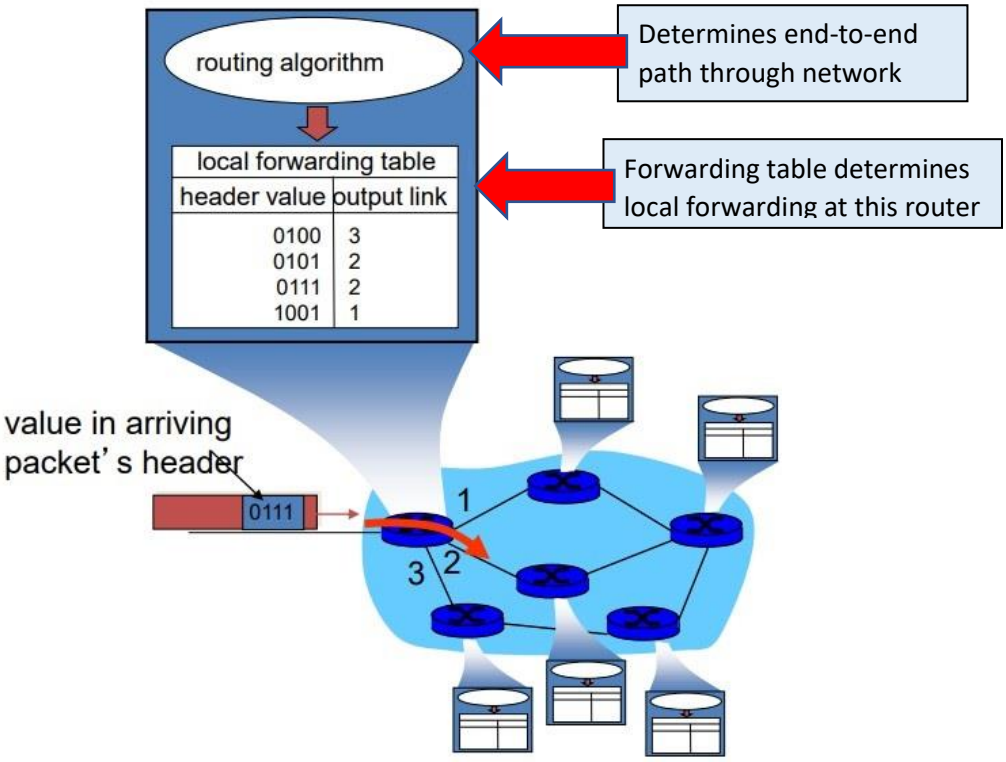
Two key network layer functions	
Forwarding: Move packets from a router's input link to appropriate router output link. Analogy: Process of getting through single interchange	**Routing**: Determine route taken by packets from source to destination (*using routing algorithms*). Analogy: Process of *planning trip* from source to destination

forwarding



routing

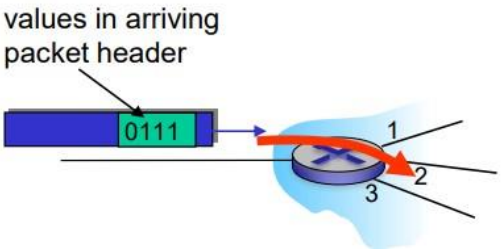
Interplay between Routing and Forwarding



Network Layer: Data Plane & Control Plane

Data plane:

- Local, per router function
- Determines how datagram arriving on router input port is forwarded to router output port

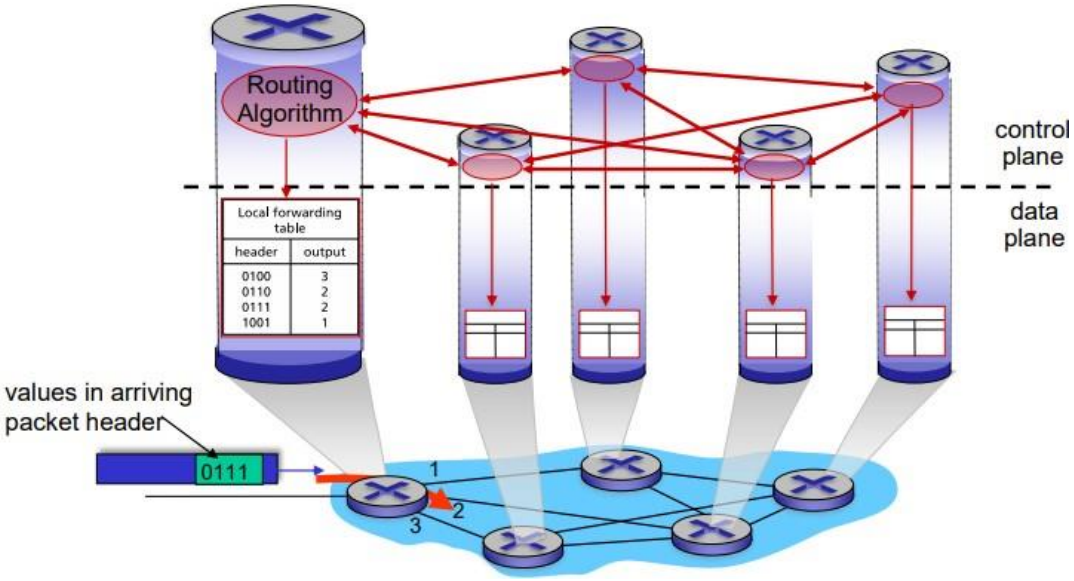


Control plane:

- Network-wide logic
- Determines how datagram is router among routers along end-to-end path from source host to destination host
- Two control-plane approaches:
  - Traditional routing algorithms – implemented in routers
  - Software-defined networking (SDN) – Implemented in (remote) servers (Not on exam)

Per-Router Control Plane

Individual routing algorithm components *in each and every router* interact in the control plane.



Network Layer: Service Model

Q: What service model for "channel" transporting datagrams from sender to receiver?

Internet "Best Effort" Service Model

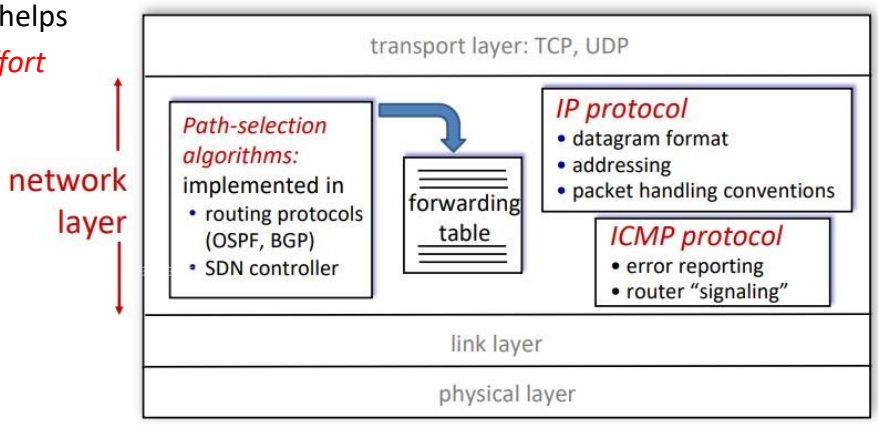
No guarantees on:

i) Successful datagram delivery to destination
ii) Timing or order of delivery
iii) Bandwidth available to end-to-end flow.

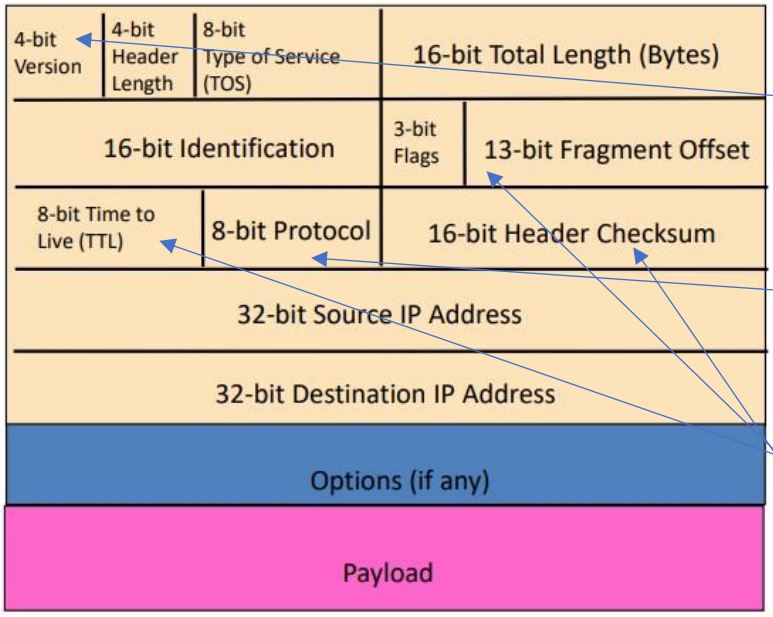
Best Effort Service:

- Simplicty of mechanism has allowed internet to be widely deployed and 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 (data centres, CDNs) connecting close to clients' networks, allow services to be provide from multiple locations.
- Congestion control of "elastic" services helps

Its hard to argue with success of best-effort service model



IP Datagram Format:



Fields for reading packet correctly:

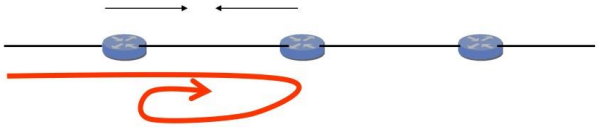
- 4-bit version (IPv4 or v6)
- Num 32-bit words in header
- Total length = bytes in packet (max = 65535 (2^{16} -1) bytes)

8-bit Protocol: Telling end-host how to handle packet. Important for demultiplexing at receiving host

Potential problems:
Loop – TTL
Header Corrupted – Checksum
Packet too large – Fragmentation

Preventing Loops (TTL)

- Forwarding loops cause packets to cycle for a long time
  - As these accumulate, eventually consume all capacity



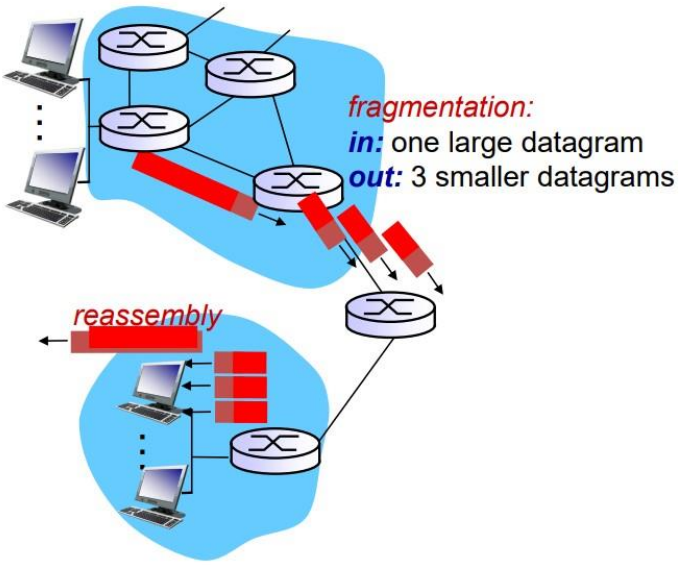
- Time-to-Live (TTL) Field (8 bits)
  - Decremented at each hop, packet discarded if reaches 0 ("time exceeded" message sent to source)
  - Recommended default val = 64

Special Handling

- "Types of Service" (TOS), or "Differentiated Services Code Point" (DSCP) – 8 bits
  - Allow packets to be treated differently based on needs
  - E.g. low delay for audio, high bandwidth for bulk transfer
  - Not widely used
- Options (not often used)

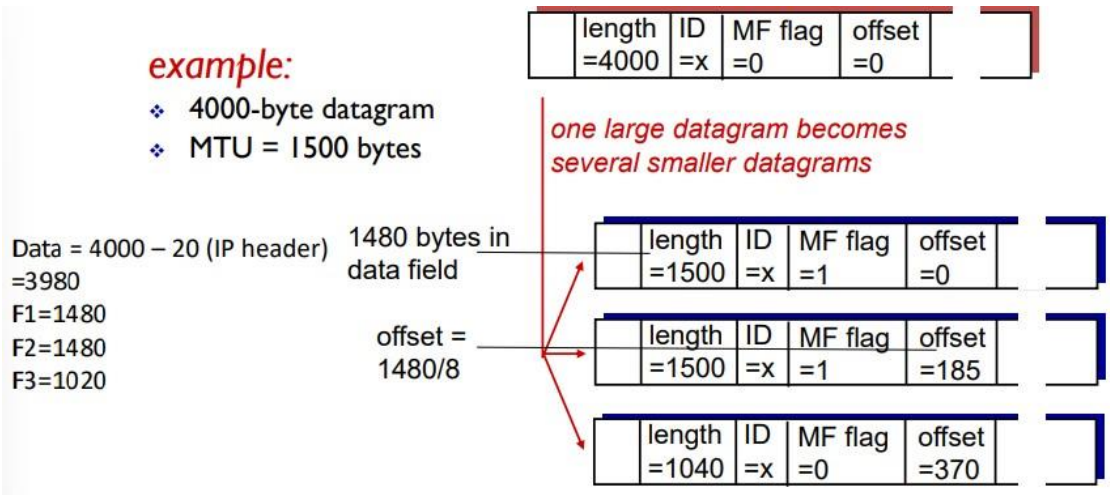
Header Corruption (Checksum – 16 bits)

- Only computed over packet header, method is same as UDP/TCP checksum
- If not correct router discards packets (avoid acting on bogus information)
- Recalculated at every router



IP Fragmentation, Reassembly

- Network links have MTU (max.transfer size) – largest possible link-level frame
  - Different link types, different MTUs
- Large IP datagram divided ("fragmented") within net
  - One datagram becomes several datagrams
  - "Reassembled" only at final destination
  - IP header bits used to identify, order related fragments



IPv4 Fragmentation procedure

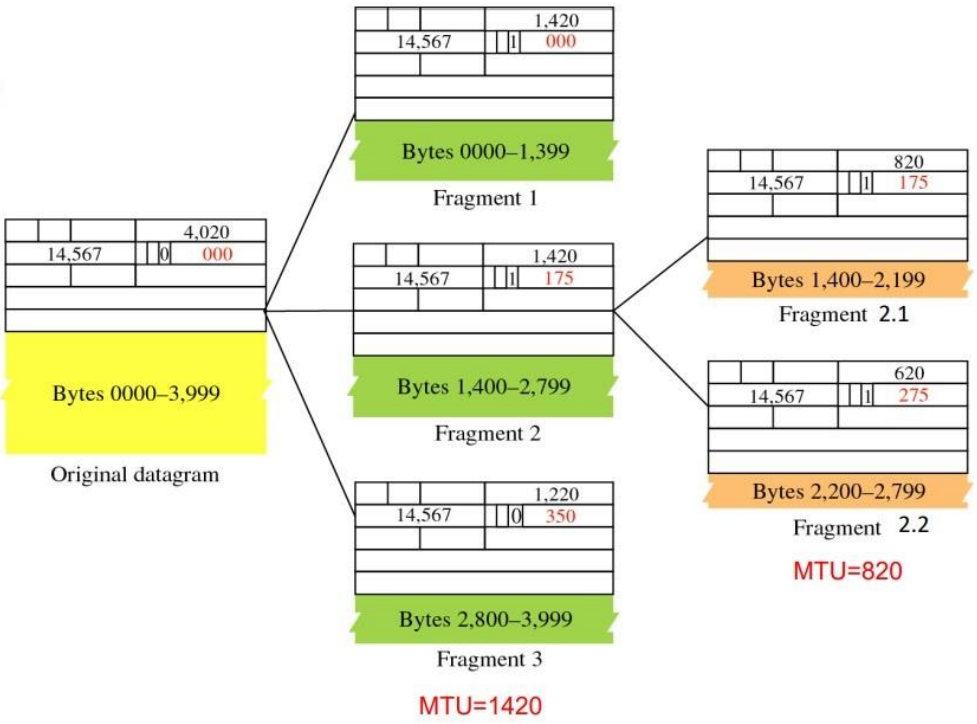
Fragmentation:

- Router breaks up datagram in size that output link can support
- Copies IP header to pieces
- Adjust length on pieces
- Set offset to indicate position
- Set MF (more fragments) flag on pieces except the last
- Re-compute checksum

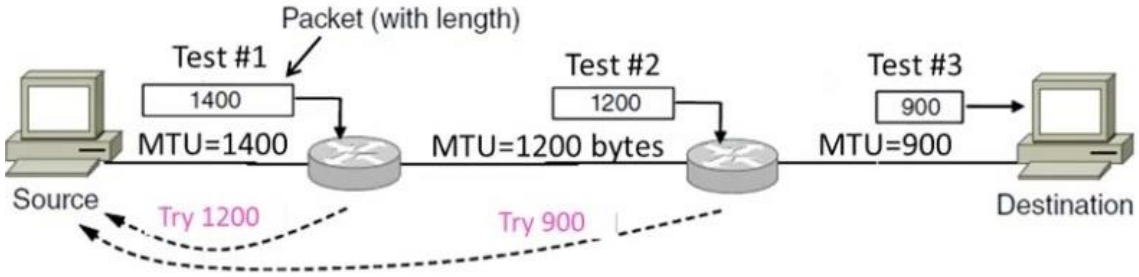
Re-assembly:

- Receiving host uses identification field with MF and offsets to complete the datagram.

Fragmentation of fragments also supported.



Path MTU Discovery Procedure



Host:
- Sends a big packet to test all routers in path to the destination can support or not
- Set DF (Do Not Fragment) flag

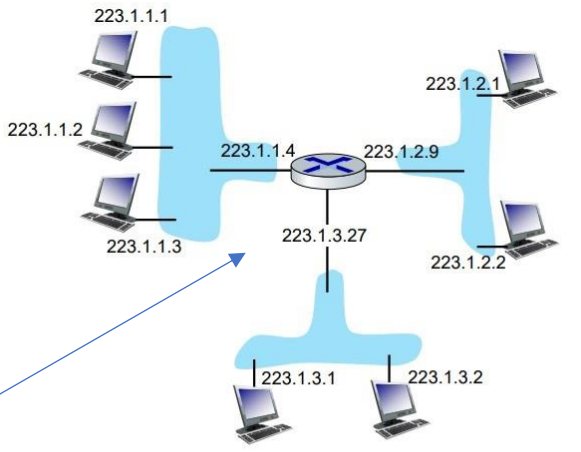
Routers:
- Drops the packet if it is too large (as DF is set)
- Provides feedback to Host with ICMP message telling the maximum supported size

IP Addressing: Introduction

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 interface
- Host typically has one or two interfaces (e.g. wired Ethernet, wireless 802.11)



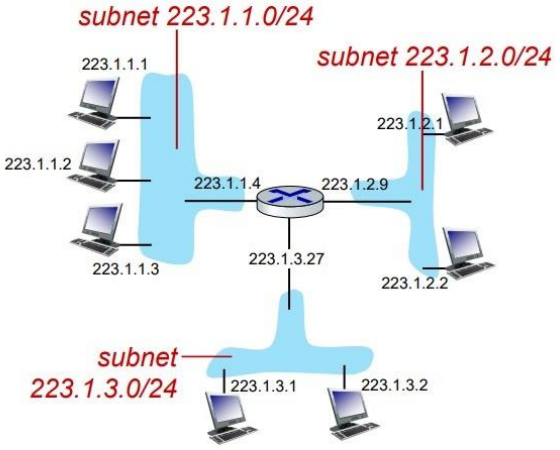
Networks consisting of 3 subnets

dotted-decimal IP address notation:

223.1.1.1 = 11011111 00000001 00000001 00000001

223 1 1 1

Subnets



subnet mask: /24
(high-order 24 bits: subnet part of IP address)

What is a subnet?

- Device interfaces that can physically reach each other without passing through an intervening router

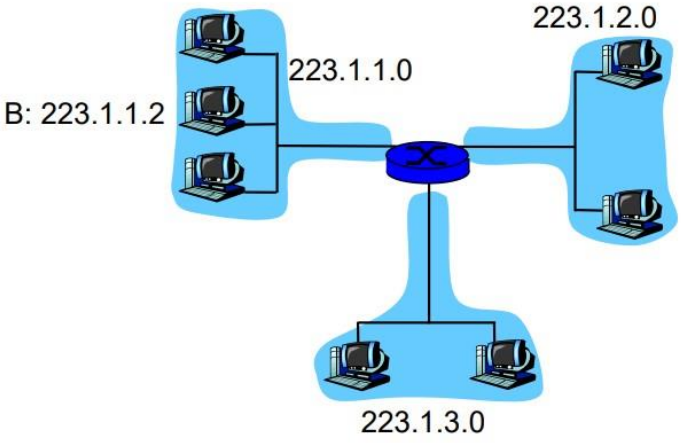
IP addresses have structure:

- Subnet part: devices in same subnet have common high order bits
- Host part: remaining low order bits

Recipe for defining subnets:

- Detach each interface from its host or router, creating "islands" of isolated networks
- Each isolated network is called a *subnet*

Network Mask



Mask:

- Used in conjunction with the low network address to indicate how many higher order bits are used for the network part of the address.
  - Bitwise AND
- 233.1.1.0 with mask 255.255.255.0

Host B	Dot-decimal address	Binary
IP address	223.1.1.2	11011111.00000001.00000001.00000010
Mask	255.255.255.0	11111111.11111111.11111111.00000000
Network Part	223.1.1.0	11011111.00000001.00000001.00000000
Host Part	0.0.0.2	00000000.00000000.00000000.00000010

Broadcast Address:

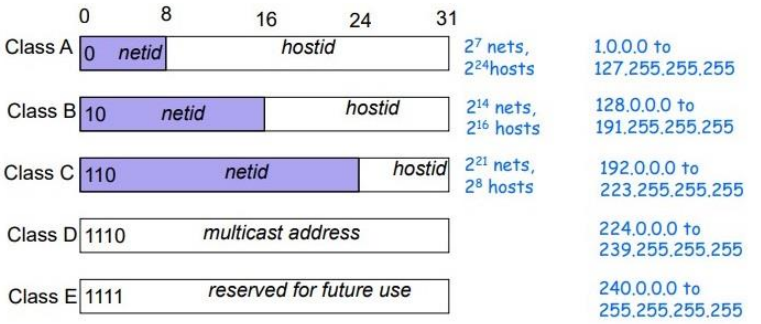
- Host part is all 111's (*E.g. 233.1.1.255*)

Network Address:

- Host part is all 0000's (*E.g. 233.1.1.0*)

Both are typically not assigned to any host.

(Early Design) Classful Addresses:



Problem: Networks only come in three sizes!

What are the issues?

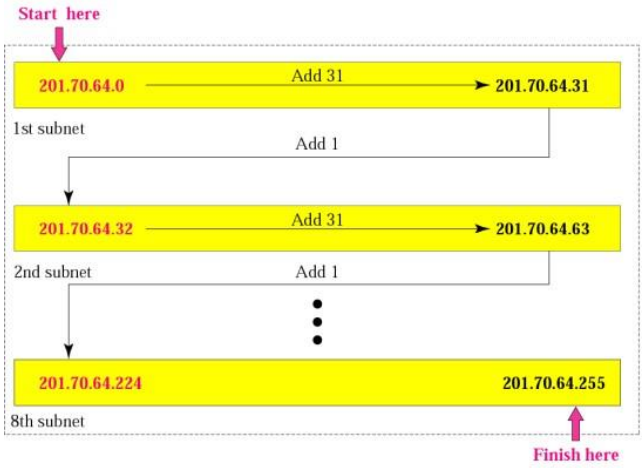
An organisation requires 6 nets each of size 30:
Does it have to buy 6 class C address blocks?

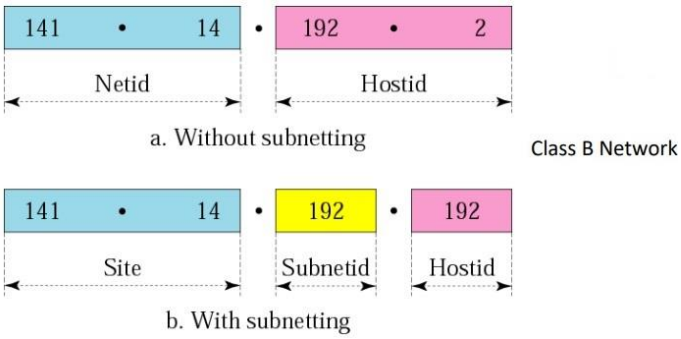
An organisation requires 512 addresses:
How many IP addresses should it buy?

Subnetting

- Subnetting is the process of dividing the class A, B, or C network into more manageable chunks that are suited to your network's size and structure.
- Subnetting allows 3 levels of hierarchy
    - netid, subnetid, hostid

The number of addresses in each subnet is 2^5 or 32.

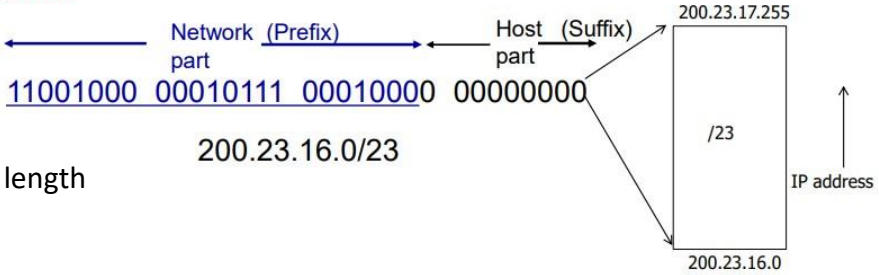




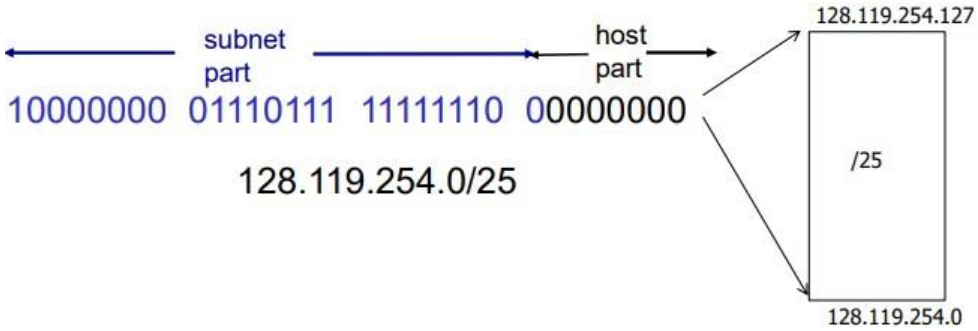
-Original netid remains the same and designates the site
-Subnetting remains transparent outside the site

-The process of subnetting simply extends the point where the 1's of Mask stop and 0's start.
-You are sacrificing some Host ID bits to gain Network ID bits

Today's Addressing: CIDR



CIDR: **C**lassless **I**nter**D**omain **R**outing
- Network portion of address of arbitrary length
- Address format: `a.b.c.d/x`
- x is # bits in network portion of address
-



-

IP Addresses: How to get one?

This comes down to 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)?

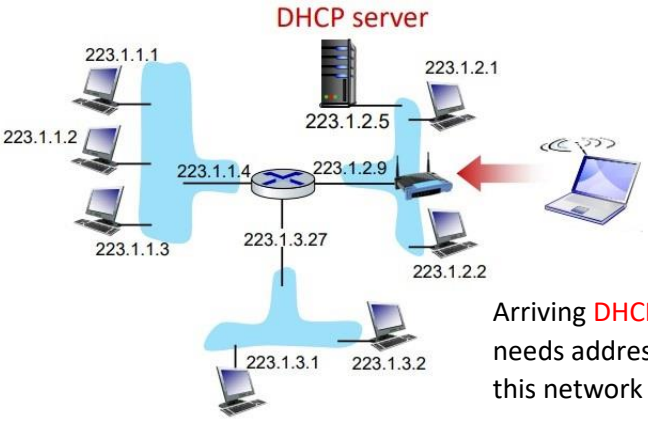
How does *host* get IP address?

- Hard coded by sysadmin in config file (e.g. /etc/rc.config in UNIX)
- **DHCP**: Dynamic Host Configuration Protocol – dynamically get address from a server upon joining network.
    - "Plug and Play"
    - 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

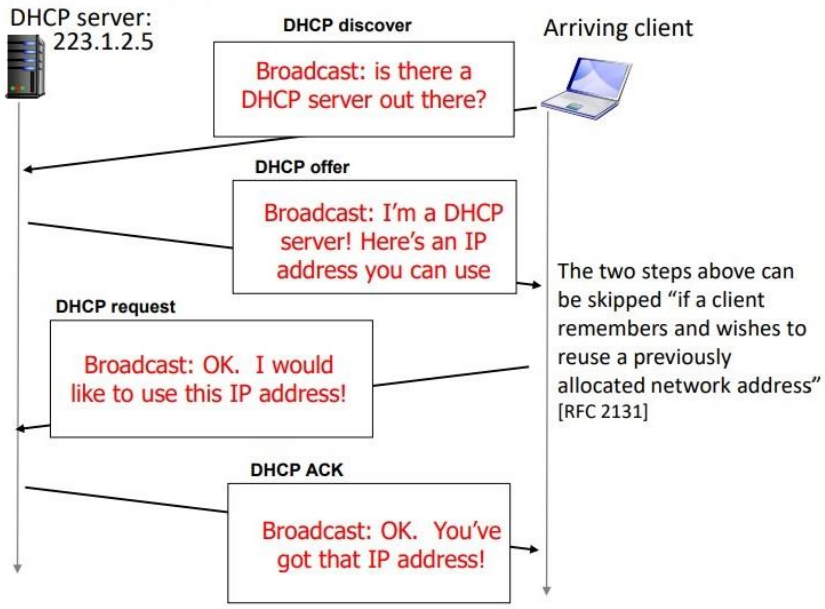
DHCP Overview

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

Typically, DHCP server will be co-located in router, serving all subnets to which router is attached



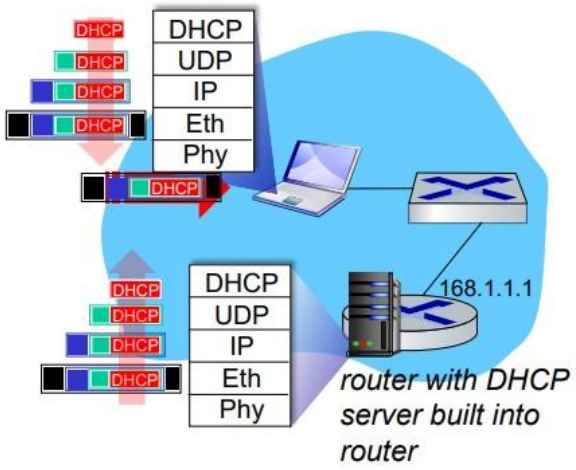
Arriving DHCP client needs address in this network



DHCP: More than IP Addresses

DHCP can return more than just allocated IP address on subnet:

- Address of first-hop router for client
- Name and IP address of DNS server
- Network mask (indicating network versus host portion of address)



DHCP example:

1. Connecting laptop will use DHCP to get IP address, address of first-hop router, address of DNS server.
2. DHCP Request message encapsulated in UDP, encapsulated in IP, encapsulated in Ethernet
3. Ethernet frame broadcast (dest: 0xFFFFFFFFFFFF) on LAN, received at router running DHCP server.
4. Ethernet demux'ed to IP demux'ed, UDP demux'ed to DHCP.
5. DCP server formulates DHCP ACK containing client's IP address, IP address of first-hop router for client, name & IP address of DNS server
6. Encapsulated DHCP server reply forwarded to client, demuxing up to DHCP at client Client now knows its IP address, name and IP address of DNS server, IP address of its first-hop router.

How does *network* get subnet part of IP address?

- Gets allocated portion of its provider ISP's address space

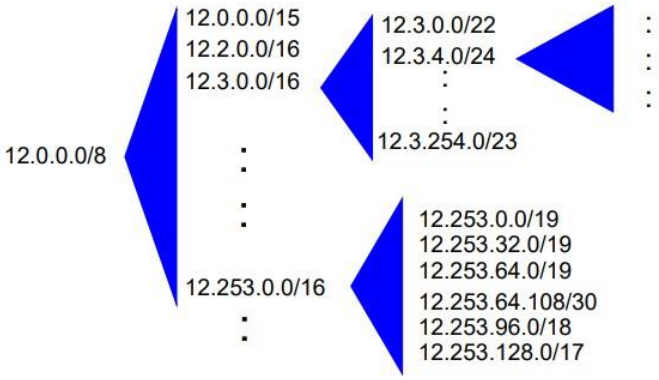
```
ISP's block:    11001000  00010111  00010000  00000000  200.23.16.0/23
```

ISP can then allocate out its address space in 8 blocks

```
Organisation 0  11001000  00010111  00010000  00000000  200.23.16.0/23
Organisation 1  11001000  00010111  00010010  00000000  200.23.18.0/23
Organisation 0  11001000  00010111  00010100  00000000  200.23.20.0/23
      …               …           …         …         …          …
Organisation 7  11001000  00010111  00011110  00000000  200.23.30.0/23
```

CIDR: Addresses allocated in contiguous prefix chunks

Recursively break down chunks as get closer to host

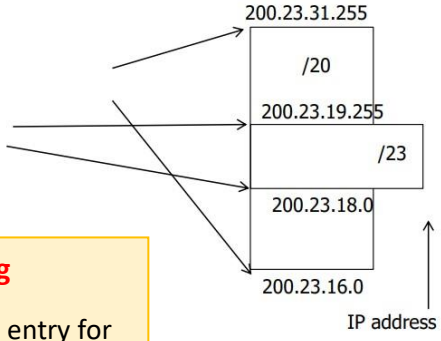


Hierarchical Addressing: Route Aggregation

- Organisation 1 moves from Fly-by-Night-ISP to ISPs-R-Us
- ISPs-R-Us now advertises a more specific route to Organisation 1

How will this work?

- Routers in the internet will have two entries in their tables:
  - 200.23.16.0/20 (Fly-by-Night
  - 200.23.18.0/23 (ISPs-R-Us)
- **Longest prefix match**



Longest Prefix Matching

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

Destination Address Range	Link interface
11001000 00010111 00010*** ********	0
11001000 00010111 00011000 ********	1
11001000 00010111 00011*** ********	2
otherwise	3

examples:

DA: 11001000 00010111 00010110 10100001 which interface?

DA: 11001000 00010111 00011000 10101010 which interface?

More on IP Addresses

- IP addresses are allocated as blocks and have geographical significance
- It is possible to determine the geographical location of an IP address

Q: How does an ISP get block of addresses?
A: ICANN - Internet Corporation for Assigned Names and Numbers

Made-up Example

- ICANN gives APNIC several /8s
- APNIC gives Telstra one /8, **129/8**
  - Network Prefix: **10000001**
- Telstra gives UNSW a /16, **129.94/16**
  - Network Prefix: **1000000101011110**
- UNSW gives CSE a /24, **129.94.242/24**
  - Network Prefix: **100000010101111011110010**
- CSE gives me a specific address **129.94.242.51**
  - Address: **10000001010111101111001000110011**

IANA works through Regional Internet Registries (RIRs):

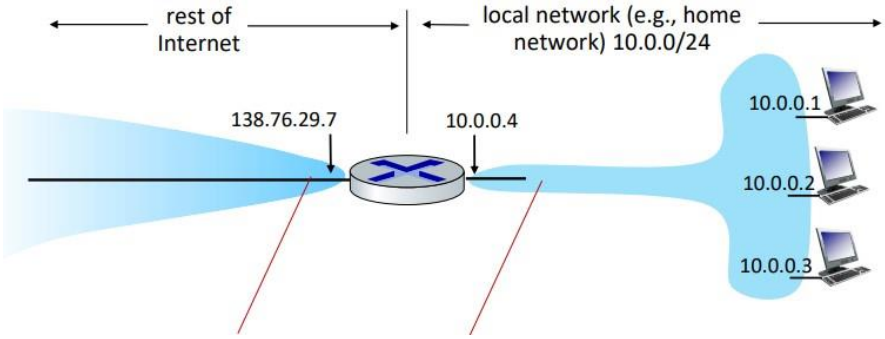
- ARIN — American Registry for Internet Numbers
- RIPE NCC — IRéseaux IP Européens Network Coordination Centre
- APNIC — Asia-Pacific Network Information Center
- LACNIC — Latin America and Caribbean Network Information Centre
- AfriNIC — African Network Information Centre

Private Addresses

Defined in [RFC 1918]:
- 10.0.0.0/8 (16,777,216 hosts)
- 172.16.0.0/12 (1,048,576 hosts)
- 192.168.0.0/16 (65536 hosts)

These addresses *cannot be routed*.
- *Anyone* can use them in a private network
- Typically used for NAT (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: 168.76.29.7, but *different* source port numbers

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

NAT: Network Address Translation

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 a 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 chance ISP without changing of devices in local network
- Security: devices inside local net not directly addressable or visible by outside world.

Nat has been controversial:

Routers "should" only process up to layer 3

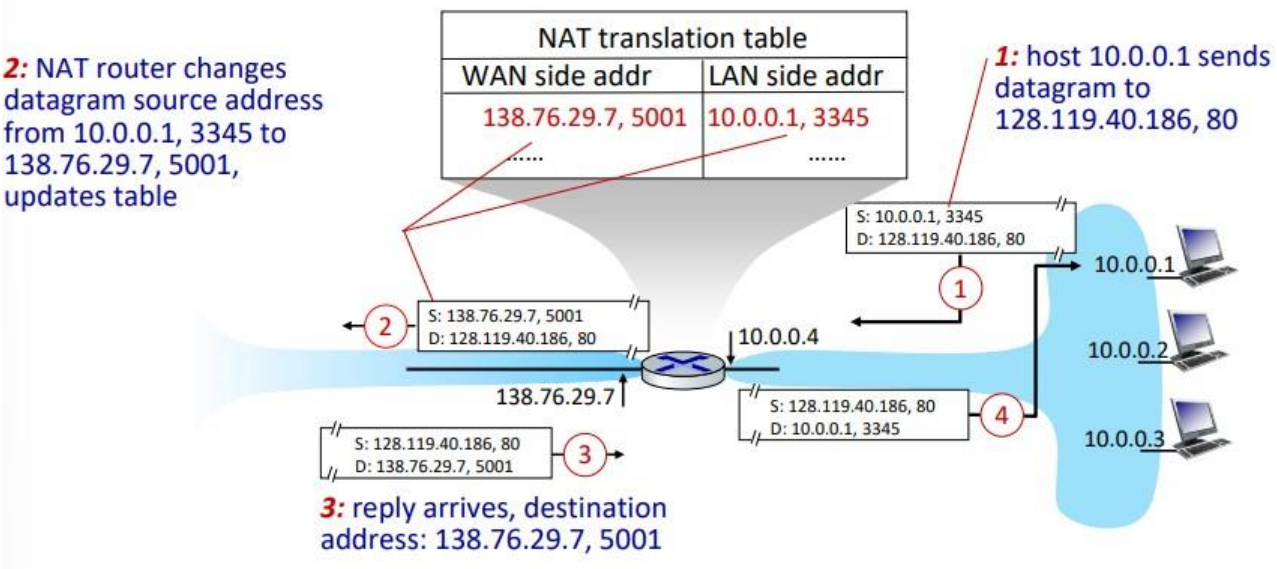
Address "shortage" should be solved by IPv6

Violates end-to-end argument (port # manipulation by network-layer device)

Nat traversal: what if client wants to connect to server behind NAT?

Despite that, NAT is extensively used in home and institutional nets, 4G/5G cellular nets

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



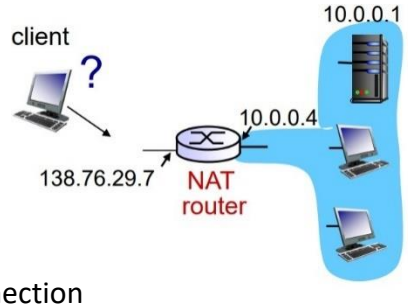
NAT: Practical Issues

NAT modifies port # and IP address
- *Requires recalculation of TCP and IP* checksum

Some applications embed IP address or port numbers in their message payloads
- DNS, FTP (PORT command), SIP, H.323
- For legacy protocols, NAT must look into these packets and translate the embedded IP addresses/port numbers
- What if these fields are encrypted – (SSL/TLS, IPSEC, etc.)?
- **Q: Why may NAT need to change TCP sequence number?**
- **A:** NAT changes the IP, and may result in a different data length, requiring sequence number as well.

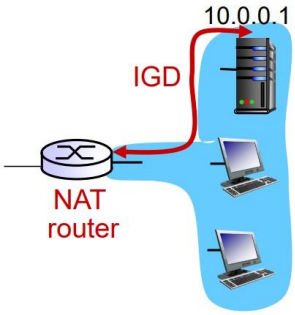
If applications change port numbers periodically, the *NAT must be aware*.

NAT Traversal Problem



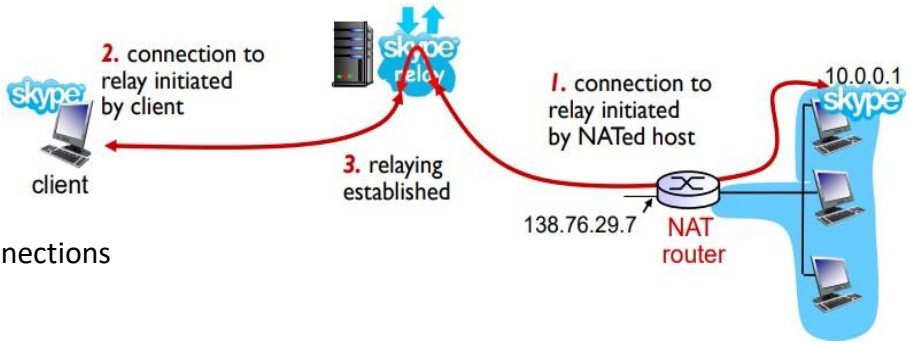
Client wants to connect to server with address 10.0.0.1:
- Server address 10.0.0.1 local to LAN
  (client can't use it as destination address)
- Only one external visible NAT'ed address: 138.76.29.7

Solution1: Inbound-NAT statically configure NAT to forward incoming connection at given port to server E.g. (138.76.29.7, port 2500) always forwarded to 10.0.0.1 port 25000



Solution2: Universal Plug and Play (UPnP) Internet Gateway Device (IGD) Protocol. Allows NAT'ed host to:
- Learn public IP address (138.76.29.7)
- Add/remove port mappings (with lease times)
  - *i.e. automate static NAT port map configuration*

Solution3: Relaying (used in Skype)
- NAT'ed client establishes connection to relay
- External client connects to relay
- Relay bridges packets between to connections



NAT: Devil in the Details

- Despite the problems, NAT has been widely deployed
- Most protocols can be successfully passed through a NAT, including VPN
- Modern hardware can easily perform NAT functions at > 100Mbps
- IPv6 is still not widely deployed commercially, so the need for NAT is real
- After years of refusing to work on NAT, the IETF has been deploying "NAT Control Protocols" for hosts
  - *Lot of practical variations – (Full-cone NAT, Restricted Cone NAT, Port Restricted Cone NAT, etc…)*

IPv6: Motivation

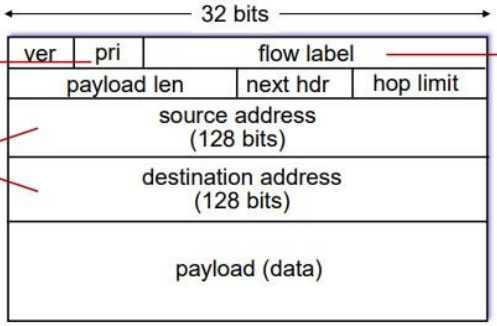
32-bit IPv4 address space would be completely allocated

Additional motivation:
- Speed processing/forwarding: 40-byte fixed length header
- Enable different network-layer treatment of "flows"

IPv6 datagram format:

priority: identify priority among datagrams in flow

128-bit IPv6 addresses



flow label: identify datagrams in same "flow." (concept of "flow" not well defined).

Transition from IPv4 to IPv6

What's missing compared with IPv4?

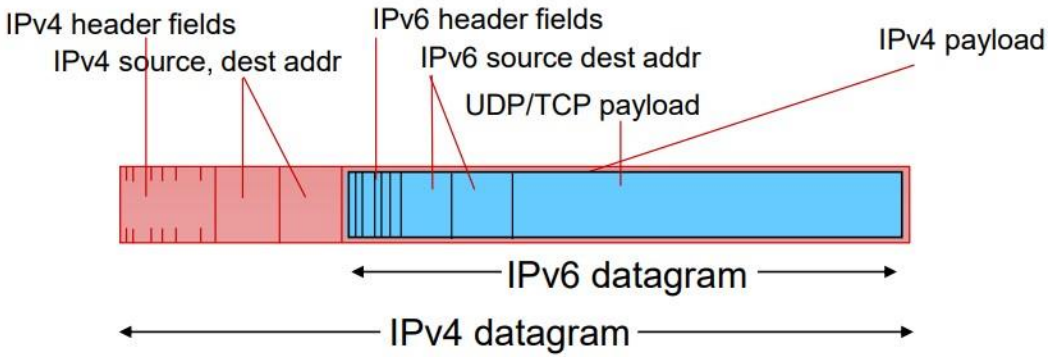
- No checksum (to speed processing at routers)
- No fragmentation/reassembly
- No options (available as upper-layer, next-header protocol at router)

Transitioning:

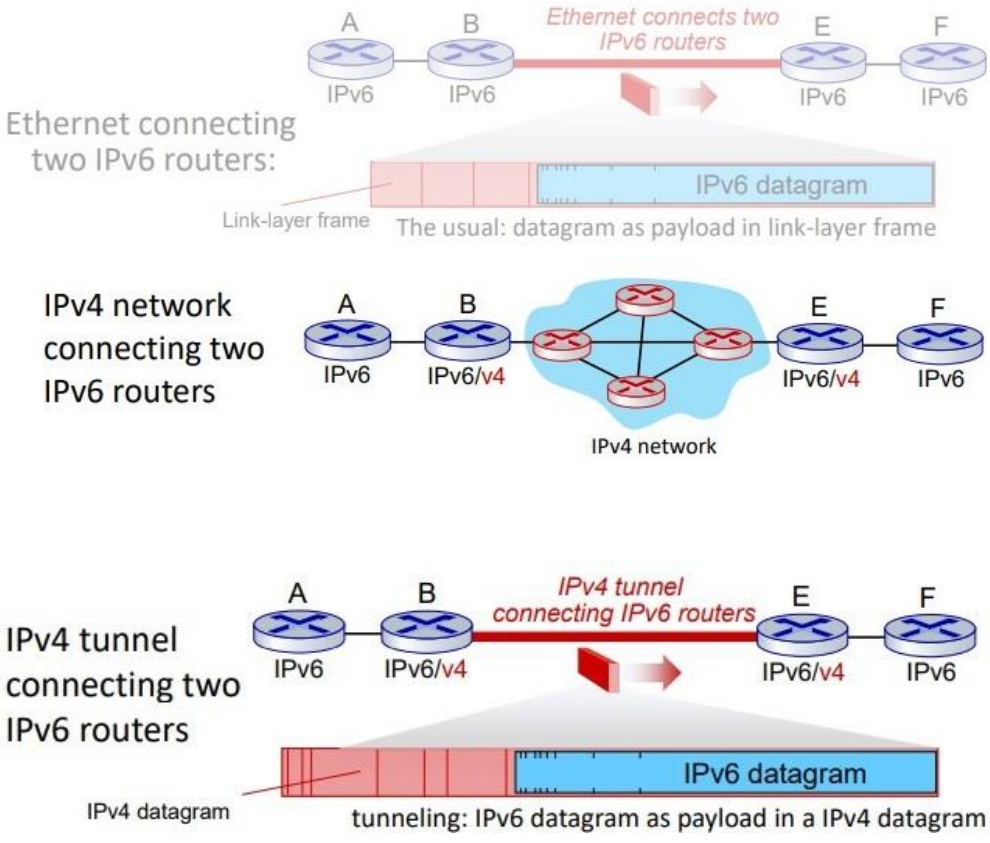
Not all routers can be upgraded simultaneously

- No "flag days"
- How will network operate with mixed IPv4 and IPv6 routers?

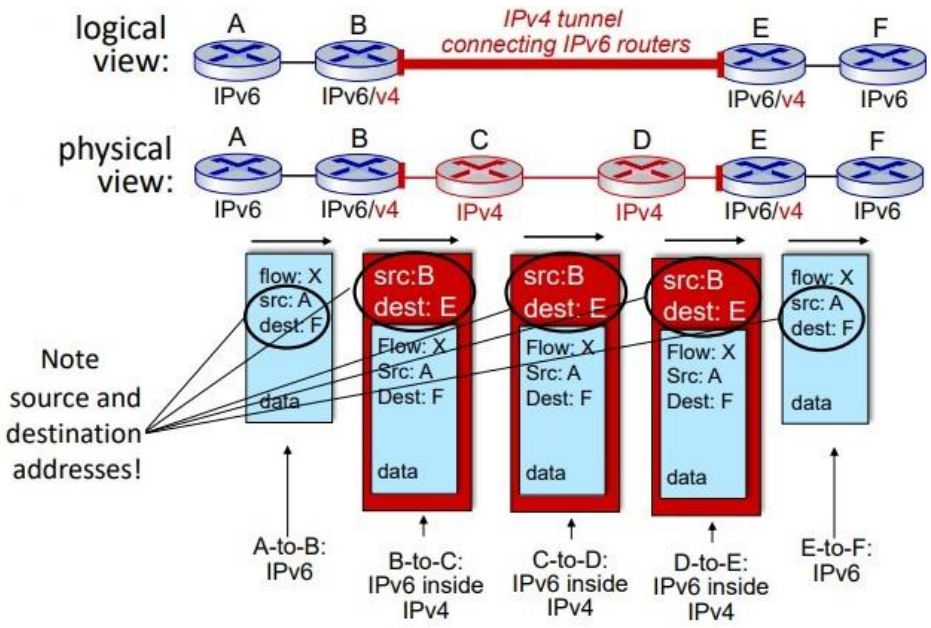
Tunnelling: IPv6 datagram carried as *payload* in IPv4 datagram among IPv4 routers ("packet within a packet"). Used extensively in other contexts (4G/5G).



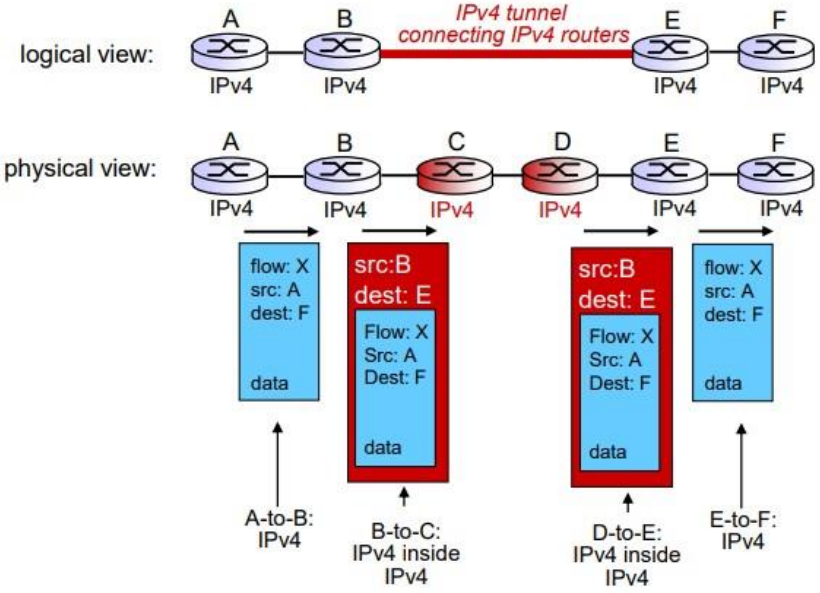
Tunneling and Encapsulation:



Tunneling (IPv4 to IPv6)



Tunneling (IPv4 over IPv4 – Used in VPNs)



IPv6 Adoption:

Google: 30% of clients access services via IPv6.
NIST: 1/3 of all US government domains are IPv6 capable

Network-layer Function

Forwarding: move packets from router's input to appropriate router output — Data plane

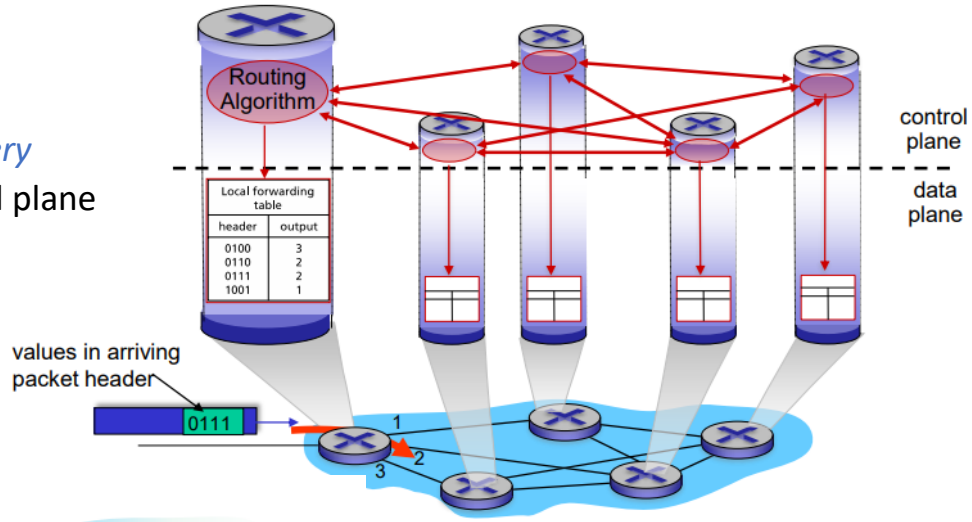
Routing: determine route taken by packets from source to destination — Control plane

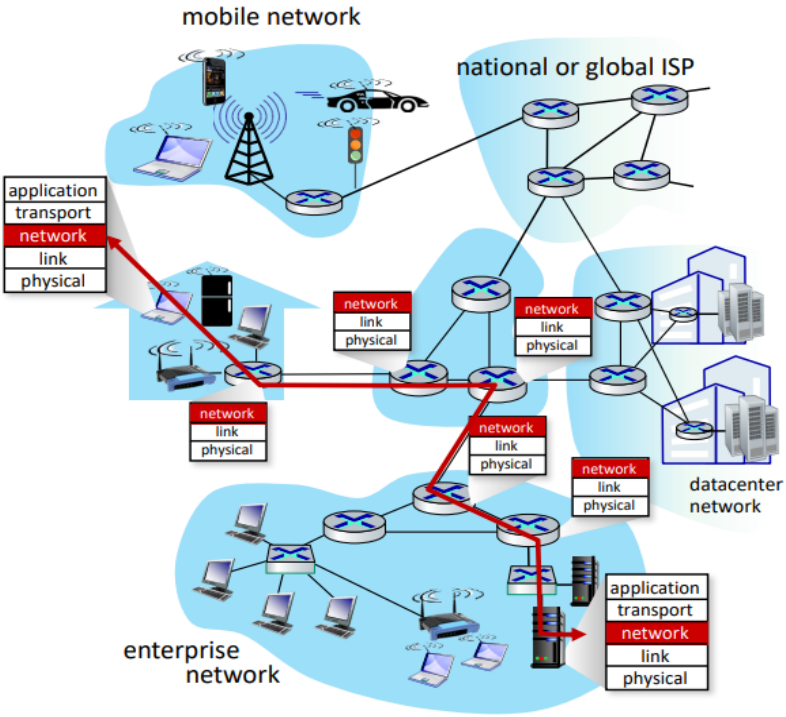
Two approaches to structuring network control plane:

- Per-router control (traditional)
- Logically centralized control (Software Defined Networking – not covered in exam)

Per-router control plane

Individual routing algorithm components *in each and every router* interact in the control plane





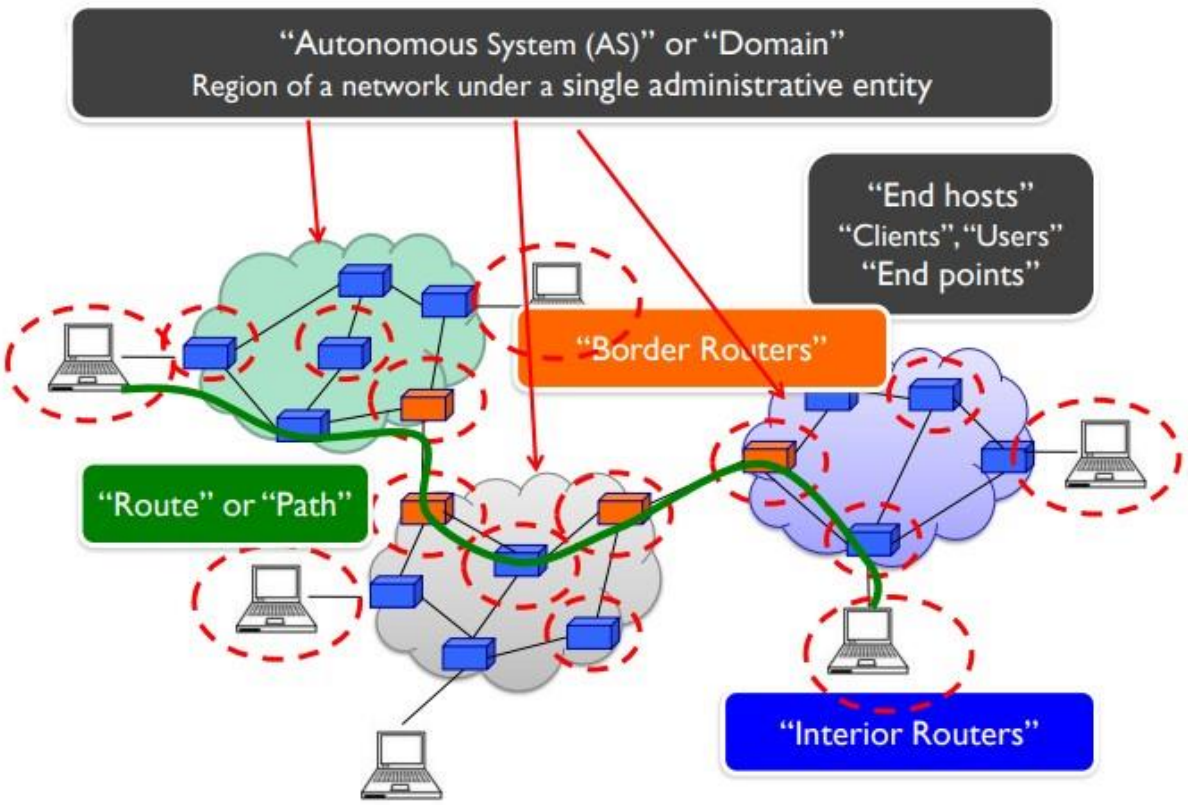
Routing protocols

Routing protocol goal: determine *"good paths"* (equivalently, routes), from sending hosts to receiving host, through network of routers

Path: sequence of routers packets traverse from given initial source host to final destination host.

"good": least "cost", "fastest", "least congested"

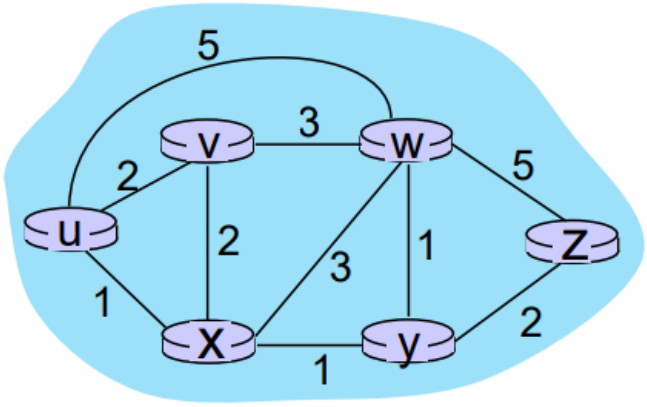
Routing: a "top-10" networking challenge!



Internet Routing

Works at two levels:

- Each AS runs an intra-domain routing protocol that establishes routes within its domain
  - AS – region of network under a single administrative entity
  - Link State, e.g., Open Shortest Path First (OSPF)
  - Distance Vector, e.g., Routing Information Protocol (RIP)
- ASes participate in an inter-domain routing protocol that establishes routes between domains
  - Path Vector, e.g., Border Gateway Protocol (BGP)



Graph abstraction: link costs

$c_{a,b}$: cost of *direct* link connecting *a* and *b*

e.g., $c_{w,z} = 5, c_{u,z} = \infty$

Cost defined by network operator –

Could always be 1, or inversely related to bandwidth, or inversely related to congestion.

Graph $G = (N, E)$

- N: set of routers = $\{u, v, w, x, y, z\}$
- E: set of links= $\{(u,v), (u,x), (v,x), (x,w), (x,y), (w,y), (w,z), (y,z)\}$

Routing algorithm classification

Global: all routers have complete topology, link cost info.

"Link State" algorithms

X axis = How fast routes change

Y axis = Centralisation of information

Static: Routes change slowly over time

Dynamic: Routes change more quickly

Periodic updates or in response to link cost changes

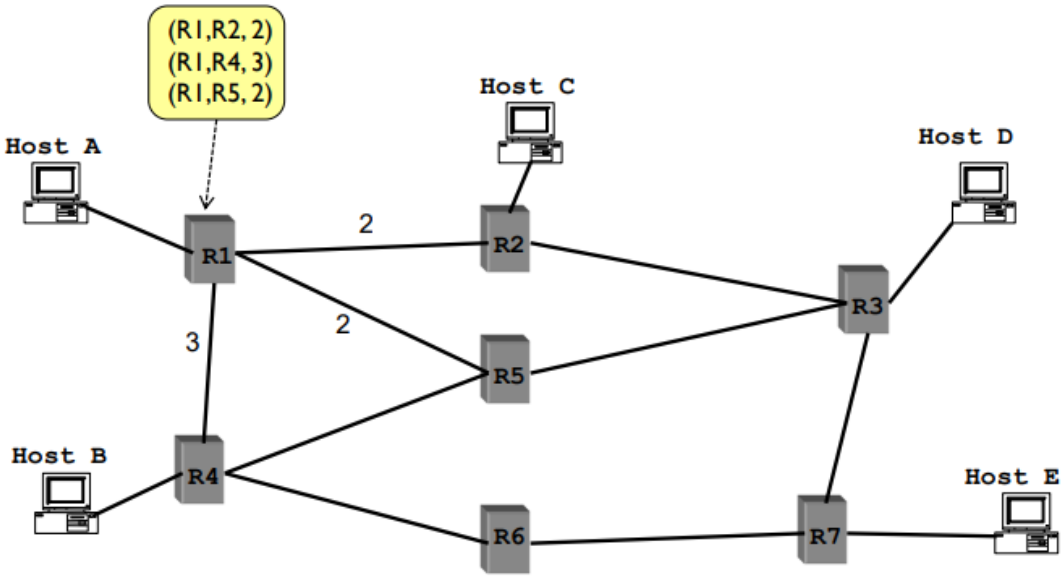
Decentralised: Iterative process of computation, exchange of info with neighbours.

Routers initially only know link costs to attached neighbours

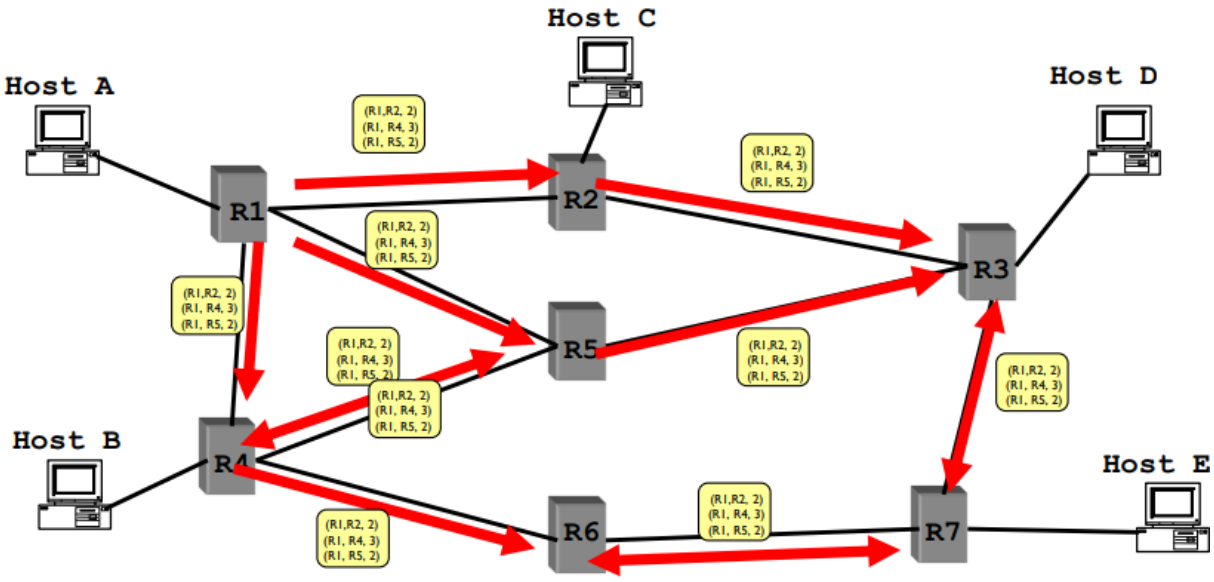
"Distance Vector" algorithms

Link State Routing

- Each node maintains its *local* "link state" (LS)
  i.e. a list of its directly attached links and their costs



- Each node floods its local link state
  On receiving a *new* LS message, a router forwards the message to all its neighbours other than the one it received the message from



Flooding LSAs

Routers transmitting Link State Advertisement (LSA) on links

- A neighbouring router forwards out on all links except incoming
- Keep a copy locally; don't forward previously-seen LSAs

Challenges:
- Packet loss
- Out of order arrival

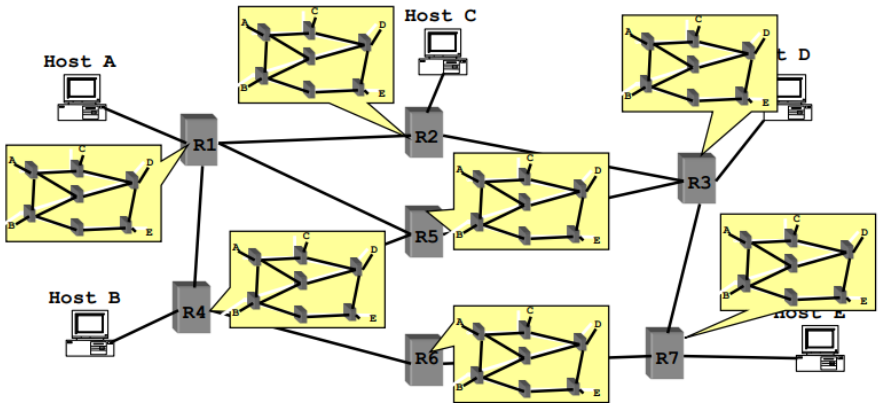
Solutions:
- Acknowledgements and retransmissions
- Sequence numbers

Time to live (TTL) for each packet

Link State Routing

Eventually each node learns the entire network topology

– Can use Dijkstra to computer shortest paths!



Dijkstra's link-state routing algorithm

Centralised: Network topology, link costs known to *all* nodes
- Accomplished via "link state broadcast"
- All nodes have same info

Computes least cost paths from one node ("source") to all other nodes
- Gives *forwarding table* for that node

Iterative: After *k* iterations, know least cost path to *k* destinations

Notation

$c_{x,y}$: direct link cost from node x to y; ∞ if not connected.

$D(v)$: current estimate of cost of least-cost-path from source to destination v

$P(v)$: predecessor node along path from source to v

N': Set of nodes whose least-cost-path *definitively* known

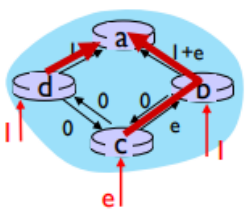
```
1  Initialization:
2    N' = {u}                    /* compute least cost path from u to all other nodes */
3    for all nodes v
4      if v adjacent to u        /* u initially knows direct-path-cost only to  direct neighbors
*/
5        then D(v) = c_u,v       /* but may not be minimum cost!
*/
6      else D(v) = ∞
7
8  Loop
9      find w not in N' such that D(w) is a minimum
10     add w to N'
11     update D(v) for all v adjacent to w and not in N' :
12         D(v) = min ( D(v), D(w) + c_w,v )
13     /* new least-path-cost to v is either old least-cost-path to v or known
14     least-cost-path to w plus direct-cost from w to v */
15 until all nodes in N'
```

Algorithm complexity	Message complexity
Each of *n* iteration: need to check all nodes, w, not in *N* $\frac{N(n+1)}{2}$ comparisons: $O(n^2)$ complexity More efficient implementations possible: $O(nlogn)$	Each router must *broadcast* its link state information to other *n* routers Efficient broadcast algorithms: $O(n)$ Each router's message crosses $O(n)$ links: overall complexity = $O(n^2)$

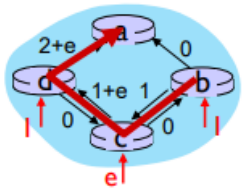
Dijkstra's Algorithm: oscillations possible

When link costs depend on traffic volume, route oscillations are possible.

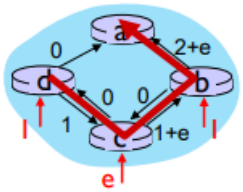
E.g. Routing to destination A, entering at D, C, E with rates 1, e (<1), 1.
Link costs are directional, and volume-dependent



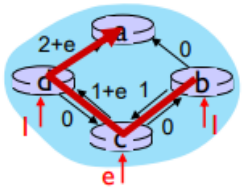
initially



given these costs, find new routing…. resulting in new costs



given these costs, find new routing…. resulting in new costs



given these costs, find new routing…. resulting in new costs

Distance Vector Algorithm

Based on *Bellman-Ford* (BF) equation (dynamic programming):

$$Let\ d_x(y): cost\ of\ least\ cost\ path\ from\ x\ to\ y$$

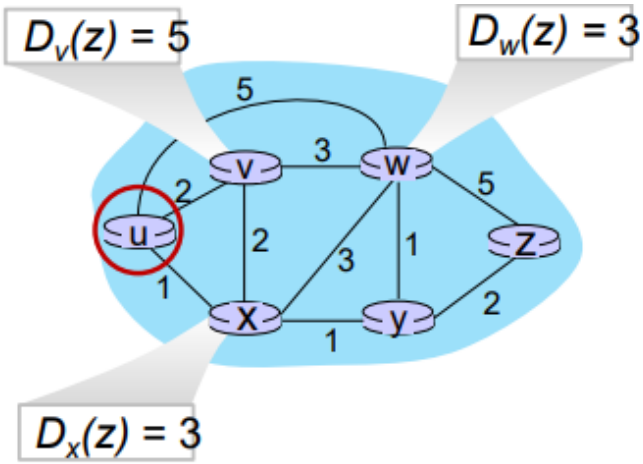
then:

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

$\boldsymbol{min_v}$ = min taken over all neighbours v of x

$\boldsymbol{c_{x,v}}$ = direct cost of link from x to v

$\boldsymbol{D_v(y)}$ = v's estimated least-cost-path cost to y



Bellman-Ford equation says:

$$D_u(z) = \min \{ c_{u,v} + D_v(z),\ c_{u,x} + D_x(z),\ c_{u,w} + D_w(z) \}$$
$$= \min \{2 + 5,\ 1 + 3,\ 5 + 3\} = 4$$

node achieving minimum (x) is next hop on estimated least-cost path to destination (z)

Key idea:

- From time-to-time, each node sends its own distance vector estimate to neighbours
- When x receives new DV estimate from any neighbour, it updates its own DV using B-F equation:

$$D_x(y) \leftarrow min_v\{c_{x,v} + D_v(Y)\}$$

- Under minor, natural conditions, the estimate $D_x(y)$ *converge to the actual least* cost $d_x(y)$

Distance Vector Algorithm:

Each node:

1. *Wait* for (change in local link cost or DV from neighbour)
2. *Recompute* DV estimates using DV received from neighbour
3. If DV to any destination has changed, *notify* neighbours.

Iterative, asynchronous: each local iteration caused by:

Local link cost change
DV update message from neighbour.

Distributed, self-stopping: each node notifies neighbours *only* when its DV changes

Neighbours then notify their neighbours – only if necessary
No notification received; no actions taken!

DV: Information diffusion

Iterative communication, computation steps diffuses information through network:

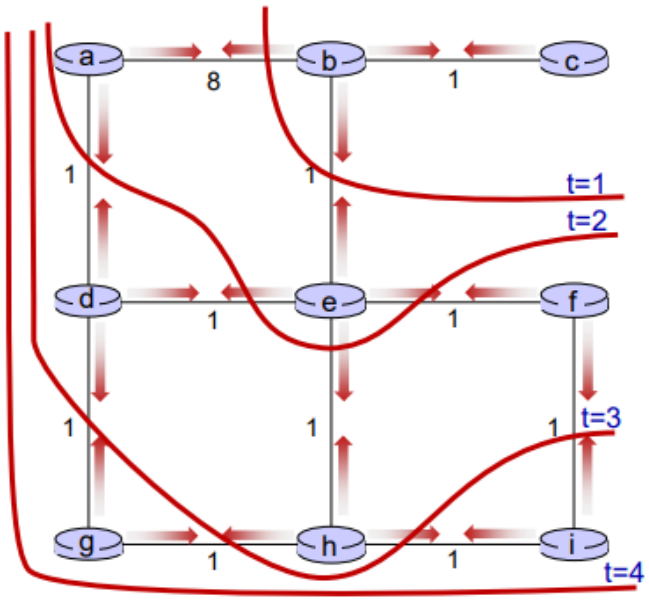
t=0 c's state at t=0 is at c only

t=1 c's state at t=0 has propagated to b, and may influence distance vector computations up to **1** hop away, i.e., at b

t=2 c's state at t=0 may now influence distance vector computations up to **2** hops away, i.e., at b and now at a, e as well

t=3 c's state at t=0 may influence distance vector computations up to **3** hops away, i.e., at b,a,e and now at c,f,h as well

t=4 c's state at t=0 may influence distance vector computations up to **4** hops away, i.e., at b,a,e, c, f, h and now at g,i as well

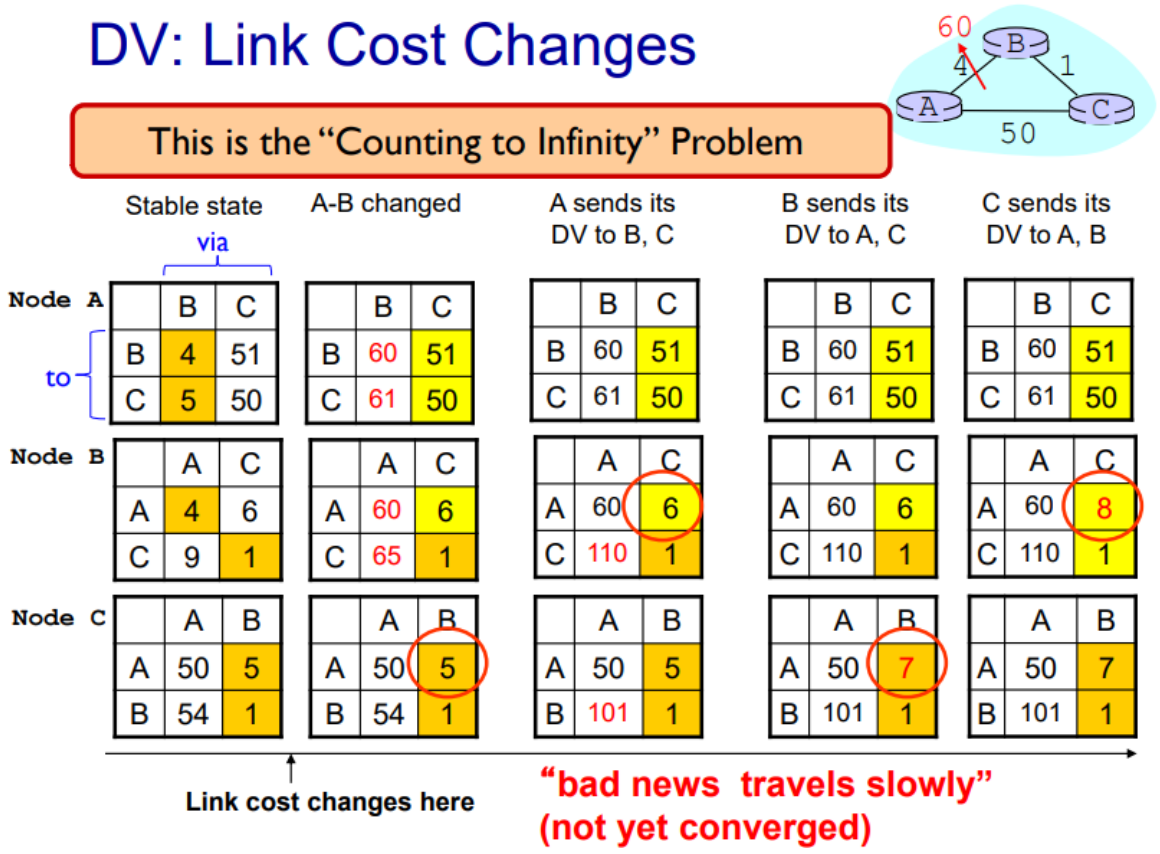


Problems with Distance Vector

- A number of problems can occur in a network using distance vector algorithm
- Most of these problems are caused by slow convergence or routers converging on incorrect information
- *Convergence* is the time during which all routers come to an agreement about the best paths through the internetwork
    - Whenever topology changes there is a period of instability in the network as the routers converge
- Reacts rapidly to good news, but leisurely to bad news.

"Poisoned Reverse" Rule

- Heuristic to avoid count-to-infinity
- If B routes via C to get to A:
    - B tells C its (B's) distance to A is infinite
      (So C won't route to A via B)



Comparison of LS and DV algorithms

Message complexity
- LS: n routers, $O(n^2)$ messages sent
- DV: exchange between neighbours; convergence time varies

Speed of convergence
- LS: $O(n^2)$ algorithm, $O(n^2)$ messages
  (May have oscillations)
- DV: Convergence time varies
  May have routing loops
  Count-to-infinity problem

Robustness: What happens if router malfunctions, or is compromised?

LS:
- Router can advertise incorrect link cost
- Each router computes only its *own* table

DV:
- DV Router can advertise incorrect path cost to everywhere: black-holing
- Each router's table used by others: error propagate through network

Real Protocols

Link State

Open Shortest Path First (OSPF)

Intermediate system to Intermediate system (IS-IS)

Distance Vector

Routing Information Protocol (RIP)

Interior Gateway Routing Protocol (IGRP-Cisco)

Border Gateway Protocol (BGP) – variant

Q: In Link State Routing (LS), each node sends information of its direct links (i.e. link state) to?

A: All nodes in the network

Q: In Distance-vector routing (VS), each node sends information of its direct links (i.e. link state) to?

A: All immediate neighbours

Distance vector routing – Convergence delay depends on the topology (nodes and links) and link weights

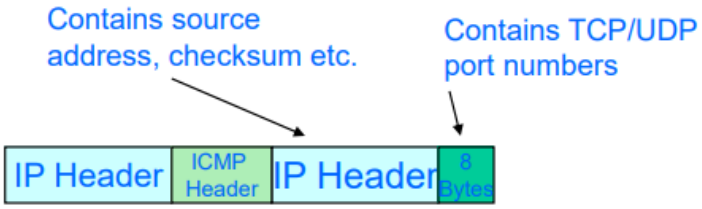
---

ICMP: Internet Control Message Protocol

Used by hosts and routers to communicate network level information
- Error reporting; unreachable host, network, port
- Echo request/reply (used by ping)

Works above IP layer
- ICMP messages carried in IP datagrams

ICMP message: type, code plus IP header and first 8 bytes of IP datagram payload causing error



Type	Code	Description
0	0	Echo reply (ping)
3	0	Destination network unreachable
3	1	Destination host unreachable
3	3	Destination port unreachable
3	4	Frag needed, DF set
8	0	Echo request (ping)
11	0	TTL expired
11	1	Frag reassembly time exceeded
12	0	Bad IP header

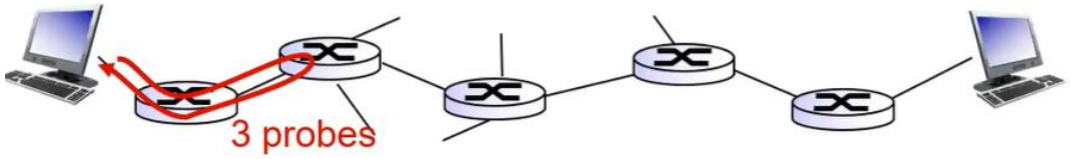
Traceroute and IMCP

Source sends series of UDP segments to dest
- 1st set has TTL = 1
- 2nd set has TTL = 2, etc
- Unlikely port number

When the *n*th set of datagrams arrives to *n*th router:
- Router discards datagrams
- Sends source ICMP messages (type 11, code 0)
- ICMP messages includes IP address of router

When ICMP messages arrives, source records RTTS

Stopping criteria:
- UDP segment eventually arrives at destination host
- Destination returns ICMP 'port unreachable' message (type 3, code 3)
- Source stops

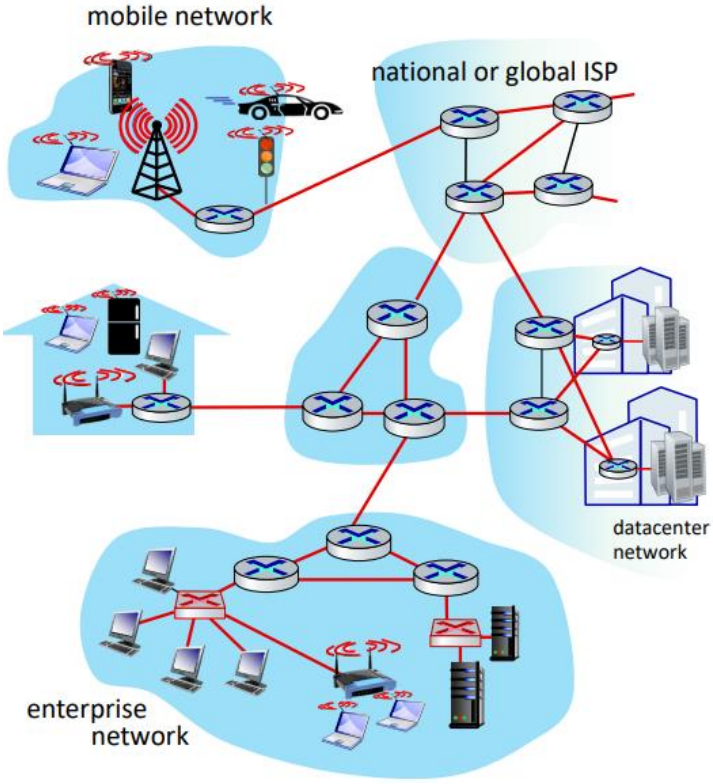


Datalink Layer

Link Layer and LANs:

- Error detection, correction
- Multiple access protocols
- LANS
  - Addressing, ARP
  - Ethernet
  - Switches

Web requests…



Terminology:

Hosts and routers: nodes

Communication channels that connect adjacent nodes along communication path: links

(Wired, wireless, LANS)

Layer-2 packet: frame, encapsulates datagram

***Link layer** has responsibility of transferring datagram from one node to **physically adjacent** node over a link*

Link layer: context

Datagram transferred by different link protocols over different links:
- E.g. WiFi on first link, Ethernet on next link

Each link protocol provides different services
- E.g., may or may not provide reliable data transfer over link

Transportation analogy:

A trip from Princeton to Luasanne
- Limo: Princeton to JFK
- Plane: JFK to Geneva
- Train: Geneva to Lausanne

Tourist = datagram

Transport segment = communication link

Transportation mode = link-layer protocol

Travel agent = routing algorithm

Link layer: services

Framing, link access:

Encapsulate datagram into frame, adding header, trailer

Channel access if shared medium

"MAC" addresses in frame headers identify source, destination (different from IP address!)

Reliable delivery between adjacent nodes

We already know how to do this!

Seldom used on low bit-error links

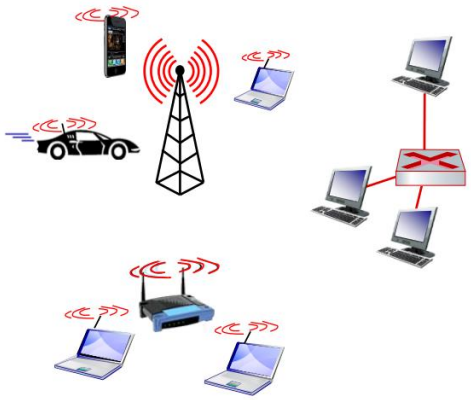
Wireless links: high error rates

Q: Why both link-level and end-end reliability?

Link Layer: Services (cont.)

Flow control:
- Pacing between adjacent sending and receiving nodes

Error detection:
- Errors caused by signal attenuation, noise.
- Receiver detects errors, signals retransmission, or drops frame

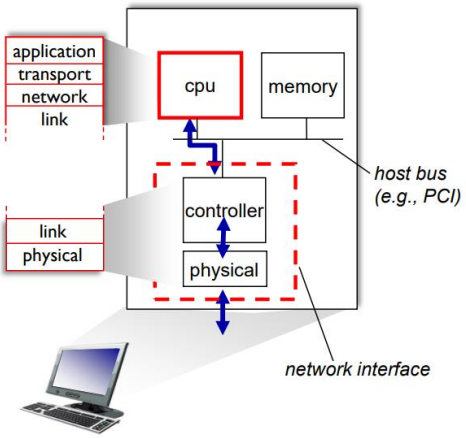


Error correction:
- Receiver identifies *and corrects* bit error(s) without retransmission

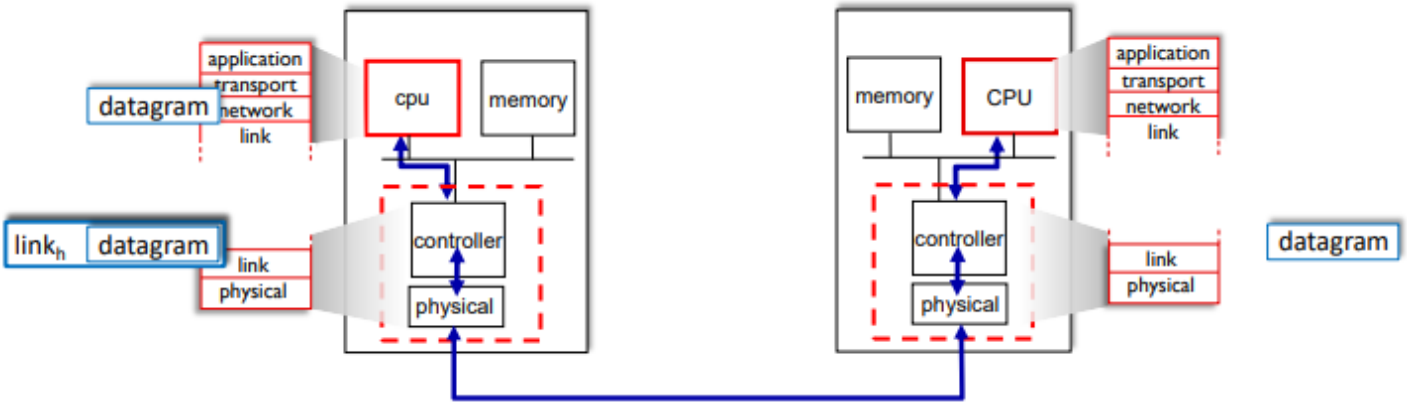
Half-duplex and full-duplex:
- With half duplex, nodes at both ends of link can transmit, but not at same time

Where is the link layer implemented?

- In each-and-every host
- Link layer implemented in network interface card (NIC) or on a chip
  - Ethernet, WiFi card or chip
  - Implements link, physical layer
- Attaches into host's system buses
- Combination of hardware, software, firmware



Interfaces communicating



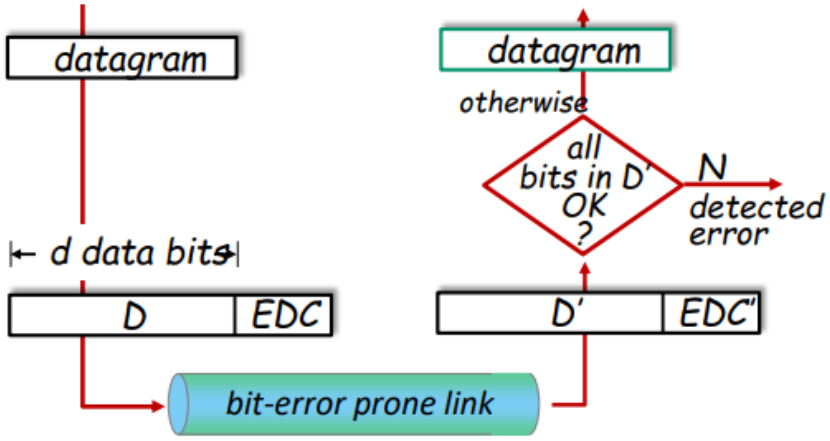
Sending side:
- Encapsulates datagram in frame
- Adds error checking bits, reliable data transfer, flow control, etc

Receiving side:
- Looks for errors, reliable data transfer, flow control, etc
- Extracts datagram, passes to upper layer at receiving side

Error Detection

EDC: Error detection and correction bits (e.g., redundancy)

D: Data protected by error checking, may include header fields

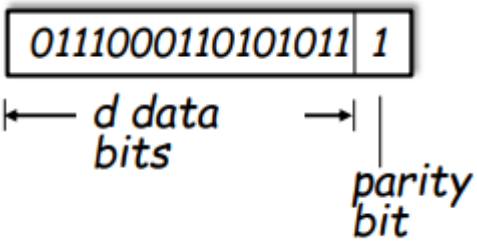


Error detection not 100% reliable!

- Protocol may miss some errors (rare)
- Large EDC field yields better detection and correction

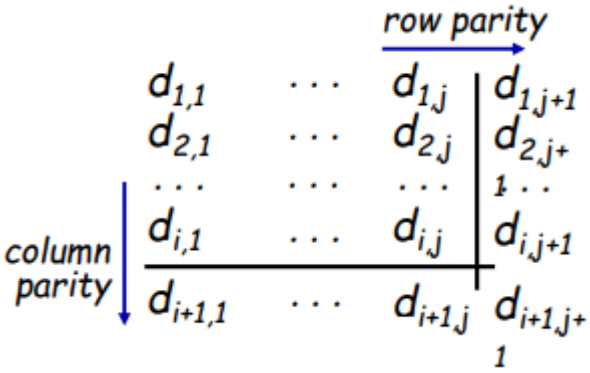
Error Detection

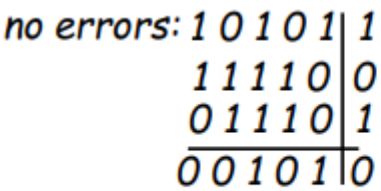
Single bit parity:
- Detect single bit errors

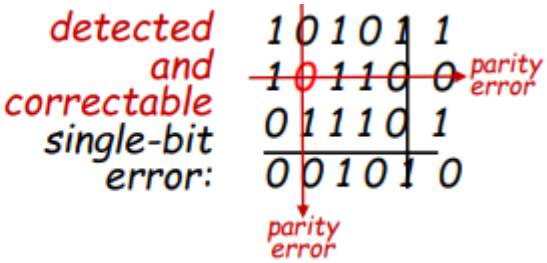


Even parity: set parity bit so there is an even number of 1's

Two-dimensional bit parity:
- Detect *and correct* single bit errors







Internet checksum (review)

Goal: detect errors (i.e. flipped bits) in transmitted segment

Sender:
Treat contents of UDP segment (incl. UDP header fields and IP addresses) as sequence of 16-bit integers

Checksum: addition (one's complement sum) of segment content

Checksum value put into UDP checksum field

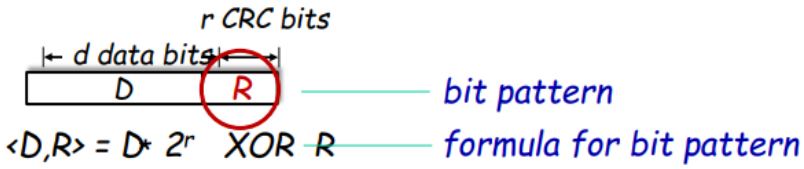
Receiver:
Compute checksum of received segment

Check if computed checksum field value:
- Not equal – error detected
- Equal – no error detected. *But maybe errors nonetheless?*

In practice:

- Bit errors occur in bursts.
- We're willing to trade computational complexity for space efficiency.
  - Make the detection routine more complex, to detect error bursts, without tons of extra data
- Insight: We need harwareto interface

Cyclic Redundancy Check (CRC)



A more powerful error-detection coding

D: data bits *(given, think of these as a binary number)*

G: bit pattern (generator), of $r + 1$ bits *(given)*

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

- Receiver knows G divides $< D, R >$ by G. If non-zero remainder: error detected!
- Can detect all burs errors less than $r + 1$ bits
- Widely used in practice

Note on Modulo-2 Arithmetic

All calculations are modulo-2 arithmetic

No carries or borrows in subtraction

Addition and subtraction are identical and both are equivalent to XOR

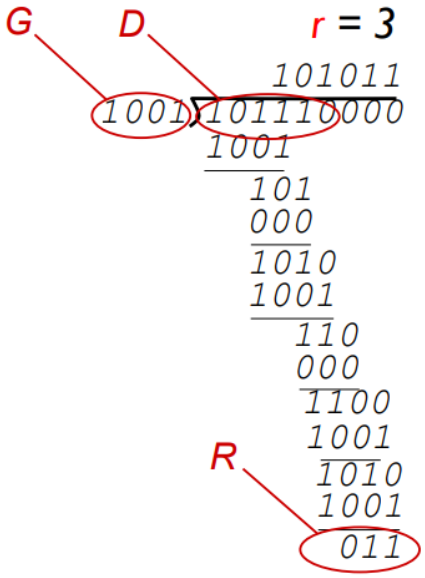
$$1011\ XOR\ 0101 = 1110$$

$$1011 - 0101 = 1110$$

$$1011 + 0101 = 1110$$

Multiplication by 2^k is essentially a left shift by *k* bits

At the sender



Example 1:

Want: $D \cdot 2^r \text{ XOR } R = nG$

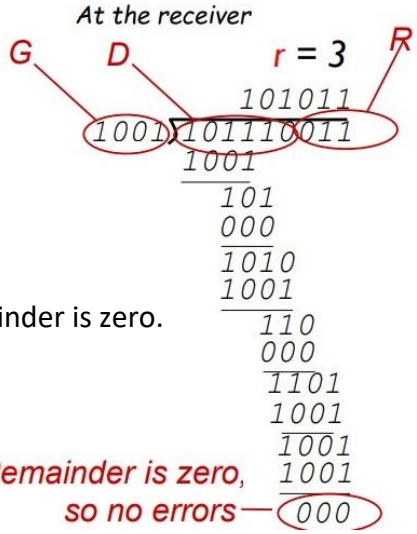
Equiv: $D \cdot 2^r = nG \text{ XOR } R$

Equiv: If we divide $D \cdot 2^r$ by G, want remainder to satisfy:

$$R = remainder\left[\frac{D \cdot 2^r}{G}\right]$$

Sender sends $< D, R >$ into the channel

At the receiver



Example 2:

Receiver divides the received frame and divides by G and checks if the remainder is zero.

In this example, there are no errors, so the receiver receives $< D, R >$ (from previous slide)

Multiple Access Protocols

- Single shared broadcast channel
- Two or more simultaneous transmissions by nodes: interference
    - Collision if node receives two or more signals at the same time

Multiple access protocol

- Distributed algorithm that determines how nodes share channel, i.e., determine when node can transmit
- Communication about channel sharing must use channel itself!
    - No out of band channel for coordination

An ideal multiple access protocol

Given: Multiple access channel (MAC) of rate *R* bps

Desired properties:

1. When one node wants to transmit, it can send at rate R.
2. When M nodes want to transmit, each can send at average rate R/M
3. Fully decentralised:
    - No special node to coordinate transmissions
    - No synchronisation of clocks, slots
4. Simple

MAC Protocols: Taxonomy

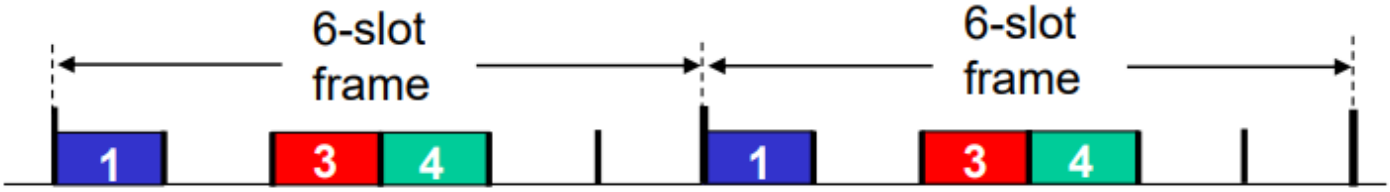
Three broad classes:

- Channel partitioning
    - Divide channel into smaller “pieces” (time slots, frequency, code)
    - Allocate piece to node for exclusive use
- Random access
    - Channel not divided, allow collisions
    - “Recover” from collisions
- “Taking turns”
    - Nodes take turns, but nodes with more to send can take longer turns

Channel partitioning MAC protocols: TDMA

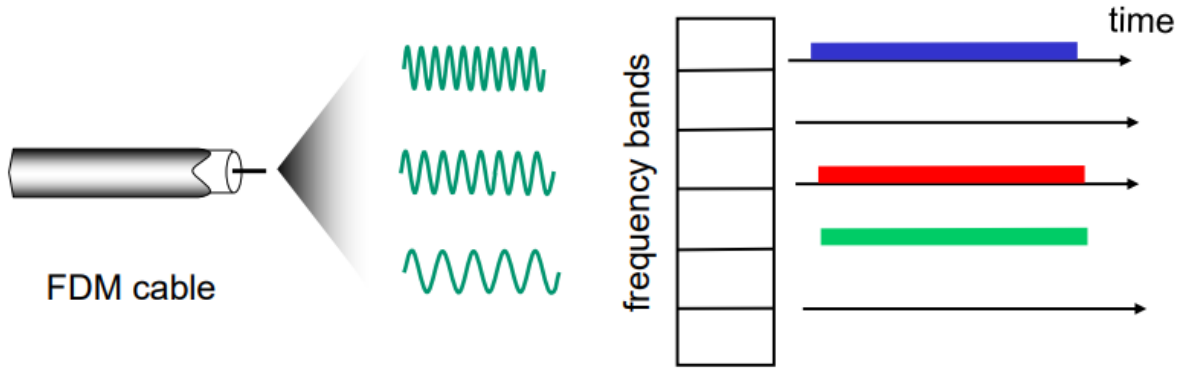
TDMA: time division multiple access

- Access to channel in “rounds”
- Each node gets fixed length slot (length = packet transmission time) in each round
- Unused slots go idle
- Example: 6-node LAN, 1,3,4 have packets to send, slots 2,5,6 idle



Channel partitioning MAC protocols: FDMA

FDMA: Frequency Division Multiple Access
- Channel spectrum divided into frequency bands
- Each node assigned fixed frequency band
- Unused transmission time in frequency bands go idle
- Example: 6-node LAN 1,3,4 have packet ot send, frequency bands 2,5,6 idle



Random access protocols

When a node has packet to send
- Transmit at full channel data rate R
- Not a priority coordination among nodes

Two or more transmitting nodes: “collision”

Random access MAC protocol specifies:
- How to detect collisions
- How to recover from colliusions (e.g., via delayed retransmissions)

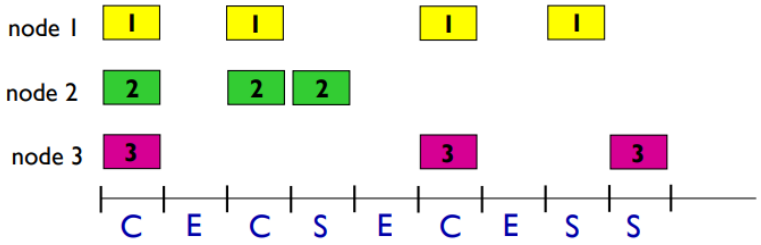
Examples of random-access MAC protocols:
- ALOHA, slotted ALOHA
- CSMA, CSMA/CD, CSMA/CA

Slotted ALOHA

Assumptions:
- All frames same size
- Time divided into equal size slots (Time to transmit 1 frame)
- Nodes start to transmit only slot beginning
- Nodes are synchronised
- If 2 or more nodes transmit in slot, all nodes detect collision

Operation:
- When node obtains fresh frame, transmits in next slot
- If no collision: node can send new frame in next slot
- If collision: node retransmits frame in each subsequent slot with probability *p* until success

Slotted ALOHA cont.



C: collision
S: success
E: empty

Pros:
- Single active node can continuously transmit at full rate of channel
- Highly decentralised: only slots in nodes need to be in sync
- Simple

Cons:
- Collisions, wasting slots
- Idle slots
- Nodes may be able to detect collisions in less than time to transmit packet
- Clock synchronisation

Slotted ALOHA: efficiency

Efficiency: long-run fraction of successful slots (many nodes, all with many frames to send)
- Suppose: N nodes with many frames to send, each transmits in slot with probability *p*

Prob that given node has success in a slot = $p(1-p)^{n-1}$

Prob that *any* node has a success = $Np(1-p)^{n-1}$

- Max efficiency: find P* that maximises $Np(1-p)^{n-1}$
- For many nodes, take limit of $Np(1-p)^{n-1}$ as N goes to infinity, gives

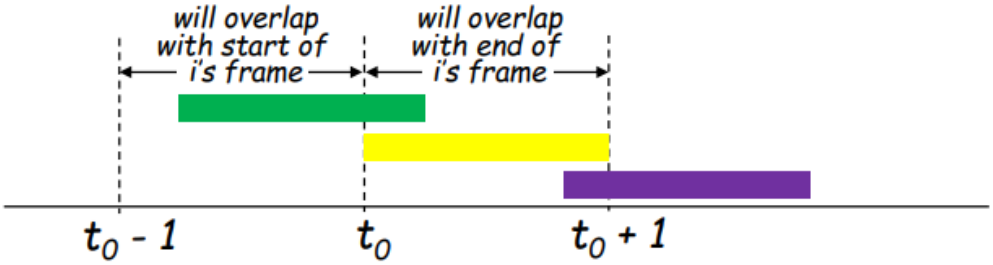
$$\max efficiency = \frac{1}{e} = 0.37$$

- At best: channel used for useful transmissions 37% of the time!

Pure ALOHA

Unslotted Aloha: simpler, no synchronisation
- When frame first arrives: transmit immediately

Collision probability increases with no synchronisation:
- Frame sent at t_0 collides with other frames sent in $[t_0 - 1, t_0 + 1]$
- Pure Aloha efficiency: 18%!



CSMA (Carrier Sense Multiple Access)

Simple CSMA: listen before transmit:
- If channel sensed idle: transmit entire frame
- If channel sensed busy: defer transmission

Human analogy: don't interrupt others!

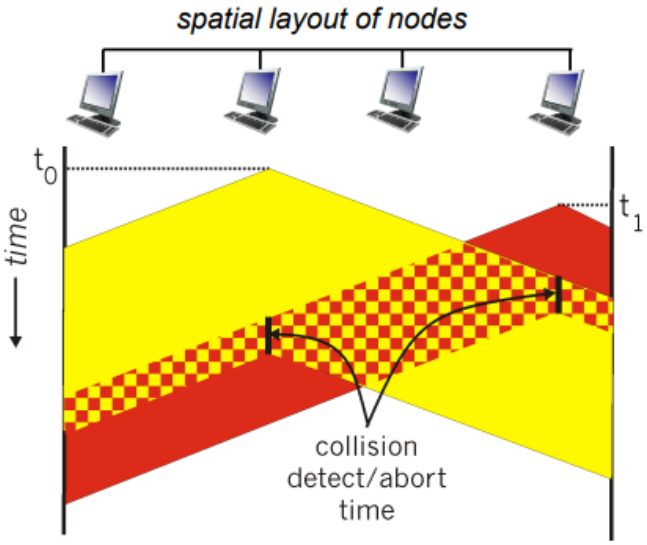
CSMA/CD: CSMA with collision detection
- Collisions detected within short time
- Colliding transmissions aborted, reducing channel wastage
- Collision detected easy in wired, difficult with wireless

Human analogy: the polite conversationalist

CSMA: Collisions

Collisions can still occur with carrier sensing:
- Propagation delay means two nodes may not hear each other's just-started transmission

Collision: entire packet transmission time wasted
- Distance & propagation delay play role is in determining collision probability



CSMA/CD:

CSMA/CD reduces the amount of time wasted in collisions
- Transmission aborted on collision detection

CSMA: Collisions

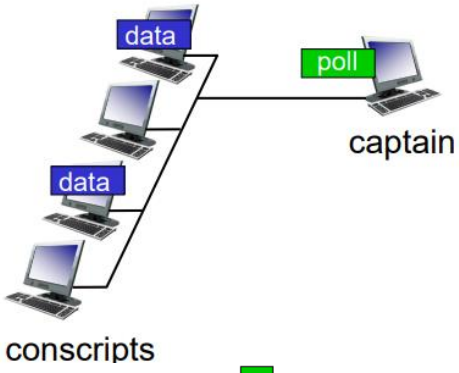
1. NIC receives datagram from network layer, creates frame
2. If NIC senses channel:
   If NIC senses channel: if idle: start frame transmission.
   If busy: wait until channel idle, then transmit
3. If NIC transmits entire frame without collision, NIC is done with frame!
4. If NIC detects another transmission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

NIC = Network Interface Card

CSMA/CD Efficiency

- T_{prop} = max prop delay between 2 nodes in LAN
- t_{trans} = time to transmit max-size frame
- Efficiency goes to 1
  - As t_{prop} goes to 0
  - As t_{trans} goes to infinity
- Better performance than ALOHA, simple, cheap and decentralised too!

“Taking turns” MAC protocols

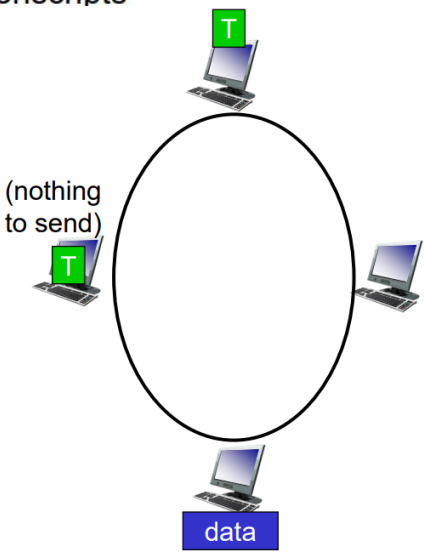


Channel partitioning MAC protocols:
- Share channel efficiently and fairly at high load
- Inefficient at low load: delay in channel access, 1/N bandwidth allocated even if only 1 active node!

Random access MAC protocols
- Efficient at low load: single node can fully utilise channel
- High load: collision overhead

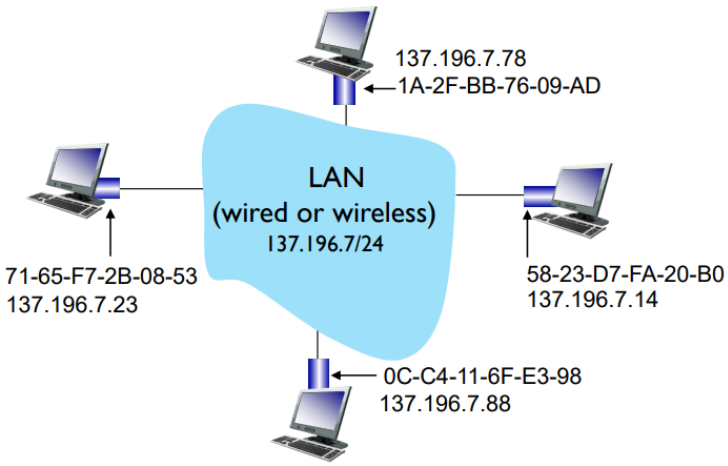
“Taking turns” protocols
- Look for best of both worlds

Polling:
- Captain node “invites” other nodes to transmit in turn
- Typically used with “dumb” devices
- Concerns:
    - Polling overhead
    - Latency
    - Single point of failure (captain)



Token passing:
- Control *token* passed from one node to next sequentially
- Token message
- Concerns:
    - Token overhead latency
    - Single point of failure (token)

MAC Addresses



32-bit IP address:
- Network-layer address for interface
- Used for layer 3 (network layer) forwarding
- E.g.: 128.119.40.136

MAC (or LAN or physical or Ethernet) address:
- Function: used “locally” to get frame from one interface to another physically connected interface (same subnet, in IP-addressing sense)
- 48-bit MAC address (for most LANs) burned in NIC ROM, also sometimes software settable
- E.g.: 1A-2F-BB-076-09-AD

Each interface on LAN:
- Has a unique 48-bit MAC address
- Has a locally unique 32-bit IP address (as we’ve seen)

MAC addresses (cont.)

- MAC address allocation administered by IEEE
- Manufacturer buys portion of MAC address space (to assure uniqueness)
- Analogy:
  - MAC address: like TFN
  - IP address: like postal address
- MAC flat address: portability
  - Can move interface from one LAN to another
  - Recall IP address not portable: depends on IP subnet to which node attached

MAC addresses vs IP Address

MAC Addresses (used in link-layer
- Hard-coded in read only memory when adapter is built
- Flat name space of 48 bits (e.g., 00-0E-9B-6E-49-76)
- Portable, and can stay the same as host moves
- Used to get packet between interfaces on same network

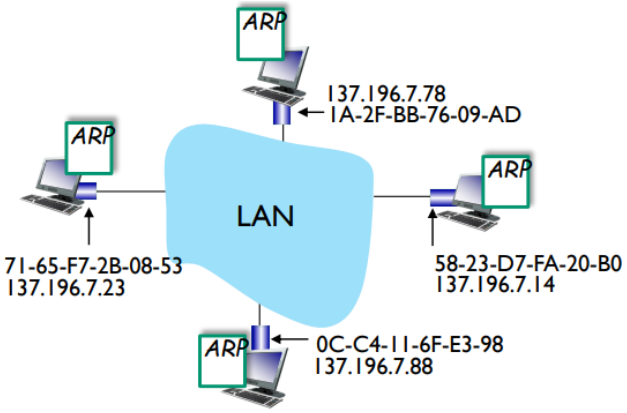
IP Addresses
- Learned dynamically
- Hierarchical name space of 32 bits (e.g., 12.178.66.9)
- Not portable, and depends on where the host is attached
- Used to get a packet to destination IP subnet

Taking Stock: Naming

Layer	Examples	Structure	Configuration	Resolution Service
App. Layer	www.cse.unsw.edu.au	organizational hierarchy	~ manual	DNS
Network Layer	129.94.242.51	topological hierarchy	DHCP	ARP
Link layer	45-CC-4E-12-F0-97	vendor (flat)	hard-coded	

ARP: Address Resolution Protocol

Q: How to determine interface's MAC address, knowing its IP address?

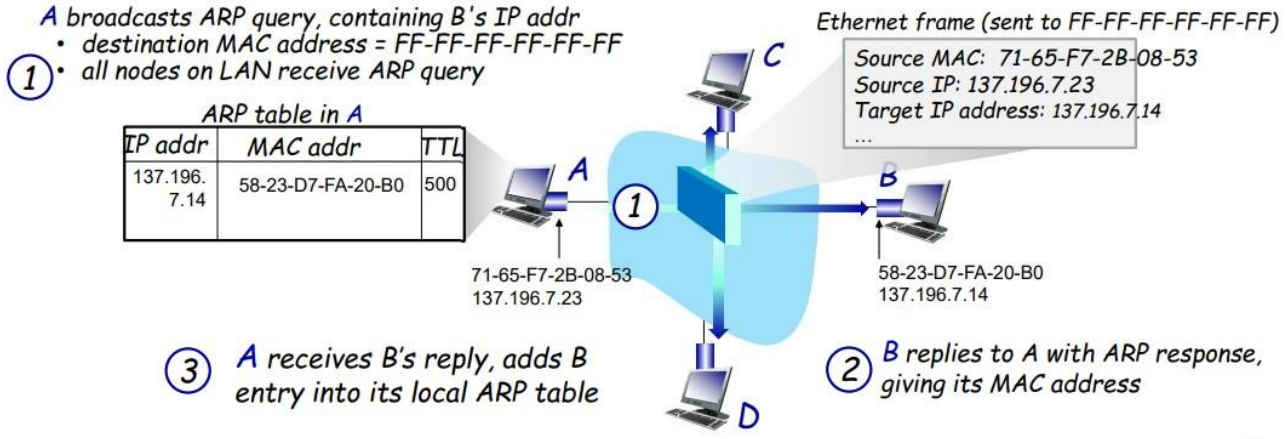


ARP table: each IP node (host, router) on LAN has table
- IP/MAC address mappings for some LAN nodes:
  < IP address; MAC address; TTL >
- TTL (Time To Live): time after which address mapping will be forgotten (typically 20 min)

Arp protocol in action

Example: *A* wants to send datagram to *B*

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



Routing to another subnet: addressing

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

Focus on addressing – at IP (datagram) and MAC layer (frame) levels

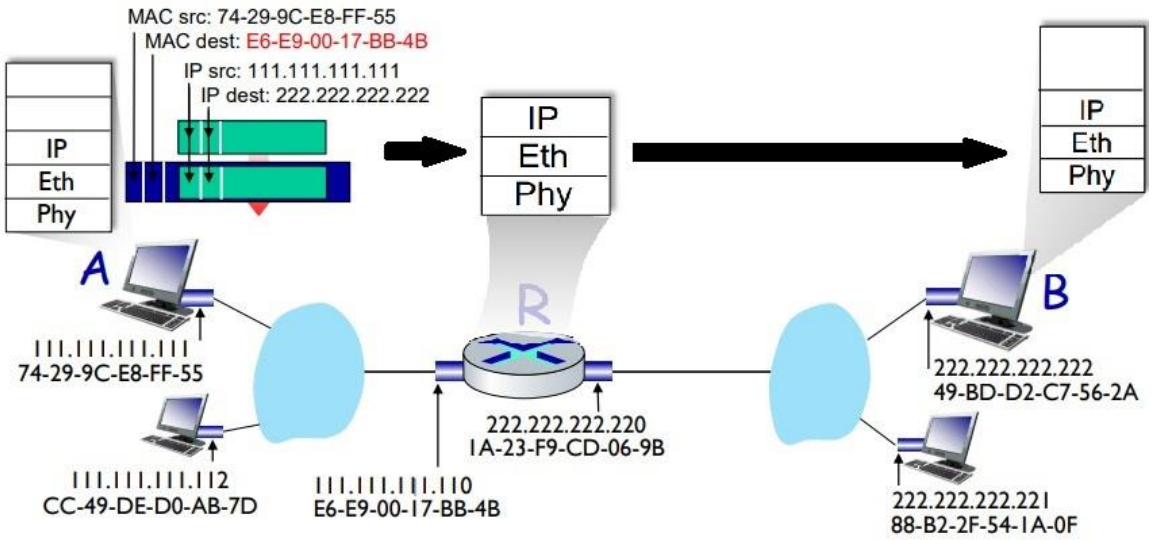
Assume that:

A knows B's IP address (how does A know that the next-hop is Router R?)

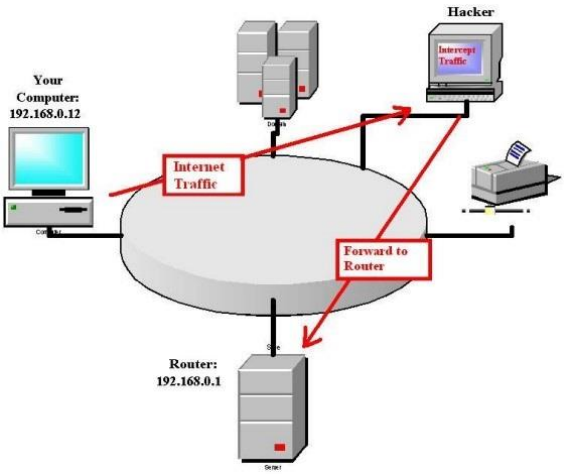
A knows IP address of first hop router, R (how?)

A knows R's MAC address (how?)

1. A creates IP datagram with IP source A, destination B
2. A creates link-layer frame containing A-to-B IP datagram
   R's MAC address is frame's destination
3. Frame sent from A to R
4. Frame received at R, datagram removed ,passed up to IP
5. R determines outgoing interface, passes datagram with IP source A, destination B to link layer
6. R creates link-layer frame containing A-to-B IP datagram. Frame destination address: B's MAC address
7. R determines outgoing interface, passes datagram with IP source A, destination B to link layer
8. R creates link-layer frame containing A-to-B IP datagram. Frame destination address: B's MAC address
9. Transmits link-layer frame
10. B receives frame, extracts IP datagram destination
11. B passes datagram up protocol stack to IP



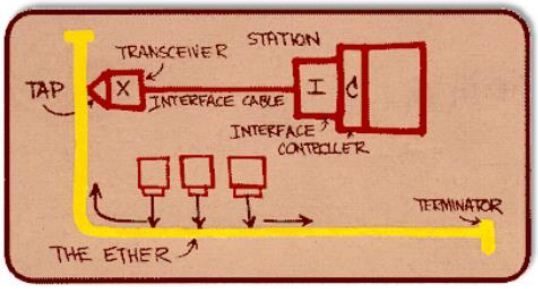
Security Issues: ARP Cache Poisoning



- Denial of Service – Hacker replies back to an ARP query for a router NIC with a fake MAC address
- Man-in-the-middle attack – Hacker can insert his/her machine along the path between victim machine and gateway router
- Such attacks are generally hard to launch as hacker needs physical access to network

Solutions –

- Use Switched Ethernet with port security enabled (i.e., one host MAC address per switch port)
- Adopt static ARP configuration for small size networks
- Use ARP monitoring tools such as ARPWatch

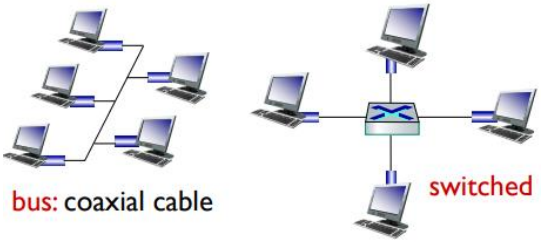


Ethernet

"Dominant" wired LAN technology

- First widely used LAN technology
- Simpler, cheap
- Kept up with speed race: 100Mbps – 400Gbps
- Single chip, multiple speeds (e.g., Broadcom BCM5761)

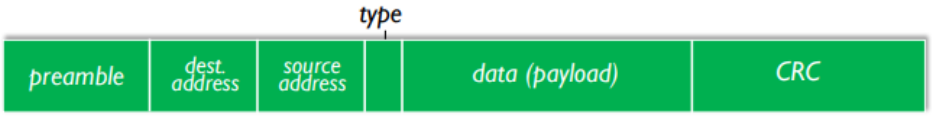
Physical Topology



- Bus: popular through mid 90s
  - All nodes in same collision domain (can collide with each other)
- Switched: prevails today
  - Active link-layer 2 switch in center
  - Each "spoke" runs a (separate) Ethernet protocol (nodes do not collide with each other)

Frame Structure

Sending interface encapsulates IP datagram (or other network layer protocol packet) in Ethernet frame



Preamble: Used to synchronise receiver, sender clock rates

- 7 bytes of `10101010` followed by one byte of `10101011`

Addresses: 6 source, destination MAC addresses

- If adapter receives frame with matching destination address, or with broadcast address (e.g., ARP packet), it passes data in frame to network layer protocol. Otherwise, adapter discards frame

Type: indicates higher layer protocol

- Mostly IP but others possible, e.g., Novell IPX, AppleTalk
- Used to demultiplex up at receiver

CRC: cyclic redundancy check at receiver

- Error detected: frame is dropped

Ethernet: Unreliable, Connectionless

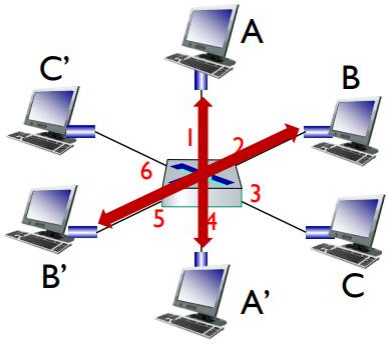
- Connectionless: no handshaking between sending and receiving NICs
- Unreliable: receiving NIC doesn't send ACKs or NAKs to sending NIC
  - Data in dropped frames recovered only if sender uses higher layer rdt (e.g., TCP), otherwise data lost
- Ethernet's MAC protocol: unslotted CSMA/CD with binary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

Ethernet Switch

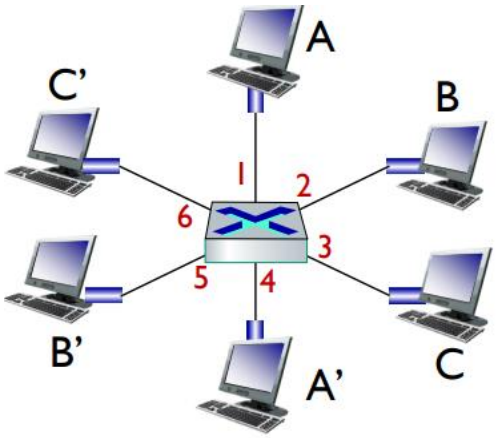
- Switch is a link-layer device: it takes an *active* role
  - Store, forward Ethernet frames
  - Examine incoming frame's MAC address, selectively forward frame to one-or-more outgoing links when frame is to be forwarded on segment, uses CSMA/CD to access segment
- Transparent: hosts unaware of presence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

Ethernet Switch – multiple simultaneous transmissions

- Hosts have dedicated, direct connection to switch
- Switches buffer packets
- Ethernet protocol used on each incoming link, so:
  - No collisions; full duplex
  - Each link is its own collision domain
- Switching: A-to-A' and B-to-B' can transmit simultaneously, without collisions
  - But A-to-A' and C-to-A' can not happen simultaneously



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



Switch forwarding table

Q: How does switch know:
A' reachable via interface 4
B' reachable via interface 5?

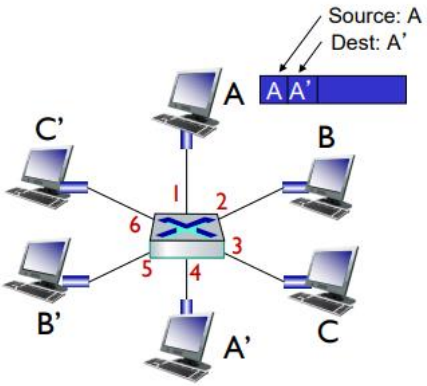
A: Each switch has a switch table, each entry:
- (MAC address of host, interface to reach host, time stamp)
- Looks like a routing table!

Q: How are entries created, maintained in switch table?
- Something like a routing protocol?

Switch: Self Learning

Switch learns which hosts can be reached through which interfaces.
When frame received, switch "learns" location of sender: incoming LAN segment:

1. Records sender/location pair in switch table
   (Incoming link, MAC address of sending host)
2. Index switch table using MAC destination address
3. `If` entry found for dest:
   `If` dest on segment from which frame arrived:
   Drop frame,
   `else` forward on interface indicated by entry
   `Else` flood *(forward on all interfaces except arriving interface)*

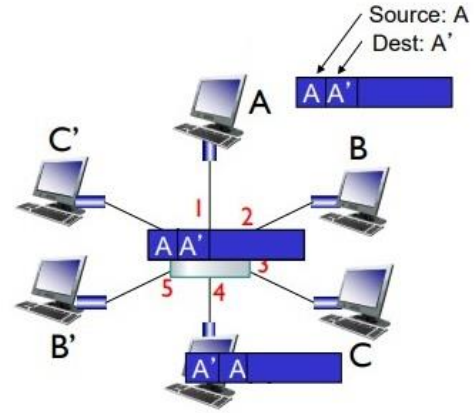


MAC addr	interface	TTL
A	1	60

Switch table (initially empty)

Example:

- Frame destination A', location unknown:
  flood
- Destination A location known:
  selectively send on just one link.

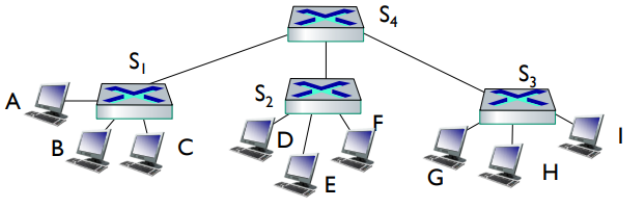


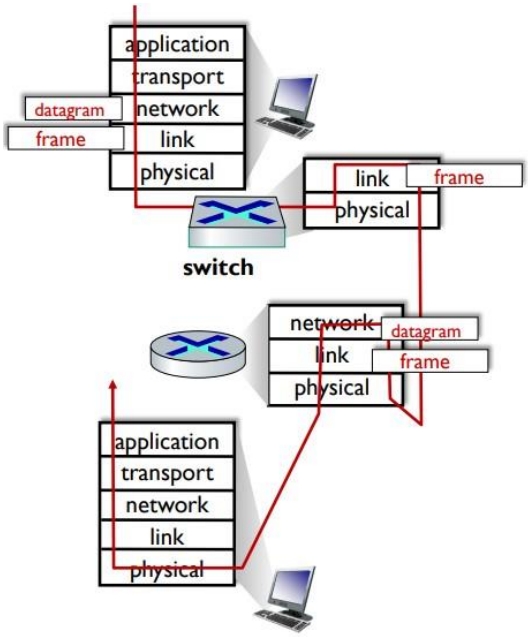
MAC addr	interface	TTL
A	1	60
A'	4	60

switch table (initially empty)

Interconnecting switches

Self learning switches can be connected together and work exactly the same as single-switch cases.





Switches vs. routers

Both are store-and-forward.

Routers: network-layer devices
(examine network-layer headers)

Switches: link-layer devices
(examine link-layer headers)

Both have forwarding tables:

Routers: compute tables using routing algorithms, IP addresses
Switches: learn forwarding table using flooding, learning, MAC addresses

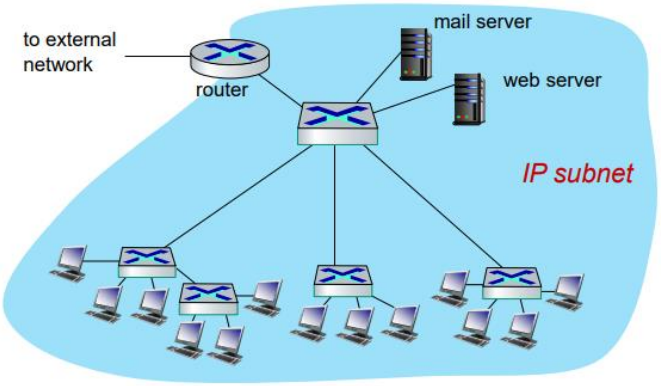


Figure 1 - Small institutional network

Security Issues

In a switched LAN once the switch table entries are established frames are not broadcast

- Sniffing frames is harder than pure broadcast LANs
- Note: attacker can still sniff broadcast frames and frames for which there are no entries (as they are broadcast)

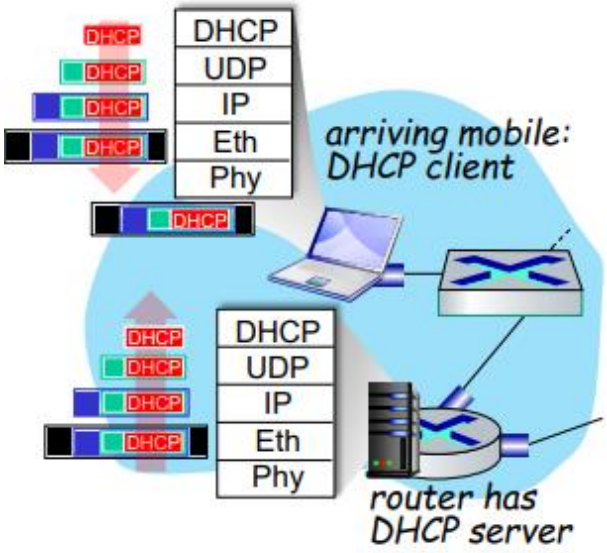
Switch poisoning: Attacker fills up switch table with bogus entries by sending large # of frames with bogus source MAC addresses

Since switch table is full, genuine packets frequently need to be broadcast as previous entries have been wiped out

Synthesis: A Day in the Life of a Web Request

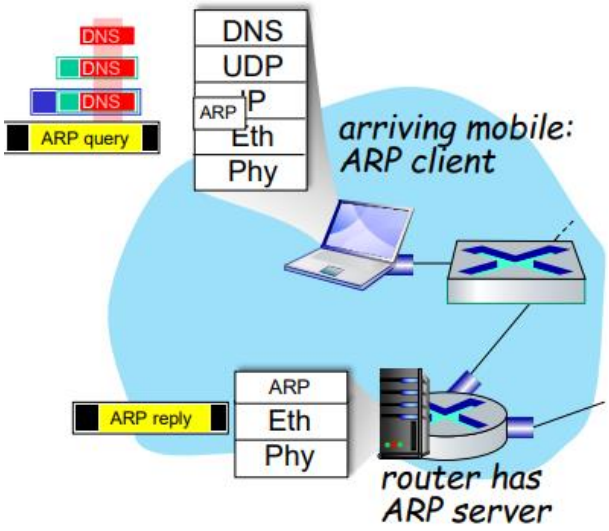
Web requests synthesise the whole journey down the protocol stack:
- Application, transport, network, link

Scenario: Connecting to internet



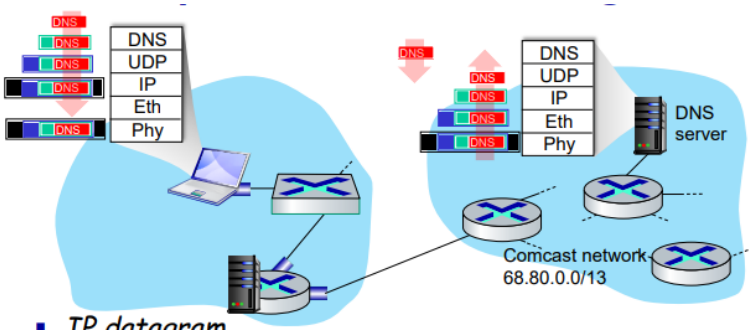
1. Connecting laptop needs to get its own IP address, addr of first-hop router, addr of DNS server: use DHCP
2. DHCP request encapsulated in UDP, encapsulated in IP, encapsulated in 802.3 Ethernet
3. Ethernet frame broadcast (dest: FFFFFFFFFFFF) on LAN, received at router running DHCP server
4. Ethernet demuxed to IP demuxed UDP demuxed to DHCP
5. DHCP server formulates DHCP ACK containing client's IP address, IP addr of first-hop router for client, name & IP addr of DNS server
6. Encapsulation at DHCP server, frame forwarded (switch learning) through LAN, demultiplexing at client
7. DHCP client receives DHCP ACK reply
8. Client now has IP addr, knows name & addr of DNS server, IP addr of its first-hop router

Scenario: ARP (before DNS, before HTTP)



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

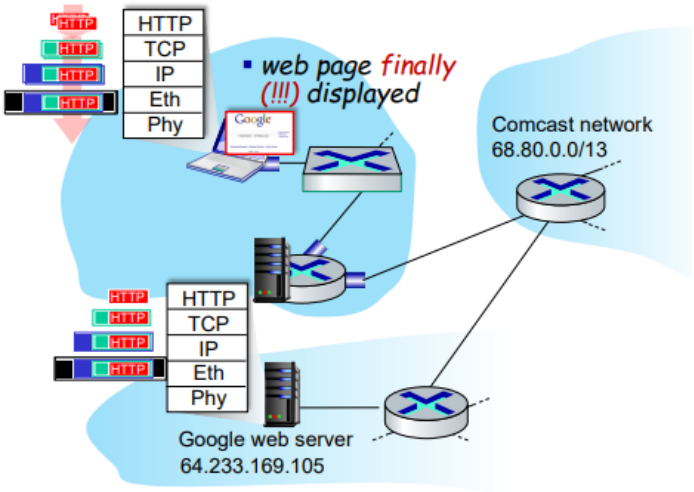
Scenario: Using DNS



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

Scenario: TCP Connection Carrying HTTP

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



Scenario: HTTP Request/Reply

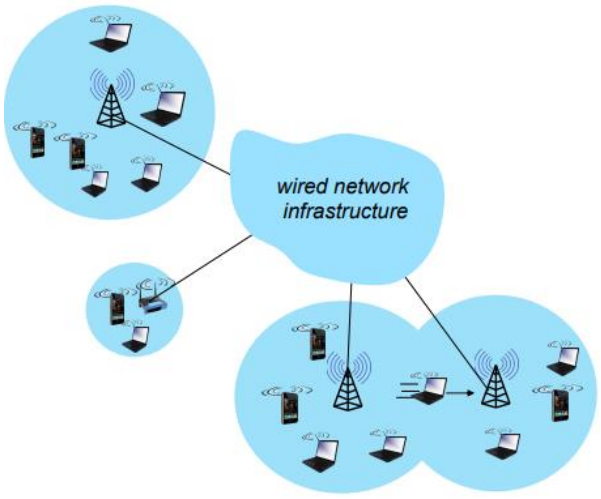
1. HTTP request sent into TCP socket
2. IP datagram containing HTTP request routed to google.com
3. Web server responds with HTTP reply (/w web page)
4. IP datagram containing HTTP reply routed back to client

Link Layer: Summary

Principles behind data link layer services:
- Error detection, correction
- Sharing a broadcast channel: multiple access
- Link layer addressing

Instantiation, im plementation of various link layer technologies
- Ethernet
- Switched LANs

Wireless and Mobile Networks



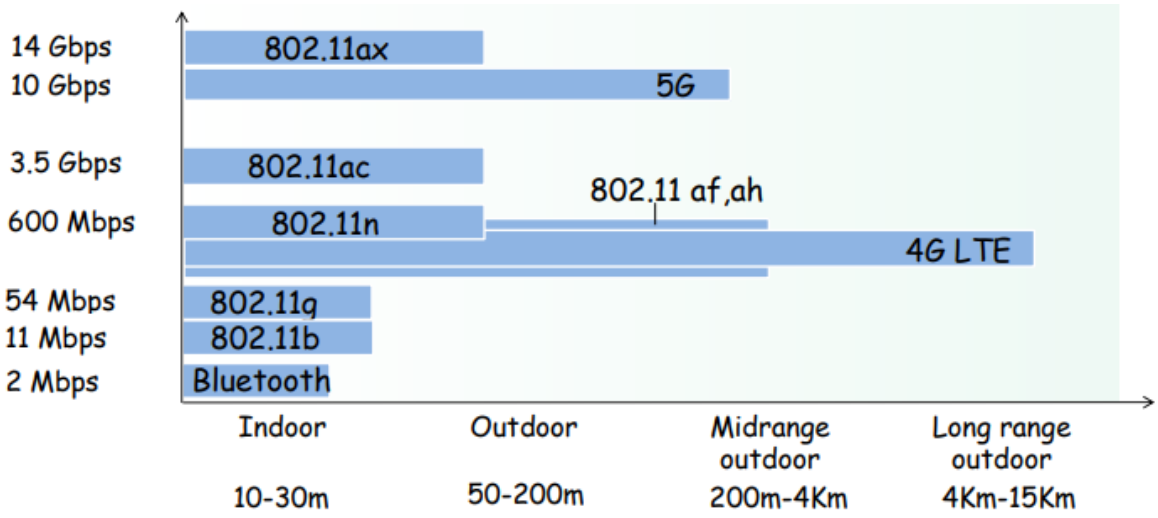
infrastructure mode
- *base station connects mobiles into wired network*
- *handoff: mobile changes base station providing connection into wired network*

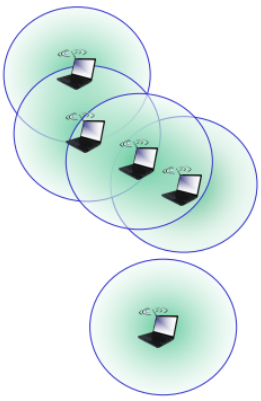
wireless hosts
- *laptop, smartphone, IoT*
- *run applications*
- *may be stationary (non-mobile) or mobile*
  - *wireless does not always mean mobility!*

base station
- *typically connected to wired network*
- *relay - responsible for sending packets between wired network and wireless host(s) in its "area"*
  - *e.g., cell towers, 802.11 access points*

wireless link
- *typically used to connect mobile(s) to base station, also used as backbone link*
- *multiple access protocol coordinates link access*
- *various transmission rates and distances, frequency bands*

Characteristics of selected wireless links





ad hoc mode

- *no base stations*
- *nodes can only transmit to other nodes within link coverage*
- *nodes organize themselves into a network: route among themselves*

Wireless Network Taxonomy

	Single Hop	Multiple Hops
Infrastructure (e.g., APs)	Host connects to base station (WiFi, cellular) which connects to larger Internet	Host may have to relay through several wireless nodes to connect to larger Internet: mesh net
No infrastructure	No base station, no connection to larger Internet (Bluetooth, ad hoc nets)	No base station, no connection to larger Internet. May have to relay to reach a given wireless node; MANET, VANET

Wireless link characteristics (I)

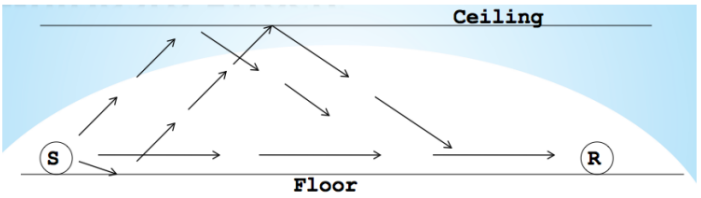
Important differences from wired link:

- Decreased signal strength: radio signal attenuates as it propagates through matter (path loss)
- Interference from other sources: Wireless network frequencies (e.g., 2.4GHz) shared by many devices (e.g., WiFi, cellular, motors): interference
- Multipath propagation: radio signal reflects off objects ground, arriving at destination at slightly different times.

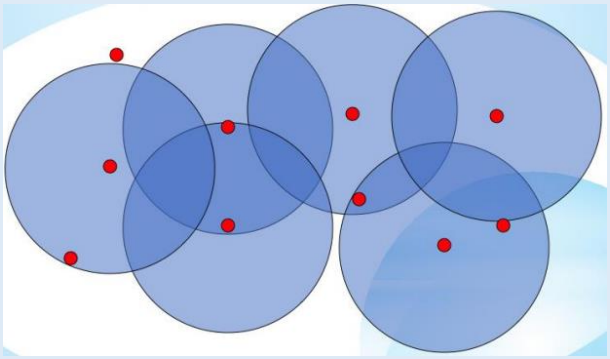
Makes communication across (even a point to point) wireless link much more "difficult"

Multipath Effects

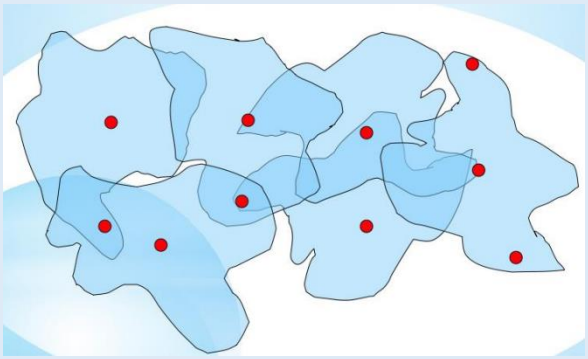
- Signals bounce off surface and interfere (constructive or destructive) with one another
- Self-interference



Ideal radios:



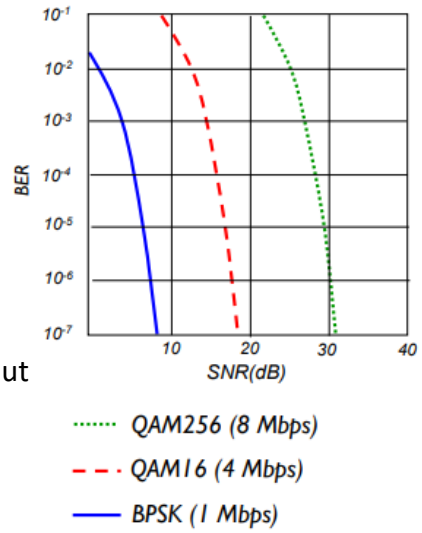
Real Radios:



Wireless Link Characteristics (II)

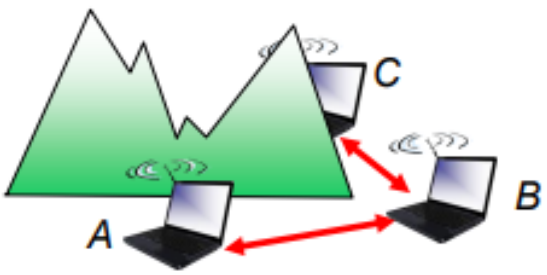
SNR: Signal-to-Noise ratio
- Larger SNR – easier to extract signal from noise (a “good thing”)

SNR versus BER tradeoffs
- Given physical layer:
  increase power -> increase SNR -> decrease SNR
- Given SNR:
  choose physical layer that meets BER requirement, giving highest throughput
  SNR may change with mobility: dynamically adapt physical layer (modulation technique, transmission rate)

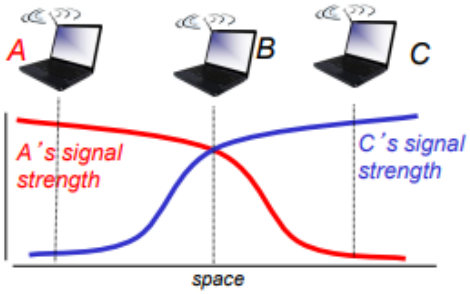


BER = Bit Error Rate

Wireless Link Characteristics (III)



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



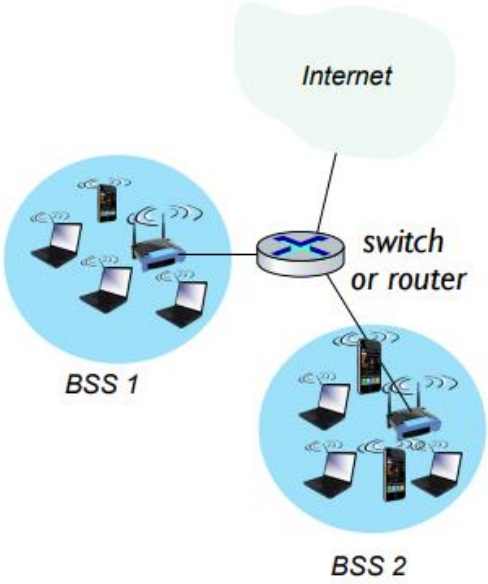
Signal attenuation:
- B, A hear each other
- B, C hear each other
- A, C can not hear each other interfering at B

IEEE 802.11 Wireless LAN

IEEE 802.11 standard	Year	Max data rate	Range	Frequency
802.11b	1999	11 Mbps	30 m	2.4 Ghz
802.11g	2003	54 Mbps	30m	2.4 Ghz
802.11n (WiFi 4)	2009	600	70m	2.4, 5 Ghz
802.11ac (WiFi 5)	2013	3.47Gpbs	70m	5 Ghz
802.11ax (WiFi 6)	2021	14 Gbps	70m	2.4, 5 Ghz
802.11af	2014	35 – 560 Mbps	1 Km	unused TV bands (54-790 MHz)
802.11ah (WiFi Halow)	2017	347Mbps	1 Km	900 Mhz

All use CSMA/CA for multiple access, and have base-station and ad-hoc network versions

802.11: LAN Architecture



Wireless host communicates with base station
- Base station = access point (AP)

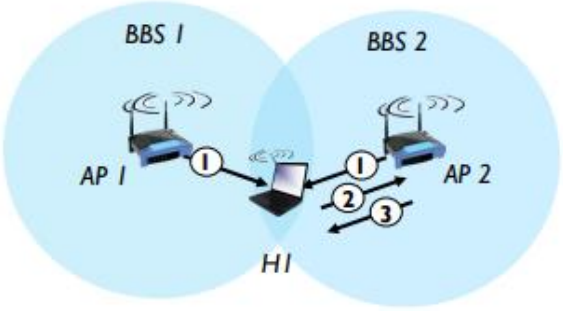
Basic Service Set (BSS) (aka "cell") in infrastructure mode contains:
- Wireless hosts
- Access point (AP): base station
- Ad hoc mode: hosts only

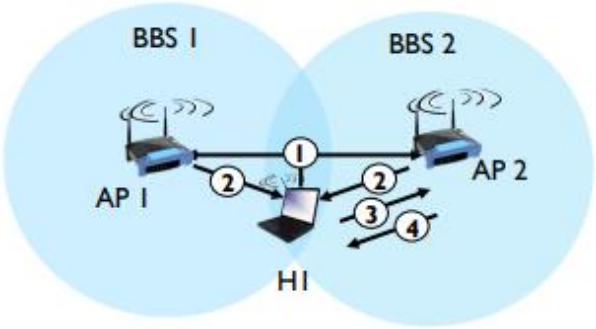
802.11: LAN Architecture

Spectrum divided into channels at different frequencies
- AP admin chooses frequency for AP
- Interference possible: channel can be same as neighbouring AP!

Arriving host: must associate with an AP
- Scans channels, listening for beacon frames containing AP's name (SSID) and MAC address
- Selects AP to associate with
- Then may perform authentication [Security]
- Then typically run DHCP to get IP address in AP's subnet

802.11: Passive/Active scanning





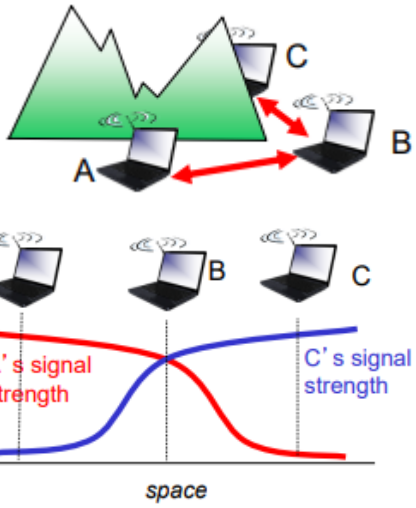
Passive scanning:

1. Beacon frames from Aps
2. Association Request frame sent: HI to selected AP
3. Association Response frame sent from selected AP to HI

Active scanning:

1. Probe Request frame broadcast from HI
2. Probe Response frames sent from Aps
3. Association Request frame sent: HI to selected AP
4. Association Response frame sent from selected AP to HI

IEEE 802.11: Multiple Access



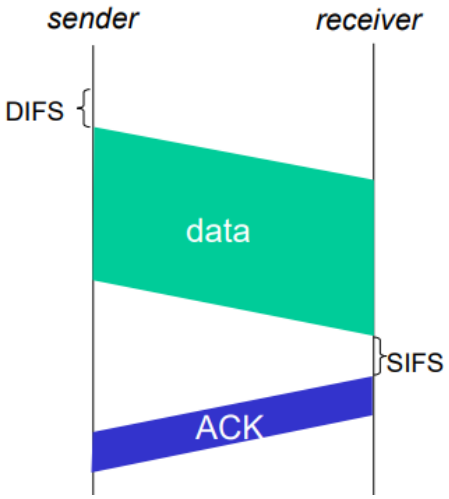
- Avoid collisions: 2+ nodes transmitting at same time
- 802.11: CSMA – sense before transmitting
    - Don't collide with detected ongoing transmission by another node
- 802.11: no collision detection!
    - Difficult to sense collisions: high transmitting signal, weak received signal due to fading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

Multiple Access: Key Points

No concept of a global collision
- Different receivers hear different signals
- Different senders reach different receives

Collisions are at receiver, not sender
- Only care if receiver can hear the sender clearly
- It does not matter if sender can hear someone else
- As long as that signal does not interfere with receiver

Goal of protocol
- Detect if receiver can hear sender
- Tell senders who might interfere with receiver to shut up



IEEE 802.11 MAC Protocol: CSMA/CA

802.11 sender

1. If sense channel idle for DIFS:
   Transmit entire frame (NO CD)
2. If sense channel busy:
   Start random backoff time
   Timer counts down while channel idle
   Transmit when timer expires
   If no ACK, double random backoff interval, repeat 2

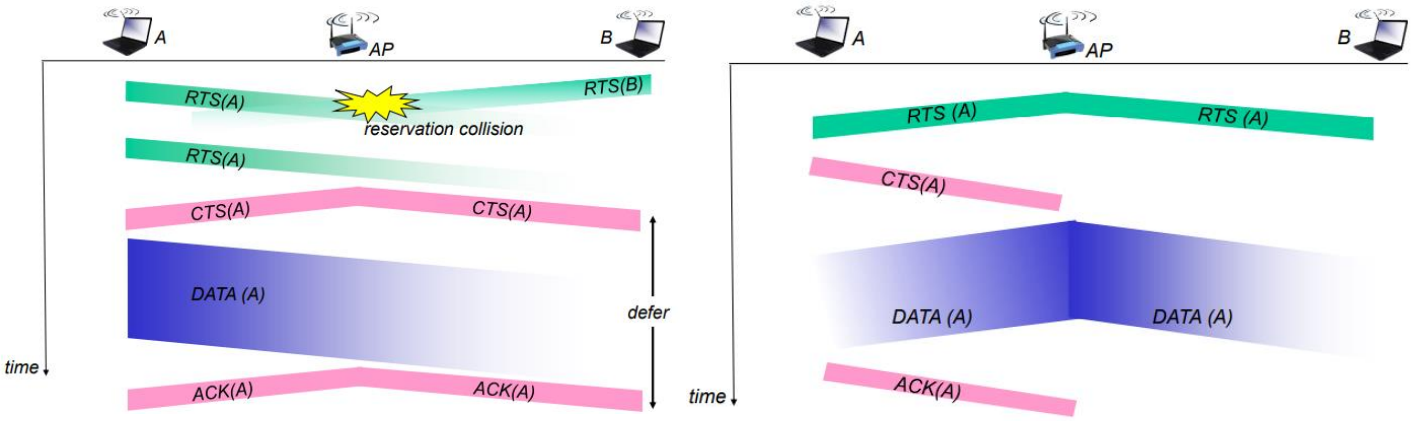
802.11 receiver

If frame received OK:
Return ACK after **SIFS** (ACK needed due to hidden terminal problem)

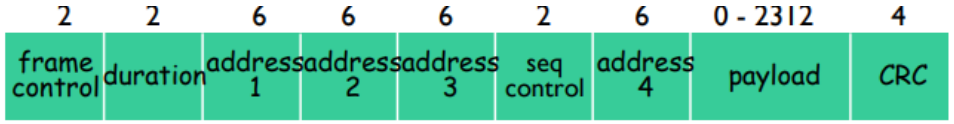
Avoiding Collisions (cont.)

Idea: sender "reserves" channel use for data frames using small reservation packets
- Sender first transmits small *request-to-send* (RTS) packet to BS using CSMA
    - RTSs may still collide with each other (but they're short)
- BS broadcasts clear-to-send CTS in response to RTS
- CTS heard by all nodes
    - Sender transmits data frame
    - Other stations defer transmissions

Collision Avoidance: RTS-CTS Exchange



802.11 Frame: Addressing

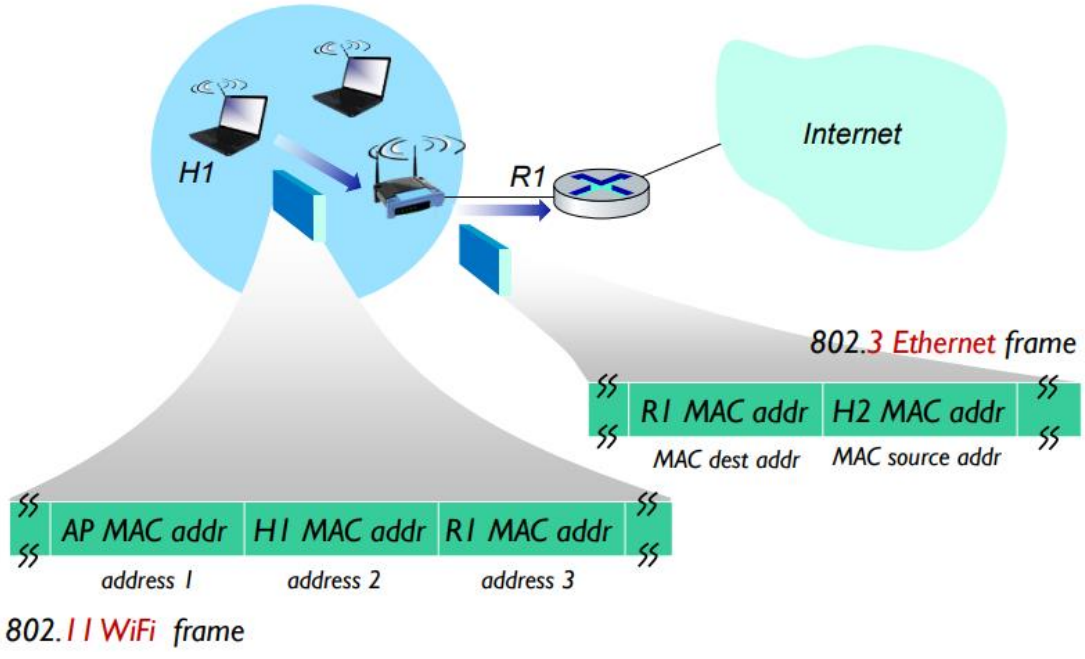


Address 1: MAC address of wireless host or AP to receive this frame

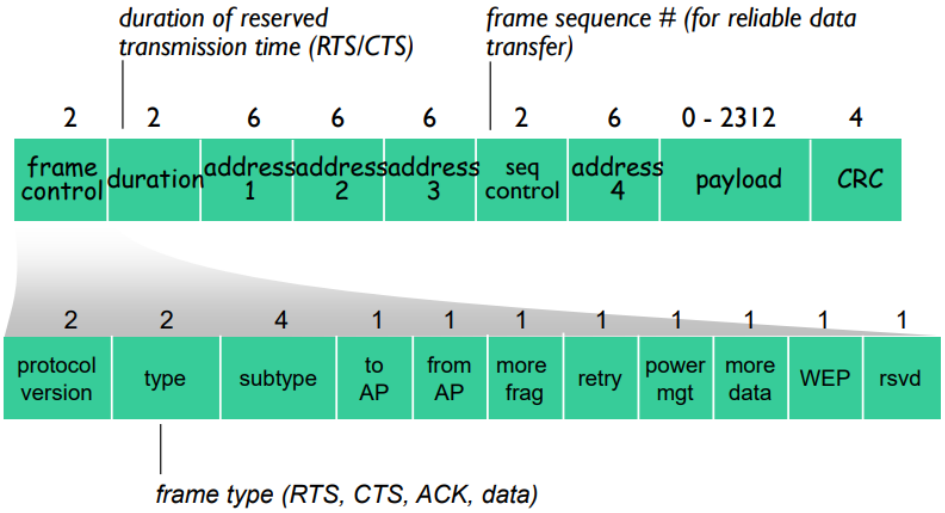
Address 2: MAC address of wireless host or AP transmitting this frame

Address 3: MAC address of router interface to which AP is attached

Address 4: used only in ad hoc mode



802.11 Frame: Addressing (cont.)

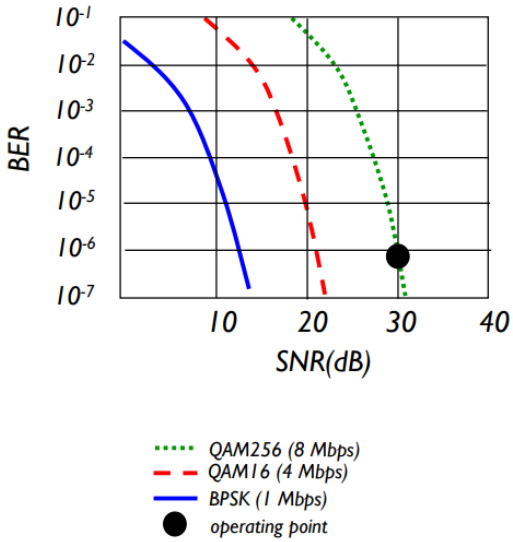


802.11: Advanced Capabilities

Rate Adaption

Base station, mobile dynamically change transmission rate (physical layer modulation technique) as mobile moves, SNR varies

1. SNR decreases, BER increase as node moves away from base station
2. When BER becomes too high, switch to lower transmission rate but with lower BER



Network Security

Confidentiality: only sender, intended receiver should "understand" message contents

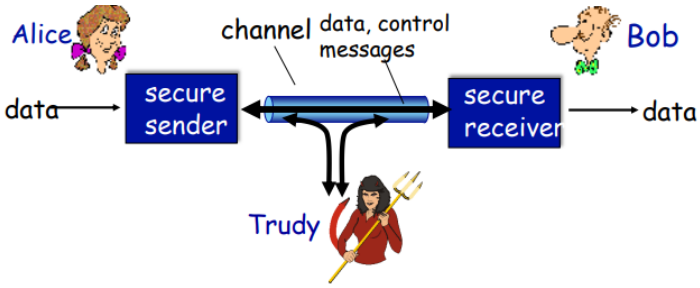
- Sender encrypts message
- Receiver decrypts message

Authentication: sender, receiver want to confirm identity of each other

Message integrity: sender, receiver want to ensure message not altered (in transit, or afterwards) without detection

Access and availability: services must be accessible and available to users

Friends and Enemies: Alice, Bob, Trudy

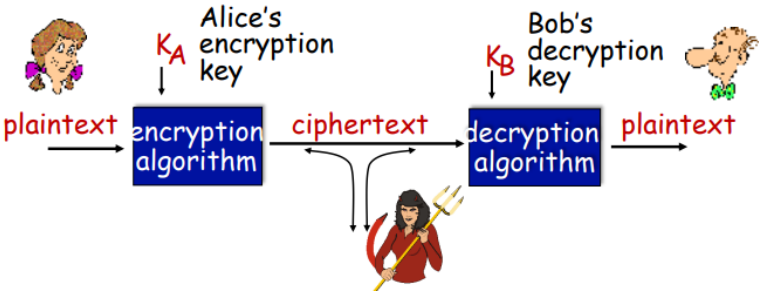


- Well-known in network security world
- Bob, Alice (lovers!) want to communicate *securely*
- Trudy (intruder) may intercept, delete, add messages

Bob and Alice can be metaphors for:
- Web browser/server for electronic transactions (e.g., on-line purchases)
- On-line banking client/server
- DNS servers
- BGP routers exchanging routing table updates
- Other examples?

What can *'Trudys'* do?
- Eavesdrop: intercept messages
- Actively insert messages into connection
- Impersonation: can fake (spoof) source address in packet (or any field in packet)
- Hijacking: "take over" ongoing connection by removing sender or receiver, inserting self in place
- Denial of Service: prevent service from being used by others (e.g., by overloading resources)

Cryptography



m: plaintext message
$K_A(\mathrm{m})$: ciphertext, encrypted with key K_A
m = $K_B(K_A(\mathrm{m}))$

Breaking an encryption scheme

Cipher-text only attack:
Trudy has ciphertext she can analyze

Two approaches:
- Brute force: search through all keys
- Statistical analysis

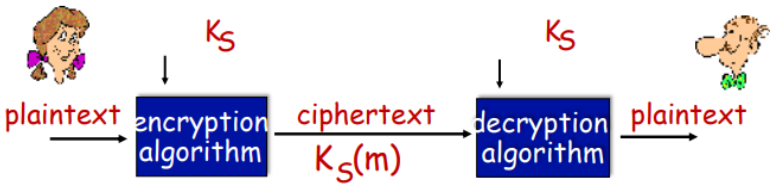
Known-plaintext attack: Trudy has plaintext corresponding to ciphertext
- E.g., in monoalphabetic cipher, Trudy determines pairings for a,l,i,c,e,b,o

Chosen-plaintext attack: Trudy can get ciphertext for chosen plaintext

Symmetric key cryptography

Symmetric key crypto: Bob and Alice share same (symmetric) key: K
- E.g. key is knowing substitution pattern in mono alphabetic substitution cipher

Q: How is a key value agreed upon?



Simple Encryption Scheme

Substitution cipher: substituting one thing for another

***Ceaser Cipher*:** replace each letter of the alphabet with the letter standing three places further down the alphabet.

Plain : a b c d e f g h i j k l m n o p q r s t u v w x y z
cipher: d e f g h i j k l m n o p q r s t u v w x y z a b c

e.g.: **Plaintext: meet me after the party**

ciphertext: phhw ph diwhu wkh sduwb

Encryption key: $c = (p + 3)\ mod\ 26$
Each plaintext letter *p* substituted by the ciphertext letter c
In general, we have $c = (p + k)\ mod\ 26$
Where *k* is in range 1 to 25

Monoalphabetic cipher: substitute one letter for another

plaintext: abcdefghijklmnopqrstuvwxyz

ciphertext: mnbvcxzasdfghjklpoiuytrewq

e.g.: **Plaintext: bob. i love you. alice**
ciphertext: nkn. s gktc wky. mgsbc

Encryption key: mapping from set of 26 letters to set of 26 letters
We have 26! (> 4 x 10^{26}) possible keys

Breaking an encryption scheme

Monoalphabetic ciphers are easy to break because they reflect the frequency data of the original alphabet

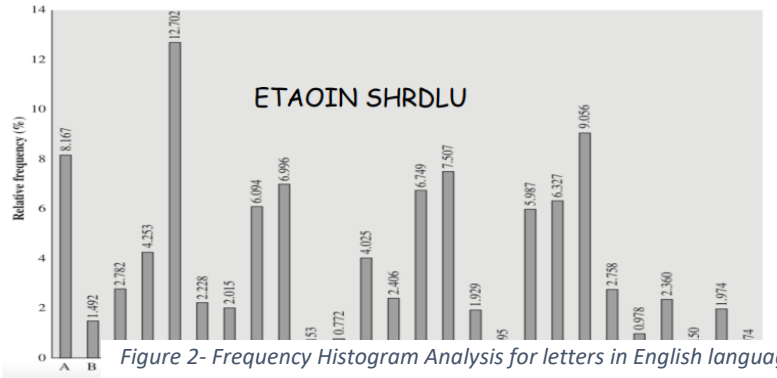


Figure 2- Frequency Histogram Analysis for letters in English language

A more sophisticated encryption approach

- *n* substitution ciphers, M_1, M_2, …, M_n
- cycling pattern:
  e.g., n=4: M_1, M_2, M_3, M_4, M_3, M_2; M_2, M_3, M_4, M_3, M_2;
- For each new plaintext symbol, use subsequent substitution pattern in cyclic pattern
  dog: d from M_1, o from M_3, g from M_4

  Encryption key: n substitution ciphers, and cyclic pattern
  Key need not just be *n*-bit pattern

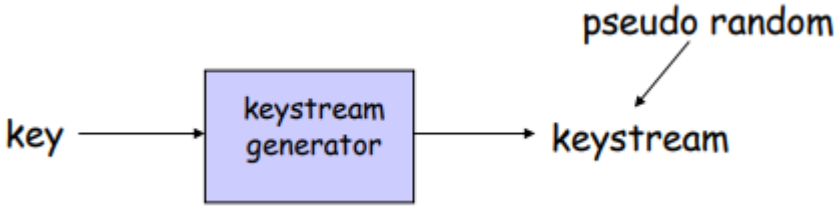
Two types of symmetric ciphers

Stream ciphers
- Encrypt one bit at time

Block ciphers
- Break plaintext message in equal size blocks
- Encrypt each block as a unit

Stream Ciphers

- Combine each bit of keystream with bit of plaintext to get bit of ciphertext
- m(i) = ith bit of message
- ks(i) = ith bit of keystream
- c(i) = ith bit of ciphertext
- c(i) = ks(i) ⊕ m(i) (XOR)
- m(i) = ks(i) ⊕ c(i)



RC4 Stream Cipher

RC4 is a popular stream cipher
- Extensively analysed and considered good
- Key can be from 1 to 256 bytes

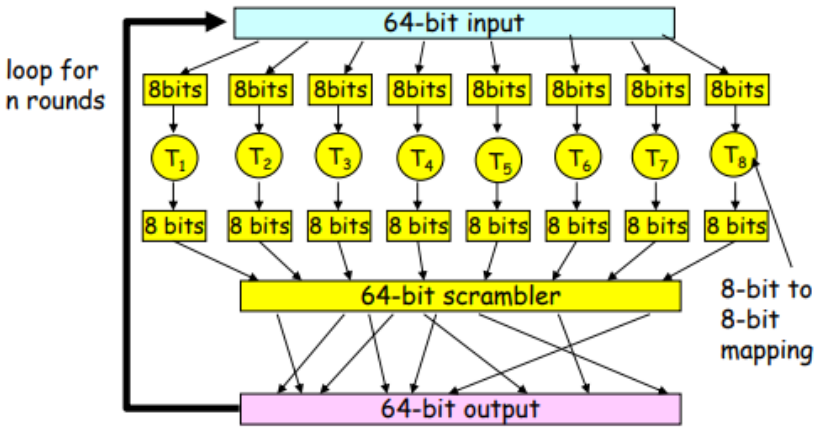
- Used in WEP, WPA for 802.11 and BitTorrent
- Known to have vulnerabilities
- Many other alternatives: ChaCha, SOBER, SEAL, …

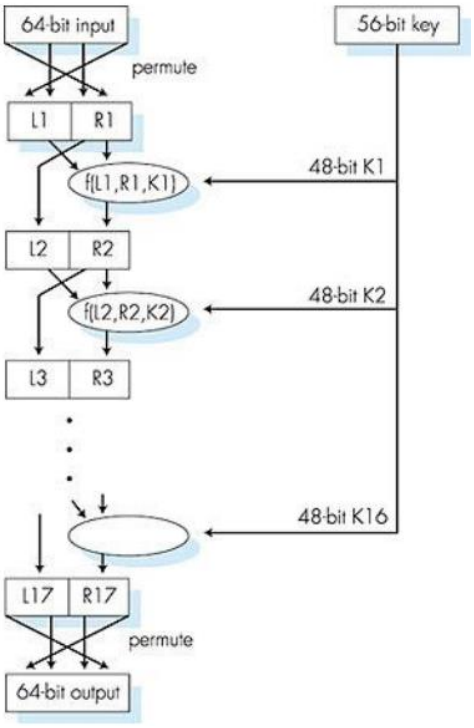
Block Cipher

- Ciphertext processed as *k* bit blocks
- 1-to-1 mapping is used to map k-bit block of plaintext to k-bit block of ciphertext
- E.g: k=3 (see table)
  - `010110001111 => 101000111001`
- Possible permutations = 8! (40,320)
- To prevent brute force attacks
  - Choose large K (64,128, etc)
- Full-table block ciphers not scalable
  - E.g., for k = 64, a table with 2^{64} entries required
  - Instead use function that simulates a randomly permutated table

Input	Output
000	110
111	001
001	111
010	101
011	100
100	011
101	010
110	000

- If only a single round, then one bit of input affects at most 8 bits of output
- In 2nd round, the 8 affected bits get scattered and inputted into multiple substitution boxes
- How many rounds?
  - How many times do you need to shuffle cards
  - Becomes less efficient as *n* increases





Symmetric Key Crypto: DES

DES: Data Encryption Standard

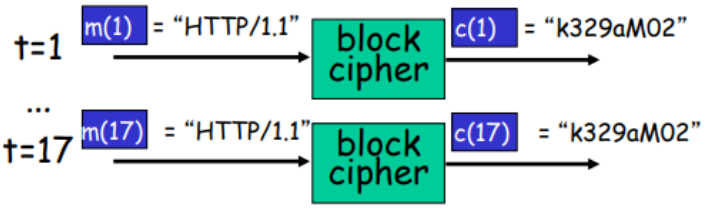
- US encryption standard [NIST 1993]
- 56-bit symmetric key, 64-bit plaintext input
- Block cipher with cipher block chaining
- How secure is DES?
  - DES Challenge: 56-bit-key-encrypted phrase decrypted (brute force) in less than a day
  - No known good analytic attack
- Making DES more secure:
  - 3DES: encrypt 3 times with 3 different keys

DES operation:
- Initial permutation
- 16 identical "rounds" of function application, each using different 48 bits of key
- Final permutation

AES: Advanced Encryption Standard

- Symmetric-key NIST standard, replaced DES (Nov 2001)
- Processes data in 128 bit blocks
- 128, 192, or 256 bit keys
- Brute force decryption (try each key) taking 1 sec on DES, takes 149 trillion years for AES

Cipher Block Chaining (CBC)



Cipher block: if input block repeated, will produce same cipher text.

Use random numbers: XOR ith input block, m(i) and random number r(i) and apply block cipher encryption algorithm

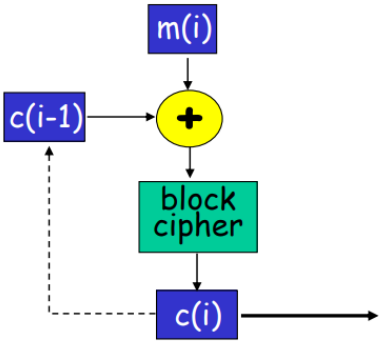
- c(i) = Ks(m(i) XOR r(i))
- Send across c(i) and r(i)

Input	Output
000	110
111	001
001	111
010	101
011	100
100	011
101	010
110	000

CBC Example

- Plaintext: `010010010`
- If no CBC, sent txt: `101 101 101`
  - 1-to-1 mapping table used
- Let's use the following random bits
  - r1: `001`, r2: `111`, r3: `100`
  - `XOR` the plaintext with these random bits
  - `010 XOR 001 = 011`
  - Now do table lookup for `011 -> 100`
- We need `c(1)=100, c(2)=010 and c(3)=000`, although plaintext is the same (`010`)
- Need to transmit twice as many bits (`c(i)` as well as `r(i)`)

Cipher Block Chaining



Send only one random value along with the very first message block, and then have the sender and receiver use the computed cipher block in place of the subsequent random number

XOR *i*th input block, m(i), with previous block of cipher text, c(i-1)

- c(0) is an initialisation vector (random) transmitted to receiver in clear

CBC generates its own random numbers

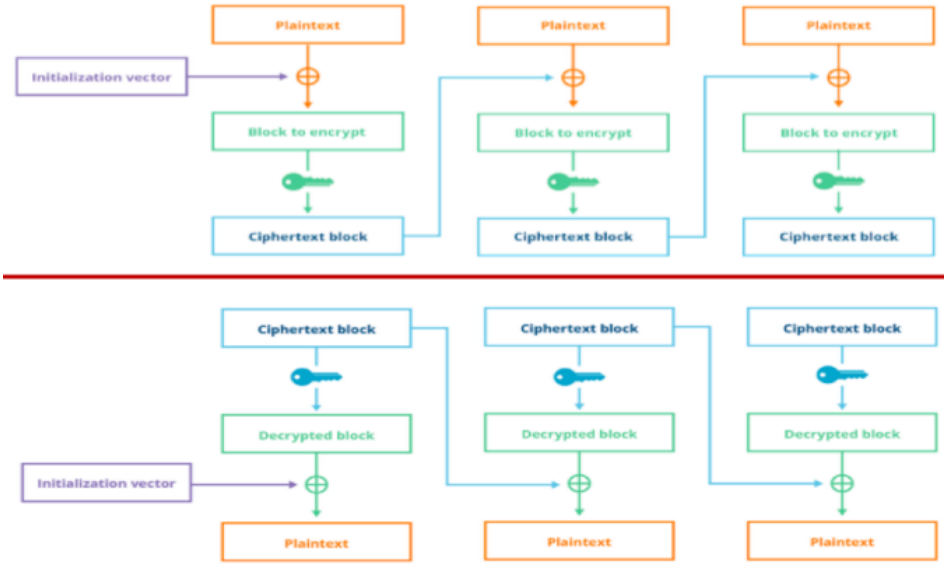
- Have encryption of current block depend on result of previous block
- $c(i) = K_s(m(i) \text{ XOR } c(i-1))$
- $m(i) = K_s(c(i)) \text{ XOR } c(i-1)$

How do we encrypt first block?

- Initialisation vector (IV): random block = c(0)
- IV does not have to be secret

Change IV for each message (or session)

- Guarantees that even if the same message is sent repeatedly, the ciphertext will be completely different each time



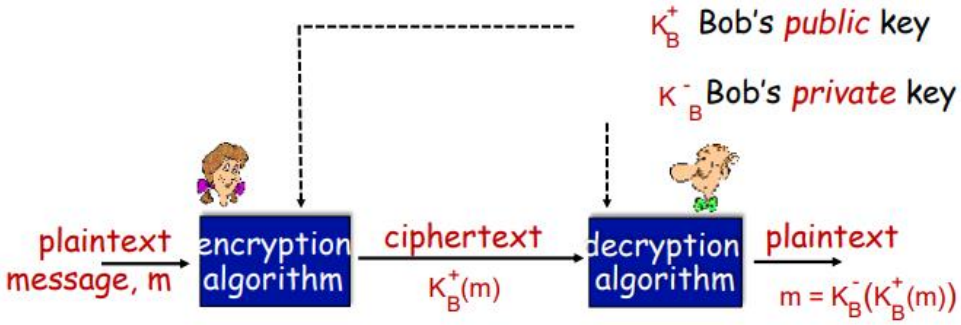
Public Key Cryptography

Symmetric key crypto:

- Requires sender, receiver know shared secret key
- **Q:** how to agree on key in first place?

Public key crypto:

- *Radically* different approach
- Sender, receiver do *not* share secret key
- Public encryption key known to *all*
- Private decryption key known only to receiver



Public Key Encryption Algorithms

Requirements:

1. Need $K_B^+(\cdot)$ and $K_B^-(\cdot)$ such that:
   $$K_B^+(K_B^+(m)) = m$$
2. Given public key K_B^+, it should be impossible to compute private key K_B^-

RSA: Rivest, Shamir, Adelson algorithm

Prerequisite: Modular Arithmetic

x mod n = remainder of x when divided by n

facts:

$$[(a \bmod n) + (b \bmod n)] \bmod n = (a+b) \bmod n$$
$$[(a \bmod n) - (b \bmod n)] \bmod n = (a-b) \bmod n$$
$$[(a \bmod n) * (b \bmod n)] \bmod n = (a*b) \bmod n$$

Thus:

$$(a \bmod n)^d \bmod n = a^d \bmod n$$

Example: x=14, n=10, d=2:

$$(x \bmod n)^d \bmod n = 4^2 \bmod 10 = 6$$

$X^d = 14^2 = 196$ $x^d \bmod 10 = 6$

RSA: getting ready

Message: just a bit pattern

Bit pattern can be uniquely represented by an integer number

Thus, encrypting a message is equivalent to encrypting a number

Example:
m=10010001. This message is uniquely represented by the decimal number 145.

To encrypt m, we encrypt the corresponding number, which gives a new number (the ciphertext).

RSA: Creating public/private key pair

1. Choose two large prime numbers, p, q. (e.g., 1024 bits each)
2. Compute $n = pq$, $z = (p-1)(q-1)$
3. Choose e (with $e<n$) that has no common factors with z (e, z are "relatively prime").
4. Choose d such that $ed-1$ is exactly divisible by z. (in other words: $ed \bmod z = 1$)
5. Public key = (n, e) K_B^+
   Private key = (n, d) K_B^-

RSA: encryption, decryption

1. Given (n, e) and (n, d) as computed above
2. To encrypt message *m* (<n), compute
   $c = m^e \bmod n$
3. To decrypt received bit pattern, c, compute
   $m = c^d \bmod n$

Magic happens!

$$m = (m^e \bmod n)^d \bmod n$$

where $c = m^e \bmod n$

RSA: another important property

The following property will be *very* useful later:

$K_B^-(K_B^+(m)) = m = K_B^+(K_B^-(m))$

Use public key first, followed by private key

Use private key first, followed by public key

Same result!

Why is RSA secure?

- Suppose you know *Bob*'s public key (n, e). How hard is it to determine d?
- Essentially need to find factors of n without knowing the two factors p and q
  Fact: factoring a big number is *hard*

RSA in practice: session keys

- Exponentiation in RSA is computationally intensive
- DES is at least 100 times faster than RSA
- Use public key crypto to establish secure connection, then establish second key – symmetric session key – for encrypting data

Session key, K_s

- Bob and Alice use RSA to exchange a symmetric session key K_s
- Once both have K_s, they use symmetric key cryptography

Authentication

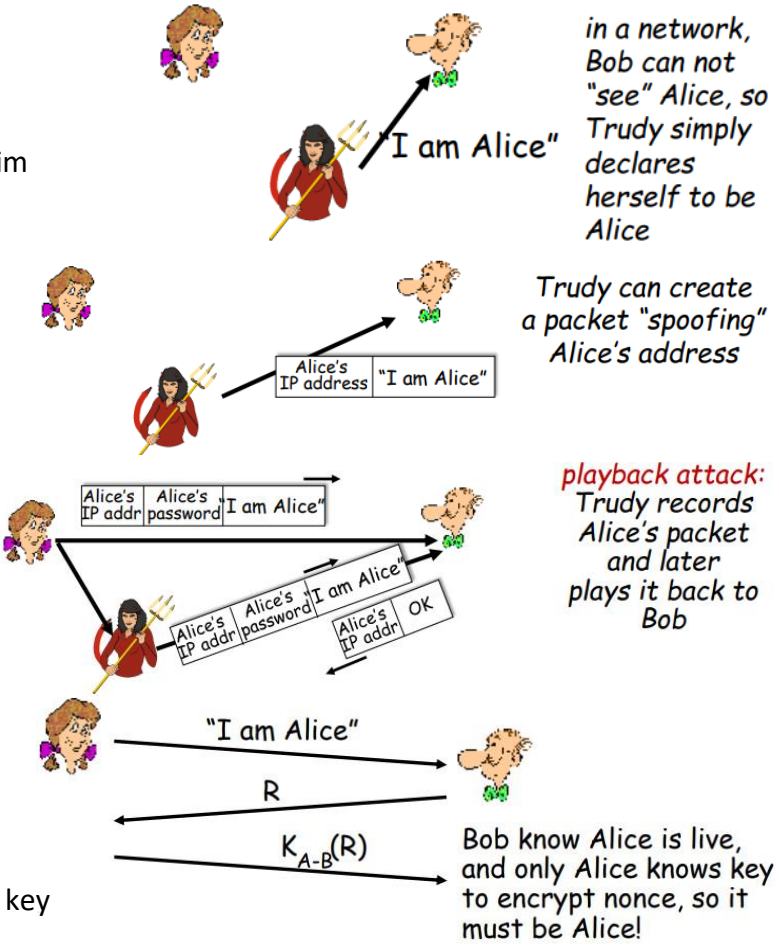
Goal: *Bob* wants *Alice* to "prove" her identity to him

Protocol ap 1.0:
Alice says "I am Alice"

Protocol ap 2.0:
Alice says "I am Alice" in an IP packet containing her source IP address

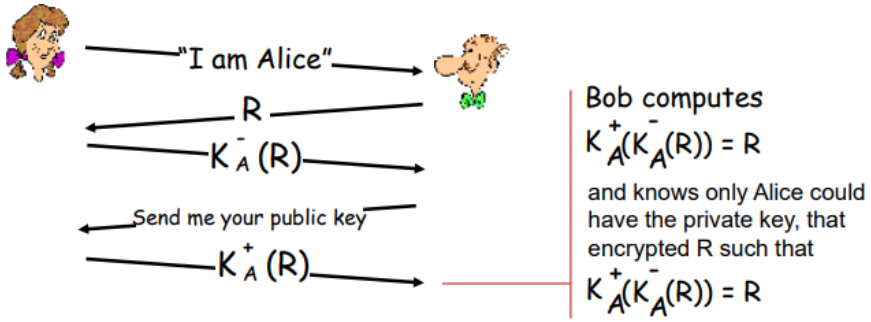
Protocol ap 3.0:
Alice says "I am Alice" and sends her secret password to "prove" it

Goal: Avoid playback attack (use symmetric key)
nonce: number ® used only once-in-a-lifetime
Protocol ap 4.0:
Prove Alice "live", Bob sends Alice nonce, R
Alice must return R, encrypted with shared secret key

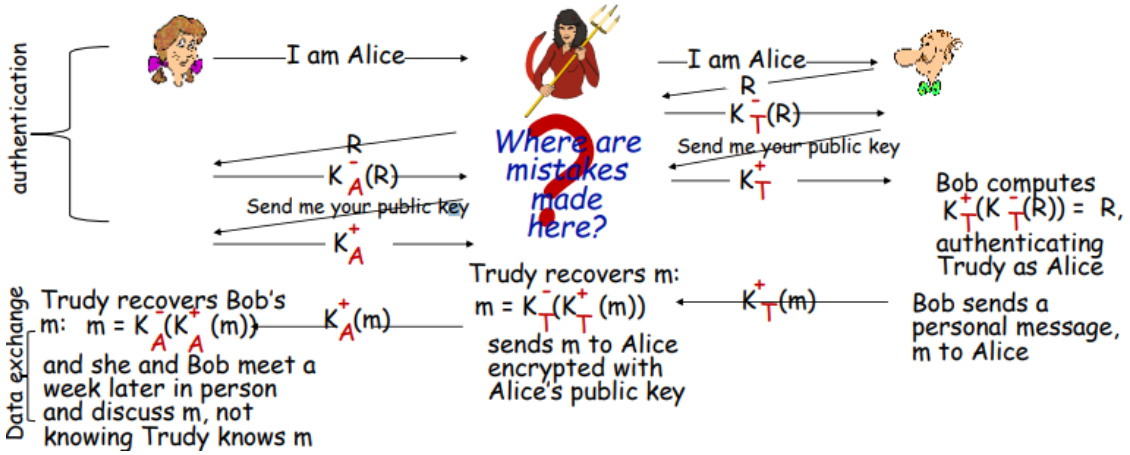


Goal: use public key techniques (ap4.0 uses symmetric)

Protocol ap 5.0: Use nonce, public key cryptography



Flaw: 'Man-in-the-middle attack' – Trudy poses as Alice (to Bob) or Bob (to Alice)



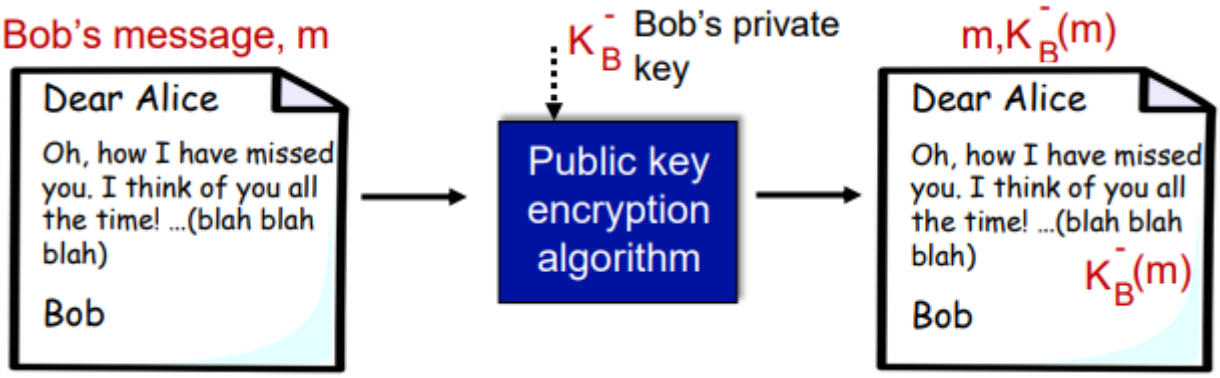
Confidentiality vs Integrity

- Confidentiality: message private and secret
- Integrity: protection against message tampering
- **Encryption alone does not guarantee integrity**
  - Attacker can modify message under encryption without learning what it is
- Public Key Crypto Standard (PKCS)
  - "RSA encryption is intended primarily to provide confidentiality, *not integrity*"
- Both confidentiality and integrity are needed for security

Digital signatures

Cryptographic technique analogous to hand-written signatures:

- Sender (Bob) digitally signs document: he is document owner/creator.
- Verifiable, nonforgeable: recipient (Alice) can prove to someone that Bob, and no one else (including Alice), must have signed document
- Simple digital signature for message m:
  - Bob signs m by encrypting with his private key K_b, creating "signed" message $K_b^-(m)$



Digital Signatures (cont.)

- Suppose Alice receives message m, with signature: m, $K_b^-(m)$
- Alice verifies m signed by Bob by applying Bob's public key K_b^+ to $K_b^-(m)$ then checks if $K_B^+(K_B^-(m)) = m$
- If $K_B^+(K_B^-(m)) = m$, whoever signed m must have used Bob's private key

Alice thus verifies that:

- Bob signed m
- No one else signed m
- Bob signed m and not m'

Non-repudiation:

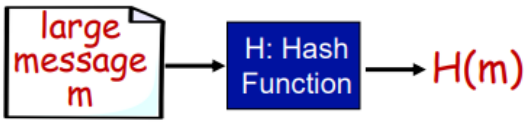
- Alice can take m, and signature $K_B^-(m)$ to court and prove that Bob signed m

Message Digests

Computationally expensive to public-key-encrypt long messages

Goal: fixed-length, easy-to-compute digital "fingerprint"

- Apply hash function H to m, get fixed size message digest, $H(m)$



Hash function properties:

- Many-to-1
- Produces fixed-size msg digest (fingerprint)
- Given message digest x, computationally infeasible to find m such that $x = H(m)$

Internet Checksum: poor crypto hash function

Internet checksum has some properties of hash function:

- Produces fixed length digest (16-bit sum) of message
- Is many-to-one

However, given message with given hash value, it is easy to find another message with same hash value:

message	ASCII format
I O U 1	49 4F 55 31
0 0 . 9	30 30 2E 39
9 B O B	39 42 D2 42
	B2 C1 D2 AC

message	ASCII format
I O U 9	49 4F 55 39
0 0 . 1	30 30 2E 31
9 B O B	39 42 D2 42
	B2 C1 D2 AC

different messages but identical checksums!

Hash function algorithms

MD5 hash function widely used (RFC 1231)

- Computes 128-bit message digest in 4-step process.
- Arbitrary 128-bit string x, appears difficult to construct message m whose MD5 hash is equal to x

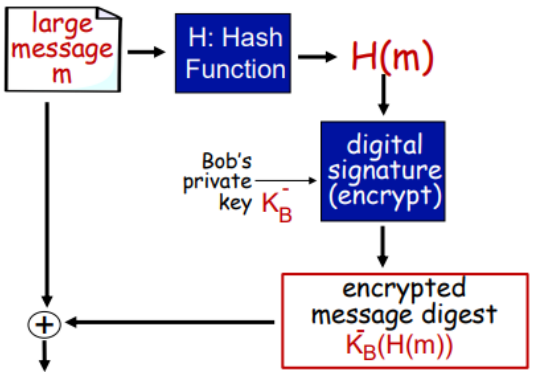
SHA-I is also used

- US standard [NIST, FIPS PUB 180-1]
- 160-bit message digest

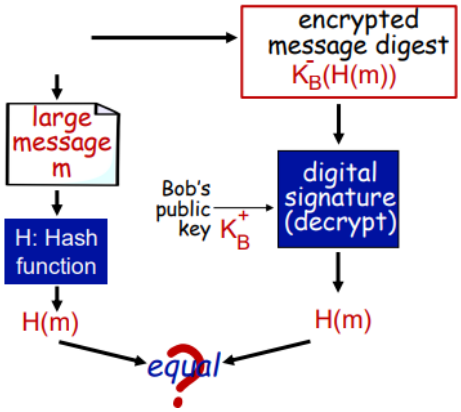
SHA-2 and SHA-3 (recent standard) are better – security

Digital signature = signed message digest

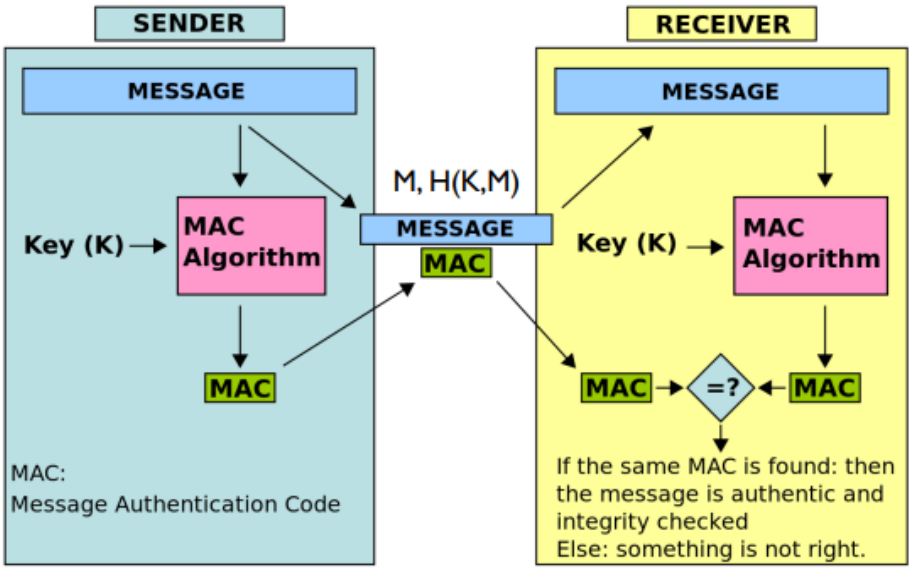
Bob sends digitally signed message:



Alice verifies signature, integrity of digitally signed message:



Message Authentication Code (MAC)



Digital signatures use asymmetric key cryptography

MAC allows a way to sign a message but using symmetric key, sender sends `(M,H(K,M))`

Requires a shared secret key K between the sender and receiver

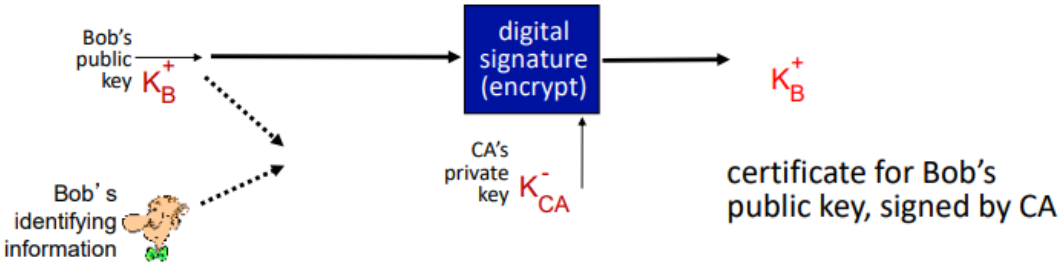
Examples: UMAC-VMAC, SipHash, Poly1305-AES

Public key Certification Authority (CA)

Certification Authority (CA): binds public key to particular entity, E

Entity (person, website, router) registers its public key with CE, provides "proof of id" to CA

- CA creates certificate bonding id E to E's pub key
- Certificate containing E's public key digitally signed by CA: CA says "this is E's public key"



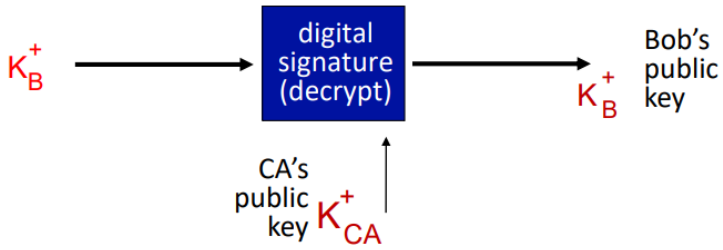
Need for certified public keys

Motivation: Trudy plays pizza prank on Bob

- Trudy creates e-mail order for 4 pizzas, posing as Bob
- Trudy signs order with her private key
- Trudy sends order to pizza store
- Trudy sends to Pizza Store "her" (Bob's) public key
- Pizza Store verifies signature, then delivers to Bob!

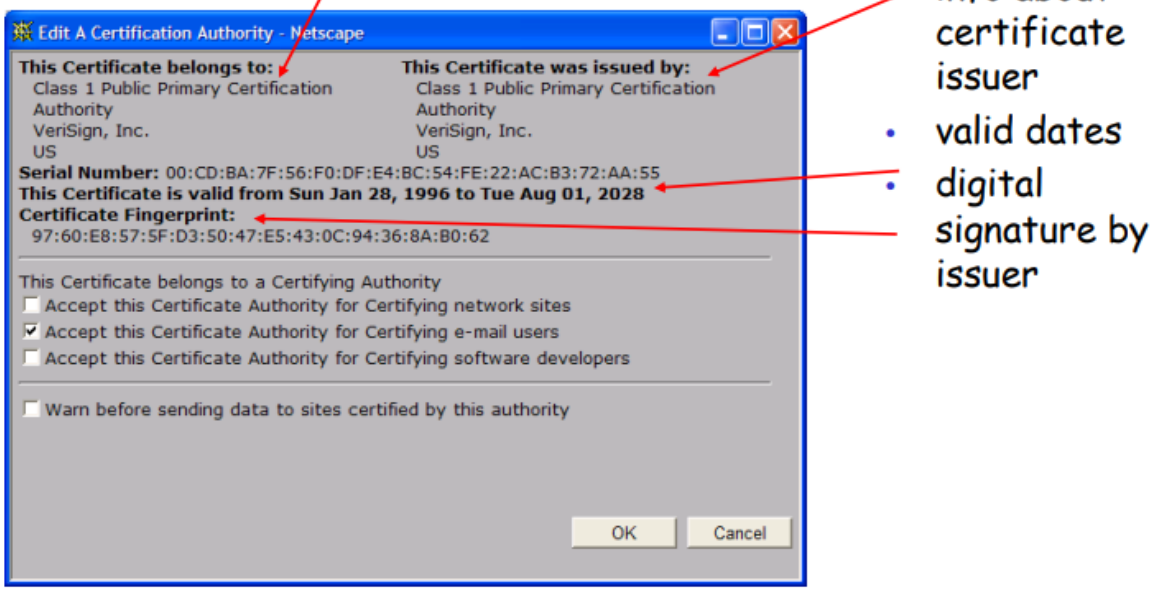
Public key Certification Authority (CA) cont.

When Alice wants Bob's public key:
- Gets Bob's certificate (Bob or elsewhere)
- Apply CA's public key to Bob's certificate, get Bob's public key



Certificate contents:

- Serial number (unique to issuer)
- info about certificate owner, including algorithm and key value itself (not shown)



- info about certificate issuer
- valid dates
- digital signature by issuer

Certificates: summary

Primary standard X.509 (RFC 2459)
Certificate contains:
- Issuer name
- Entity name, addr, domain name, etc.
- Entity's public key
- Digital signature (signed with issuer's private key)

Public-Key Infrastructure (PKI)
- Certificates and certification authorities
- Often considered "heavy"

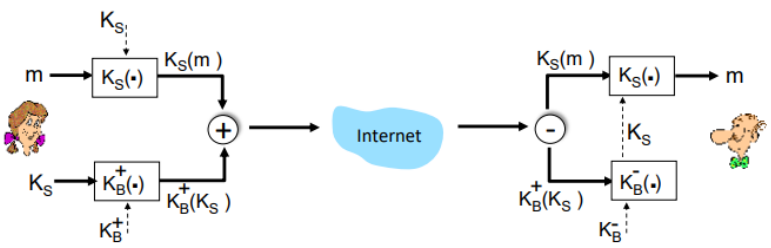
Secure E-mail: Confidentiality

Alice wants to send *confidential* email, m, to Bob.

Alice:
- Generates random symmetric private key, K_s
- Encrypts message with K_s (for efficiency)
- Also encrypts K_s with Bob's public key
- Send both $K_s(m)$ to $K_b^+(K_s)$ to Bob

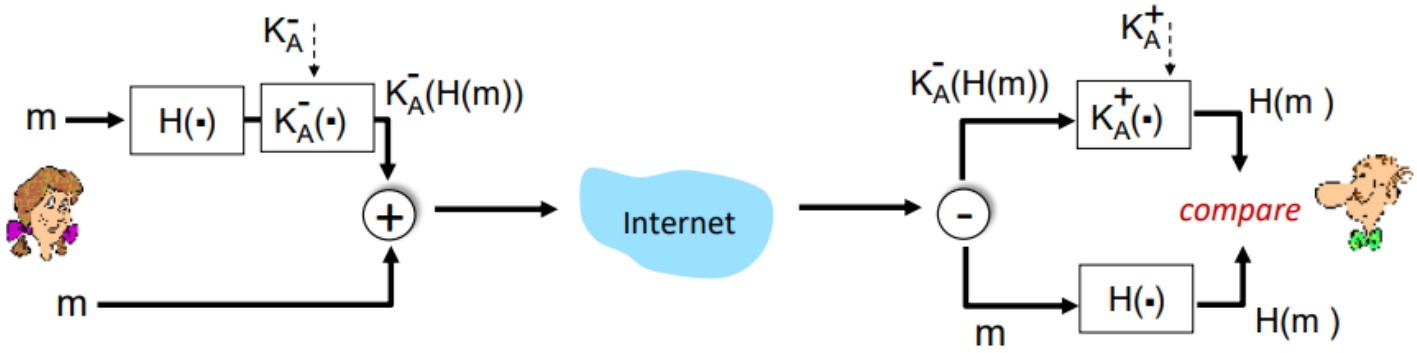
Bob:
- Uses his private key to decrypt and recover K_s

Uses K_s to decrypt $K_s(m)$ to recover m



Secure E-mail: Integrity, Authentication

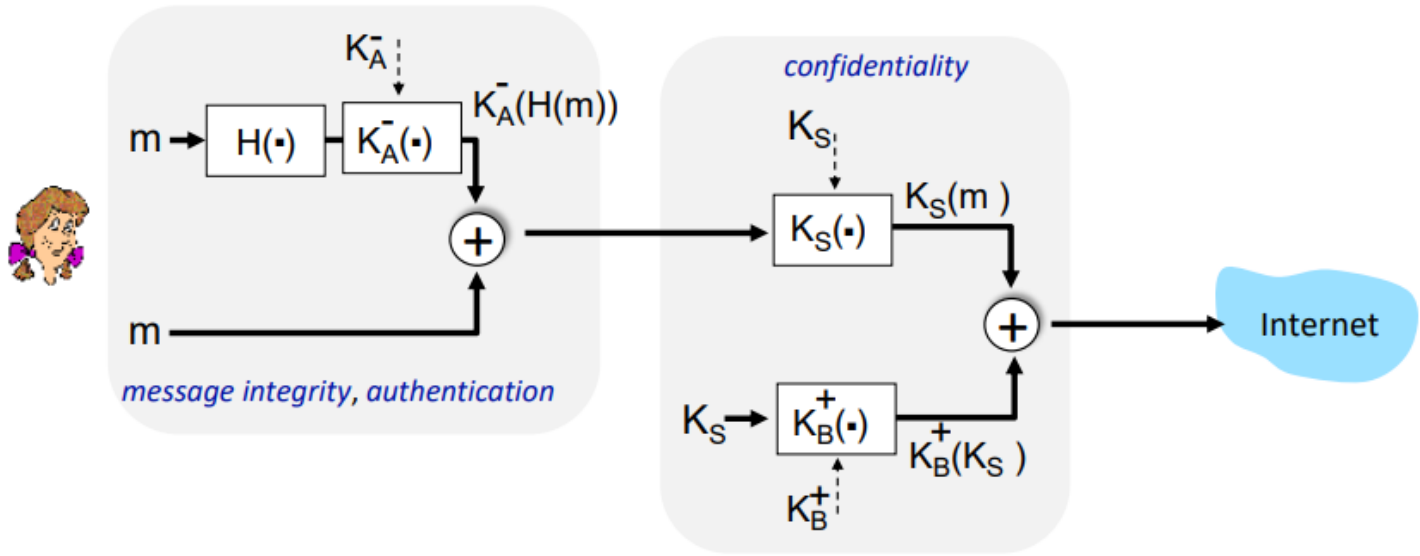
Alice wants to send m to Bob, with *message integrity, authentication*



- Alice digitally signs hash of her message with her private key, providing *integrity and authentication.*
- Sends both message (in the clear) and digital signature.

Secure E-mail: Confidentiality, Integrity, Authentication

Alice wants to send m to Bob, with confidentiality, *message integrity, authentication*



Alice uses three keys: her private key, Bob's public key, new symmetric key

Secure E-mail: PGP

- De-factor standard for email encryption
- On installation PGP creates public, private key pair
  - Public key posted on user's webpage or placed in a public key server
  - Private key protected by password
- Option to digitally sign the message, encrypt the message or both
- MD5 or SHA for message digest
- CAST, triple-DES or DEA for symmetric key encryption
- RSA for public key encryption

```
-----BEGIN PGP SIGNED MESSAGE-----
Hash:  SHA1
Bob:
Can I see you tonight?
Passionately yours, Alice
-----BEGIN PGP SIGNATURE-----
Version: PGP for Personal Privacy 5.0
Charset:  noconv
yhHJRHhGJGhgg/12EpJ+lo8gE4vB3mqJhFEvZP9t6n7G6m5Gw2
-----END PGP SIGNATURE-----
```

Figure 8.22 ♦ A PGP signed message

```
-----BEGIN PGP MESSAGE-----
Version: PGP for Personal Privacy 5.0
u2R4d+/jKmn8Bc5+hgDsqAewsDfrGdszX68liKm5F6Gc4sDfcXyt
RfdS10juHgbcfDssWe7/K=lKhnMikLo0+1/BvcX4t==Ujk9PbcD4
Thdf2awQfgHbnmKlok8iy6gThlp
-----END PGP MESSAGE
```

Figure 8.23 ♦ A secret PGP message