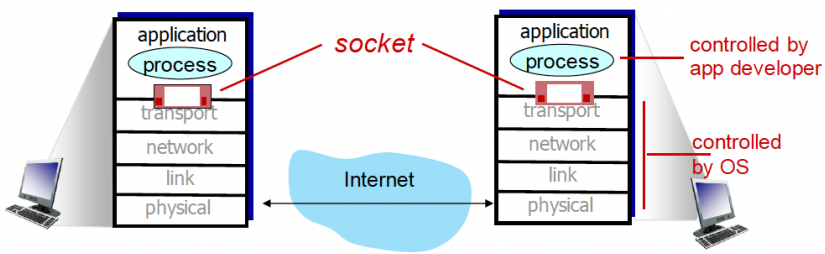
COMP211 Computer Networks

- **Application Layer**
  - Princliples
    - Network apps
      - social networking
      - Web
      - text messaging
      - e-mail
      - multi-user network games
      - streaming stored video (YouTube, Hulu, Netflix)
      - P2P file sharing
      - voice over IP (e.g., Skype)
      - real-time video conferencing
      - Internet search
      - remote login
      - ...
    - Client-Server architechture
      - server:
        - always-on host
        - permanent IP address
        - data centers for scaling
      - clients:
        - communicate with server
        - may be intermittently connected
        - may have dynamic IP addresses
        - do not communicate directly with each other
    - P2P architechture

- *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 addresses
  - complex management

Sockets

- process sends/receives messages to/from its socket
- socket analogous to door
  - sending process shoves message out of 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.umass.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 & respond to messages

open protocols:

- defined in RFCs, everyone has access to protocol definition
- allows for interoperability
- e.g., HTTP, SMTP

proprietary protocols:

- e.g., Skype

- Internet Transport Protocols 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 processes
  - 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.

- 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*
- see Chapter 8

- Web and HTTP
  - Web

First, a quick 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,...
- web page consists of *base HTML-file* which includes *several referenced objects, each* addressable by a *URL,* e.g.,

`www.someschool.edu` / `someDept/pic.gif`

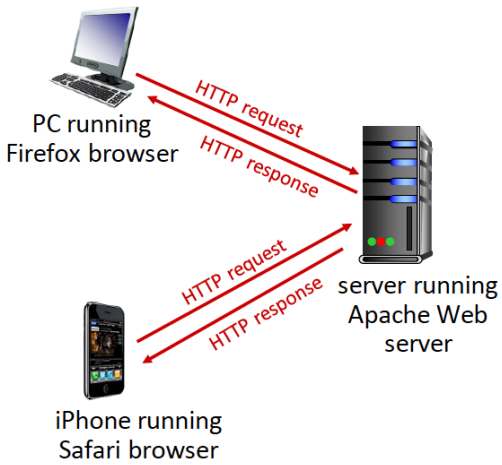
host name | path name

  - 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

aside

protocols that maintain "state" are complex!

- past history (state) must be maintained
- if server/client crashes, their views of "state" may be inconsistent, must be reconciled

- Two Types

Non-persistent HTTP

1. TCP connection opened
2. at most one object sent over TCP connection
3. TCP connection closed

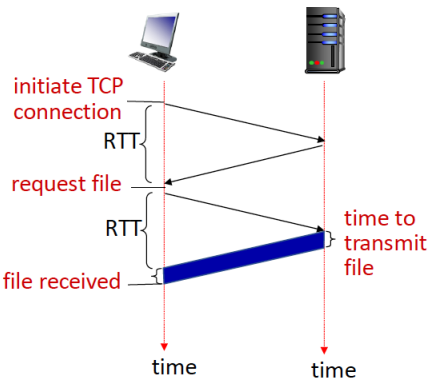
downloading multiple objects required multiple connections

Persistent HTTP

1. TCP connection opened to a server
2. multiple objects can be sent over *single* TCP connection between client, and that server
3. TCP connection closed

- Non-persistent
  - RTT (definition): time for a small packet to travel from client to server and back

    HTTP response time (per object):
    - one RTT to initiate TCP connection
    - one RTT for HTTP request and first few bytes of HTTP response to return
    - obect/file transmission time



Non-persistent HTTP response time = 2RTT+ file transmission time

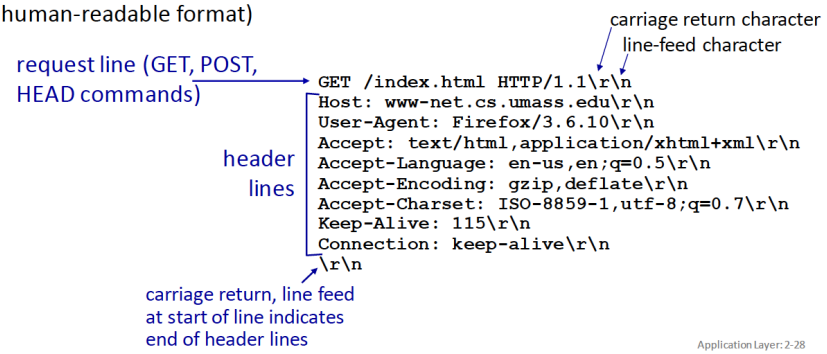
- Persistent
  - *Non-persistent HTTP issues:*
    - requires 2 RTTs per object
    - OS overhead for *each* TCP connection
    - browsers often open multiple parallel TCP connections to fetch referenced objects in parallel

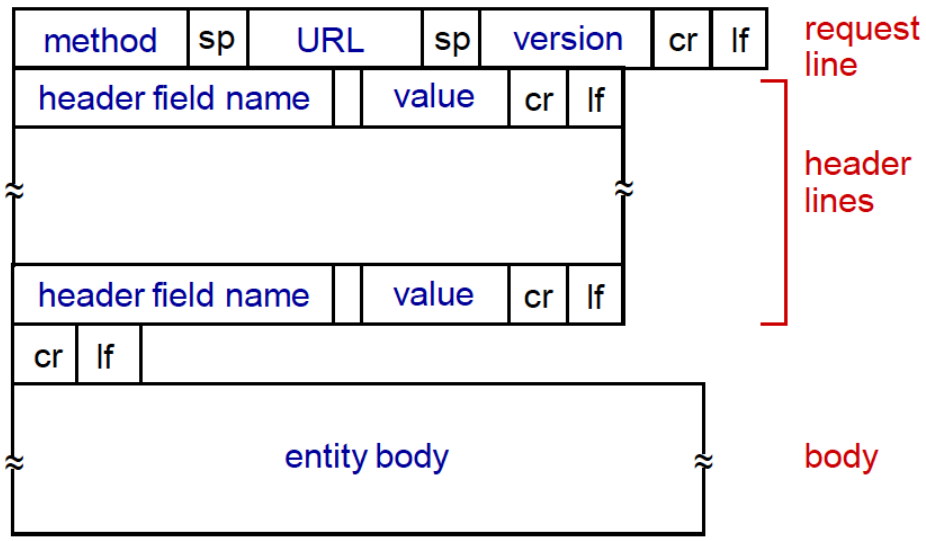
  Persistent HTTP (HTTP1.1):
    - server leaves connection open after sending response
    - subsequent HTTP messages between same client/server sent over open connection
    - client sends requests as soon as it encounters a referenced object
    - as little as one RTT for all the referenced objects (cutting response time in half)

- HTTP message

- Request Message

- 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

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

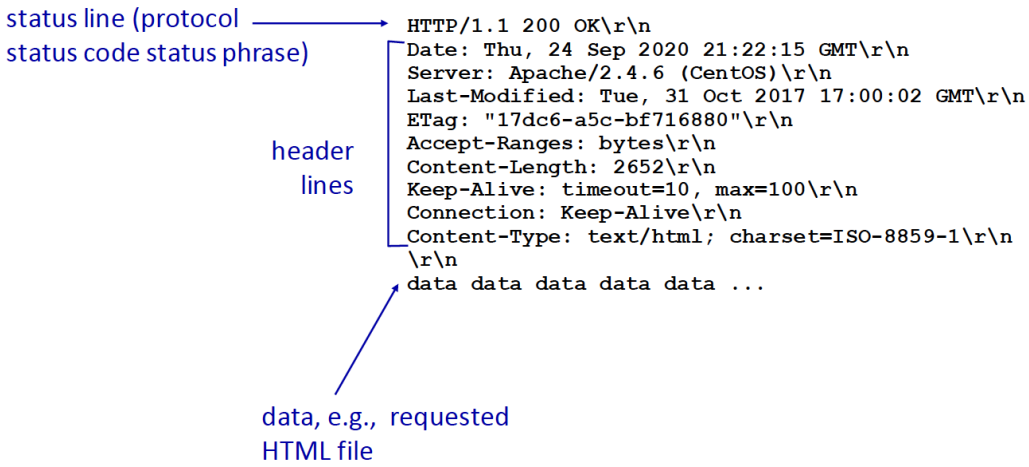
PUT method:
- uploads new file (object) to server
- completely replaces file that exists at specified URL with content in entity body of POST HTTP request message

GET method

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

```
www.somesite.com/animalsearch?monkeys&banana
```


- Respond Message



Response Status Code

- status code appears in 1st line in server-to-client response message.
- some sample codes:

200 OK
- request succeeded, requested object later in this message

301 Moved Permanently
- requested object moved, new location specified later in this message (in Location: field)

400 Bad Request
- request msg not understood by server

404 Not Found
- requested document not found on this server

505 HTTP Version Not Supported

Cookies

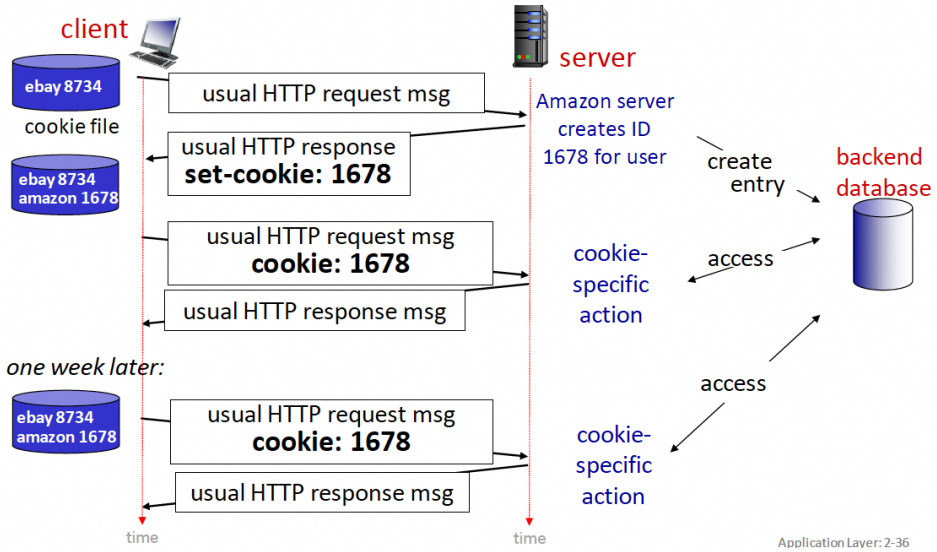
Many Web sites use

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, 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



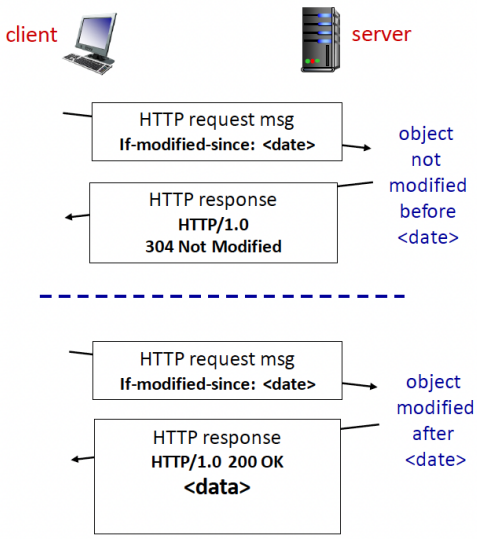
- Web caches (proxy servers)
- Conditional GET

Conditional GET

Goal: don't send object if cache has up-to-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



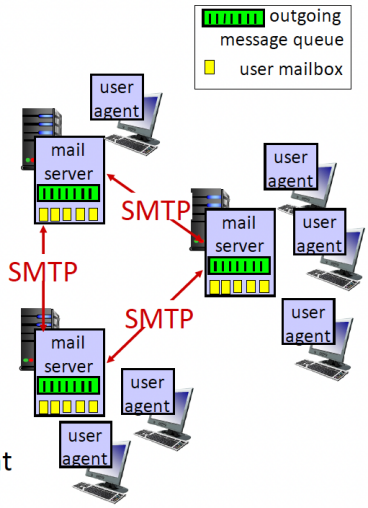
- E-Mail, SMTP, IMAP
  - EMail

Three major components:

- user agents
- mail servers
- simple mail transfer protocol: SMTP

User Agent

- a.k.a. "mail reader"
- composing, editing, reading mail messages
- e.g., Outlook, iPhone mail client
- outgoing, incoming messages stored on server



Mail Servers

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: sending mail server
  - "server": receiving mail server

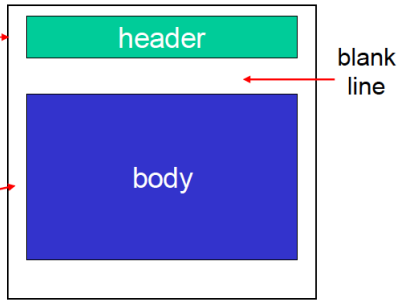
Mail message format

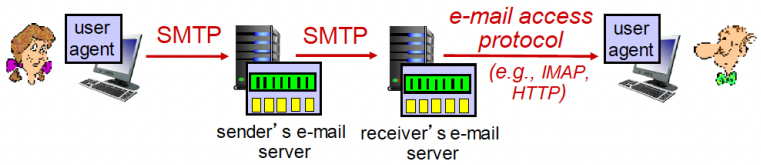
SMTP: protocol for exchanging e-mail messages, defined in RFC 531 (like HTTP)

RFC 822 defines *syntax* for e-mail message itself (like HTML)

- header lines, e.g.,
  - To:
  - From:
  - Subject:

  these lines, within the body of the email message area different from SMTP MAIL FROM:, RCPT TO: commands!
- Body:
  the "message" , ASCII characters only



- Mail Access Protocols
  - 

    - SMTP: delivery/storage of e-mail messages to receiver's server
    - mail access protocol: retrieval from server
      - IMAP: Internet Mail Access Protocol [RFC 3501]: messages stored on server, IMAP provides retrieval, deletion, folders of stored messages on server
      - HTTP: gmail, Hotmail, Yahoo!Mail, etc. provides web-based interface on top of SMTP (to send), IMAP (or POP) to retrieve e-mail messages
- Sample SMTP
  - ```
    S: 220 hamburger.edu
    C: HELO crepes.fr
    S: 250  Hello crepes.fr, pleased to meet you
    C: MAIL FROM: <alice@crepes.fr>
    S: 250 alice@crepes.fr... Sender ok
    C: RCPT TO: <bob@hamburger.edu>
    S: 250 bob@hamburger.edu ... Recipient ok
    C: DATA
    S: 354 Enter mail, end with "." on a line by itself
    C: Do you like ketchup?
    C: How about pickles?
    C: .
    S: 250 Message accepted for delivery
    C: QUIT
    S: 221 hamburger.edu closing connection
    ```
- SMTP vs, HTTP
  - *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
    ```

- The Domain Name System, DNS

people: many identifiers:
- SSN, name, passport #

Internet hosts, routers:
- IP address (32 bit) - used for addressing datagrams
- "name", e.g., www.liverpool.ac.uk used by humans

Q: how to map between IP address and name, and vice versa ?

Domain Name System:

- *distributed database* implemented in hierarchy of many *name servers*
- *application-layer protocol:* hosts, name servers communicate to *resolve* names (address/name translation)
  - note: core Internet function, *implemented as application-layer protocol*
  - complexity at network's "edge"

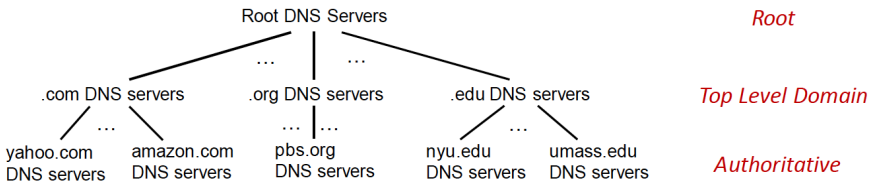
- Services, Structure

DNS services
- hostname to IP address translation
- host aliasing
  - canonical, alias names
- mail server aliasing
- load distribution
  - replicated Web servers: many IP addresses correspond to one name

Q: Why not centralize DNS?
- single point of failure
- traffic volume
- distant centralized database
- maintenance

A: doesn't scale!

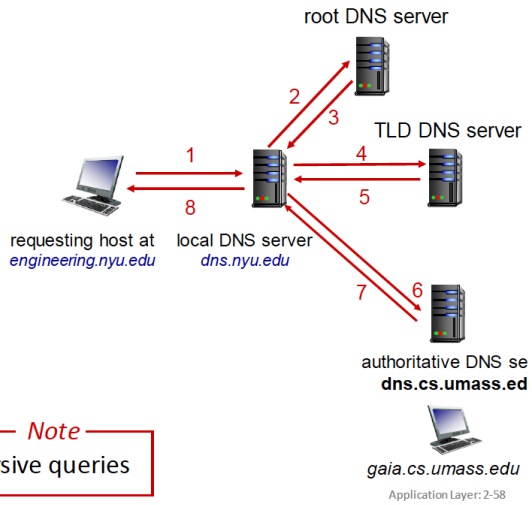
DNS: a distributed, hierarchical database



Client wants IP address for www.amazon.com; 1st approximation:
- client queries root server to find .com DNS server
- client queries .com DNS server to get amazon.com DNS server
- client queries amazon.com DNS server to get IP address for www.amazon.com

- 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: authoritative registry for .com, .net TLD
    - Educause: .edu TLD
  - **Authoritative DNS servers:**
    - organization's own DNS server(s), providing authoritative hostname to IP mappings for organization's named hosts
    - can be maintained by organization or service provider
- DNS name resolution: iterated query
  - Example: host at engineering.nyu.edu wants IP address for gaia.cs.umass.edu

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

    Note

    DNS allows also recursive queries
- 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

DNS records

DNS: distributed database storing resource records (RR)

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

type=A
- `name` is hostname
- `value` is IP address

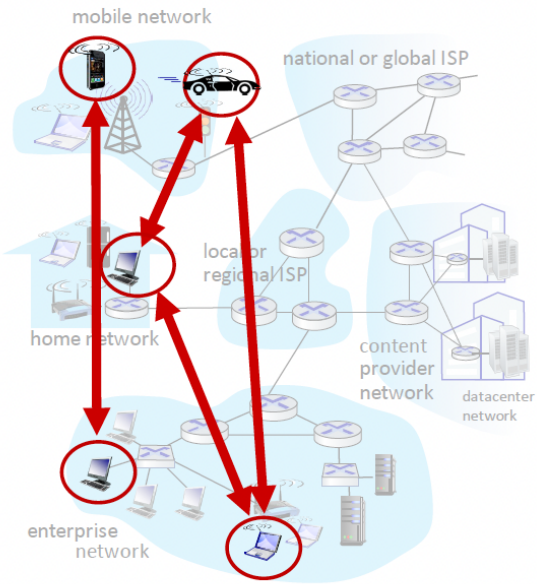
type=NS
- `name` is domain (e.g., foo.com)
- `value` is hostname of authoritative name server for this domain

type=CNAME
- `name` is alias name for some "canonical" (the real) name
- www.ibm.com is really servereast.backup2.ibm.com
- `value` is canonical name

type=MX
- `value` is name of mailserver associated with `name`

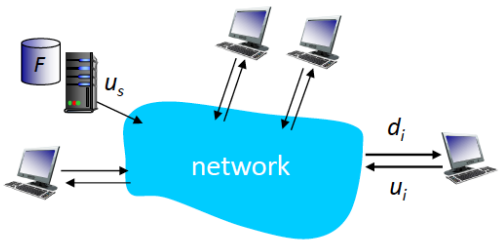
- P2P applications
  - P2P architechture

- *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, and new service demands
- peers are intermittently connected and change IP addresses
  - complex management
- examples: P2P file sharing (BitTorrent), streaming (KanKan), VoIP (Skype)



  - File Distribution Time
    - Client-Server

- *server transmission:* must sequentially send (upload) *N* file copies:
  - time to send one copy: F/u_s
  - time to send *N* copies: NF/u_s
- *client:* each client must download file copy
  - d_{min} = min client download rate
  - min client download time: F/d_{min}

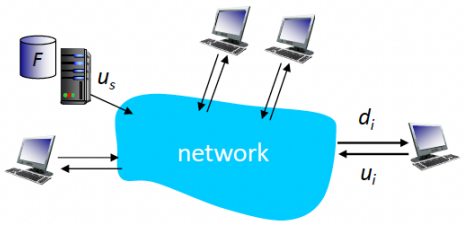


time to distribute F to N clients using client-server approach $D_{c\text{-}s} \geq max\{NF/u_s, F/d_{min}\}$

increases linearly in N

P2P

- *server transmission:* must upload at least one copy:
  - time to send one copy: F/u_s
- *client:* each client must download file copy
  - min client download time: F/d_{min}
- *clients:* as aggregate must download *NF* bits
  - max upload rate (limiting max download rate) is $u_s + \Sigma u_i$

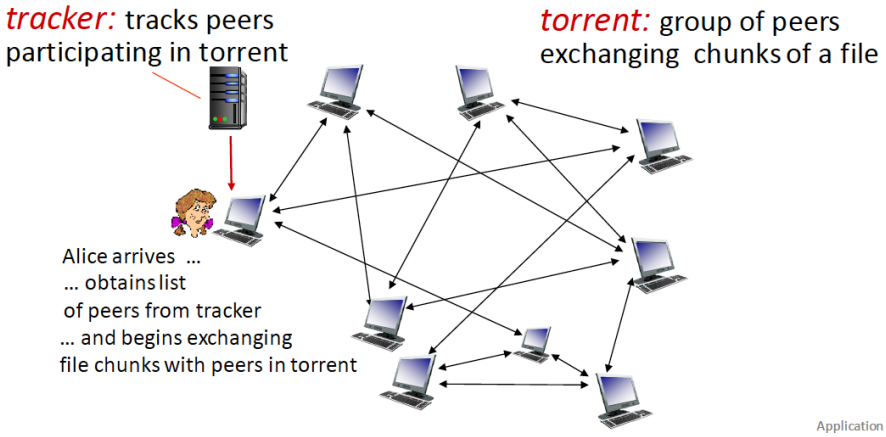


time to distribute F to N clients using P2P approach $D_{P2P} \geq max\{F/u_s, F/d_{min}, NF/(u_s + \Sigma u_i)\}$

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

P2P File Distribution: BitTorrent

- file divided into 256Kb chunks
- peers in torrent send/receive file chunks



Application

Requesting chunks:

- at any given time, different peers have different subsets of file chunks
- periodically, Alice asks each peer for list of chunks that they have
- Alice requests missing chunks from peers, rarest first

Sending chunks: tit-for-tat

- Alice sends chunks to those four peers currently sending her chunk *at highest rate*
  - other peers are choked by Alice (do not receive chunks from her)
  - re-evaluate top 4 every10 secs
- every 30 secs: randomly select another peer, starts sending chunks
  - "optimistically unchoke" this peer
  - newly chosen peer may join top 4

- socket programming with UDP and TCP
  - TCP

Client must contact server

- server process must first be running
- server must have 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

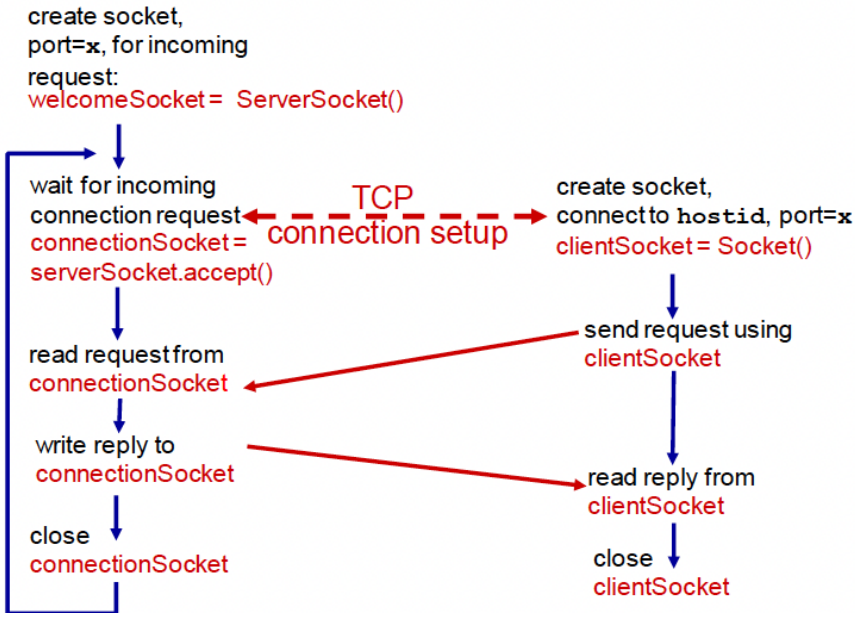
- when contacted by client, *server TCP creates new socket* for server process to communicate with that particular client
  - allows server to talk with multiple clients
  - source port numbers used to distinguish clients (more in Chap 3)

Application viewpoint

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

server (running on hostid)

client 



- UDP

Socket programming with UDP

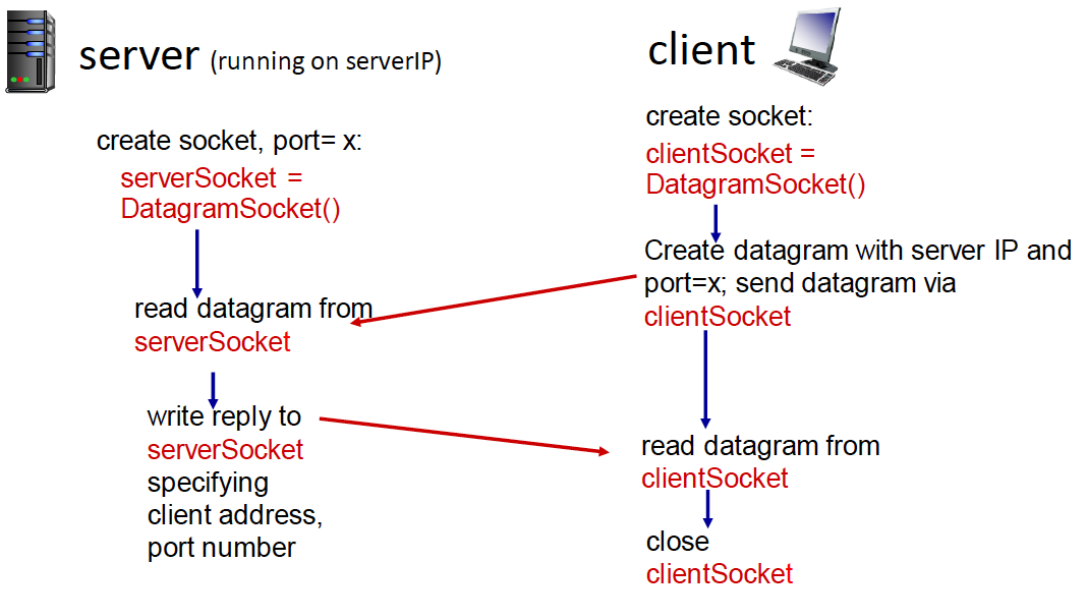
UDP: no "connection" between client & server

- no handshaking before sending data
- sender explicitly attaches IP destination address and port # to each packet
- receiver extracts sender IP address and port# from received packet

UDP: transmitted data may be lost or received out-of-order

Application viewpoint:

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



Summary

- 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
- socket programming: TCP, UDP sockets

- Most importantly: learned about *protocols*!
  - typical request/reply message exchange:
    - client requests info or service
    - server responds with data, status code
  - message formats:
    - *headers*: fields giving info about data
    - *data:* info(payload) being communicated

  important themes:
  - centralized vs. decentralized
  - stateless vs. stateful
  - scalability
  - reliable vs. unreliable message transfer
  - "complexity at network edge"

Review Questions

1. What information is used by a process running on one host to identify a process on another host?
2. Explain the differences between the client-server and peer-to-peer models for distributed computing.
3. For a communication session between a pair of processes, which process is the client and which is the server?
4. Why are there two types of protocols for transmission of email messages? Are these protocols push or pull protocols? Why? Draw a diagram to show the standard protocol or protocols used for sending email, and where they apply.

5. Suppose a webpage consists of many small pictures. You have the choice of two browsers, one using persistent HTTP and the other one non-persistent HTTP. Which one would you choose? Why?
6. Consider an HTTP client that wants to retrieve a Web document at a given URL. The IP address of the HTTP server is initially unknown. What transport and application layer protocols are needed in this scenario?
7. Why do HTTP and SMTP run on top of TCP rather than on UDP?

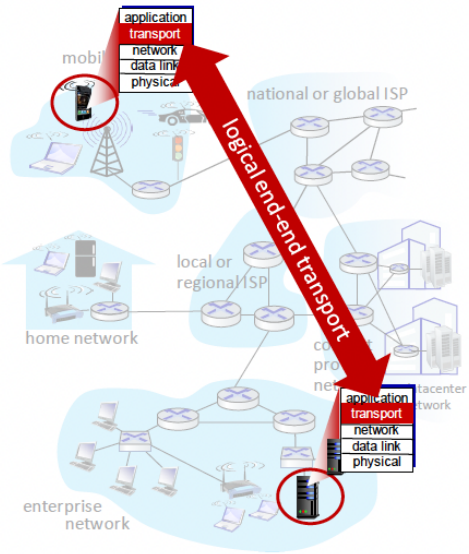
8. We discussed a UDP server and TCP server. The UDP server needed only one socket whereas the TCP server needed two sockets.
   - Why?
   - If the TCP server were to support n simultaneous connections, how many sockets would the TCP server need?

Transport Layer

Transport-Layer Services

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

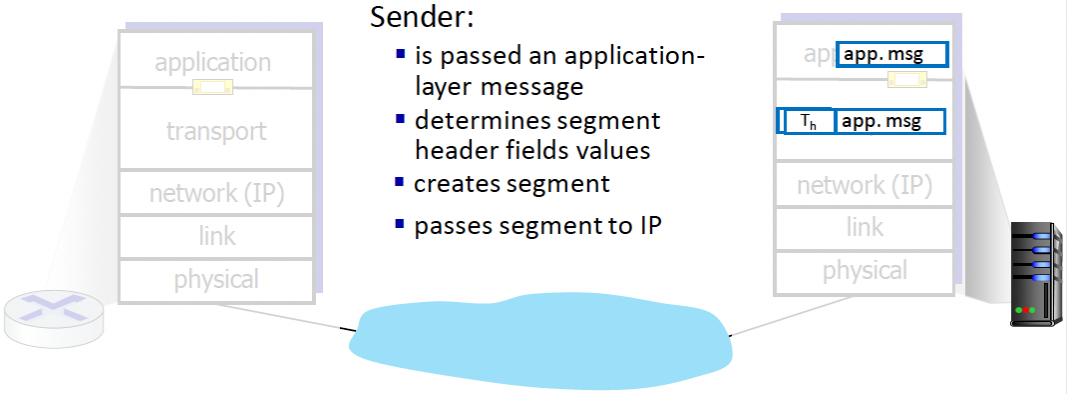


Transport vs. Network Layer Services and Protocols

- network layer: logical communication between *hosts*
  - *Similar to postal service that send a letter to a postal address*
- transport layer: extends network service to provide a logical communication between *processes*
  - relies on, enhances, network layer services

Sender Action

Transport Layer Actions

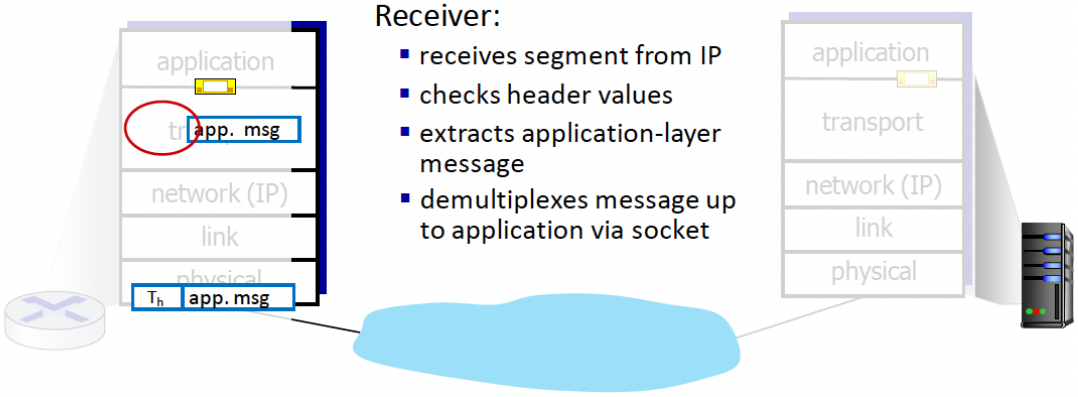


Sender:

- is passed an application-layer message
- determines segment header fields values
- creates segment
- passes segment to IP

- Receiver Action

Transport Layer Actions



Receiver:

- receives segment from IP
- checks header values
- extracts application-layer message
- demultiplexes message up to application via socket

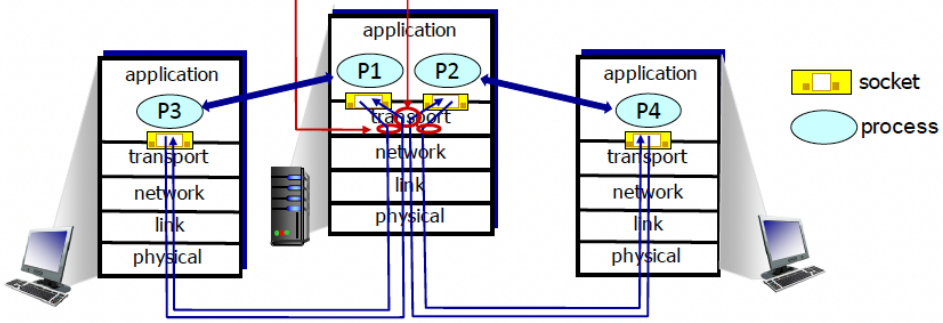
- Two Principal Internet Transport Protocols
  - TCP
    - **TCP:** Transmission Control Protocol
      - reliable, in-order delivery
      - congestion control
      - flow control
      - connection setup
  - UDP
    - **UDP:** User Datagram Protocol
      - unreliable, unordered delivery
      - no-frills extension of "best-effort" IP

- Multiplexing and Demultiplexing
  -

Multiplexing/demultiplexing

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

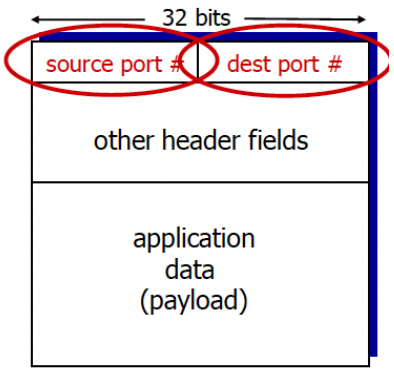
demultiplexing at receiver:
use header info to deliver received segments to correct socket



  - How work
    -

How demultiplexing works

- host receives 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 appropriate socket



TCP/UDP segment format

  - Connectionless Demultiplexing

Connectionless demultiplexing

Recall:

- when creating socket, must specify *host-local* port #:

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

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

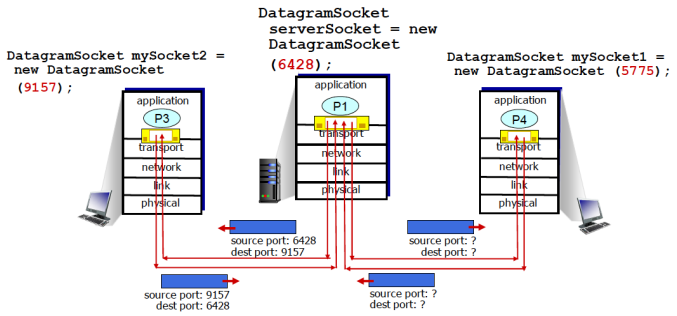
when receiving host receives *UDP* segment:

- 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

- Example

Connectionless demultiplexing: an example

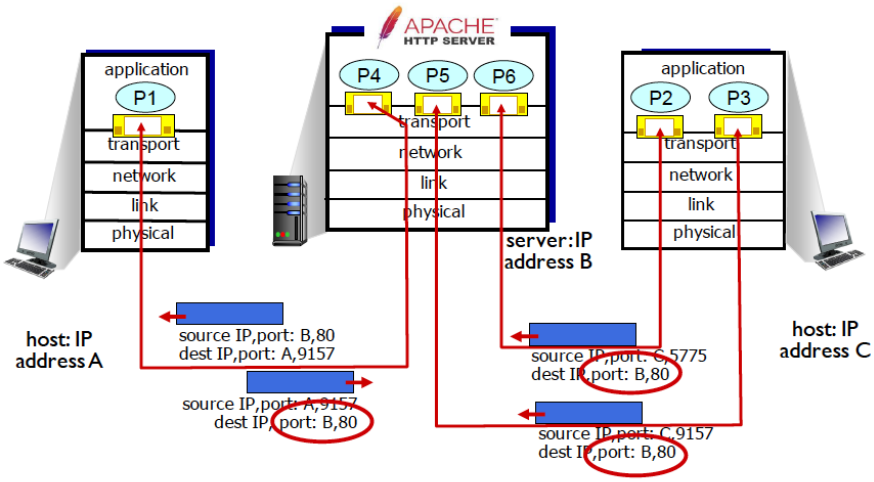


- Connection-Oriented Demultiplexing

- TCP socket identified by 4-tuple:
  - source IP address
  - source port number
  - dest IP address
  - dest port number
- demux: receiver uses *all four values (4-tuple)* to direct segment to appropriate socket
- server may support many simultaneous TCP sockets:
  - each socket identified by its own 4-tuple
  - each socket associated with a different connecting client

- Example

Connection-oriented demultiplexing: example



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

- Summary

  - Multiplexing, demultiplexing: based on segment, datagram header field values
  - **UDP:** demultiplexing using destination port number (only)
  - **TCP:** demultiplexing using 4-tuple: source and destination IP addresses, and port numbers

- Connectionless Transport: UDP

UDP: User Datagram Protocol

- "no frills," "bare bones" Internet transport protocol
- "best effort" service, UDP segments may be:
  - lost
  - delivered out-of-order
- *connectionless:*
  - no handshaking between sender, receiver
  - each UDP segment handled independently of others

Why is there a UDP?

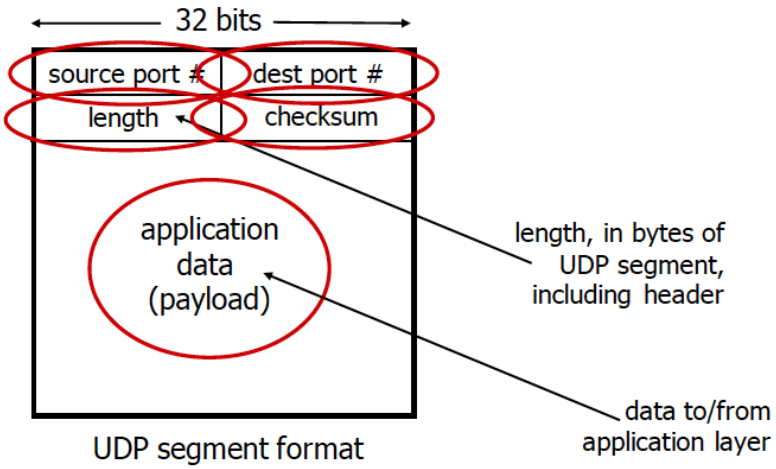
- no connection establishment (which can add RTT delay)
- simple: no connection state at sender, receiver
- small header size
- no congestion control
  - UDP can blast away as fast as desired!
  - can function in the face of congestion

UDP: User Datagram Protocol

- UDP use:
  - 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

UDP Header

UDP segment header



- UDP checkSum

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

	1st number	2nd number	sum
Transmitted:	5	6	11
Received:	4	6	11

receiver-computed checksum ≠ sender-computed checksum (as received)

- Internet checkSum

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

sender:

- treat contents of UDP segment (including 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 equals checksum field value:
  - not equal - error detected
  - equal - no error detected. *But maybe errors nonetheless?* More later ....

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

- weak protection

Internet checksum: weak protection!

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

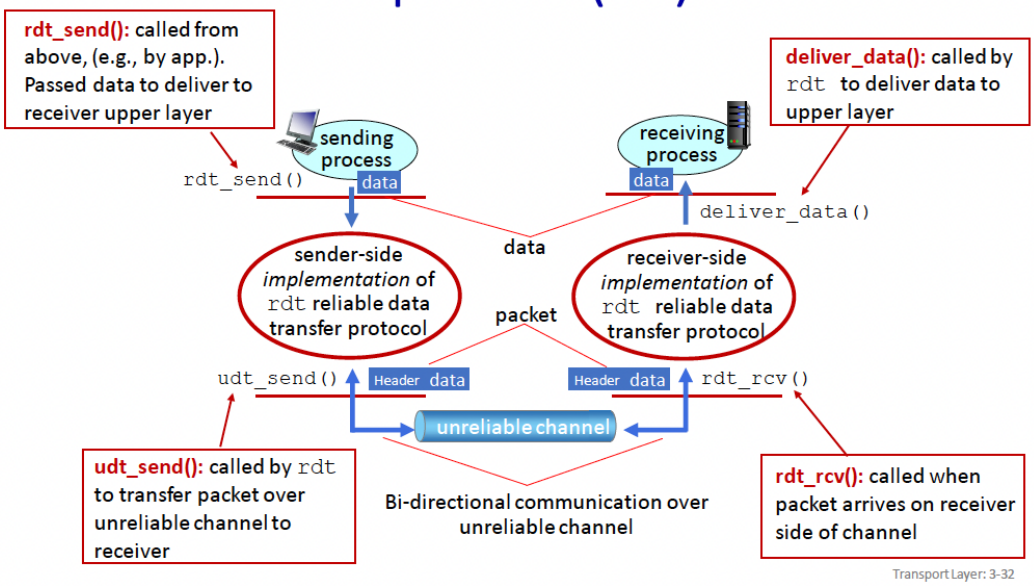
0 1
1 0

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

- Summary UDP
  - ## Summary: UDP

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

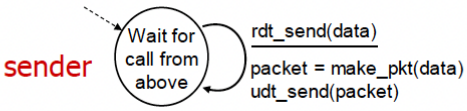
- Principles of reliable data transfer
  - Stop and Wait
    - ## Reliable data transfer protocol (rdt): interfaces

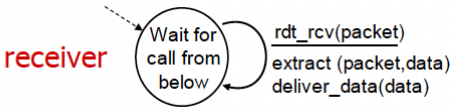
      

  - rdt1.0

rdt1.0: reliable transfer over a reliable channel

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





- rdt2.0

rdt2.0: channel with bit errors

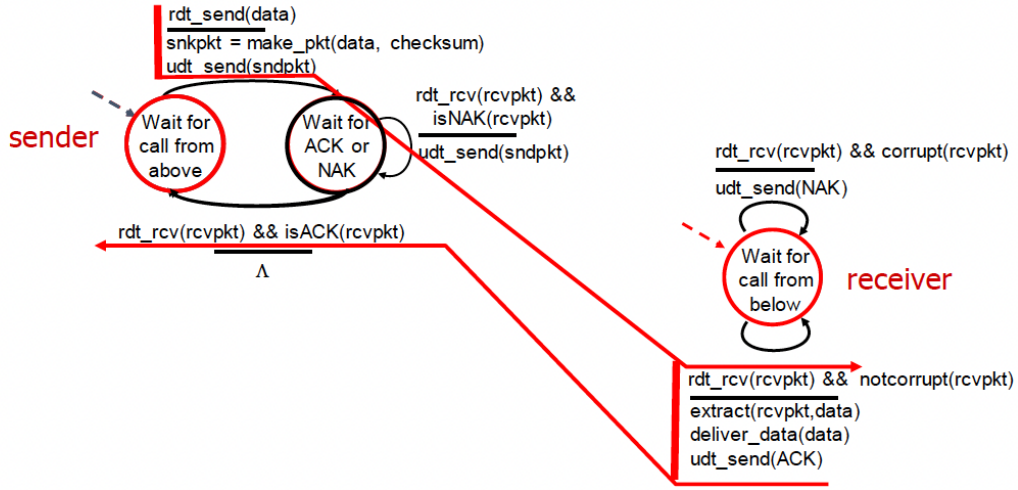
- underlying channel may flip bits in packet
  - checksum to detect bit errors
- *the* question: how to recover from errors?
  - *acknowledgements (ACKs):* receiver explicitly tells sender that pkt received OK
  - *negative acknowledgements (NAKs):* receiver explicitly tells sender that pkt had errors
  - sender *retransmits* pkt on receipt of NAK

stop and wait

sender sends one packet, then waits for receiver response

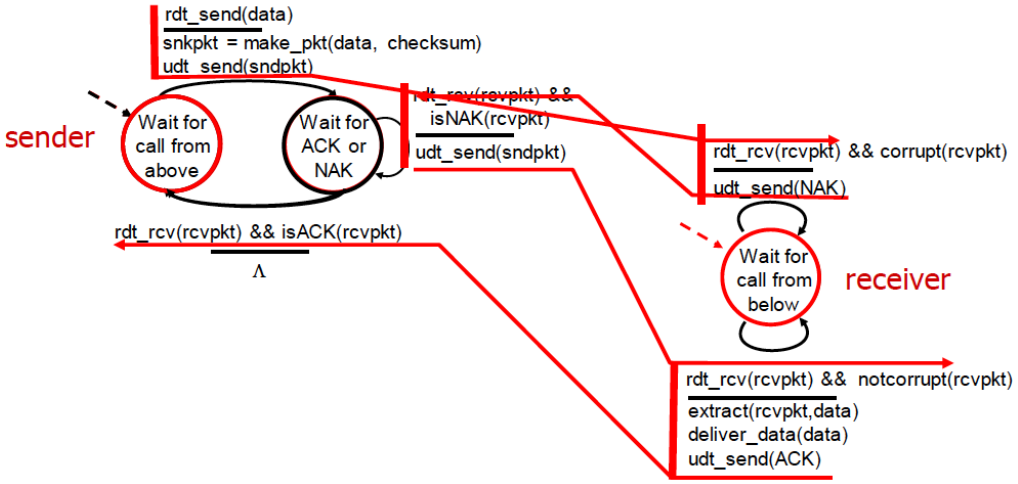
- operation with no errors

rdt2.0: operation with no errors



- corrupted packet scenario

rdt2.0: corrupted packet scenario



- fatal flaw

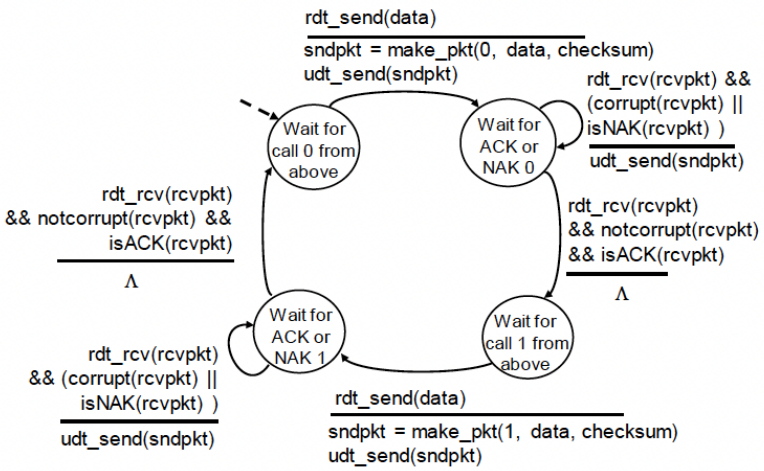
-

  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 pkt if ACK/NAK corrupted
  - sender adds *sequence number* to each pkt
  - receiver discards (doesn't deliver up) duplicate pkt

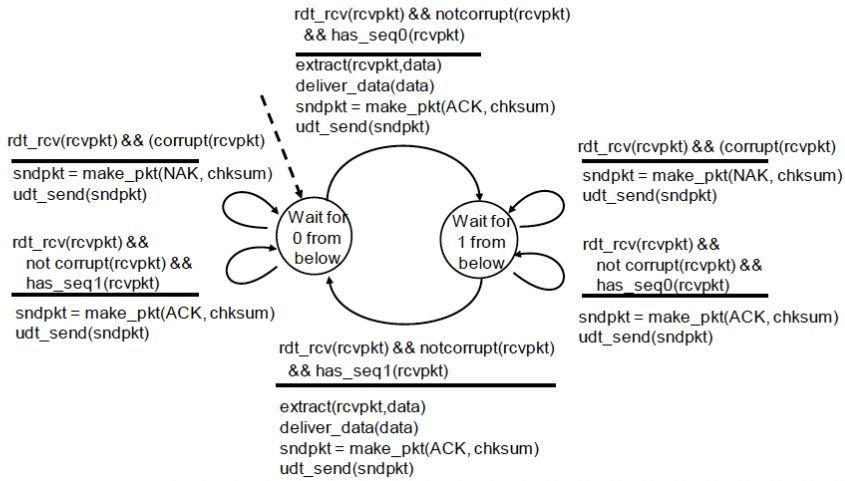
- rdt2.1
  - sender
    -

      # rdt2.1: sender, handling garbled ACK/NAKs

  - receiver

rdt2.1: receiver, handling garbled ACK/NAKs



- discussion

rdt2.1: discussion

sender:

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

receiver:

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

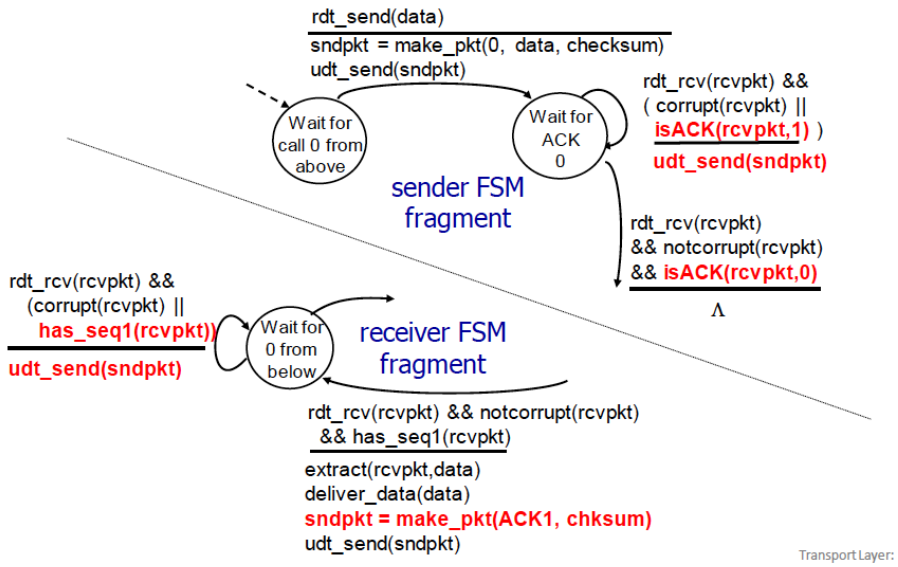
- rdt2.2

rdt2.2: a NAK-free protocol

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

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

rdt2.2: sender, receiver fragments



- rdt3.0

rdt3.0: channels with errors *and* loss

New channel assumption: underlying channel can also *lose* packets (data, ACKs)

- checksum, sequence #s, ACKs, retransmissions will be of help … but not quite enough

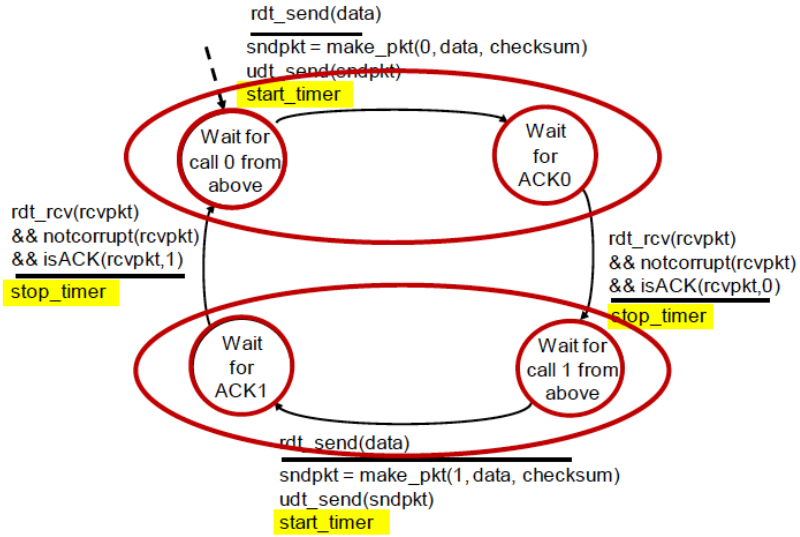
Q: How do *humans* handle lost sender-to-receiver words in conversation?

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

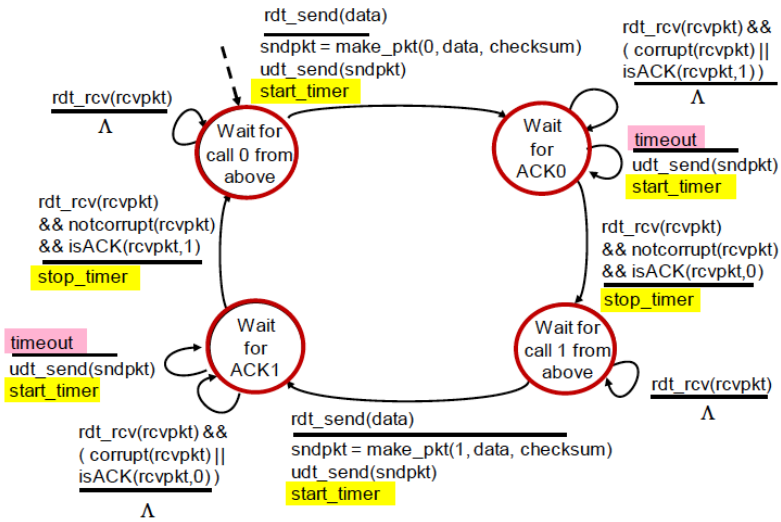
- retransmits if no ACK received in this time
- if pkt (or ACK) just delayed (not lost):
  - retransmission will be duplicate, but seq #s already handles this!
  - receiver must specify seq # of packet being ACKed
- use countdown timer to interrupt after "reasonable" amount of time

timeout

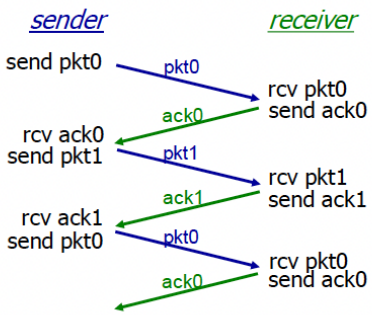
rdt3.0 sender



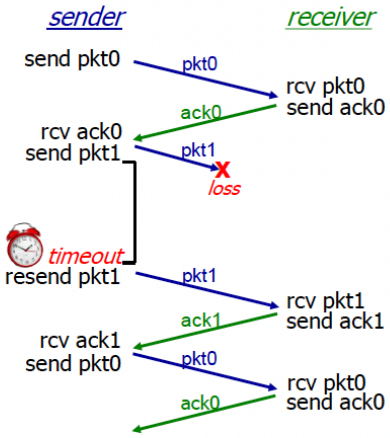
rdt3.0 sender



rdt3.0 in action

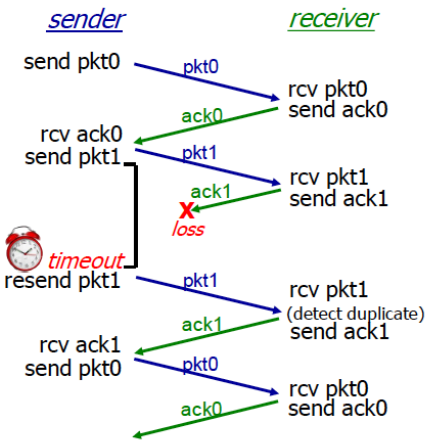


(a) no loss

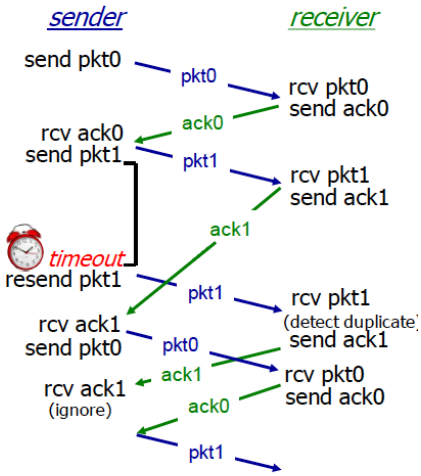


(b) packet loss

rdt3.0 in action



(c) ACK loss



(d) premature timeout/ delayed ACK

- Performance of rdt3.0

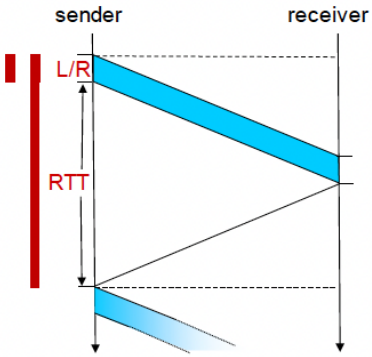
Performance of rdt3.0 (stop-and-wait)

- U_{sender}: *utilization* – fraction of time sender busy sending
- example: 1 Gbps link, 15 ms prop. delay, 8000 bit packet
  - time to transmit packet into channel:

$$D_{trans} = \frac{L}{R} = \frac{8000 \text{ bits}}{10^9 \text{ bits/sec}} = 8 \text{ microsecs}$$

rdt3.0: stop-and-wait operation

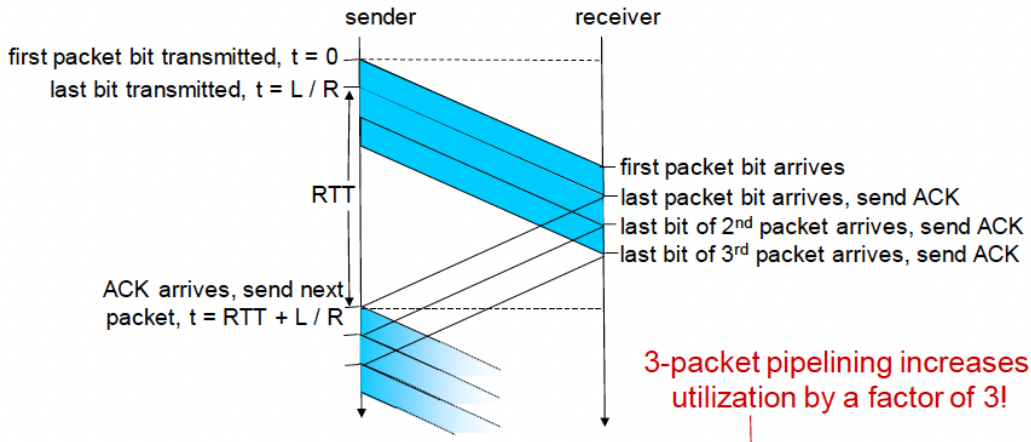
$$U_{sender} = \frac{L / R}{RTT + L / R} = \frac{.008}{30.008} = 0.00027$$



- rdt 3.0 protocol performance stinks!
- Protocol limits performance of underlying infrastructure (channel)

▼ Pipelined Protocols

Pipelining: increased utilization



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

- Go-Back-N
  - Sender

Go-Back-N: sender

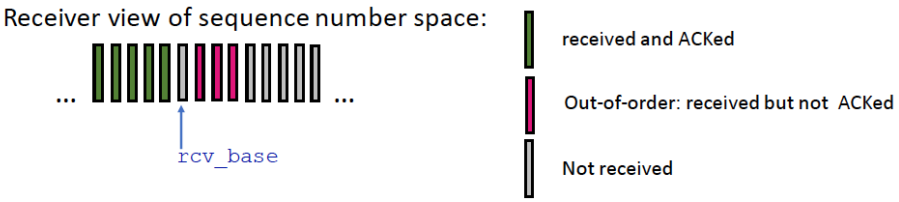
- sender: "window" of up to N, consecutive transmitted but unACKed pkts
  - k-bit seq # in pkt header

- *cumulative ACK:* ACK(*n*): ACKs all packets up to, including seq # *n*
  - on receiving ACK(*n*): move window forward to begin at *n+1*
- timer for oldest in-flight packet
- *timeout(n):* retransmit packet n and all higher seq # packets in window

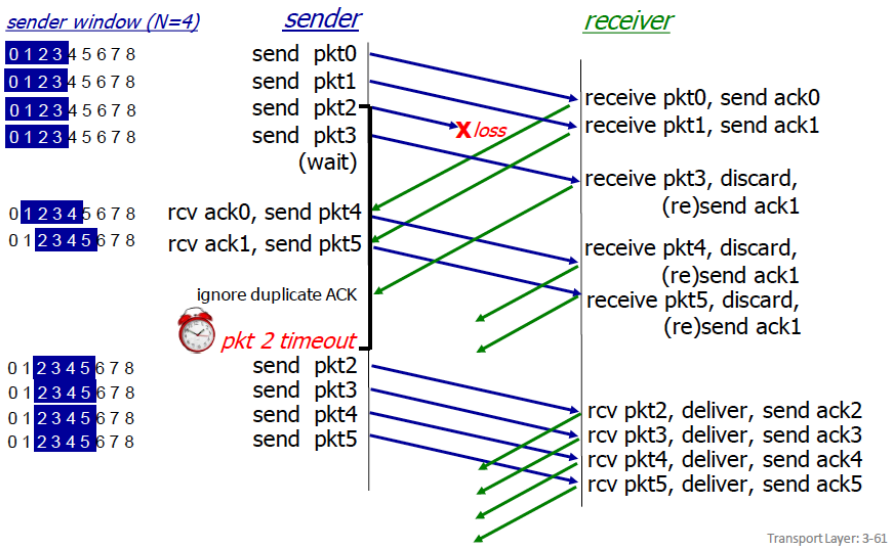
  - Receiver

Go-Back-N: receiver

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

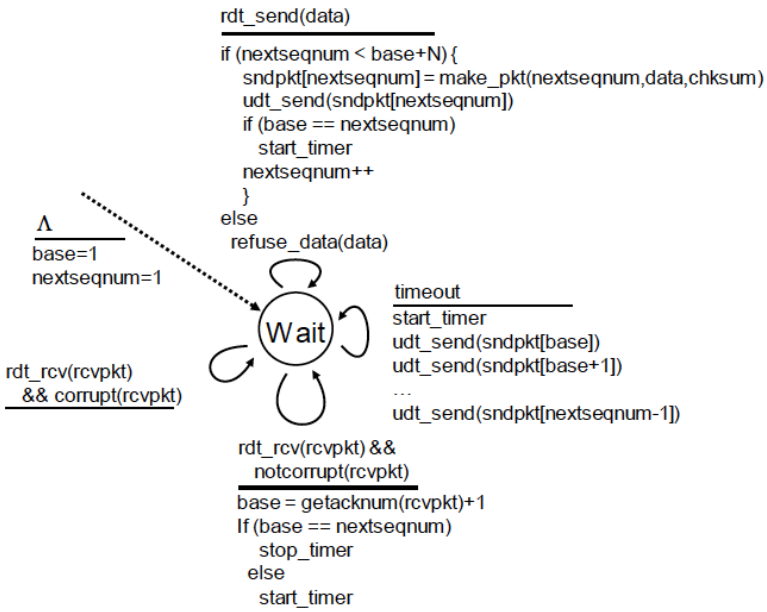


Go-Back-N in action

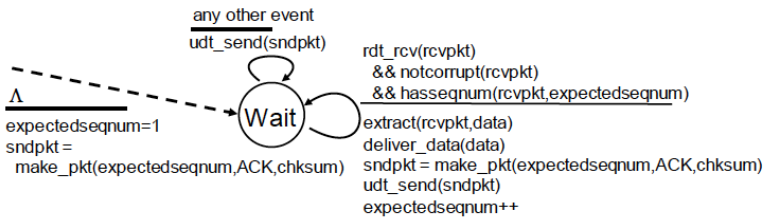


- Additional

Go-Back-N: sender extended FSM



Go-Back-N: receiver extended FSM



ACK-only: always send ACK for correctly-received packet with highest *in-order* seq #

- may generate duplicate ACKs
- need only remember **`expectedseqnum`**

- out-of-order packet:
  - discard (don't buffer): *no receiver buffering!*
  - re-ACK pkt with highest in-order seq #

- Selective Repeat

Selective repeat

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

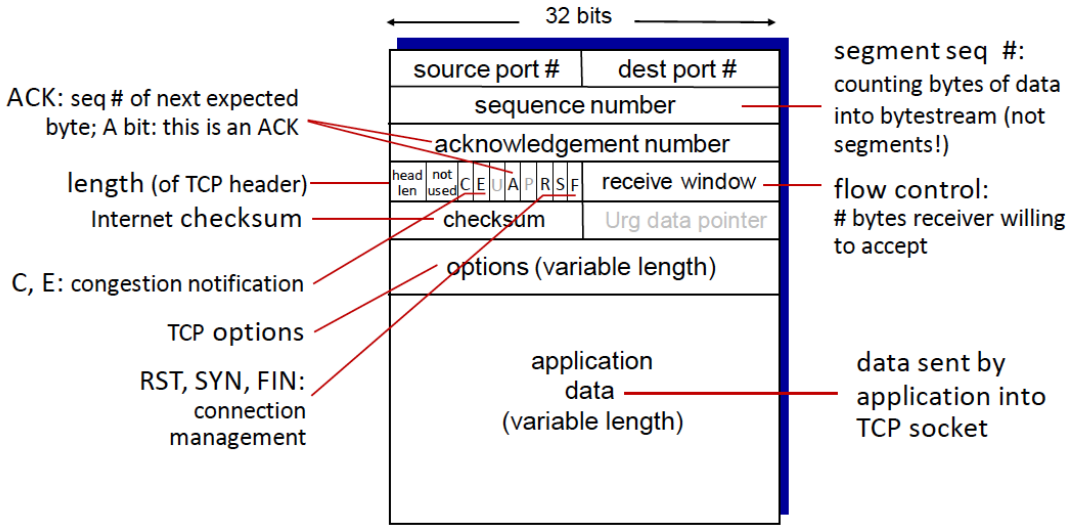
- Connection-Oriented Transport: TCP
  - Overview

TCP: overview RFCs: 793, 1122, 2018, 5681, 7323

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

  - TCP segment structure

TCP segment structure



- Sequence #, ACKS

TCP sequence numbers, ACKs

Sequence numbers:

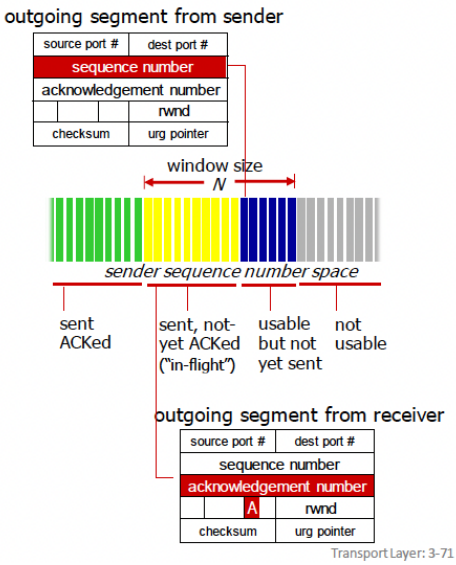
- byte stream "number" of first byte in segment's data

Acknowledgements:

- seq # of next byte expected from other side
- cumulative ACK

Q: how receiver handles out-of-order segments

- *A:* TCP spec doesn't say, - up to implementor



- TCP round trip time, timeout

TCP round trip time, timeout

Q: how to set TCP timeout value?

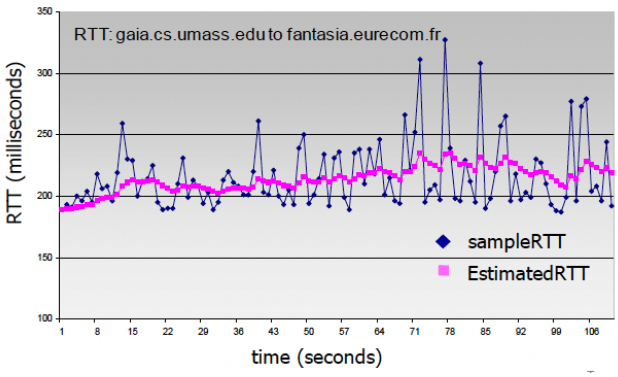
- longer than RTT, but RTT varies!
- *too short:* premature timeout, unnecessary retransmissions
- *too long:* slow reaction to segment loss

Q: how to estimate RTT?

- `SampleRTT`: measured time from segment transmission until ACK receipt
  - ignore retransmissions
- `SampleRTT` will vary, want estimated RTT "smoother"
  - average several *recent* measurements, not just current `SampleRTT`

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

- exponential weighted moving average (EWMA)
- influence of past sample decreases exponentially fast
- typical value: α = 0.125



- timeout interval: **`EstimatedRTT`** plus "safety margin"
  - large variation in **`EstimatedRTT`**: want a larger 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)

TCP reliable data transfer

- TCP creates rdt service on top of IP's unreliable service
  - pipelined segments
  - cumulative acks
  - single retransmission timer
- retransmissions triggered by:
  - timeout events
  - duplicate acks

let's initially consider simplified TCP sender:
- ignore duplicate acks
- ignore flow control, congestion control

- TCP Sender

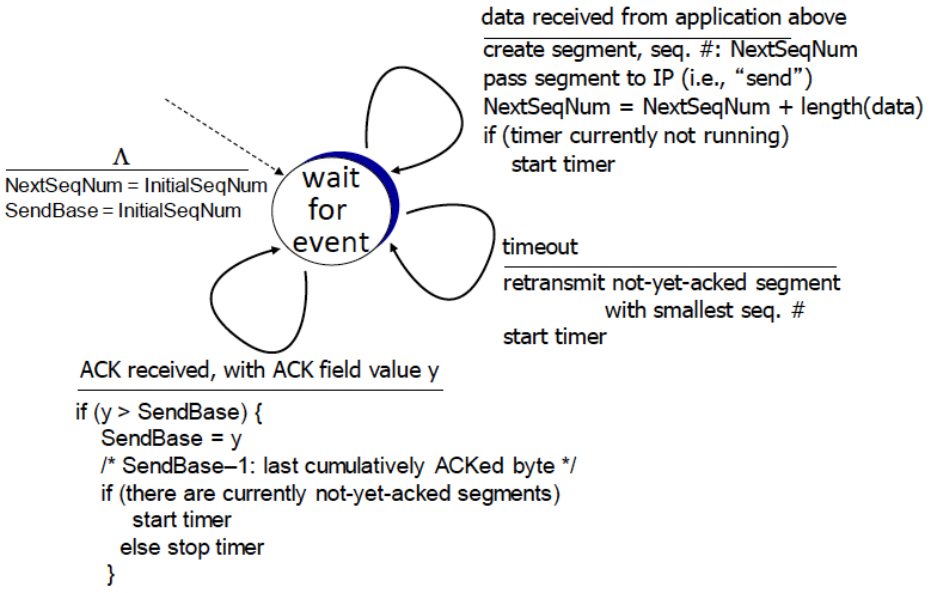
TCP Sender (simplified)

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

event: timeout
- retransmit segment that caused timeout
- restart timer

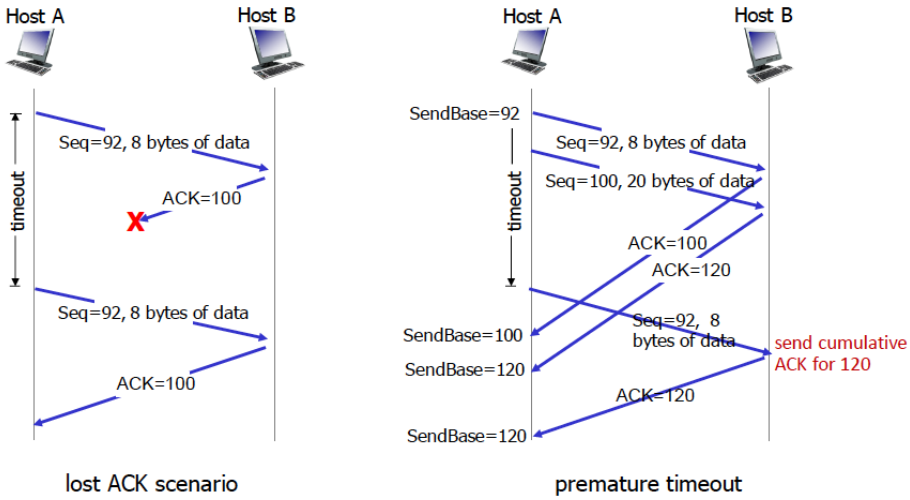
event: ACK received
- if ACK acknowledges previously unACKed segments
  - update what is known to be ACKed
  - start timer if there are still unACKed segments

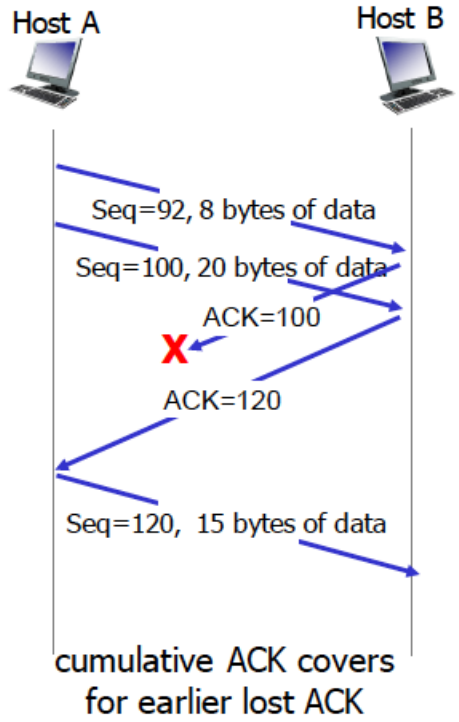
TCP sender (simplified)



- Retransmission Scenarios

TCP: retransmission scenarios





- TCP Receiver

TCP Receiver: ACK generation [RFC 5681]

Event at receiver	*TCP receiver action*
arrival of in-order segment with expected seq #. All data up to expected seq # already ACKed	delayed ACK. Wait up to 500ms for next segment. If no next segment, send ACK
arrival of in-order segment with expected seq #. One other segment has ACK pending	immediately send single cumulative ACK, ACKing both in-order segments
arrival of out-of-order segment higher-than-expect seq. # . Gap detected	immediately send *duplicate ACK*, indicating seq. # of next expected byte
arrival of segment that partially or completely fills gap	immediate send ACK, provided that segment starts at lower end of gap

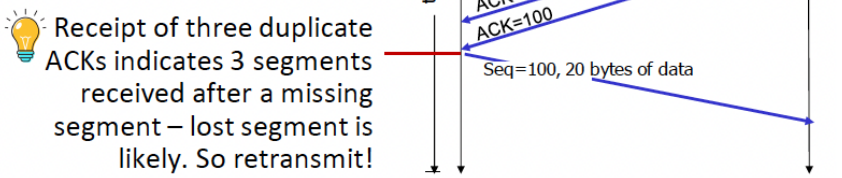
- TCP fast retransmit

TCP fast retransmit

TCP fast retransmit

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

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



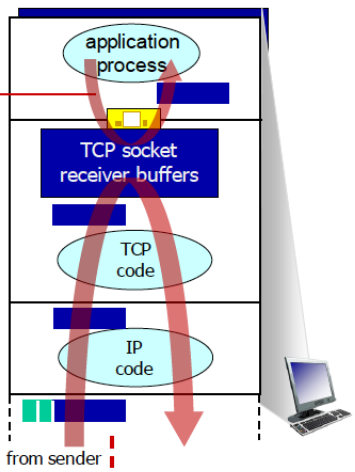
Receipt of three duplicate ACKs indicates 3 segments received after a missing segment – lost segment is likely. So retransmit!

- TCP Flow Control

TCP flow control

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

Application removing data from TCP socket buffers



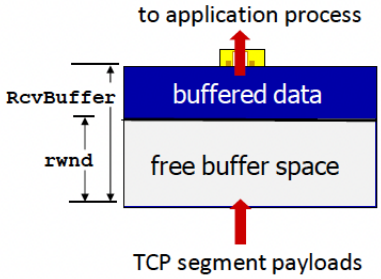
receiver protocol stack

flow control

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

TCP flow control

- TCP receiver "advertises" free buffer space in **`rwnd`** field in TCP header
  - **`RcvBuffer`** size set via socket options (typical default is 4096 bytes)
  - many operating systems autoadjust **`RcvBuffer`**
- sender limits amount of unACKed ("in-flight") data to received **`rwnd`**
- guarantees receive buffer will not overflow



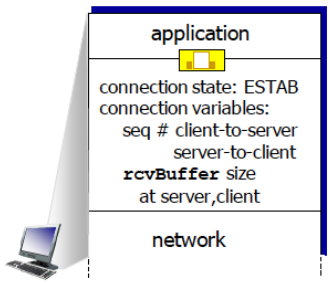
TCP receiver-side buffering

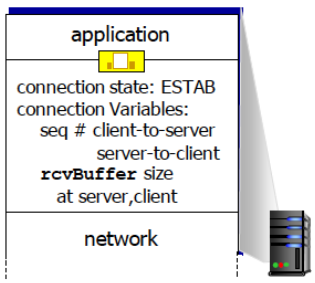
- TCP connection management

TCP connection management

before exchanging data, sender/receiver "handshake":

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



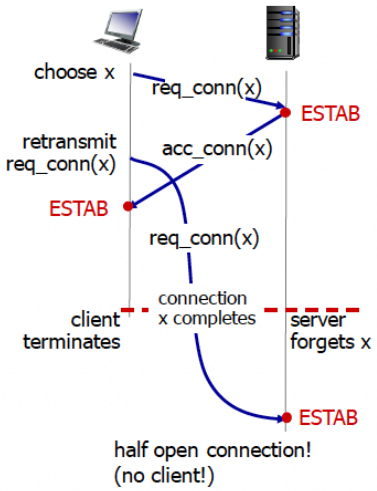


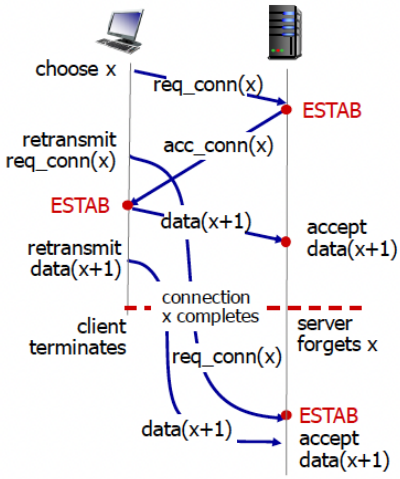

```
Socket clientSocket =
  newSocket("hostname",
            "port number");
```

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


- 2-way handshake

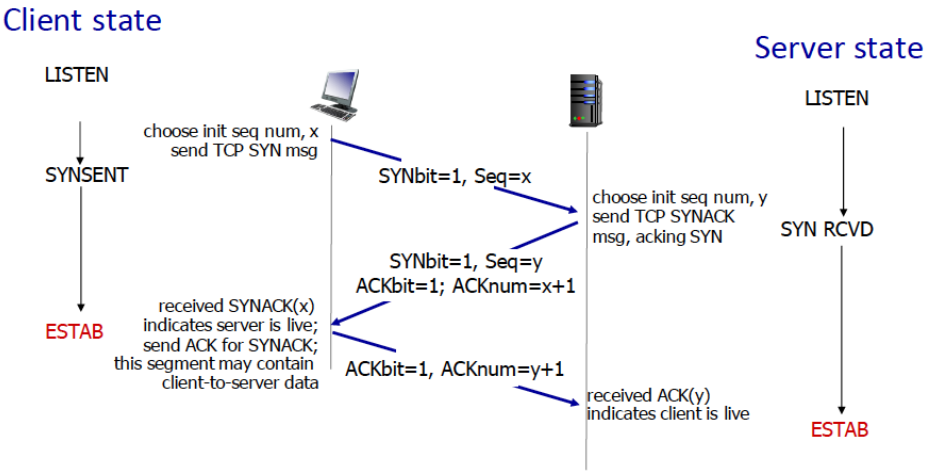
2-way handshake scenarios





- 3-way handshake

TCP 3-way handshake

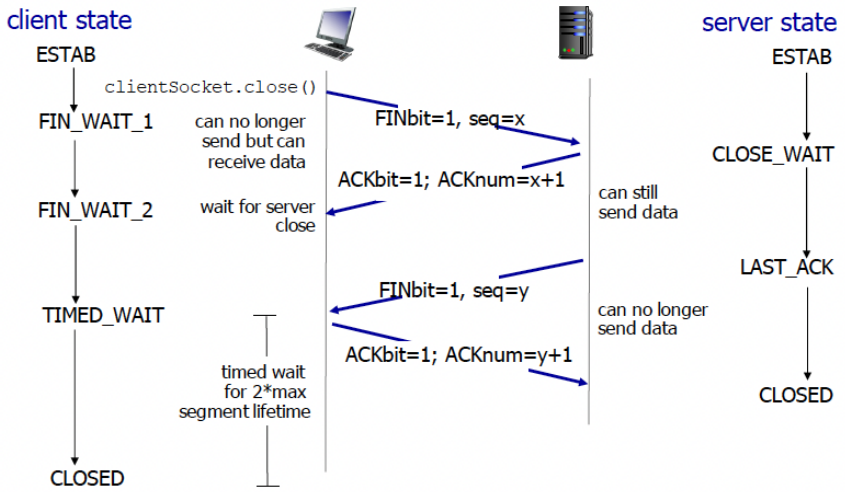


- closing TCP connection

Closing a TCP 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 exchanges can be handled

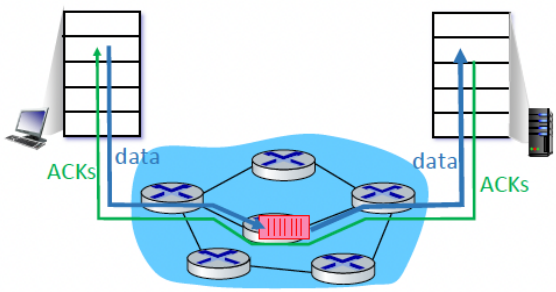
Closing a TCP connection



- Principles of Congestion Control

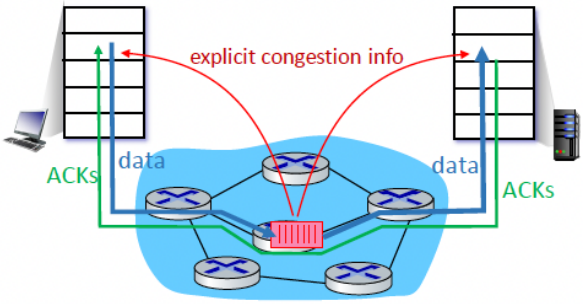
End-end congestion control:

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



Network-assisted congestion control:

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



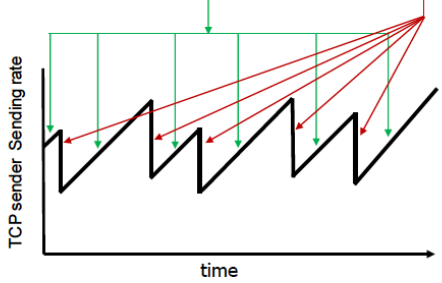
- TCP congestion control
  - AIMD

TCP congestion control: AIMD

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

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

Multiplicative Decrease
cut sending rate in half at each loss event



AIMD sawtooth behavior: *probing* for bandwidth

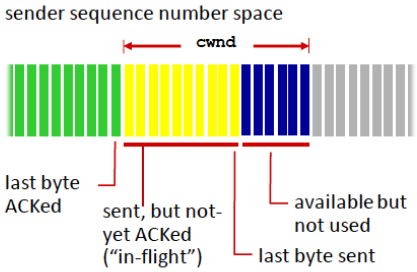
TCP AIMD: more

Multiplicative decrease detail: sending rate is
- Cut in half on loss detected by triple duplicate ACK (TCP Reno)
- Cut to 1 MSS (maximum segment size) when loss detected by timeout (TCP Tahoe)

Why AIMD?
- AIMD – a distributed, asynchronous algorithm – has been shown to:
  - optimize congested flow rates network wide!
  - have desirable stability properties

- Details

TCP congestion control: details



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

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

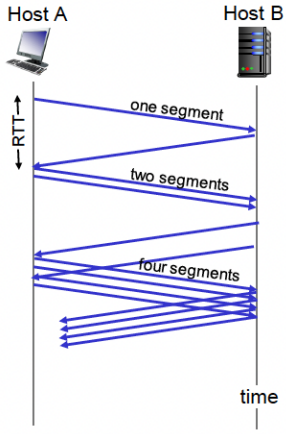
- TCP sender limits transmission:

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

- Slow Start to congestion avoidance

TCP slow start

- when connection begins, increase rate exponentially until first loss event:
  - initially **cwnd** = 1 MSS
  - double **cwnd** every RTT
  - done by incrementing **cwnd** for every ACK received

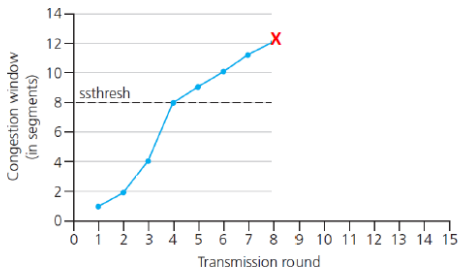


- *summary:* initial rate is slow, but ramps up exponentially fast

TCP: from slow start to congestion avoidance

Q: when should the exponential increase switch to linear?

A: when **cwnd** gets to 1/2 of its value before timeout.



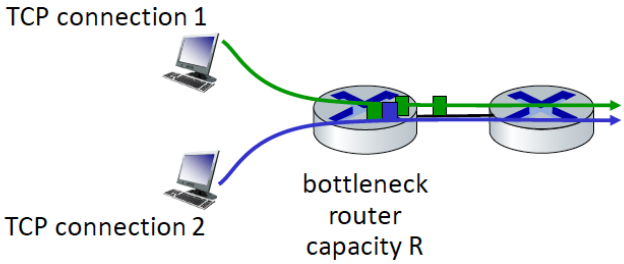
Implementation:

- variable **ssthresh**
- on loss event, **ssthresh** is set to 1/2 of **cwnd** just before loss event

- TCP fairness

TCP fairness

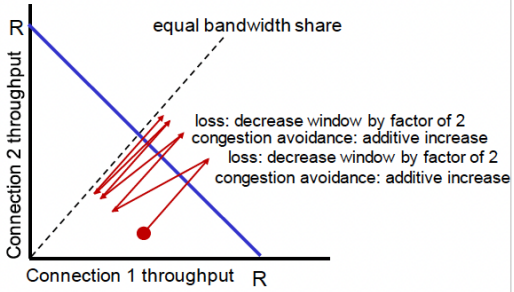
Fairness goal: if *K* TCP sessions share same bottleneck link of bandwidth *R*, each should have average rate of *R/K*



Q: is TCP Fair?

Example: two competing TCP sessions:

- additive increase gives slope of 1, as throughput increases
- multiplicative decrease decreases throughput proportionally



Is TCP fair?

A: Yes, under idealized assumptions:

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

Fairness: must all network apps be "fair"?

Fairness and UDP

- multimedia apps often do not use TCP
  - do not want rate throttled by congestion control
- instead use UDP:
  - send audio/video at constant rate, tolerate packet loss

Fairness, parallel TCP connections

- application can open *multiple* parallel connections between two hosts
- web browsers do this , e.g., link of rate R with 9 existing connections:
  - new app asks for 1 TCP, gets rate R/10
  - new app asks for 9 TCPs, gets R/2

Evolution of transport-layer functionality

Evolving transport-layer functionality

- TCP, UDP: principal transport protocols for 40 years
- different "flavors" of TCP developed, for specific scenarios:

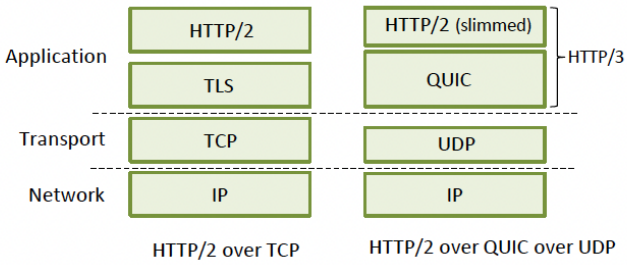
Scenario	Challenges
Long, fat pipes (large data transfers)	Many packets "in flight"; loss shuts down pipeline
Wireless networks	Loss due to noisy wireless links, mobility; TCP treat this as congestion loss
Long-delay links	Extremely long RTTs
Data center networks	Latency sensitive
Background traffic flows	Low priority, "background" TCP flows

- moving transport–layer functions to application layer, on top of UDP
  - HTTP/3: QUIC

QUIC

QUIC: Quick UDP Internet Connections

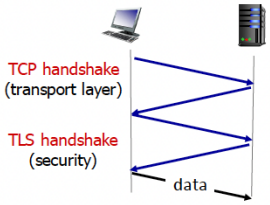
- application-layer protocol, on top of UDP
  - increase performance of HTTP
  - deployed on many Google servers, apps (Chrome, mobile YouTube app)



- adopts approaches we've studied in this chapter for connection establishment, error control, congestion control
  - **error and congestion control:** "Readers familiar with TCP's loss detection and congestion control will find algorithms here that parallel well-known TCP ones." [from QUIC specification]
  - **connection establishment:** reliability, congestion control, authentication, encryption, state established in one RTT
- multiple application-level "streams" multiplexed over single QUIC connection
  - separate reliable data transfer, security
  - common congestion control

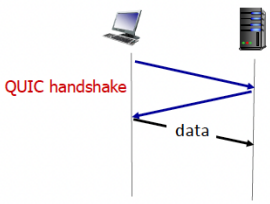
QUIC Connection Establishment

QUIC: Connection establishment



TCP (reliability, congestion control state) + TLS (authentication, crypto state)

- 2 serial handshakes



QUIC: reliability, congestion control, authentication, crypto state

- 1 handshake

Summary

Chapter 3: summary

- principles behind transport layer services:
  - multiplexing, demultiplexing
  - reliable data transfer
  - flow control
  - congestion control
- instantiation, implementation in the Internet
  - UDP
  - TCP

Transport Layer: Main Lessons

- Logical Communications between:
  - App Layer: End-users
  - Transport: Processes
  - Network: Hosts
- Transport Layer Protocols
  - Produces Segment = Data from Application + Header
  - Hands over to Network Layer (unreliable IP)
  - Header serves to multiplex demultiplex
- Two main protocols:
  - TCP: reliable packet delivery
  - UDP: unreliable packet delivery (blasting away)

Features of TCP

- Connection- Management (ie, a handshake & tear-down at end)
- Use of ACKs (Cumacks)
  - Cumulative ACKs
- Sequence numbers for packets
- Pipelining
- Checksum
- Flow control
  - Bob tells Alice current buffer window size
- Congestion control
  - End-to-end congestion control
  - ie, Alice infers network congestion from acks, timeouts, resends
  - Then Alice throttles down or up the flow of packets.

TCP

- Advantages
  - Reliability
  - Respectful of receiver (Bob) – flow control
  - Flexibility w.r.t. receiver
  - Respectful of other users in the Internet
    – Congestion control
- Disadvantages
  - Slower
  - Heavier overhead

UDP

- Advantages
  - Fast
  - Small header → low overheads (comms and processing)
- Disadvantages
  - Unreliability

▾ Review Questions

Review Questions

1. Describe why an application developer might choose to run an application over UDP rather than TCP.
2. Is it possible for an application to enjoy reliable data transfer even when the application runs over UDP? If so, how?
3. Suppose you have the following 8-bit bytes: 01010101, 01110000, 01001100.
   What is the 1s complement sum of these 8bit-bytes?
   Why do UDP and TCP take the 1s complement?
4. In our rdt protocol, why did we need to introduce timers?
5. … sequence numbers?

6. The Internet has been described as an "unreliable" communications medium.
   a) What characteristics of Internet communications lead to this description?
   b) Which protocol was developed to deal with these undesirable features?
   c) Explain how the features of this protocol address the characteristics you listed in (a).

7. True or false?
   a) Suppose host A is sending a large file to host B over a TCP connection. If the sequence number for a segment is m, then the sequence number of the subsequent segment is necessarily m+1.
   b) The number of unacknowledged bytes that A sends cannot exceed the receive buffer.
   c) The size of the TCP receive window (rwnd) never changes throughout the duration of the connection.
   d) The TCP segment has a field in its header for rwnd.
   e) Suppose host A sends one segment with sequence number 38 and 4 bytes of data over a TCP connection to host B. In the same segment the ack number is necessarily 42.

- **Network Layer: Data Plane**
  - Overview

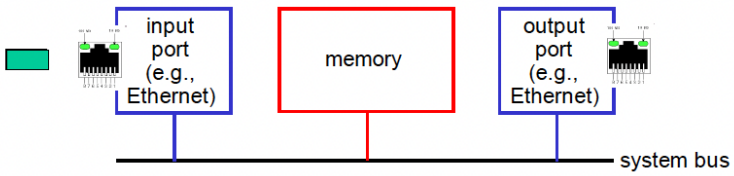
- transport segment from sending to receiving host
  - sender: encapsulates segments into datagrams, passes to link layer
  - receiver: delivers segments to transport layer protocol
- network layer protocols in *every Internet device*: hosts, routers
- routers:
  - examines header fields in all IP datagrams passing through it
  - moves datagrams from input ports to output ports to transfer datagrams along end-end path

- What's inside a router?
  - Router Architecture
    - Switching Fabrics
      - via memory

Switching via memory

first generation routers:

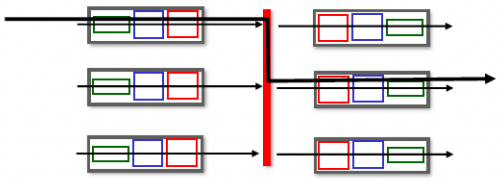
- traditional computers with switching under direct control of CPU
- packet copied to system's memory
- speed limited by memory bandwidth (2 bus crossings per datagram)



- via a bus

Switching via a bus

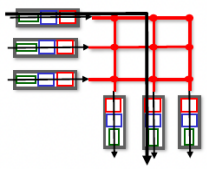
- datagram from input port memory to output port memory via a shared bus
- *bus contention:* switching speed limited by bus bandwidth
- 32 Gbps bus, Cisco 5600: sufficient speed for access routers



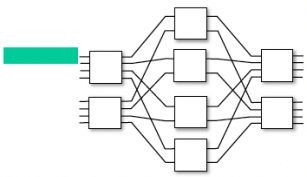
- via interconnection network

Switching via interconnection network

- Crossbar, Clos networks, other interconnection nets initially developed to connect processors in multiprocessor
- multistage switch: *nxn* switch from multiple stages of smaller switches


3x3 crossbar

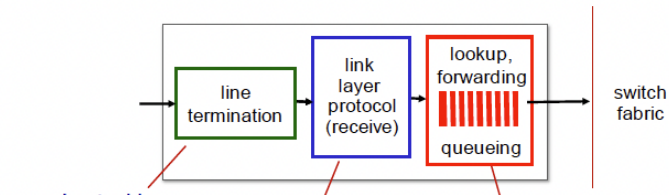
- exploiting parallelism:
  - fragment datagram into fixed length cells on entry
  - switch cells through the fabric, reassemble datagram at exit


8x8 multistage switch built from smaller-sized switches

- Switching Rate

- Input ports

Input port functions



physical layer: bit-level reception

link layer: e.g., Ethernet

decentralized switching:

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

- Destination-Based Forwarding
  - Longest Prefix Matching
    -

Longest prefix matching

longest prefix match

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

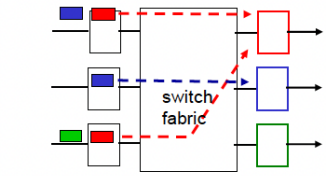
examples:

11001000 00010111 00010110 10100001 which interface?

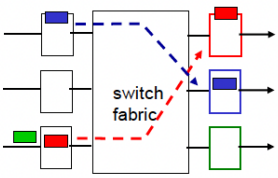
11001000 00010111 00011000 10101010 which interface?

- Input Port Queueing
  - Head-of-the-Line(HOL) blocking

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



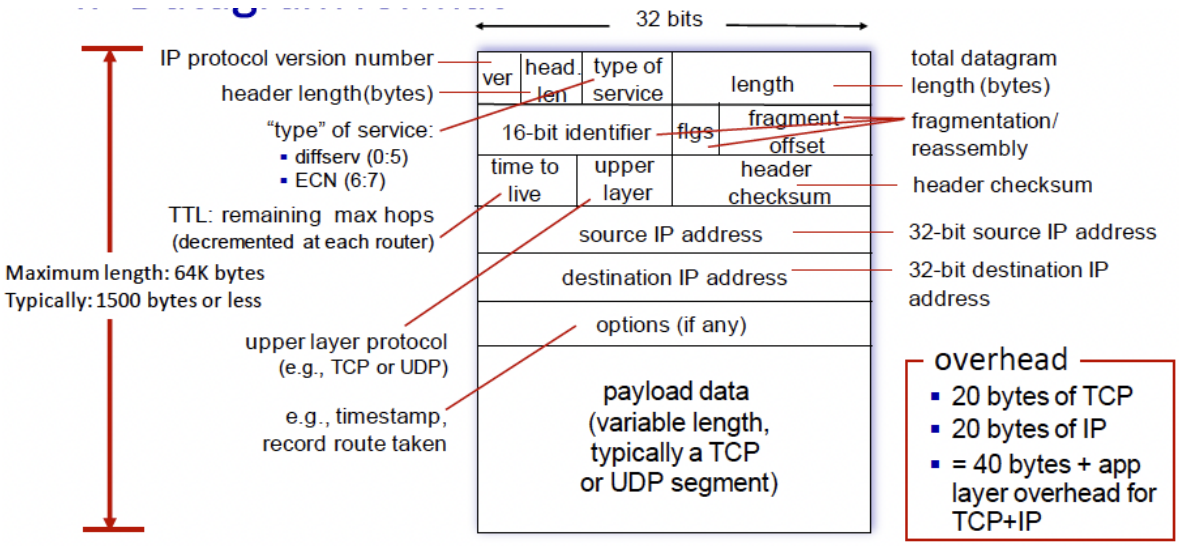
output port contention: only one red datagram can be transferred. lower red packet is *blocked*



one packet time later: green packet experiences HOL blocking

- Output ports
  - Buffering
  - Drop policy
  - Scheduling Discipline

- IP: Internet Protocol
  - Datagram Format

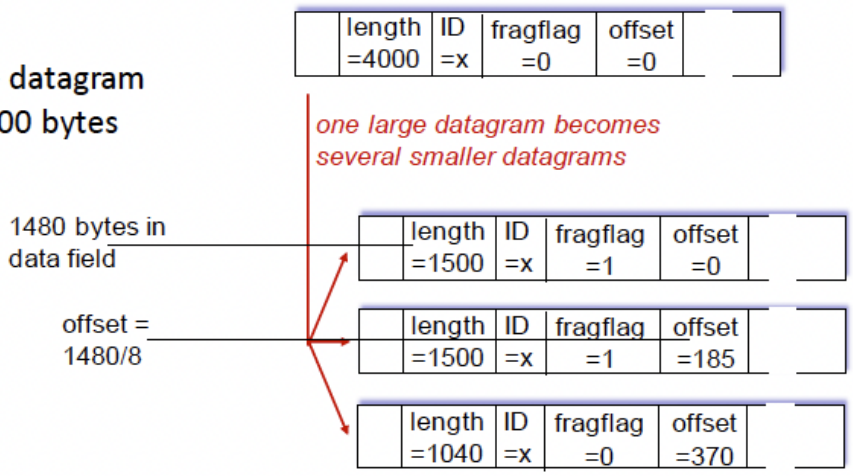


  - IP fragmentation/reassembly
    - MTU max. transfer size
    - large IP datagram divided within net

-

example:

- 4000 byte datagram
- MTU = 1500 bytes



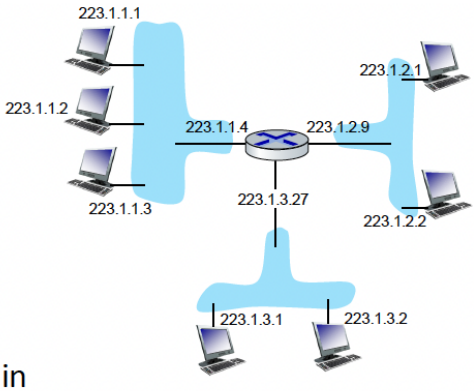
- Addressing
  - Introduction
    - 32-bit
    - interface: connection between host/router and physical link
  - Subnet
    -

What's 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



network consisting of 3 subnets

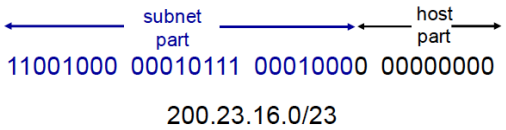
  - CIDR: Classless InterDomain Routing
    - a.b.c.d/x, where x is # bits in subnet portion of address

-

IP addressing: CIDR

CIDR: Classless InterDomain Routing (pronounced “cider”)

- subnet portion of address of arbitrary length
- address format: a.b.c.d/x, where x is # bits in subnet portion of address



200.23.16.0/23

- How to get an IP address?
  - How does host
    - Hardcode
    - DHCP: Dynamic Host Configuration Protocol
      -

        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

      - DHCP can return more than just allocated IP address on subnet
  - How does network
    - Gets allocated portion of its provider ISP's address space
    - Hierarchical addressing
  - How does an ISP get block of addresses?
    - ICANN

IP addressing: last words ...

Q: how does an ISP get block of addresses?

A: ICANN: Internet Corporation for Assigned Names and Numbers http://www.icann.org/

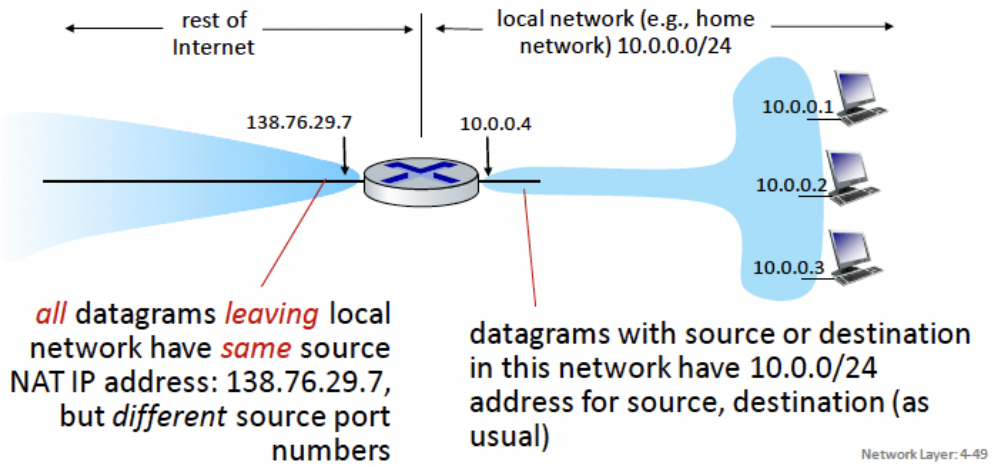
- allocates IP addresses
- manages DNS
- assigns domain names

Q: are there enough 32-bit IP addresses?

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

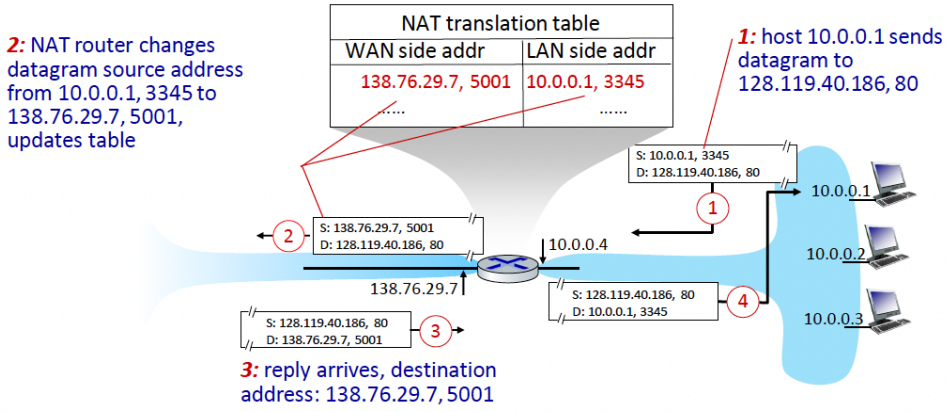
"Who the hell knew how much address space we needed?"
Vint Cerf (reflecting on decision to make IPv4 address 32 bits long)

- Network Address Translation
  - All devices in local network share just one IPv4 address as far as outside world is concerned



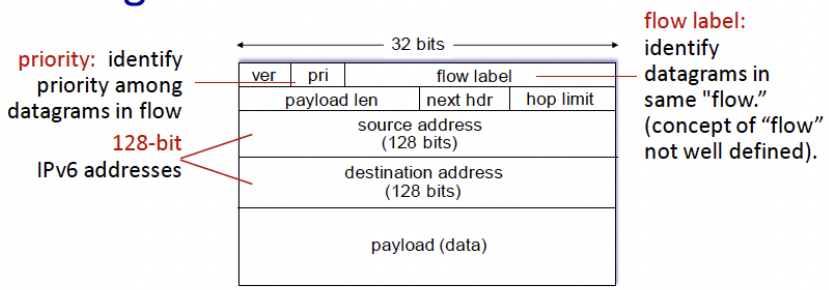
  - NAT translation table

NAT: network address translation



- IPv6
  - 128bit
  - Datagram Format
    -

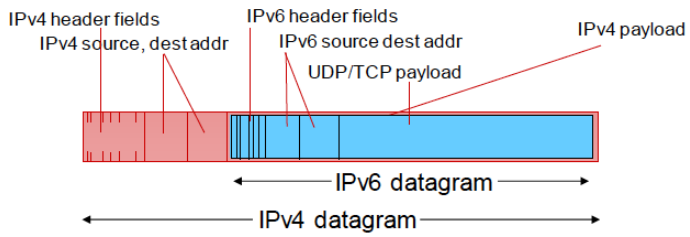
IPv6 datagram format



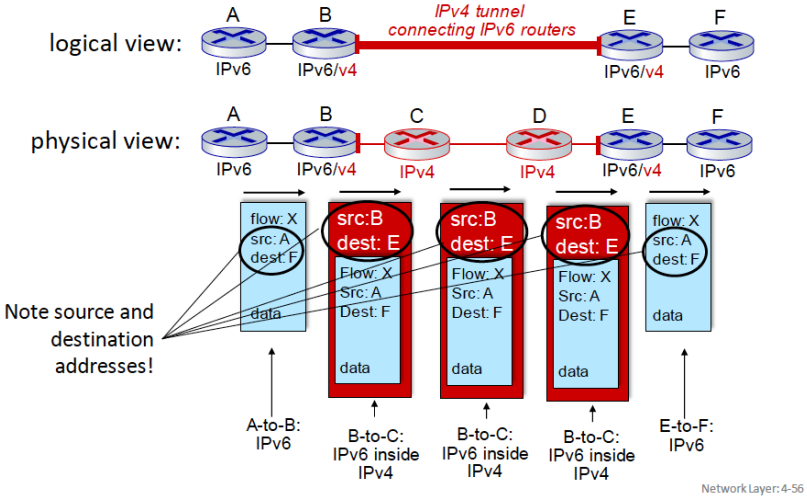
What's missing (compared with IPv4):
- no checksum (to speed processing at routers)
- no fragmentation/reassembly
- no options

  - Transition form IPv4 to IPv6
    - Tunneling
      -

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



Tunneling



- Generalized Forwarding, SDN

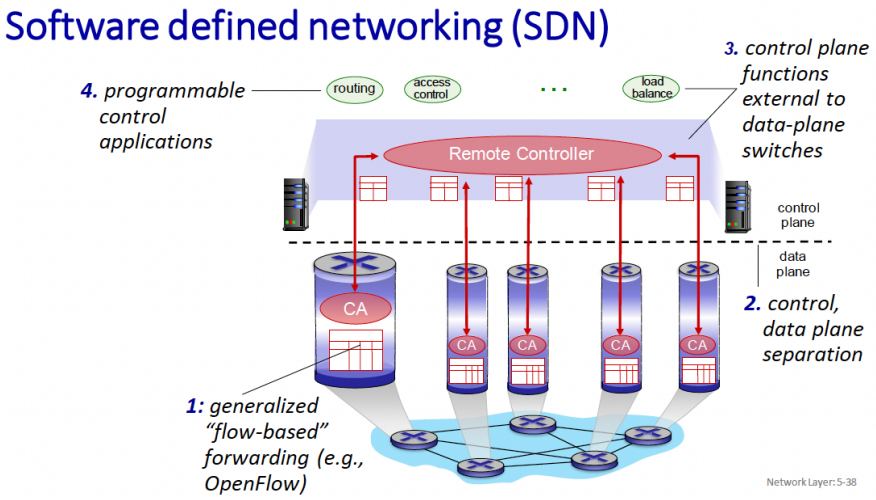
Software defined networking (SDN)

Why a *logically centralized* control plane?

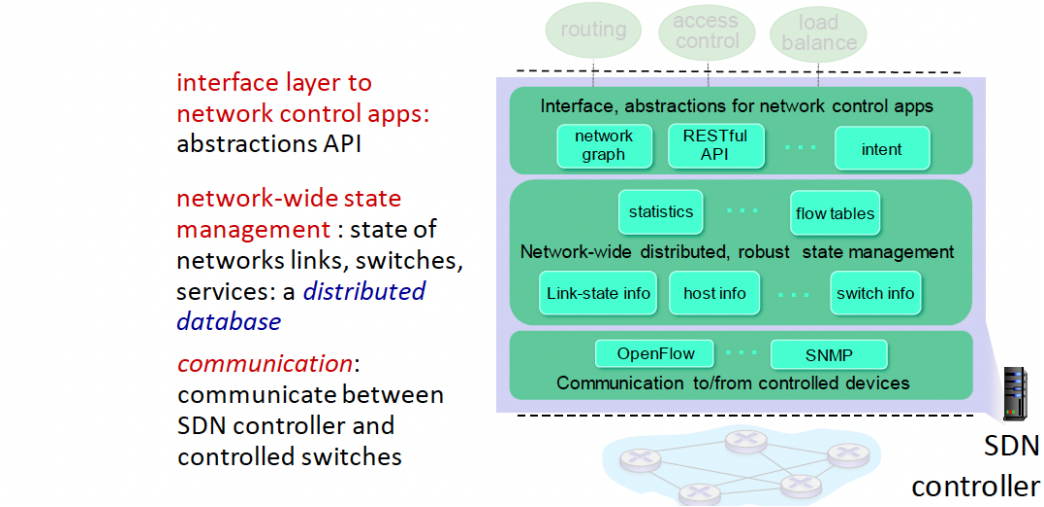
- easier network management: avoid router misconfigurations, greater flexibility of traffic flows
- table-based forwarding (recall OpenFlow API) allows "programming" routers
  - centralized "programming" easier: compute tables centrally and distribute
  - distributed "programming" more difficult: compute tables as result of distributed algorithm (protocol) implemented in each-and-every router
- open (non-proprietary) implementation of control plane
  - foster innovation: let 1000 flowers bloom

- Destination-Based Forwarding
  - forward based on IP address
- Implementation

-



- SDN Controller
  -



- Generalized Forwarding
- Open Flow
  - flow table

OpenFlow abstraction

- match+action: abstraction unifies different kinds of devices

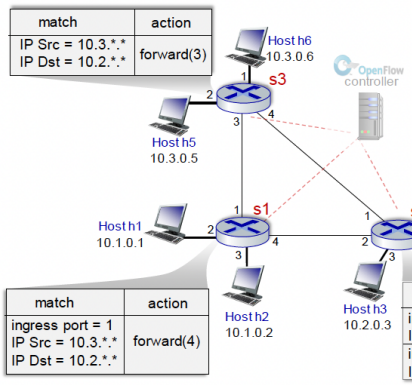
Router
- *match:* longest destination IP prefix
- *action:* forward out a link

Switch
- *match:* destination MAC address
- *action:* forward or flood

Firewall
- *match*: IP addresses and TCP/UDP port numbers
- *action:* permit or deny

NAT
- *match:* IP address and port
- *action:* rewrite address and port

OpenFlow example



Orchestrated tables can create *network-wide* behavior, e.g.,:
- datagrams from hosts h5 and h6 should be sent to h3 or h4, via s1 and from there to s2

Network Layer: Control Plane

Two approaches to structuring network control plane:

- per-router control (traditional)
- logically centralized control (software defined networking)

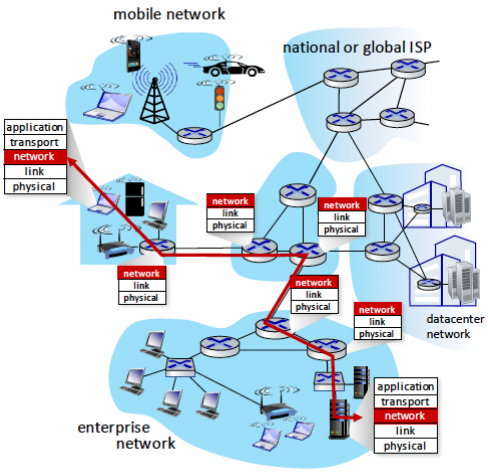
- Routing Algorithms
  - Routing Protocols

-

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!



- Classification
  -

Routing algorithm classification

global: all routers have *complete* topology, link cost info
- "link state" algorithms

How fast do routes change?

static: routes change slowly over time

dynamic: routes change more quickly
- periodic updates or in response to link cost changes

decentralized: iterative process of computation, exchange of info with neighbors
- routers initially only know link costs to attached neighbors
- "distance vector" algorithms

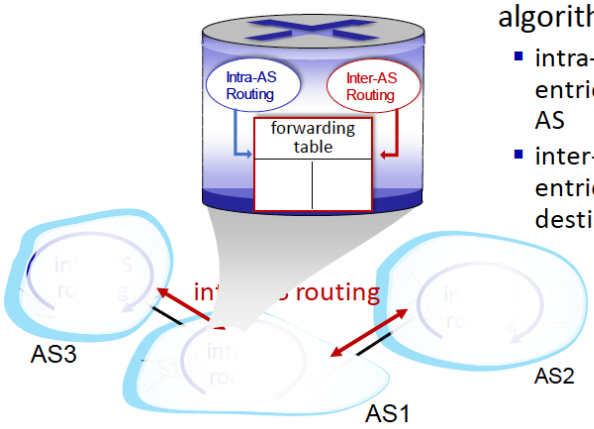
global or decentralized information?

- Intra-AS Routing in the Internet: OSPF
  - AS
    - Autonomous System
    - Interconnected Ases

Interconnected ASes

forwarding table configured by intra- and inter-AS routing algorithms

- intra-AS routing determine entries for destinations within AS
- inter-AS & intra-AS determine entries for external destinations

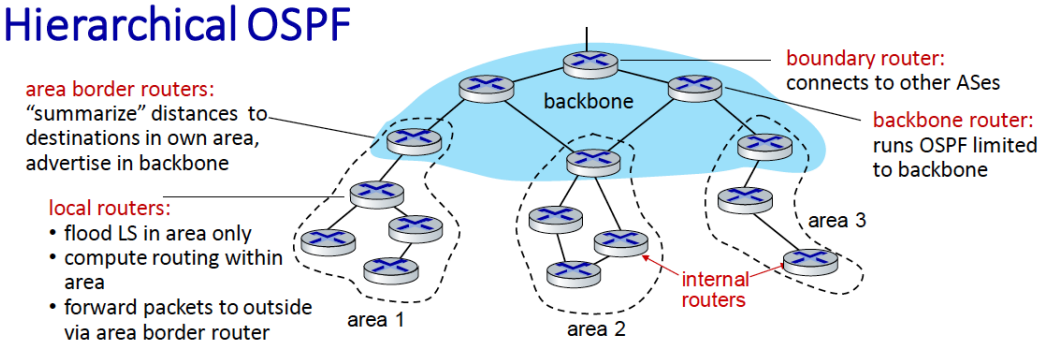


- Inter-AS
  - RIP: Routing Information Protocol
  - EIGRP: Enhanced Interior Gateway Routing Protocol
  - OSPF: Open Shortest Path First

OSPF (Open Shortest Path First) routing

- "open": publicly available
- classic link-state
  - each router floods OSPF link-state advertisements (directly over IP rather than using TCP/UDP) to all other routers in entire AS
  - multiple link costs metrics possible: bandwidth, delay
  - each router has full topology, uses Dijkstra's algorithm to compute forwarding table
- *security:* all OSPF messages authenticated (to prevent malicious intrusion)

Hierarchical OSPF



- two-level hierarchy: local area, backbone.
  - link-state advertisements flooded only in area, or backbone
  - each node has detailed area topology; only knows direction to reach other destinations

- Intra-AS
- Why different Intra-, Inter-AS Routing?

Why different Intra-, Inter-AS routing ?

policy:

- inter-AS: admin wants control over how its traffic routed, who routes through its network
- intra-AS: single admin, so policy less of an issue

scale:

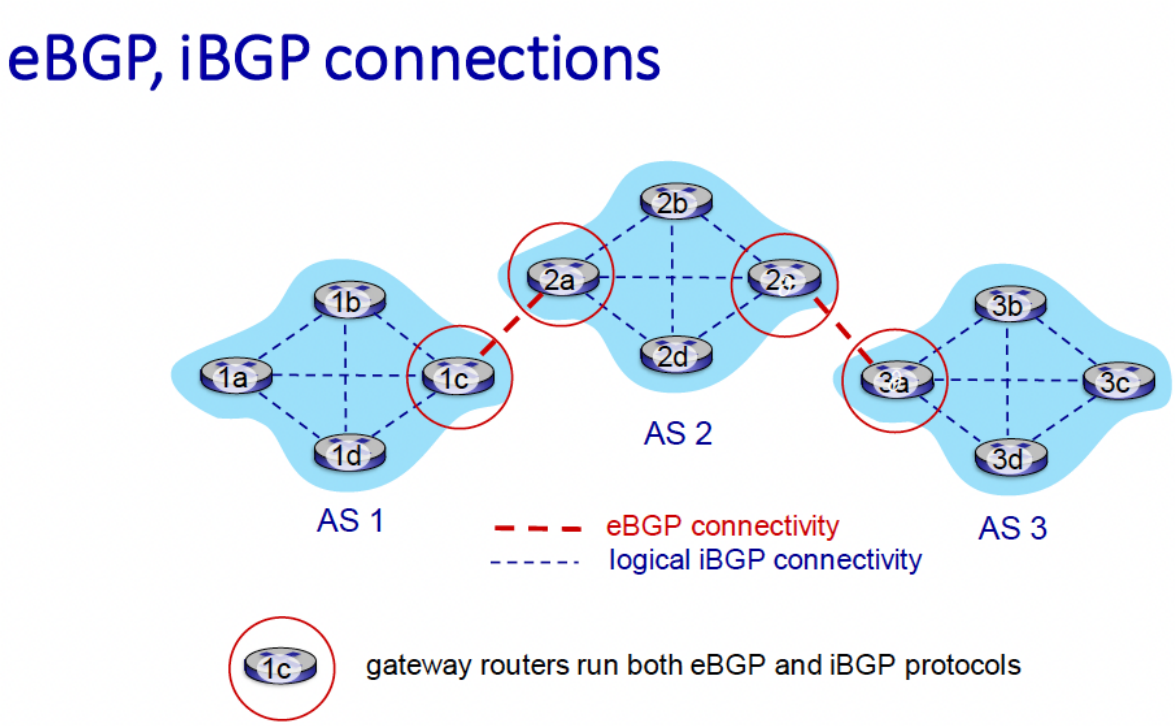
- hierarchical routing saves table size, reduced update traffic

performance:

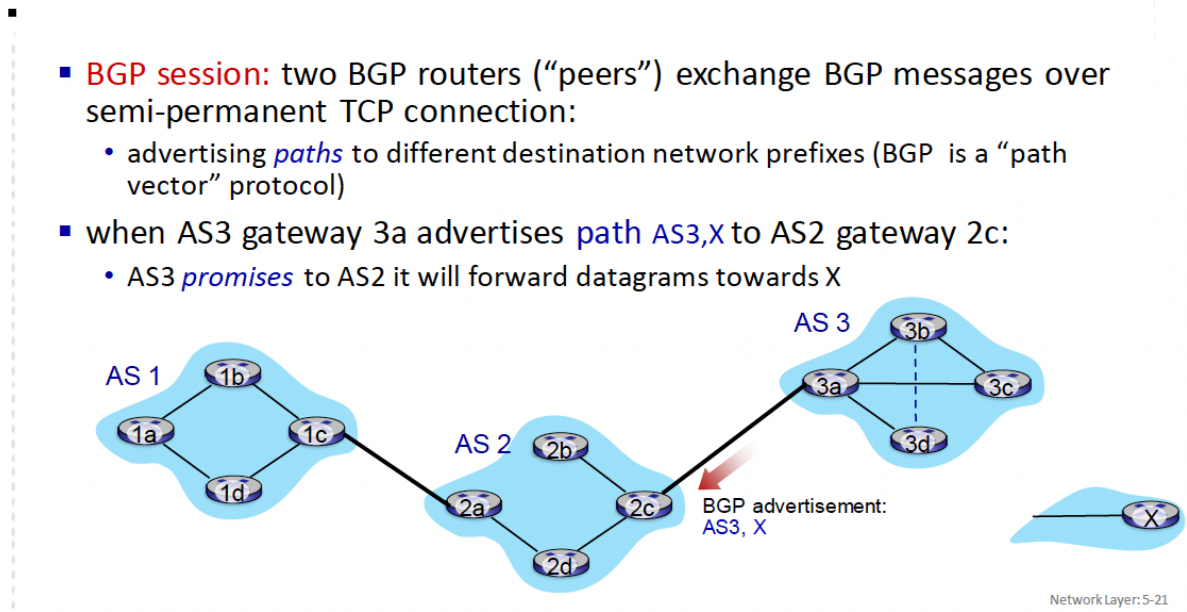
- intra-AS: can focus on performance
- inter-AS: policy dominates over performance

- Routing Among the ISPs: BGP
  - BGP: Border Gateway Protocol
    - eBGP
    - iBGP
    - AS-PATH
    - NEXT-HOP

-



- BGP Basics
  -



- The SDN Control Plane
  - Motivation
    - Software Defined Networking
  - Implementation
    - OpenFlow

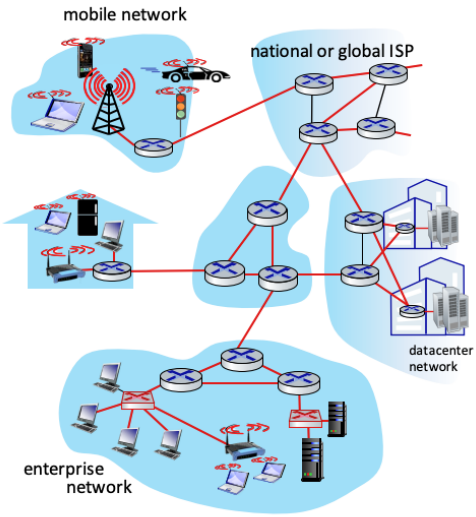
- **Link Layer**
  - Introduction
    -

Link layer: introduction

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



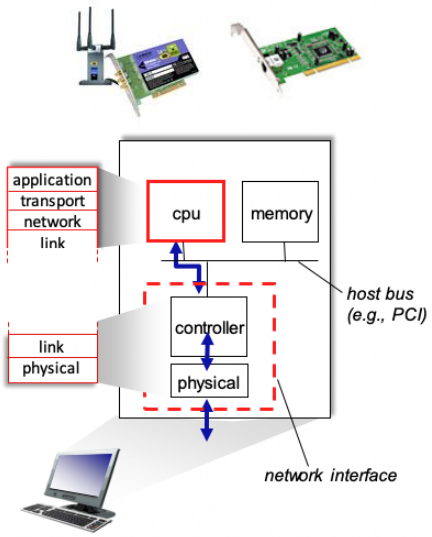
  - Services

- 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?

- 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

- Where 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



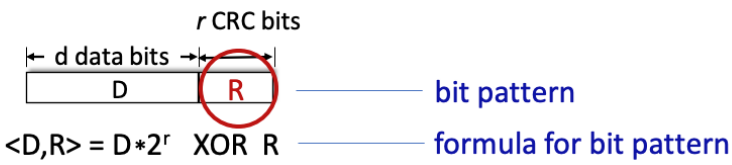
- Error Detection
  - Parity Checking
  - Internet checkSum
  - Cyclic Redundancy Check(CRC)

Cyclic Redundancy Check (CRC)

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

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 burst errors less than r+1 bits
- widely used in practice (Ethernet, 802.11 WiFi)



CRC example

Here: r = 3, because G has 4 bits

We want:

$D \cdot 2^r \ XOR \ R = nG$

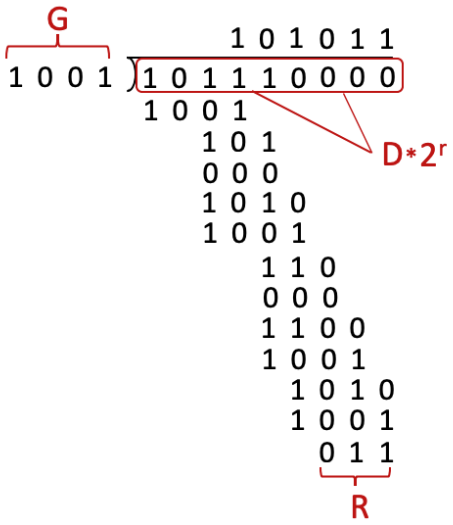
or equivalently:

$D \cdot 2^r = nG \ XOR \ R$

or equivalently:

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

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

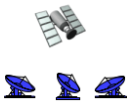


- Multiple access protocols
  - Multiple Access Links, protocols

Multiple access links, protocols

two types of "links":

- point-to-point
  - point-to-point link between Ethernet switch, host
  - PPP for dial-up access
- broadcast (shared wire or medium)
  - old-fashioned Ethernet
  - upstream HFC in cable-based access network
  - 802.11 wireless LAN, 4G/5G, satellite


shared radio: satellite


humans at a cocktail party (shared air, acoustical)


shared wire (e.g., cabled Ethernet)


shared radio: 4G/5G


shared radio: WiFi

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

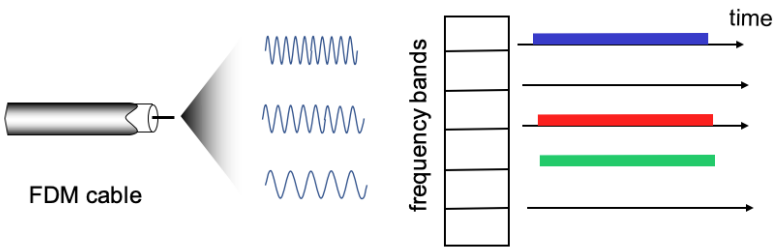
- distributed algorithm that determines how nodes share channel, i.e., determine when node can transmit
- communication about channel sharing must use channel itself!
  - no out-of-band channel for coordination

- Multiple Access Channel(MAC)
  - Channel Partitioning MAC Protocols
    - Frequency division multiple access

Channel partitioning MAC protocols: FDMA

FDMA: frequency division multiple access

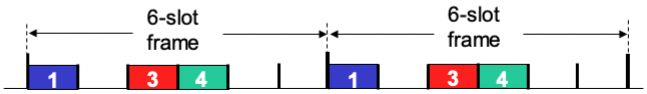
- channel spectrum divided into frequency bands
- each station assigned fixed frequency band
- unused transmission time in frequency bands go idle
- example: 6-station LAN, 1,3,4 have packet to send, frequency bands 2,5,6 idle



    - Time division multiple access

TDMA: time division multiple access

- access to channel in "rounds"
- each station gets fixed length slot (length = packet transmission time) in each round
- unused slots go idle
- example: 6-station LAN, 1,3,4 have packets to send, slots 2,5,6 idle



- Random access protocols
  - ALOHA

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 synchronized
- 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

randomization – *why*?

  - slotted ALOHA
    - pros

- Pros:
  - single active node can continuously transmit at full rate of channel
  - highly decentralized: only slots in nodes need to be in sync
  - simple
- cons
  - Cons:
    - collisions, wasting slots
    - idle slots
    - nodes may be able to detect collision in less than time to transmit packet
    - clock synchronization
- CSMA
  - Carrier Sense Multiple Access
    - CSMA (carrier sense multiple access)

      simple CSMA: listen before transmit:
      - if channel sensed idle: transmit entire frame
      - if channel sensed busy: defer transmission

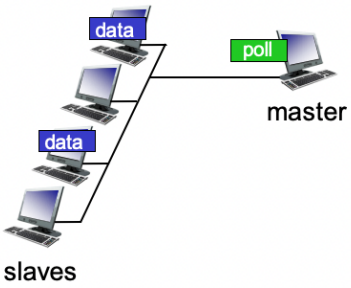
      - human analogy: don't interrupt others!
- CSMA/CD
  - CSMA with collision detection
    - CSMA/CD: CSMA with *collision detection*
      - collisions *detected* within short time
      - colliding transmissions aborted, reducing channel wastage
      - collision detection easy in wired, difficult with wireless

      - human analogy: the polite conversationalist

- Ethernet
- CSMA/CA
  - 802.11
- Taking Turns MAC protocols
  - polling
    -

polling:

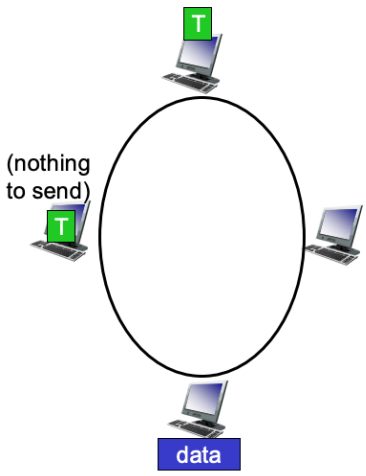
- master node "invites" other nodes to transmit in turn
- typically used with "dumb" devices
- concerns:
  - polling overhead
  - latency
  - single point of failure (master)



  - token passing
    -

token passing:

- control *token* passed from one node to next sequentially.
- token message
- concerns:
  - token overhead
  - latency
  - single point of failure (token)



- Summary

Summary of MAC protocols

- channel partitioning, by time, frequency or code
  - Time Division, Frequency Division
- random access (dynamic),
  - ALOHA, S-ALOHA, CSMA, CSMA/CD
  - carrier sensing: easy in some technologies (wire), hard in others (wireless)
  - CSMA/CD used in Ethernet
  - CSMA/CA used in 802.11
- taking turns
  - polling from central site, token passing
  - Bluetooth, IBM token ring

MAC addresses

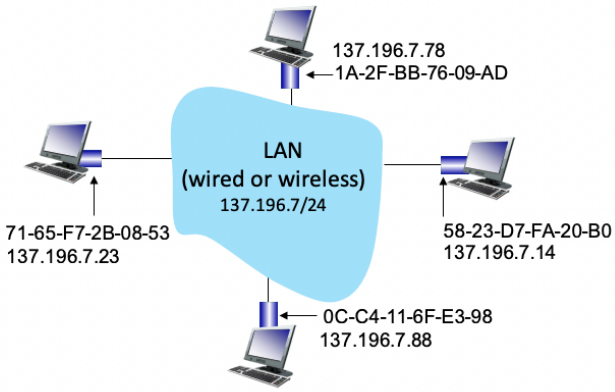
- 32-bit IP address:
  - *network-layer* address for interface
  - used for layer 3 (network layer) forwarding
  - e.g.: 128.119.40.136
- MAC (or LAN or physical or Ethernet) address:
  - function: used "locally" to get frame from one interface to another physically-connected interface (same subnet, in IP-addressing sense)
  - 48-bit MAC address burned in NIC ROM, also sometimes software settable
  - e.g.: 1A-2F-BB-76-09-AD

hexadecimal (base 16) notation (each "numeral" represents 4 bits)

MAC addresses

each interface on LAN

- has a locally unique 32-bit IP address (as we've seen)
- has unique 48-bit MAC address



MAC addresses

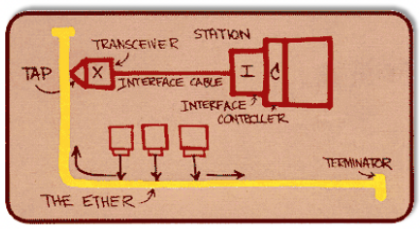
- MAC address allocation administered by IEEE
- manufacturer buys portion of MAC address space (to assure uniqueness)
- analogy:
  - MAC address: like National Insurance Number
  - IP address: like postal address
- MAC flat address: portability
  - can move interface from one LAN to another
  - recall IP address *not* portable: depends on IP subnet to which node is attached

- ARP: Address Resolution Protocol
  - ARP table
  - TTL
- Ethernet

Ethernet

"dominant" wired LAN technology:

- first widely used LAN technology
- simpler, cheap
- kept up with speed race: 10 Mbps – 400 Gbps



Metcalfe's Ethernet sketch

https://www.uspto.gov/learning-and-resources/journeys-innovation/audio-stories/defying-doubters

- Physical Topology
  - Bus
  - Switched
- Frame Structure

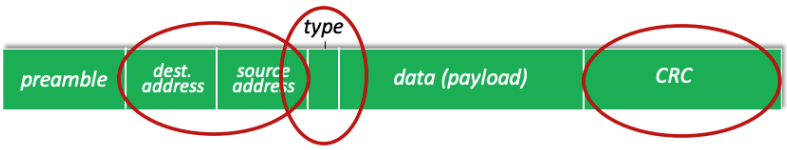
sending interface encapsulates IP datagram (or other network layer protocol packet) in Ethernet frame

preamble:

- used to synchronize receiver, sender clock rates
- 7 bytes of 10101010 followed by one byte of 10101011

-

Ethernet frame structure (more)



- addresses: 6 byte 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

- Unreliable, connectionless
  -

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

- Switches
  - Ethernet Switch

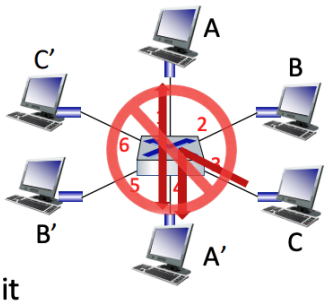
Ethernet switch

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

▾ Multiple Simultaneous Transmissions

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

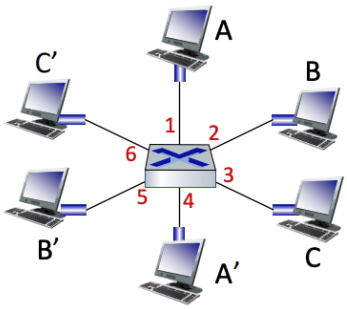
- Self-learning
- Interconnecting switches
- Switches vs. Routers

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!

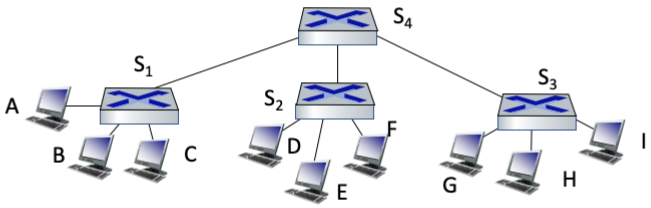


Q: how are entries created, maintained in switch table?

- something like a routing protocol?

Interconnecting switches

self-learning switches can be connected together:



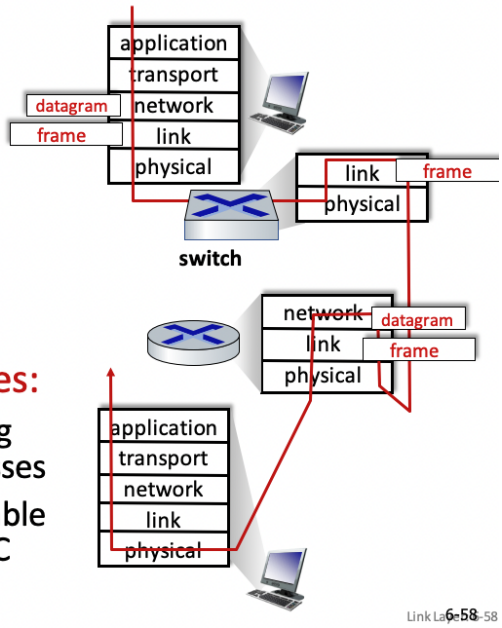
Q: sending from A to G - how does S_1 know to forward frame destined to G via S_4 and S_3?

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

Switches vs. routers

both are store-and-forward:

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



both have forwarding tables:

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

- Wireless links and WLAN
  - Wireless Link Characteristics

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., 2.4 GHz) 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

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

  - IEEE 802.11 Wireless LAN

IEEE 802.11 Wireless LAN

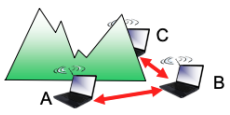
IEEE 802.11 standard	Year	Max data rate	Range	Frequency
802.11b	1999	11 Mbps	30 m	2.4 Ghz
802.11g	2003	54 Mbps	30m	2.4 Ghz
802.11n (WiFi 4)	2009	600	70m	2.4, 5 Ghz
802.11ac (WiFi 5)	2013	3.47Gpbs	70m	5 Ghz
802.11ax (WiFi 6)	2020 (exp.)	14 Gbps	70m	2.4, 5 Ghz
802.11af	2014	35 – 560 Mbps	1 Km	unused TV bands (54-790 MHz)
802.11ah	2017	347Mbps	1 Km	900 Mhz

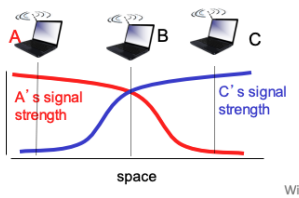
- all use CSMA/CA for multiple access, and have base-station and ad-hoc network versions

- Multiple Access

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





- MAC Protocol

IEEE 802.11 MAC Protocol: CSMA/CA

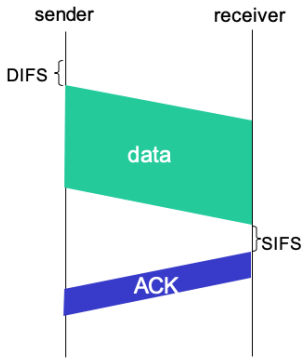
802.11 sender

1 if sense channel idle for **DIFS** then
transmit entire frame (no CD)

2 if sense channel busy then
start random backoff time
timer counts down while channel idle
transmit when timer expires
if no ACK, increase random backoff interval, repeat 2

802.11 receiver

if frame received OK
return ACK after **SIFS** (ACK needed due to hidden terminal problem)



- Avoiding Collision

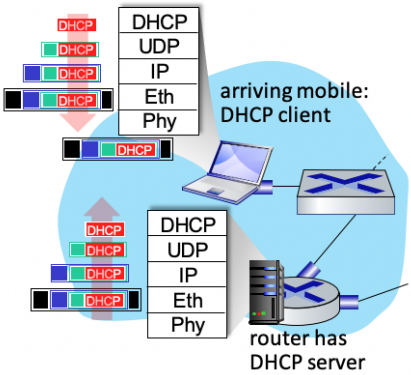
Avoiding collisions (more)

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

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

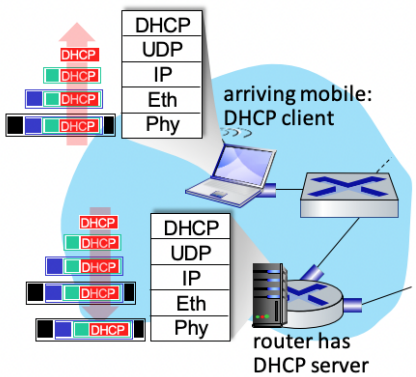
- A day in the life of a web request

A day in the life: connecting to the Internet



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

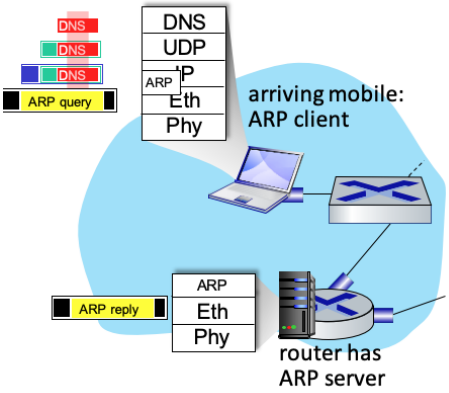
A day in the life: connecting to the Internet



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

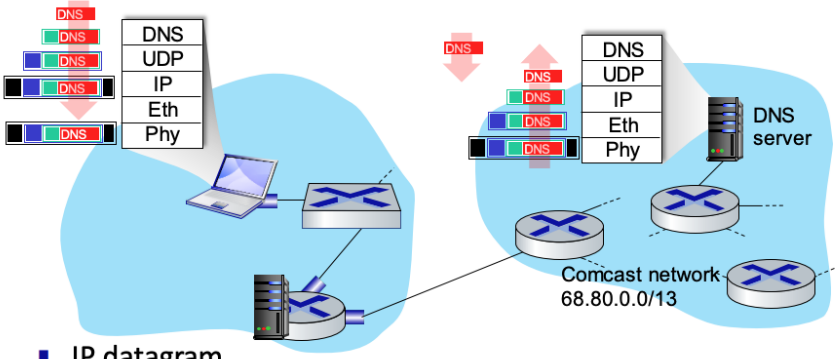
Client now has IP address, knows name & addr of DNS server, IP address of its first-hop router

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



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

A day in the life… using DNS



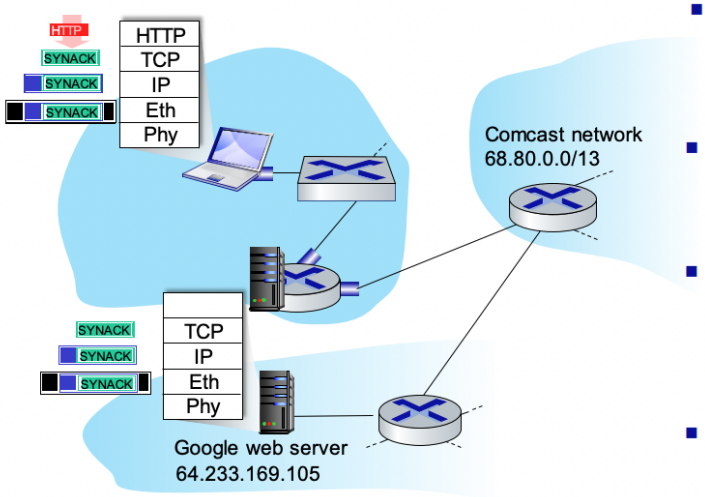
- demuxed to DNS
- DNS replies to client with IP address of www.google.com

- IP datagram containing DNS query forwarded via LAN switch from client to 1st hop router

- IP datagram forwarded from campus network into Comcast network, routed (tables created by RIP, OSPF, IS-IS and/or BGP routing protocols) to DNS server

-

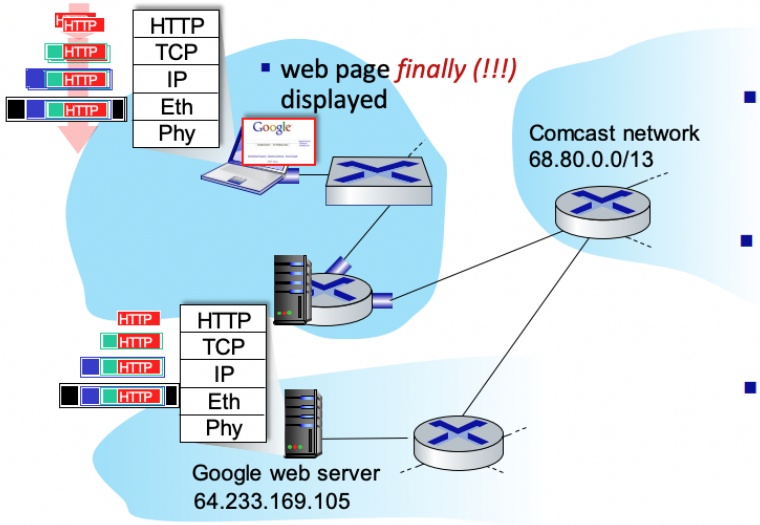
A day in the life…TCP connection carrying HTTP



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

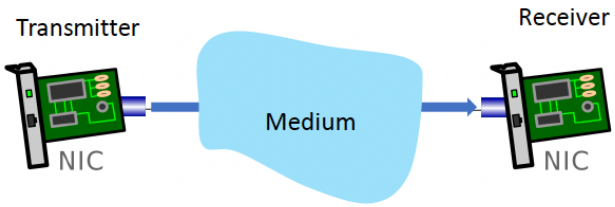
-

A day in the life… HTTP request/reply



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

- **Physical Layer**
  - Terminology



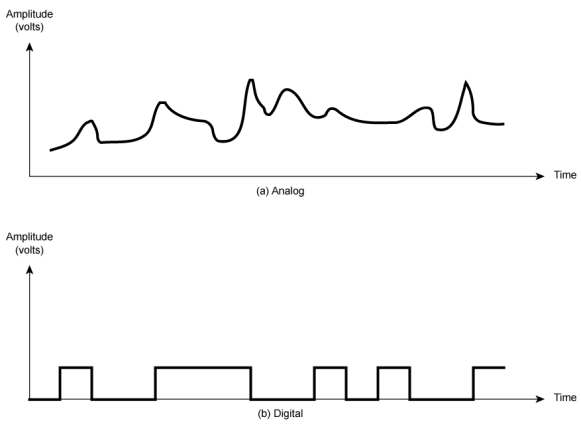
- Transmitter
- Receiver
- Medium
  - Guided medium
    - e.g. twisted pair, optical fiber
  - Unguided medium
    - e.g. air, water, vacuum

Data and Signals

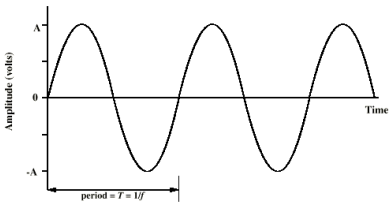
Frequency, Spectrum and Bandwidth

- Time domain concepts
  - Analog signal
    - Varies in a smooth way over time
  - Digital signal
    - Maintains a constant level then changes to another constant level
  - Periodic signal
    - Pattern repeated over time
  - Aperiodic signal
    - Pattern not repeated over time

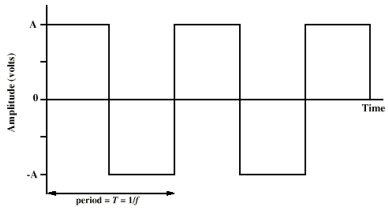
Analog & Digital



Periodic Signal



(a) Sine wave



(b) Square wave

- Sine wave
  - Peak Amplitude (A)
    - maximum strength of signal
    - volts
  - Frequency (f)
    - Rate of change of signal
    - Hertz (Hz) or cycles per second
    - Period = time for one repetition (T)
    - T = 1/f
  - Phase (ϕ)
    - Relative position in time

- Spectrum & Bandwidth

Spectrum & Bandwidth

- Spectrum
  - range of frequencies contained in signal
- Absolute bandwidth
  - width of spectrum
- Effective bandwidth
  - Often just bandwidth
  - Narrow band of frequencies containing most of the energy
- May have component of 0 frequency (direct current)

- Analog and Digital Transmission

- Data
  - Entities that convey meaning
  - Analog
    - Continuous values within some interval (e.g. sound)
  - Digital
    - Discrete values (e.g. text)
- Signals
  - Electric or electromagnetic representations of data
- Transmission
  - Communication of data by propagation and processing of signals

Analog Transmission

- Analog signal transmitted without regard to content
- May be analog or digital data
- Attenuated over distance
- Use amplifiers to boost signal
- Also amplifies noise

Digital Transmission

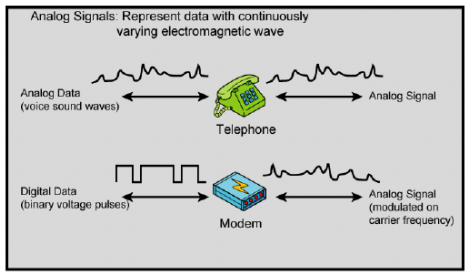
- Concerned with content
- Integrity endangered by noise, attenuation etc.
- Repeaters used
  - Repeater receives signal
  - Extracts bit pattern
  - Retransmits
- Attenuation is overcome
- Noise is not amplified

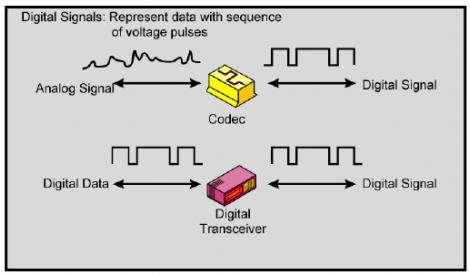
Advantages of Digital Transmission

- Digital technology lower cost
- Data integrity
  - Longer distances over lower quality lines
- Capacity utilization
  - High bandwidth links economical
  - High degree of multiplexing easier with digital techniques
- Security & Privacy
  - Encryption
- Integration
  - Can treat analog and digital data similarly

Data and Signals

- Usually use digital signals for digital data and analog signals for analog data
- Can use analog signal to carry digital data
  - Modem
- Can use digital signal to carry analog data
  - e.g. audio mp3 codec





- Transmission Impairments

- Signal received may differ from signal transmitted
- Analog - degradation of signal quality
- Digital - bit errors
- Caused by
  - Attenuation and attenuation distortion
  - Delay distortion
  - Noise

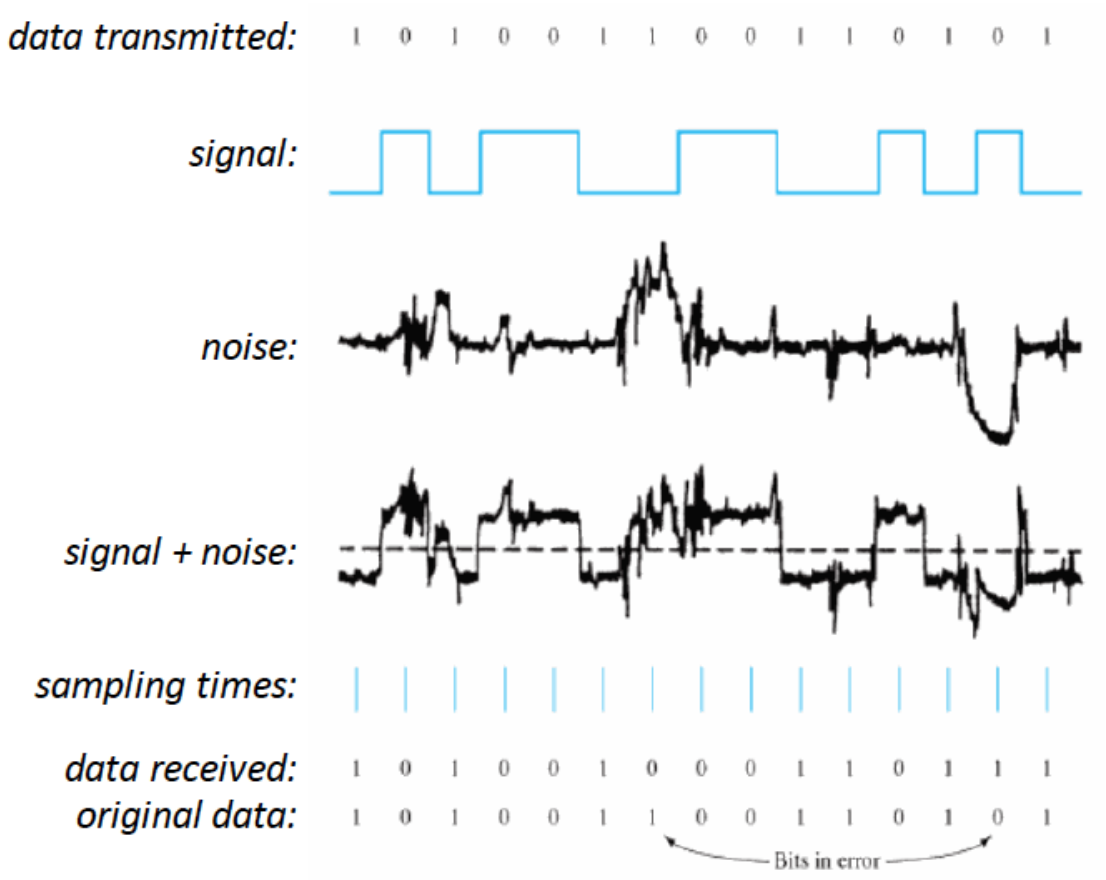
- Attenuation

- Signal strength falls off with distance
- Depends on medium
- Received signal strength:
  - must be enough to be detected
  - must be sufficiently higher than noise to be received without error
- Attenuation is an increasing function of frequency

- Delay Distortion
  - Occurs because the velocity of propagation of a signal through a guided medium varies with frequency.
    - Only in guided media
    - Propagation velocity varies with frequency
    - Different parts of the spectrum arrive at different times.

- Noise
  - Additional signals inserted between transmitter and receiver
  - Thermal
    - Due to thermal agitation of electrons
    - Uniformly distributed
    - White noise
  - Intermodulation
    - Signals that are the sum and difference of original frequencies sharing a medium

- Crosstalk
  - A signal from one line is picked up by another
- Impulse
  - Irregular pulses or spikes
  - e.g. External electromagnetic interference
  - Short duration
  - High amplitude



- Data Rate Limits
  - Channel Capacity

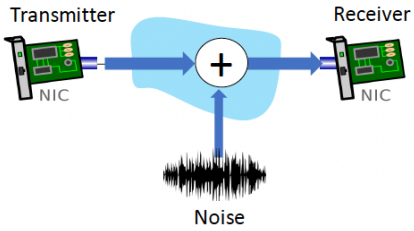
- Data rate
  - In bits per second
  - Rate at which data can be communicated
- Bandwidth
  - The frequency width of the transmitted signal
  - In cycles per second or Hertz
  - Constrained by transmitter and medium.

▾ Nyquist Bandwidth

- Nyquist: "If bandwidth is B, then highest signal transmission (baud) rate is 2B."
- For binary signal: data rate supported by B Hz is 2B bps.
- Can be increased by using M signal states.
  - each state encodes multiple bits
  - How many bits can we encode with M signal states ?
- No noise: Max data rate = 2·B·log[base 2](M)
- Example:
  - Suppose a voice channel (B=3100 Hz) is used to transmit digital data via modem which uses 4 different signal states.
  - Then, max data rate = 2 · 3100 · 2 bps = 12400 bps.

▾ Shannon's Capacity Formula

Shannon's Capacity Formula



- Communications channels have noise present
  - For example, the motion of molecules in the system create random thermal noise.
- The amount of thermal noise present is measured by the ratio of signal power to noise power
  - This is called the signal-to-noise ratio, S/N.
- Usually the ratio itself is not quoted, but this quantity:
  - 10 log[base 10] (S/N)
    - This is measured in decibels (dB)
  - S/N = 10 → 10 dB
  - S/N = 100 →. 20 dB
  - S/N = 1000 → 30 dB
- Fixed analog voice telephone network is typically:
  - 30 dB

Shannon's Law

- The maximum data rate of a noisy channel with bandwidth B and signal-to-noise ratio of S/N is
  - Max data rate = B · log[base 2] (1 + S/N)
- Example:
  - A typical analog voice telephone channel:
  - The channel has bandwidth of 3100 Hz
  - And Signal-to-noise ratio of 30 dB (ie S/N = 1000)
- Max bits per second:

  = B · log[base 2] (1 + S/N)

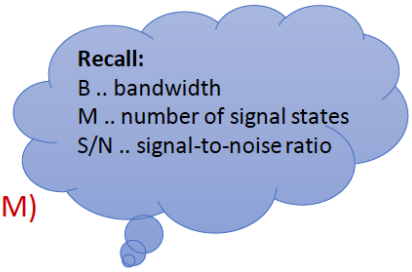
  = 3100 · log[base 2] (1 + 1000)

  = 3100 · log[base 2] (1001)

  = 3100 · 9.9658 = 30,894 bps.

Excercise

Nyquist / Shannon



- Noise free channel (Nyquist)
  - Max data rate = 2 · B · log[base 2](M)
- noisy channel (Shannon)
  - Max data rate = B · log[base 2] (1 + S/N)
- Exercise:
  - Consider a communication channel with bandwidth B = 3000 Hz.
    - Suppose the channel has a signal-to-noise ratio S/N = 1023. What is the maximum data rate of this channel?
    - What is the minimum number of signal states M needed to achieve a data rate of 24000 bps? How many bits must each signal state encode?

Guided Transmission Media

- Magnetic Media
- Twisted Pair
- Coaxial Cable
- Fiber Optics

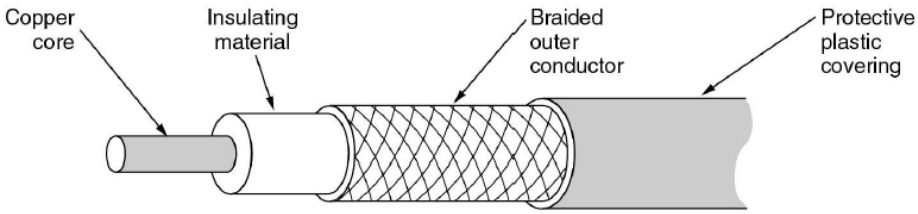
Twisted Pair

(a)

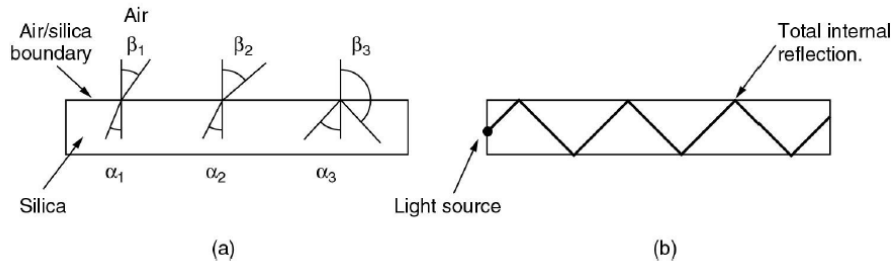
(b)

(a) Category 3 UTP (Unshielded Twisted Pair).
(b) Category 5 UTP: less crosstalk, better quality signal.

Coaxial Cable



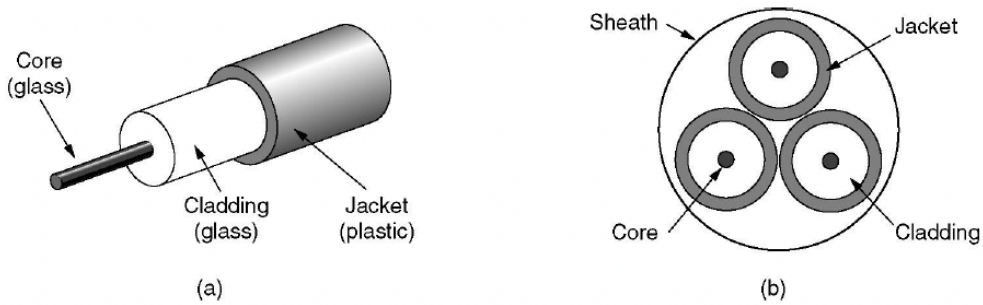
Fiber Optics



(a) Three examples of a light ray from inside a silica fiber impinging on the air/silica boundary at different angles.

(b) Light trapped by total internal reflection.

Fiber Cables



(a) Side view of a single fiber.

(b) End view of a sheath with three fibers.

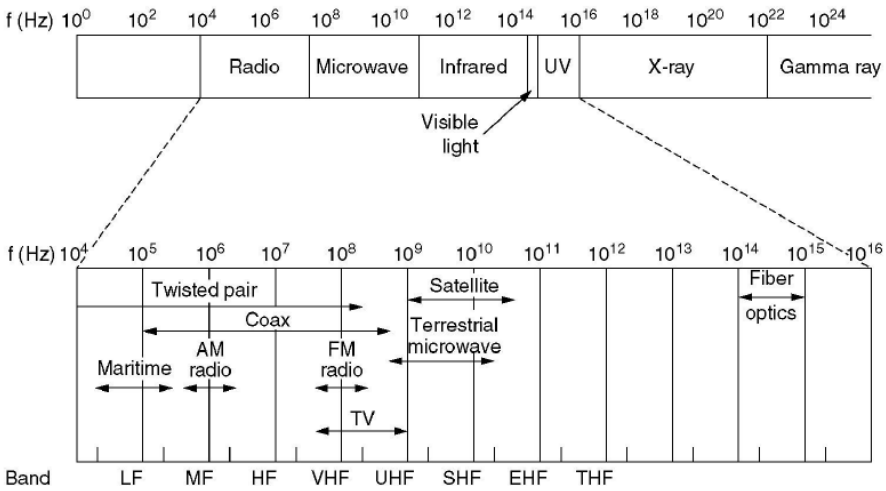
- Wireless Transmission

Wireless Transmission

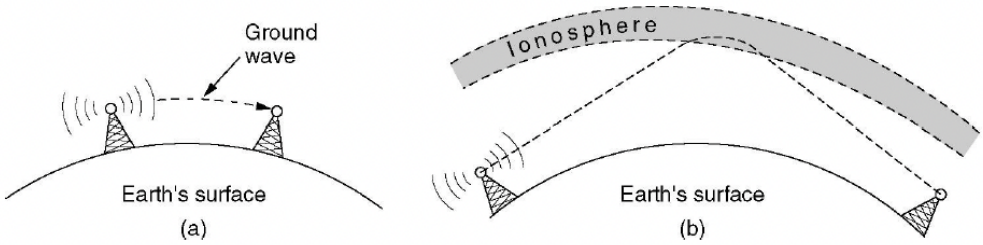
- The Electromagnetic Spectrum
- Radio Transmission
- Microwave Transmission
- Infrared and Millimeter Waves
- Lightwave Transmission

The Electromagnetic Spectrum

... and its uses for communication.



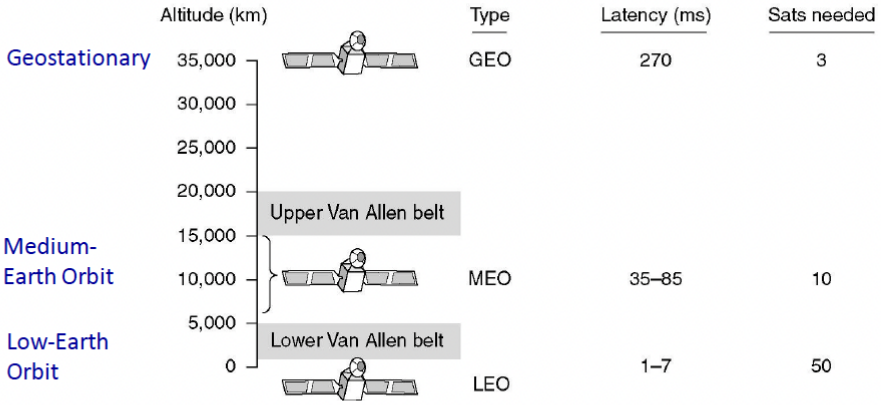
Radio Transmission



(a) In the VLF, LF, and MF bands, radio waves follow the curvature of the earth.

(b) In the HF band, they bounce off the ionosphere.

Communication Satellites

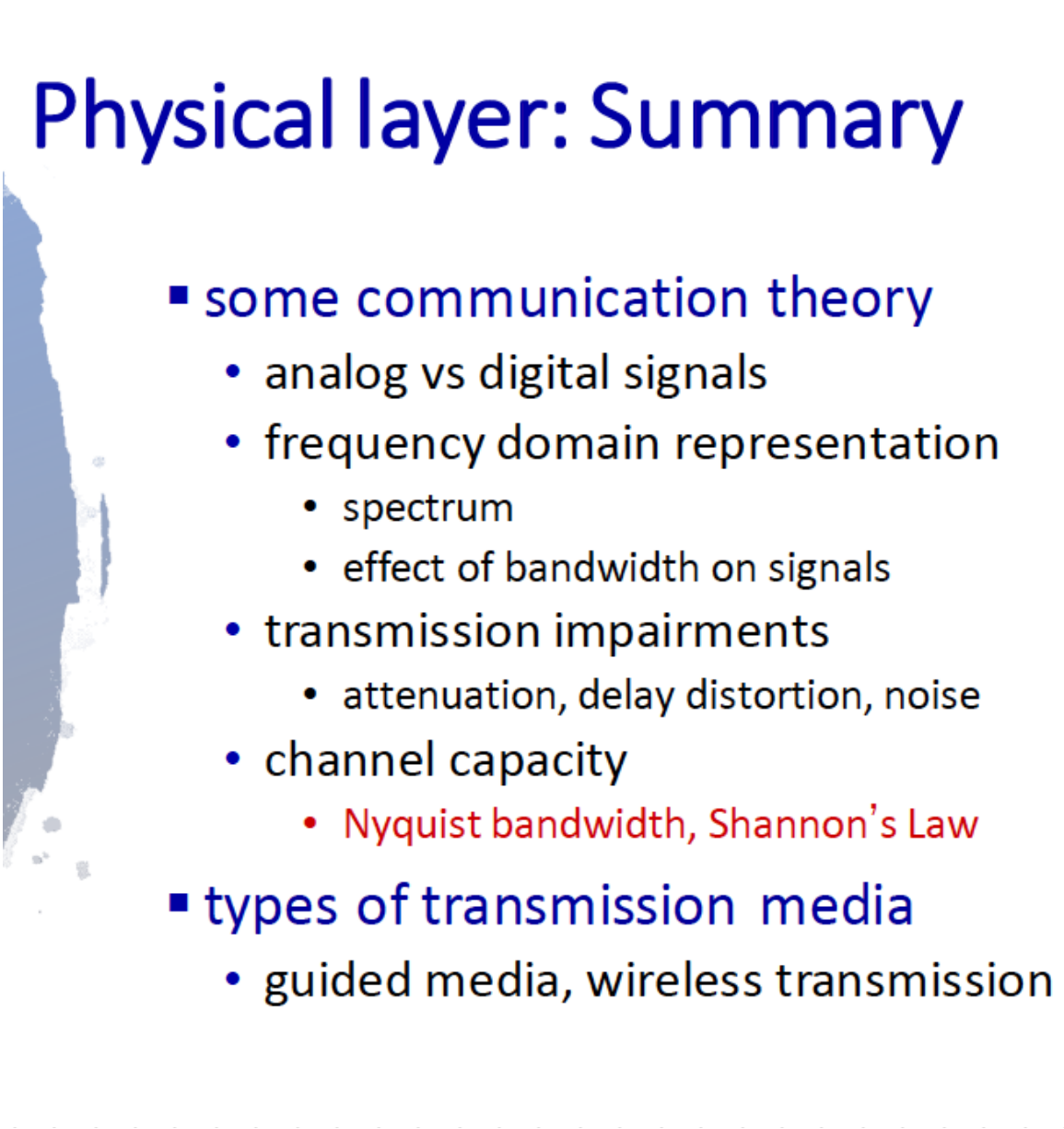


Communication satellites and some of their properties, including altitude above the earth, round-trip delay time and number of satellites needed for global coverage.

Band	Downlink	Uplink	Bandwidth	Problems
L	1.5 GHz	1.6 GHz	15 MHz	Low bandwidth; crowded
S	1.9 GHz	2.2 GHz	70 MHz	Low bandwidth; crowded
C	4.0 GHz	6.0 GHz	500 MHz	Terrestrial interference
Ku	11 GHz	14 GHz	500 MHz	Rain
Ka	20 GHz	30 GHz	3500 MHz	Rain, equipment cost

The principal satellite bands.

- Summary



- Review Questions

Review Questions

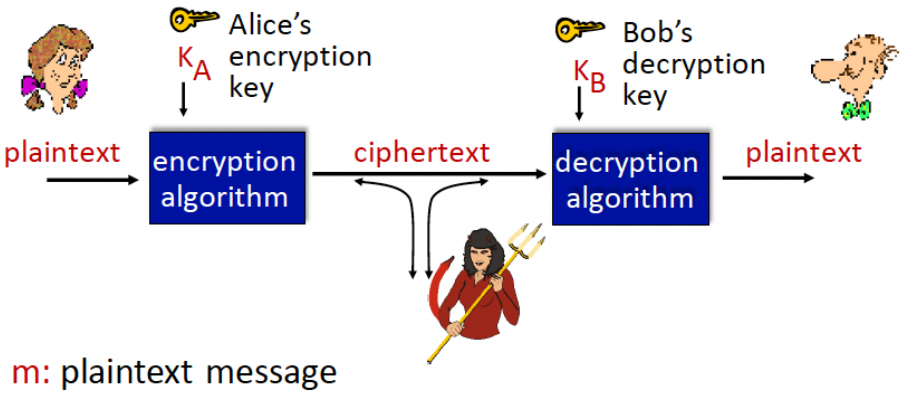
1. What are the advantages of digital transmission?
2. What is the problem of "Delay Distortion"? "Attenuation"?
3. Name and describe four different sources of noise?
4. What is the signal-to-noise ratio corresponding to 20dB?
5. Consider a communication channel with bandwidth B=5000Hz.
   a) Suppose S/N=255. What is the maximum data rate of this channel?
   b) What is the minimum number of signal states M needed to achieve a data rate of 20000 bps? How many bits must each state encode?

- **Network Security**
  - Introduction
    - Cryptography

Cryptography

- From the Greek words: 'Cryptos' (= secret) and 'Grafien (= writing)
- From ancient times to around 30 years ago: essentially private communications for personal, political and military matters
- Today: study and application of techniques relying on the existence of hard problems
- A lot of historic uses of Cryptography...

The language of cryptography



m: plaintext message

$K_A(m)$: ciphertext, encrypted with key K_A

$m = K_B(K_A(m))$

- Symmetric Key Cryptography
  - Block Ciphers

    - Message to be encrypted is processed in blocks of k bits (e.g., 64-bit blocks).
    - 1-to-1 mapping is used to map k-bit block of plaintext to k-bit block of ciphertext

Example with k=3:

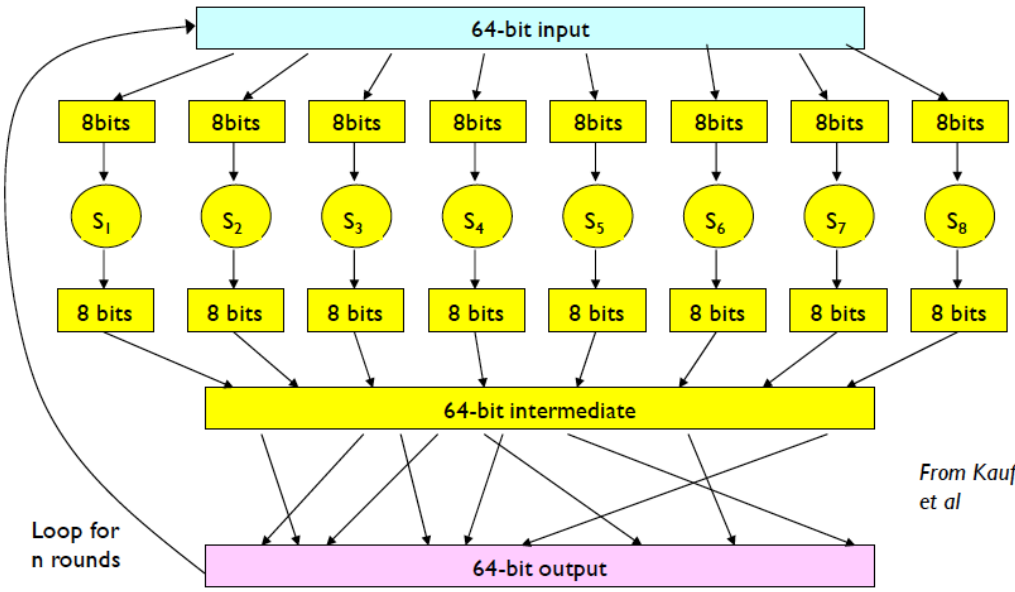
input	output	input	output
000	110	100	011
001	111	101	010
010	101	110	000
011	100	111	001

What is the ciphertext for 010110001111 ?

Block ciphers

- How many possible mappings are there for k=3?
  - How many 3-bit inputs?
  - How many permutations of the 3-bit inputs?
  - Answer: 8! = 40,320; not very many!
- In general, $2^k!$ mappings; huge for k=64
- Problem:
  - Table approach requires table with 2^{64} entries, each entry with 64 bits
- Table too big: instead use function that simulates a randomly permuted table

- Prototype Function



From Kaufman et al

- 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

- 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

- Public Key
  - RSA
    - 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* mod *z* = *1*).
5. *public* key is $\underbrace{(n,e)}_{K_B^+}$. *private* key is $\underbrace{(n,d)}_{K_B^-}$.

RSA: encryption, decryption

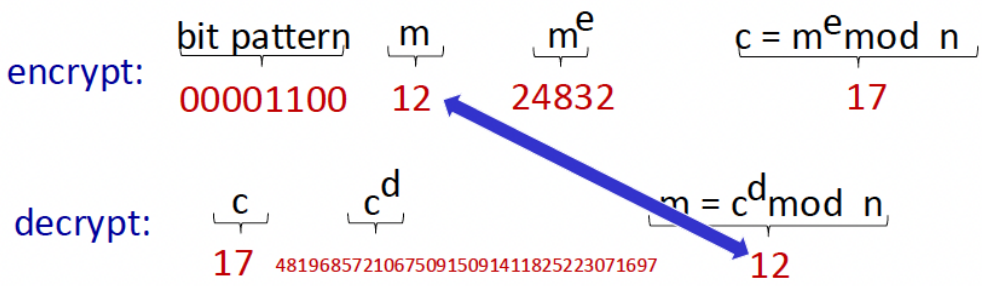
0. given (*n,e*) and (*n,d*) as computed above
1. to encrypt message *m (<n)*, compute
   $c = m^e \bmod n$
2. to decrypt received bit pattern, *c*, compute
   $m = c^d \bmod n$

magic happens! $m = (\underbrace{m^e \bmod n}_{c})^d \bmod n$

RSA example:

Bob chooses *p=5, q=7*. Then *n=35, z=24*.
e=5 (so *e, z* relatively prime).
d=29 (so *ed-1* exactly divisible by z).

encrypting 8-bit messages.



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

follows directly from modular arithmetic:

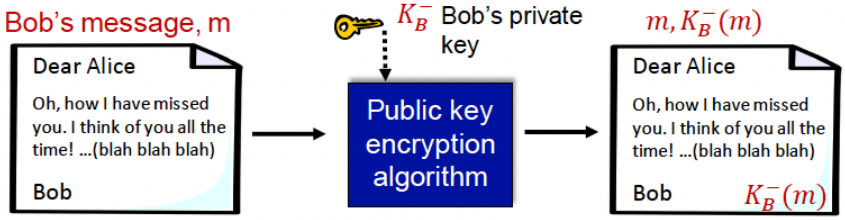
$$(m^e \bmod n)^d \bmod n = m^{ed} \bmod n$$
$$= m^{de} \bmod n$$
$$= (m^d \bmod n)^e \bmod n$$

- Authentication
- Integrity
  - Digital Signature

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 Bc 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)



- suppose Alice receives msg 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 $K_B^+ K_B^-(m) = m$.
- 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:

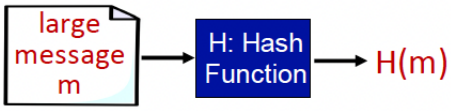
- ✓ Alice can take m, and signature $K_B^-(m)$ to court and prove that Bob signed m

▾ 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)*

Public Key Cryptography

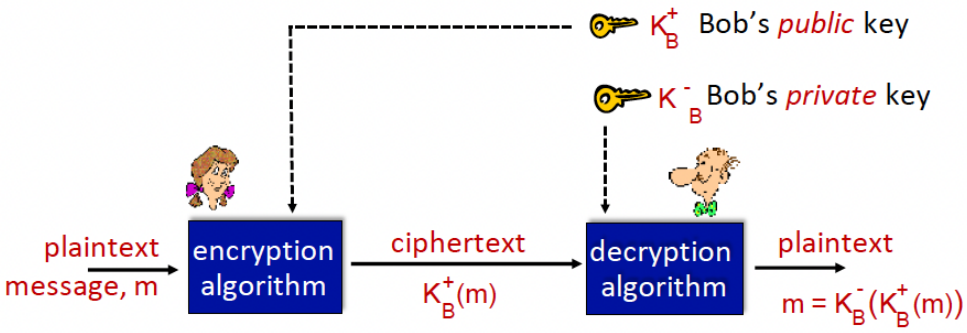
symmetric key crypto:

- requires sender, receiver know shared secret key
- Q: how to agree on key in first place (particularly if never "met")?

public key crypto

- *radically* different approach [Diffie-Hellman76, RSA78]
- sender, receiver do *not* share secret key
- *public* encryption key known to *all*
- *private* decryption key known only to receiver

Public Key Cryptography



Wow - public key cryptography revolutionized 2000-year-old (previously only symmetric key) cryptography!

- similar ideas emerged at roughly same time, independently in US and UK (classified)

- Requirements

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 computationally infeasible to determine private key K_B^-

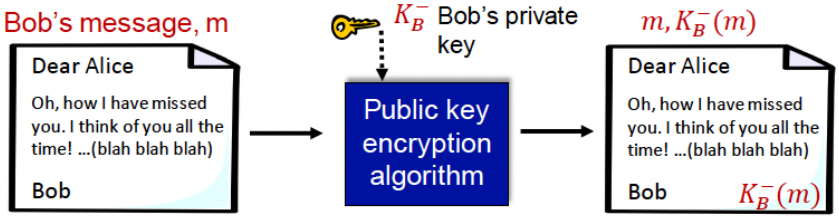
RSA: Rivest, Shamir, Adelson algorithm

- Digital Signature

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, $K_B^-(\text{m})$



Digital signatures

- suppose Alice receives msg 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 $K_B^+ K_B^-(m) = m$.
- If $K_B^+ K_B^-(m) = m$, whoever signed m must have used Bob's private key

Alice thus verifies that:

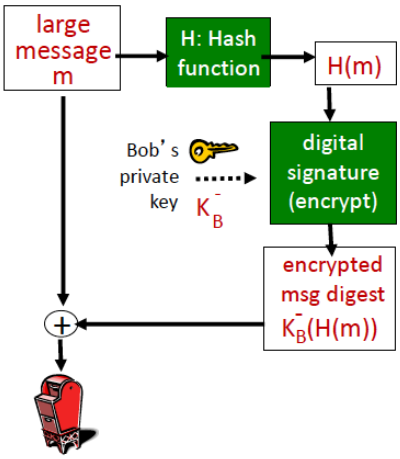
- Bob signed m
- no one else signed m
- Bob signed m and not m'

non-repudiation:

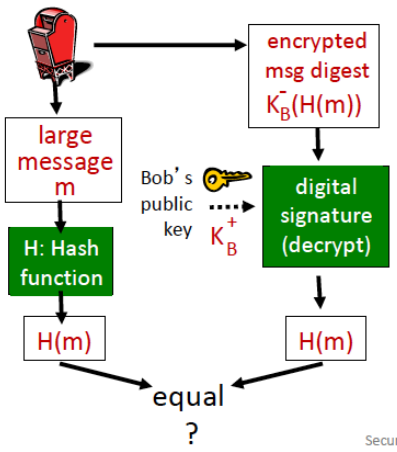
✓ Alice can take m, and signature $K_B^-(m)$ to court and prove that Bob signed m

Digital signature = signed message digest

Bob sends digitally signed message:



Alice verifies signature, integrity of digitally signed message:

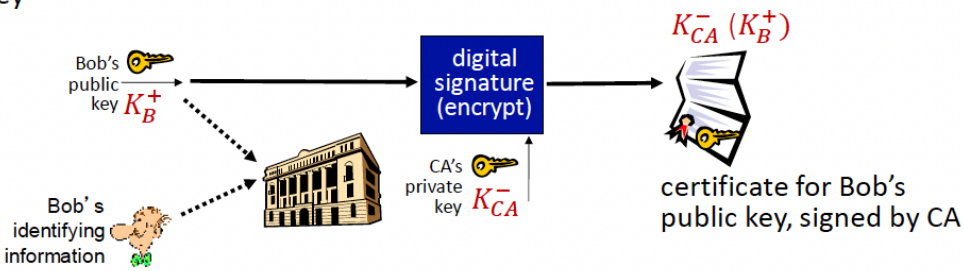


- MD5 hash function widely used (RFC 1321)
  - computes 128-bit message digest in 4-step process.
  - arbitrary 128-bit string x, appears difficult to construct msg m whose MD5 hash is equal to x
- SHA-1 is also used
  - US standard [NIST, FIPS PUB 180-1]
  - 160-bit message digest

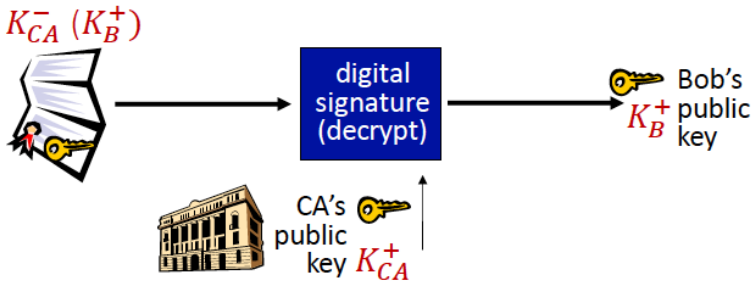
- Public Key Certification Authorities (CA)

Public key Certification Authorities (CA)

- certification authority (CA): binds public key to particular entity, E
- entity (person, website, router) registers its public key with CE provides "proof of identity" to CA
  - CA creates certificate binding identity E to E's public key
  - certificate containing E's public key digitally signed by CA: CA says "this is E's public key"



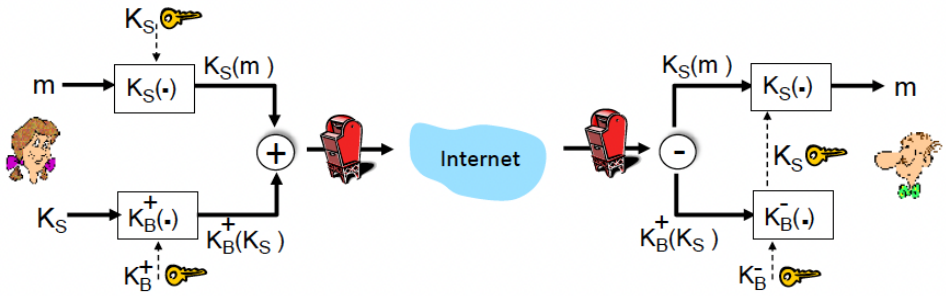
- when Alice wants Bob's public key:
  - gets Bob's certificate (Bob or elsewhere)
  - apply CA's public key to Bob's certificate, get Bob's public key



- Security in Internet Protocol Stack
  - Securing EMail

Secure e-mail: confidentiality

Alice wants to send *confidential* e-mail, m, to Bob.

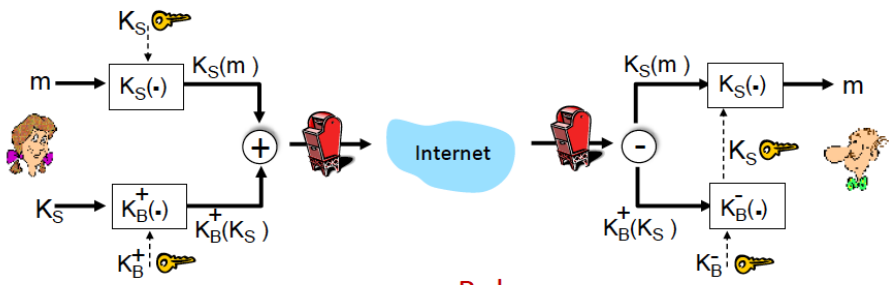


Alice:

- generates random *symmetric* private key, K_S
- encrypts message with K_S (for efficiency)
- also encrypts K_S with Bob's public key
- sends both $K_S(m)$ and $K^+_B(K_S)$ to Bob

Secure e-mail: confidentiality (more)

Alice wants to send *confidential* e-mail, m, to Bob.

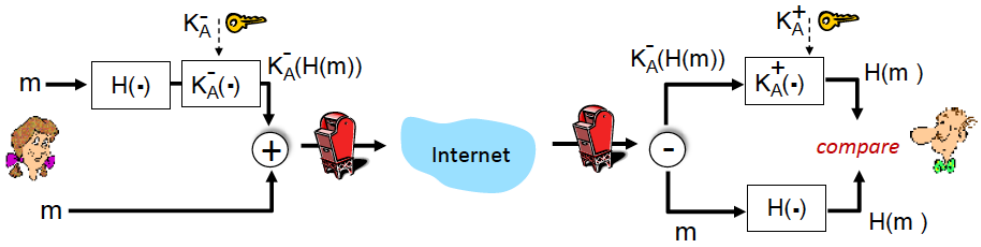


Bob:

- 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

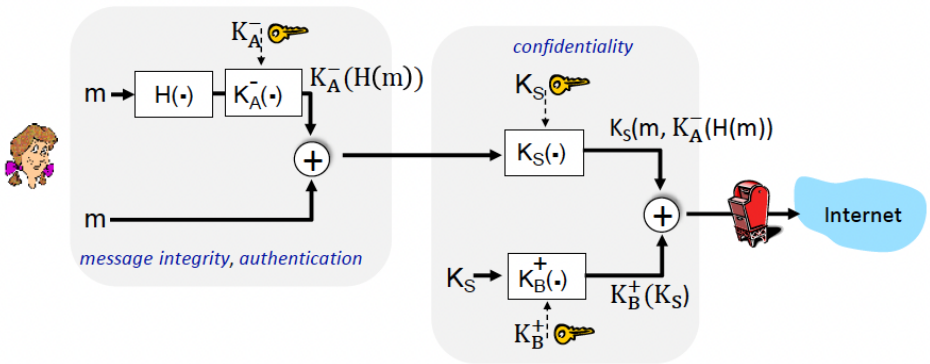
Alice wants to send m to Bob, with *message integrity*, *authentication*



- 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: integrity, authentication

Alice sends m to Bob, with *confidentiality, message integrity*, *authentication*



Alice uses three keys: her private key, Bob's public key, new symmetric key

What are Bob's complementary actions?

- TLS

Transport-layer security (TLS)

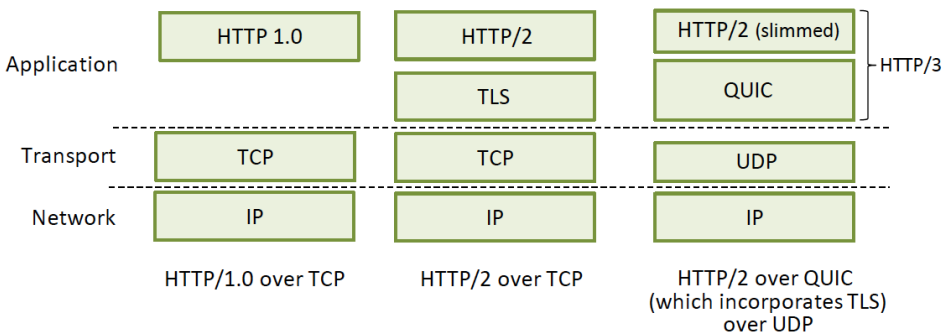
- widely deployed security protocol above the transport layer
  - supported by almost all browsers, web servers: https (port 443)
- provides:
  - confidentiality: via *symmetric encryption*
  - integrity: via *cryptographic hashing*
  - authentication: via *public key cryptography*

all techniques we have studied!

- history:
  - secure socket layer (SSL)
    - Originally designed by Netscape [1993]
    - deprecated [2015]
  - TLS 1.3: RFC 8846 [2018]

Transport-layer security (TLS)

- TLS provides an API that *any* application can use
  - Libraries for all major programming languages readily available
- an HTTP view of TLS:



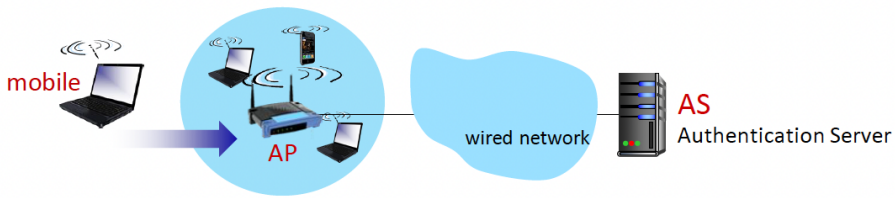
TLS encrypted browser-server connections

- Server Authentication:
  - Public keys for trusted CAs included in browser/OS
  - Browser requests server certificate, issued by trusted CA
  - Browser checks certificate and extracts public key of server

- Encrypted TLS session
  - Browser generates symmetric session key, encrypts it with server's public key, sends encrypted key to server
  - Using private private key, server decrypts session key
  - All data sent into TCP socket (by client and server) encrypted with session key.

- WIFI Security

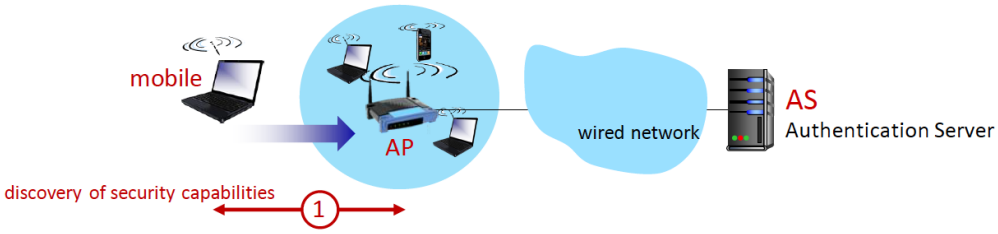
802.11: authentication, encryption



Arriving mobile must:

- associate with access point: (establish) communication over wireless link
- authenticate to network

802.11: authentication, encryption

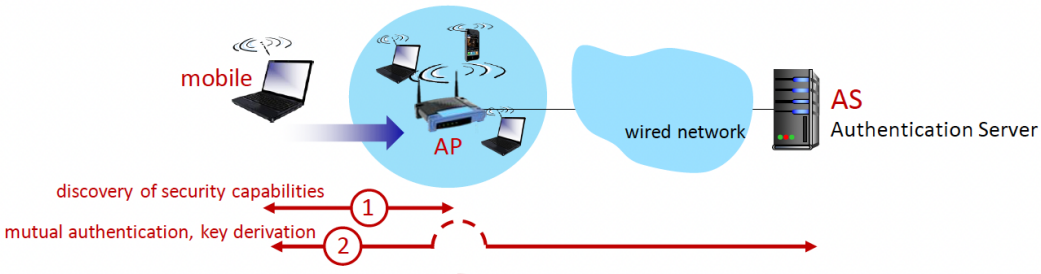


1 discovery of security capabilities:

- AP advertises its presence, forms of authentication and encryption provided
- device requests specific forms authentication, encryption desired

although device, AP already exchanging messages, device not yet authenticated, does not have encryption keys

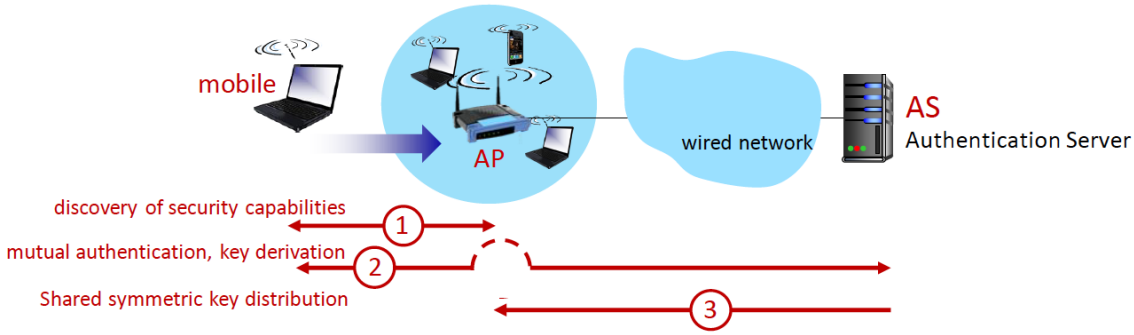
802.11: authentication, encryption



2 mutual authentication and shared symmetric key derivation:

- AS, mobile already have shared common secret (e.g., password)
- AS, mobile use shared secret, nonces (prevent relay attacks), cryptographic hashing (ensure message integrity) to authenticating each other
- AS, mobile derive symmetric session key

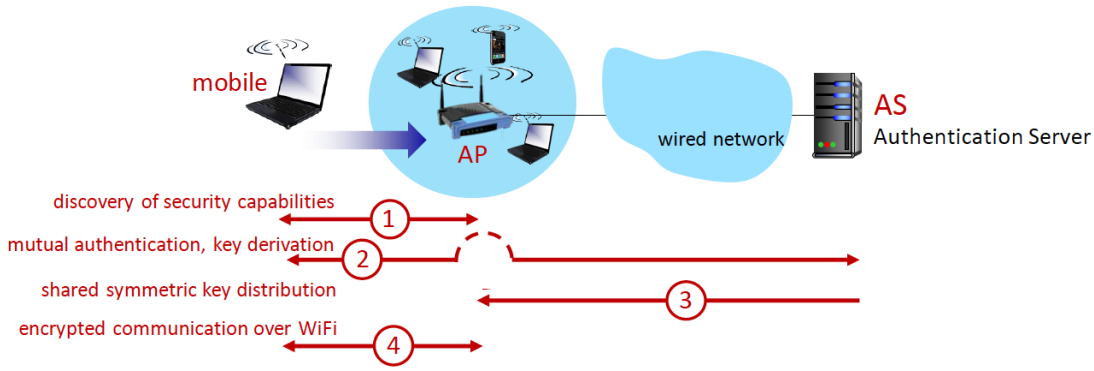
802.11: authentication, encryption



3 shared symmetric session key distribution (e.g., for AES encryption)

- same key derived at mobile, AS
- AS informs AP of the shared symmetric session

802.11: authentication, encryption



4 encrypted communication between mobile and remote host via AP