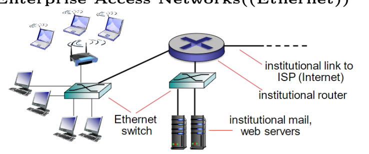
1 Introduction

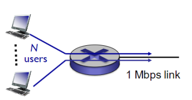
Internet: Billions of connected computing devices,
Protocol: Protocols define format, order of messages sent and received among network entities, and actions taken on message transmission, receipt. (control sending, receiving of messages e.g., TCP, IP, HTTP, Skype, 802.11)
Network edge: have 1) hosts: clients and servers. 2) access networks, physical media: wired, wireless communication links, (residential access nets, institutional access nets, mobile access networks)
Digital Subscriber Line: voice, data transmitted at different frequencies over **dedicated** line to central office Use existing telephone line to central office DSLAM, data over DSL phone line goes to Internet, voice over DSL phone line goes to telephone net, $\le$ 2.5 Mbps upstream transmission rate (typically $\le$ 1 Mbps), $\le$ 24 Mbps downstream transmission rate (typically $\le$ 10 Mbps)
Cable Network: Uses HFC: hybrid fiber coax, asymmetric: up to 30Mbps downstream transmission rate, 2Mbps upstream transmission rate, Network of cable, fiber attaches homes to ISP router, homes share access network to cable headend, unlike DSL, which has dedicated access to central officedifferent channels transmitted in different frequency bands (FDM)
Enterprise Access Networks((Ethernet))



- Typically used in companies, universities, etc.
- 10 Mbps, 100Mbps, 1Gbps, 10Gbps transmission rates
- Today, end systems typically connect into Ethernet switch

Wireless Access Networks: Shared wireless access network connects end system to router via base station aka "access point"
Wireless LANs:within building (100 ft.), 802.11b/g/n (WiFi): 11,54,450 Mbps transmission rate
Wide-area wireless access: provided by telco (cellular) operator, 10's km, between 1 and 10 Mbps, 3G, 4G: LTE
Host:Host sending function: takes application message, breaks into smaller chunks known as packets, of length L bits, transmits packet into access network at transmission rate R, link transmission rate, aka link capacity, aka link bandwidth
Physical Media: bit: propagates between transmitter/receiver pairs, physical link: what lies between transmitter & receiver, guided media: signals propagate in solid media: copper, fiber, coax, Fiber optic cable, unguided media: signals propagate freely (Terrestrial microwave, LAN, Wide-area, Satellite), twisted pair (TP) two insulated copper wires
Network core: interconnected routers, network of networks
Packet-switching: takes L/R seconds to transmit (push out) L-bit packet into link at R bps, **store and forward**: entire packet must arrive at router before it can be transmitted on next link, end-end delay = 2L/R (assuming zero propagation delay)
Queuing and Loss: if arrival rate (in bits) to link exceeds transmission rate of link for a period of time: packets will queue, wait to be transmitted on link, packets can be dropped (lost) if memory (buffer) fills up 2nd width (bps), L: packet length(bits), a: average packet arrival rate. La/R ∼ 0: avg. queueing delay small, La/R → 1: avg. queueing delay large, La/R ¿ 1: more "work" arriving than can be serviced, average delay infinite!
Circuit Switching: end-end resources allocated to, **reserved** for "call" between source & destination: dedicated resources: no sharing, circuit segment idle if not used by call (no sharing), commonly used in traditional telephone networks

Example:
- 1 Mb/s link
- Each user:
  - 100 kb/s when "active"
  - active 10% of time

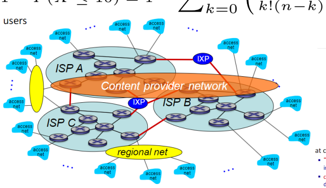


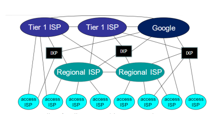
- *Circuit-switching:*
  - 10 users
- *Packet switching:*
  - with 35 users, probability > 10 active at same time is less than .0004

Q: how did we get value 0.0004?
Q: what happens if > 35 users ?

$$1 - P(X \le 10) = 1 - \sum_{k=0}^{10} \left(\frac{n!}{k!(n-k)!}\right) p^k (1-p)^{n-k}$$

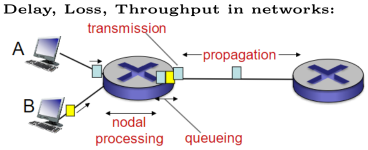
users



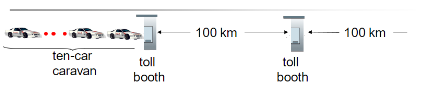


at center: small # of well-connected large networks
- "tier-1" commercial ISPs (e.g., Level 3, Sprint, AT&T, NTT), national & international coverage
- content provider network (e.g., Google): private network that connects it data centers to Internet, often bypassing tier-1, regional ISPs

Delay, Loss, Throughput in networks:



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



1) cars "propagate" at 100 km/hr 2)toll booth takes 12 sec to service car (bit transmission time) 3) car b̃it; caravan ; time to "push" entire caravan through toll booth onto highway: 12x10 = 120 sec, time for last car to propagate from 1st to 2nd toll both: 100km/(100km/hr)= 1 hr, i.e. 60 + 2 = 62 min
Queueing Delay: **R**: link bandwidth (bps), **L**: packet length (bits), **a**: average packet arrival rate. La/R ∼ 0: avg. queueing delay small, La/R → 1: avg. queueing delay large, La/R > 1: more "work" arriving than can be serviced, average delay infinite!
Traceroute analysis:

traceroute: gaia.cs.umass.edu to www.eurecom.fr

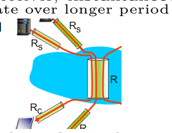
3 delay measurements from gaia.cs.umass.edu to cs-gw.cs.umass.edu

```
1  cs-gw (128.119.240.254)  1 ms  1 ms  2 ms
2  border1-rt-fa5-1-0.gw.umass.edu (128.119.3.145)  1 ms  1 ms  2 ms
3  cht-vbns.gw.umass.edu (128.119.3.130)  6 ms 5 ms 5 ms
4  jn1-at1-0-0-19.wor.vbns.net (204.147.132.129)  16 ms 11 ms 13 ms
5  jn1-so7-0-0-0.wae.vbns.net (204.147.136.136)  21 ms 18 ms 18 ms
6  abilene-vbns.abilene.ucaid.edu (198.32.11.9)  22 ms  18 ms  22 ms
7  nycm-wash.abilene.ucaid.edu (198.32.8.46)  22 ms  22 ms  22 ms
8  62.40.103.253 (62.40.103.253)  104 ms 109 ms 106 ms
9  de2-1.de1.de.geant.net (62.40.96.129)  109 ms 102 ms 104 ms
10  de.fr1.fr.geant.net (62.40.96.50)  113 ms 121 ms 114 ms
11  renater-gw.fr1.fr.geant.net (62.40.103.54)  112 ms  114 ms  112 ms
12  nio-n2.cssi.renater.fr (193.51.206.13)  111 ms  114 ms  116 ms
13  nice.cssi.renater.fr (195.220.98.102)  123 ms  125 ms  124 ms
14  r3t2-nice.cssi.renater.fr (195.220.98.110)  126 ms  126 ms  124 ms
15  eurecom-valbonne.r3t2.ft.net (193.48.50.54)  135 ms  128 ms  133 ms
16  194.214.211.25 (194.214.211.25)  126 ms  128 ms  126 ms
17  * * *
18  * * *
19  fantasia.eurecom.fr (193.55.113.142)  132 ms  128 ms  136 ms
```

trans-oceanic link (hop 7 → 8)
* means no response (probe lost, router not replying)

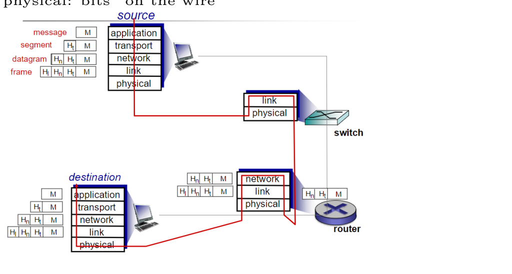
Packet Loss: queue (aka buffer) preceding link in buffer has finite capacity, packet arriving to full queue dropped (aka lost), lost packet may be retransmitted by previous node, by source end system, or not at all
Throughput: rate (bits/time unit) at which bits transferred between sender/receiver, **instantaneous**: rate at given point in time, **average**: rate over longer period of time

per-connection end-end throughput: $min(R_c,R_s,R/10)$



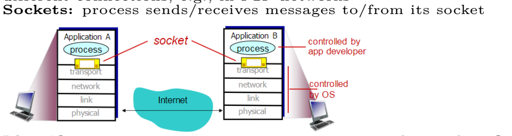
in practice: R_c or R_s is often bottleneck

Internet protocol stack: application: supporting network applications (FTP, SMTP, HTTP) transport: process-process data transfer (TCP, UDP), network: routing of packets from sourceto destination (IP, routing protocols), link: data transfer between neighboring network elements (Ethernet, 802.11 (WiFi), PPP), physical: bits "on the wire"



2 Application Layer

Client-server: 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 architecture:no always-on server, arbitrary end systems directly communicate, peers request service from other peers, provide service in return to other peers, self scalability – new peers bring new service capacity, as well as new service demands, Peers are intermittently connected and change IP addresses → Complex Management.
Processes communicating: processes in different hosts communicate by exchanging messages, same process can be both a client (initiates communication) and a server(waits to be contacted) for different connections; e.g., in P2P networks
Sockets: process sends/receives messages to/from its socket



Identifier: to receive messages, process must have identifier, identifier includes both IP address and port numbers associated with process on host.
App-layer protocol: defines types of messages exchanged, message syntax, message semantics, rules for when and how processes send & respond to messages

application	data loss	throughput	time sensitive
file transfer	no loss	elastic	no
e-mail	no loss	elastic	no
Web documents	no loss	elastic	no
real-time audio/video	loss-tolerant	audio: 5kbps-1Mbps video:10kbps-5Mbps	yes, 100's msec yes, few secs
stored audio/video	loss-tolerant	same as above	yes, 100's ms
interactive games	loss-tolerant	few kbps up	yes and no
text messaging	no loss	elastic	no

Socket programming: Two socket types for two transport services: UDP: unreliable datagram (User Datagram Protocol) TCP: reliable, byte stream-oriented (Transmission Control Protocol)

Socket programming *with UDP*

UDP: no "connection" between client & server
- no handshaking before sending data
- sender explicitly attaches IP destination address and port # to each packet
- receiver extracts sender IP address 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

Socket programming *with 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

HTTP: hypertext transfer 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 Steps: 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.

Non-persistent HTTP

suppose user enters URL:
www.someSchool.edu/someDepartment/home.index
(contains text, references to 10 jpeg images)

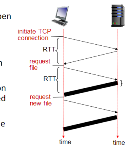
1a. HTTP client initiates TCP connection to HTTP server (process) at www.someSchool.edu on port 80
1b. HTTP server at host www.someSchool.edu waiting for TCP connection at port 80. "accepts" connection, notifying client
2. HTTP client sends HTTP *request message* (containing URL) into TCP connection socket. Message indicates that client wants object someDepartment/home.index
3. HTTP server receives request message, forms *response message* containing requested object, and sends message into its socket
4. HTTP server closes TCP connection.
5. HTTP client receives response message containing html file, displays html. Parsing html file, finds 10 referenced jpeg objects

time

REPEAT 10 TIMES FOR 10 IMG

HTTP response time: one RTT to initiate TCP connection + one RTT for HTTP request and first few bytes of HTTP response to return + file transmission time = non-persistent HTTP response time = 2RTT+ file transmission time

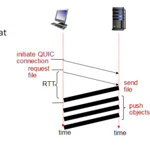
Persistent HTTP:
- server leaves connection open after sending response
- subsequent HTTP messages between same client/server sent over open connection
- client sends requests as soon as it encounters a referenced object
- as little as one RTT for all the referenced objects



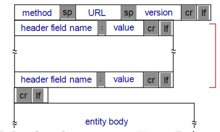
Pipelining
- Send several requests at once

HTTP/2
- Push resources

QUIC
- Eliminate first RTT



HTTP request message: two types of HTTP messages: request, response



request line (GET, POST, HEAD commands)
header lines
carriage return, line feed at start of line indicates end of header lines
carriage return character
line-feed character

```
GET /index.html HTTP/1.1\r\n
Host: www-net.cs.umass.edu\r\n
User-Agent: Firefox/3.6.10\r\n
Accept: text/html,application/xhtml+xml\r\n
Accept-Language: en-us,en;q=0.5\r\n
Accept-Encoding: gzip,deflate\r\n
Accept-Charset: ISO-8859-1,utf-8;q=0.7\r\n
Keep-Alive: 115\r\n
Connection: keep-alive\r\n
\r\n
```

Method types: HTTP/1.0: GET, POST, HEAD (asks server to leave requested object out of response) HTTP/1.1: GET, POST, HEAD, PUT, (uploads file in entity body to path specified in URL field), DELETE (deletes file specified in the URL field)

status line (protocol status code status phrase)
header lines
data, e.g., requested HTML file

```
HTTP/1.1 200 OK\r\n
Date: Sun, 26 Sep 2010 20:09:20 GMT\r\n
Server: Apache/2.0.52 (CentOS)\r\n
Last-Modified: Tue, 30 Oct 2007 17:00:02 GMT\r\n
ETag: "17dc6-a5c-bf716880"\r\n
Accept-Ranges: bytes\r\n
Content-Length: 2652\r\n
Keep-Alive: timeout=10, max=100\r\n
Connection: Keep-Alive\r\n
Content-Type: text/html; charset=ISO-8859-1\r\n
\r\n
data data data data data ...
```

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

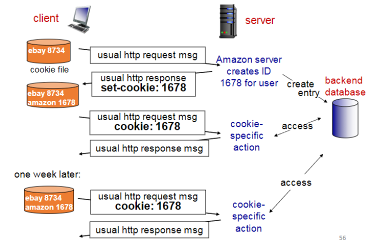
301 Moved Permanently
- requested object moved, new location specified later in this msg (Location:)

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: four components: 1) C ookie 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 Website

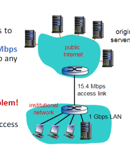


Web caches (proxy server) user sets browser: Web accesses via cache. browser sends all HTTP requests to cache. object in cache: cache returns object. else cache requests object from origin server, then returns object to client

assumptions:
- avg object size: 1 Mbit
- avg request rate from browsers to origin servers: 15/sec
- avg data rate to browsers: 15 Mbps
- RTT from institutional router to any origin server: 2 sec
- access link rate: 15.4 Mbps

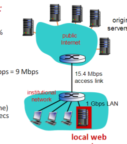
consequences:
- LAN utilization: 1.5%
- access link utilization = 97%
- total delay = Internet delay + access delay + LAN delay = 2 sec + minutes + usecs

Problem!

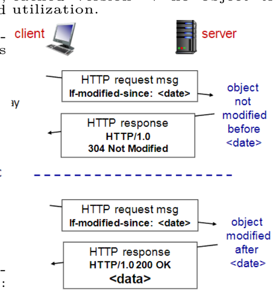


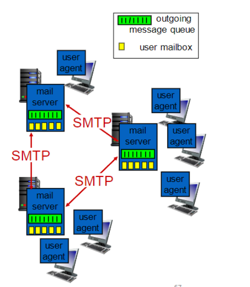
Calculating access link utilization, delay with cache:
- suppose cache hit rate is 0.4
  - 40% requests satisfied at cache, 60% requests satisfied at origin
- access link utilization:
  - 60% of requests use access link
  - data rate to browsers = 0.6*15 Mbps = 9 Mbps
  - utilization = 9/15.4 = 0.58 → 58%
- total delay
  = 0.6 * (delay from origin servers)
  + 0.4 * (delay when satisfied at cache)
  = 0.6 (2.01) + 0.4 (~msecs) = ~ 1.2 secs
  < less than with 100 Mbps link

Cost: web cache (cheap!)

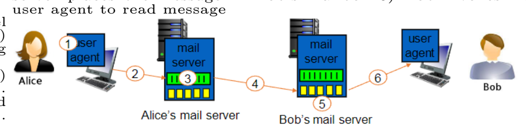


Conditional GET Goal: don't send object if cache has up-to-date cached version → no object transmission delay → lower link utilization.





Electronic mail Three major components: 1)user agents 2) mail servers 3) SMTP: Simple Mail Transfer Protocol
User Agent: a.k.a. "mail reader", composing, editing, reading mail messages, e.g., Outlook, Thunderbird, iPhone mail client, outgoing, incoming messages stored on server **Mail Servers**: 1) mailbox contains incoming messages for user 2) 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).
SMTP Example: 1) Alice uses UA to compose message "to" bob@someschool.edu 2) Alice's UA sends message to her mail server; message placed in message queue 3) client side of SMTP opens TCP connection with Bob's mail server 4) SMTP client sends Alice's message over the TCP connection 5) Bob's mail server places the message in Bob's mailbox 6) Bob invokes his user agent to read message

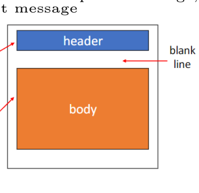


SMTP uses persistent connections, SMTP requires message (header & body) to be in 7- bit ASCII, SMTP server uses CRLF.CRLF to determine end of message
comparison with HTTP: HTTP: pull, SMTP: push. both have ASCII command/response interaction, status codes. HTTP: each object encapsulated in its own response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

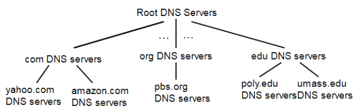
SMTP: protocol for exchanging email messages

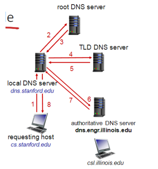
RFC 822: standard for text message format:
- header lines, e.g.,
  - To:
  - From:
  - Subject:

  different from SMTP MAIL FROM, RCPT TO: commands!
- Body: the "message"
  - ASCII characters only

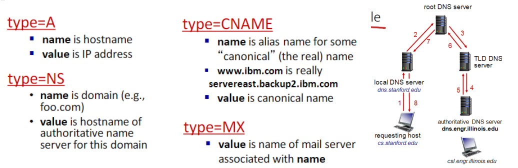


Mail access protocols retrieval from server: POP: Post Office Protocol [RFC 1939]: authorization, download. IMAP: Internet Mail Access Protocol [RFC 1730]: more features, including manipulation of stored messages on server. HTTP: gmail, Hotmail, Yahoo! Mail, etc.
DNS: domain name system: DNS services: 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
client wants IP for www.amazon.com; 1 st approximation: 1) client queries root server to find com DNS server 2) client queries .com DNS server to get amazon.com DNS server 3) client queries amazon.com DNS server to get IP address for www.amazon.com

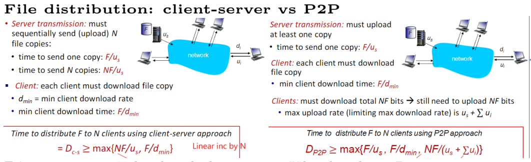




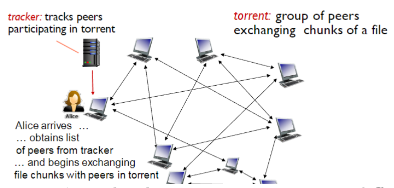
Root Name Servers: Contacts authoritative name server if name mapping not known Gets mapping, Returns mapping to local name server.
Top-level domain (TLD) servers: responsible for com, org, net, edu, aero, jobs, museums, and all top-level country domains, e.g.: uk, fr, ca, jp, Network Solutions maintains servers for .com TLD, Educause for .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



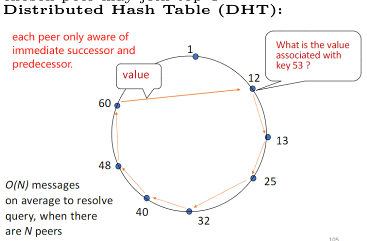
File distribution: client-server vs P2P

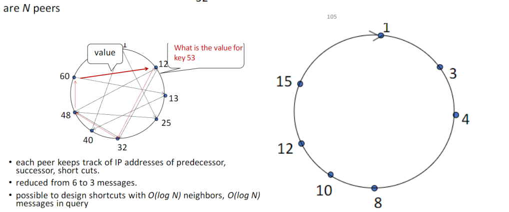


BitTorrent: File divided into 256Kb chunks. Peers in torrent send/receive file chunks. peer joining torrent has no chunks, but will accumulate them over time from other peers, registers with tracker to get list of peers, connects to subset of peers ("neighbors") while downloading, peer uploads chunks to other peers, peer may change peers with whom it exchanges chunks. churn: peers may come and go. once peer has entire file, it may (selfishly) leave or (altruistically) remain in torrent.

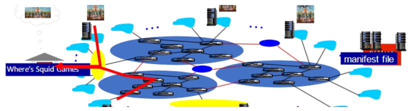


requesting chunks: at any given time, different peers have different subsets of file chunks, periodically, Alice asks each peer for list of chunks that they have, Alice requests missing chunks from peers, rarest first
sending chunks tit-for-tat: Alice sends chunks to those **four peers** currently sending her chunks at highest rate, other peers are choked by Alice (do not receive chunks from her), re-evaluate **top 4** every 10 secs. every 30 secs, randomly select another peer, starts sending chunks, "optimistically unchoke" this peer, newly chosen peer may join top 4
Distributed Hash Table (DHT):



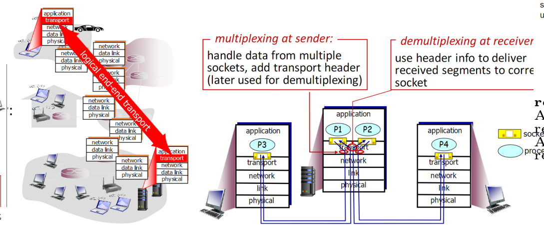


handling peer churn: peers may come and go (churn),each peer knows address of its two successors, each peer periodically pings its two successors to check aliveness, if immediate successor leaves, choose next successor as new immediate successor.
example: peer 5 abruptly leaves,peer 4 detects peer 5's departure; makes 8 its immediate successor, 4 asks 8 who its immediate successor is; makes 8's immediate successor its second successor.
Video Streaming: challenge: scale - how to reach $\sim$ 1B users? heterogeneity different users have different capabilities
Multimedia: video CBR: (constant bit rate): video encoding rate fixed, VBR: (variable bit rate): video encoding rate changes as amount of spatial, temporal coding changes. **spatial coding** example: instead of sending N values of same color (all purple), send only two values: color value (purple) and number of repeated values (N), **temporal coding example:** instead of sending complete frame at i+1, send only differences from frame i
DASH: Dynamic, Adaptive Streaming over HTTP server: divides video file into multiple chunks, each chunk stored, encoded at different rates, manifest file: provides URLs for different chunks, **client**:, periodically measures server-to-client bandwidth, consulting manifest, requests one chunk at a time, chooses maximum coding rate sustainable given current bandwidth, can choose different coding rates at different points in time (depending on available bandwidth at time), "**intelligence**" at client: client determines 1) when to request chunk (so that buffer starvation, or overflow does not occur), 2)what encoding rate to request (higher quality when more bandwidth available), 3)where to request chunk (can request from URL server that is "close" to client or has high available bandwidth) 125
Content Distribution Networks (CDNs): stores copies of content at CDN nodes, subscriber requests content from CDN (directed to nearby copy, retrieves content may choose different copy if network path congested)

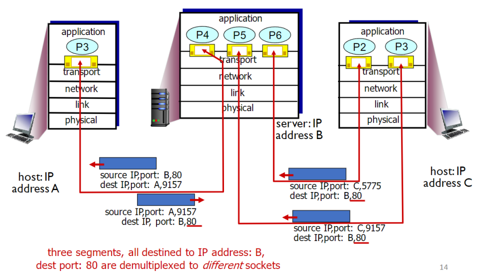


3 Transport Layer

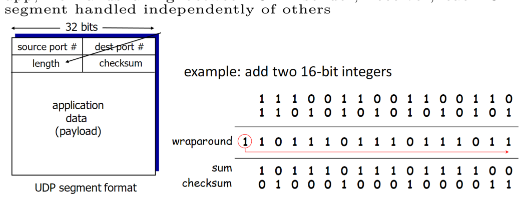
Transport services and protocols provide logical communication between app **processes** running on different hosts. **send side:** breaks app messages into segments, passes to network layer, **rcv side:** reassembles segments into messages, passes to app layer, more than one transport protocol available to apps (TCP and UDP)
Multiplexing/Demultiplexing



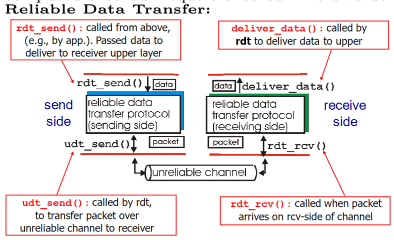
Demultiplexing: host receives IP datagrams, each datagram has source IP address, destination IP address, each datagram carries one transport-layer segment, each segment has **source, destination port number**, host uses IP addresses & port numbers to direct segment to appropriate socket
Connection-oriented demux:



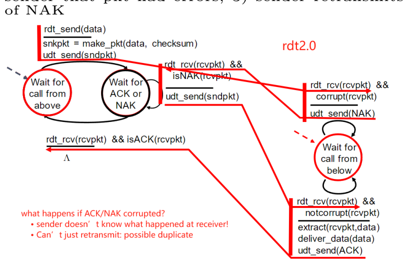
TCP socket identified by 4-tuple: source IP address, source port number, dest IP address, dest port number TCP socket identified by 4-tuple:, source IP address, source port number, dest IP address, dest port number **Connectionless UDP:** "no frills," "bare bones" Internet transport protocol, "best effort" service, UDP segments may be:, lost, delivered out-of-order to app, no handshaking between UDP sender, receiver, each UDP segment handled independently of others



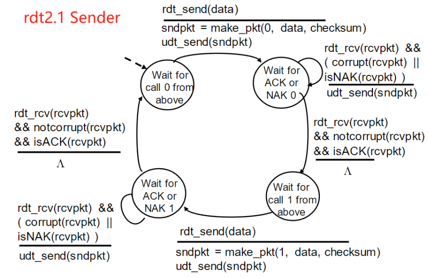
UDP checksum:detect "errors" in transmitted segment. **sender:** treat segment contents, including header fields, as sequence of 16-bit integers, checksum: addition (one's complement sum) of segment contents, sender puts checksum value into UDP checksum field. **receiver:** compute checksum of received segment, check if computed checksum equals checksum field value.
Reliable Data Transfer:

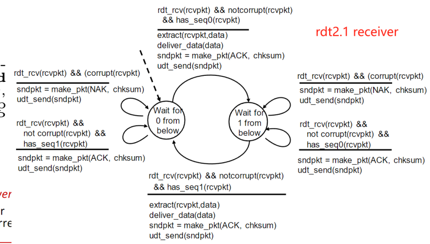


rdt1.0: Reliable transfer over a reliable channel. underlying channel perfectly reliable, no bit errors, no loss of packets, separate FSMs for sender, receiver: sender sends data into underlying channel, receiver reads data from underlying channel.
rdt2.0: Channel with bit errors. Uses 1) acknowledgements (ACKs): receiver explicitly tells sender that pkt received OK. 2) negative acknowledgements (NAKs): receiver explicitly tells sender that pkt had errors, 3) sender retransmits pkt on receipt of NAK

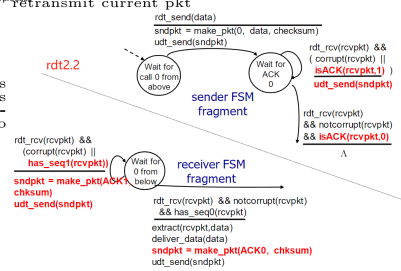


rdt2.1: Sender, handles garbled ACK/NAKs, Handles bit corruptions that are detected by checksum, Uses a 1-bit sequence number to detect retransmission at receiver

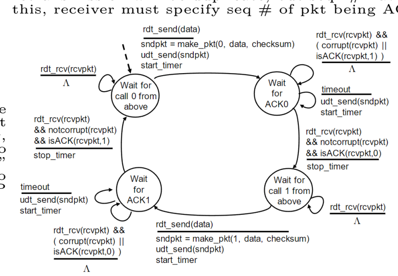


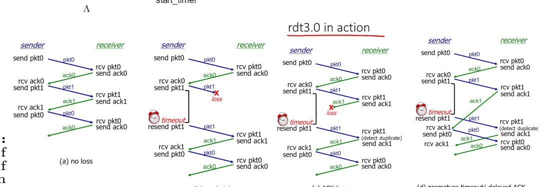


rdt2.2: A NAK-free protocol, same functionality as rdt2.1, using ACKs only. instead of NAK, receiver sends ACK for last pkt received OK (receiver must explicitly include seq # of pkt being ACKed), duplicate ACK at sender results in same action as NAK: retransmit current pkt

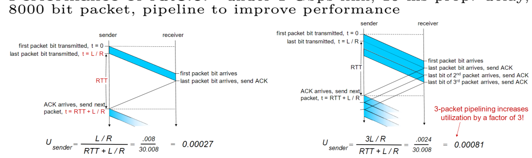


rdt3.0: Channels with errors and loss of packets (data, ACKs). **approach:** 1) sender waits "reasonable" amount of time for ACK, 2) retransmits if no ACK received in this time, 3) requires countdown timer, 4) if pkt (or ACK) just delayed (not lost): retransmission will be duplicate, but seq. # have already handles this, receiver must specify seq # of pkt being ACKed

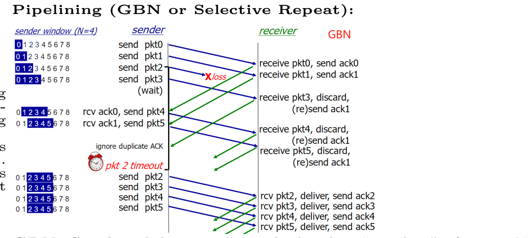




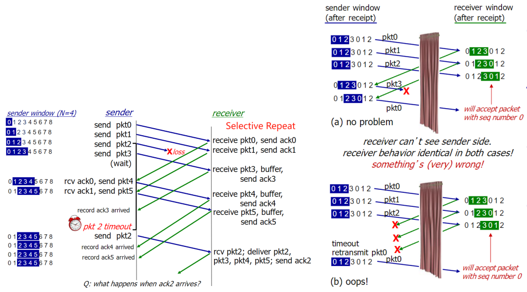
Performance of rdt3.0: under 1 Gbps link, 15 ms prop. delay, 8000 bit packet, pipeline to improve performance



Pipelining (GBN or Selective Repeat):



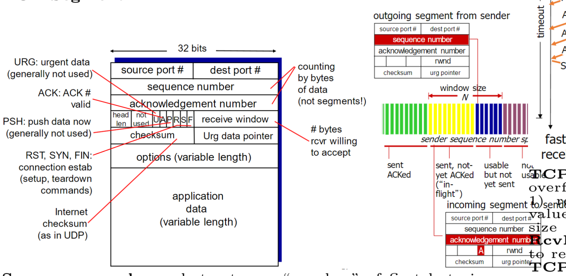
GBN: Sender: k-bit seq # in pkt header, "window" of up to N, consecutive unack'ed pkts allowed. ACK(n): ACKs all pkts up to, including seq # n – "cumulative ACK" timer for oldest in-flight pkt. timeout(n): retransmit packet n and all higher seq # pkts in window. **Receiver:** ACK-only: always send ACK for correctly-received pkt with highest in-order seq #, may generate duplicate ACKs, need only remember expectedseqnum, out-of-order pkt:, discard (don't buffer): no receiver buffering! re-ACK pkt with highest in-order seq #



What relationship between seq # size and window size to avoid problem in (b)? In technical terms, if N is the size of the sequence number space and W is the window size, the condition to avoid such problems is: N>2W

TCP Overview: 1) full duplex data (bi-directional data flow in same connection, MSS: maximum segment size) 2)connection-oriented (handshaking inits sender, receiver state before data exchange) 3) flow controlled (sender will not overwhelm receiver) 4) point-to-point (one sender, one receiver) 5) reliable, in-order byte steam (no "message boundaries"), 6) pipelined (TCP congestion and flow control set window size)

TCP Segment



Sequence numbers: byte stream "number" of first byte in segment's data acknowledgements: (seq # of next byte expected from other side, cumulative ACK).

TCP timeout interval: EstimatedRTT plus "safety margin"

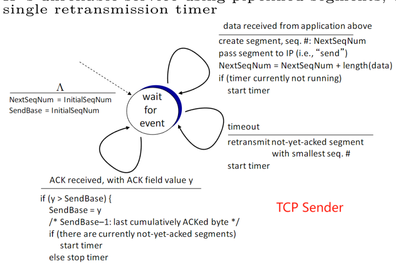
$$\text{DevRTT} = (1-\beta)\times\text{DevRTT} + \beta\times|\text{SampleRTT} - \text{EstimatedRTT}|$$

(typically, β = 0.25)

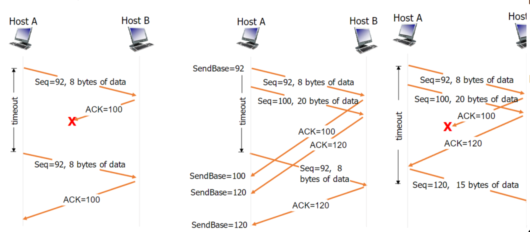
$$\text{TimeoutInterval} = \text{EstimatedRTT} + 4\times\text{DevRTT}$$

estimated RTT — "safety margin"

TCP reliable data transfer TCP creates rdt service on top of IP's unreliable service using pipelined segments, cumulative acks, single retransmission timer

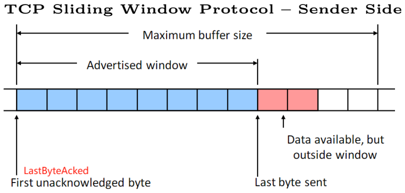


TCP Sender

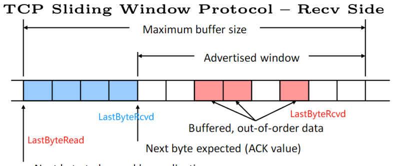


lost ACK scenario — TCP Retransmission — premature timeout — cumulative ACK

TCP Sliding Window Protocol – Sender Side



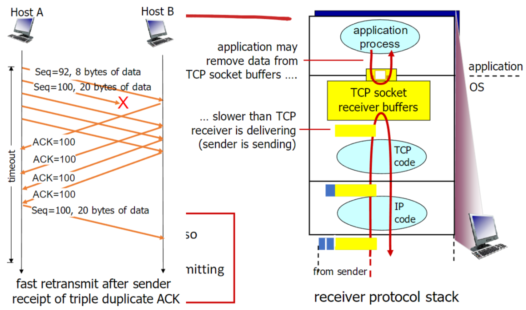
TCP Sliding Window Protocol – Recv Side



TCP ACK generation:

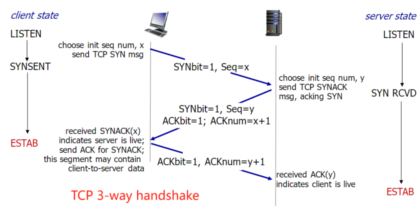
event at receiver	TCP receiver action
arrival of in-order segment with expected seq #. All data up to expected seq # already ACKed	delayed ACK. Wait up to 500ms for next segment. If no next segment, send ACK
arrival of in-order segment with expected seq #. One other segment has ACK pending	immediately 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: if sender receives 3 ACKs for same data ("triple duplicate ACKs"), resend unacked segment with smallest seq #. (likely that unacked segment lost, so don't wait for timeout)

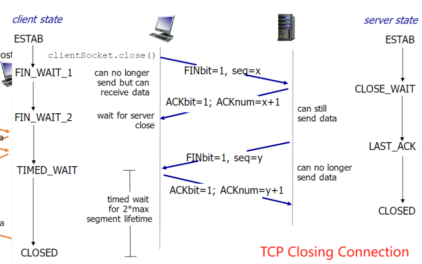


TCP flow control: receiver controls sender, so sender won't overflow receiver's buffer by transmitting too much, too fast. 1) receiver "advertises" free buffer space by including **rwnd** value in TCP header of receiver-to-sender segments, (**RcvBuffer** size set via socket options, many operating systems autoadjust **RcvBuffer**) 2) sender limits amount of unacked ("in-flight") data to receiver's rwnd value.

TCP Connection Management:



TCP 3-way handshake

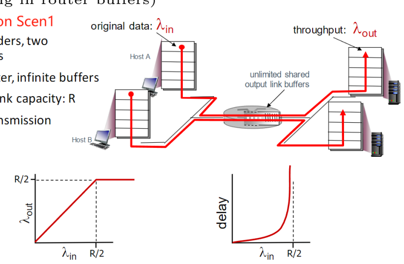


TCP Closing Connection

Congestion control: "too many sources sending too much data too fast for network to handle", different from flow control! manifestations: lost packets (buffer overflow at routers), long delays (queueing in router buffers)

Congestion Scen1

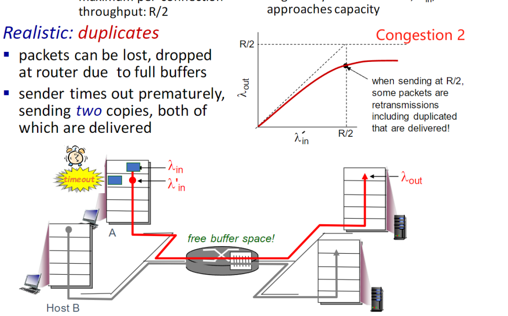
- two senders, two receivers
- one router, infinite buffers
- output link capacity: R
- no retransmission



- maximum per-connection throughput: R/2
- large delays as arrival rate, λ_{in}, approaches capacity

Realistic: duplicates — Congestion 2

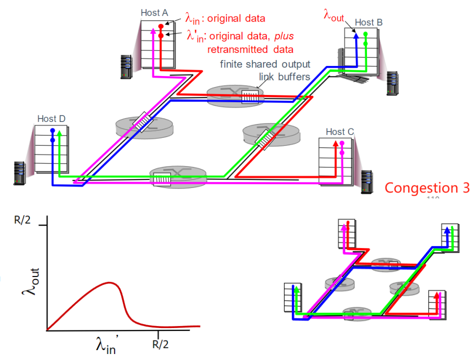
- packets can be lost, dropped at router due to full buffers
- sender times out prematurely, sending *two* copies, both of which are delivered



- four senders
- multihop paths
- timeout/retransmit

Q: what happens as λ_{in} and λ_{in}' increase ?

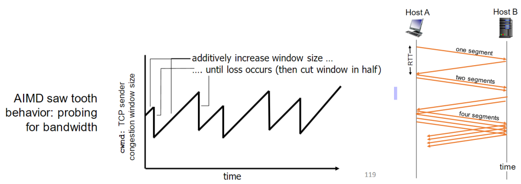
A: as red λ_{in}' increases, all arriving blue pkts at upper queue are dropped, blue throughput → 0



Congestion 3

TCP Congestion Control: CWND sender limits transmission to *cwnd*, cwnd is dynamic, function of perceived network congestion. Thus: TCP sending rate roughly: send cwnd bytes, wait RTT for ACKS, then send more bytes $rate = \frac{cwnd}{RTT}$ bytes/sec

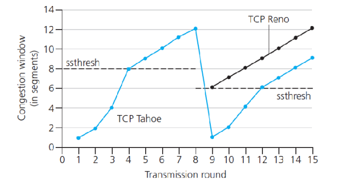
TCP Slow Start: 1) initially cwnd = 1 MSS, 2) double cwnd every RTT, (done by incrementing cwnd for every ACK received up to some threshold) 3) AIMD: Additive Increase Multiplicative Decrease.

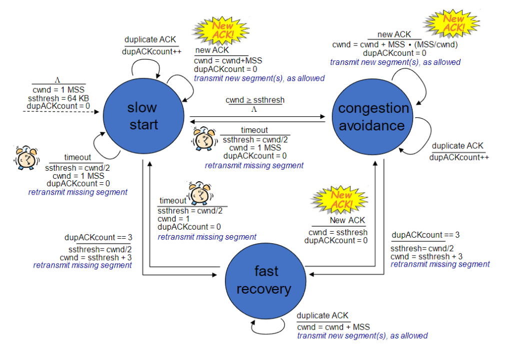


cwnd when detecting, reacting to loss: Loss indicated by timeout: cwnd set to 1 MSS;, window then grows exponentially (as in slow start) to threshold, then grows linearly (as in additive increase), Loss indicated by 3 duplicate ACKs:, **TCP Tahoe:** always sets cwnd to 1 (timeout or 3 duplicate acks), **TCP Reno:** (Fast Recovery), dup ACKs indicate network capable of delivering some segments, cwnd is cut in half window then grows linearly (as in additive increase)

Implementation:

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

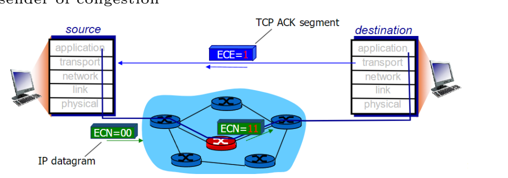




TCP throughput: avg. TCP throughput as function of window size, RTT? ignore slow start, assume always data to send, W: window size (measured in bytes) where loss occurs, avg. window size (# in-flight bytes) is 3/4 W, avg. TCP throughput = $\frac{3/4W}{RTT}$

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

Explicit Congestion Notification: network-assisted congestion control: two bits in IP header (ToS field) marked by network router to indicate congestion, congestion indication carried to receiving host, receiver (seeing congestion indication in IP datagram)) sets ECE bit on receiver-to-sender ACK segment to notify sender of congestion

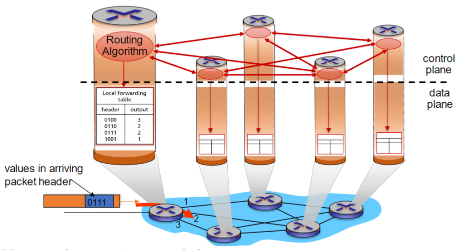


4 Network Layer

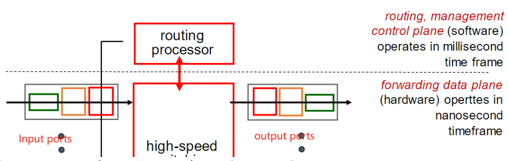
Network-layer functions: 1) **forwarding**: move packets from router's input to appropriate router output. 2) **routing**: determine route taken by packets from source to destination, routing algorithms

Data plane: 1) local, per-router function, 2) determines how datagram arriving on router input port is forwarded to router output port. 3) forwarding function

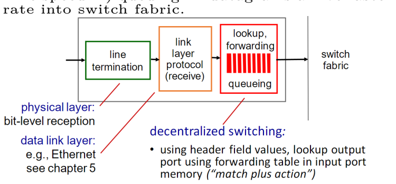
Control plane: 1) network-wide logic 2) determines how datagram is routed among routers along end-end path from source host to destination host 3) two control-plane approaches: (traditional routing algorithms: implemented in routers, software-defined networking (SDN): implemented in (remote) servers).



Network service model: **example services for individual datagrams:**, guaranteed delivery, guaranteed delivery with less than 40 msec delay. **example services for a flow of datagrams:**, in-order datagram delivery, guaranteed minimum bandwidth to flow, restrictions on changes in inter-packet spacing
Router architecture overview



Input port functions 1) goal: complete input port processing at 'line speed' 2) queuing: if datagrams arrive faster than forwarding rate into switch fabric.

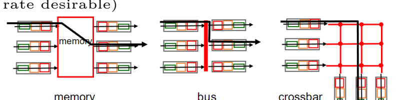


Longest prefix matching: when looking for forwarding table entry for given destination address, use longest address prefix that matches destination address.

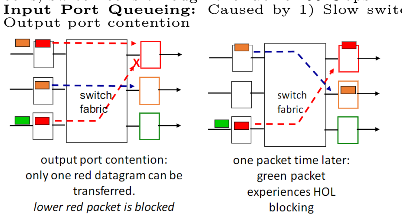
Destination Address Range	Link interface
11001000 00010111 00010*** *********	0
11001000 00010111 00011000 *********	1
11001000 00010111 00011*** *********	2
otherwise	3

examples:
DA: 11001000 00010111 00010110 10100001 which interface? 0
DA: 11001000 00010111 00011000 10101010 which interface? 1

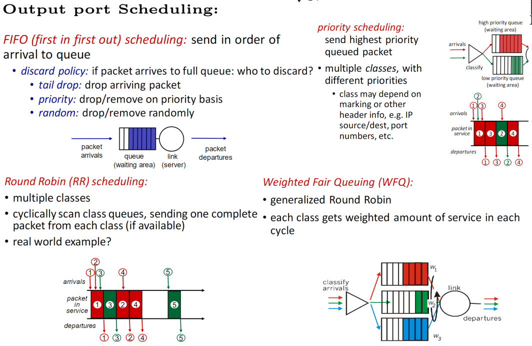
Switching fabrics:transfer packet from input buffer to appropriate output buffer, switching rate: rate at which packets can be transferred from inputs to outputs, (often measured as multiple of input/output line rate, N inputs: switching rate N times line rate desirable)



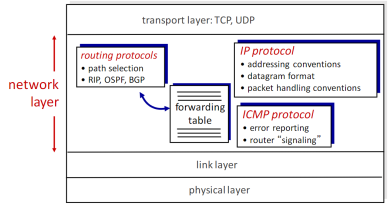
Switching via a bus, datagram from input port memory to output port memory via a shared bus, bus contention: switching speed limited by bus bandwidth. 30 Gbps.
Switching via interconnection network, overcome bus bandwidth limitations, banyan networks, crossbar, other interconnection nets initially developed to connect processors in multiprocessor, advanced design: fragmenting datagram into fixed length cells, switch cells through the fabric. 60 Gbps.
Input Port Queueing: Caused by 1) Slow switching fabric, 2) Output port contention



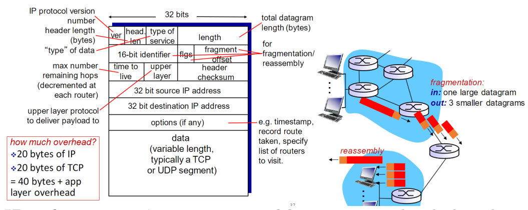
Reducing Input Queueing: Why? Reduce HOL blocking, Avoid packet drops at input queues, Save on queue memory, **How?** Increase switch fabric speed, Increase inbound capacity of output ports
Output ports buffer: required when datagrams arrive from fabric faster the transmission rate (How much buffering? RFC 3439 rule of thumb: average buffering equal to "typical" RTT (say 250 msec) times link capacity C, e.g., C = 10 Gpbs link: 2.5 Gbit buffer, recent recommendation [Appenzellet'04]: with N flows, buffering equal to: $\frac{RTT \times C}{\sqrt{N}}$)
Output port Scheduling:



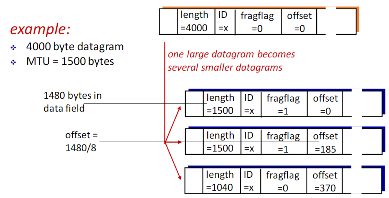
Internet network layer



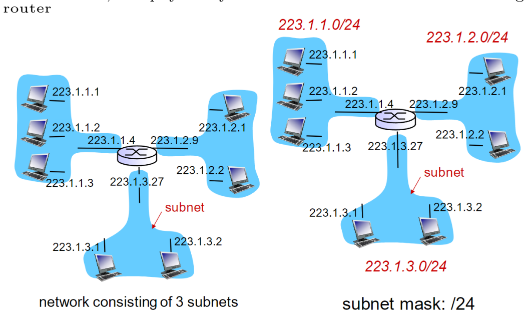
IP datagram format

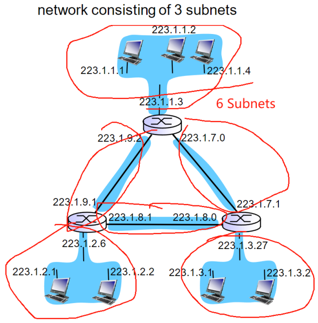


IP fragmentation, reassembly: network links have MTU(max.transfer size). So large IP datagram divided ("fragmented") within net



IP address: 32-bit identifier for host, router interface, interface: connection between host/router and physical link, Router's typically have multiple interfaces, host typically has one or two interfaces (e.g., wired Ethernet, wireless 802.11), IP addresses associated with each interface
Subnets:
IP address: subnet part - high order bits, host part - low order bits, What's a subnet ?, device interfaces with same subnet part of IP address, can physically reach each other without intervening router

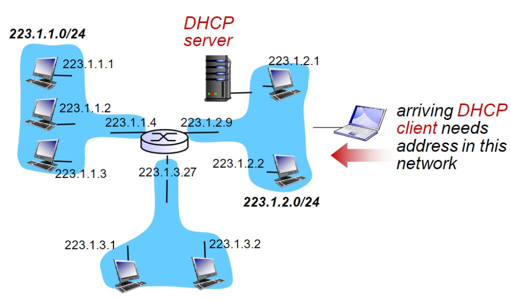




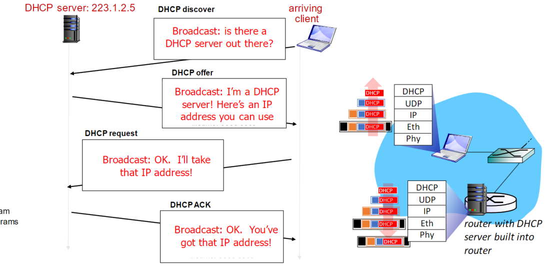
CIDR: Classless InterDomain Routing, subnet portion of address of arbitrary length, address format: a.b.c.d/x, where x is # bits in subnet portion of address

subnet part | host part
11001000 00010111 00010000 00000000
200.23.16.0/23
200.23.16.0–200.23.17.255

DHCP: Dynamic Host Configuration Protocol dynamically get ip address from a server
DHCP overview: 1) host broadcasts "DHCP discover" msg [optional], 2) DHCP server responds with "DHCP offer" msg [optional], 3) host requests IP address: "DHCP request" msg, 4) DHCP server sends address: "DHCP ack"

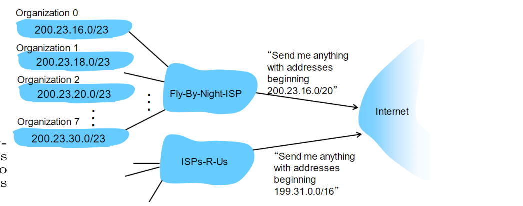


DHCP: example 1) connecting laptop needs its IP address, addr of first-hop router, addr of DNS server: use DHCP router with DHCP server built into router. 2) DHCP request encapsulated in UDP, encapsulated in IP, encapsulated in 802.1 Ethernet. 3) Ethernet frame broadcast (dest: FFFFFFFFFFFF) on LAN, received at router running DHCP server. 4) Ethernet demuxed to IP demuxed, UDP demuxed to DHCP. 5) DHCP server formulates DHCP ACK containing client's IP address, IP address of first-hop router for client, name & IP address of DNS server 6) encapsulation of DHCP server, frame forwarded to client, demuxing up to DHCP at client. 7) client now knows its IP address, name and IP address of DNS server, IP address of its first-hop router

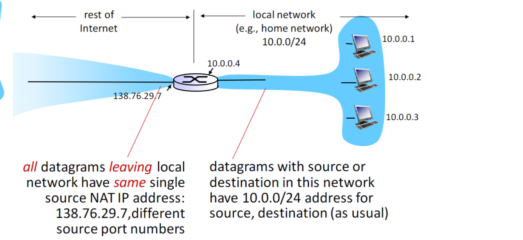


Hierarchical addressing: allows efficient advertisement of routing information. (network get subnet part of IP addr from allocated portion of its provider ISP's address space)

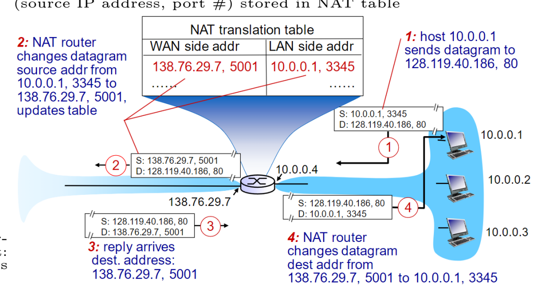
ISP's block	11001000 00010111 00010000 00000000	200.23.16.0/20
Organization 0	11001000 00010111 00010000 00000000	200.23.16.0/23
Organization 1	11001000 00010111 00010010 00000000	200.23.18.0/23
Organization 2	11001000 00010111 00010100 00000000	200.23.20.0/24
Organization 3	11001000 00010111 00010101 00000000	200.23.21.0/24
...	.....	....
Organization n	11001000 00010111 00011110 00000000	200.23.30.0/23



NAT: network address translation local network uses just one IP address as far as outside world is concerned. **Advantages:** 1) range of addresses not needed from ISP: just one IP address for all devices 2) can change addresses of devices in local network without notifying outside world 3) can change ISP without changing addresses of devices in local network 4) devices inside local net not explicitly addressable, visible by outside world (a security plus)



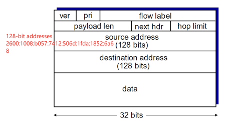
NAT router must: 1) outgoing datagrams: replace (source IP address, port #) of every outgoing datagram to (NAT IP address, new port #) 2) remember (in NAT translation table) every (source IP address, port #) to (NAT IP address, new port #) translation pair 3) incoming datagrams: replace (NAT IP address, new port #) in dest fields of every incoming datagram with corresponding (source IP address, port #) stored in NAT table



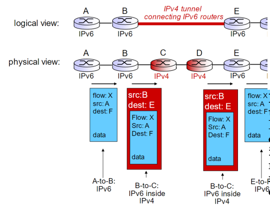
IPv6: initial motivation: 32-bit address space soon to be completely allocated. IPv6 datagram format: fixed-length 40 byte header, **no fragmentation allowed**

IPv6 datagram format

priority: identify priority among datagrams in flow
flow Label: identify datagrams in same "flow." (concept of "flow" not well defined).
next header: identify upper layer protocol for data



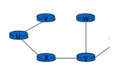
Tunneling



Routing Protocols Classification: global ("link state" algorithms) or decentralized ("distance vector" algorithms) information? static (routes change slowly over time) or dynamic (routes change more quickly, periodic update, in response to link cost changes)?

Step	N'	D(v),p(v)	D(w),p(w)	D(x),p(x)	D(y),p(y)	D(z),p(z)
0	u	2,u	5,u	1,u	∞	∞
1	ux	2,u	4,x		2,x	∞
2	uxy	2,u	3,y			4,y
3	uxyv		3,y			4,y
4	uxyvw					4,y
5	uxyvwz					

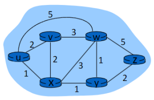
resulting shortest-path tree from u



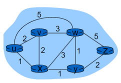
resulting forwarding table in u:

destination	link
v	(u,v)
x	(u,x)
y	(u,x)
w	(u,x)
z	(u,x)

Dijkstra



Distance vector algorithm: Dx(y) = estimate of least cost from x to y, x maintains distance vector Dx = [Dx(y): y ∈ N], node x:, knows cost to each neighbor v: c(x,v), maintains its neighbors' distance vectors. For each neighbor v, x maintains Dv = [Dv(y): y ∈ N]



clearly, $d_v(z) = 5$, $d_x(z) = 3$, $d_w(z) = 3$

B-F equation says:

$$d_u(z) = \min \{ c(u,v) + d_v(z), c(u,x) + d_x(z), c(u,w) + d_w(z) \} = \min \{2 + 5, 1 + 3, 5 + 3\} = 4$$

node achieving minimum is next hop in shortest path, used in forwarding table

key idea:

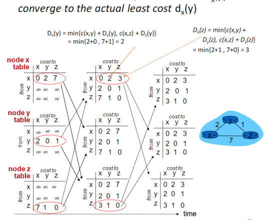
- from time-to-time, each node sends its own distance vector estimate to neighbors
- when x receives new DV estimate from neighbor, it updates its own DV using B-F equation:
  $D_x(y) \leftarrow \min_v\{c(x,v) + D_v(y)\}$ *for each node y ∈ N*
- under certain conditions, the estimate $D_x(y)$ *converge to the actual least cost* $d_x(y)$

iterative, asynchronous: each local iteration caused by:
- local link cost change
- DV update message from neighbor

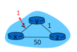
distributed:
- each node notifies neighbors *only* when its DV changes
  - neighbors then notify their neighbors if necessary

each node:

wait for (change in local link cost or msg from neighbor) → *recompute* estimates → if DV to any dest has changed, *notify* neighbors



link cost changes:
- node detects local link cost change
- updates routing info, recalculates distance vector
- if DV changes, notify neighbors



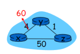
"good news travels fast"

t_0 : *y* detects link-cost change, updates its DV, informs its neighbors.

t_1 : *z* receives update from *y*, updates its table, computes new least cost to *x* , sends its neighbors its DV.

t_2 : *y* receives *z*'s update, updates its distance table. *y*'s least costs do *not* change, so *y* does *not* send a message to *z*.

link cost changes:
- node detects local link cost change
- *bad news travels slow* - "count to infinity" problem!
- 44 iterations before algorithm stabilizes: see text



poisoned reverse:
- If Z routes through Y to get to X :
  - Z tells Y its (Z's) distance to X is infinite (so Y won't route to X via Z)
- will this completely solve count to infinity problem?

Comparison of LS and DV algorithms

message complexity
- ***LS:*** with n nodes, E links, O(nE) msgs sent
- ***DV:*** exchange between neighbors only
  - convergence time varies

speed of convergence
- ***LS:*** O(n²) algorithm requires O(nE) msgs
  - may have oscillations
- ***DV:*** convergence time varies
  - may be routing loops
  - count-to-infinity problem

robustness: what happens if router malfunctions?

LS:
- node can advertise incorrect *link* cost
- each node computes only its *own* table

DV:
- DV node can advertise incorrect *path* cost
- each node's table used by others
  - error propagate thru network

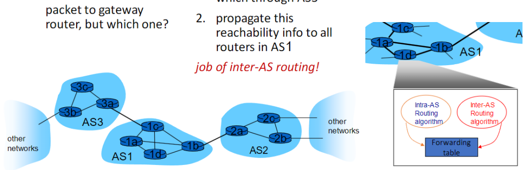
Scalable Routing: forwarding table configured by both intra- and inter-AS routing algorithm, intra-AS routing determine entries for destinations within AS, inter-AS & intra-AS determine entries for external destinations

Inter-AS tasks

- suppose router in AS1 receives datagram destined outside of AS1:
  - router should forward packet to gateway router, but which one?

AS1 must:
1. learn which dests are reachable through AS2, which through AS3
2. propagate this reachability info to all routers in AS1

job of inter-AS routing!



Intra-AS Routing in the internet: also known as interior gateway protocols (IGP), most common intra-AS routing protocols:, RIP: Routing Information Protocol, OSPF: Open Shortest Path First, IGRP: Interior Gateway Routing Protocol (Cisco proprietary for decades, until 2016)

OSPF (Open Shortest Path First)

"open": publicly available

uses link-state algorithm
- link state packet dissemination
- topology map at each node
- route computation using Dijkstra's algorithm

router floods OSPF link-state advertisements to all other routers in *entire* AS
- carried in OSPF messages directly over IP (rather than TCP or UDP
- link state: for each attached link

security: all OSPF messages authenticated (to prevent malicious intrusion)

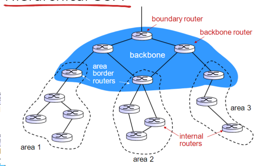
multiple same-cost paths allowed (only one path in RIP)

for each link, multiple cost metrics for different ToS (e.g., satellite link cost set low for best effort ToS; high for real-time ToS)

integrated uni- and multi-cast support:
- Multicast OSPF (MOSPF) uses same topology data base as OSPF

hierarchical OSPF in large domains.

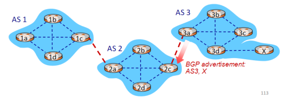
Hierarchical OSPF



- *two-level hierarchy:* local area, backbone.
  - link-state advertisements only in area
  - each nodes has detailed area topology; only know direction (shortest path) to nets in other areas.
- *area border routers:* "summarize" distances to nets in own area, advertise to other Area Border routers.
- *backbone routers:* run OSPF routing limited to backbone.
- *boundary routers:* connect to other AS'es.

Internet inter-AS routing: BGP (Border Gateway Protocol): the de facto inter-domain routing protocol, "glue that holds the Internet together", BGP provides each AS a means to:, **eBGP**: obtain subnet reachability information from neighboring ASes, **iBGP**: propagate reachability information to all AS- internal routers., determine "good" routes to other networks based on reachability information and policy, allows subnet to advertise its existence to rest of Internet

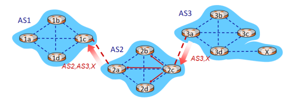
- BGP session: two BGP routers ("peers") exchange BGP messages over semi-permanent TCP connection:
  - advertising *paths* to different destination network prefixes (BGP is a "path vector" protocol)
- when AS3 gateway router 3a advertises path AS3,X to AS2 gateway router 2c:
  - AS3 *promises* to AS2 it will forward datagrams towards X



advertised prefix includes BGP attributes
- prefix + attributes = "route"

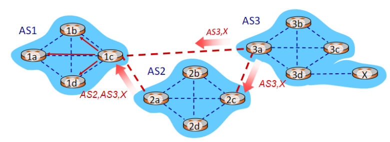
two important attributes:
- AS-PATH: list of ASes through which prefix advertisement has passed
- NEXT-HOP: indicates specific internal-AS router to next-hop AS

Policy-based routing:
- gateway receiving route advertisement uses *import policy* to accept/decline path (e.g., never route through AS Y).
- AS policy also determines whether to *advertise* path to other other neighboring ASes



- AS2 router 2c receives path advertisement AS3,X (via eBGP) from AS3 router 3a
- Based on AS2 policy, AS2 router 2c accepts path AS3,X, propagates (via iBGP) to all AS2 routers
- Based on AS2 policy, AS2 router 2a advertises (via eBGP) path AS2, AS3, X to AS1 router 1c

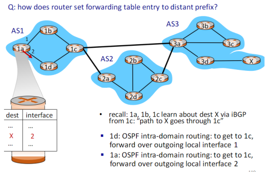
BGP messages:
- OPEN: opens TCP connection to remote BGP peer and authenticates sending BGP peer
- UPDATE: advertises new path (or withdraws old)
- KEEPALIVE: keeps connection alive in absence of UPDATES; also ACKs OPEN request
- NOTIFICATION: reports errors in previous msg; also used to close connection



gateway router may learn about multiple paths to destination:
- AS1 gateway router 1c learns path *AS2,AS3,X* from 2a
- AS1 gateway router 1c learns path *AS3,X* from 3a
- Based on policy, AS1 gateway router 1c chooses path *AS3,X, and advertises path within AS1 via iBGP*

BGP, OSPF, forwarding table entries

Q: how does router set forwarding table entry to distant prefix?

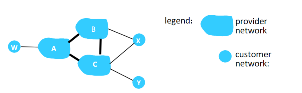


BGP route selection

- router may learn about more than one route to destination AS, selects route based on:
  1. local preference value attribute: policy decision
  2. shortest AS-PATH
  3. closest NEXT-HOP router: hot potato routing
  4. additional criteria

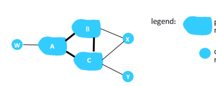
hot potato routing: choose local gateway that has least intra-domaincost (e.g., 2d chooses 2a, even though more AS hops to X): don't worry about inter-domain cost!

BGP: achieving policy via advertisements



Suppose an ISP only wants to route traffic to/from its customer networks (does not want to carry transit traffic between other ISPs)

- A advertises path A-w to B and to C
- B *chooses not to advertise* B-A-w to C:
  - B gets no "revenue" for routing C-B-A-w, since none of C, A, w are B's customers
  - C does not learn about C-B-A-w path
- C will route C-A-w (not using B) to get to w



Suppose an ISP only wants to route traffic to/from its custom (does not want to carry transit traffic between other

- A,B,C are *provider networks*
- X,W,Y are customer (of provider networks)
- X is *dual-homed:* attached to two networks
- *policy to enforce:* X does not want to route from I
  - .. so X will not advertise to B a route to C

Why different Intra-, Inter-AS routing ?

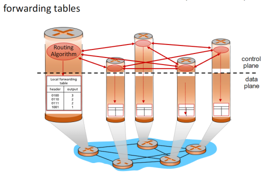
policy:
- inter-AS: admin wants control over how its traffic routed, who routes through its net.
- intra-AS: single admin, so no policy decisions needed

scale:
- hierarchical routing saves table size, reduced update traffic

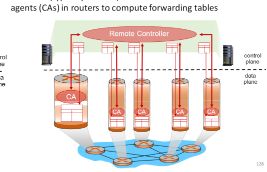
performance:
- intra-AS: can focus on performance
- inter-AS: policy may dominate over performance

Internet network layer: historically has been implemented via distributed, per-router approach, **monolithic** router contains switching hardware, runs proprietary implementation of Internet standard protocols (IP, RIP, IS-IS, OSPF, BGP) in proprietary router OS (e.g., Cisco IOS), different "middleboxes" for different network layer functions: firewalls, load balancers, NAT boxes, ..

Individual routing algorithm components *in each and every router* interact with each other in control plane to compute forwarding tables

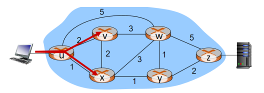


A distinct (typically remote) controller interacts with local control agents (CAs) in routers to compute forwarding tables

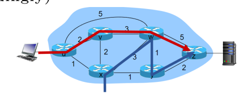


Software defined networking (SDN) a logically centralized control plane?, easier network management: avoid router misconfigurations, greater flexibility of traffic flows, table-based forwarding allows "programming" routers, centralized "programming" easier: compute tables centrally and distribute, distributed "programming: more difficult: compute tables as result of distributed algorithm (protocol) implemented in each and every router, open (non-proprietary) implementation of control plane

Traffic engineering: difficult traditional routing what if network operator wants u-to-z traffic to flow along uvwz, x-to-z traffic to flow xwyz? (need to define link weights so traffic routing algorithm computes routes accordingly)

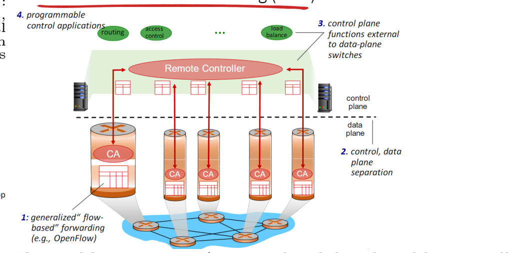


Q: what if network operator wants to split u-to-z traffic along uvwz *and* uxyz (load balancing)?
A: can't do it (or need a new routing algorithm)



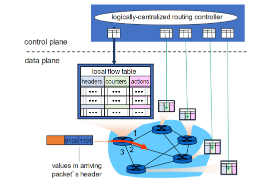
Q: what if w wants to route blue and red traffic differently?
A: can't do it (with destination based forwarding, and LS, DV routing)

Software defined networking (SDN)

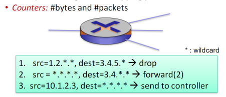


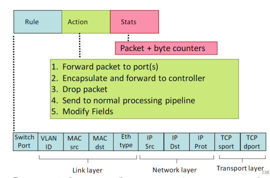
Flow table in a router (computed and distributed by controller) define router's match+action rules

Each router contains a *flow table* that is computed and distributed by a *logically centralized* routing controller



- *flow:* defined by header fields
- generalized forwarding: simple packet-handling rules
  - *Pattern:* match values in packet header fields
  - *Actions: for matched packet:* drop, forward, modify, matched packet or send matched packet to controller
  - *Priority:* disambiguate overlapping patterns
  - *Counters:* #bytes and #packets





Destination-based forwarding:

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
*	*	*	*	*	*	51.6.0.8	*	*	*	port6

IP datagrams destined to IP address 51.6.0.8 should be forwarded to router output port 6

Firewall:

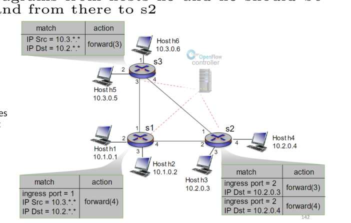
Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
*	*	*	*	*	*	*	*	*	22	drop

do not forward (block) all datagrams destined to TCP port 22

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
*	*	*	*	*	128.119.1.1	*	*	*	*	drop

do not forward (block) all datagrams sent by host 128.119.1.1

OpenFlow abstraction match+action: unifies different kinds of devices, Example: datagrams from hosts h5 and h6 should be sent to h3 or h4, via s1 and from there to s2



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

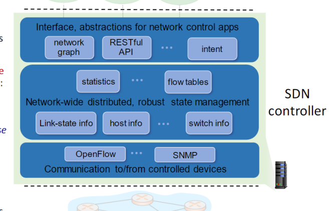
SDNData plane switches, 1) fast, simple, commodity switches implementing generalized data- plane forwarding in hardware, 2) switch flow table computed, installed by controller, 3) API for table-based switch control (e.g., OpenFlow), defines what is controllable and what is not, 4) protocol for communicating with controller (e.g., OpenFlow)

SDN controller (network OS): 1) maintain network state information 2) interacts with network control applications "above" via northbound API interacts with network switches "below" via southbound API 3) implemented as distributed system for performance, scalability, fault-tolerance, robustness

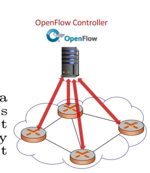
Interface layer to network control apps: abstractions API

Network-wide state management layer: state of networks links, switches, services: a *distributed database*

communication layer: communicate between SDN controller and controlled switches

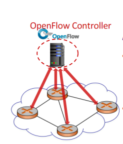


OpenFlow protocol

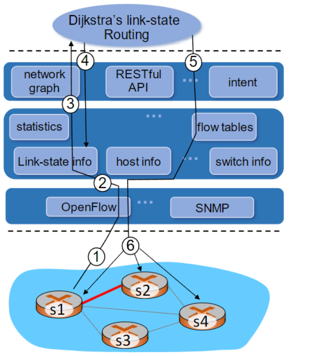


- operates between controller, switch
- TCP used to exchange messages
  - optional encryption
- three classes of OpenFlow messages:
  - controller-to-switch
  - asynchronous (switch to controller)
  - symmetric (misc)

Key controller-to-switch messages
- *features:* controller queries switch features, switch replies
- *configure:* controller queries/sets switch configuration parameters
- *modify-state:* add, delete, modify flow entries in the OpenFlow tables
- *packet-out:* controller can send this packet out of specific switch port



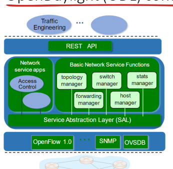
Key switch-to-controller messages
- *packet-in:* transfer packet (and its control) to controller. See packet-out message from controller
- *flow-removed:* flow table entry deleted at switch
- *port status:* inform controller of a change on a port.



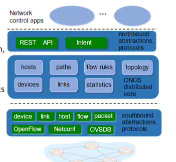
1. S1, experiencing link failure using OpenFlow port status message to notify controller
2. SDN controller receives OpenFlow message, updates link status info
3. Dijkstra's routing algorithm application has previously registered to be called when ever link status changes. It is called.
4. Dijkstra's routing algorithm access network graph info, link state info in controller, computes new routes
5. link state routing app interacts with flow-table-computation component in SDN controller, which computes new flow tables needed
6. Controller uses OpenFlow to install new tables in switches that need updating

SDN network-control apps: 1) "brains" of control: implement control functions using lower-level services, API provided by SDN controller 2) unbundled: can be provided by 3rd party: distinct from routing vendor, or SDN controller

OpenDaylight (ODL) controller



- ODL Lithium controller
- network apps may be contained within, or be external to SDN controller
- Service Abstraction Layer: interconnects internal, external applications and services



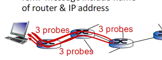
- control apps separate from controller
- intent framework: high-level specification of service: what rather than how
- considerable emphasis on distributed core: service reliability, replication performance scaling

SDN challenges, hardening the control plane: dependable, reliable, performance-scalable, secure distributed system, robustness to failures: leverage strong theory of reliable distributed system for control plane, dependability, security: "baked in" from day one?, networks, protocols meeting mission-specific requirements, e.g., real-time, ultra-reliable, ultra-secure, Internet-scaling
ICMP: internet control message protocol: used by hosts & routers to communicate network- level information, error reporting: unreachable host, network, port, protocol, echo request/reply (used by ping), network-layer "above" IP:, ICMP msgs carried in IP datagrams,
ICMP message: type, code plus first 8 bytes of IP datagram causing error

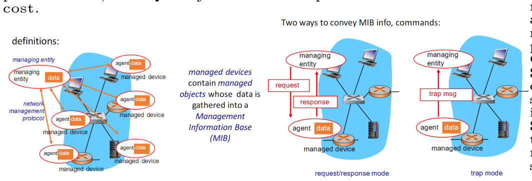
Type	Code	description
0	0	echo reply (ping)
3	0	dest. network unreachable
3	1	dest host unreachable
3	2	dest protocol unreachable
3	3	dest port unreachable
3	6	dest network unknown
3	7	dest host unknown
4	0	source quench (congestion control - not used)
8	0	echo request (ping)
9	0	route advertisement
10	0	router discovery
11	0	TTL expired
12	0	bad IP header

- source sends series of UDP segments to destination
  - first set has TTL =1
  - second set has TTL=2, etc.
  - unlikely port number
- when datagram in *n*th set arrives to nth router:
  - router discards datagram and sends source ICMP message (type 11, code 0)
  - ICMP message include name of router & IP address
- when ICMP message arrives, source records RTTs

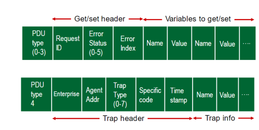
stopping criteria:
- UDP segment eventually arrives at destination host
- destination returns ICMP "port unreachable" message (type 3, code 3)
- source stops



Network management includes the deployment, integration and coordination of the hardware, software, and human elements to monitor, test, poll, configure, analyze, evaluate, and control the network and element resources to meet the real-time, operational performance, and Quality of Service requirements at a reasonable cost.



Message type	Function
GetRequest GetNextRequest GetBulkRequest	manager-to-agent: "get me data" (data instance, next data in list, block of data)
InformRequest	manager-to-manager: here's MIB value
SetRequest	manager-to-agent: set MIB value
Response	Agent-to-manager: value, response to Request
Trap	Agent-to-manager: inform manager of exceptional event

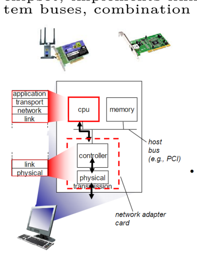


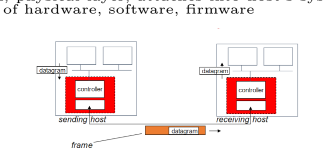
Link layer: introduction has responsibility of transferring datagram from one node to physically adjacent node over a link (terminology): hosts and routers: **nodes**, communication channels that connect adjacent nodes along communication path: **links**, wired links, wireless links, LANs, layer-2 packet: **frame**, encapsulates **datagram**)

Link layer services

- *framing, link access:*
  - encapsulate datagram into frame, adding header, trailer
  - channel access if shared medium
  - "MAC" addresses used in frame headers to identify source, destination
    - different from IP address!
- *reliable delivery between adjacent nodes*
  - we learned how to do this already (chapter 3)!
  - seldom used on low bit-error link (fiber, some twisted pair)
  - 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 presence of errors:
    - signals sender for retransmission or drops frame
- *error correction:*
  - receiver identifies *and corrects* bit error(s) without resorting to retransmission
- *half-duplex and full-duplex*
  - with half duplex, nodes at both ends of link can transmit, but not at same time

Where is the link layer implemented? 1) in each and every host, 2) link layer implemented in "**adaptor**" (aka network interface card NIC) or on a chip, Ethernet card, 802.11 card; Ethernet chipset, implements link, physical layer, attaches into host's system buses, combination of hardware, software, firmware



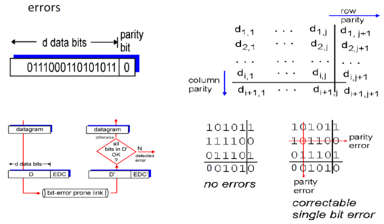


- **sending side:**
  - encapsulates datagram in frame
  - adds error checking bits, rdt, flow control, etc.
- **receiving side :**
  - looks for errors, rdt, flow control, etc.
  - extracts datagram, passes to upper layer at receiving side

Error detection EDC = Error Detection and Correction bits (redundancy) D = Data protected by error checking, may include header fields, Error detection not 100% reliable!, protocol may miss some errors, but rarely, larger EDC field yields better detection and correction
Parity checking

single bit parity:
- detect single bit errors

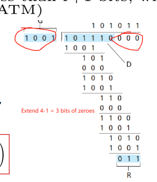
two-dimensional bit parity:
- detect and correct single bit errors



Cyclic redundancy check more powerful error-detection coding, view data bits, D, as a binary number, choose r+1 bit pattern (generator), G, goal: choose r CRC bits, R, such that, ¡D,R¿ exactly divisible by G (modulo 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, ATM)

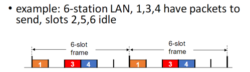
want:
$D \cdot 2^r$ XOR R = nG
equivalently:
$D \cdot 2^r$ = nG XOR R
equivalently:
if we divide $D \cdot 2^r$ by G, want remainder R to satisfy:

$$R = \text{remainder}\left(\frac{D \cdot 2^r}{G}\right)$$

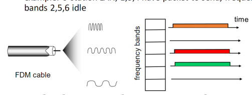


Multiple access links, protocols 1) point-to-point, PPP for dial-up access, point-to-point link between Ethernet switch, host, 2) broadcast (shared wire or medium), old-fashioned Ethernet, upstream HFC, 802.11 wireless LAN
Multiple access protocol Given: single shared broadcast channel, two or more simultaneous transmissions by nodes: interference, **collision** if node receives two or more signals at the same time. **Multiple access protocols distributed algorithm** that determines how nodes share channel, i.e., determine when node can transmit, communication about channel sharing must use channel itself!, no out-of-band channel for coordination
An ideal multiple access protocol given: broadcast channel of rate R bps desiderata: 1. when one node wants to transmit, it can send at rate R. 2. when M nodes want to transmit, each can send at average rate R/M 3. fully decentralized:, no special node to coordinate transmissions, no synchronization of clocks, slots 4. simple
MAC three broad classes: 1. channel partitioning, divide channel into smaller "pieces" (time slots, frequency, code), allocate piece to node for exclusive use, 2. random access, channel not divided, allow collisions, "recover" from collisions, "taking turns", 3. nodes take turns, but nodes with more to send can take longer turns

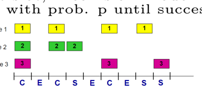
TDMA: time division multiple access
- access to channel in "rounds"
- each station gets fixed length slot (length = packet transmission time) in each round
- unused slots go idle
- example: 6-station LAN, 1,3,4 have packets to send, slots 2,5,6 idle



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



Random access protocol: when node has packet to send, transmit at full channel data rate R., no a priori coordination among nodes, two or more transmitting nodes -¿ "collision",, random access MAC protocol specifies:, how to detect collisions, how to recover from collisions (e.g., via delayed retransmissions), examples of random access MAC protocols:, slotted ALOHA, ALOHA, CSMA, CSMA/CD, CSMA/CA
Slotted ALOHA assumptions: 1) all frames same size, 2) time divided into equal size slots (time to transmit 1 frame), 3) nodes start to transmit only slot beginning, 4) nodes are synchronized, if 2 or more nodes transmit in slot, these nodes detect collision
Slotted ALOHA 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 prob. p until success



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:
- collisions, wasting slots
- idle slots
- nodes may be able to detect collision in less than time to transmit packet
- clock synchronization

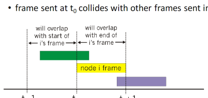
efficiency: long-run fraction of successful slots (many nodes, all with many frames to send)

- *suppose:* N nodes with many frames to send, each transmits in slot with probability p
- prob that given node has success in a slot = $p(1-p)^{N-1}$
- prob that *any* node has a success = $Np(1-p)^{N-1}$
- max efficiency: find p^* that maximizes $Np(1-p)^{N-1}$
- for many nodes, take limit of $Np^*(1-p^*)^{N-1}$ as N goes to infinity, gives:

max efficiency = 1/e = .37

at best: channel used for useful transmissions 37% of time! **!**

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



P(success by given node) = P(node transmits) ·
P(no other node transmits in $[t_0-1,t_0]$ ·
P(no other node transmits in $[t_0-1,t_0]$

$= p \cdot (1-p)^{N-1} \cdot (1-p)^{N-1}$

$= p \cdot (1-p)^{2(N-1)}$

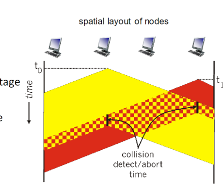
… choosing optimum p and then letting $n \rightarrow \infty$

$= 1/(2e) = .18$

even *worse* than slotted Aloha!

CSMA (carrier sense multiple access): listen before transmit: if channel sensed idle: transmit entire frame, if channel sensed busy, defer transmission. **collisions can still occur**: propagation delay means two nodes may not hear each other's transmission, **collision**: entire packet transmission time wasted, distance & propagation delay play role in determining collision probability spatial layout of nodes

CSMA/CD: carrier sensing, deferral as in CSMA
- collisions *detected* within short time
- colliding transmissions aborted, reducing channel wastage
- collision detection:
  - easy in wired LANs: measure signal strengths, compare transmitted, received signals
  - difficult in wireless LANs: received signal strength overwhelmed by local transmission strength
- human analogy: the polite conversationalist



Ethernet CSMA/CD algorithm

1. NIC receives datagram from network layer, creates frame
2. If NIC senses channel idle, starts frame transmission. If NIC senses channel busy, waits until channel idle, then transmits.
3. If NIC transmits entire frame without detecting another transmission, NIC is done with frame !
4. If NIC detects another transmission while transmitting, aborts and sends jam signal
5. After aborting, NIC enters *binary (exponential) backoff:*
   - after *m*th collision, NIC chooses *K* at random from $\{0,1,2, \ldots, 2^m-1\}$. NIC waits K·512 bit times, returns to Step 2
   - longer backoff interval with more collisions

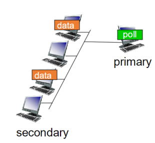
- t_{prop} = max prop delay between 2 nodes in LAN
- t_{trans} = time to transmit max-size frame

$$efficiency = \frac{1}{1 + 5\, t_{prop}/t_{trans}}$$

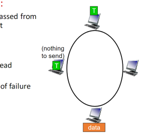
- efficiency goes to 1
  - as t_{prop} goes to 0
  - as t_{trans} goes to infinity
- better performance than ALOHA: and simple, cheap, decentralized!

"Taking turns" MAC protocols

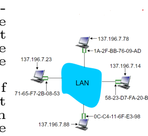
polling:
- primary node "invites" secondary nodes to transmit in turn
- concerns:
  - polling overhead
  - latency
  - single point of failure (primary)



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



Summary of MAC protocols 1) channel partitioning, by time, frequency or code, Time Division, Frequency Division, 2) 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, 3) taking turns, polling from central site, token passing, Bluetooth, token ring
MAC (or LAN or physical or Ethernet) address:, function: used "locally" to get frame from one interface to another physically-connected interface (same network, in IP-addressing sense), 48 bit MAC address (for most LANs) 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), each adapter on LAN has unique LAN address. MAC address allocation administered by IEEE, manufacturer buys portion of MAC address space (to assure uniqueness), analogy:, MAC address: like Social Security Number, IP address: like postal address, MAC flat address -¿ portability, can move LAN card from one LAN to another, IP hierarchical address not portable, address depends on IP subnet to which node is attached
ARP address resolution protocol: determine interface's MAC address, knowing its IP address



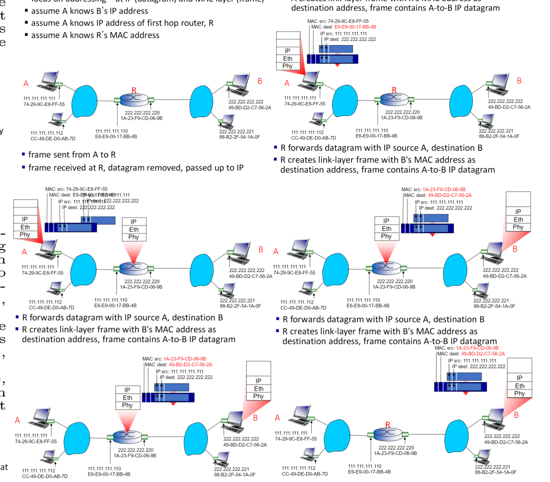
ARP table: each IP node (host, router) on LAN has table
- IP/MAC address mappings for some LAN nodes:
  < IP address; MAC address; TTL>
- TTL (Time To Live): time after which address mapping will be forgotten (typically 20 min)

1. A wants to send datagram to B
   - B's MAC address not in A's ARP table.
2. A broadcasts ARP query packet, containing B's IP address
   - destination MAC address = FF-FF-FF-FF-FF-FF
   - all nodes on LAN receive ARP query
3. B receives ARP packet, replies to A with its (B's) MAC address
   - frame sent to A's MAC address (unicast)
4. A caches (saves) IP-to-MAC address pair in its ARP table until information becomes old (times out)
   - soft state: information that times out (goes away) unless refreshed
5. ARP is "plug-and-play":
   - nodes create their ARP tables *without intervention from net administrator*

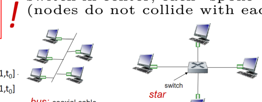
Addressing: routing to another LAN

walkthrough: send datagram from A to B via R
- focus on addressing – at IP (datagram) and MAC layer (frame)
- assume A knows B's IP address
- assume A knows IP address of first hop router, R
- assume A knows R's MAC address

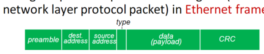
- A creates IP datagram with IP source A, destination B
- A creates link-layer frame with R's MAC address as destination address, frame contains A-to-B IP datagram



Ethernet "dominant" wired LAN technology:, single chip, multiple speeds (e.g., Broadcom BCM5761), first widely used LAN technology, simple, cheap, kept up with speed race: 10 Mbps – 10 Gbps. **bus**: popular through mid 90s, all nodes in same collision domain (can collide with each other), **star**: prevails today, active switch in center, each "spoke" runs a (separate) Ethernet protocol (nodes do not collide with each other)



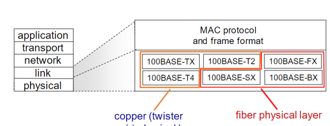
sending adapter encapsulates IP datagram (or other network layer protocol packet) in Ethernet frame



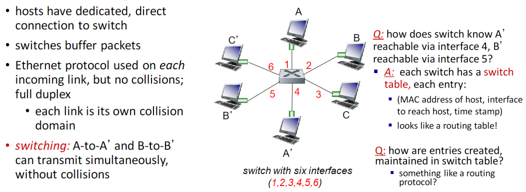
preamble:, 7 bytes with pattern 10101010 followed by one byte with pattern 10101011, used to synchronize receiver, sender clock rates
addresses: 6 byte source, destination MAC addresses, if adapter receives frame with matching destination address, or with broadcast address (e.g. ARP packet), it passes 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),
CRC: cyclic redundancy check at receiver, error detected: frame is dropped
Ethernet connectionless: no handshaking between sending and receiving NICs, **unreliable**: receiving NIC doesn't send acks or nacks 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**.

802.3 Ethernet standards: link & physical layers

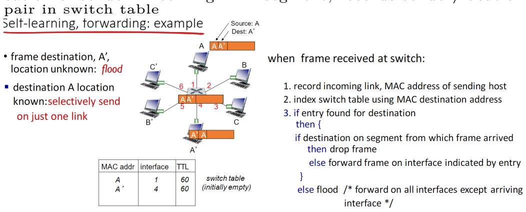
- *many* different Ethernet standards
  - common MAC protocol and frame format
  - different speeds: 2 Mbps, 10 Mbps, 100 Mbps, 1Gbps, 10 Gbps, 40 Gbps
  - different physical layer media: fiber, cable



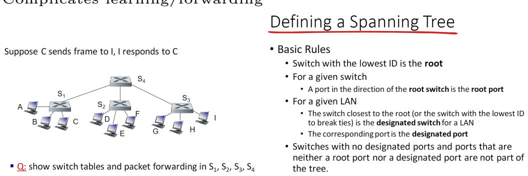
Ethernet switch, link-layer device: takes an active role, store, forward Ethernet frames, examine incoming frame's MAC address, selectively forward frame to one-or-more outgoing links when frame is to be forwarded on segment, uses CSMA/CD to access segment, **transparent**, hosts are unaware of presence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

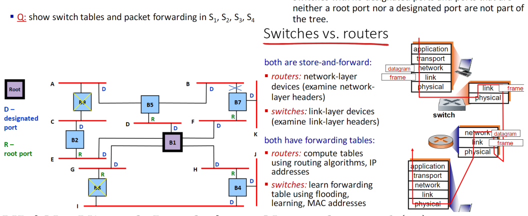


Switch: self-learning: switch learns which hosts can be reached through which interfaces, when frame received, switch "learns" location of sender: incoming LAN segment, records sender/location pair in switch table

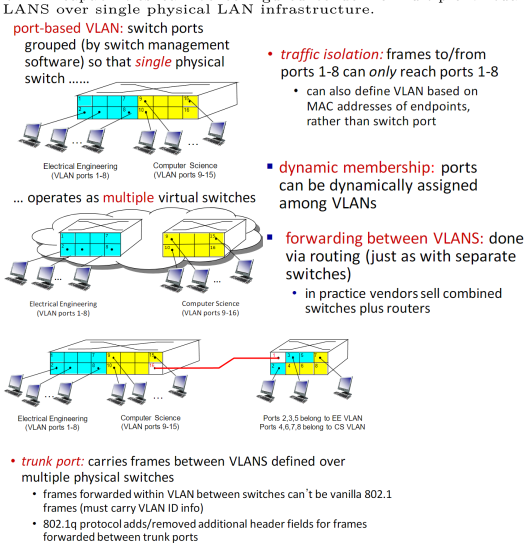


Problem: If there is a loop in the extended LAN, a packet could circulate forever, Side question: Why would we have loops?, Solution, Select which switches should actively forward, Create a spanning tree to eliminate unnecessary edges, Adds robustness, Complicates learning/forwarding

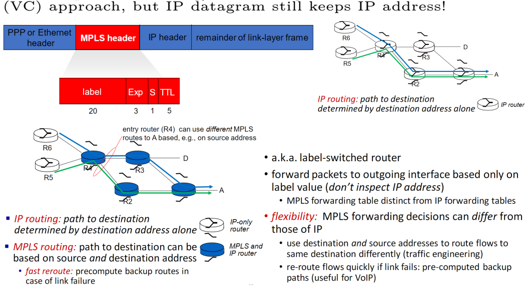




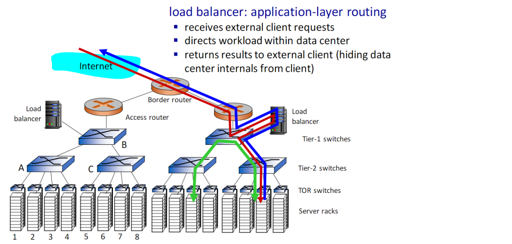
VLANs Virtual Local Area Network switch(es) supporting VLAN capabilities can be configured to define multiple virtual LANS over single physical LAN infrastructure.



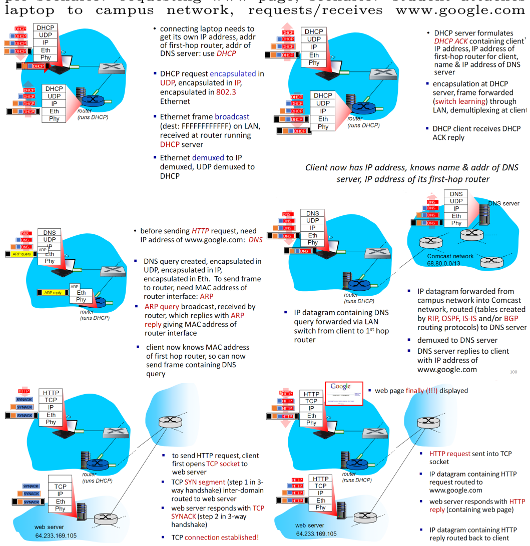
Multiprotocol label switching (MPLS): initial goal: high-speed IP forwarding using fixed length label (instead of IP address), fast lookup using fixed length identifier (rather than shortest prefix matching), borrowing ideas from Virtual Circuit (VC) approach, but IP datagram still keeps IP address!



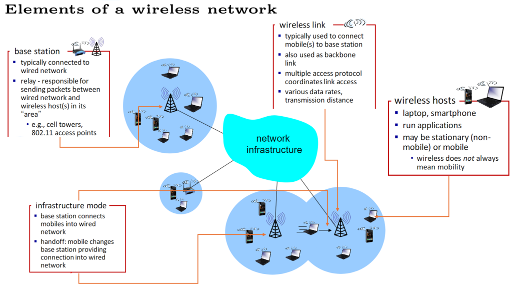
Data center networks: 10's to 100's of thousands of hosts, often closely coupled, in close proximity:, e-business (e.g. Amazon), content-servers (e.g., YouTube, Akamai, Apple, Microsoft), search engines, data mining (e.g., Google)
Challenges: 1) multiple applications, each serving massive numbers of clients 2) managing/balancing load, avoiding processing, networking, data bottlenecks



A day in the life of a web request, journey down protocol stack complete!, application, transport, network, link, putting-it-all-together: synthesis!, goal: identify, review, understand protocols (at all layers) involved in seemingly simple scenario: requesting www page, scenario: student attaches laptop to campus network, requests/receives www.google.com

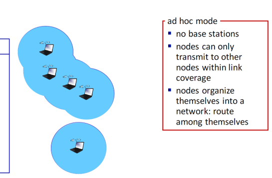


Elements of a wireless network

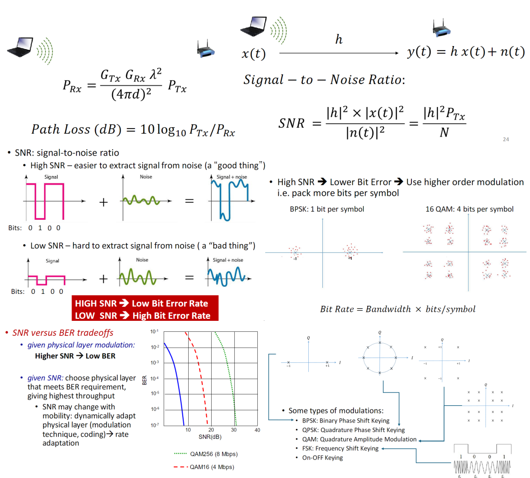


Wireless network taxonomy

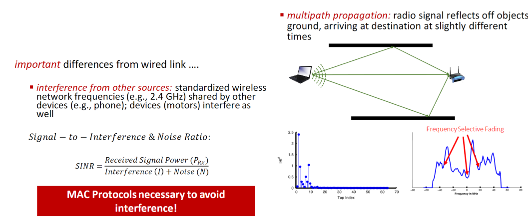
	single hop	multiple hops
infrastructure (e.g., APs)	host connects to base station (WiFi, WiMAX, cellular) which connects to larger Internet	host may have to relay through several wireless nodes to connect to larger Internet: *mesh net*
no infrastructure	no base station, no connection to larger Internet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



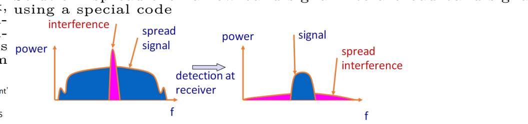
Wireless Link Characteristics: important differences from wired link **decreased signal strength:** radio signal attenuates as it propagates through matter (path loss) **interference from other sources:** standardized wireless network frequencies (e.g., 2.4 GHz) shared by other devices (e.g., phone); devices (motors) interfere as well **multipath propagation:** radio signal reflects off objects ground, arriving at destination at slightly different times

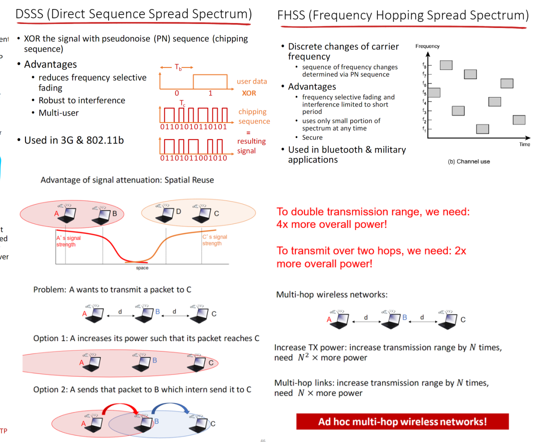


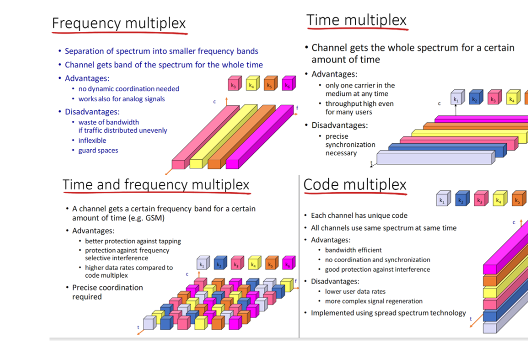
Shannon Capacity Theorem: $Capacity = Bandwidth \times lgo_2(1+SNR)$



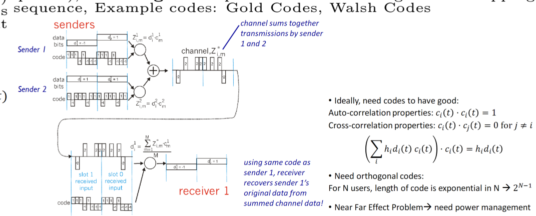
Problem of radio transmission: frequency dependent fading can wipe out narrow band signals for duration of the interference. Solution: spread the narrow band signal into a broad band signal using a special code



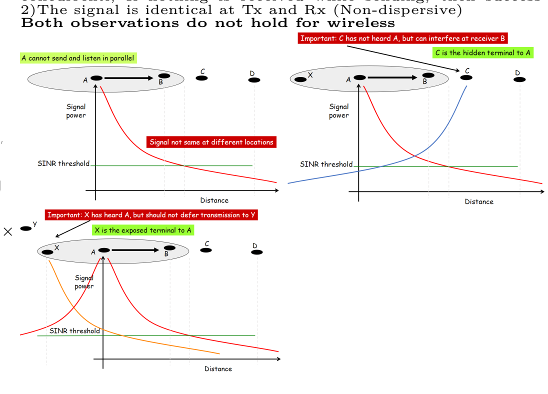




Code Division Multiple Access (CDMA) 1) unique "code" assigned to each user; i.e., code set partitioning, all users share same frequency, but each user has own "chipping" sequence (i.e., code) to encode data, allows multiple users to "coexist" and transmit simultaneously with minimal interference (if codes are "orthogonal"), **encoded signal** = (original data) X (chipping sequence), **decoding:** inner-product of encoded signal and chipping sequence, Example codes: Gold Codes, Walsh Codes



The Channel Access Problem: Multiple nodes share a channel, Pairwise communication desired, Simultaneous communication not possible, MAC Protocols, Suggests a scheme to schedule communication, Maximize number of communications, Ensure fairness among all transmitters
2 Observations on CSMA/CD 1) Transmitter can send/listen concurrently, If nothing is received while sending, then success 2)The signal is identical at Tx and Rx (Non-dispersive)
Both observations do not hold for wireless



IEEE 802.11 Wireless LAN

802.11b
- 2.4-5 GHz unlicensed spectrum
- up to 11 Mbps
- direct sequence spread spectrum (DSSS) in physical layer
  - all hosts use same chipping code

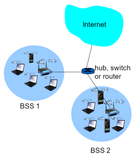
802.11ad/ay: Millimeter wave
- 2.4, 5, 60 GHz range
- Up to 7 Gbps

802.11a
- 5-6 GHz range
- up to 54 Mbps

802.11g
- 2.4-5 GHz range
- up to 54 Mbps

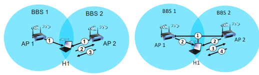
802.11n: multiple antenna
- 2.4-5 GHz range
- up to 200 Mbps

802.11ac: multiple antenna
- 2.4-5 GHz range
- Up to 1.69 Gbps



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

- 802.11b: 2.4GHz-2.485GHz spectrum divided into 11 channels at different frequencies
  - AP admin chooses frequency for AP
  - interference possible: channel can be same as that chosen by neighboring AP!
- host: must *associate* with an AP
  - scans channels, listening for *beacon frames* containing AP's name (SSID) and MAC address
  - selects AP to associate with
  - may perform authentication
  - will typically run DHCP to get IP address in AP's subnet



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

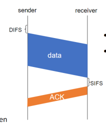
active scanning:
(1) Probe Request frame broadcast from H1
(2) Probe Response frames sent from APs
(3) Association Request frame sent: H1 to selected AP
(4) Association Response frame sent from selected AP to H1

IEEE 802.11: multiple access: 1) avoid collisions: 2+ nodes transmitting at same time, 802.11: CSMA - sense before transmitting, don't collide with ongoing transmission by other node. 2) no collision detection!, difficult to receive (sense collisions) when transmitting due to weak received signals (fading), can't sense all collisions in any case: hidden terminal, fading. So, goal: avoid collisions: CSMA/C(ollision)A(voidance)

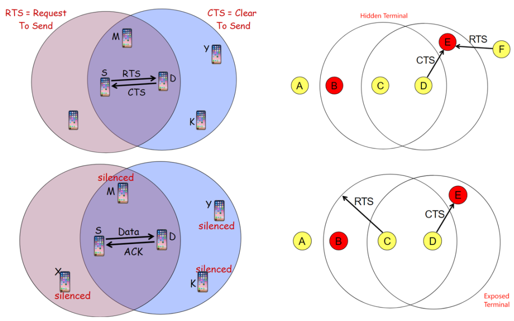
idea: allow sender to "reserve" channel rather than random access of data frames: avoid collisions of long data frames
- sender first transmits *small* request-to-send (RTS) packets to BS using CSMA
  - RTSs may still collide with each other (but they're short)
- BS broadcasts clear-to-send CTS in response to RTS
- CTS heard by all nodes
  - sender transmits data frame
  - other stations defer transmissions

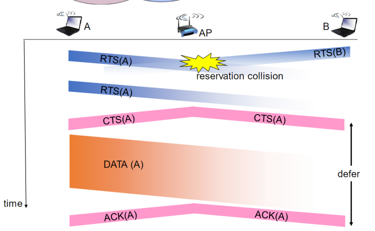
802.11 sender
1 if sense channel idle for **DIFS** then transmit entire frame (no CD)
2 if sense channel busy then
start random backoff time
timer counts down while channel idle
transmit when timer expires
if no ACK, increase random backoff interval, repeat 2

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

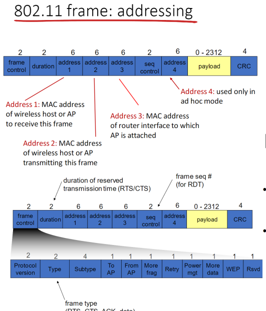


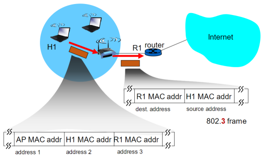
avoid data frame collisions completely using small reservation packets!



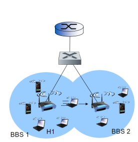


802.11 frame: addressing

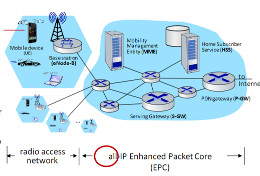




- H1 remains in same IP subnet: IP address can remain same
- switch: which AP is associated with H1?
  - self-learning (Ch. 5): switch will see frame from H1 and "remember" which switch port can be used to reach H1



4G/5G cellular networks, the solution for wide-area mobile Internet, widespread deployment/use: more mobile-broadband-connected devices than fixed- broadband-connected devices (5-1 in 2019)! 4G availability: 97% of time in Korea (90% in US), transmission rates up to 100's Mbps, technical standards: 3rd Generation Partnership Project (3GPP), 4G: Long-Term Evolution (LTE)standard



similarities to wired Internet
- edge/core distinction, but both belong to same carrier
- global cellular network: a network of networks
- widespread use of protocols we've studied: HTTP, DNS, TCP, UDP, IP, NAT, separation of data/control planes, SDN, Ethernet, tunneling
- interconnected to wired Internet

differences from wired Internet
- different wireless link layer
- mobility as a 1st class service
- user "identity" (via SIM card)
- business model: users subscribe to a cellular provider
  - strong notion of "home network" versus roaming on visited nets
  - global access, with authentication infrastructure, and inter-carrier settlements

Mobile device:, smartphone, tablet, laptop, IoT, ... with 4G LTE radio, 64-bit International Mobile Subscriber Identity (IMSI), stored on SIM (Subscriber Identity Module) card, LTE jargon: User Equipment (UE)

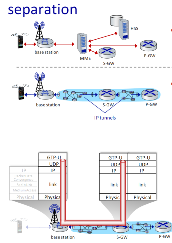
Base station:, at "edge" of carrier's network, manages wireless radio resources, mobile devices in its coverage area ("cell"), coordinates device authentication with other elements, similar to WiFi AP but: 1) active role in user mobility 2) coordinates with nearby base stations to optimize radio use, LTE jargon: eNode-B

Home Subscriber Service, stores info about mobile devices for which the HSS's network is their "home network", works with MME in device authentication

Serving Gateway (S-GW), PDN Gateway (P-GW), lie on data path from mobile to/from Internet, P-GW: gateway to mobile cellular network, Looks like nay other internet gateway router, provides NAT services, other routers: extensive use of tunneling

Mobility Management Entity device authentication (device-to-network, network-to-device) coordinated with mobile home network HSS, mobile device management: device handover between cells, tracking/paging device location, path (tunneling) setup from mobile device to P-GW

LTE: data plane control plane separation

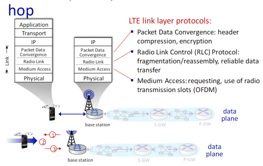


control plane
- new protocols for mobility management, security, authentication (later)

data plane
- new protocols at link, physical layers
- extensive use of tunneling to facilitate mobility

tunneling:
- mobile datagram encapsulated using GPRS Tunneling Protocol (GTP), sent inside UDP datagram to S-GW
- S-GW re-tunnels datagrams to P-GW
- supporting mobility: only tunneling endpoints change when mobile user moves

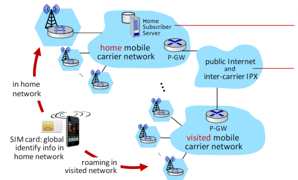
LTE data plane protocol stack: first hop



LTE link layer protocols:
- Packet Data Convergence: header compression, encryption
- Radio Link Control (RLC) Protocol: fragmentation/reassembly, reliable data transfer
- Medium Access: requesting, use of radio transmission slots (OFDM)

1. BS broadcasts primary synch signal every 5 ms on all frequencies
   - BSs from multiple carriers may be broadcasting synch signals
2. mobile finds a primary synch signal, then locates 2nd synch signal on this freq.
   - mobile then finds info broadcast by BS: channel bandwidth, configurations; BS's cellular carrier info
   - mobile may get info from multiple base stations, multiple cellular networks
3. mobile selects which BS to associate with (e.g., preference for home carrier)
4. more steps still needed to authenticate, establish state, set up data plane

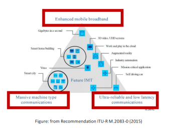
LTE mobiles: sleep modes as in WiFi, Bluetooth: LTE mobile may put radio to "sleep" to conserve battery:, light sleep: after 100's msec of inactivity, wake up periodically (100's msec) to check for downstream transmissions, deep sleep: after 5-10 secs of inactivity, mobile may change cells while deep sleeping – need to re-establish association



home network HSS:
- identify & services info, while in home network and roaming

all IP:
- carriers interconnect with each other, and public internet at exchange points
- legacy 2G, 3G: not all IP, handled otherwise

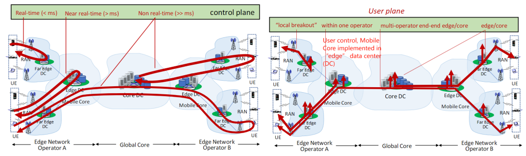
5G



"initial standards and launches have mostly focused on enhanced Mobile Broadband, 5G is expected to increasingly enable new business models and countless new use cases, in particular those of massive Machine Type Communications and Ultra-reliable and Low Latency Communications."

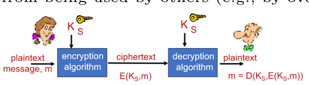
4G mobile core: boxes and protocols

5G: microservice-like architecture



Security: confidentiality: only sender, intended receiver should "understand" message contents, sender encrypts message, receiver decrypts message authentication: sender, receiver want to confirm identity of each other message integrity: sender, receiver want to ensure message not altered (in transit, or afterwards) without detection access and availability: services must be accessible and available to users

An adversary do 1) eavesdrop: intercept messages 2) actively insert messages into connection 3) impersonation: can fake (spoof) source address in packet (or any field in packet) 3) hijacking: "take over" ongoing connection by removing sender or receiver, inserting himself in place 4) denial of service: prevent service from being used by others (e.g., by overloading resources).

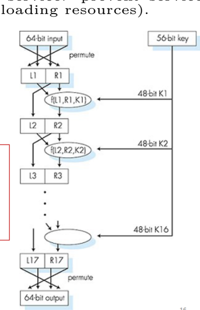


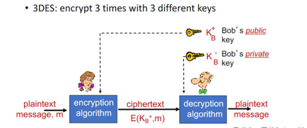
symmetric key crypto: Bob and Alice share same (symmetric) key: K_S

m plaintext message
$K_A(m)$ ciphertext, encrypted with key K_A
$m = D(K_A, E(K_A, m))$

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
- making DES more secure:
  - 3DES: encrypt 3 times with 3 different keys

DES operation
initial permutation
16 identical "rounds" of function application, each using different 48 bits of key
final permutation





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

RSA: Creating public/private key pair
1. choose two large prime numbers p, q. (e.g., 1024 bits each)
2. compute $n = pq$, $z = (p-1)(q-1)$
3. choose e (with $e<n$) that has no common factors with z (e, z are "relatively prime").
4. choose d such that $ed-1$ is exactly divisible by z. (in other words: $ed \bmod z = 1$).
5. public key is (n,e). private key is (n,d).

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

- fact: for any x and y: $x^y \bmod n = x^{(y \bmod z)} \bmod n$
  - where $n= pq$ and $z = (p-1)(q-1)$
- thus,
$c^d \bmod n = (m^e \bmod n)^d \bmod n$
$= m^{ed} \bmod n$
$= m^{(ed \bmod z)} \bmod n$
$= m^1 \bmod n$
$= m$

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

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

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.

	bit pattern	m	m^e	$c = m^e \bmod n$
encrypt:	0000l000	12	24832	17

	c	c^d	$m = c^d \bmod n$
decrypt:	17	481968572106750915091411825223071697	12

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

Protocol ap1.0: Alice says "I am Alice"

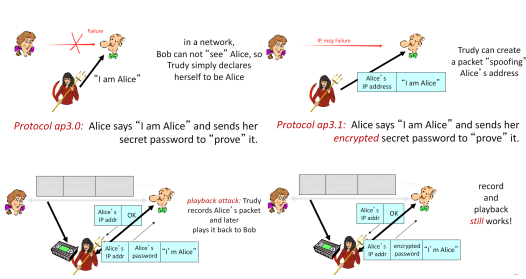
Protocol ap2.0: Alice says "I am Alice" in an IP packet containing her source IP address

in a network, Bob can not "see" Alice, so Trudy simply declares herself to be Alice

Trudy can create a packet "spoofing" Alice's address

Protocol ap3.0: Alice says "I am Alice" and sends her secret password to "prove" it.

Protocol ap3.1: Alice says "I am Alice" and sends her *encrypted* secret password to "prove" it.



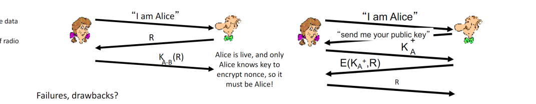
playback attack: Trudy records Alice's packet and later plays it back to Bob

record and playback *still* works!

Goal: avoid playback attack
nonce: number (R) used only *once-in-a-lifetime*
ap4.0: to prove Alice "live", Bob sends Alice *nonce*, R. Alice must return R, encrypted with shared secret key

ap4.0 requires shared symmetric key
- can we authenticate using public key techniques?

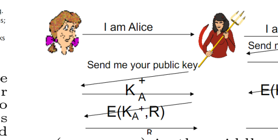
ap5.0: use nonce, public key cryptography



Alice is live, and only Alice knows key to encrypt nonce, so it must be Alice!

Failures, drawbacks?

man (or woman) in the middle attack: Trudy poses as Alice (to Bob) and as Bob (to Alice)



man (or woman) in the middle attack: Trudy poses as Alice (to Bob) and as Bob (to Alice) **difficult to detect:** Bob receives everything that Alice sends, and vice versa. (e.g., so Bob, Alice can meet one week later and recall conversation!). problem is that Trudy receives all messages as well!

Digital signatures

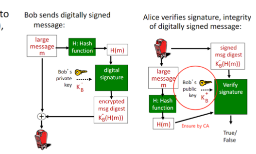
cryptographic technique analogous to hand-written signatures:
- sender (Bob) digitally signs document, establishing he is document owner/creator.
- *verifiable, nonforgeable:* recipient (Alice) can prove to someone that Bob, and no one else (including Alice), must have signed document

simple digital signature for message m:
- Bob signs m by signing with his private key K_B, creating signed message, $K_B(m)$

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)

computationally expensive to public-key-encrypt long messages

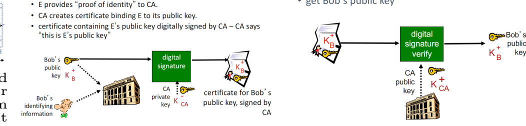
goal: fixed-length, easy-to-compute digital "fingerprint"
- apply hash function H to m, get fixed size message digest, H(m).



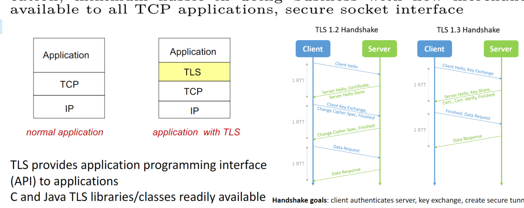
Certification authority (CA): binds public key to particular entity, E.

- E (person, router) registers its public key with CA.
  - E provides "proof of identity" to CA.
  - CA creates certificate binding E to its 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 verify Bob's certificate
  - get Bob's public key



SSL/TLS: Transport Layer Security, Originally called Secure Sockets Layer (SSL) until 1996, widely deployed security protocol, supported by almost all browsers, web servers, https, billions $/year over TLS, Current version: TLS1.3, TLS 1.3 is backwards compatible with TLS 1.2 but restricts usage on certain ciphers, provides, confidentiality, integrity, authentication, original goals:, Web e-commerce transactions, encryption (especially credit-card numbers), Web-server authentication, optional client authentication, minimum hassle in doing business with new merchant, available to all TCP applications, secure socket interface

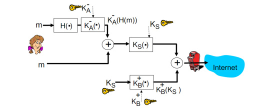


TLS provides application programming interface (API) to applications

C and Java TLS libraries/classes readily available

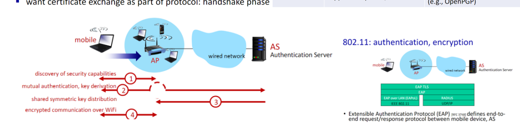
Handshake goals: client authenticates server, key exchange, create secure tunnel

TLS 1.2 vs 1.3: Faster handshake process: TLS 1.3 reduces the number of round trips required, 0-RTT resumption: TLS 1.3 allows session resumption without requiring a full handshake, Improved cipher suite efficiency, 1.3 uses more efficient cipher suites, 1.3 removes support for legacy cryptographic algorithms -¿ less computational overhead. TLS 1.3 enforces perfect forward secrecy, Removal of insecure cryptographic algorithms, i.e., SHA-1, RC4 cipher, CBC ciphers, No renegotiation in cipher during connection, Removes risk of DoS, MITM, and Downgrade attacks, Encrypted handshake, Prevents tampering handshake parameters



	TLS	PGP
End-to-End Encryption	Encrypts data in transit	- Keeps data encrypted at rest - Better suited for untrusted servers
Authentication	Authenticates servers using certs from Certificate Authorities (CAs)	Decentralized, but prone to key management issues
Usability	Built-in to apps and protocols	Requires more manual setup
Compatibility	Widely adopted standard, supported by OSes, browsers	Requires both endpoints to use the same application (e.g., OpenPGP)

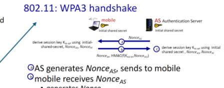
- but want to send byte streams & interactive data
- want set of secret keys for entire connection
- want certificate exchange as part of protocol: handshake phase



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
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
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
4. encrypted communication between mobile and remote host via AP
   - same key derived at mobile, AS
   - AS informs AP of the shared symmetric session

802.11: WPA3 handshake



- AS generates $Nonce_{AS}$, sends to mobile
- mobile receives $Nonce_{AS}$
  - generates $Nonce_M$
  - generates symmetric shared session key K_{M-AP} using $Nonce_{AS}$, $Nonce_M$, and initial shared secret
  - sends $Nonce_M$, and HMAC-signed value using $Nonce_{AS}$ and initial shared secret
- AS derives symmetric shared session key K_{M-AP}