Chapter 1

September 22, 2020 4:20 PM

Nuts and bolts view:

- Host (end-system): devices which the apps are running on. Edge of the Internet
- Packet switches: forward packets (e.g. routers, switches)
- Communication links: fiber, copper, radio, satellite
  - Transmission rate is called bandwidth
- Networks are collections of above
  - The Internet is network of networks
    - ISP (Internet Service Providers): provide access to the Internet to the hosts
- Internet standards:
  - IETF (Internet Engineering Task Force): proposes RFCs (request for comments) as standards

Service view:

- Infrastructure:
  - provide services to applications
    - E.g. web, streaming video
  - Provide programming interface to distributed applications
- Network edges: host (end system)
  - Access network and physical media

Access:

- DSL (digital subscriber line)
  - Use existing telephone line to connect
  - Downstream: 24-52 Mbps
  - Upstream 3.5-16 Mbps
  - ADSL (asymmetric DSL)
- Cable
  - Use FDM to divide frequency bands for different purposes
  - HFC (hybrid fiber coax)
    - Combining optical and coaxial cable
    - Asymmetric transmission rate
      - Downstream: 40 Mbps-1.2Gbps
      - Upstream: 30-100 Mbps
- Ethernet
  - Wired access at 100 Mbps, 1 Gbps, 10 Gbps
- Wireless (WiFi)
  - 802.11b/g/n/ac: 11,54,450,>1000 Mbps
- 3G/4G/5G
  - 10's Mbps
- Home share access network:
  - Network of cable, fiber attaches home to ISP router, has security issue

Links: physical media:

- Bit: propagates between transmitter/receiver pairs
- Physical link: lies between transmitter/receiver
- Guided media: copper, fiber, coax
- Unguided media: radio
- Twisted pair (TP): reduce interference

The Internet is a packet switched network

- Because original Internet applications need to send bursty data, and the packet switched network can handle it better with statistical multiplexing
- However, it may have excessive congestion
- Hosts: break application layer messages into packets
- Router: forward packets across links on path from source to destination
- Store-and-forward: entire packet must arrive at the router before it can be transmitted on next link
  - Avoid corruption
- Packet queueing and loss:
  - Happens when arrival rate to link exceeds transmission rate of link $d_{trans} = \frac{length\ of\ packet}{transmission\ rate}$
  - Packets will queue, waiting to be transmitted
  - If memory in router fills up, packets can be lost

Key network-core functions

- Forwarding: move input to appropriate router output link
  - Local action
- Routing: determine source-destination path
  - Global action

Circuit switching:

- End-to-end resources allocated and reserved for call between source and destination
- Dedicated, idle if not in use

Multiplexing methods

- FDM (frequency division multiplexing)
  - Divide frequency into specific chunks
  - Each person always get the same frequency domain and same bandwidth
  - May be wasted if not in use
  - Hard to divide frequency
- TDM (Time division multiplexing)
  - Divide time into slots
  - In each time slot, a user can get full bandwidth
  - Better use of resources

Performance and delay

- For any nodal, there are four sources of delay
  - d_{proc}: hardware dependent and usually small
  - d_{queue}: statistical, depends on congestion level of router
    - Traffic intensity= $\frac{aL}{R}$
      - □ a is the average packet arrival rate
      - □ L is packet length
      - □ R is link bandwidth
    - When traffic intensity ~ 0: queuing delay is small
    - When traffic intensity →1: queuing delay increases
    - When traffic intensity >1: queuing delay→ ∞
  - $d_{trans} = \frac{L}{R} = \frac{length\ of\ packet}{transmission\ rate}$
  - $d_{prop} = \frac{d}{s} = \frac{length\ of\ physical\ link}{propogation\ speed}$
  - Total delay $d_{nodal} = d_{proc} + d_{queue} + d_{trans} + d_{prop}$
- Throughput: rate($\frac{bit}{time}$) at which bits are sent from sender to receiver
  - Can be either instantaneous or average
  - If the link is sender-R_s-router-R_c-receiver

        - If $R_s > R_c$, average = R_c
        - If $R_s < \mathrm{R}_c$, average = R_s
        - This is called the bottleneck link

Malware:

- Virus: receiving/executing triggered
- Worm: passively triggered by receiving
- Spyware malware: record keystrokes, websites visited
- Infected host can become a botnet used for spam/DDoS attack

Common attacks

- Denial of Service:
    - Make resources unavailable to legitimate traffic by overwhelming resource with bogus traffic
    - Types:
        - Vulnerability attack: send messages to a vulnerable application or OS
        - Bandwidth flooding: sends a deluge of packets to the targeted host
        - Connection flooding: establish a large number of connection to the host
- Packet interception(sniffing):
    - Reads/records all packets passing by
- Fake identity (IP spoofing): send packet with false source address

Layers

- Each layer implements a service via its own internal-layer actions
- Rely on services provided by layer below
- As message is passed to the lower layers, additional info (header) is added
- Good for dealing with complex systems
- May have trouble when going beyond the defined layers

Protocols define format and order of messages exchanged, actions to be taken

- Different layers of the internet have different protocols (Internet Protocol Stack)
    - Application: SMTP, HTTP
        - This include the functionality of presentation and session layers
        - Protocols that are part of a distributed network application
    - Transport: TCP, UDP
        - Transfer of data between one process and another process (on different hosts)
    - Network: IP, routing protocols
        - Delivery of datagrams from a source to a destination
    - Link: Ethernet
        - Transfer of data between neighboring network devices
    - Physical: bits on wire
        - Transfer of bit into and out of a transmission media

5. Protocol layers:

(a) What are the five protocol layers, from top to bottom, in the Internet?

Ans: application, transport, network, link, physical

(b) For each of the five layers, what is the name of the packets processed at the layer?

Ans: Application layer: message; Transport layer: segment; Network layer: datagrams; Link layer: frames; Physical layer deals with individual bits within frame.

(c) An end-system processes up to which layer?

Ans: it processes all five layers (up through the application layer)

(d) A router processes up to which layer?

Ans: it processes network, link and physical layers (up through the network layer)

(e) A link-layer switch processes up to which layer?

Ans: it processes link and physical layers (up through the link layer)

Network security

- Digital signatures: used to detect tampering/changing of message contents
- Encryption: provides confidentiality by encoding contents
- Firewall: specialized middleboxes filtering of blocking traffic, inspecting packet contents inspections
- Access control: limiting use of resources or capabilities to given users
- Authentication: proving you are who you say you are

Internet structure: network of networks

- Use internet exchange point or peering link to connect different (regional/global) ISPs

Chapter 2

September 22, 2020 10:33 AM

Transmission delay: (M+K-1)L/R

Create a network app
Write programs that run on different end systems, communicate over network
No need to write software for network-core devices

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

Client-server paradigm

- Eg. HTTP, IMAP, FTP
- Server
  - Always-on host
  - Permanent IP address
  - Often in data centers
- Clients
  - Contact with server
  - May be intermittently connected
  - May have dynamic IP address
  - Do not communicate directly with each other

Peer-peer (P2P) architecture

- E.g. P2P file sharing
- Characteristic
  - No always-on server
  - Arbitrary end systems directly communicate
  - Peers send request and provide services one to another
    - Self scalability
  - Peers are intermittently connected and change IP addresses
    - Complex management

Processes communicating

- Process: program running within a host
- Inter-process communication: within the same host
- Exchanging messages: in different hosts
- Applications with P2P architectures have both client and server processes
  - Client process: initiates communication
  - Server process: waits to be contacted

Sockets

- Used to identify the process and send/receive messages

Addressing processes

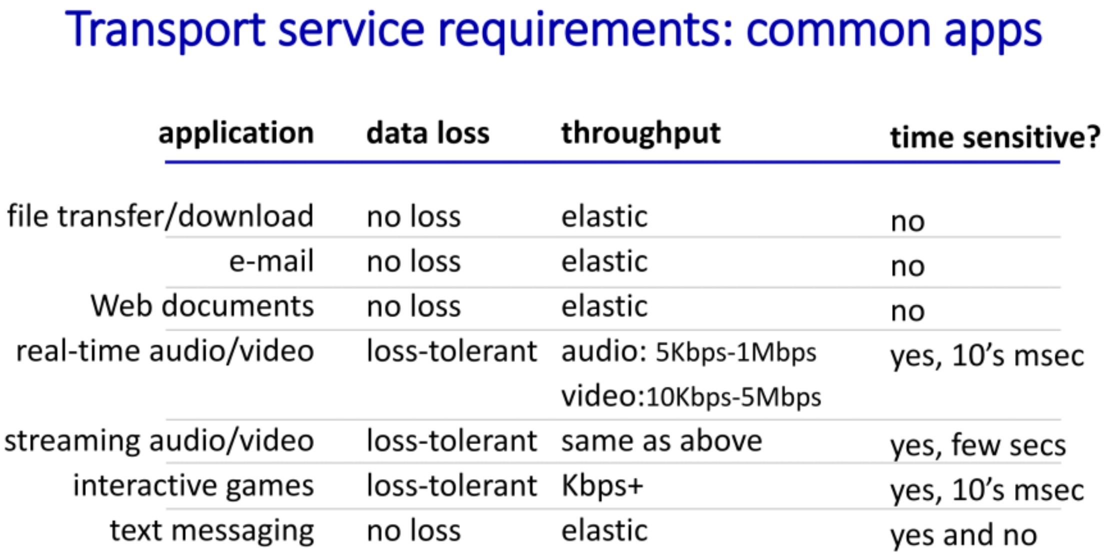
- To receive messages, process must have identifier
- Identifier: IP address + port number

An application-layer protocol defines:

- Types of messages exchanged
- Message syntax
- Message semantics
- Rules for when and how processes send & respond to messages

An app need transport service:

- Data integrity
  - ○ 100% reliable data transfer
- Timing
  - ○ Require low delay
- Throughput
  - ○ Require minimum amount of throughput
  - ○ Elastic: make use of whatever throughput they get
- Security
  - ○ encryption

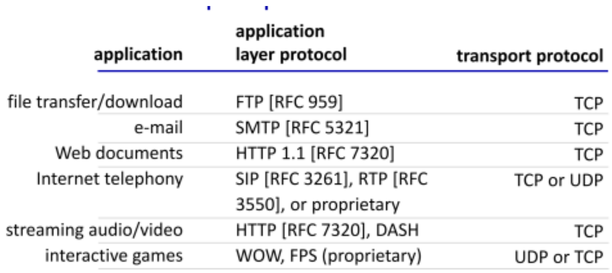


Transport service requirements: common apps

application	data loss	throughput	time sensitive?
file transfer/download	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, 10's msec
streaming audio/video	loss-tolerant	same as above	yes, few secs
interactive games	loss-tolerant	Kbps+	yes, 10's msec
text messaging	no loss	elastic	yes and no

Internet transport protocol services

- TCP
  - ○ Reliable transport
  - ○ Flow control
  - ○ Congestion control
  - ○ No timing, minimum throughput guarantee, security
  - ○ Connection-oriented
- UDP
  - ○ Unreliable data transfer
  - ○ No control, no other things
  - ○ But faster and higher capacity



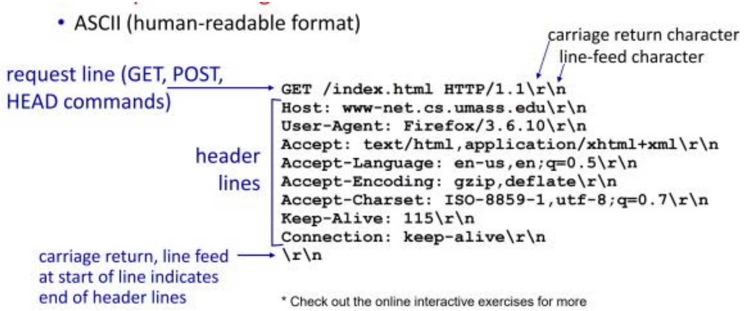
application	application layer protocol	transport protocol
file transfer/download	FTP [RFC 959]	TCP
e-mail	SMTP [RFC 5321]	TCP
Web documents	HTTP 1.1 [RFC 7320]	TCP
Internet telephony	SIP [RFC 3261], RTP [RFC 3550], or proprietary	TCP or UDP
streaming audio/video	HTTP [RFC 7320], DASH	TCP
interactive games	WOW, FPS (proprietary)	UDP or TCP

Securing TCP

- Vanilla TCP & UDP sockets
  - ○ No encryption
  - ○ Cleartext passwords sent into socket traverse Internet in cleartext
- Transport Layer Security (TLS)
  - ○ Provide encrypted TCP connections

    - Data integrity
    - End-point authentication

Web and HTTP

- URL: protocal name://host name/path name
    - Path name could be case sensitive depending on the system
- HTTP: hypertext transfer protocol (pull)
    - Client/server model
        - Browser requests/receives and displays web objects
        - Web server sends objects in response to requests
    - Uses TCP, port 80
    - HTTP is stateless
        - Does not maintain past client requests
        - So the protocol is simple
    - RTT: time for a small packet to travel from client to server and back
    - Two types
        - Non-persistent (need multiple TCPs for different requests)
            - □ At most one request
            - □ Response time (per object)=2RTT + file transmission time ($d_{prop}, d_{proc}, d_{queue}$),
                - ◆ transmission delay is not included, because the packet is very small
        - Persistent (only 1 TCP segment)
            - □ Multiple requests over the same server
    - Request messages
        - Request, response
        - Request message:
            - □ 

• ASCII (human-readable format)

```
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
```

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

* Check out the online interactive exercises for more

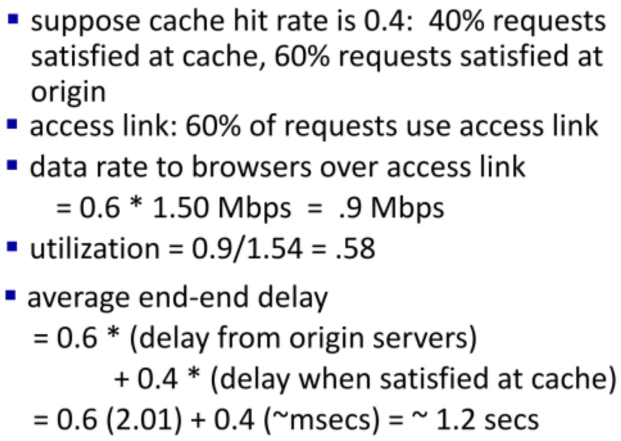
        - Response message:
            - □ May be empty, when (304 not modified)

Cookies:

- HTTP GET/response is stateless
- We use cookies to maintain user/server state between transactions
- Components
    - Cookie header line of HTTP response message
    - Cookie header line in next HTTP request message
    - Cookie file kept on user's host, managed by user's browser
    - Back-end database at website
- Used for
    - Authorization
    - Shopping carts
    - Recommendations
    - User session state
- Challenges
    - Protocol endpoints: maintain state at sender/receiver over multiple transactions
    - Cookies: HTTP messages carry state

Web caches (proxy servers)

- Goal: satisfy client request without involving origin sever

- User configures browser to point to a web cache
- Browser sends all HTTP requests to cache
    - If object in cache, cache returns the object
    - Else, cache request from origin server
- Cache acts as client and server
- Reason:
    - Reduce response time
    - Reduce traffic on institution's access link
    - Internet is dense with cache
    - Using less bandwidth
- Conditional GET
    - Send HTTP request to see if the cached object has changed
        - If not changed, 304 not modified, use cached object
        - Else, get the new object
- Example calculation
    - No cache:
        - When access link utilization is close to 1, will have large delays, but faster access link is expensive
    - With cache:
        - 

          - suppose cache hit rate is 0.4: 40% requests satisfied at cache, 60% requests satisfied at origin
          - access link: 60% of requests use access link
          - data rate to browsers over access link
            = 0.6 * 1.50 Mbps = .9 Mbps
          - utilization = 0.9/1.54 = .58
          - average end-end delay
            = 0.6 * (delay from origin servers)
            + 0.4 * (delay when satisfied at cache)
            = 0.6 (2.01) + 0.4 (~msecs) = ~ 1.2 secs

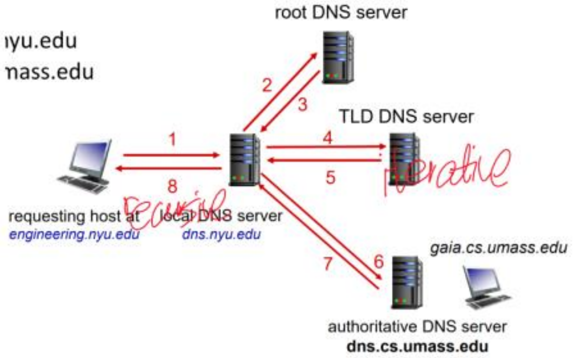
HTTP versions

- 1.1: introduced multiple, pipelined GETs over single TCP connection
    - Mostly is 1.1
- 2: increased flexibility at server in sending objects to client
    - Divides large objects into frames, frame transmission interleaved
    - Allows objects in a persistent connection to be sent in a client-specified priority order
    - Mitigate HOL (head-of-line) blocking
- 3: adds security, error and congestion control over UDP

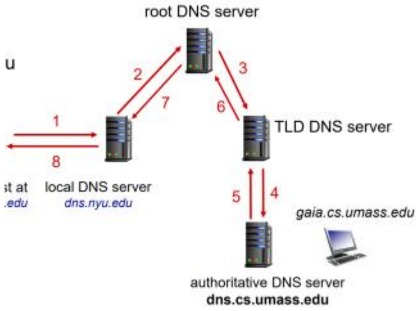
DNS: domain name system

- Distributed database implemented in hierarchy of many name servers
    - Root
        - Important Internet function, implemented as application-layer protocol
    - Top level domain (TLD)
        - Responsible for .com. .org, .net, TLD country domains .cn, .uk, and .edu
    - Authoritative
        - Organization's own DNS servers
        - Can be maintained by organization of service provider
    - Local DNS:
        - Does not strictly belong to hierarchy
        - Each ISP has one DNS
        - When host makes DNS query, query is sent to the local DNS
- DNS service
    - Hostname to IP address translation

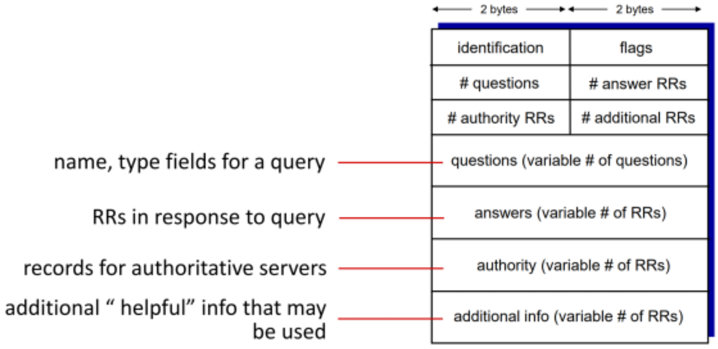
-
    - Host aliasing
        - Host with a complicated hostname as its canonical hostname can have one or more alias names. DNS can be invoked to obtain the canonical hostname for an alias hostname
    - Mail server aliasing
        - Similar to above but for the mail server
    - Load distribution
        - Load distribution among replicated servers. A host can have a set of IP addresses. The DNS server responds to DNS queries with the entire set of IP addresses, but rotates the ordering of the addresses within in each reply
- No centralized DNS
    - Single point of failure
    - Traffic volume
    - Distant centralized DNS
    - Maintenance
    - Doesn't scale
- DNS name resolution
    - Iterated query
        - Do whatever iteration is needed to get the IP
        -



-
    - Recursive query
        -



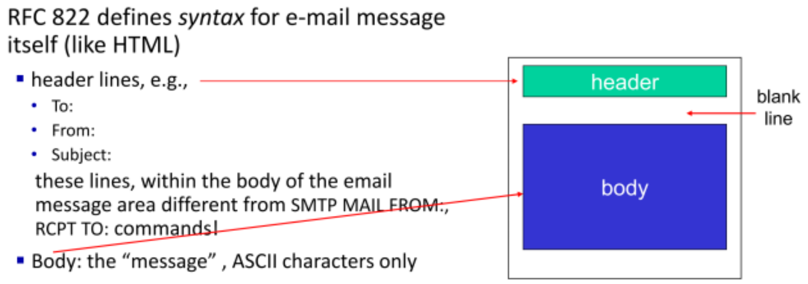
- DNS caching
    - Once the name server learns mapping, it caches it
- DNS resource records (RR)
    - Format (name, value, type, ttl)
    - Type A
        - Name is hostname
        - Value is IP address
    - Type NS
        - Name is domain
        - Value is hostname of authoritative name server for this domain
    - Type CNAME
        - Name is alias name for some canonical name
        - Value is canonical name
    - Type MX
        - Value is name of mail server associated with name
- DNS messages
    - Query and reply messages are the same format

    - Header:
        - Identification: 16 bits number for query, reply with the same number
        - Flags: query/reply, recursion desired, recursion available, reply is authoritative
    - 
2 bytes
2 bytes
identification
flags
# questions
# answer RRs
# authority RRs
# additional RRs
name, type fields for a query
questions (variable # of questions)
RRs in response to query
answers (variable # of RRs)
records for authoritative servers
authority (variable # of RRs)
additional " helpful" info that may be used
additional info (variable # of RRs)

- Inserting records
    - Register name at DNS registrar
        - Provide names, IP addresses of authoritative name server
        - Registrar inserts NS, A RRs into TLD server
    - Create authoritative server locally with DNS IP address
- DNS security
    - DDoS attacks (root servers (not successful), TLD servers (more dangerous))
    - Redirect attacks
    - Exploit DNS for DDoS

Email, SMTP, IMAP

- SMTP Major components
    - User agent
        - apps such as outlook
        - Outgoing, incoming messages stored on server
    - Mail servers
        - e.g. @ece.ubc.ca, @hotmail.com
        - Mailbox contains incoming messages for user
        - Message queue for outgoing mails
        - SMTP protocol between servers to send email messages
            - Client: sending mail server
            - Server: receiving mail server
    - Simple mail transfer protocol: SMTP (makes Email work)
        - SMTP is only push
        - Multiple objects sent in multipart message
        - Uses persistent connections
        - Requires message to be in 7-bit ASCII
        - Server uses CRLF.CRLF to determine end of message
    - RFC
        - Uses TCP port 25 to transfer message
            - Message must be 7-bit ASCII
        - Direct transfer (sending server to receiving server)
        - Three phases
            - Handshaking
            - Transfer
            - Closure
        - Command/response interaction
            - Commands: ASCII text
            - Response: status code and phrase
    - Message format

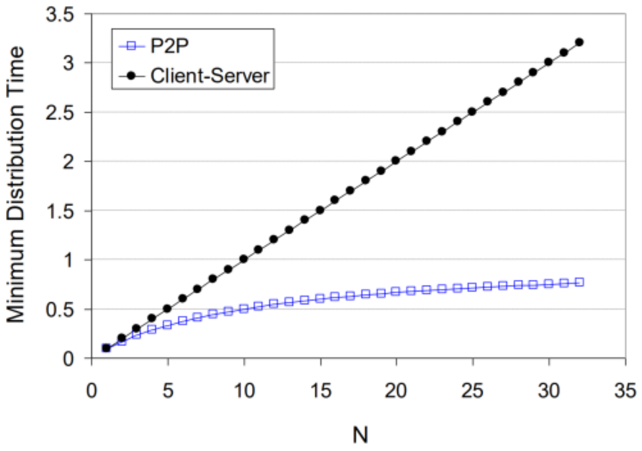
- 
  RFC 822 defines *syntax* for e-mail message itself (like HTML)
  - header lines, e.g.,
    - To:
    - From:
    - Subject:

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

- Can use persistent TCP
- Mail access protocols
  - SMTP only sends and stores messages
  - Use IMAP to retrieve from server (HTTP for web-based interface)

Peer-to-peer (P2P) architecture

- Characteristics
  - No always-on server, end systems directly communicate
  - Self-scalable
  - E.g. P2P file sharing (BitTorrent), streaming (KanKan), VoIP (Skype)
  - Peer upload/download capacity is limited
- File distribution time
  - Client server model
    - Server transmission (sequentially send N file copies)
      - □ Time to send one copy $\frac{F}{u_s}$ (u_s is the upload capacity)
      - □ Time to send N copies $\frac{NF}{u_s}$
    - Client: each client must download file copy
      - □ d_{min}=minimum client download rate
      - □ Maximum client download time:$\frac{F}{d_{min}}$
    - Time to distribute F to N clients using client-server approach
      - □ $D_{C-S} \geq max\{\frac{NF}{u_s}, \frac{F}{d_{min}}\}$
  - P2P, mostly similar
    - But for multiple clients, max upload rate is $u_s + \Sigma u_i$
    - So $\mathrm{D_{P2P}} \geq \max\{\frac{F}{u_s}, \frac{F}{d_{min}}, \frac{NF}{u_s+\Sigma u_i}\}$

    client upload rate = *u*, *F/u* = 1 hour, *u*$_s$ = *10u*, *d*$_{min}$ ≥ *u*$_s$
  - 

- E.g. BitTorrent
  - File divided into 256Kb chunks
  - Torrent: group of peers exchanging chunks of a file
    - Peers in torrent send/receive file chunks
    - Tracker: tracks peers participating in torrent
    - Churn: peers may come and leave
      - □ Once peer has entire file, it may leave or remain in torrent

        - Requesting chunks
            - At any given time, different peers have different subsets of file chunks
            - Requests missing chunks from peers, rarest first
        - Sending chunks
            - Send chunks to those who is sending the chunks at highest rate
            - Every 30 sec, randomly select another peer, and starts sending chunk

Video streaming and CDNs

- Challenge: Large traffic, scale, different users have different capabilities (heterogeneity)
- Manifest: a file containing the location and encoding rate of files corresponding to video segments in a video
- Solution: distributed, application-level infrastructure
- Video: sequence of images displayed at constant rate
    - Digital image: array of pixels, each pixel represented by bits
    - Coding: use redundancy within and between images to decrease # bits used to encode image
        - Spatial (within image)
            - □ Send only color and number of repeated values
        - Temporal (between images)
            - □ Send only differences from frame i for frame $i + 1$
    - CBR (constant bit rate): video encoding rate fixed
    - VBR (variable bit rate): video encoding rate changes as amount of spatial, temporal coding changes
        - Most commonly used
- Streaming video
    - Challenges:
        - sever-to-client bandwidth will vary due to congestion levels
        - Packet loss and delay causing poor video quality
    - Continuous playout constraint: once client playout begins, playback must match original timing
        - But network delays are variable, we need client-side buffer to match requirements
            - □ Buffering and playout delay compensate for network-added delay
    - Other challenges
        - Client interactivity: pause, fast-forward, rewind
        - Video packets may be lost, retransmitted
- Streaming multimedia DASH
    - DASH: dynamic adaptive streaming over HTTP
    - Server:
        - Divide video into multiple chunks
        - Store, encode at different rates
        - Manifest file provides URLs for different chunks
    - Client determines
        - When to request chunk
        - What encoding rate to request
        - Where to request
- Streaming video = encoding + DASH + playout buffering

CDN (content distribution networks):

- Challenge: how to stream content to hundreds of thousands of simultaneous users
- Solutions
    - Single, large mega-server
        - Doesn't scale
    - Store/serve multiple copies of videos at multiple geographically distributed sites (CDN)
        - Enter deep (close to users): push CDN servers deep into many access networks
        - Bring home: smaller number of larger clusters in POPs near access networks
- CDNs stores copies of content at CDN nodes

    - Subscriber requests content from CDN
- Over the top (OTT): Internet host-host communication as a service
    - Challenges: coping with a congested network

Socket programing

- Socket: door between application process and end-end-transport protocol
- UDP: unreliable datagram
    - No connection between client and server
    - Transmitted data may be lost or received our-of-order
    - Can receive data from different clients on the same socket
    - SOCK_DGRAM
    - Sender explicitly attaches IP destination address and port number to each packet
- TCP
    - Client must contact server
        - Create TCP socket
        - Client TCP establishes connection to server TCP
        - When contacted by client, server TCP creates new socket for server process to communicate with the client
        - Server can perform an accept
            - Binds the server and client together so that the server can identify the receiver
        - Client uses connect to explicitly bind its socket to specific server

Port numbers

- 22: SSH
- 25: SMTP (push, TCP)
- 53: DNS (TCP+UDP)
- 80: HTTP (pull)
- 143: IMAP (pull)

Chapter 3

October 6, 2020 10:13 AM

Transport layer (together with the network layer) is the heart of the protocol hierarchy of the Internet.

Provides process-to-process or application-to-application delivery

TCP and UDP are most important protocols in Transport layer

Transport layer actions

- Sender
  - Passed an application layer message
  - Determines segment header fields values
  - Creates segment
  - Passes segment to IP (network layer)
- Receiver
  - Receives segment from IP
  - Checks header values
  - Extracts application-layer message
  - Demultiplexes message to application via socket

Transport layer provides logical communication between application processes running on different hosts
Network layer provides logical communication between hosts

Protocols

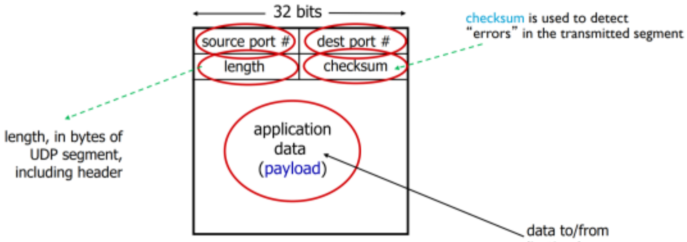
- TCP: Transmission Control Protocol
  - Reliable, in-order delivery
  - Congestion control
    - If the network is already congested
    - Deals with the network
  - Flow control
    - Don't overwhelm the network
    - Deals with two ends only
  - Connection setup (connection-oriented)
    - Started with creating a connection, then transfer data
    - Three way handshaking
- UDP: User Datagram Protocol
  - Unreliable, unordered delivery
  - No-frills extension of "best-effort" IP
  - Connectionless
    - No handshaking is needed

Multiplexing/demultiplexing

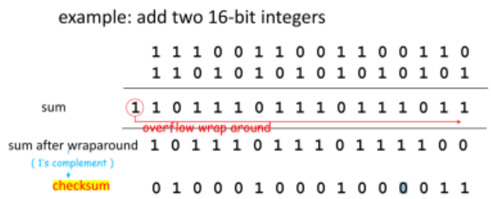
- Multiplexing/demultiplexing happen at all layers
- Multiplexing: (at sender) gather data chunks at the source host from different sockets, encapsulate data to create segments and pass segments to network layer
- Demultiplexing: (at receiver) delivering the data in a transport-layer segment to the correct socket.
  - Host use IP addresses and port numbers to direct segment to appropriate socket
  - Host receives IP datagrams
    - Source/destination IP address
    - One datagram carries one transport-layer segment

    - Each segment has source/destination port number
  - Connectionless demultiplexing
    - When creating datagram to send into UDP socket, we must specify destination IP address, port number
    - When receiving UDP segment, checks destination port number and direct the segment to the specific port
    - IP datagram with same destination port number will always be directed to the same socket at receiving host
    - Demultiplexing using destination port number only
  - Connection-oriented demultiplexing
    - TCP socket identified by 4-tuple
      - Source IP address
      - Source port number
      - Destination IP address
      - Destination port number
    - Receiver needs to use all four values to direct the segment to appropriate socket
    - Server may support multiple simultaneous TCP sockets

UDP (User Datagram Protocol)

- UDP segments may be lost, delivered out-of-order
- Advantages
  - Connectionless, reduce RTT delay
  - simple
  - Smaller header size
  - No congestion control
    - Can be as fast as desired
    - But bad for congestions if misuse
- UDP used in DNS, streaming multimedia apps, SNMP, HTTP/3
  - If reliable transfer needed over UDP (HTTP/3)
    - Add needed reliability/congestion control at application layer
- UDP layer actions
  - Sender:
    - Passed an application layer message
    - Determines UDP segment header
    - Create UDP segment
    - Pass segment to network layer (IP)
  - Receiver
    - Receives segment from network layer (IP)
    - Checks UDP checksum header value
      - Detect errors
      - 
        32 bits
source port #
dest port #
length
checksum
application data (payload)
checksum is used to detect "errors" in the transmitted segment
length, in bytes of UDP segment, including header
data to/from application layer

        UDP segment format
      - Checksum (16bit)
        - Checksum is computed over the entire segment, except the checksum field, but includes the IP sender and receiver address fields
        - Sender adds the segment content and put the sum in the field
        - Needs to wrap around the overflow and flip the bits

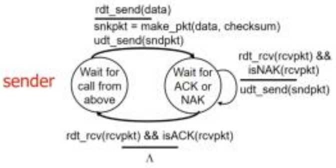
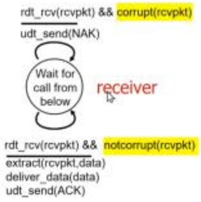
                    - 

                - Receiver compute checksum of received segment, check if the checksum equals to provided value
                    - If not equal, there is an error
                - A single flipped bit in one of the numbers will always result in a changed checksum
                - A single flipped bit in both of the two numbers may not lead to a changed checksum
        - Extracts application-layer message
        - Demultiplexes message up to application layer

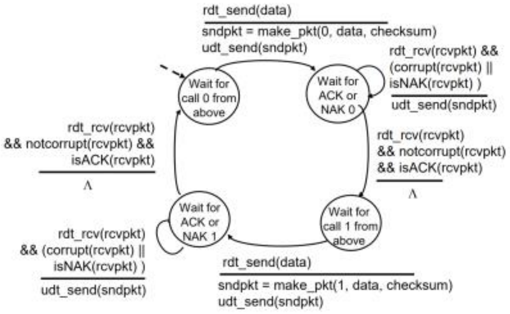
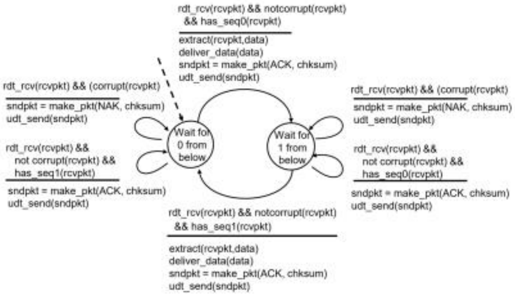
Principles of reliable data transfer

- With a reliable data transfer, no transferred data bits are corrupted or lost, and all are delivered in the order in which they were sent
- A packet that is received in error will be retransmitted by the sender
- ARQ (automatic repeat request)
    - For ARQ we need:
        - Error detection
        - Receiver feed back
            - ACK: (acknowledgements) receiver explicitly tells the sender that packet received is OK
            - NAK: (negative acknowledgements) receiver explicitly tells sender that packet had errors
        - Retransmission
    - Stop-and-wait: the simplest ARQ
        - Sender waits until it is sure that the receiver has correctly received the current packet, then send the new ones
    - Countdown timer: time-based retransmission mechanism
        - Interrupt the sender after timeout
        - Do not wait forever
    - Recover from errors
        - Resends the packet, but have duplicate packets
        - Put a sequence number

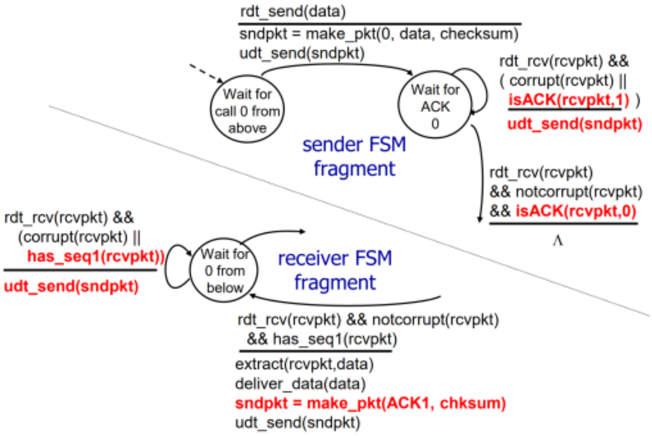
Reliable data transfer (rdt) interface

- Use finite state machine to specify sender, receiver
    - Separate FSMs for sender, receiver
    - Send data by underlying channels
    - Rdt2.0
        - 
rdt_send(data)
snkpkt = make_pkt(data, checksum)
udt_send(sndpkt)
sender
Wait for call from above
Wait for ACK or NAK
rdt_rcv(rcvpkt) && isNAK(rcvpkt)
udt_send(sndpkt)
rdt_rcv(rcvpkt) && isACK(rcvpkt)
Λ


        - ACK/NAK are corrupted
        - Duplicates
            - Using sequence number
    - Rdt2.1

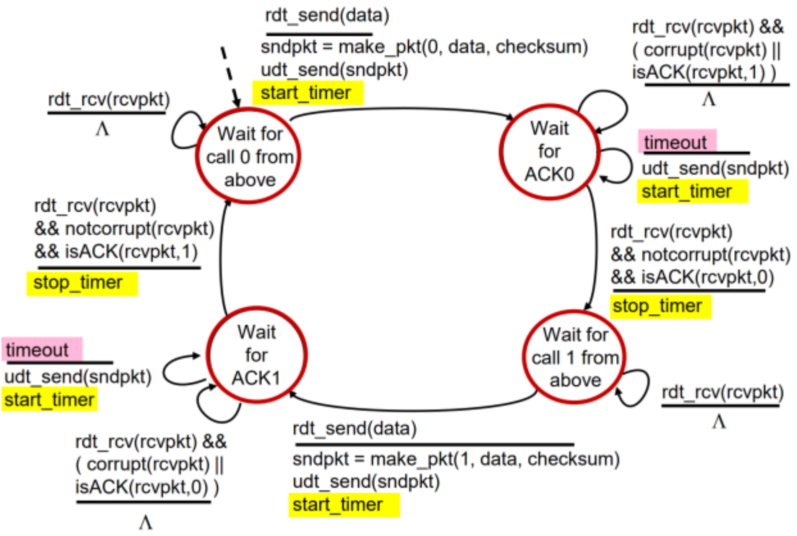
- Sender
  - 
rdt_send(data)
sndpkt = make_pkt(0, data, checksum)
udt_send(sndpkt)
Wait for call 0 from above
Wait for ACK or NAK 0
rdt_rcv(rcvpkt) && (corrupt(rcvpkt) || isNAK(rcvpkt))
udt_send(sndpkt)
rdt_rcv(rcvpkt) && notcorrupt(rcvpkt) && isACK(rcvpkt)
Λ
rdt_rcv(rcvpkt) && notcorrupt(rcvpkt) && isACK(rcvpkt)
Λ
Wait for ACK or NAK 1
Wait for call 1 from above
rdt_rcv(rcvpkt) && (corrupt(rcvpkt) || isNAK(rcvpkt))
udt_send(sndpkt)
rdt_send(data)
sndpkt = make_pkt(1, data, checksum)
udt_send(sndpkt)
  - Sequence number (0 or 1) added to the packet
  - Must check if received ACK/NAK is corrupted
  - State must remember whether expected packet should have sequence number of 0 or 1
- Receiver
  - 
rdt_rcv(rcvpkt) && notcorrupt(rcvpkt) && has_seq0(rcvpkt)
extract(rcvpkt,data)
deliver_data(data)
sndpkt = make_pkt(ACK, chksum)
udt_send(sndpkt)
rdt_rcv(rcvpkt) && (corrupt(rcvpkt)
sndpkt = make_pkt(NAK, chksum)
udt_send(sndpkt)
rdt_rcv(rcvpkt) && (corrupt(rcvpkt)
sndpkt = make_pkt(NAK, chksum)
udt_send(sndpkt)
rdt_rcv(rcvpkt) && not corrupt(rcvpkt) && has_seq1(rcvpkt)
sndpkt = make_pkt(ACK, chksum)
udt_send(sndpkt)
Wait for 0 from below
Wait for 1 from below
rdt_rcv(rcvpkt) && not corrupt(rcvpkt) && has_seq0(rcvpkt)
sndpkt = make_pkt(ACK, chksum)
udt_send(sndpkt)
rdt_rcv(rcvpkt) && notcorrupt(rcvpkt) && has_seq1(rcvpkt)
extract(rcvpkt,data)
deliver_data(data)
sndpkt = make_pkt(ACK, chksum)
udt_send(sndpkt)
  - Must check if received packet is duplicated (whether 0 or 1 is expected packet sequence number)

○ Rdt2.2: similar to 2.1 but NAK-free (TCP uses this)

- Instead of NAK, receiver sends ACK for last packet received OK (explicitly include sequence number)
- Duplicate ACK at sender results in same action as NAK
- 
rdt_send(data)
sndpkt = make_pkt(0, data, checksum)
udt_send(sndpkt)
Wait for call 0 from above
Wait for ACK 0
rdt_rcv(rcvpkt) && (corrupt(rcvpkt) || isACK(rcvpkt,1))
udt_send(sndpkt)
sender FSM fragment
rdt_rcv(rcvpkt) && notcorrupt(rcvpkt) && isACK(rcvpkt,0)
Λ
rdt_rcv(rcvpkt) && (corrupt(rcvpkt) || has_seq1(rcvpkt))
udt_send(sndpkt)
Wait for 0 from below
receiver FSM fragment
rdt_rcv(rcvpkt) && notcorrupt(rcvpkt) && has_seq1(rcvpkt)
extract(rcvpkt,data)
deliver_data(data)
sndpkt = make_pkt(ACK1, chksum)
udt_send(sndpkt)

○ Rdt3.0: channels with errors and loss

- Stop and wait for reasonable amount of time for ACK
  - Retransmits if no ACK received in this time
  - Using a countdown at sender
  - Receiver must specify sequence number of packet being ACK

    - 

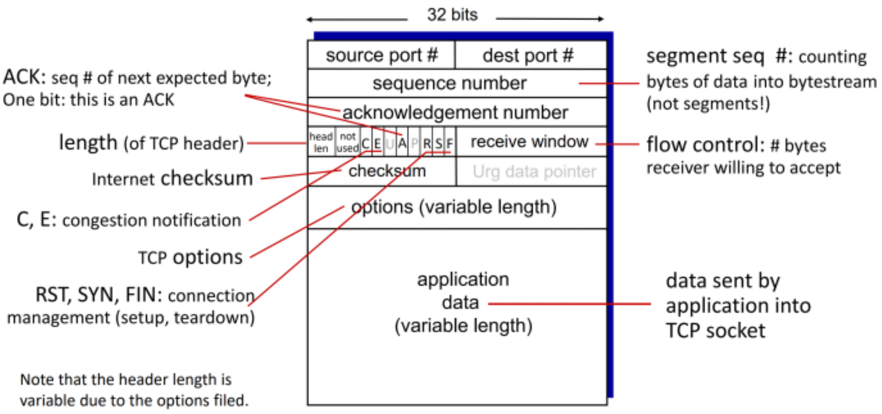
    - Performance of rdt3.0
      - $U_{sender} = \frac{\frac{L}{R}}{RTT+\frac{L}{R}}$: utilization-fration of time sender busy sending
        - Usually very low
      - Performance is bad
    - Pipeline
      - Sender allows multiple in-flight, yet-to-be-acknowledged packets
- In rdt2.0-2.2 we assume underlying channel perfectly reliable
  - No bit errors
  - No loss of packets
  - May flip bits in packet
  - Use ACK, NAK to tell the sender the packet received are OK or not
    - Retransmit if NAK
- Stop and wait
  - Sender sends one packet, then waits for receiver response
- Pipelined
  - Allows multiple, "in-flight", yet-to-be-acknowledged packets
  - Performance $U_{sender} = \frac{n\frac{L}{R}}{RTT+\frac{L}{R}}$, n is the pipelined packet number
- Go back N (GBN)
  - Sender transmit multiple packets without waiting for an acknowledgement
    - Cumulative ACK: ACKs all packets up to, including sequence number n
      - On receiving, move window forward to begin at $n+1$
    - Timer for oldest in-flight packet
    - Timeout: retransmit packet n and all higher sequence number packets in window
  - Constrained to have no more than some maximum allowable number N of unacknowledged packets in the pipeline
  - Receiver
    - Ack-only, send ACK for correctly-received packet so far, with highest in-order sequence number
    - Receiving out of order packet
      - Discard or buffer
        - Discard because the sender will resend the packet anyway, and the implementation at the receiver is simpler
        - Buffer because it is perfectly well-received, but next transmission may be corrupted
      - Re-ACK with highest in order sequence number
- Selective repeat
  - Avoid unnecessary retransmission, only retransmit the error received packets
  - Sender: times out/ retransmits individually for un-ACK packets

  - Maintains timer for each un-ACK packet
    - If time out, resend packet and restart timer
  - Receive ACK for n in $[sendbase, sendbase + N]$
    - Mark packet n as received
    - If n smallest un-ACK packet, advance window base to next un-ACK sequence number
- Receiver individually acknowledges all correctly received packets
  - Buffers packets for eventual in-order delivery to upper layer
  - If packet is within $[rcvbase, rcvbase + N - 1]$
    - Send ACK
    - Out-of-order packets go in buffer
    - In order buffer delivered to higher layer
  - If packet is in previous receive base $[rcvbase \ - N, rcvbase - 1]$
    - Only send ACK
  - Otherwise ignore
  - Cannot receive a packet with a sequence number larger than the window size N
- Sender window
  - N consecutive sequence number
  - Limit sequence number of sent, un-ACK packets
- May have problem when sequence number base is greater than window size
  - E.g. sequence number = 0,1,2,3 with window size 3

TCP

- Point to point: one sender, one receiver
- Reliable: in order byte stream
  - No message boundaries
- Full duplex data
  - Bi-directional data flow in the same connection
  - MSS: maximum segment size
- Cumulative ACKs
  - The acknowledgment field has the sequence number of the next byte expected from the other side
- Pipelining
  - TCP congestion and flow control set window size
- Connection oriented
  - Handshaking initializes sender, receiver state before data exchange
  - Before data transfer: Three-way handshake
    - Agree to establish connection
    - Agree on connection parameters (e.g. starting sequence numbers)
    - SYN, SYNACK, ACK
    - 2-way handshake may not always work in network
      - Variable delays
      - Retransmitted messages (request connection) due to message loss
      - Message reordering
      - Can't see other side
  - Closing a connection
    - Client and server close their side of connection
      - Send TCP with FIN bit = 1
    - Respond to received FIN with ACK
      - Can be combined with own FIN
    - Simultaneous FIN exchanges can be handled
- Flow controlled
  - Sender will not overwhelm receiver
  - Receiver advertises free buffer space in rwnd field (receive window) in TCP header
    - Receive buffer size set via socket options (default is 4096 bytes)
    - Many OS adjust the size automatically
  - Sender limits amount of NACK (in flight) data to rwnd

    - Guarantees receive buffer will not overflow
- TCP segment structure


32 bits
source port #
dest port #
sequence number
acknowledgement number
head len
not used
C
E
U
A
P
R
S
F
receive window
checksum
Urg data pointer
options (variable length)
application data (variable length)
ACK: seq # of next expected byte; One bit: this is an ACK
length (of TCP header)
Internet checksum
C, E: congestion notification
TCP options
RST, SYN, FIN: connection management (setup, teardown)
Note that the header length is variable due to the options filed.
segment seq #: counting bytes of data into bytestream (not segments!)
flow control: # bytes receiver willing to accept
data sent by application into TCP socket

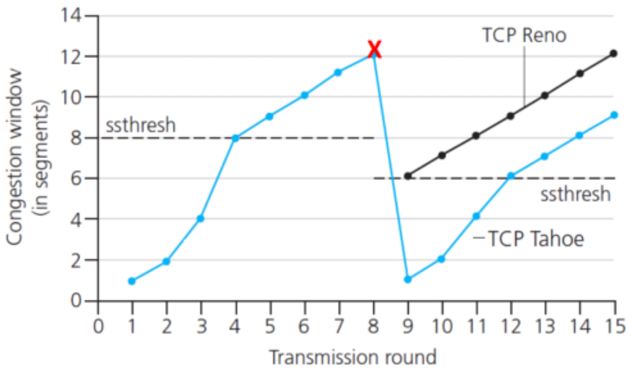
    - No length for payload
        - Calculated by total length - IP header length - TCP header length
    - Header length is 4 bit
        - But minimum value is 5: 20 bytes header
        - Maximum value is 15: maximum for 4 bit
- Sequence number
    - Used to identify the byte stream number of the first byte in the segment data
- TCP timeout, use estimated TCP RTT
    - $EstimatedRTT = (1-\alpha)EstimatedRTT\ + \alpha \times SampleRTT$
        - Exponential Weighted Moving Average
        - Influence of past sample decreases exponentially fast
        - Typically $\alpha\ = 0.125$
    - We will plus a safety margin to the estimated RTT for timeout interval
        - $DevRTT = (1-\beta)DevRTT + \beta\left|SampleRTT - EstimatedRTT\right|$
        - Timeout interval = $EstimatedRTT\ +\ 4DevRTT$
- Simplified sender
    - Event: data received from application
        - Create segment with sequence number
        - Sequence number is byte-stream number of first data byte in segment
        - Start timer if not already running
            - Timer: for oldest NACK segment
            - Expiration interval: timeout interval
    - Event: timeout
        - Retransmit segment that caused timeout
        - Restart timer
    - Event: ACK received
        - Update what is known to be ACK
        - Start timer if there are still NACK segment
- Simplified receiver
    - Event: arrival of in-order segment with expected sequence number and all data up to this already ACK
        - Delayed ACK. Wait up to 500ms for next segment, if no next segment, send ACK
    - Event: arrival of in-order segment with expected sequence number, but one other segment has ACK pending
        - Immediately send single cumulative ACK for both in-order segments
    - Event: arrival of out-of-order segment higher than expected sequence number
        - Immediately send duplicate ACK, indicating sequence number of next expected byte
    - Event: arrival of segment that partially or completely fills the gap
        - Immediately send ACK, provided that segment starts at lower end of gap
    - Always send the highest in-order ACK
- Fast retransmit
    - If sender receives 3 additional ACKs for the same data (triple duplicate ACKs), resend

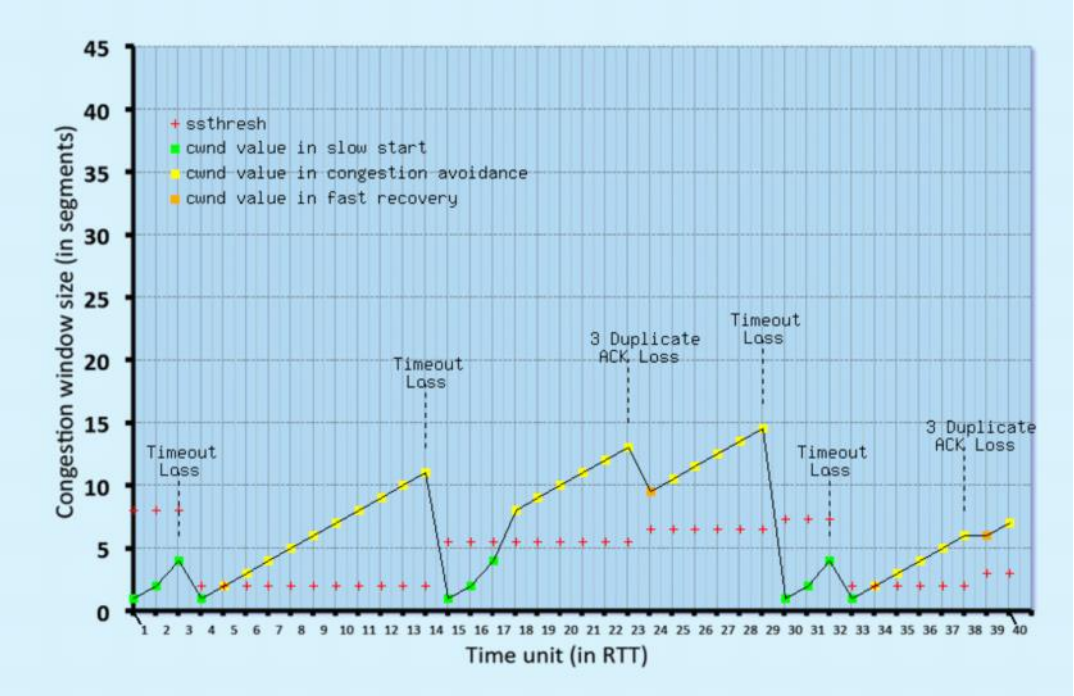
NACK segment with smallest sequence number

Congestion Control

- Congestion: too much data (sender) too fast for network to handle
    - Long delays (queuing)
    - Packet loss (buffer overflow)
- Flow: too much data for the receiver to handle
- End-end congestion control
    - No explicit feedback from network
    - Congestion inferred from end-system observed loss, delay
    - Approach taken by TCP
- Network-assisted congestion control
    - Routers provide feedback to end system
    - Not currently in use

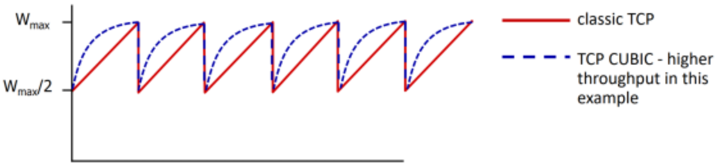
TCP congestion control: AIMD

- Additive Increase: increase sending rate by 1 maximum segment size every RTT until loss detected
- Multiplicative Decrease: cut sending rate in half at each loss event
    - Cut in half on loss detected by triple duplicate ACK (TCP Reno)
    - Cut to 1 MSS (maximum segment size) when loss detected by timeout (TCP Tahoe)
- Optimize congested flow rates network-wide
- Have desirable stability properties
- Sending behavior:
    - Send cwnd (congestion window size) bytes, wait RTT for ACKs, then send more bytes
        - $TCP\ rate\ = \frac{cwnd}{RTT}$ bytes/sec
    - TCP sender limits transmission:
        - $LastByteSent\ -\ LastByteAcked \leq\ cwnd$
    - Cwnd is dynamically adjusted in response to observed network congestion
        - $LastByteSent\ -\ LastByteAcked \leq\ min\ (cwnd, rwnd)$ handles both flow control and congestion control
        - rwnd (receive window size)
- TCP slow start
    - When connection begins, increase rate exponentially until first loss event
        - Initial cwnd = 1 MSS
        - Double cwnd every RTT for every ACK received
        - Or we can say cwnd is increased by 1MSS for every first ACK
- Congestion avoidance (when hits ssthresh)
    - Switch to linear increase, when cwnd gets to 1/2 of its value before timeout (ssthresh)
    - On a loss event due to triple duplicate ACK
        - Cwnd set to 1/2 of original cwnd (+3 to account for the triple duplicate ACKs) before loss event, also continues to increase linearly (fast recovery)
    - On a loss event due to timeout
        - Cwnd set to 1, ssthresh set to 1/2 cwnd, restart slow start
    - 

- 

- TCP detecting reacting to loss
  - Loss due to timeout:
    - Cwnd set to 1 MSS
    - Then follows slow start
  - Loss due to triple duplicate ACK:
    - TCP RENO:
      - Fast recovery
      - Cwnd is cut in half, then grows linearly
    - TCP Tahoe:
      - Always set cwnd to 1

TCP CUBIC

- 

- Better way than AIMD to probe for usable bandwidth
- W_{max}: sending rate at which congestion loss was detected
- Congestion state of bottleneck link hasn't changed much
- After cutting rate/window in half on loss, initially grow faster, then slowly
- K: point in time when TCP window size reaches W_{max}
- Increase W as a function of the cube of the distance between current time and K
  - Larger increases when further away from K
  - Smaller increases when nearer K

Bottleneck link

- Packet loss occurs at some router's output, the router is the bottleneck link
- Delay-based control:
  - Keep sender-to-receiver pipe busy transmitting, but avoid high delays/buffering
  - RTT_{min} - minimum observed RTT (uncongested path)
  - Uncongested throughput with congestion window cwnd is $\frac{cwnd}{RTT_{min}}$
    - If measured throughput close to uncongested throughput (RTT is small), increase cwnd linearly
    - If measured throughput far below uncongested throughput (RTT is large), decrease cwnd linearly

    - Congestion control without inducing/forcing loss
    - Maximizing throughput, while keeping delay low

Explicit congestion notification (ECN):

- Network-assisted congestion control
    - Two bits in IP header marked by network router to indicate congestion
        - Policy to determine marking chosen by network operator
    - Congestion indication carried to destination
    - Destination sets ECE bit on ACK segment to notify sender of congestion
    - Involves both IP (ECN bit) and TCP (C, E bits)

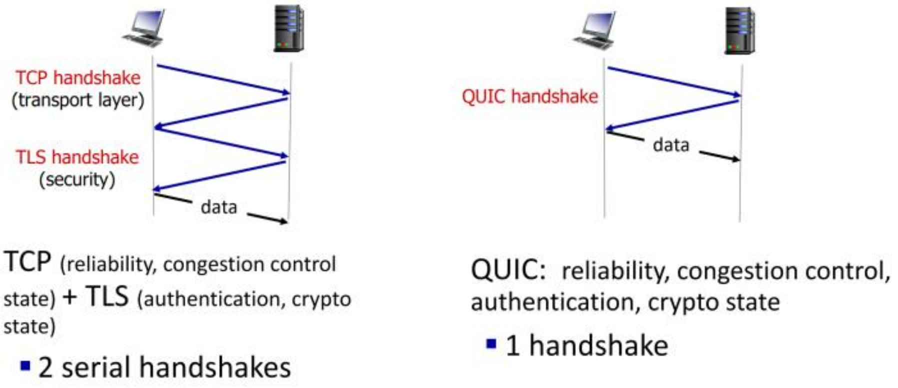
TCP fairness

- If K TCP sessions share same bottleneck link of bandwidth R, each should have average rate of $\frac{R}{K}$
- TCP is fair under idealized assumptions
    - Same RTT
    - Fixed number of sessions only in congestion avoidance
- Fairness and parallel TCP connections
    - Application can open multiple parallel connections between two hosts
    - Web browser do this
        - E.g. link of rate R with 9 existing connections
            - □ New app asks for 1 TCP, gets rate R/10
            - □ New app asks for 11 TCPs, gets R/2

Fairness and UDP:

- Multimedia apps often do not use TCP
    - Do not want rate throttled by congestion control
- Use UDP
    - Send audio/video at constant rate, tolerate packet loss
- No policing use of congestion control

HTTP/3: QUIC (Quick UDP Internet Connections):

- Moving transport-layer functions to application layer, on top of UDP
- Application-layer protocol, increase performance of HTTP
- Error and congestion control
- Connection establishment
- Multiple application-level streams multiplexed over single QUIC connection
    - Separate reliable data transfer, security
    - Common congestion control
- Connection establishment
    - Can establish all connection parameters in just one handshake rather than in two
    - 

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

      ▪ 2 serial handshakes

      QUIC: reliability, congestion control, authentication, crypto state

      ▪ 1 handshake

- As an application layer protocol QUIC can be updated/modified at app frequency rather than at the frequency of operating system updates

Chapter 4

October 29, 2020 9:51 AM

Network layer services and protocols

- Transport segment from sending to receiving host
  - Sender: encapsulates segments into datagrams pass to link layer
  - Receiver: delivers segment to transport layer
- Network layer protocols in every internet device(hosts and routers)

Network-layer functions:

- Forwarding: local action of moving packets from a router's input link to appropriate router output link
- Routing: global action of determining route taken by packets from source to destination (provides forwarding table)

Data plane and control plane

- Data plane:
  - Local, per-router function
  - Determines how datagram arriving on router input port is forwarded to router output port
  - Looking up address bits in an arriving datagram header in the forwarding table
  - Moving an arriving datagram from a router's input port to output port
  - Dropping a datagram due to a congested output buffer
- Control plane:
  - Network-wide
  - Determines how datagram is routed among routers along end-end path from source host to destination host
  - Computing the contents of the forwarding table
  - Monitoring and managing the configuration and performance of an IP device
  - Two approaches
    - Traditional routing algorithms: implemented in routers (per-router model)
      - Individual routing algorithm components in each and every router interact in the control plane
      - Routers send information about their incoming and outgoing links to other routers in the network
    - Software-defined networking (SDN): implemented in servers
      - Remote controller computes, installs forwarding tables in routers
    - In both ways:
      - A router uses its forwarding table to determine the appropriate outgoing link for an arriving datagram
      - A router receives an incoming datagram from a directly attached host

Network service models

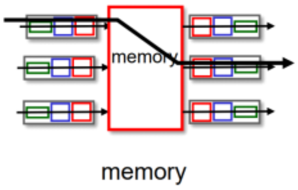
- For individual datagrams
  - Guaranteed delivery in some time delay
- For a flow of datagrams
  - In-order datagram delivery
  - Guaranteed minimum bandwidth
  - Restrictions on changes in inter-packet spacing
- Internet best effort services has no guarantees on
  - Successful datagram delivery to destination
  - Timing or order of delivery
  - Bandwidth available to end-end flow
- Reflections on Internet best-effort service

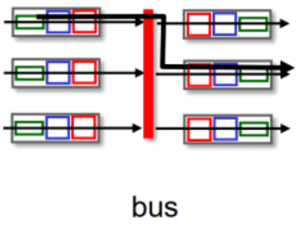
  - Simplicity of mechanism allows Internet to be widely deployed
  - Sufficient provisioning of bandwidth allows performance of real-time applications
  - Replicated, application-layer distributed services allow services to be provided from multiple locations
  - Congestion control of elastic services

Router architecture

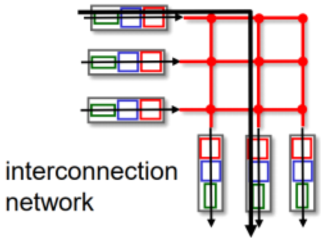
- Input port functions (match + action)
  - Line termination: physical layer bit level reception
  - Link layer protocol: link layer
  - Lookup, forwarding, queuing: decentralized switching
    - Using header field values, lookup output port using forwarding table in input port memory
    - Need to complete input port processing at line speed
    - Input port queuing: if datagrams arrive faster than forwarding rate into switch fabric
    - Destination-based forwarding (traditional): forward based only on destination IP address
      - Longest prefix match: when looking for forwarding table entry for given destination address, use longest address prefix that matches destination address
        - Often used Ternary Content Addressable Memories (TCAMs)
          - Content addressable: Retrieve address in one clock cycle, regardless of table size
    - Generalized forwarding: forward based on any set of header field values

Switching fabrics:

- Transfer packet from input link to appropriate output link
- Switching rate: rate at which packets can be transfer from inputs to outputs
  - Measured as multiple of input/output line rate
    - N inputs: NR (R is the line rate desirable)
- Types
  - Memory
    - 

    - Traditional computers with switching under direct control of CPU
    - Packet copied to system's memory
    - Speed limited by memory bandwidth
    - 2 bus crossing per datagram
  - Bus
    - 

    - Datagram from input port to output port memory via a shared bus
    - Speed limited by bus bandwidth
    - 32 Gbps bus: sufficient speed for access routers
  - Interconnection network

- 

- Crossbar, clos network, other interconnection nets initially developed to connect processors in multiprocessor
- Multistage switch: $n \times n$ switch from multiple stages of smaller switches
- Exploiting parallelism
  - □ Fragment datagram into fixed length cells on entry
  - □ Switch cells through the fabric reassemble datagram at exit
- Scaling, using multiple switching planes in parallel
  - □ Speed up, scale up via parallelism
- Cisco CRS router
  - □ Basic unit: 8 switching planes
  - □ Each plane: 3-stage interconnection network
  - □ Up to 100's Tbps switching capacity

Input, output port queuing

- Input queuing:
  - Switch fabric slower than input ports combined causes queuing delay and input buffer overflow
  - Head-of-the-Line (HOL) blocking: queued datagram at front of queue prevents others in queue from moving forward
- Output queuing:
  - Buffering required when datagrams arrive from fabric faster than link transmission rates
  - Scheduling discipline chooses among queued datagrams for transmission
    - Prioritize who gets best performance, network neutrality

Buffering:

- RFC 3439: average buffering equal to typical RTT (250 msec) times link capacity C
- With N flows, buffering equal to $\frac{RTT \cdot C}{\sqrt{N}}$
- Too much buffering can increase delay
  - Long RTTs
  - We need to keep bottleneck link just full enough but not fuller
- Buffer management
  - Drop when buffer is full
    - Tail drop: drop arriving packet
    - Priority: drop/remove based on priority
  - Marking: which packets to mark to signal congestion (ECN, RED bits)

Packet scheduling

- Deciding which packet to send next on link
- Methods:
  - First come fist serve (FCFS/FIFO)
    - Packets transmitted in order of arrival
  - Priority
    - Arriving traffic classified, queued by class
      - □ Any header fields can be used for classification
    - Send packet from highest priority queue that has buffered packets
      - □ FCFS within the queue
  - Round robin
    - Arriving traffic classified, queued by class
    - Server cyclically, repeatedly scans class queues, sending one complete packet from

each class in turn

- Weighted fair queuing
    - Generalized round robin
    - Each class i has weight w_i, and gets weighted amount of service $\frac{w_i}{\Sigma w_j}$ in each cycle
    - Minimum bandwidth guarantee (per-traffic-class)

Network neutrality:

- Technical: how an ISP should share/allocate its resources
    - Packet scheduling, buffer management are the mechanisms
- Social, economic principles
    - Protect free speech
    - Encourage innovation, competition
- Enforced legal rules and policies
    - No blocking
    - No throttling
    - No paid prioritization

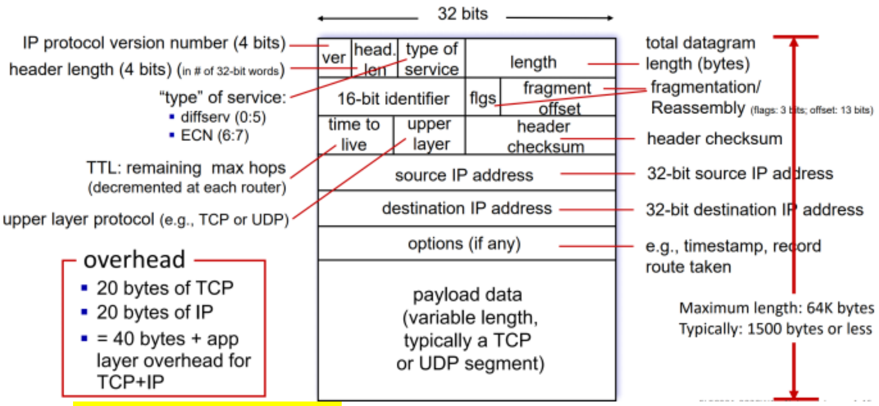
IP Protocol:

- Datagram format
- Addressing
- Packet handling conventions

ICMP protocol:

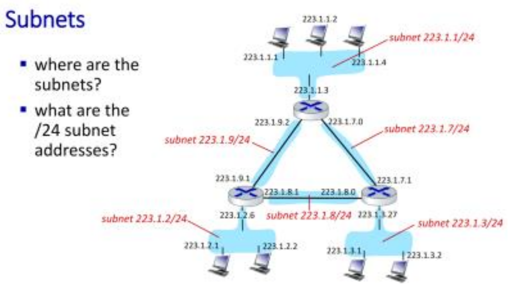
- Error reporting
- Router signaling

IP datagram:

- 

32 bits
IP protocol version number (4 bits)
header length (4 bits) (in # of 32-bit words)
"type" of service:
diffserv (0:5)
ECN (6:7)
TTL: remaining max hops (decremented at each router)
upper layer protocol (e.g., TCP or UDP)
overhead
20 bytes of TCP
20 bytes of IP
= 40 bytes + app layer overhead for TCP+IP
ver
head. len
type of service
length
16-bit identifier
flgs
fragment offset
time to live
upper layer
header checksum
source IP address
destination IP address
options (if any)
payload data (variable length, typically a TCP or UDP segment)
total datagram length (bytes)
fragmentation/ Reassembly (flags: 3 bits; offset: 13 bits)
header checksum
32-bit source IP address
32-bit destination IP address
e.g., timestamp, record route taken
Maximum length: 64K bytes
Typically: 1500 bytes or less

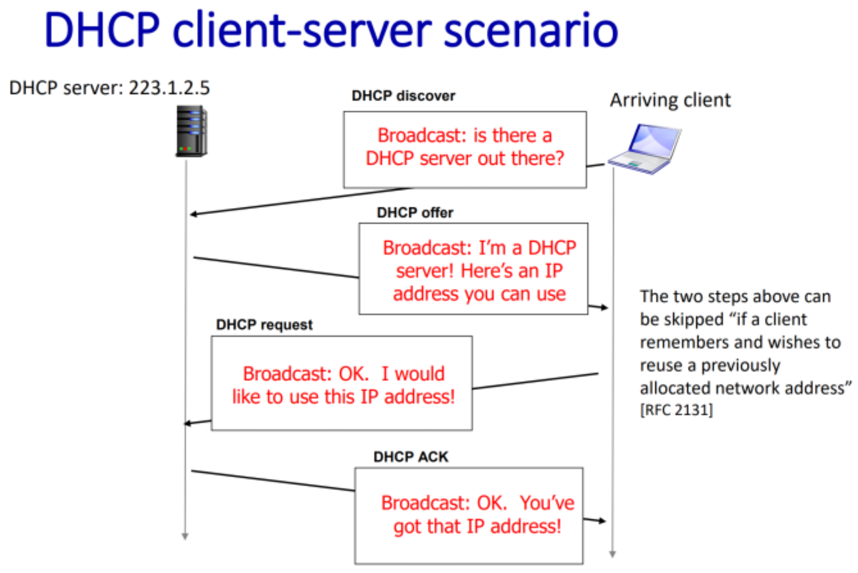
    - Identification (ID) field is a 16-bit value that is unique for every datagram for a given source address, destination address, and protocol, such that it does not repeat within the maximum datagram lifetime
- IP address: 32-bit identifier associated with each host or router interface
    - Interface: connection between host/router and physical link
        - Routers typically have multiple interfaces
        - Host typically has one or two interfaces (wired, wireless)
        - IP addresses associated with each interface
- Subnet
    - Device/ device interfaces that can physically reach each other without passing through an intervening router
    - IP address structure
        - Subnet part: devices in the same subnet have common high order bits
        - Host part: remaining low order bits (max number of host = $2^x - 2$)
        - Subnet mask: /n is the subnet mask indicating that the left most 24 bits define the subnet address
    - Each isolated network (isolated by routers) is a subnet

- 

  Subnets

  - where are the subnets?
  - what are the /24 subnet addresses?

  - There are 6 subnets

- CIDR (Classless InterDomain Routing)
  - Subnet can have an arbitrary length
  - Format a.b.c.d/x, where x is the number of bits in subnet portion
  - Broadcast address is the last IP in the range
  - Ending address is broadcast address-1
  - Previously less flexible (classful addressing)
    - Class A similar to /8
    - Class B similar to /16
    - Class C similar to /24
    - Causes poor utilization of addresses and rapid depletion of classes
    - Though class C is still used
- DHCP (Dynamic Host Configuration Protocol)
  - Host dynamically obtains IP address from network server when it joins network
    - Plug and play: no manual configuration
    - Can renew its lease on address in use
    - Allows reuse of address (only hold address while connected)
    - Support for mobile users
    - Provide IP address, deliver the address of first-hop router, the name and IP of DNS server and the network mask
  - Overview
    - Host broadcasts DHCP discover message (optional when already know which network to join)
    - DHCP server responds with DHCP offer message (optional when already know which network to join)
    - Host requests IP address: DHCP request message
    - DHCP server sends address: DHCP ack message
  - 

    DHCP client-server scenario

    DHCP server: 223.1.2.5

    Arriving client

    DHCP discover

    Broadcast: is there a DHCP server out there?

    DHCP offer

    Broadcast: I'm a DHCP server! Here's an IP address you can use

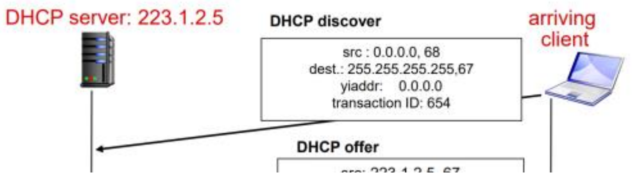
    DHCP request

    Broadcast: OK. I would like to use this IP address!

    DHCP ACK

    Broadcast: OK. You've got that IP address!

    The two steps above can be skipped "if a client remembers and wishes to reuse a previously allocated network address" [RFC 2131]



DHCP server: 223.1.2.5

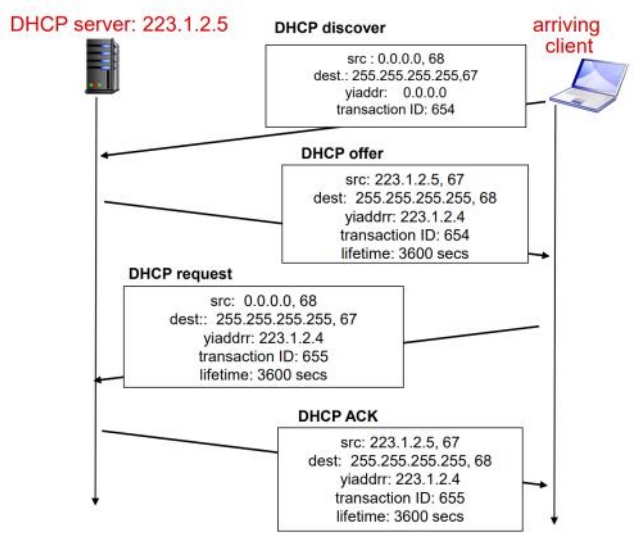
arriving client

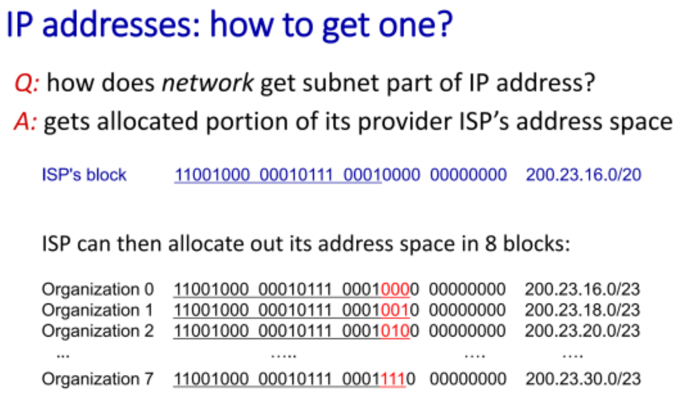
DHCP discover

src : 0.0.0.0, 68
dest.: 255.255.255.255,67
yiaddr: 0.0.0.0
transaction ID: 654

DHCP offer

src: 223.1.2.5, 67

- 

- 

IP addresses: how to get one?

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

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

ISP's block	11001000 00010111 00010000 00000000	200.23.16.0/20

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

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

- DHCP uses UDP

Hierarchical addressing

- Route aggregation: allows efficient advertisement of routing information
- More specific routes

ICANN (Internet Corporation for Assigned Names and Numbers)

- ICANN allocated addresses through 5 regional registries
- Manages DNS root zone, including delegation of individual TLD management
- ISP get blocks of addresses by ICANN

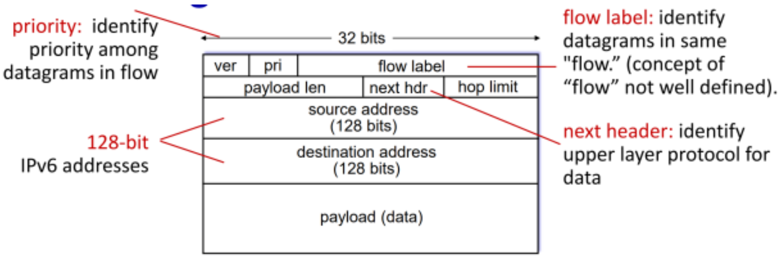
Last chunk of IPv4 addresses were allocated to RRs in 2011
NAT helps IPv4 address space exhaustion
IPv6 has 128-bit address space

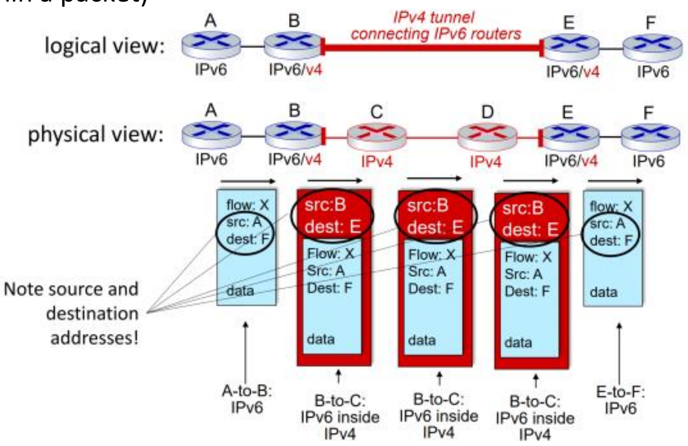
NAT (network address translation):

- All devices in local network share just one IPv4 address as far as outside world is concerned
- All datagrams leaving local network have the same source NAT IP address
- Datagrams with source or destination within the network have a private IP address
    - 10/8, 172.16/12, 192.168/16 can only be used in local network
- Advantages
    - Just one IP address needed from provider ISP for all local devices
    - Can change addresses of host in local network without notifying outside world
    - Can change ISP without changing addresses of devices in local network
    - Security: devices inside local net not directly addressable, visible by outside world
- Implementation: NAT router
    - Outgoing datagrams: replace source IP address, port number of every outgoing datagram to NAT IP address and port number

        - Remote clients/servers respond using NAT IP address
    - Remember (in translation table) every translation pair
    - Incoming datagrams: replace destination fields with corresponding local IP, port in NAT table
- 16-bit port number field can store more than 60000 simultaneous connections with single LAN-side address
- Extensively used in home/institutional nets, 4G/5G cellular nets
- Problems
    - Routers should only process up to layer 3 (network layer)
    - Address shortage solved by IPv6
    - Violates end-to-end argument (port number manipulation)
    - NAT traversal (want to skip the NAT to connect to the server behind it)

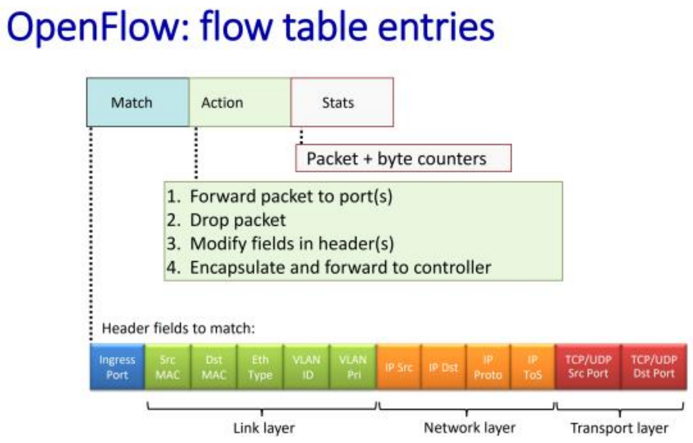
IPv6:

- Motivation: 32-bit IPv4 address space would be completely allocated
    - Use 128-bit IP addresses instead
    - Additional motivation:
        - New header format helps speed processing/forwarding
            - □ 40-byte fixed length header
        - Enable different network-layer treatment of flows
- Datagram
    - 
priority: identify priority among datagrams in flow
32 bits
ver
pri
flow label
payload len
next hdr
hop limit
source address (128 bits)
destination address (128 bits)
payload (data)
128-bit IPv6 addresses
flow label: identify datagrams in same "flow." (concept of "flow" not well defined).
next header: identify upper layer protocol for data

      What's missing (compared with IPv4):
      - no checksum (removed entirely to reduce processing time at each router hop)
      - no fragmentation/reassembly (disallowed to speed up IP forwarding)
      - no options (removed from the header to simplify it (fixed size); they are still allowed, but outside of header, indicated by "Next Header" field.))
- Transition from IPv4 to IPv6
    - Not all routers can be upgraded simultaneously, network operates with mixed IPv4 and IPv6
    - Tunneling
        - IPv6 datagram carried as payload in IPv4 datagram among IPv4 routers packet within a packet)
        - 

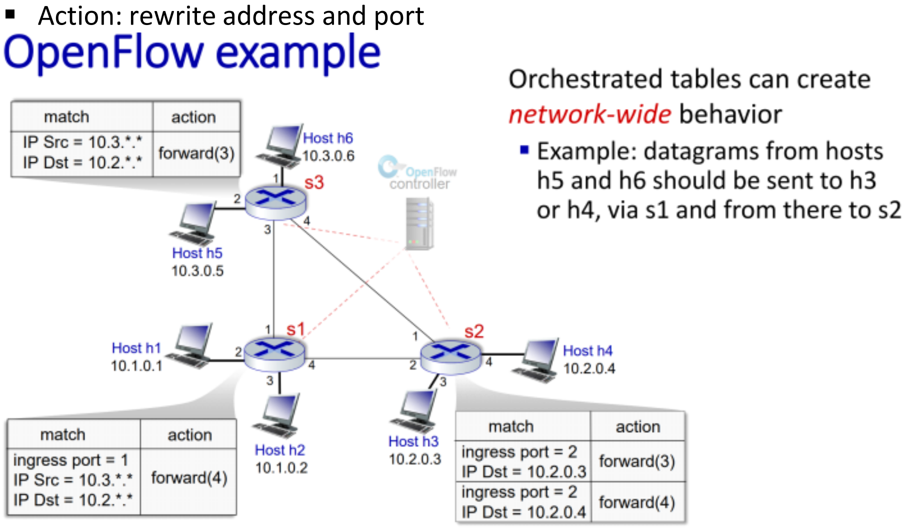
Generalized forwarding: match plus action:

- Each router contains a forwarding table (flow table)
- Match plus action abstraction: match bits in arriving packet, take action
    - Destination-based forwarding: forward based on destination IP address
    - Generalized forwarding:
        - Many header fields can determine action

        - Many action possible: drop/copy/modify/log packet
- Flow table abstraction
    - Flow: defined by header field values
    - Generalized forwarding: simple packet-handing rules
        - Match: pattern 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: number of bytes and packets
    - 

      OpenFlow: flow table entries

      1. Forward packet to port(s)
      2. Drop packet
      3. Modify fields in header(s)
      4. Encapsulate and forward to controller

        - Match-able values:
            - □ IP source/destination address
            - □ IP type of service
            - □ Upper layer protocol field
            - □ Source/destination port number
- Open flow abstraction
    - Match + action: abstraction unifies different kinds of devices
    - Router:
        - Match: longest destination IP prefix
        - Action: forward out a link
    - Firewall:
        - Match: IP addresses and TCP/UDP port numbers
        - Action: permit or deny
    - Switch:
        - Match: destination MAC address
        - Action: forward or flood
    - NAT:
        - Match: IP address and port
        - Action: rewrite address and port
    - 

      OpenFlow example

match	action
IP Src = 10.3.*.* IP Dst = 10.2.*.*	forward(3)

match	action
ingress port = 1 IP Src = 10.3.*.* IP Dst = 10.2.*.*	forward(4)

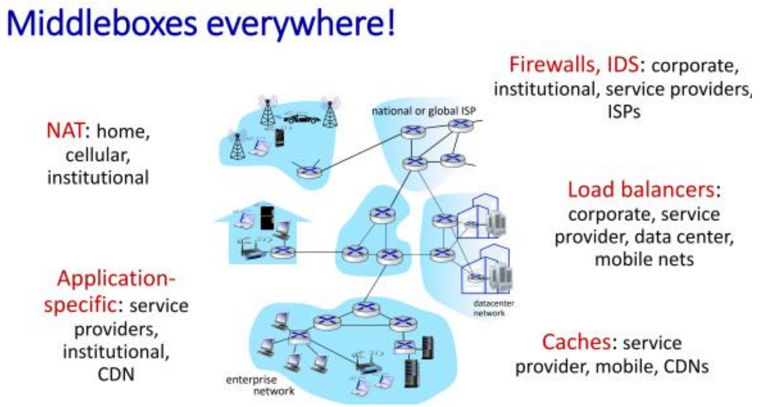
match	action
ingress port = 2 IP Dst = 10.2.0.3	forward(3)
ingress port = 2 IP Dst = 10.2.0.4	forward(4)

      Orchestrated tables can create *network-wide* behavior

      - Example: datagrams from hosts h5 and h6 should be sent to h3 or h4, via s1 and from there to s2
- Summary:
    - Match plus action abstraction: match bits in arriving packet headers in any layers, take

    action
        - Matching over many fields (link, network, transport)
        - Local actions: drop, forward, modify, or send matched packet to controller
        - Program network-wide behaviors
    - Simple form of network programmability
        - Programmable, per-packet processing
        - Historical roots: active networking

Middle boxes



Middleboxes everywhere!

NAT: home, cellular, institutional

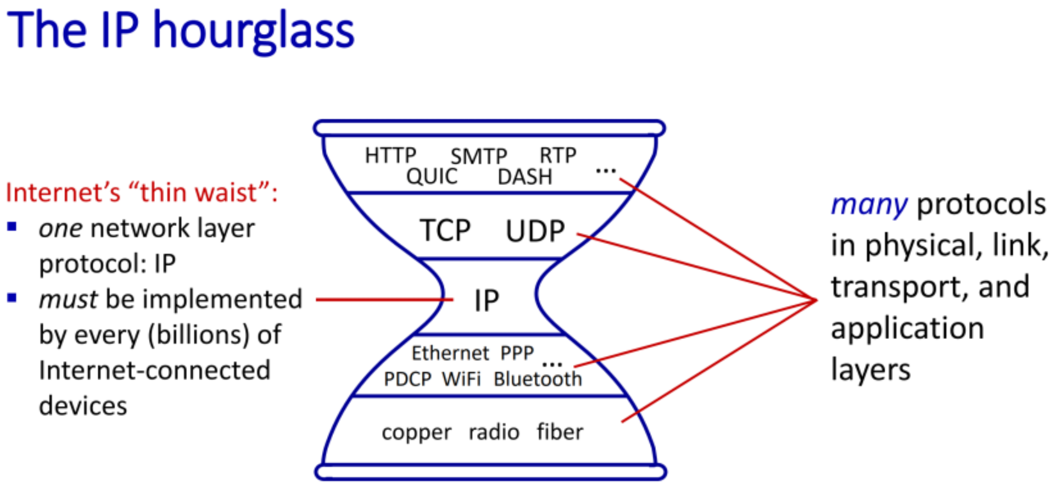
Application-specific: service providers, institutional, CDN

Firewalls, IDS: corporate, institutional, service providers, ISPs

Load balancers: corporate, service provider, data center, mobile nets

Caches: service provider, mobile, CDNs

- Initially: proprietary hardware solutions
- Move towards white box hardware implementing open API
    - Move away from proprietary hardware solutions
    - Programmable local actions via match + action
    - Move towards innovation/differentiation in software
- SDN: centralized control and configuration management often in private/public cloud
- Network functions virtualization (NFV): programmable services over white box networking, computation, storage



The IP hourglass

Internet's "thin waist":

- *one* network layer protocol: IP
- *must* be implemented by every (billions) of Internet-connected devices

many protocols in physical, link, transport, and application layers

- Middle boxes added to waist (NAT, caching, NFV, Firewalls)

Middle boxes: NAT, HTTP cache, HTTP load balancer

- NAT: home, cellular, institutional
- Application specific: service providers, institutional, CDN
- Firewalls, IDS: corporate, institutional, service providers, ISPs
- Load balancers

Router is not a middle box

Architectural principles of the Internet

- Simple connectivity
- IP protocol

- Intelligence, complexity at network edge

End-end argument

- Some network functionality (reliable data transfer, congestion) can be implemented in network, or at network edge
- The end-to-end argument advocates placing functionality at the network edge because some functionality cannot be completely and correctly implemented in the network, and so needs to be placed at the edge in any case, making in-network implementation redundant
- The end-to-end argument allows that some redundant functionality might be placed both in-network and at the network edge in order to enhance performance

Chapter 5

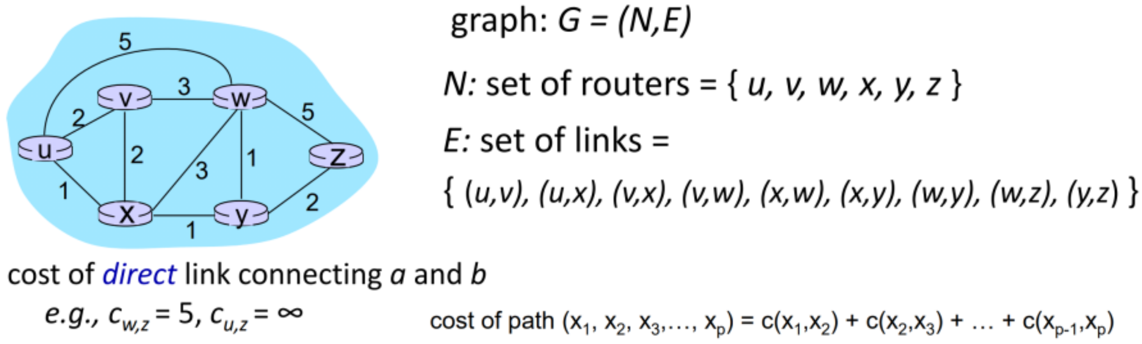
November 5, 2020 10:37 AM

Control plane: routing

Routing protocols:

- Goal: determine good paths/routes, from sending hosts to receiving host, through network of routers
  - Path: sequence of routers packets to traverse from given initial source host to final destination host
  - Least cost, fastest, least congested
  - Very challenging

Graph abstraction: link costs

- 

  graph: $G = (N,E)$

  N: set of routers = { *u, v, w, x, y, z* }

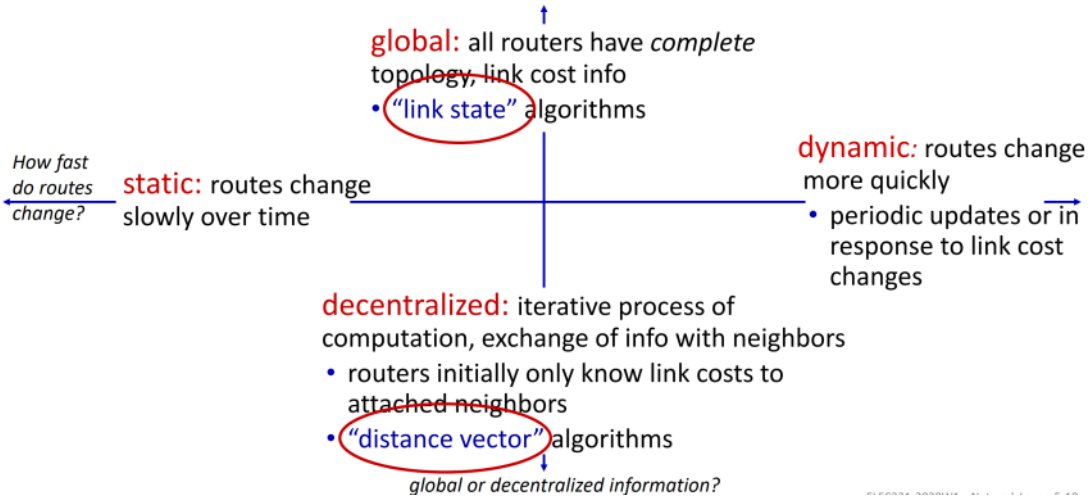
  E: set of links = { (u,v), (u,x), (v,x), (v,w), (x,w), (x,y), (w,y), (w,z), (y,z) }

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

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

  cost of path $(x_1, x_2, x_3,\dots, x_p) = c(x_1,x_2) + c(x_2,x_3) + \dots + c(x_{p-1},x_p)$

Routing algorithm classification

- 

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

  How fast do routes change?

  static: routes change slowly over time

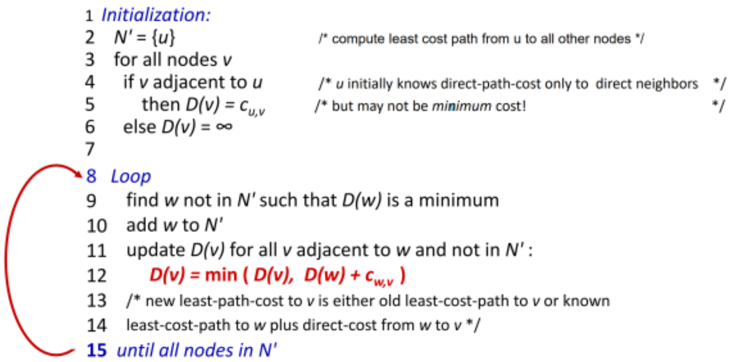
  dynamic*:* routes change more quickly
  - periodic updates or in response to link cost changes

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

  global or decentralized information?

Dijkstra's Link state (shortest path) algorithms:

- Centralized: network topology, link costs known to all nodes
  - Accomplished by link state broadcast
  - All nodes have same info
- Computes least cost paths from one node to all other nodes
  - Gives forwarding table for that node
- Iterative: after k iterations, know least cost path to k destinations
- Algorithms
  - Notations
    - $C_{x,y}$: direct link cost from node x to y, ∞ if not direct neighbors
    - $D(v)$: current estimate of cost of least cost path from source to destination
    - $p(v)$: predecessor node along path from source to v
    - N': set of nodes whose least cost path definitively known

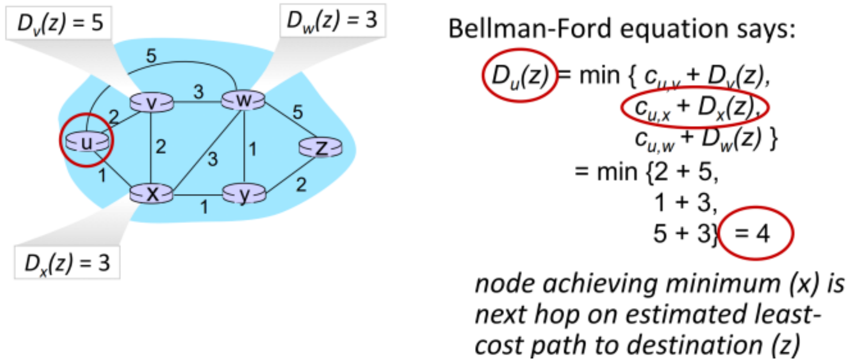
- 

```
Initialization:
  N' = {u}                          /* compute least cost path from u to all other nodes */
  for all nodes v
    if v adjacent to u              /* u initially knows direct-path-cost only to  direct neighbors   */
      then D(v) = $c_{u,v}$          /* but may not be minimum cost!                                   */
    else D(v) = ∞

Loop
  find w not in N' such that D(w) is a minimum
  add w to N'
  update D(v) for all v adjacent to w and not in N' :
     D(v) = min ( D(v),  D(w) + $c_{w,v}$ )
  /* new least-path-cost to v is either old least-cost-path to v or known
  least-cost-path to w plus direct-cost from w to v */
until all nodes in N'
```

- Complexity: n nodes
  - $\frac{n(n+1)}{2}$ comparisons: O(n^2) complexity
  - More efficient implementation possible: O($nlogn$)
- Message complexity
  - Each router must broadcast its link state information to other n routers
  - Efficient broadcast algorithms: O(n) link crossings to disseminate a broadcast message from one source
  - Each router's message crosses O(n) links, overall message complexity O(n^2)
- When link costs depend on traffic volume, route oscillations possible

Distance vector algorithm

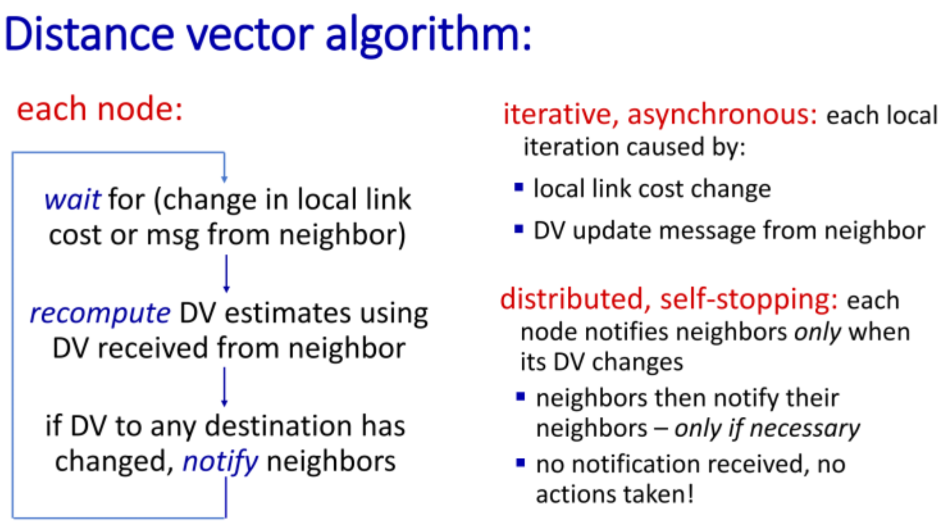
- Based on Bellman-Ford(BF) equation (dynamic programming):
  - Only consider the neighbor, immediate cost
  - $D_x(y) = \min_v\{c_{x,y} + D_v(y)\}$ is the cost of least cost path from x to y
    - $D_v(y)$ is v's estimated least cost path cost to y
    - $c_{x,y}$ is direct cost of link from x to v
  - 

Bellman-Ford equation says:

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

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

- Key idea:
  - From time to time, each node sends its own distance vector estimate to neighbors
  - When x receives new estimate from any neighbor, it updates its own distance vector using B-F equation
  - Under minor, natural conditions, the estimate $D_x(y)$ converge to the actual least cost $d_x(y)$

- 

Distance vector algorithm:

each node:

wait for (change in local link cost or msg from neighbor)

recompute DV estimates using DV received from neighbor

if DV to any destination has changed, *notify* neighbors

iterative, asynchronous: each local iteration caused by:

- local link cost change
- DV update message from neighbor

distributed, self-stopping: each node notifies neighbors *only* when its DV changes

- neighbors then notify their neighbors – *only if necessary*
- no notification received, no actions taken!

- Link cost changes:
    - Node detects local link cost change
    - Updates routing info, recalculates local distance vector
    - If the distance vector changes, notify neighbors
    - Good news travels fast
    - Bad news travels slow (count to infinity problem)

Comparison of link-state and distance-vector algorithms

- Message complexity
    - LS: n router, O(n^2) messages
    - DV: convergence time varies (exchange between neighbors)
- Speed of convergence
    - LS: O(n^2), may have oscillations
    - DV: time varies, may have routing loops and count-to-infinity problem
- Robustness
    - LS:
        - router can advertise incorrect link cost
        - Each router computes its own table
    - DV:
        - Can advertise incorrect path cost: black holing
        - Each router's table used by others, error propagates through the network

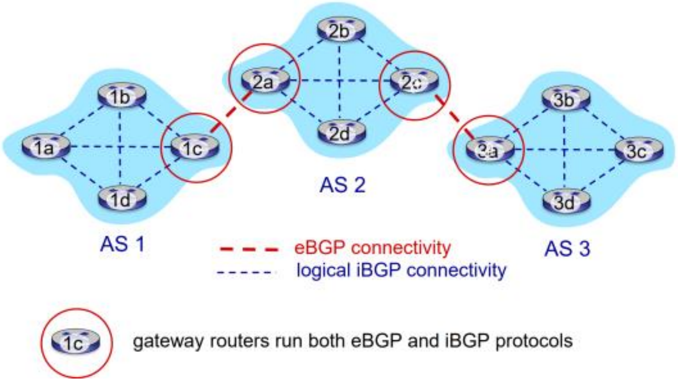
Problem with idealized routing:

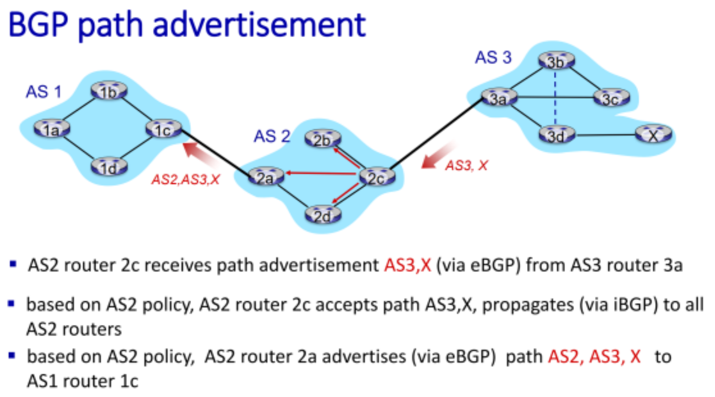
- Scalability
    - Can't store all destinations in routing table
    - Routing table exchange would swamp links
- Administrative autonomy
    - Each network admin may want to control routing in its own network

Autonomous systems (AS, domain)

- To achieve scalable routing, aggregate routers into regions know as domain
- Intra-AS (intra-domain): routing within same AS
    - All routers in AS run same intra-domain protocol
    - Routers in different AS scan run different intra-domain protocols
    - Gateway router: at edge of its own AS, has link to routers in other AS
    - Protocols
        - RIP (routing information protocol)
            - □ Classic DV: DVs exchanged every 30 second
            - □ No longer widely used
        - EIGRP (enhanced interior gateway routing protocol)
            - □ DV based
            - □ Formerly Cisco-proprietary for decades

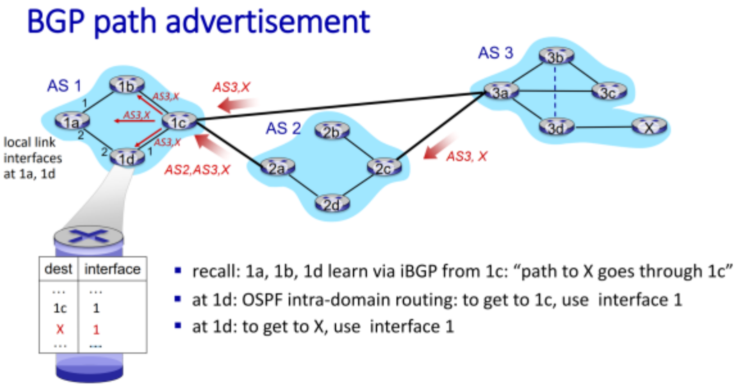
        - OSPF (open shortest path first)
            - Link-state routing
            - IS-IS protocol (ISO standard) essentially same as OSPF
            - Open: publicly available
            - Classic link-state
                - Each router floods OSPF link-state advertisements to all other routers in entire AS directly over IP (no use of TCP or UDP)
                - Multiple link costs metrics: bandwidth, delay
                - Each router has full topology, uses Dijkstra's algorithm to compute forwarding table
            - Security: all OSPF messages authenticated
            - Hierarchical OSPF in large domains
                - Two level hierarchy: local area, backbone
                    - Link-state advertisements flooded only in area, or backbone
                    - Each node has detailed area topology, only knows direction to reach other destinations

- Inter-AS (inter-domain): routing among AS
    - Gateways perform inter-domain routing
    - Forwarding table configured by both the intra and inter-AS routing algorithms
        - Intra-AS routing determine entries for destinations within AS
        - Inter-AS and intra-AS determine entries for external destinations
    - BGP (Border Gateway protocol)
        - Allows subnet to advertise its existence and the destinations it can reach to rest of the Internet
        - eBGP (external): obtain subnet reachability information from neighboring Ases (usually edge routers)
        - iBGP (internal): propagate reachability information to all AS-internal routers (usually internal routers)
            - 
2b
2a
2c
1b
1a
1c
1d
2d
3b
3a
3c
3d
AS 1
AS 2
AS 3
eBGP connectivity
logical iBGP connectivity
1c
gateway routers run both eBGP and iBGP protocols
        - BGP session: two BGP routers exchange BGP messages over semi-permanent TCP connection
        - BGP advertised route: prefix + attributes
            - Prefix: destination being advertised
            - Two important attributes:
                - AS-PATH: list of AS 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
                - AS policy also determines whether to advertise path to other neighboring AS

□ 

BGP path advertisement

- 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 path advertisement

dest	interface
…	…
1c	1
X	1
…	…

- recall: 1a, 1b, 1d learn via iBGP from 1c: “path to X goes through 1c”
- at 1d: OSPF intra-domain routing: to get to 1c, use interface 1
- at 1d: to get to X, use interface 1

□ BGP messages
- OPEN: opens TCP connection to remote BGP peer
- UPDATE: advertises new path
- KEEPALIVE: keeps connection alive in absence of UPDATES, ACKs OPEN request
- NOTIFICATION: reports errors in previous message, close connection

Comparing intra-, inter-AS routing:

- Policy
  - Inter-AS: admin wants control over how its traffic routed, who routes through its network
  - Intra-AS: single admin, so policy less of an issue
- Scale
  - Hierarchical routing saves table size, reduced update traffic
- Performance:
  - Intra-AS: can focus on performance
  - Inter-AS: policy dominates over performance

Hot potato routing: choose local gateway that has least intra-domain cost, don't worry about inter-domain cost

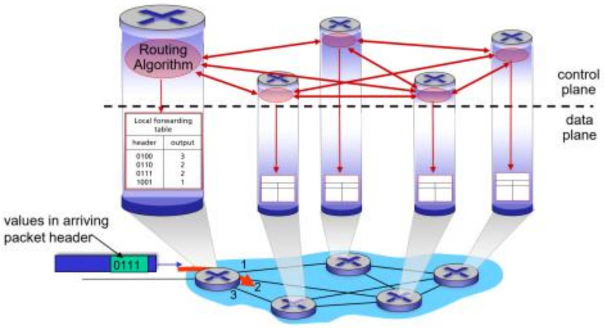
BGP route selection: router may learn about more than one route to destination AS, selects route based on

- Local preference value attribute: policy decision
- Shortest AS-PATH
- Closest NEXT-HOP router: hot potato routing
- Additional criteria

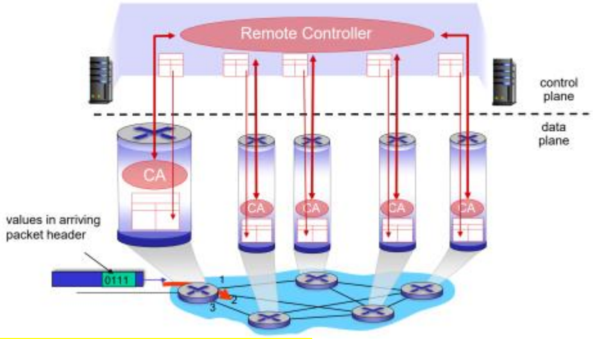
Software defined networking(SDN):

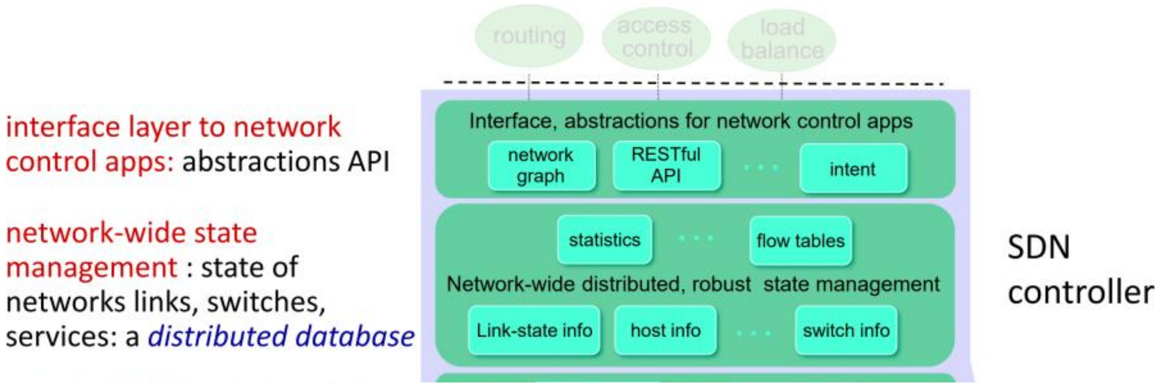
- Per-router control plane: individual routing algorithm components in each and every router interact in the control plane to computer forwarding tables
  - Traffic engineering is difficult
    - Need to re-define link weights to update the routing
    - Cannot do load balancing

        - Cannot have different routes with the same destination.

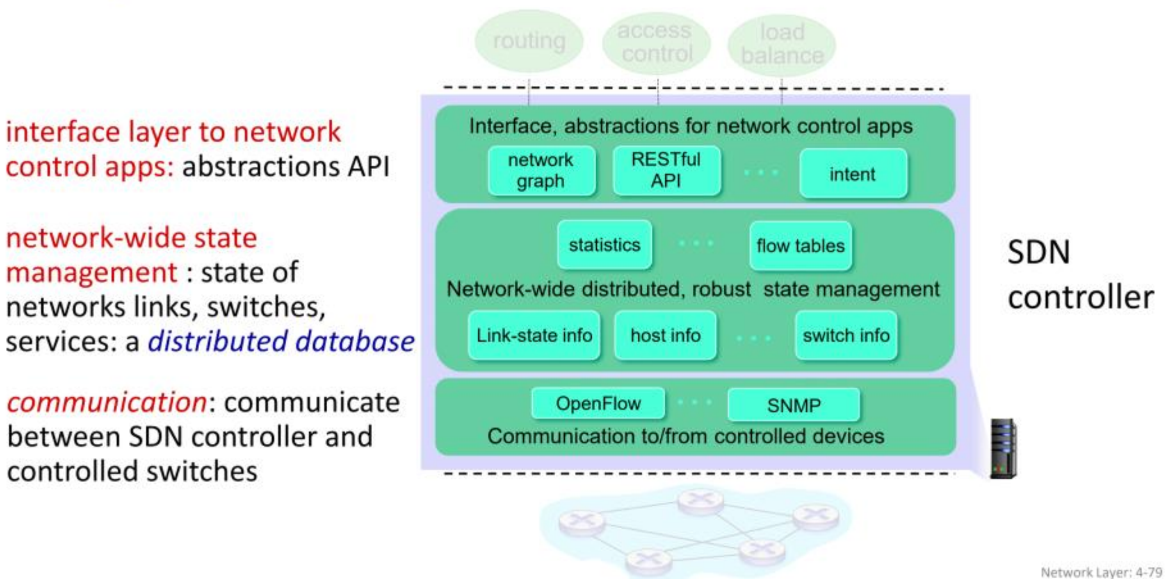


- SDN: Remote controller computes, installs forwarding tables in routers
    - Generalized flow-based forwarding
    - Control, data plane separation
    - Control plane functions (loading balance, routing, access control, etc.) external to data-plane switches
    - Programmable control applications (routing)



- Logically centralized control plane:
    - Easier network management
    - Table-based forwarding allows programming routers
        - Centralized programming is easier (compute tables centrally and distribute)
        - Distributed programming is more difficult (compute tables as result of distributed algorithm implemented in each and every router)
    - Open implementation of control plane
- Data-plane switches
    - Fast, simple, commodity switches implementing generalized data-plane forwarding in hardware
    - Flow table computed, installed under controller supervision
    - API for table-based switch control
    - Protocol for communicating with controller
- SDN controller (network OS):
    - Maintain network state information
    - Interacts with network control application above via northbound API
    - Interacts with network switches below via southbound API
    - Implemented as distributed system for performance, scalability, fault-tolerance, robustness
    - 

routing
access control
load balance
interface layer to network control apps: abstractions API
Interface, abstractions for network control apps
network graph
RESTful API
. . .
intent
network-wide state management : state of networks links, switches, services: a distributed database
statistics
. . .
flow tables
Network-wide distributed, robust state management
Link-state info
host info
. . .
switch info
SDN controller

    - 

interface layer to network control apps: abstractions API

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

communication: communicate between SDN controller and controlled switches

- Network-control apps
    - Brains of control: implement control functions using lower-level services, API provided by SDN controller
    - Unbundled: can be provided by 3rd party, distinct from vendor or SDN controller
- Open Flow protocol
    - Operates between controller, switch
    - Use TCP to exchange messages
    - Three classes of Open Flow messages
        - Controller to switch
            - Features: controller queries switch, switch replies
            - Configure: controller queries/sets switch configuration parameters
            - Modify-state: add, delete, modify flow entries in the open flow table
            - Packet-out: controller can send the packet our of specific switch port
        - Asynchronous (switch to controller)
            - Packet-in: transfer packet to controller
            - Flow-removed: flow table entry deleted at switch
            - Port status: inform controller of a change on a port
        - Symmetric
        - Note: network operators don't program switches by creating/sending open flow messages directly. Instead use higher-level abstraction at controller
    - Distinct from Open Flow API
        - API used to specify generalized forwarding actions
- example controllers
    - Open Daylight (ODL)
        - Service abstraction layer: interconnects internal, external applications and services
    - ONOS
        - Control apps separate from controller
        - Intent framework: high-level specification of service
        - Considerable emphasis on distributed core: service reliability, replication performance scaling
- SDN challenges
    - Hardening the control plane: dependable, reliable, performance-scalable, secure distributed system
    - Networks, protocols meeting mission-specific requirements
    - Internet-scaling: beyond a single AS

ICMP: internet control message protocol

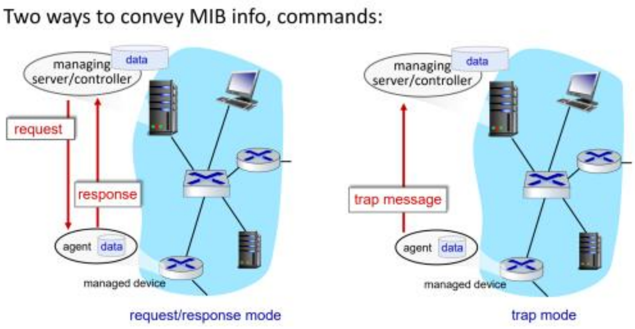
- Mostly used for error reporting or echo request/reply
- ICMP messages are carried in IP datagrams
- ICMP message: type, code plus first 8 bytes of IP datagram causing error

- Traceroute and ICMP
    - Source sends sets of UDP segments to destination
        - 1st set has TTL=1, 2nd has TTL=2…
    - Datagram in nth set arrives to nth router
        - Router discards datagram and sends source ICMP message (type 11, code 0 TTL expired)
        - ICMP message possibly includes name of router and IP address
    - When ICMP arrives at source, record RTTs
    - Stopping criteria:
        - UDP segment arrives at destination host
        - Destination returns ICMP message port unreachable (type3, code 3)
        - Source stops

Network management

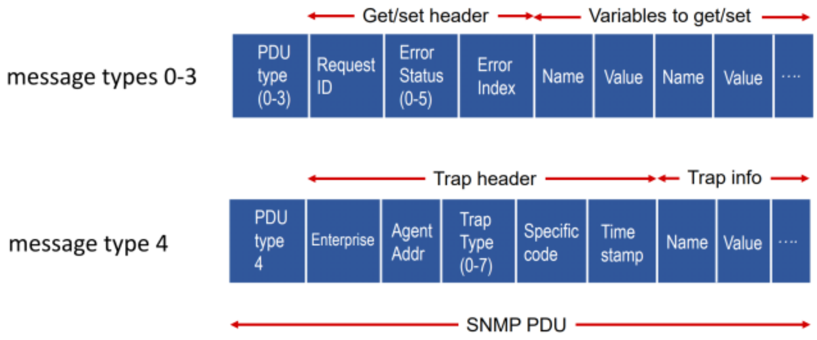
- Components
    - Managing server: application, typically with the network managers (human) in the loop
    - Network management protocol: used by managing server to query, configure, manage device. Used by devices to inform managing server of data, events
    - Managed device: equipment with manageable, configurable hardware, software component
    - Data: device state, configuration data, operational data, device statistics
- Approaches:
    - CLI: operator issues direct to individual devices
    - SNMP/MIB: operator queries/sets devices data (MIB) using Simple Network Management Protocol (SNMP)
    - NETCONF/YANG:
        - More abstract, network-wide, holistic
        - Emphasis on multi-device configuration management
        - YANG: data modeling language
        - NETCONF: communicate YANG-compatible actions/data to/from/among remote devices

SNMP protocol

- 

Two ways to convey MIB info, commands:

- Message types
    - Get: manager-to-agent
    - Set: manager-to-agent, set MIB value
    - Response: agent-to-manager, value, response to request
    - Trap: agent-to-manager, inform manager of exceptional event

- 

- Management Information Base(MIB)
  - Managed device's operational (some configuration) data
  - Gathered into device MIB module
  - Structure of Management Information (SMI): data definition language
  - Usually using UDP

NETCONF
- Goal: actively manage/configure devices network-wide
- Operates between managing server and managed network devices
  - Actions: retrieve, set, modify, activate configurations
  - Atomic-commit actions over multiple devices
  - Query operational data and statistics
  - Subscribe to notifications from devices
- Remote procedure call (RPC) paradigm
  - NETCONF protocol messages encoded in XML
  - Exchanged over secure, reliable transport protocol
- YANG
  - Data modeling language used to specify structure, syntax, semantics of NETCONF network management data
    - Built-in data types similar to SMI
  - XML document describing device, capabilities can be generated from YANG description
  - Can express constraints among data that must be satisfied by a valid NETCONF configuration
  - Ensure NETCONF configurations satisfy correctness, consistency constraints

TCPs used in this chapter:
- BGP
- YANG
- OPEN FLOW

UDPs used:
- MIB
- ICMP

Chapter 6

November 5, 2020 10:37 AM

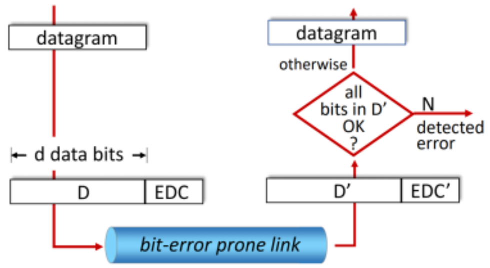
Link layer services

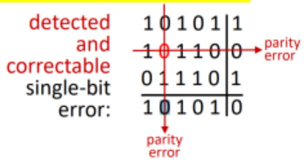
- Basic: move a datagram from one node (host/router) to an adjacent node over a single communication link
- Framing, link access
- Reliable delivery between adjacent nodes (usually on wireless links)
  - TCP is end-to-end, certain link has high error rates, we want to know the error as soon as possible
- Error detection
- Error correction
- Flow control
- Half-duplex and full-duplex
  - Half-duplex: nodes at both ends of link can transmit but not at same time
  - Links are full duplex A-to-C and C-to-A can happen at the same time.

Implemented in:

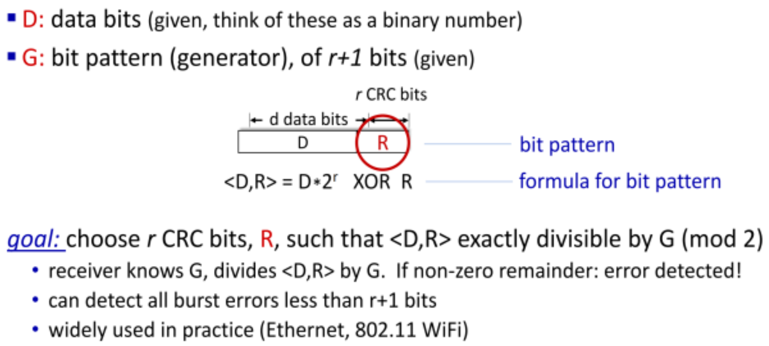
- Each and every host
- Network interface card (NIC) or on a chip
  - Ethernet, WiFi card or chip
- Attaches into host's system buses
- Combination of hardware, software, firmware

Error detection:

- Forward error detection (FEC): sender encodes the data using an error-correcting code (ECC) prior to transmission. Additional information (redundancy) added by the code is used by the receiver to detect errors and possibly recover the original data
  - 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 error
    - Larger EDC field yields better detection and correction
- Parity check
  - Single bit parity: detect single bit errors
    - Even parity: set parity bit so there is an even number of 1s
  - Two-dimensional bit parity: detect and correct single bit errors
    - 

- Internet checksum
- Cyclic redundancy check (CRC)
  - More powerful error-detection coding

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

    <D,R> = D∗2^r XOR R

    goal: choose *r* CRC bits, R, such that <D,R> exactly divisible by G (mod 2)
    - receiver knows G, divides <D,R> by G. If non-zero remainder: error detected!
    - can detect all burst errors less than r+1 bits
    - widely used in practice (Ethernet, 802.11 WiFi)
  - We want $R = remainder\left(\frac{D\cdot 2^r}{G}\right)$

Multiple access links, protocols

- Two types of links
  - Point-to-point
    - Point-to-point link between Ethernet switch, host
    - PPP for dial-up access
  - Broadcast (shared wire or medium)
    - Cabled Ethernet
    - Upstream HFC in cable-based access network
    - 802.11 wireless LAN, 4G/satellite
- Protocols:
  - Single shared broadcast channel
  - Two or more simultaneous transmissions by nodes:
    - Collision if node receives two or more signals at the same time
  - Ideal protocol for multiple access channel (MAC) of rate R bps
    - When one node wants to transmit, it can send at rate R
    - When M nodes want to transmit, each can send at average rate R/M
    - Fully decentralized
      - No special node to coordinate transmissions
      - No synchronization of clocks, slots
    - Simple
  - Three broad classes
    - Channel partitioning
      - TDMA (time division multiple access)
      - FDMA (frequency division multiple access)
    - Take turns
      - Similar to TDMA, but one can have longer time interval than others
    - Random access (common)
      - When node has packet to send, transmit at full channel data rate R, no coordination among nodes
      - Two or more transmitting nodes: collision
      - Random access MAC protocol specifies
        - How to detect collisions
        - How to recover from collisions
      - Examples: ALOHA, slotted ALOHA, CSMA, CSMA/CD (Ethernet), CSMA/CA(WiFi)

Slotted ALOHA

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

  - When node obtains fresh frame, transmits in next slot
  - If collision: node transmits frame in each subsequent slot with probability p until success
    - The probability gives randomization
- Pros
  - Simple
  - Highly decentralized: only slots in nodes need to be in sync
  - Single active node can continuously transmit at full rate of channel
- Cons:
  - Collisions and idle slots waste slots
  - Nodes may be able to detect collision faster than transmitting packet
  - Need synchronization
- Efficiency: long run fraction of successful slots
  - Max efficiency 1/e=0.37
  - $np(1-p)^{n-1}$
  - At best channel used for useful transmission 37% of time

Pure ALOHA

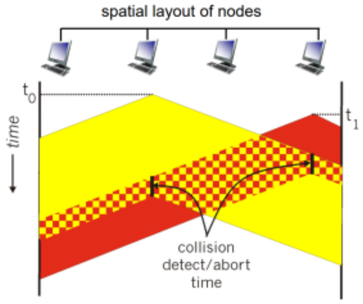
- Unslotted, simpler, no synchronization
  - When frame first arrives, transmit immediately
- Collision probability increases with no synchronization (frame sent at t_0 collides with other frames sent in $[t_0-1, t_0+1]$)
- Max Efficiency: 18%
- $p(1-p)^{2(n-1)}$

CSMA: carrier sense multiple access

- Simple CSMA: listen before transmit
  - If channel sensed idle: transmit entire frame
  - If channel sensed busy: defer transmission
  - Analogy: don't interrupt others
- Collisions: entire packet transmission time wasted
  - Propagation delay: two nodes may not hear each other's just started transmission
  - Distance & propagation delay play role in determining collision probability

CSMA/CD: CSMA with collision detection

- Collisions detected within short time
  - Easy in wired LANs
  - Difficult in wireless LANs: received signal strength overwhelmed by local transmission strength
- Colliding transmissions aborted, reducing channel wastage
- Collision handling
  - when a node performs collision detection, it ceases transmission as soon as it detects a collision



Stop transmission at the bar

- Analogy: polite conversationalist
- Algorithm:
  - NIC receives datagram from network layer, create frame

  - If NIC senses channel:
    - Idle, start frame transmission
    - Busy, wait until idle, then transmit
  - If NIC transmits entire frame without collision, NIC is done with this frame
  - If NIC detects another transmission while sending, abort and send jam signal
  - After aborting, NIC enters binary (exponential) back off:
    - After mth collision, NIC chooses K at random from $\{0,1,2,\ldots,2^m-1\}$, wait for $K \cdot 512 \cdot time\ to\ transmit\ one\ bit$, return to step 2
    - More collisions means longer back off interval
- Efficiency: $\frac{1}{1+\frac{5t_{prop}}{t_{trans}}}$
  - t_{prop}=maximum propagation delay between 2 nodes in LAN
  - t_{trans}=time to transmit max-size frame
  - Efficiency goes to 1 as t_{prop} goes to 0 or t_{trans} goes to infinity
  - Better performance than ALOHA and simple, cheap, decentralized

Channel partitioning MAC protocols:

- Share channel efficiently and fairly at high load
- Inefficient at low load: delay in channel access, $\frac{1}{N}$ bandwidth allocated even if there is only 1 active node.

Random access MAC protocols:

- Efficient at low load: single node can fully utilize channel
- High load: collision overhead

Taking turns protocols:

- Look for best of both high and low loads
- Polling from central site:
  - Master node invites other nodes to transmit in turn
  - Typically used with dumb devices
  - Problems:
    - Polling overhead
    - Latency
    - Single point of failure at master
- Token passing
  - Control token passes from one node to next sequentially
  - Token message
  - Problems:
    - Token overhead
    - Latency
    - Single point of failure at token

Cable access network: FDM + TDM + random access

- Multiple downstream (broadcast) FDM channel
  - Single CMTS (cable modem termination system) transmits into channels
- Multiple upstream channels
  - Multiple access: all users contend (random access) for certain upstream channel time slots, others assigned TDM
- DOCSIS (data over cable service interface specification)
  - FDM over upstream, downstream frequency channels
  - TDM upstream, some slots assigned, some have contention
    - Downstream MAP frame: assigns upstream slots
    - Request for upstream slots transmitted random access in selected slots

MAC (LAN, physical, Ethernet) address:

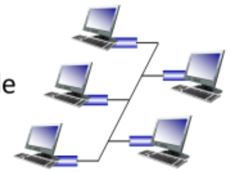
- Used locally to get frame from one interface to another physically-connected interface (same subnet, in IP-addressing sense)

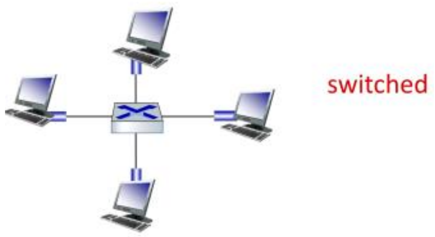
- 48-bit MAC address default in NIC ROM, sometimes software settable
- Each interface on LAN
  - Unique 48-bit MAC address
  - Locally unique 32-bit IP address
- MAC address allocation administered by IEEE
- Analogy:
  - MAC address: social security number
  - IP address: postal address
- MAC flat address: portability
  - Can move interface from one LAN to another
  - IP address not portable

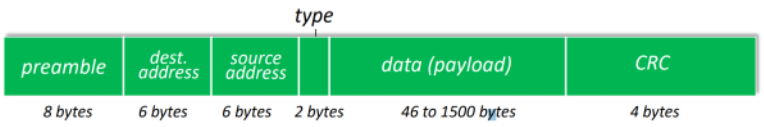
ARP(address resolution protocol)

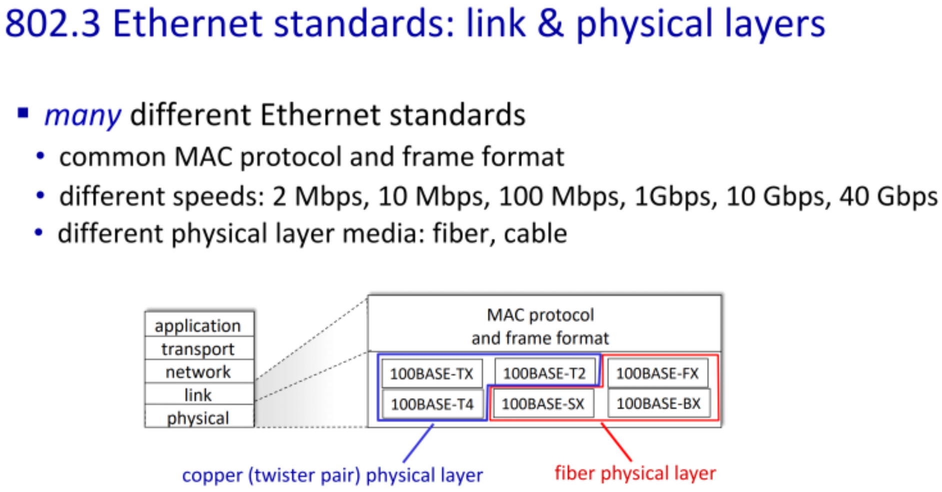
- Determine interface's MAC address, knowing its IP address
- ARP table: each IP node (host, router) on LAN has this table
  - IP/MAC address mappings for some LAN nodes: <IP address; MAC address; TTL>
  - TTL: time after which address mapping will be forgotten (typically 20 min)
- A sends datagram to B in the same subnet, MAC address of B not in A's ARP table
  - A broadcasts ARP query, containing B's IP address
    - Destination MAC address: FF:FF:FF:FF:FF:FF
    - All nodes on LAN receive ARP query
  - B replies to A with ARP response, giving its MAC address
    - Only seen by B and A
  - A receives B's reply, adds B entry into its local ARP table
- Routing to another subnet: A sends a datagram to B via R
  - Assumes that
    - A knows B's IP address
    - A knows IP address of first hop router, R (from DHCP)
    - A knows R's MAC address (they are in the same subnet, just send an ARP broadcast)
  - A creates IP datagram with IP source A, destination B
  - A creates link-layer frame containing A-to-B IP datagram
    - R's MAC address is frame's destination
  - Frame sent to R
  - R receives the frame, datagram removed, passed up to IP
  - R determines outgoing interface, passes datagram with IP source A, destination B to link layer
  - R creates link-layer frame containing A-to-B IP datagram, with destination address being B's MAC address

Ethernet:

- Dominant wired LAN technology
  - First widely used
  - Simpler, cheap
  - Kept up with speed race
  - Single chip, multiple speeds
- Bus: popular in 90s
  - All nodes in same collision domain
  - bus: coaxial cable 
- Switched (star): used now
  - Active link-layer 2 switch in center

  - Each spoke runs a separate protocol (nodes do not collide with each other)
  - 

- Frame structure:
  - Sending interface encapsulates IP datagram in Ethernet frame
  - 

  - Preamble: 7 byte with 10101010 followed by one byte 10101011
    - Used to synchronize receiver, sender clock rates
  - Address: MAC addresses
    - If adapter receives frame with matching destination address, or with broadcast address, it passes data in frame to network layer protocol
    - Otherwise, adapter discards frame
  - Type: indicate upper layer protocol
    - Mostly IP
    - Used to demultiplex up at receiver
  - CRC: cyclic redundancy check at receiver
    - If error detected, the frame is dropped
  - Payload:
    - Ethernet MTU is 1,500 bytes
      - □ If IP datagram exceeds MTU, datagram has to be fragmented
    - Minimum size is 46 bytes
      - □ If less than 46 bytes, the data field has to be appended to fill out to 46 bytes
      - □ Receiver uses the length field in the IP datagram to remove the appending
- Ethernet is unreliable and connectionless
  - Unreliable: NIC doesn't send ACKs or NAKs, data in dropped frames recovered only if initial sender uses higher layer rdt (TCP)
  - No handshaking between sender and receiver
- Ethernet's MAC protocol: unslotted CSMA/CD with binary back off
- 

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

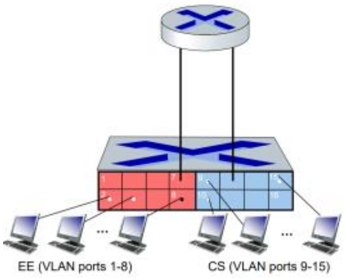
Ethernet switch

- Store, and selectively forward Ethernet frames to one-or-more outgoing links
- Transparent: hosts unaware of presence of switches
- Plug-and-play, self-learning
  - Switches do not need to be configured

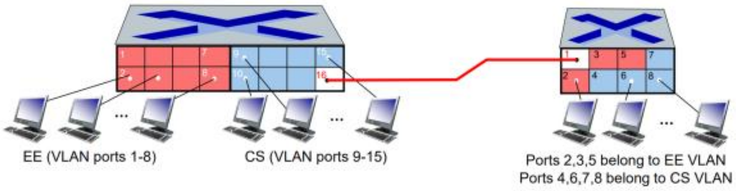
- Multiple simultaneous transmissions
    - Hosts have dedicated, direct connection to switch
    - Switches buffer packets
    - Ethernet protocol used on each incoming link
        - No collisions, full duplex
        - Each link is its own collision domain
    - Switching: A-to-A' and B-to-B' can transmit simultaneously, without collisions
        - But A-to-A' and C-to-A' cannot happen simultaneously
- Each switch has a switch table
    - Entry: MAC address + interface to reach host + time stamp
    - Looks like a routing table
- Switch learns which hosts can be reached through which interface
    - When frame received, switch learns location of sender
    - Records sender/location pair in switch table
    - Frame destination unknown: flooding send
    - Frame destination known: selectively send on just one link
- Switches vs. routers
    - Both are store and forward:
        - Routers: network-layer device, examine network-layer headers
        - Switches: link-layer device, examine link-layer headers
    - Both have forwarding tables
        - Routers: compute tables using routing algorithms, IP addresses
        - Switches: learn forwarding table using flooding, learning, MAC addresses

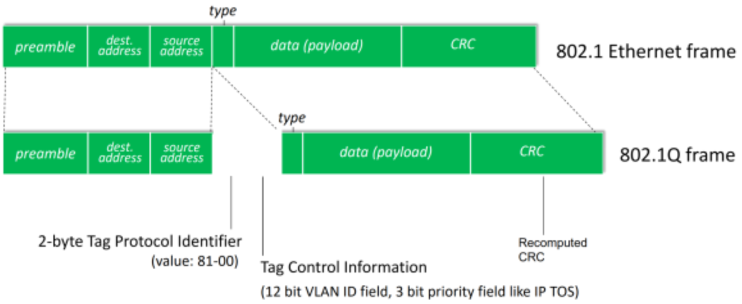
Virtual LAN (VLAN): switches supporting VLAN capabilities can be configured to define multiple virtual LANS over single physical LAN infrastructure

- Motivation
    - To have one physical switch with multiple virtual switches
    - Single broadcast domain:
        - Scaling: all layer-2 broadcast traffic (ARP, DHCP, unknown MAC) must cross entire LAN
        - Efficiency, security, privacy issues
    - Administrative issues
        - Physically attached to one switch but wants to remain logically attached to another switch
- Port-based VLANs
    - Switch ports grouped by switch management software so that single physical switch operates as multiple virtual switches
    - Traffic isolation: frames to/from ports 1-8 can only reach ports 1-8
        - Can also define VLAN based on MAC addresses rather than switch port



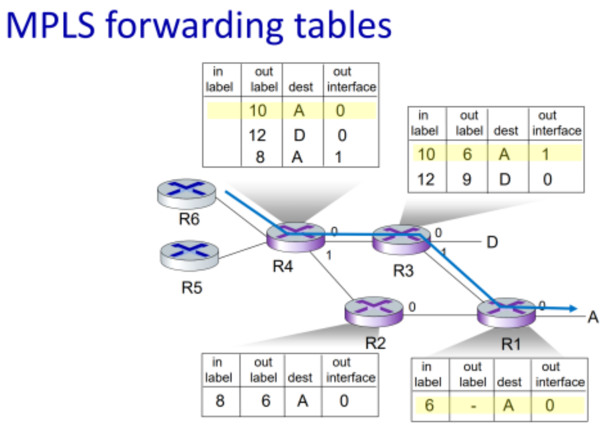
    - Dynamic membership: ports can be dynamically assigned among VLANs
    - Forwarding between VLANs: done via routing as with separate switches
- VLANS spanning multiple switches

    - 
EE (VLAN ports 1-8)
CS (VLAN ports 9-15)
Ports 2,3,5 belong to EE VLAN
Ports 4,6,7,8 belong to CS VLAN
    - Trunk port: carries frame between VLANS defined over multiple physical switches
        - Frames forwarded within VLAN between switches must carry VLAN ID info
        - 802.1q protocol adds/removes additional header fields for frames forwarded between trunk ports



Multiprotocol label switching (MPLS)

- Goal: high-speed IP forwarding among network of MPLS-capable routers, using fixed length label (instead of shortest prefix matching)
    - Faster lookup using fixed length identifier
    - Borrowing ideas from Virtual Circuit (VC) approach
    - IP datagram still keeps IP address
- MPLS capable routers
    - Forward packets to outgoing interface based only on label value (don't inspect IP address)
        - MPLS forwarding table distinct from IP forwarding tables
    - Flexibility: MPLS forwarding decisions can differ from those of IP
        - Use destination and source addresses to route flows to same destination differently (traffic engineering)
        - Re-route flows quickly if link fails: pre-computed backup paths
- MPLS versus IP paths
    - IP routing: path to destination determined by destination address alone
    - MPLS routing: path to destination can be based on source and destination address
        - Generalized forwarding
        - Fast reroute: precompute backup routes in case of link failure
- MPLS signaling
    - Modify OSPF, IS-IS link-state flooding protocols to carry info used by MPLS routing
    - Entry MPLS router uses RSVP-TE signaling protocol to set up MPLS forwarding at downstream routers

- 

MPLS forwarding tables

in label	out label	dest	out interface
	10	A	0
	12	D	0
	8	A	1

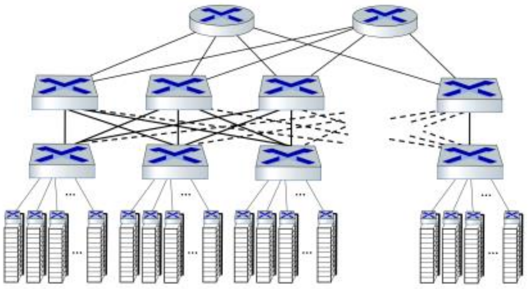
in label	out label	dest	out interface
10	6	A	1
12	9	D	0

in label	out label	dest	out interface
8	6	A	0

in label	out label	dest	out interface
6	-	A	0

Datacenter networks

- 10's to 100's of thousands of hosts, often closely coupled, in close proximity
- Challenges:
  - Multiple applications, each serving massive numbers of clients
  - Reliability
  - Managing/balancing load, avoiding processing, networking, data bottlenecks
- Network elements
  - Border routers: connections outside datacenter
  - Tier-1 switches: connecting to ~16 T-2 switches
  - Tier-2 switches: connecting to ~16 TOR switches
  - Top of Rack (TOR) switches: one per rack, 40-100Gbps Ethernet to blades
  - Server racks: 20-40 server blades as hosts



- Rich interconnection among switches, racks:
  - Increased throughput between racks (multiple routing paths possible)
  - Increased reliability via redundancy
- Load balancer: application-layer routing
  - Receives external client requests
  - Directs workload within data center
  - Returns results to external client (hiding data center internals from clients)
- Protocol innovations
  - Link layer:
    - RoCE: remote DMA (RDMA) over Converged Ethernet
  - Transport layer
    - ECN (explicit congestion notification) used in transport-layer congestion control (DCTCP, DCQCN)
    - Experimentation with hop-by-hop (backpressure) congestion control
  - Routing, management
    - SDN widely used within/among organizations' datacenters
    - Place related services, data as close as possible to minimize tier-2, tier-1 communicatioN