8/27 - Basic Concepts

Friday, August 30, 2024 9:22 AM

Networks
Chapter 1: Introduction
- What is the Internet?
hosts == end systems
packet switches (e.g. routers, switches) forward packets (chunks of data)
packet switches travel between "links" (cables and wireless):
    communication links are fiber, copper, radio, satellite
    all these links have a certain speed, they include a transmission rate: bandwidth (bandwidth == speed)
"networks" are a collection of devices, routers, and links: all managed by an organization
the "internet" is a collection of "networks", interconnected ISPs (e.g. local/region ISP is Spectrum and national or global ISP is Sprint or Verizon)
a "gateway" is a device that transmits your data to the Internet
a "protocol" is a standardized way of communicating (example: like communicating between English and German)
protocols control sending and receiving of messages
examples of protocols include HTTP (Web), streaming video, Skype, TCP, IP, WiFi, 4G, ethernet
all protocols needs to be standardized, so we have Internet standards
two Internet standards we have for protocols are RFC (Request for Comments) and IETF (Internet Engineering Task Force)
every protocol for the Internet has an RFC number
you can look at the Internet as a 'service': an 'infrastructure' that provides services to applications such as web, streaming video, multimedia conferencing, email, games, e-commerce, social media, inter-connected appliances, etc.
you can look at the Internet as a 'service': the Internet provides a 'programming interface' to distribute applications
protocols must be received in a specific order ("two-way handshake"; example: "hello" -> "hello" -> "what time is it?" -> "2:00")

8/29 - Types of Networks

Friday, August 30, 2024 9:22 AM

6/29
A network "edge" is a device at the end of a network
    Hosts include clients and servers (both clients and servers are network edges), servers are often in data centers (often in the cloud)
"Access networks" and physical media are done via wired (ethernet) or wireless communication links
"Network core": interconnected routers and network of networks (e.g. Starlink)
We measure access network with transmission rate (in bits pers second) of access network
An "enterprise access network" is an edge router that connects everyone in a school or company to the Internet
Shared media/link vs. dedicated media/link
Access network: DSL (it's dying), it's a dedicated link to your home and you pay for whatever Internet you use -> it uses a multiplexer to determine if your data goes toa phone network or Internet network; DSL is a dedicated line that only you have

download speed is much faster than upload speed (usually 10x) for the Internet because people don't upload as often

Access network: cable-based - DSL got replaced with cable-based access, this was faster than DSL, old TVs are cable-based where you had a cable box and a bunch of channels (now it's all apps), cable-based TV is dying, but cable-based Internet is still around; multiplexer (called CMTS) determines if data should go to the Internet; fiber optics connect cable headend to neighborhood junctions (500-5000 homes) vs. coaxial cables that connect neighborhood junctions to individual homes; cable-based is a shared access network

frequency division multiplexing (FDM): virtual coaxial cable splitting to share the cable-based access network

access network: fiber to the home (FTTH) - optical fiber from CO directly to home, fiber leaving CO is shared by many homes (it's not dedicated, until it reaches the home), each home has an ONT (optical network terminator) with functionality similar to a DSL/cable model, optical line terminator (OLT) at CO provides conversation between optical and electrical signals, fiber optics happens with light and it's flexible

access network: 5G fixed wireless - common with mobile smartphones but this is for your house, your modem will communicate with the cell tower (your Internet provider would be Verizon, T-Mobile, etc.), WiFi wireless router connects to your modem

you used to have to buy an access point, a router/switch, and a modem separately; now you just buy a wireless router

Internet connected devices can all be wireless connected to the access point and all talk to your mobile phone which is the basis for the Internet of Things (IoT)

Wireless access networks:
WLANs (wireless local area networks) - WiFi or 802.11 - helps devices talk to each other, it's a network protocol; 802.11b/g/n had 11, 54, then 450 Mbps transmission rate respectively, now 802.11ac/ax it supposed to go 3.5Gbps/14Gbps but in practice only goes to 800 Mbps

Wide-area cellular access networks - 4G/5G cellular phone networks

access network: enterprise networks - a switch has a bunch of jacks you connect to with ethernet and you connect one of those cables to the router; ethernet has wired access at 100Mbps, 1Gbps, and 10Gbps; WiFi has wireless access points at 11, 54, 450 Mbps

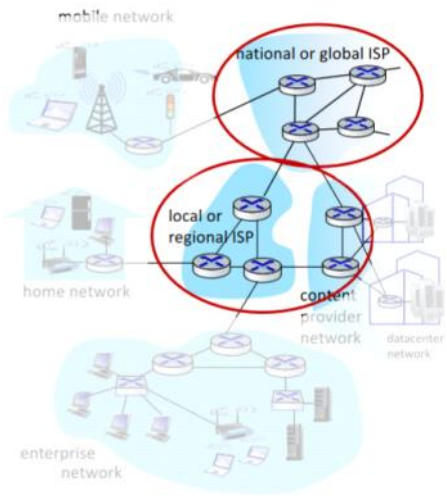
host sends packets of data - take application message; break message into smaller chunks, known as packets, of length L bits; transmits packet into access network at transmission rate R, link transmission rate, AKA link capacity, AKA link bandwidth
packet transmission delay = time needed to transmit L-bit packet into link = L (bits) / R (bits/sec)

links: physical media:
a bit is a signal (1V or 5V is a 1-bit and no electrical signal is a 0-bit)
physical link - transmitter and receiver
guided media - solid media such as copper, fiber, or coaxial cable
unguided media - radio signals
twisted pair (TP) - two insulated copper wires (cat5 -> cat6 ethernet cables, RJ45)
coaxial cable - two concentric copper conductors, bidirectional
fiber optic cables - glass fiber carrying light pulses, each pulse a bit, very fast, low error rate (repeaters spaced far apart and immune to electromagnetic noise)
wireless radio - radio link types: terrestrial microwave, wireless LAN or WLAN (WiFi), wide-area (e.g. cellular), satellite including geostationary and low-earth-orbit (e.g. starlink)

9/3 - Packet Transmission & Delays

Tuesday, September 3, 2024 4:52 PM

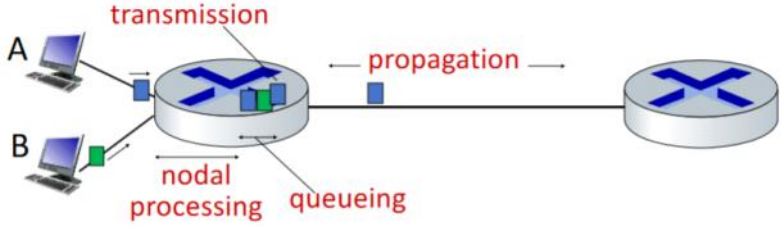
-This is the core of the Internet:



-Packet switching - hosts break application-layer messages into packets
-Routers on the Internet today do "store and forward" - entire packets must arrive at router before it can be transmitted on the next link - there's a transmission delay that takes L/R seconds to transmit (where L bits per packet into link at R bps)

-Packets are queued at the link's output buffer, waiting to be transmitted on an output link
-Packets can be dropped/lost if the memory buffer in the router fills up
    -The speed of the output link is slower than the arrival
-Packet being transmitted (transmission delay) and packets in buffers (queueing delay)
-Nodal processing
-There's also a delay called propagation delay
-Packet delays:

Packet delay: four sources

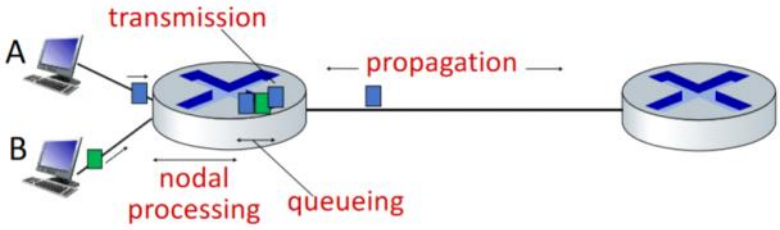


$$d_{nodal} = d_{proc} + d_{queue} + d_{trans} + d_{prop}$$

d_{proc}: nodal processing
- check bit errors
- determine output link
- typically < msec

d_{queue}: queueing delay
- time waiting at output link for transmission
- depends on congestion level of router

Packet delay: four sources



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

d_{trans}: transmission delay:

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

d_{prop}: propagation delay:

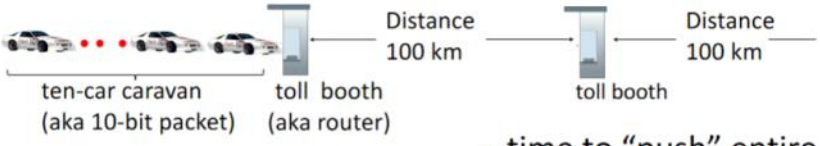
- d: length of physical link
- s: propagation speed (~2×10^8 m/sec)
- $d_{prop} = d/s$

d_{trans} and d_{prop} *very* different

* Check out the online interactive exercises: http://gaia.cs.umass.edu/kurose_ross

50

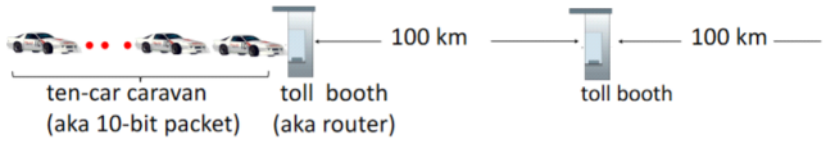
Caravan analogy



- cars "propagate" at 100 km/hr
- toll booth takes 12 sec to service car (bit transmission time)
  num of cars per caravan (L)= 10
  transmission rate (R): 5 cars/min
- car ~ bit; caravan ~ packet
- *Q:* How long until caravan is lined up before 2nd toll booth?
- time to "push" entire caravan through toll booth onto highway = 10 cars / (5 cars/min) = 2 mins (**transmission delay**)
- time for last car to propagate from 1st to 2nd toll booth: 100km/(100km/hr) = 1 hr (**propagation delay**)
- *A:* 62 minutes (2mins + 1hr)

51

Caravan analogy



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

A: Yes! after **7 mins** *first* car arrives at second booth (**1min** to be pushed/ transmitted off the first toll booth **+ 6mins** to be propagated to second booths)

Next 6 cars are still in transit; and *last 3 cars* still at first booth

52

-Dijkstra's Shortest Path algorithm is run as a routing algorithm on routers, a global action

-Instead of packet switching, we used to do circuit switching (commonly used in traditional telephone networks, a connection/call = circuit, an operator would switch you)
    -Packet switching is more optimal than circuit switching

-Frequency Division Multiplexing (FDM) - an implementation of circuit switching, you have frequency bands similar to radio stations (e.g. 88MHz to 108MHz where each station is somewhere between)
-Time Division Multiplexing (TDM) = time divided into frames (n slots / frame)

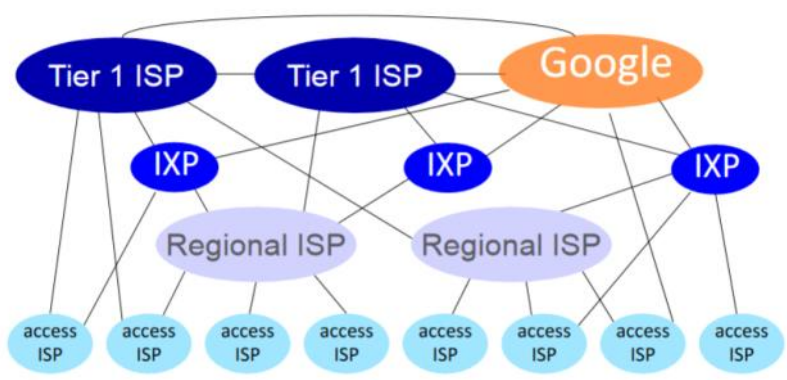
-Packet switching versus circuit switching example:
    -1 Gb/s link
    -Each user has:
        -100 Mb/s when "active"
        -Active 10% of the time (you're not using your link 90% of the time)
    -Answer: with circuit switching, 10 users can use the link (1Gbps / 100Mbs)
    -Answer: with packet switching, it's 0.04% (a lot less, use Central Limit Theorem)

-"Bursty" data is a very common pattern for data being sent, this is why you experienced buffering when loading something (e.g. YouTube videos)

-Video and audio require circuit switching speed

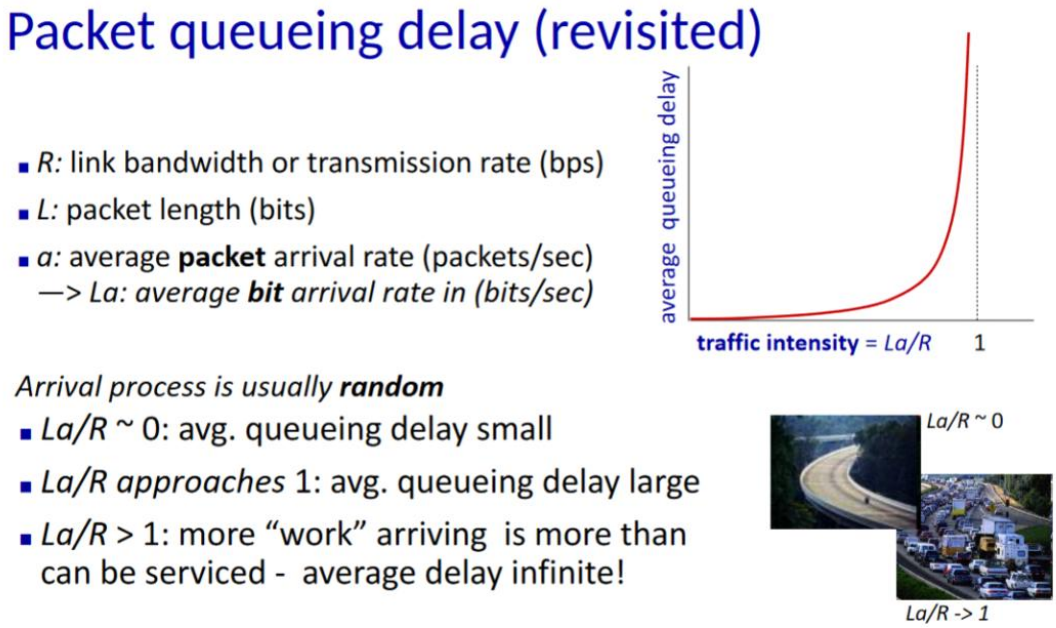
-Given millions of access Internet Service Providers (ISPs), the best way to connect all of them is having several larger Internet Service Providers (ISPs), benefits include preventing a single-point of failure (having several ISPs) and larger ISPs being closer to their own Internet
    -Internet Exchange Point (IXP) is used to connect Internet Service Providers (ISPs) along with peering links
    -Regional ISPs (e.g. Verizon)
    -Content provider networks - what Google and Facebook do, they have their own Internet because they have their own fiber cable network, bypasses tier 1 ISPs
    -Tier 1 ISP - commercial ISPs, national & international coverage (e.g. Sprint, AT&T)
    -Tier 2 ISP - regional ISPs
    -Tier 3 ISP

Internet structure: a "network of networks"



-Point of presence (POP)

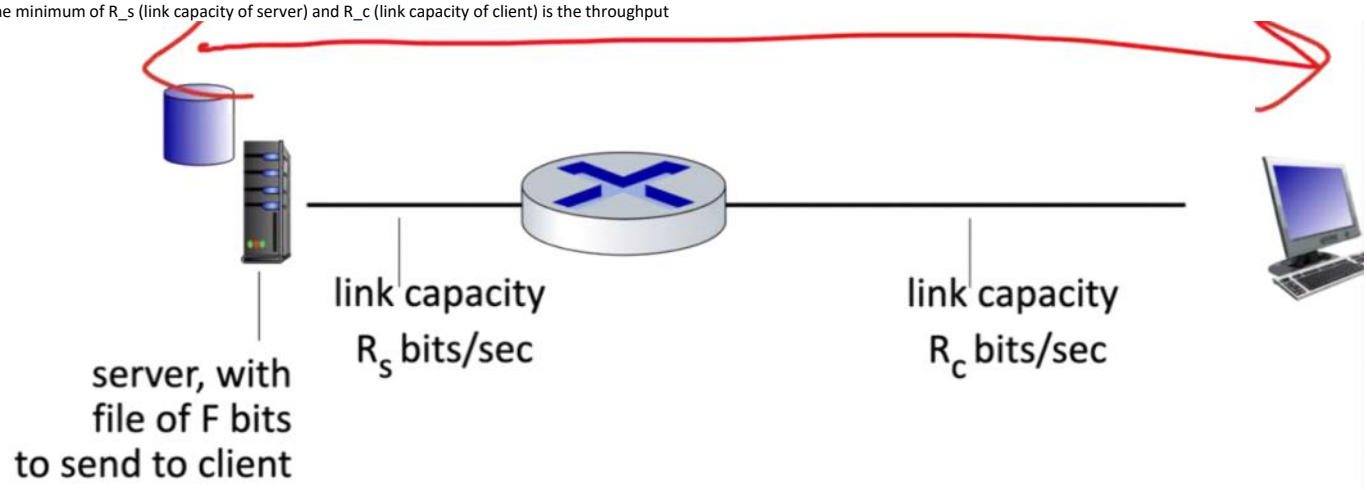
-Arrival time = traffic intensity (number of bits in packet * average packet arrival rate / transmission rate) vs. average queuing delay graph is IMPORTANT:



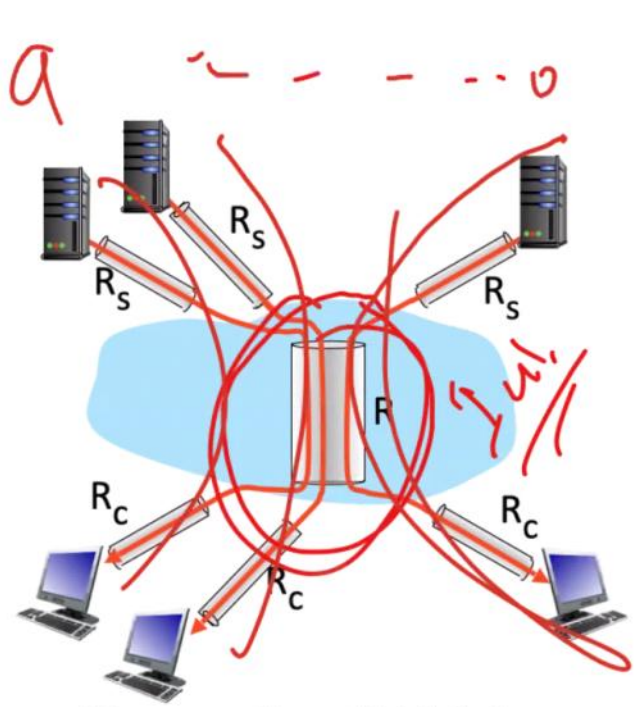
-"traceroute" is a program that provides a measurement for delay - it sends your packets three separate times and takes the average

-Sum of the arriving packets will be much faster than the output speed of the link, that's why we have a queue (buffer) causing packet delay/loss
-End-to-end throughput - time from source to destination
    -There's instantaneous throughput which is the rate at a point and time
    -We more commonly use average end-to-end throughput, or throughput over a long time

-The minimum of R_s (link capacity of server) and R_c (link capacity of client) is the throughput



-Bottleneck links are close to you (the access network)
-Minimum of R_c, R_s, and R



10 connections (**fairly**) share backbone **bottleneck link *R* bits/sec**

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

* Check out the online interactive exercises for more examples: http://gaia.cs.umass.edu/kurose_ross/

9/5 - Networks Layers

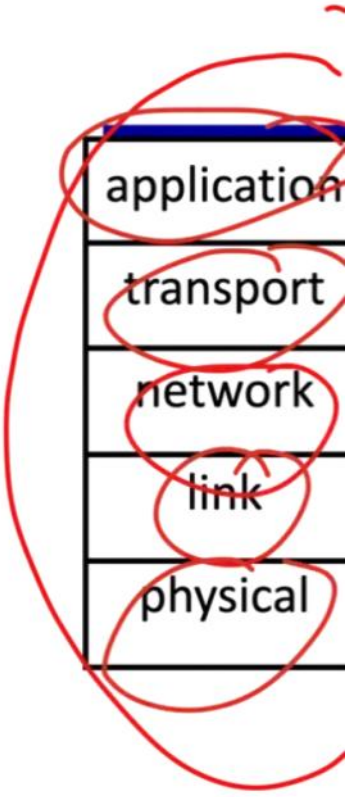
Tuesday, September 3, 2024 6:19 PM

PROTOCOL LAYERS, SERVICE MODELS:
-Airplane analogy, there's several layers to traveling on an airplane

-The main idea:

Internet protocol stack

- *application:* supporting network applications
  - IMAP, SMTP, HTTP
- *transport:* process-process data transfer
  - TCP, UDP
- *network:* routing of datagrams from source to destination
  - IP, routing protocols
- *link:* data transfer between neighboring network elements
  - Ethernet, 802.11 (WiFi), PPP
- *physical:* bits "on the wire"

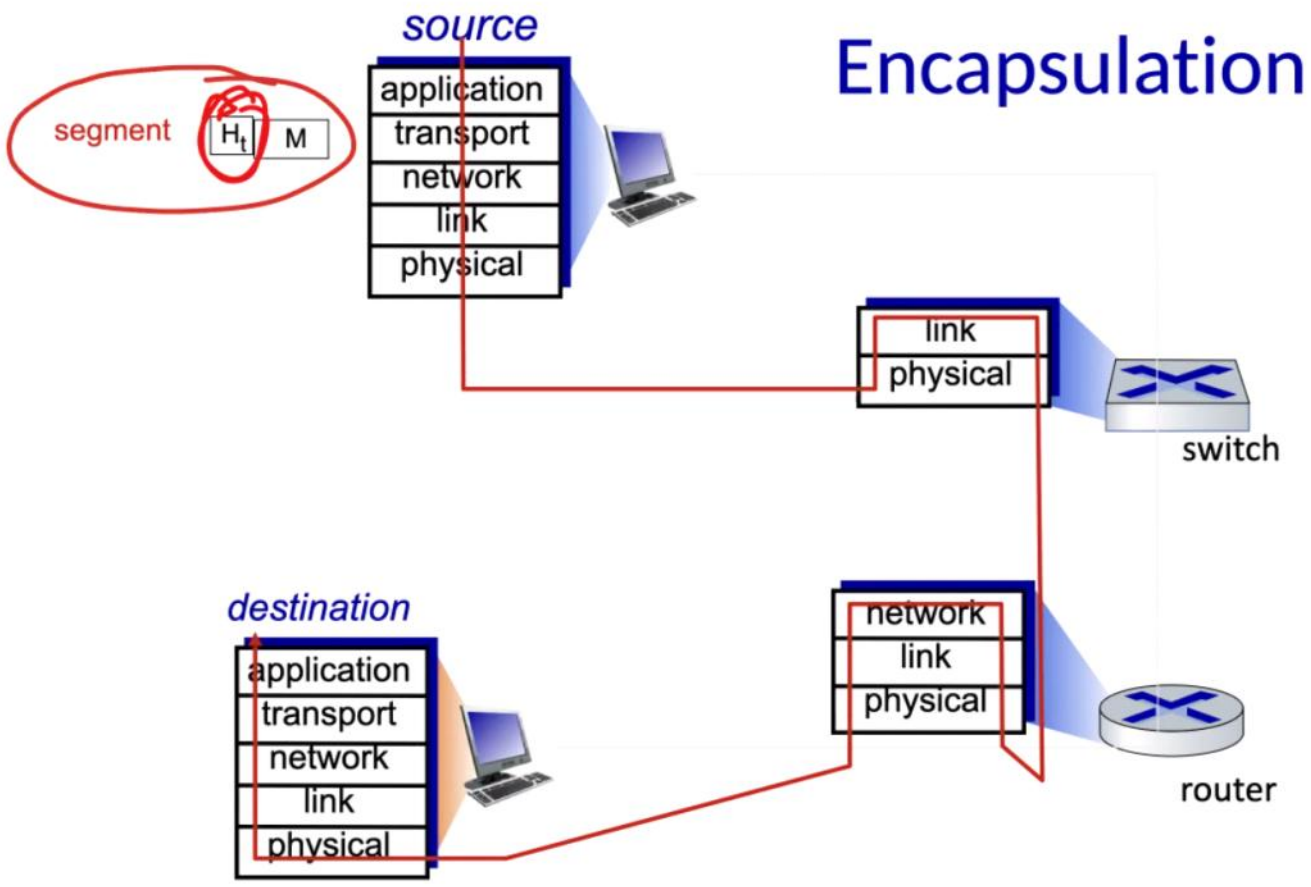


-We split the network into several layers for the same reason you're not the only one in charge of flying your plane and getting through airport security: it's less complex, it causes less problems for the next layer if one layer fails (separation of responsibilities),

-Application layer is facing the user/customer, gives the user/customer an application to use (e.g. using HTTP to serve the user/customer); handles your request and returns you your web page, that's it
-Transport layer is responsible for sending your request to the Google server (if you're going to google.com) process reliably (using TCP) and is responsible for reliably sending your Web page back to you - you don't want a Web page without something correctly loaded on it
-The network layer is responsible for actual routing, it determines the best route/path for your request to go to the destination (this is done through routing protocols), these paths use Dijkstra's shortest path algorithm, these paths are communicated via IP addresses
-The link layer is responsible for sending the data from each hub to the next using the correct protocol (e.g. if your data gets sent to fiber optics, if it gets sent to a coaxial cable, if it gets sent to an ethernet, etc. it doesn't matter, link layer will handle this communication between these different hubs)
-The physical layer is how data gets converted from bits and bytes to a light, electric, or radio signal (which we don't care about in CS, that's computer engineering)

-The whole process is also is called TCP-IP stack

-Data is sent with header and payload, we add more data to the packet before sending it:



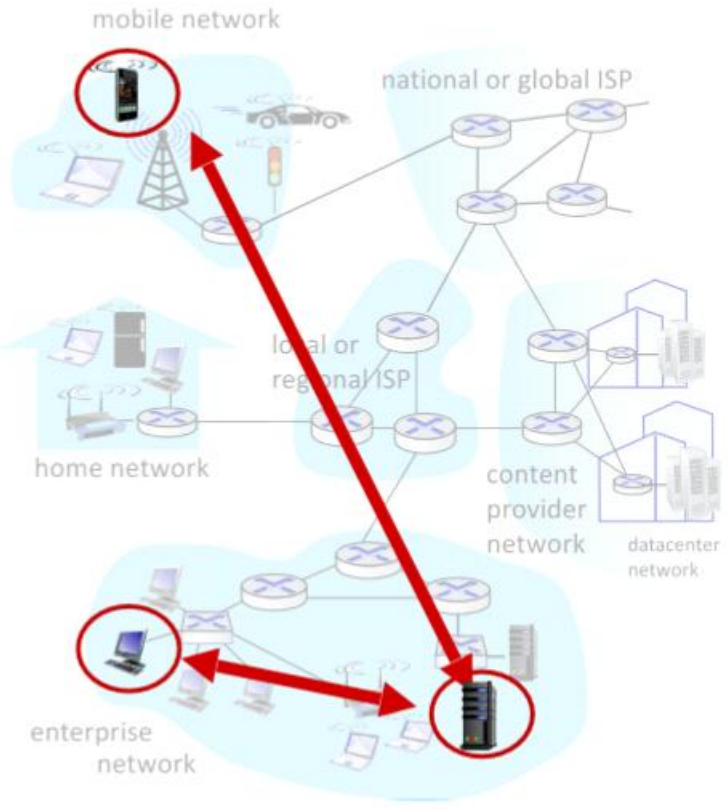
-Encapsulation is about converting data into a lower form, decapsulation is the opposite

9/5 - Client-Server vs P2P

Monday, September 16, 2024 6:24 PM

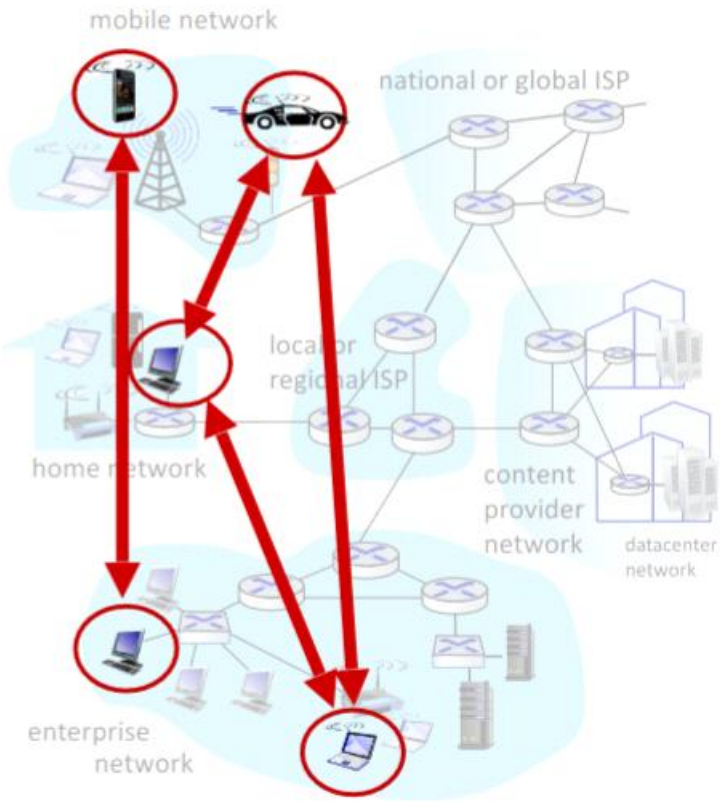
APPLICATION LAYER
-Goal of the application layer: come up with some application to serve a client-server
-Transport, network, data/link, and physical layers are handled by the OS; application layer is not
-Some network applications: social networking, web apps, text messaging, email, multi-user network games, streaming stored videos (Netflix, Hulu, etc.), P2P sharing, etc

Client-server paradigm/architecture:
-Web developers build software for the client and the server - you don't have to write software on the routers along the path
-Network core devices do not run user applications (router does not run HTTP, your computer's OS does)
-The server is a host that is always on (you don't want google.com coming down), it's a permanent address (you expected google.com to remain the same IP address), it's often in the data center (or cloud) for scaling
-Clients are intermittently connected (you might not always be connected), you may have a dynamic IP address (can connect from multiple different locations), clients don't communicate with each other (you cannot see someone else's web browser)
-Examples of the client-server paradigm: HTTP, IMAP, FTP
-Client-server diagram:



-Alternatively, if you use P2P (peer-to-peer) paradigm/architecture such as BitTorrent, a client can communicate with another client to download so there's no permanent server (if you turn off your computer, you can't access the "server" because you, as the client, are the server); this is a good design

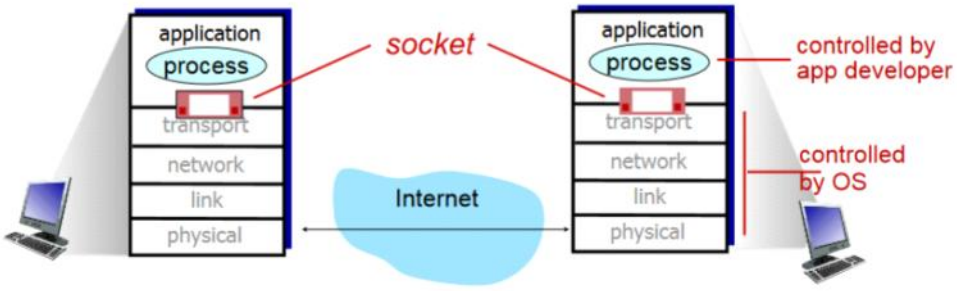
because it's decentralized (there's no single-point of failure like a major server for google.com) and there's self-scalability (the more users/clients that join the network, the faster some other client can download)
-P2P diagram:



9/10 - Sockets, HTTP

Thursday, September 5, 2024 3:13 PM

-Process: a program running within a host machine (from OS)
-If you have two different processes running on two different machines, they can communicate via sockets
    -Sockets make is possible to have process-to-process communication between two physical computers
    -Sockets are defined as inter-process communication (IPC) in OS
    -In networking, we call it "message exchange" - processes in different hosts communicate by exchanging messages
    -We have a client process and a server process - the client process initiates communication and the server process waits to be contacted
    -This all only applies to client-server architecture, on peer-to-peer architecture, a client and server process are both run on your machine
-Process sends/receives messages to/from its socket
    -A socket is an interface - a door to the operating system (it lets you enter the room); every door has a number in a hotel too (which is equivalent to a port number) and the actual address where the hotel is located is the IP address, you address the socket by the port number AND the IP address (you need to know the hotel address and the hotel room number to deliver a pizza):



    -Example: port 80 (HTTP) and IP address 128.119.245.12 brings us to the UMass website
-Each computer has a NIC (network interface card)
-Application layer is controlled by the developer, but once you want to access a socket, you need to talk to the operating system
-Example port numbers:
    -HTTP server is port 80, mail server is port 20
-Types of messages exchanged: requests (from client asking the server) and responses (from server back to the client)
-Each message has syntax - what fields in messages and how fields are delineated
-Message semantics - meaning of information in fields
-Rules for when and how processes send and respond to messages
-Open protocols:
    -Defined in RFCs (Request for Comments)
    -Allows for interoperability
    -Examples: HTTP, SMTP
-Proprietary protocols:
    -They have no RFC numbers
    -Skype, Zoom

Transport services for an app:
-Data integrity - how close data is to what's expected
    -The operating system has to provide us exactly what we want with no loss - some file we're downloading or a web page
    -For audio and video, we don't need 100% data integrity (reliability) because the OS would have to do more work than is necessary - some minor buffering on Zoom isn't going to hurt
-Timing - some apps are sensitive to timing (for video games, it's very important), some are not
-Throughput - the amount of data (bits) you can send per second; some apps require minimum throughput to be effective (you need high throughput to see more frames or video for video conferencing apps), other apps don't require much throughput
-Security - also important for apps
-Transport service requirements for different 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 service vs. UDP service
- -TCP is the protocol that provides 100% reliability (data integrity)
- -UDP does not provide reliability (data integrity)

-TCP:
- -Flow control ensures the sender won't be overwhelmed by receiver (DDoSing)
- -Congestion control - throttle sender when network overloaded which causes data loss
- -Connection-oriented - a client-server relationship must be well established in TCP
- -Does not provide: timing, minimum throughput guarantee, security

-UDP:
- -Does not provide: reliability, flow control, congestion control, timing, throughput guarantee, security, or connection setup
- -We use UDP because it's easy to implement

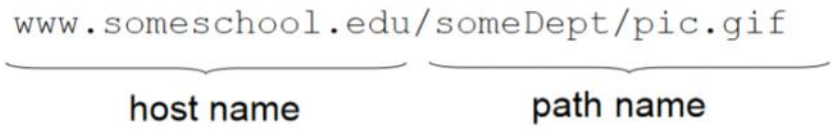
-Internet applications and transport protocols:

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

-Interactive games usually don't require reliability so you can use UDP

-Transport Layer Security (TLS) - TCP doesn't encrypt your data (such as your password), this is a layer on top of TCP which hides your password before you pass it

Web and HTTP
-Base HTML files consist of objects (images, audio files, etc.) which are referenced each by an addressable URL:



-Client makes an HTTP request to the server, server replies back with an HTTP response to the client
-HTTP uses TCP under the hood

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

-HTTP is "stateless" - meaning when a Web browser acts for a Web page, the server doesn't have to know who the client/user is

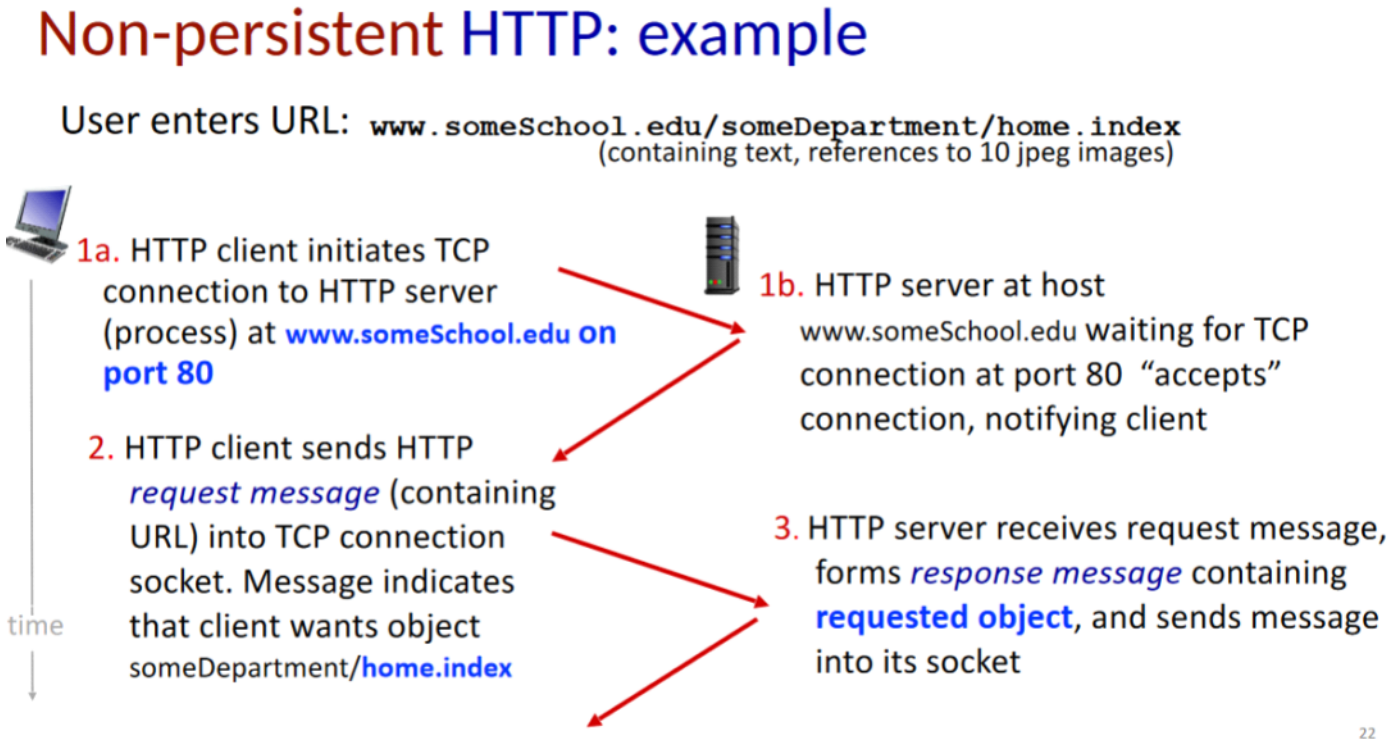
-Server cookies are the way of keeping state of the Web page (that way Google and Facebook can track who their client/user is)

-This means that if you ask for a Web page, then the server shouldn't know you visited when you come back 5 minutes later

-Two types of HTTP connections:

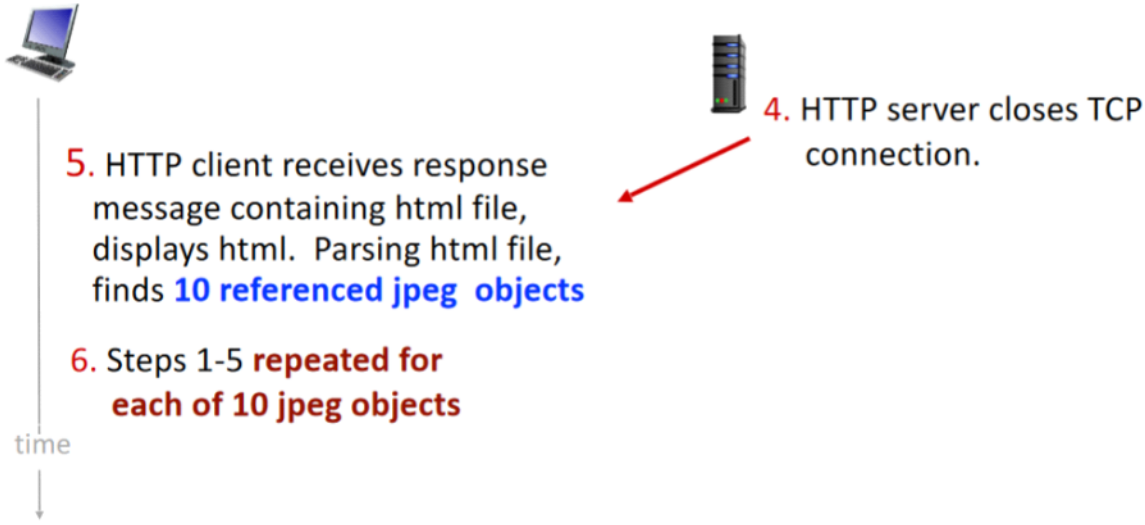
-Non-persistent HTTP connection

-You can only download or send one object then you close the connection, you have to keep reopening the TCP connection



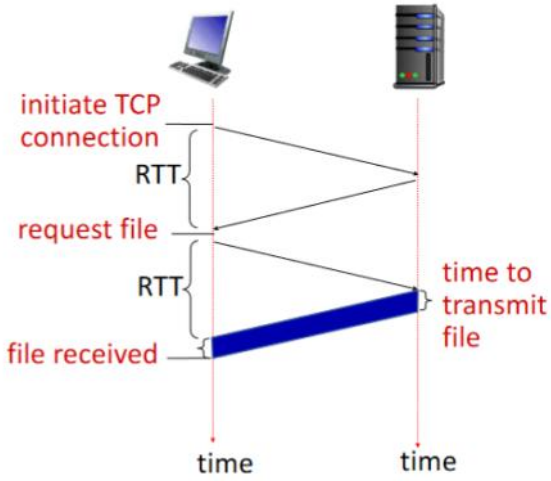
Non-persistent HTTP: example (cont.)

User enters URL: `www.someSchool.edu/someDepartment/home.index`
(containing text, references to 10 jpeg images)



23

-RTT (round trip time) - time for a small packet to travel from client to server and back:



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

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

-Persistent HTTP connection (HTTP 1.1)

-You can download multiple objects on a TCP connection then you can close the connection

-Issues with non-persistent HTTP as to why we use persistent HTTP now: requires 2 RTTs per object, OS overhead for each TCP connection, browsers often open multiple parallel TCP connections to fetch referenced objects in parallel (which overwhelms the server)

-Main issue that persistent HTTP connection fixes: as little as one RTT for all referenced objects (cutting response time in half)

-Uses "pipelining"

-With persistent HTTP, you can see all the images on the Web page download at once (improving user experience); with non-persistent HTTP, images download one a time on the Web page; performance is still the same regardless of both

9/12 - More HTTP

Thursday, September 12, 2024 1:57 PM

-HTTP request message includes some header lines:

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

carriage return character

line-feed character

request line (GET, POST, HEAD commands)

header lines

```
GET /index.html HTTP/1.1\r\n
Host: www-net.cs.umass.edu\r\n
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X
   10.15; rv:80.0) Gecko/20100101 Firefox/80.0 \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
Connection: keep-alive\r\n
\r\n
```

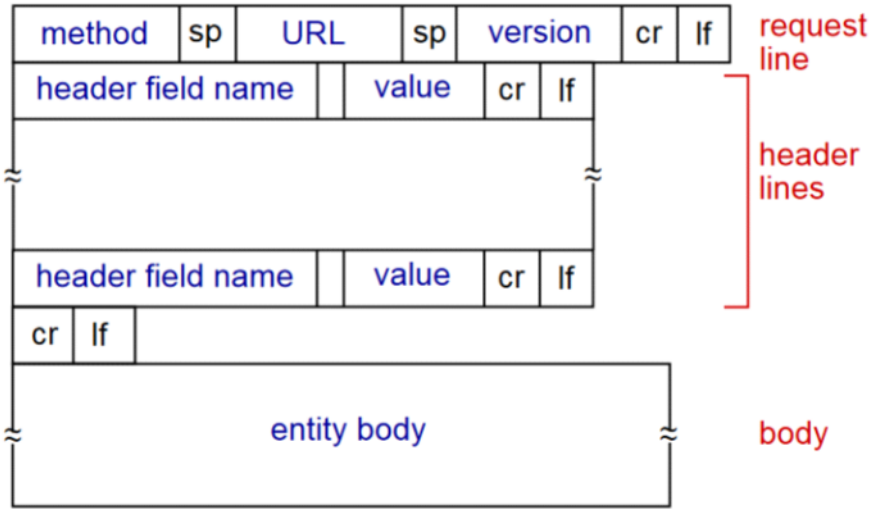
carriage return, line feed at start of line indicates end of header lines

* Check out the online interactive exercises for more examples: http://gaia.cs.umass.edu/kurose_ross/interactive/

-IMPORTANT: use this for the example of how to make an HTTP request message above, including using the carriage return character and line-feed character ("\r\n")

-HTTP request message format:

HTTP request message: general format



-Kinda cool tidbit on HEAD type of HTTP request message:

Other HTTP request messages

POST method:

- web page often includes form input
- user input sent from client to server in entity body of HTTP POST request message

GET method (for sending data to server):

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

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

HEAD method:

- requests headers (**only**) that would be returned *if* specified URL were requested with an HTTP GET method.
- Same as GET method but response does not return anything in the body

PUT method:

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

28

-If you use the "HEAD" HTTP request, you can use it to test if your API actually works or not
-Giovanni said that "PUT" methods caused a lot of security issues
-Look up: "PATCH" HTTP request (Giovanni said this probably is a security risk too)

-HTTP response message includes some header also:

HTTP response message

status line (protocol status code status phrase)

header lines

data, e.g., requested HTML file

```
HTTP/1.1 200 OK
Date: Tue, 08 Sep 2020 00:53:20 GMT
Server: Apache/2.4.6 (CentOS) OpenSSL/
   1.0.2k-fips PHP/7.4.9 mod_perl/2.0.11
   Perl/v5.16.3
Last-Modified: Tue, 01 Mar 2016 18:57:50 GMT
ETag: "a5b-52d015789ee9e"
Accept-Ranges: bytes
Content-Length: 2651
Content-Type: text/html; charset=UTF-8
\r\n
data data data data data ...
```

-HTTP response status codes:

HTTP response status codes

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

200 OK

- request succeeded, requested object later in this message

301 Moved Permanently

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

400 Bad Request

- request msg not understood by server

404 Not Found

- requested document not found on this server

505 HTTP Version Not Supported

-Bonus: https://http.cat/

-Here's a program called "netcat" or "nc" that you can use to connect to a server:

Trying out HTTP (client side) for yourself

1. netcat to your favorite Web server:

% nc -c -v gaia.cs.umass.edu 80

- opens TCP connection to port 80 (default HTTP server port) at gaia.cs.umass. edu.
- anything typed in will be sent to port 80 at gaia.cs.umass.edu

2. type in a GET HTTP request:

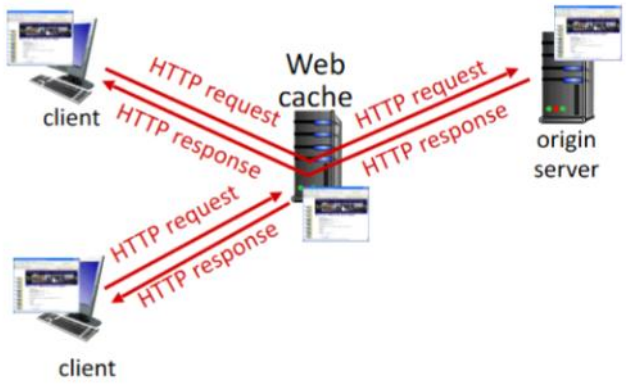
```
GET /kurose_ross/interactive/index.php HTTP/1.1
Host: gaia.cs.umass.edu
```

- by typing this in (hit carriage return twice), you send this minimal (but complete) GET request to HTTP server

3. look at response message sent by HTTP server!

(or use Wireshark to look at captured HTTP request/response)

-Cache = a store of recently accessed data
-We have Web caches to reduce load (e.g. network traffic, response time) on the Web page
-Instead of asking the same server over and over for the page, you ask the Web cache instead which will load the Web page for you instead of asking the server
-Goal: Satisfy client requests without involving origin server
-A Web cache is also referred to as a Web proxy
-Web cache is server to the original requesting client and client to the original server, it's an intermediary:



-You can add an extra HTTP header line to explicitly tell the Web cache not to cache anything

- server tells cache about object's allowable caching in response header:

```
Cache-Control: max-age=<seconds>
```

```
Cache-Control: no-cache
```

-The Internet has plenty of caches and enables "poor" content providers to more effectively deliver content

-Example of Web caching (useful for homework):

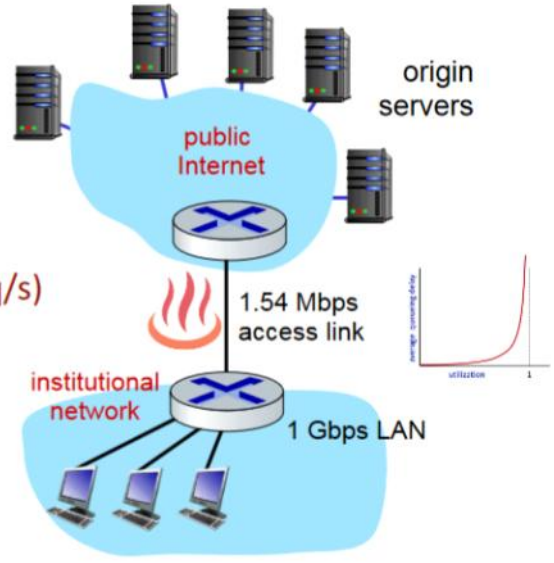
Caching example

Scenario:

- access link rate: **1.54 Mbps**
- RTT from institutional router to server: **2 sec**
- web object size: **100K bits**
- average request rate from browsers to origin servers: **15 requests/sec**
  - avg data rate to browsers: **1.50 Mbps** (100K x 15 req/s)

Performance:

- access link utilization = .97 *problem:* large queueing delays at high utilization!
- LAN utilization: .0015
- end-end delay = Internet delay + access link delay + LAN delay
  = 2 sec + minutes + usecs



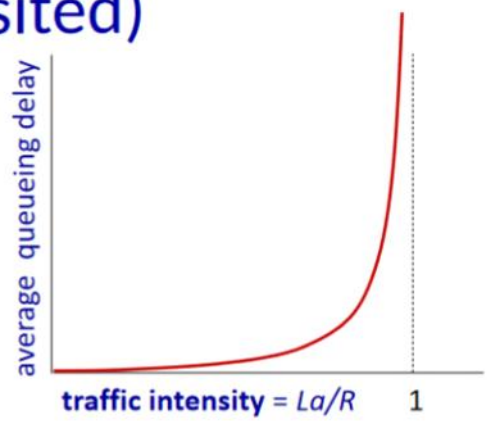
-The graph above is that of arrival time, a throwback from one of the previous lectures, here's a refresher:

Packet queueing delay (revisited)

- R: link bandwidth or transmission rate (bps)

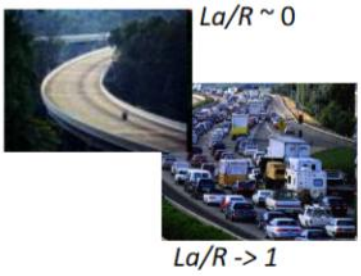
Packet queueing delay (revisited)

- *R:* link bandwidth or transmission rate (bps)
- *L:* packet length (bits)
- *a:* average **packet** arrival rate (packets/sec)
  —> *La: average **bit** arrival rate in (bits/sec)*



*Arrival process is usually **random***

- *La/R* ~ 0: avg. queueing delay small
- *La/R approaches* 1: avg. queueing delay large
- *La/R* > 1: more "work" arriving is more than can be serviced - average delay infinite!



-Solutions to the slow Web caching caused by larging queuing delay at high utilization:

Option 1: buy a faster access link

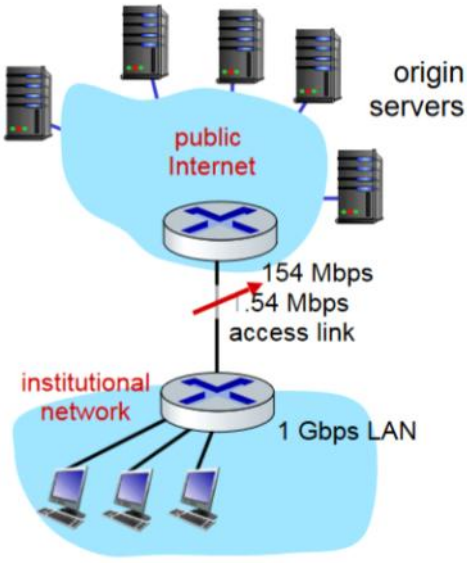
Scenario:

- access link rate: ~~1.54~~ 154 Mbps
- RTT from institutional router to server: 2 sec
- web object size: 100K bits
- average request rate from browsers to origin servers: 15/sec
  - avg data rate to browsers: 1.50 Mbps

Performance:

- access link utilization = ~~.97~~ .0097
- LAN utilization: .0015
- end-end delay = Internet delay + access link delay + LAN delay
  = 2 sec + ~~minutes~~ + usecs msecs

Cost: faster access link (expensive!)



Option 2: install a web cache

Scenario:

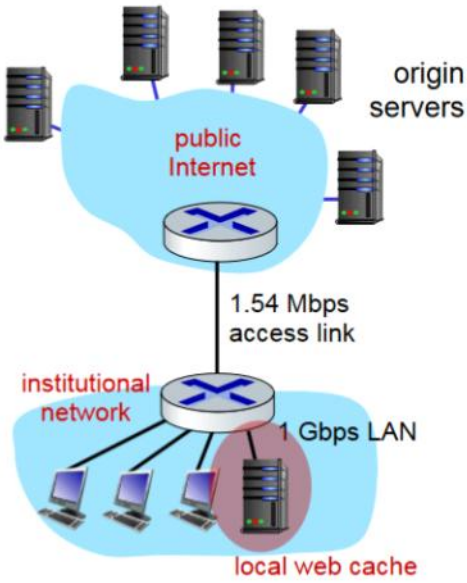
- access link rate: 1.54 Mbps
- RTT from institutional router to server: 2 sec
- web object size: 100K bits
- average request rate from browsers to origin servers: 15/sec
  - avg data rate to browsers: 1.50 Mbps

Cost: web cache (cheap!)

Performance:

- LAN utilization: .?
- access link utilization = ?
- average end-end delay = ?

How to compute link utilization, delay?

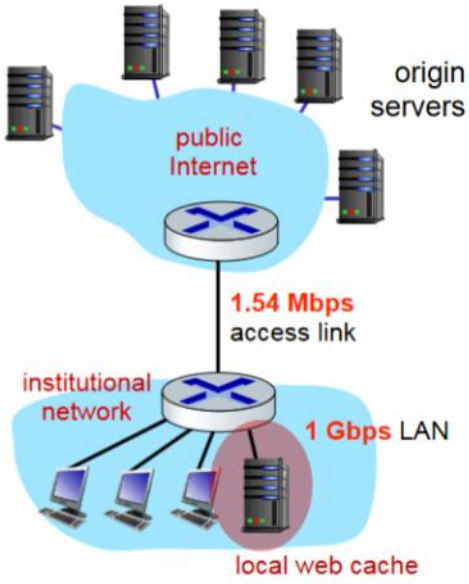


36

Calculating access link utilization, end-end delay with cache:

suppose cache hit rate is 0.4:

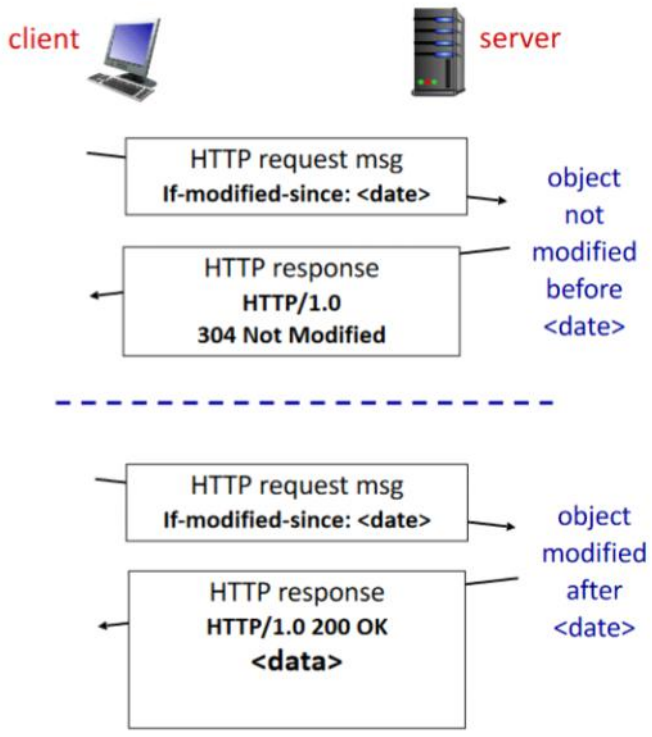
- **40% requests served by cache**, with low (msec) delay
- **60% requests satisfied at origin**
  - rate to browsers over access link
    = 0.6 * 1.50 Mbps = .9 Mbps
  - *access link utilization* = 0.9/1.54 = .58 means low (msec) queueing delay at access link
- average end-end delay:
  = 0.6 * (delay from origin servers)
  + 0.4 * (delay when satisfied at cache)
  = 0.6 (2.01) + 0.4 (~msecs) = ~ 1.2 secs



lower average end-end delay than with 154 Mbps link (and cheaper too!)

37

-Conditional GET - checks if you have a file that's been modified since the last time you got it
-Can be accomplished by adding a line to the HTTP header:



-Cookies, refresher:

-HTTP is "stateless" - meaning when a Web browser acts for a Web page, the server doesn't have to know who the client/user is

-Server cookies are the way of keeping state of the Web page (that way Google and Facebook can track who their client/user is)

-This means that if you ask for a Web page, then the server shouldn't know you visited when you come back 5 minutes later

-Four components that explain how cookies work:

four components:

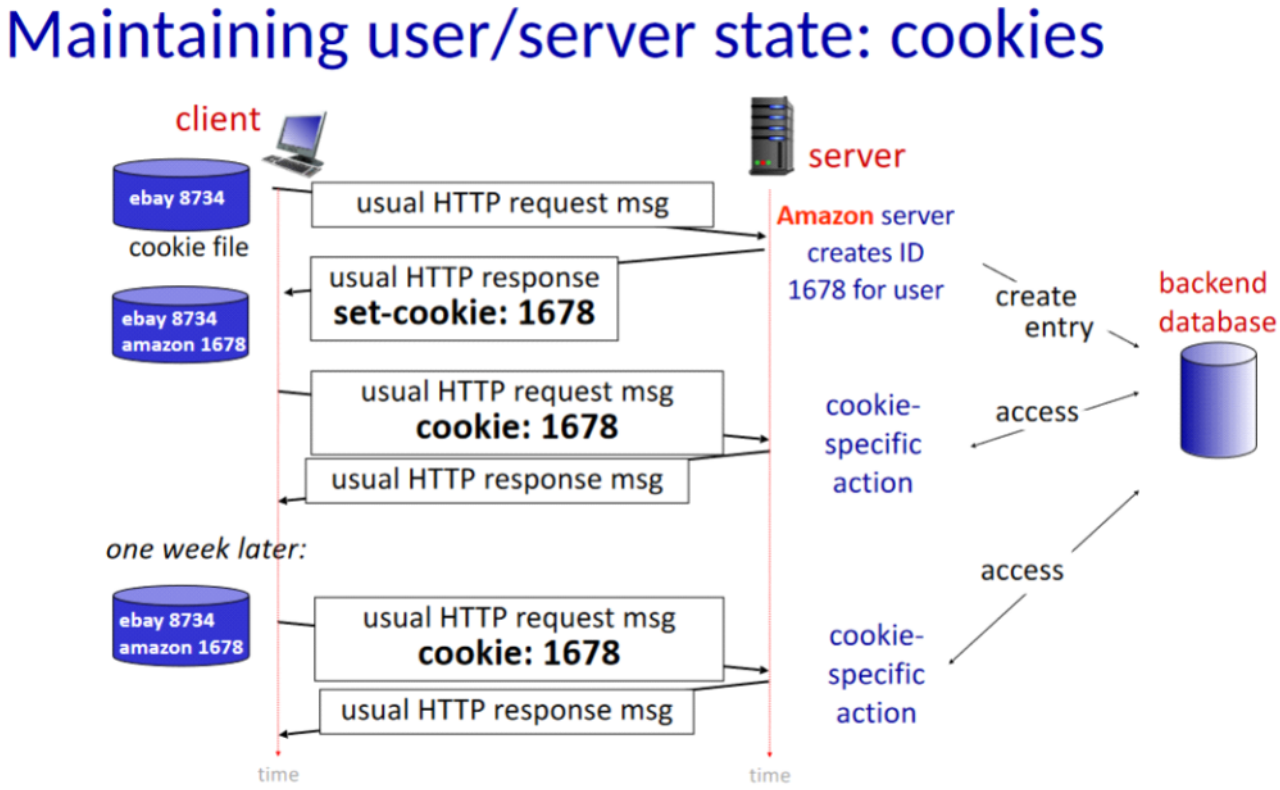
1) cookie header line of HTTP *response* message
2) cookie header line in next HTTP *request* message
3) cookie file kept on user's host, managed by user's browser
4) back-end database at Web site

-Example of how cookies work:

Example:

- Susan uses browser on laptop, visits specific e-commerce site for first time
- when initial HTTP request arrives at site, site creates:
  - unique ID (aka "cookie")
  - entry in backend database for ID
- subsequent HTTP requests from Susan to this site will contain cookie ID value, allowing site to "identify" Susan

-Diagram of how cookies work:



-Ebay already saved a cookie from us visiting their website before
-Amazon saves a cookie in their database (server-side) after I visit their website
-Then, set the cookie for the client locally on the client's browser database
-Amazon knows you're cookie number 1678, so it will use this data to better market to you

-Benefits of cookies:
-Your session data gets saved such as sign-in information (authorization) so you can just log in like you were there before
-Your shopping cart gets saved
-Your data could be used to recommend you better or more optimal items
-User session state (Web email)

-Downsides of cookies:
-Privacy violations - a "tracking cookie" is when Facebook knows what you're doing on Amazon's website
-People can also steal your session IDs

-HTTP/2

-Key goal: decreased delay in multi-object HTTP requests

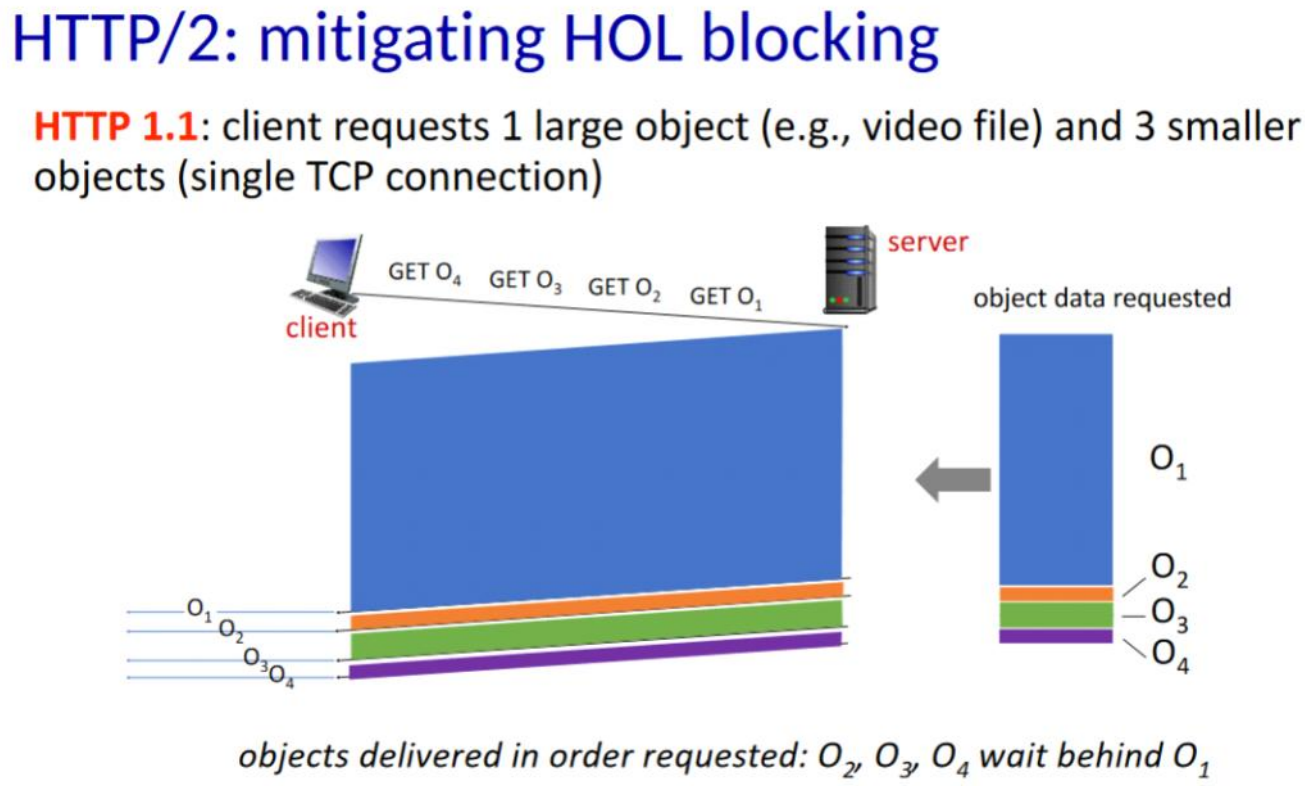
-Drawbacks of HTTP1.1 - did not download frames (TCP connections) for files equally

-HTTP/2 breaks large files into frames that you download equally, there is also a notion of object (image, video, etc.) priority

-Smaller objects were not broken down

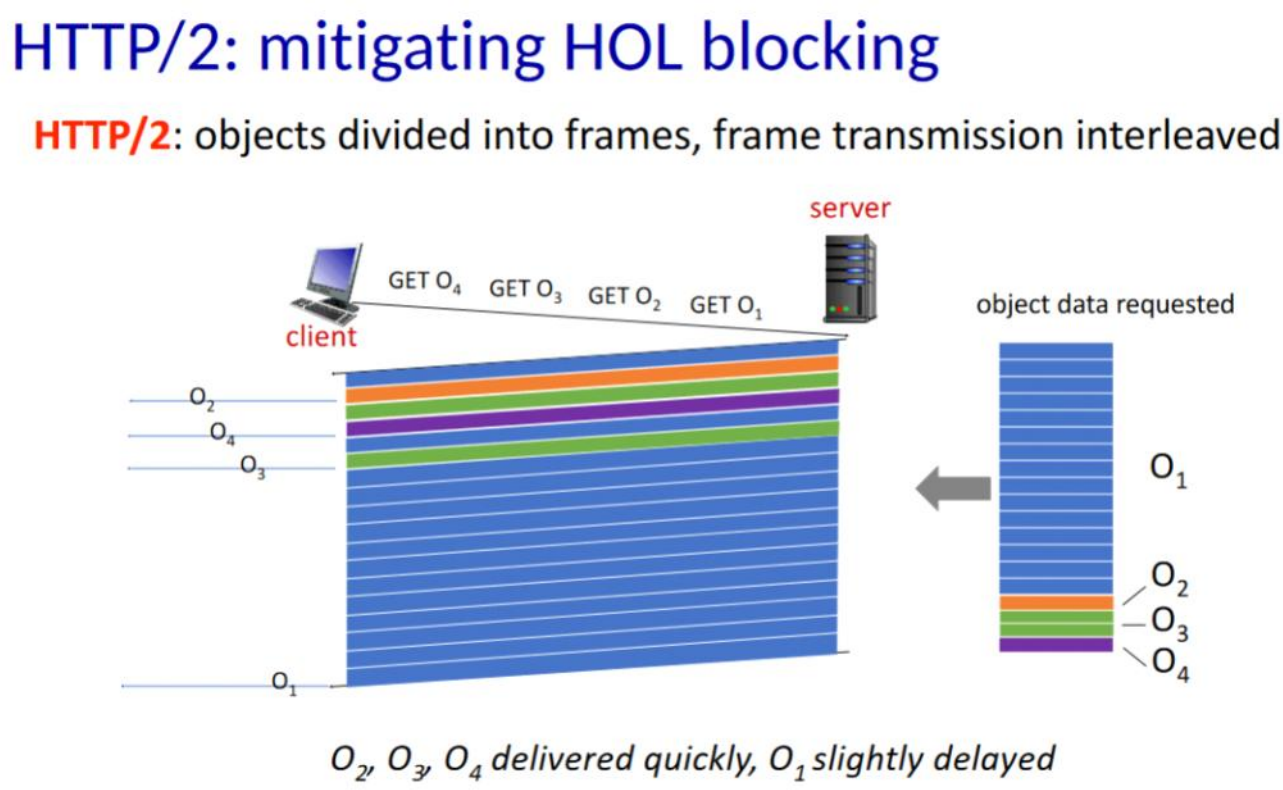
Here's 4 images being loaded in HTTP 1.1 vs. HTTP/2:

-HTTP 1.1:



-In HTTP 1.1, there's no priority so image 1 is loaded in first, slowing down images 2, 3, and 4

-HTTP/2:



-The big thing is that image 2, 4, and 3 will load faster and image 1 will load slower in the background, then finally load in

-Interestingly, here's the time it takes to do TCP connections on HTTP 1.1 vs. HTTP/2:
https://freecontent.manning.com/animation-http-1-1-vs-http-2-vs-http-2-with-push/

-HTTP/3:
- -Adds security
- -Per object error-control and congestion-control - more pipeline over UDP

9/17 - FTP, DNS

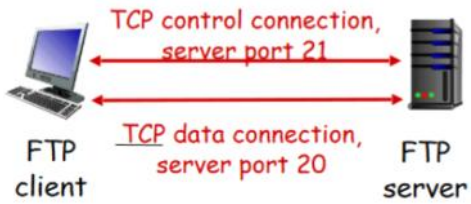
Tuesday, September 17, 2024 1:58 PM

-FTP (File Transfer Protocol)

-File transfer protocol does not encrypt files, everything is in clear text (not secure), any machine can sniff your traffic and know exactly what data you're moving

-FTP is "out of band" - it uses 2 ports and 2 TCP connections (one for control and one for data) - you make a GET request and then the server opens another connection and sends you a Web page on a different connection

-FTP is on port 20/21:



-FTP server maintains state: the server knows the specific user because you're authenticating with the server (unlike HTTP, which has to use cookies to track you)

-Your username and password are also sent in clear text to the server (which is even more unsecure)

-FTP commands and server response codes:

FTP commands, responses

sample commands:

- sent as ASCII text over control channel
- **USER *username***
- **PASS *password***
- **LIST** return list of file in current directory
- **RETR filename** retrieves (gets) file
- **STOR filename** stores (puts/uploads) file onto remote host (ftp server)

sample return codes

- status code and phrase (as in HTTP)
- **331 Username OK, password required**
- **125 data connection already open; transfer starting**
- **425 Can't open data connection**
- **452 Error writing file**

-More of them here: https://www.ietf.org/rfc/rfc959.txt

-IMPORTANT: you still need the carriage return and line feed (newline) after every request line, otherwise the TCP connection won't work

-For the project, use: "Show detailed log" on FileZilla to figure out what FTP status codes are returned for commands you run

-Domain Name System (DNS)

-DNS is a protocol that helps you know the IP address of a certain domain name (aliasing names)

-DNS exists because it's easier to memorize names instead of numbers

-But we still need the IP address for every domain name, so DNS maps the domain name to the IP address

-If you open Wireshark and run it while visiting a Web page, you should see a DNS request come through

-A distributed database contains all the DNS mappings - to separate complexity, we only want a DNS server to tell us what domain name is what IP address, not our router because that would put more work on our router

-DNS services:

-Hostname-to-IP-address translation

-Host aliasing: conical, alias names

-Mail server aliasing

-Load distribution - replicated Web servers have many IP addresses correspond to one name (load balancers)

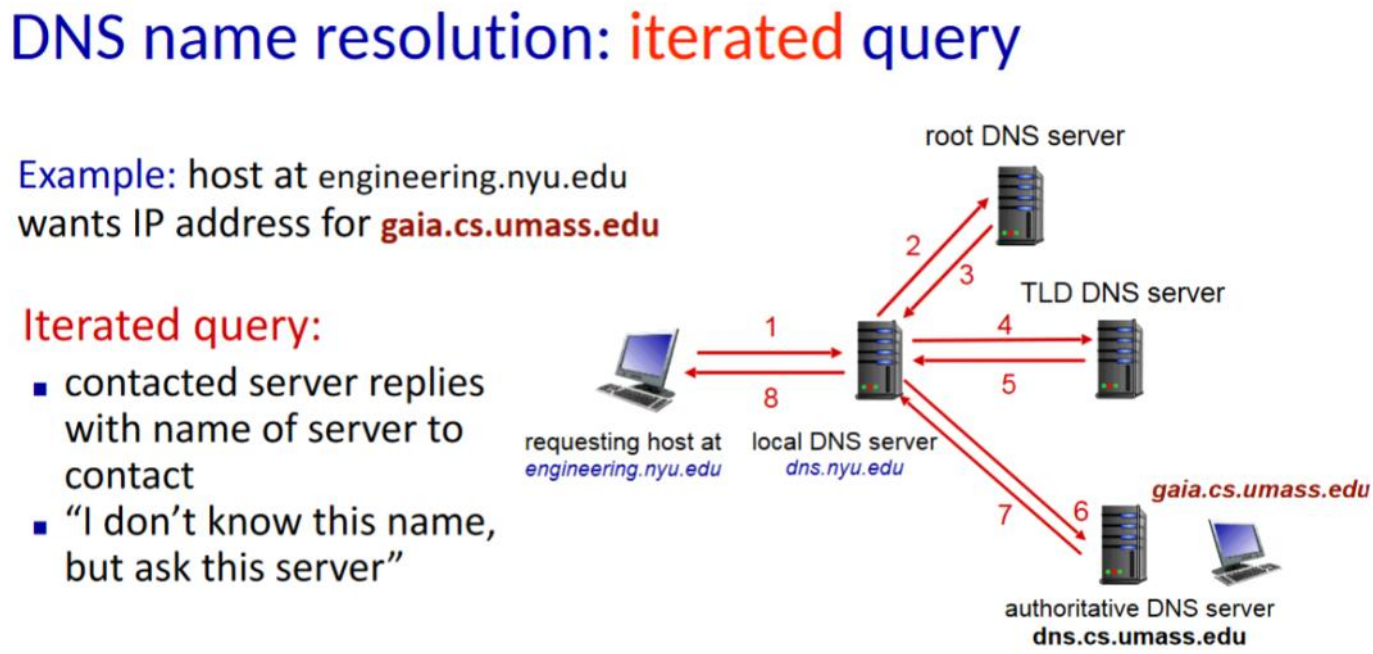
-We don't want to centralize DNS servers because then we have a single point of failure, higher traffic volume, longer distance, and maintenance issues - it doesn't scale well

-Instead, we have very large distributed databases containing billions of records for DNS

-These distributed databases handle trillions of queries per day where there are many more reads than writes (e.g. google.com doesn't change IP address)

-DNS is physically decentralized

-One server has all the mappings but other servers help you get there:



-You (the request host) send the domain to the local DNS server -> local DNS server doesn't have it but sends you to a more root DNS server -> top level domain (TLD) knows you're looking for an .edu site -> then you look for dns.cs.umass.edu with an authoritative DNS server by bouncing it around other servers who might know what the IP address

-You cache the IP address on the local DNS server (your local Web cache) to decrease the load on the edge router of the website you're requesting; caching improves response time

-Time to live (TTL) says how long your DNS translation is saved on your Web cache before it's dropped

-Top level domain (TLD) servers are typically cached in local DNS servers

-If cached entries are out of date, your TTL expires -> best-effort name-to-address translation -> authoritative server tells other DNS servers what TTL value to use

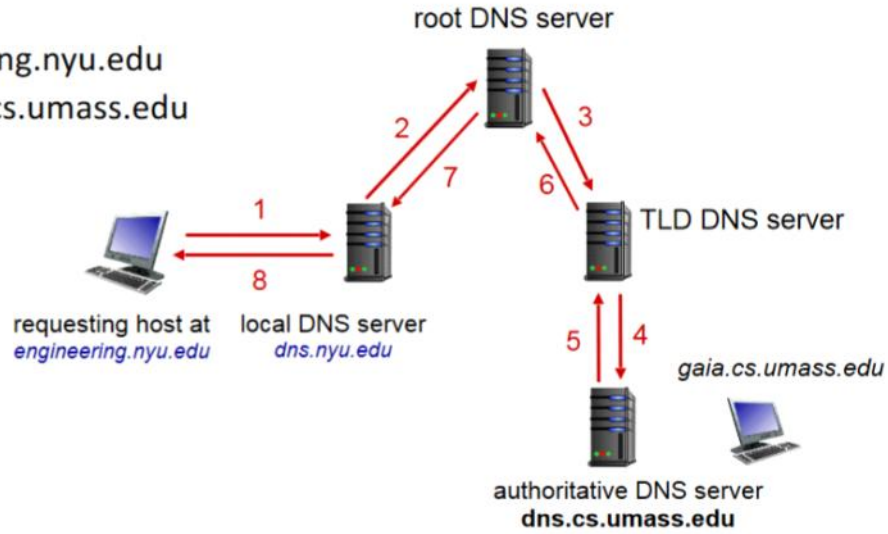
-Recursive DNS servers:

DNS name resolution: recursive query

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

Recursive query:

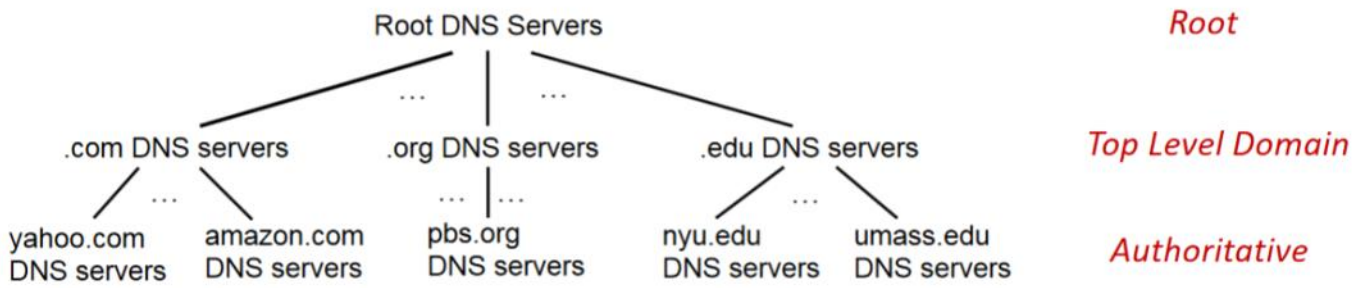
- puts burden of name resolution on contacted name server
- heavy load at upper levels of hierarchy?



62

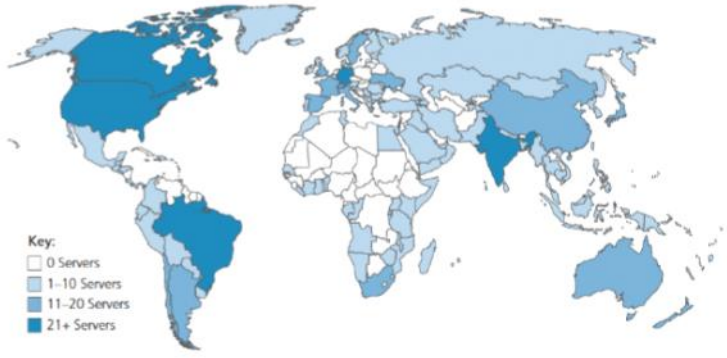
-Iterative query asks the local DNS server, which does all the work for you, recursive query asks around

-DNS hierarchy:



-Root DNS servers - ICANN or Internet Corporation for Assigned Names and Numbers (non-profit organization) manages root DNS servers, most are in the USA:

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



-Top-level domain (TLD) servers:

Top-Level Domain (TLD) servers:

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

-Authoritative DNS servers - maintained by the organization or service provider (has the actual DNS mapping)

9/19 - More DNS, SMTP, Socket Programming

Tuesday, September 17, 2024 1:58 PM

-Each ISP has a local DNS server, you can find it using: "ipconfig /all" on Windows or "scutil --dns" on macOS (and Linux)
-DNS resource records (RR) are stored in this format:

DNS records

DNS: distributed database storing resource records (RR)

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

type=A

- name is hostname
- **value** is **IP address**

type=NS

- name is domain (e.g., foo.com)
- **value** is **hostname** of authoritative name server for this domain

type=CNAME

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

type=MX

- **value** is **name** of SMTP mail server associated with name

-A is hostname
-NS is an unaliased domain name
-CNAME is an alias to the original (canonical) name (canonical names are usually longer)
-MX is a mail server

-Actual structure of a DNS protocol message:

DNS protocol messages

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

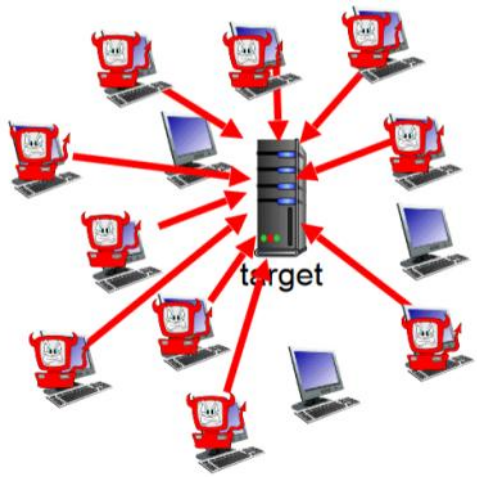
message header:

- identification: 16 bit # for query, reply to query uses same #
- flags:
  - query or reply
  - recursion desired
  - recursion available
  - reply is authoritative

2 bytes	2 bytes
identification	flags
# questions	# answer RRs
# authority RRs	# additional RRs
questions (variable # of questions)	
answers (variable # of RRs)	
authority (variable # of RRs)	
additional info (variable # of RRs)	

-"nslookup" command shows you the DNS mapping between the domain and the IP address
-A DNS registrar (e.g. GoDaddy) is the person who inserts an NS or A type DNS RR (request record) into the .com top-level domain (TLD) DNS server

-Denial of Service (DoS) attack - attackers make resources (server, bandwidth) unavailable to legitimate traffic by overwhelming resource with bogus traffic

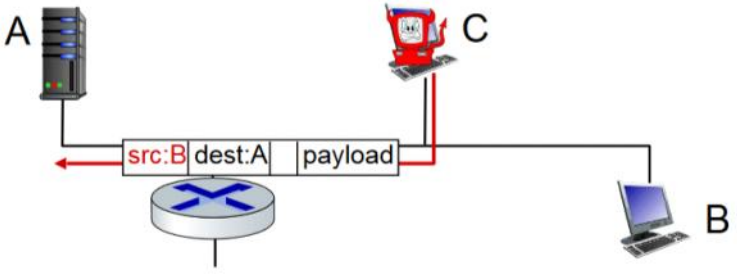


-No root server being hit with a bunch of traffic has worked up to this point due to traffic filtering and local DNS servers caching Ips of TLD servers, allowing root server bypass
-TLD servers being hit with a DoS attack is potentially more dangerous, if you can shut down a .com or .edu DNS server
-Distributed Denial of Service (DDoS) attack - use everyone else's computer (zombie machines) to run a DoS attack

-Spoofing attacks - intercept DNS queries, returning bogus replies; pretending to be someone else, someone found a way to be in the middle of the communication and send a reply faster than Google could to tell you that google.com is a certain IP address because someone told you google.com was another IP address before Google did

Spoofing attacks illustration:

E.g., IP spoofing: send packet with false source address



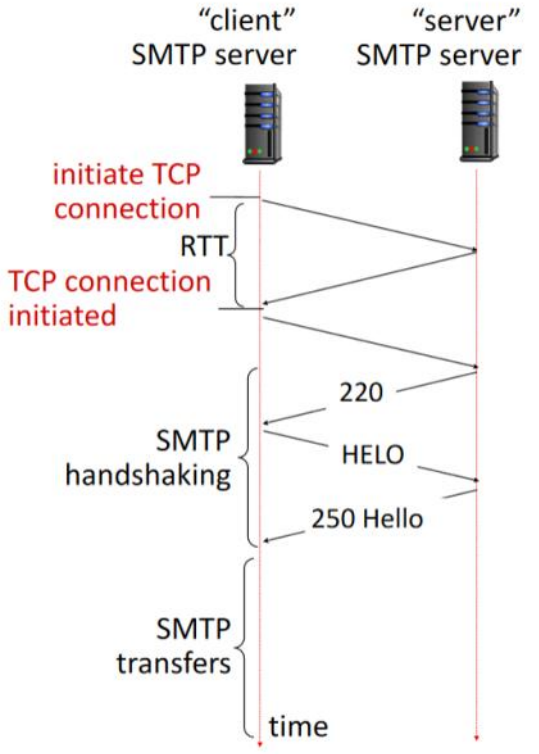
-If you modify "/etc/hosts", you can add a new custom DNS mapping for your machine that can act as a spoofing attack (or you'll get an SSL error)

-SMTP

-Email has three major components: user agents, mail servers, and simple mail transfer protocol (SMTP)
-Mail servers: mailbox contains incoming messages for users and message queue of outgoing (soon to be sent) mail messages
-SMTP uses TCP to reliably transfer email message from client (mail server initiating the connection) to server
-Occurs over port 25
-Three phases of transfer: SMTP handshaking (greetings), transfer of messages, closure
-There's SMTP commands and responses (like how HTTP has 200 for OK and 404 and such):

- command/response interaction (like HTTP)
  - commands: ASCII text
  - response: status code and phrase

-Diagram of SMTP:



78

-Example of how SMTP works:

Scenario: Alice sends e-mail to Bob

1) Alice uses UA to compose e-mail message "to" bob@someschool.edu

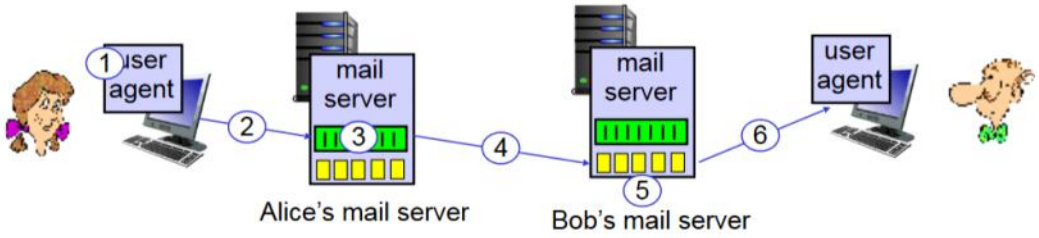
2) **Alice's UA** sends message to her mail server using SMTP; message placed in **message queue**

3) **client** side of **SMTP** at mail server opens TCP connection with Bob's mail server

4) **SMTP client** sends Alice's message over the TCP connection

5) Bob's mail server places the message in **Bob's mailbox**

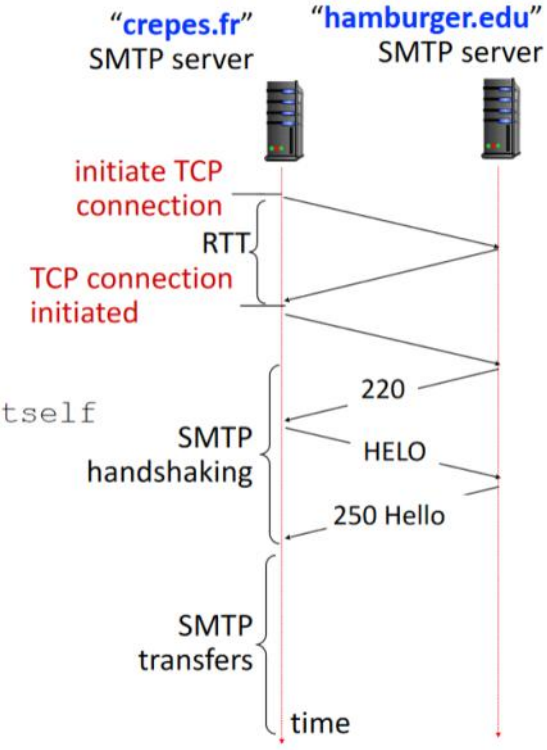
6) **Bob** invokes his **user agent** to read message



-What it actually looks like:

Sample SMTP interaction

```
S: 220 hamburger.edu
C: HELO crepes.fr
S: 250 Hello crepes.fr, pleased to meet you
C: MAIL FROM: <alice@crepes.fr>
S: 250 alice@crepes.fr... Sender ok
C: RCPT TO: <bob@hamburger.edu>
S: 250 bob@hamburger.edu ... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: Do you like ketchup?
C: How about pickles?
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 hamburger.edu closing connection
```



80

-SMTP vs. HTTP:

SMTP: observations

comparison with HTTP:

- HTTP: client pull
- SMTP: client push
- both have ASCII command/response interaction, status codes
- HTTP: each object encapsulated in its own response message
- SMTP: multiple objects sent in **multipart** message

-Attachments on emails are sent in multi-part messages to the server
-SMTP uses persistent connection so there's only one TCP connection and you send everything you're going to then close it
-SMTP uses the "." to end the message

- SMTP uses persistent connections
- SMTP requires message (header & body) to be in 7-bit ASCII
- SMTP server uses CRLF.CRLF to determine end of message

-Mail message format (syntax) has its own RFC separate from the SMTP
-Internet Mail Access Protocol (IMAP) allows you to have retrieval, deletion, and folder creation for messages stored on the mail server, IMAP is used to retrieve email messages

-Socket programming
-Goal of socket programming: learn how to build client/server applications that communicate using sockets
-Two socket types for transport services: UDP (unreliable datagram) and TCP (reliable, byte stream-oriented)
-Socket programming example:

Application Example:

1. client reads a line of characters (data) from its keyboard and sends data to server
2. server receives the data and converts characters to uppercase
3. server sends modified data to client
4. client receives modified data and displays line on its screen

-Socket programming with UDP - no connection between client and server, no handshaking, sender explicitly attaches IP destination address and port number to each packet, receiver extracts sender IP address and port number from received packets; transmitted data may be lost or received out of order

-Socket programming with TCP - client must contact server where client contacts server by creating a TCP socket which specifies the IP address and port number of the server process, when contacted by the client, the server creates a new TCP socket for the that client but can talk to multiple clients where the source port number is used to distinguish between clients

-Python code for socket programming uses the socket library, for UDP

```
# UDPClient.py
from socket import *

clientSocket = socket(AddressFamily.AF_INET, SocketKind.SOCK_DGRAM)
clientSocket.sendto(input("Enter Message: ").encode(), ("", 2000))
message, sadadad = clientSocket.recvfrom(2048)
print(message.decode())

# UDPServer.py
from socket import *
serverSocket = socket(AddressFamily.AF_INET, SocketKind.SOCK_DGRAM)
serverSocket.bind(("", 2000))
while True:
    message, clientAddress = serverSocket.recvfrom(2048)
    sererSocket.sendto(message.decode().upper().encode(), clientAddress)
```

-Python code for socket programming with TCP:

Python TCPClient

```
from socket import *
serverName = 'servername'
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName,serverPort))
sentence = raw_input('Input lowercase sentence:')
clientSocket.send(sentence.encode())
modifiedSentence = clientSocket.recv(1024)
print ('From Server:', modifiedSentence.decode())
clientSocket.close()
```

create TCP socket for server, remote port 12000 → clientSocket = socket(AF_INET, SOCK_STREAM)

No need to attach server name, port → clientSocket.send(sentence.encode())

Python TCPServer

create TCP welcoming socket

server begins listening for incoming TCP requests

loop forever

server waits on accept() for incoming requests, new socket created on return

read bytes from socket (but not address as in UDP)

close connection to this client (but *not* welcoming socket)

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET,SOCK_STREAM)
serverSocket.bind(('',serverPort))
serverSocket.listen(1)
print 'The server is ready to receive'
while True:
    connectionSocket, addr = serverSocket.accept()

    sentence = connectionSocket.recv(1024).decode()
    capitalizedSentence = sentence.upper()
    connectionSocket.send(capitalizedSentence.
                                          encode())
    connectionSocket.close()
```

-TCP uses a SOCK_STREAM instead of SOCK_DGRAM (socket datagram)

-SNMP = Simple Network Management Protocol (uses UDP) - protocol used for monitoring a network

9/24 - Demultiplexing

Tuesday, September 24, 2024 4:42 PM

Transport Layer

-Goal of the transport layer: processes (sockets) talking to each other

-Refresher:
- -HTTP and FTP use TCP because it's reliable, we don't want to load half a large website or image
- -DNS uses UDP because it doesn't require reliability, we don't want to overload the server with overhead that TCP requires so we'd be better off just asking for another translation if it doesn't go through the first time because there's so many DNS requests hitting servers at one time

-Flow control means don't overflow the server with many requests vs. congestion control means make sure the router buffer isn't full and having data loss

-The transport layer exists (has the goal of) transporting messages between application processes running on different hosts (whereas the application layer is about transporting messages between client and server)
-Two transport layer protocols: TCP and UDP
- -They both work the same way, they both have the sender split applications into multiple *segments* which are passed to the network layer

-Network layer is communication between hosts vs. transport layer is communication between processes on those hosts

-Sender in transport layer:
- -Passes an application layer message
- -Determines *segment* header field values
  - -A "message" in the application layer (e.g. an HTTP message) becomes transport layer "segments"

-Receiver in transport layer:
- -Receives segment from IP (network layer)
- -Checks header values
- -Extracts application layer message
- -Demultiplexes message to application via a socket

-Difference between TCP and UDP

- **TCP:** Transmission Control Protocol
  - reliable, in-order delivery
  - congestion control
  - flow control
  - connection setup

UDP: User Datagram Protocol

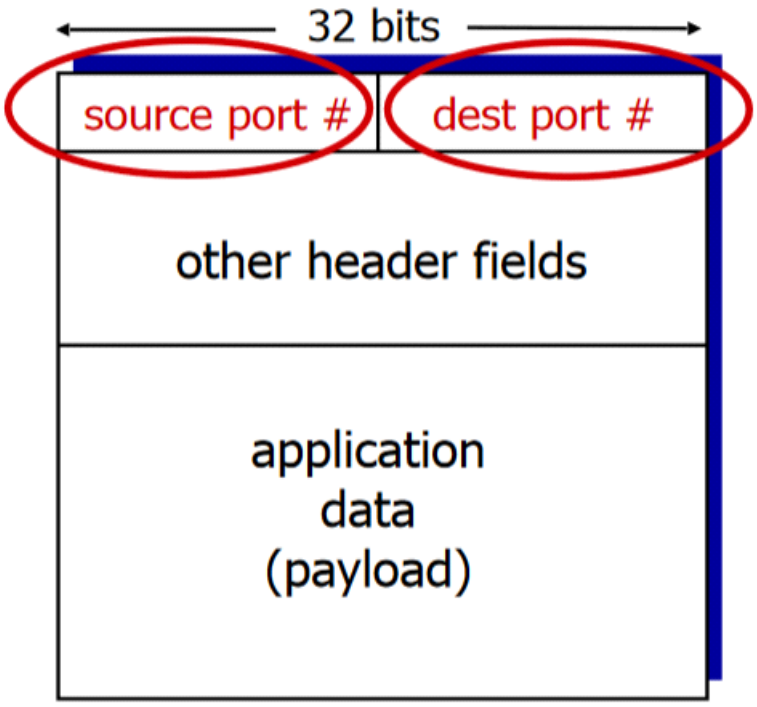
- **TCP:** Transmission Control Protocol
  - reliable, in-order delivery
  - congestion control
  - flow control
  - connection setup
- **UDP:** User Datagram Protocol
  - unreliable, unordered delivery
  - no-frills extension of "best-effort" IP
- services not available:
  - delay guarantees
  - bandwidth guarantees

-Multiplexing is how you know which process to send data to because you have the right IP address and port number

-Multiplex at the sender

-Demultiplex at the receiver

-How demultiplexing works:



TCP/UDP segment format

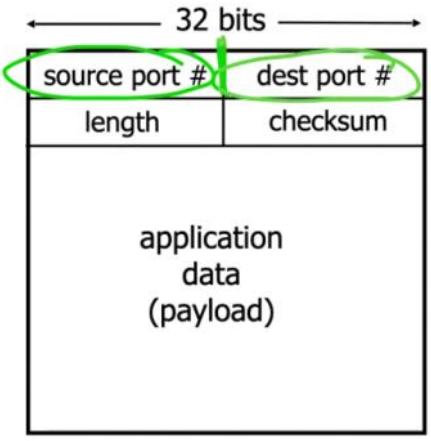
-TCP will have more fields of data for reliability and congestion control, UDP has less fields of data
-You need both the source port and destination port

-UDP demultiplexes using only destination port number
-TCP socket is identified by 4-tuple: source IP, source port number, destination IP address, destination port number

9/26 - UDP, RDT (ACKs)

Thursday, September 26, 2024 5:03 PM

-UDP = User Datagram Protocol
- -UDP is a "best effort" service because segments may be lost or sent out of order, in which case, you will resend UDP segments
- -UDP is connectionless, there is no pipe or handshaking, each UDP segment is handled independently
- -Why do we have UDP?
  - -UDP eliminates the overhead of establishing a connection
  - -UDP has less RTT delay
  - -UDP header is smaller than TCP
  - -UDP performs well in the face of congestion
- -UDP is used in the following application layer protocols:
  - -Streaming multimedia apps (loss is acceptable), includes gaming
  - -DNS (needs to happen fast)
  - -SNMP (Simple Network Management Protocol) - needs to be minimal to prevent overhead on the network
  - -HTTP/3 - used when traffic is congested
- -UDP is RFC 768, it's only 3 pages and is pretty easy
- -UDP segment header:



UDP segment format

- -Source port is 16-bits, destination port is 16-bits
- -You can have 2^16 (65k) number of source and destination ports, that's a lot!
- -Ports 0 to 1024 are "reserved" ports for common protocols (e.g. HTTP on 80)
- -Length is the length of the UDP segment including the headers
- -The checksum is used for data integrity on a transmitted UDP segment
  - -How the checksum works:

sender:

- treat contents of UDP segment (including UDP header fields and IP addresses) as **sequence of 16-bit integers**
- checksum: **addition** (**one's complement sum**) of segment content
- checksum value put into UDP checksum field

receiver:

- compute checksum of received segment
- check if computed checksum equals checksum field value:
  - **Not equal** - **error** detected
  - **Equal** - **no error** detected. ***But maybe errors*** *nonetheless?* More later ....

-NOTE: look at fields in the IP header to help you figure out the checksum of UDP for the extra credit in the UDP Wireshark lab
-Example of how to compute checksum:

Internet checksum: an example

example: add two 16-bit integers

	1	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0
	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
wraparound	1	1	0	1	1	1	0	1	1	1	0	1	1	1	0	1	1
sum	1	0	1	1	1	0	1	1	1	0	1	1	1	1	0	0
checksum	0	1	0	0	0	1	0	0	0	1	0	0	0	0	1	1

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

-If these bits flip, our checksum is compromised, this is why UDP is not very secure

-UDP summary:

Summary: UDP

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

-Reliable data transfer

-Reliable Data Transfer (RDT) protocol vs. Unreliable Data Transfer (UDT) protocol

-Use finite state machines (FSM) to explain how transfer works

-rdt1.0

-Assumes an unreliable channel is reliable (no bit flips or loss of packets caused by router queuing / congestion control)

-Must have separate FSM for sender and receiver

-rdt2.0

-Underlying channel may flip bits in packet, checksum to detect errors:

-Acknowledgement (ACKs) from receiver to tell sender that packet was received OK

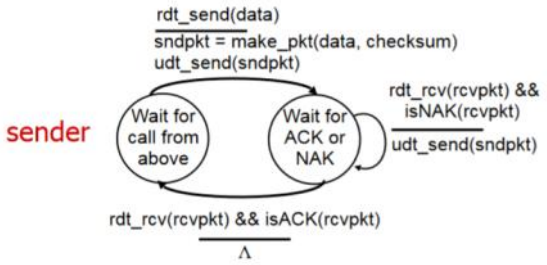
-Negative acknowledgement (NAKs) from receiver to tell sender that packet had errors

-Sender retransmit packets on receipt of NAK

-Stop and wait - sender sends one packet, then waits for receiver response

-FSM for rdt2.0:

rdt2.0: FSM specifications



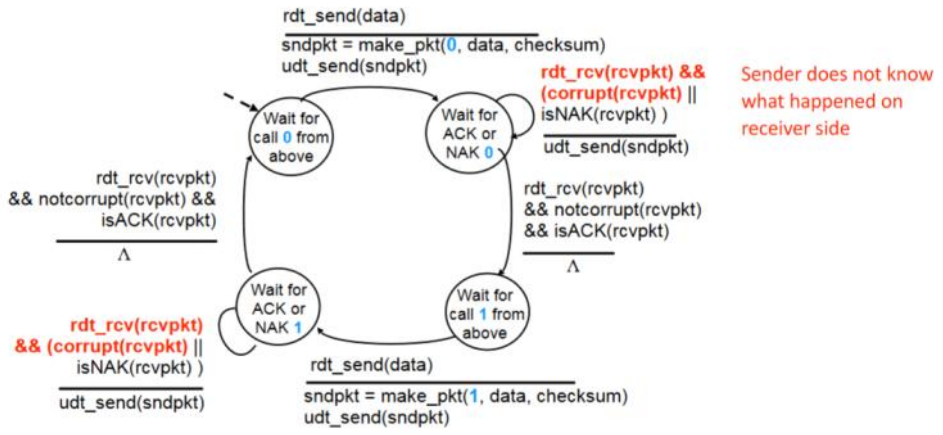
-rdt2.1

-What rdt2.1 adds:

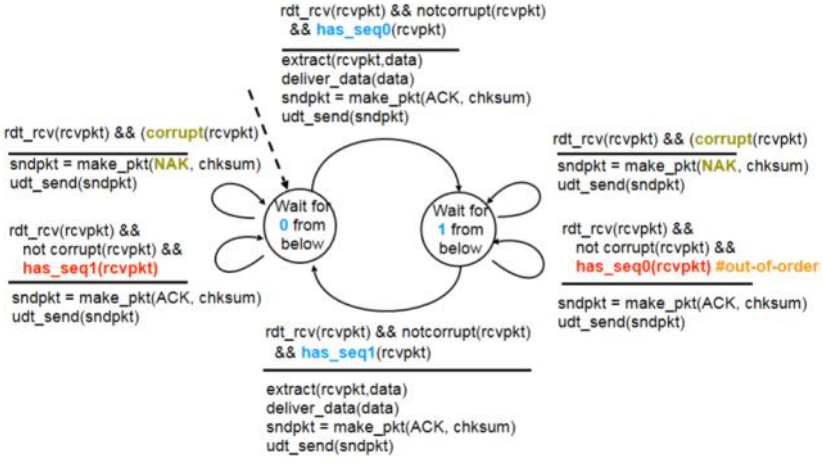
-rdt2.0 has the issue - if your ACK or NAK is corrupted, the sender won't know what happened to the receiver, can't just resubmit if there's possible duplicates - we fix this in rdt2.1 by tagging each packet with some sequence number to handle duplicates so the receiver can discard duplicates with stop and send (sender sends one packet then waits for receiver response)

-FSM for rdt2.1:

rdt2.1: sender, handling garbled ACK/NAKs



rdt2.1: receiver, handling garbled ACK/NAKs



10/3 - More RDT (ACKs)

Tuesday, October 1, 2024 1:56 PM

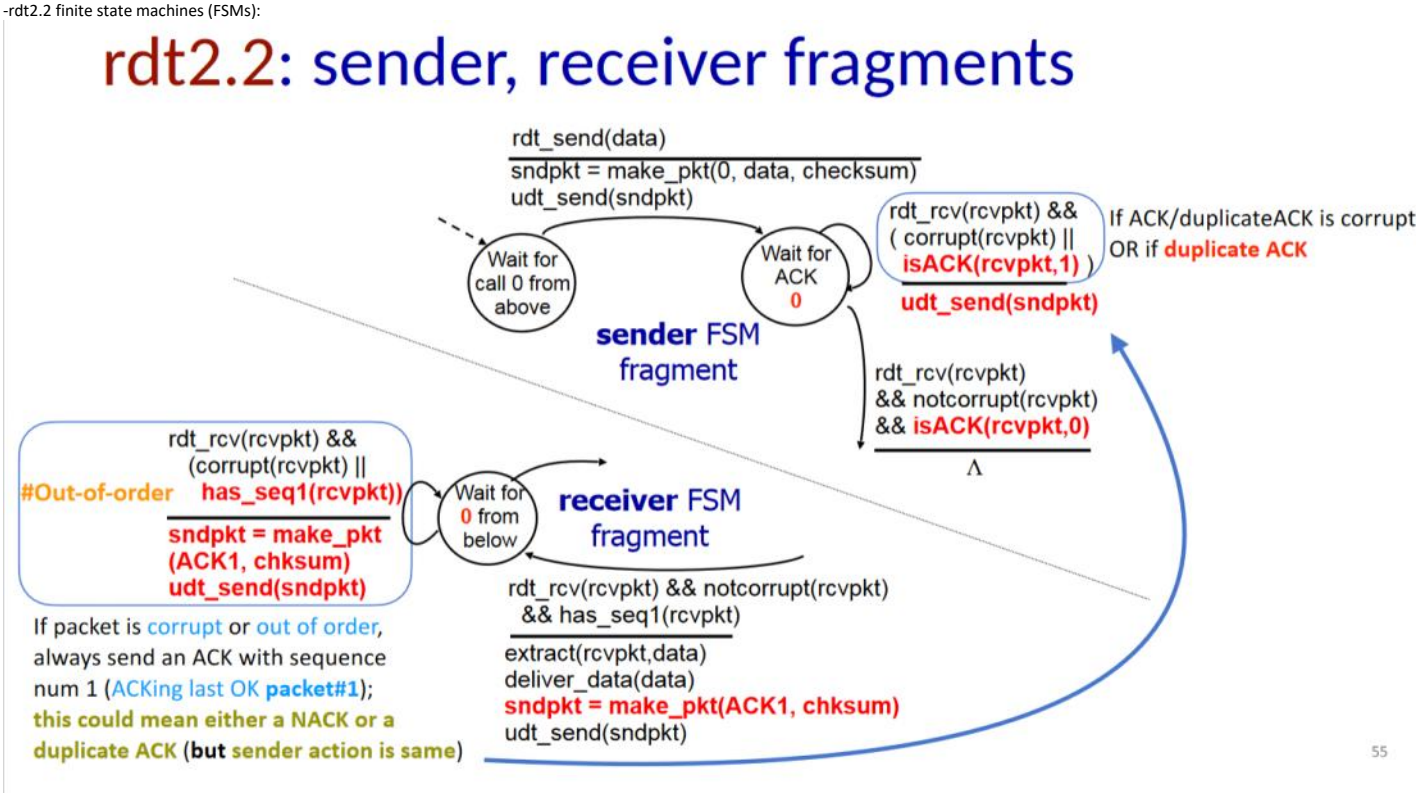
-rdt2.2

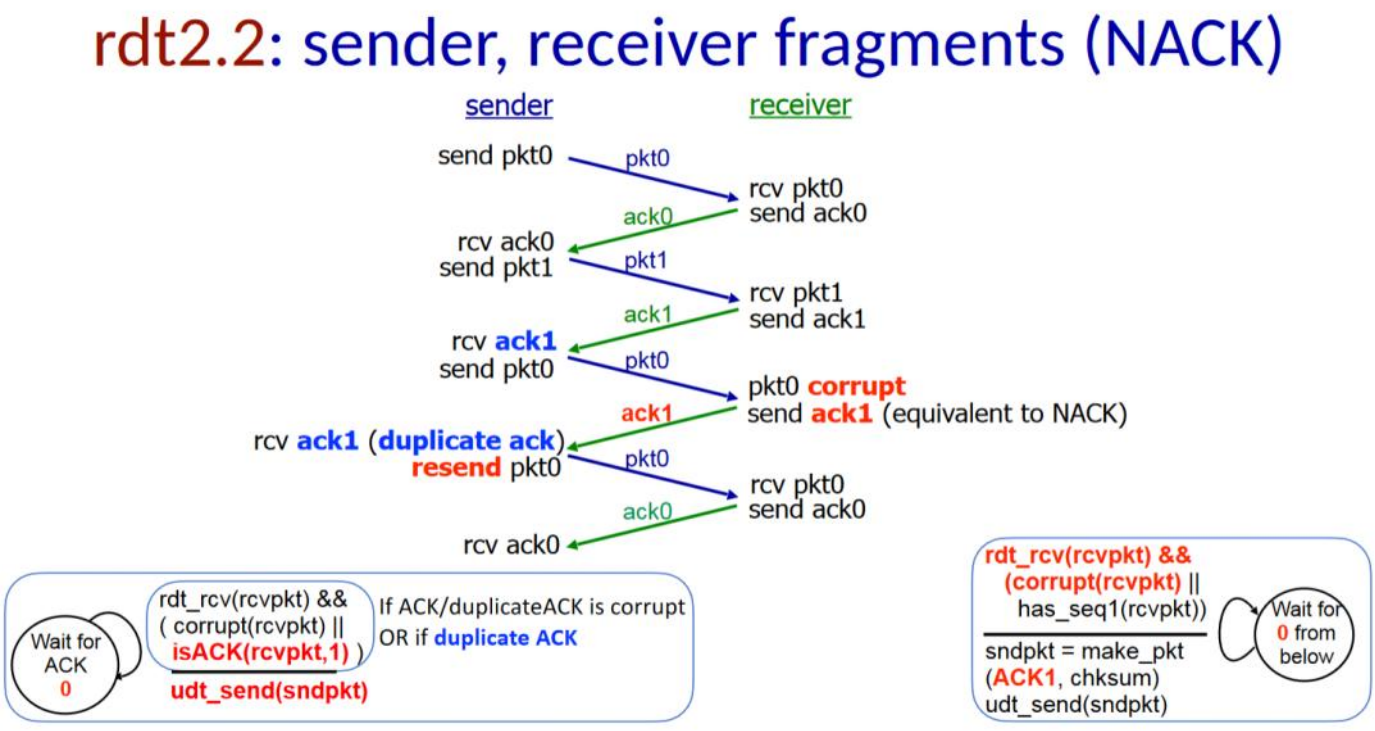
-rdt2.2 has the same functionality as rdt2.1 except using ACKs only, so instead of NAK, the receiver sends ACK for the last packet received OK where the receiver must explicitly include the sequence number of the packet being ACKed, also, a duplicate ACK at the sender results in the same action as NAK which is a retransmit of the current packet

-A corrupted ACK would also result in a retransmit

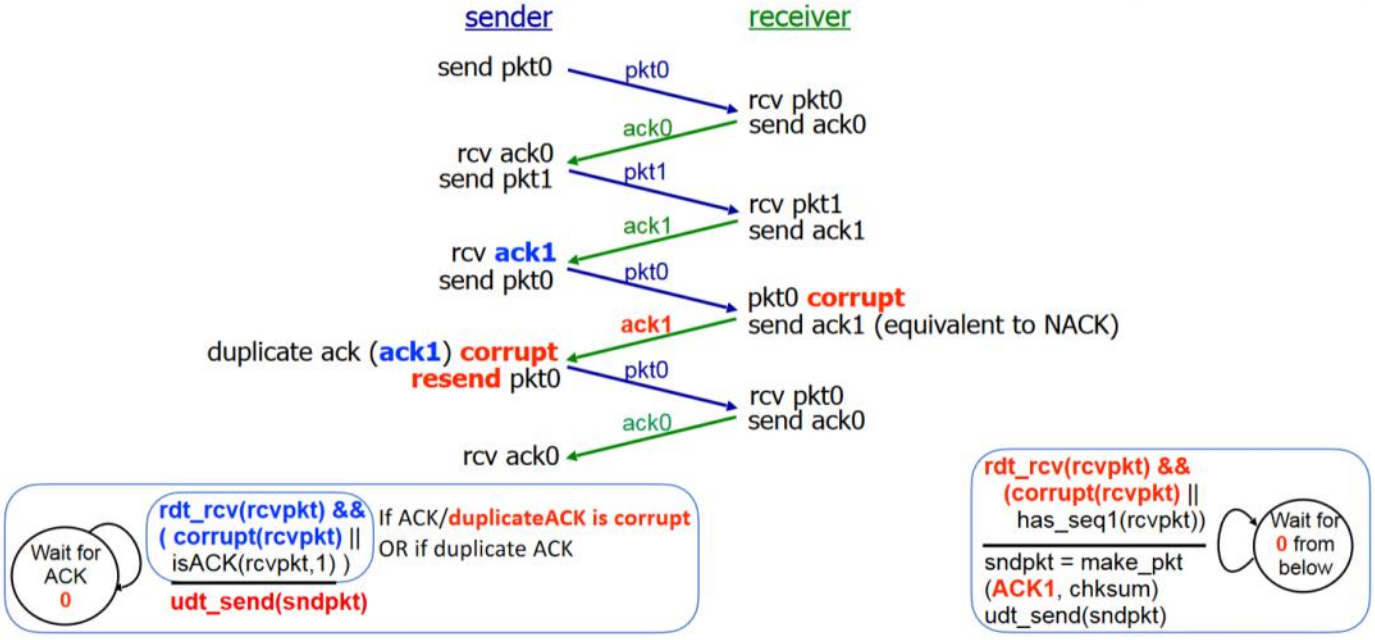
-TCP uses this approach to be NAK-free

-rdt2.2 finite state machines (FSMs):

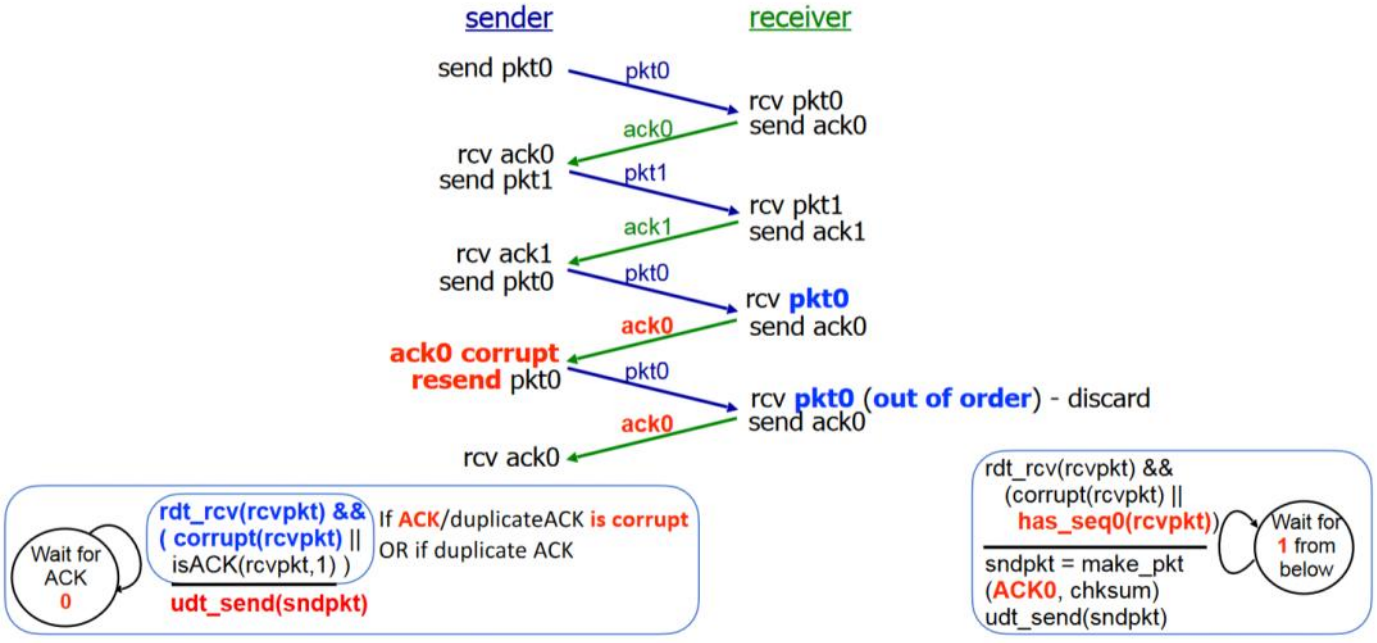




rdt2.2: sender, receiver fragments (Corrupt NACK)



rdt2.2: sender, receiver fragments (Corrupt ACK)



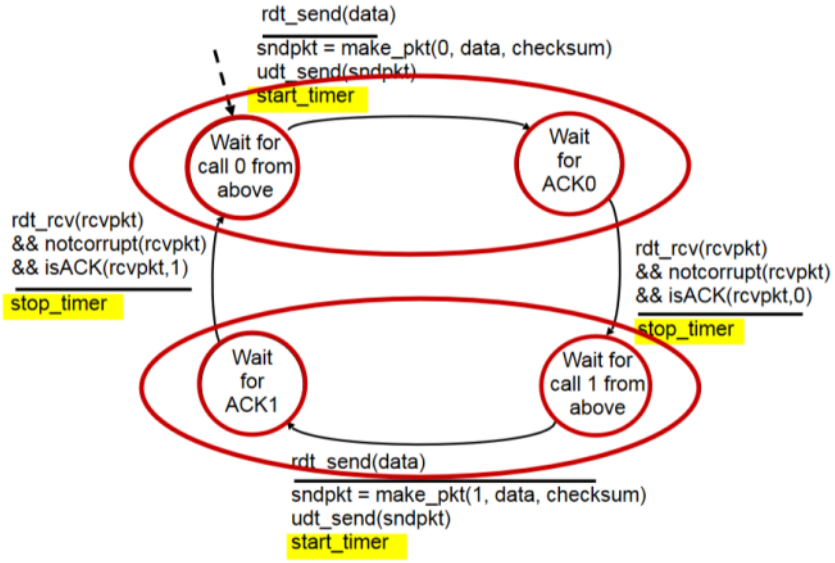
-rdt3.0

-We typically have packet loss due to the queue on the router overfilling (congestion control), but with rdt3.0, the sender waits a "reasonable" amount of time for an ACK then retransmits if no ACK is received in time ("timing out"), so, if a packet (or ACK) is just delayed (and not lost), then retransmission will be duplicate but the sequence numbers already handle this which means the receiver must specify the sequence number of packet being ACKed; we use a countdown timer to interrupt after a "reasonable" amount of time

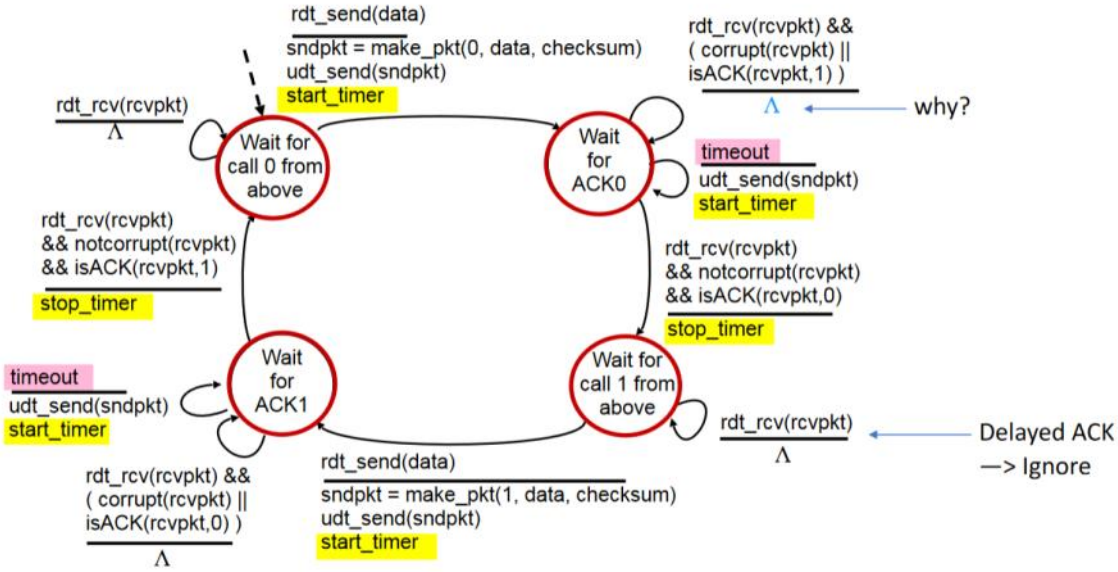
-The timeout is variable/dynamic, it will increase relative to the network characteristics (so if we receive a timed-out ACK, then we'd probably increase the time-out time) - we'll discuss this more and how to calculate this in TCP

-The finite state machines representing rdt3.0:

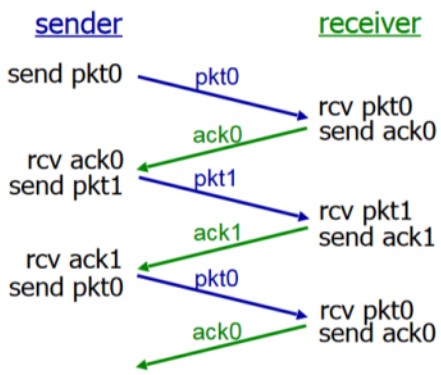
rdt3.0 sender



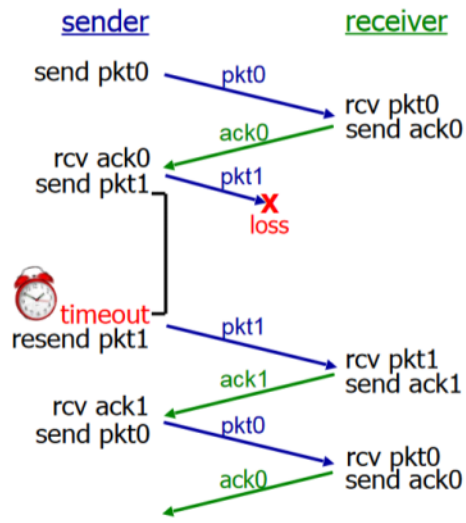
rdt3.0 sender



rdt3.0 in action

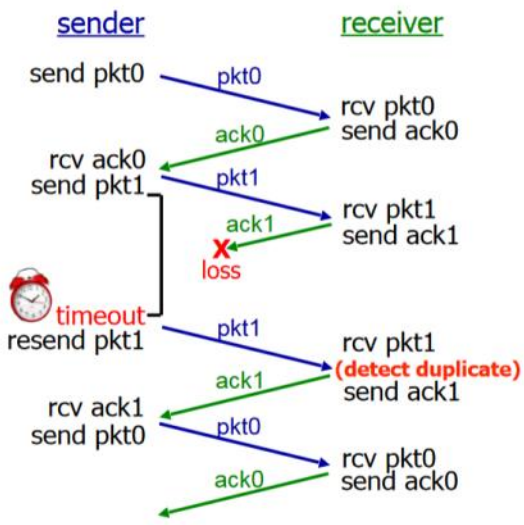


(a) no loss

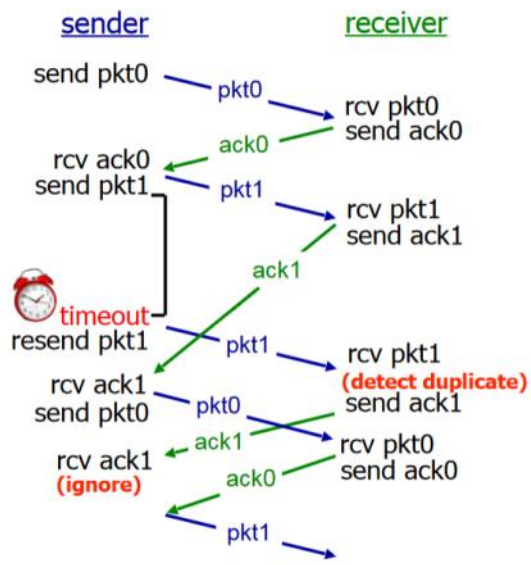


(b) packet loss

rdt3.0 in action



(c) ACK loss



(d) premature timeout/ delayed ACK

64

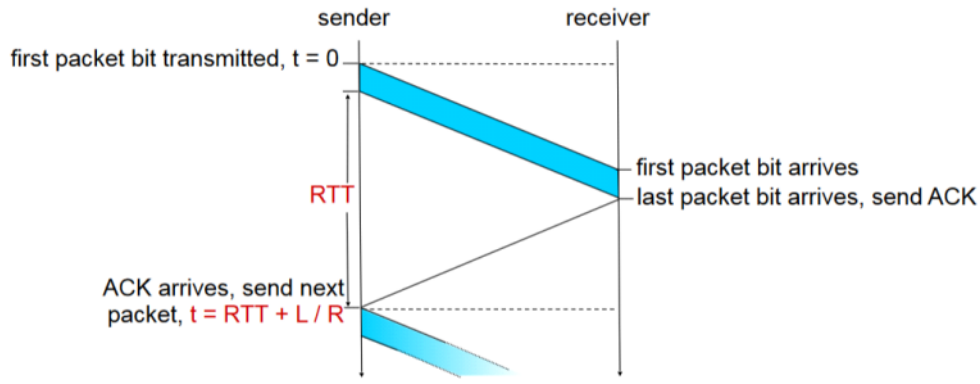
-The performance of rdt3.0 stop-and-wait (sender sends one packet, then waits for receiver response) can be calculated as the following, also including an example:

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

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

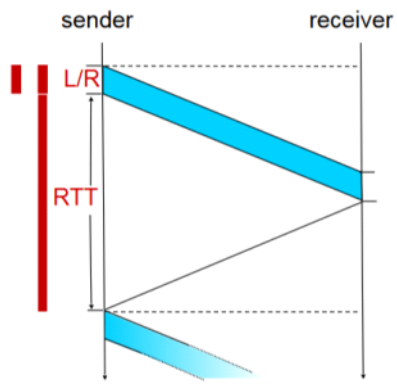
$$D_{trans} = \frac{L}{R} = \frac{8000\ bits}{10^9\ bits/sec} = 8\ microsecs$$

rdt3.0: stop-and-wait operation



rdt3.0: stop-and-wait operation

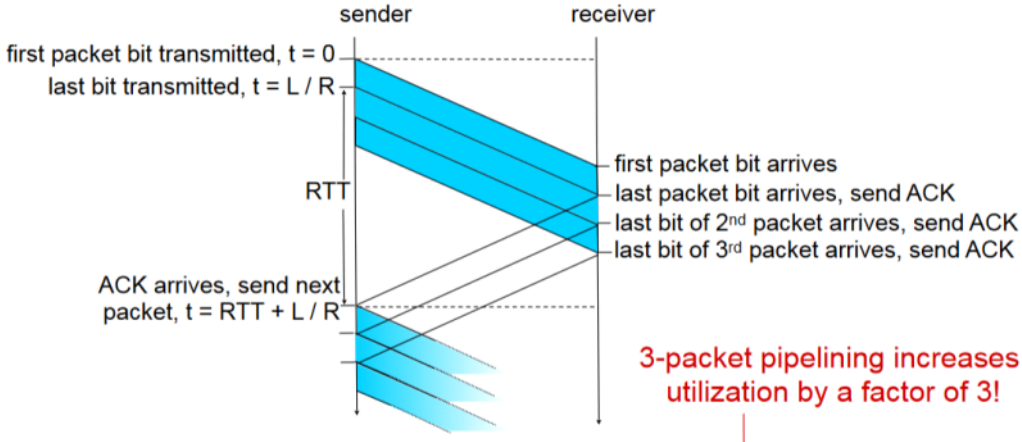
$$U_{sender} = \frac{L / R}{RTT + L / R} = \frac{.008}{30.008} = 0.00027$$



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

-NOTE: rdt3.0 performance sucks because we're waiting too long for ACKs
-rdt3.0 pipelining fixes this performance issue - pipelining is where sender allows multiple, "in-flight", yet-to-be-acknowledged packets where the range of sequence numbers must be increased and there's buffering at the sender and/or receiver
-How to calculate rdt3.0 pipelining performance and example:

Pipelining: increased utilization



3-packet pipelining increases utilization by a factor of 3!

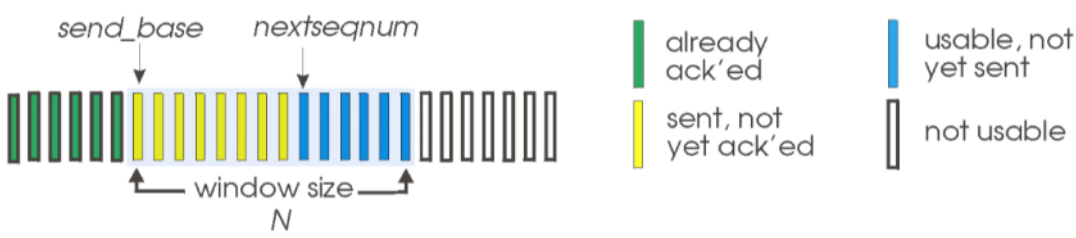
$$U_{sender} = \frac{3L/R}{RTT + L/R} = \frac{.0024}{30.008} = 0.00081$$

-Cumulative ACK - all the packets are acknowledged together, one ACK could acknowledge a hundred packets, doesn't matter if all the packets were dropped or lost, as long as we receive one ACK that acknowledges all the previous packets (TCP uses this)

-This uses a sliding window:

Go-Back-N: sender

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



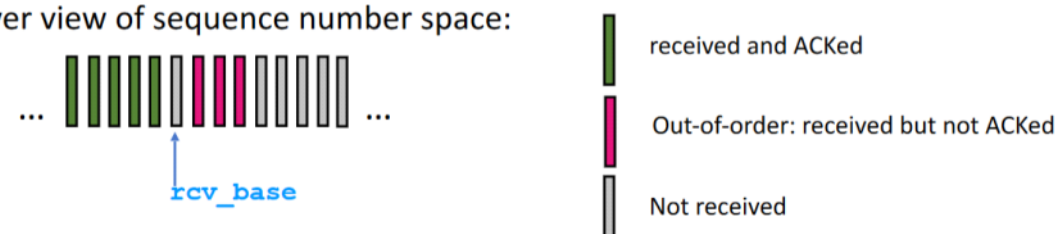
- *cumulative ACK:* ACK(*n*): ACKs all packets up to, including seq # *n*
  - on receiving ACK(*n*): move window forward to begin at *n+1*
- timer for oldest in-flight packet
- *timeout(n):* **retransmit packet n and all higher seq # packets** in **window**

70

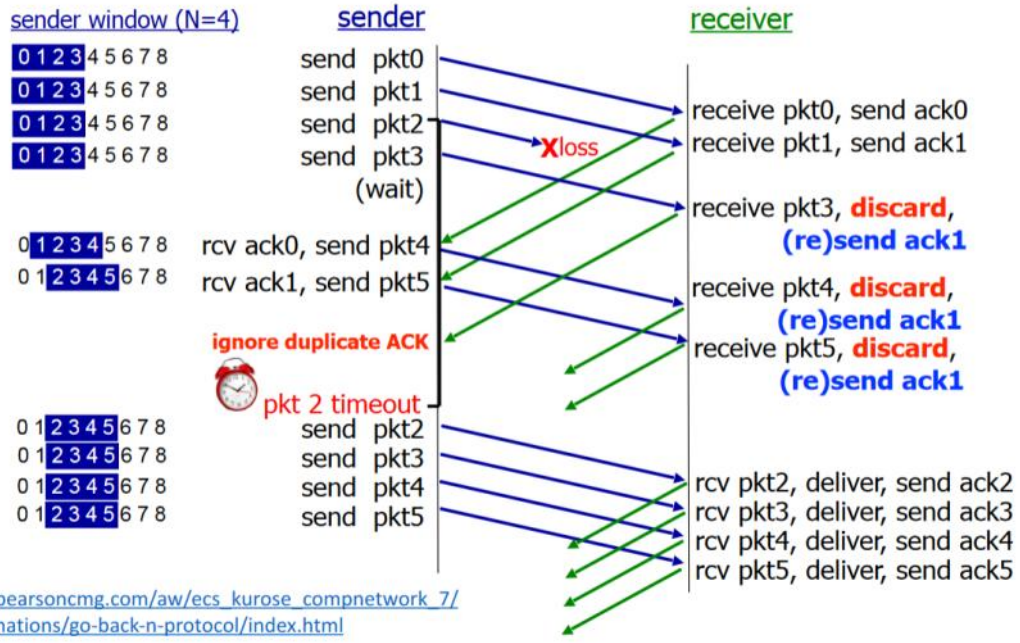
Go-Back-N: receiver

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

Receiver view of sequence number space:



Go-Back-N in action



Interactive: https://media.pearsoncmg.com/aw/ecs_kurose_compnetwork_7/cw/content/interactiveanimations/go-back-n-protocol/index.html

72

10/8 - More ACKs, TCP

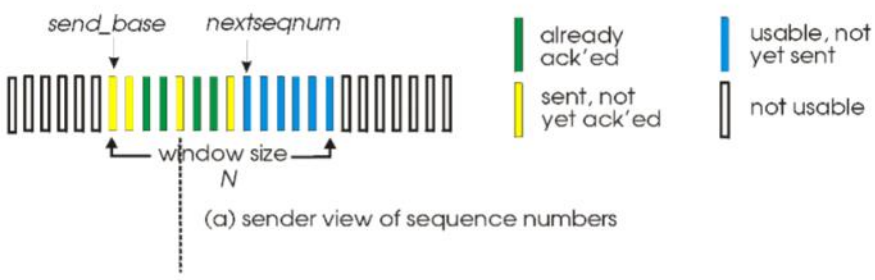
Tuesday, October 8, 2024 5:03 PM

-Selective repeat

Selective repeat

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

Selective repeat: sender, receiver windows



(a) sender view of sequence numbers

Selective repeat: sender and receiver

sender

data from above:

- if next available seq # in window, send packet

timeout(*n*):

- resend **packet *n***, restart timer

If ACK(*n*) in [sendbase,sendbase+N]:

- mark **packet *n*** as received
- if **n** was **smallest unACKed packet**, advance window base **to next unACKed seq #**

receiver

packet *n* in [rcvbase, rcvbase+N-1]

- send ACK(*n*)
- **out-of-order: buffer**
- in-order: deliver (also deliver buffered, in-order packets), advance window to next not-yet-received packet

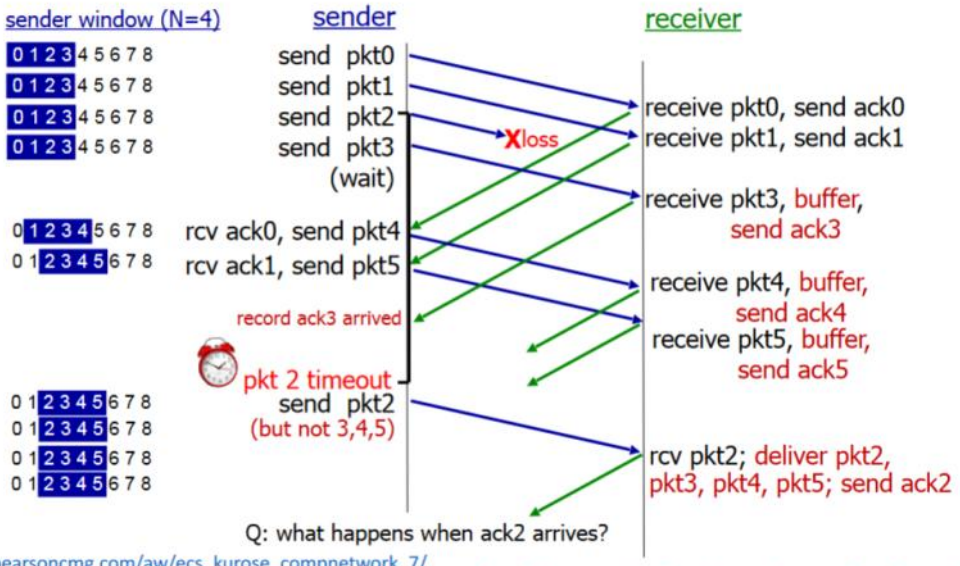
packet *n* in [rcvbase-N,rcvbase-1]

- ACK(*n*)

otherwise:

- ignore

Selective Repeat in action



Interactive: https://media.pearsoncmg.com/aw/ecs_kurose_compnetwork_7/cw/content/interactiveanimations/selective-repeat-protocol/index.html

A: Advance window to next un-acked seq#

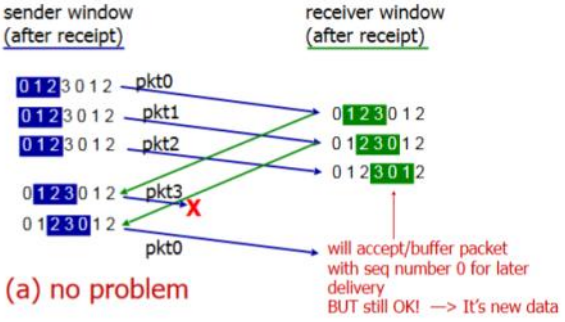
Selective repeat: a dilemma!

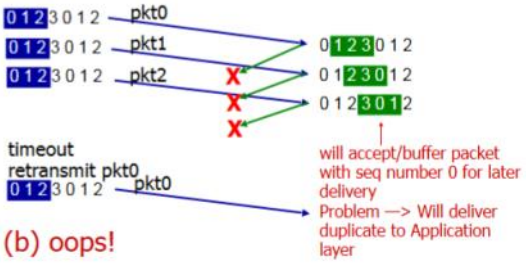
example:

- seq #s: 0, 1, 2, 3 (base 4 counting)
  - **Finite range of sequence numbers**
- window size=3

(a) Accepted pkt0 contains new data (OK)

(b) Accepted pkt0 is a retransmit (problem)





77

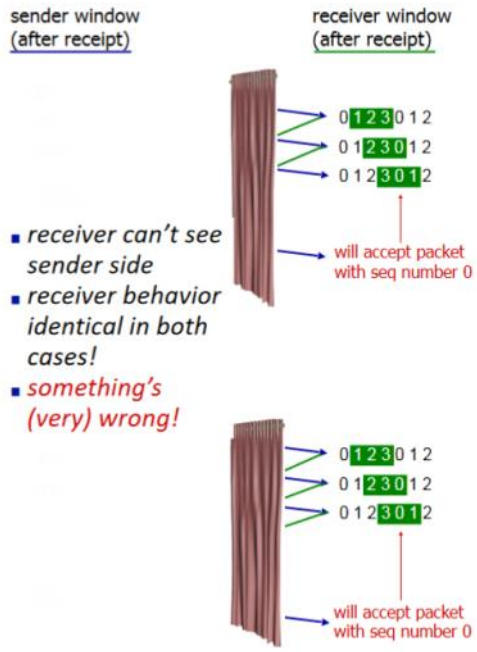
Selective repeat: a dilemma!

example:

- seq #s: 0, 1, 2, 3 (base 4 counting)
- window size=3
- Window size = seq num space - 1

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

window size <= 1/2 seq num space for SR protocols



78

-The window size in TCP is big, it's half of 2^64

-TCP - Transmission Control Protocol

-One of the most important protocols on the Internet besides IP, it's also one of the most complex protocols so it can be overwhelming

-TCP has 5 RFC documents, each is a few dozen pages, the congestion control one is over a hundred pages

-TCP is point-to-point - there's one sender and one receiver that send each other messages

-TCP is a reliable, in-order byte stream - received in order, byte-by-byte, all on the transport layer (sent in the correct order)

-We identify units of TCP by bytes, not by packets

-Bi-directional data flow - sender and receiver transfer data back and forth

-MSS = maximum segment size in TCP, we'll talk about why later

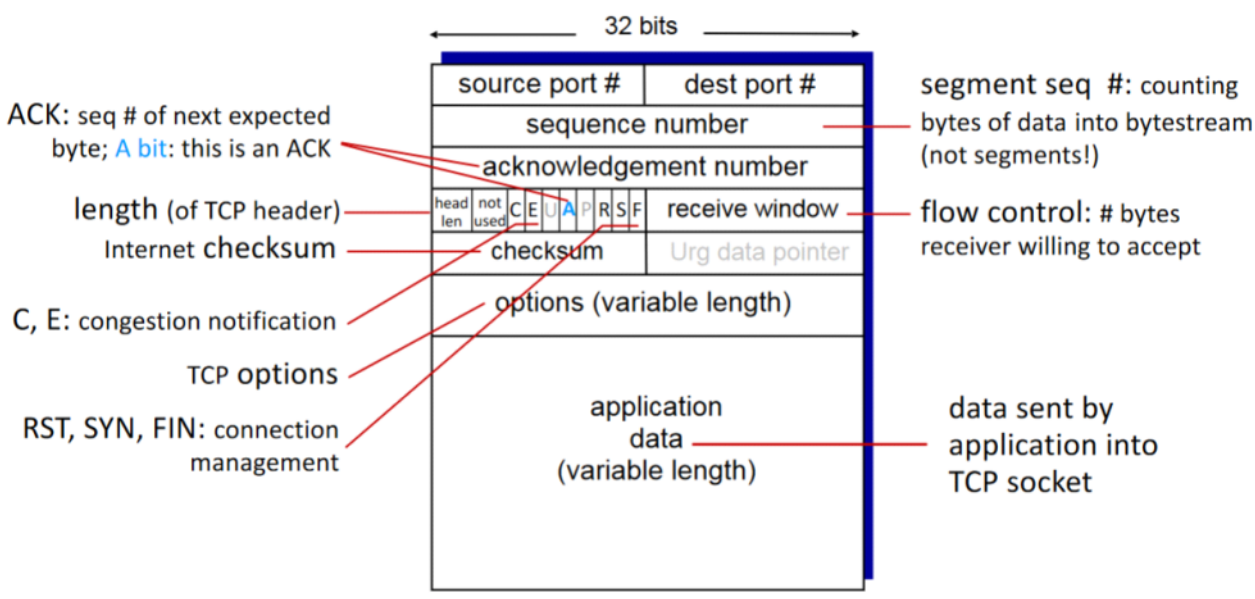
-Pipelining for TCP congestion and flow control dealing with window size

-Connection-oriented, uses handshaking

-Flow controlled - sender will not overwhelm receiver

-TCP segment structure:

TCP segment structure



-32 bits = 4 bytes

-Compare TCP above with UDP, which only had 4 fields: sort port number, destination port number, checksum, and length

-Total possible number of sequence numbers = 2^32 - 1

-The A-bit is an ACK bit, determines if our TCP segment is an ACK or not

-Internet checksum for TCP is pretty similar to the UDP checksum (with how it's calculated)
-The length of the TCP header - we include this because unlike UDP, our header size is variable since we have an "options" field of variable length
    -The length field in UDP included the header and content, not just the header
    -The header is only 4 bits
-A typical TCP header is 20 bytes long
-Flags used for connection management: R, S, and F or RST, SYN, and FIN
-Receive window - tells the sender (via an ACK) how to handle flow control
-Other flags: C and E used for congestion notification
-Lesser used field: the urgent data pointer field which also uses the U and P flags - this is all for urgent data delivery

-TCP sequence number:

TCP sequence numbers, ACKs

Sequence numbers:

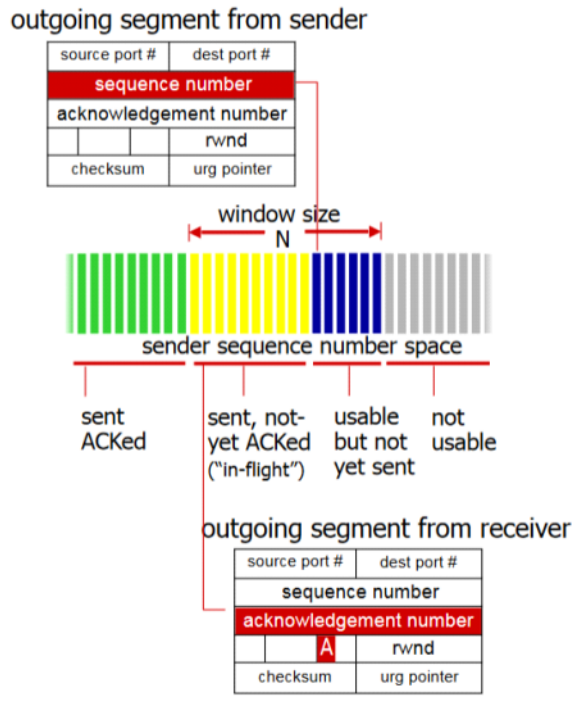
- byte stream "number" of first byte in segment's data

Acknowledgements:

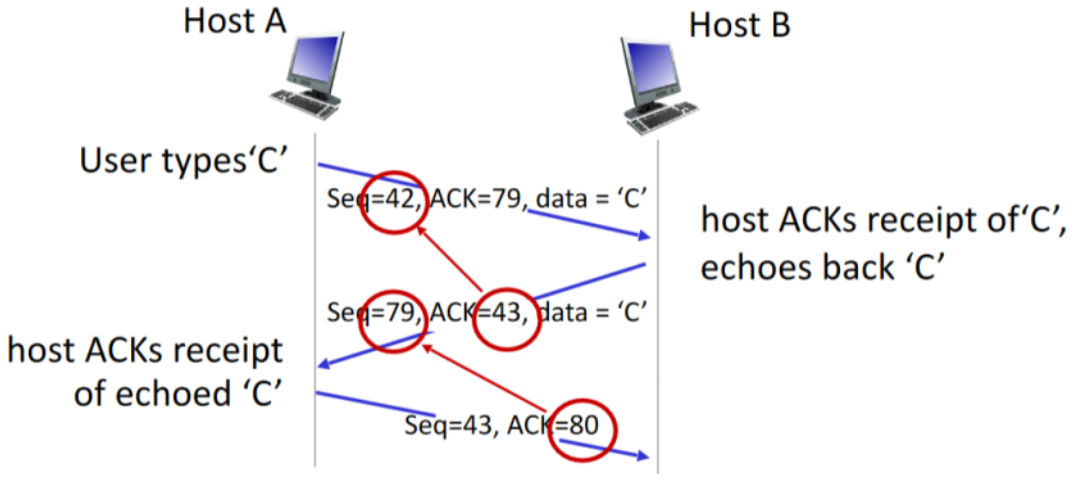
- seq # of next byte expected from other side
- cumulative ACK

Q: how receiver handles out-of-order segments

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



TCP sequence numbers, ACKs



simple telnet scenario

-Telnet - application layer protocol that runs on top of TCP, this was SSH before SSH, Telnet was not very secure and sent username and password unencrypted

-TCP round trip time (RTT) and timeouts, how it works and how to calculate it:

TCP round trip time, timeout

Q: how to set TCP timeout value?

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

Q: how to estimate RTT?

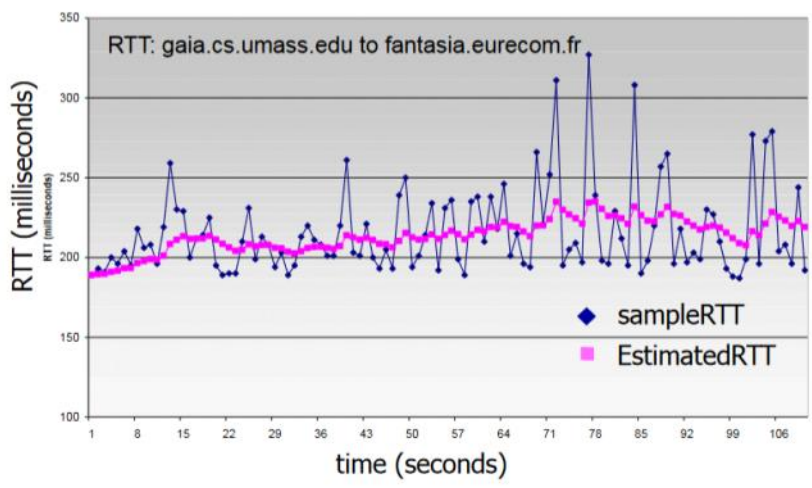
- `SampleRTT`: measured time from segment transmission until ACK receipt
  - ignore retransmissions
- `SampleRTT` will vary, want estimated RTT "smoother"
  - average several *recent* measurements, not just current `SampleRTT`

-If the network is doing well, the timeout will be smaller and vice versa

TCP round trip time, timeout

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

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



85

TCP round trip time, timeout

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

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

estimated RTT "safety margin"

- **DevRTT**: EWMA of **SampleRTT** deviation from **EstimatedRTT**:

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

(typically, $\beta = 0.25$)

* Check out the online interactive exercises for more examples: http://gaia.cs.umass.edu/kurose_ross/interactive/

-The timeout is the estimate RTT (important to know how this is calculated) but we include a "safety margin" (like a standard deviation)

10/10 - More TCP

Thursday, October 10, 2024 5:01 PM

-TCP sender and receiver

TCP Sender (simplified)

event: data received from application

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

event: timeout

- retransmit segment that caused timeout
- restart timer

event: ACK received

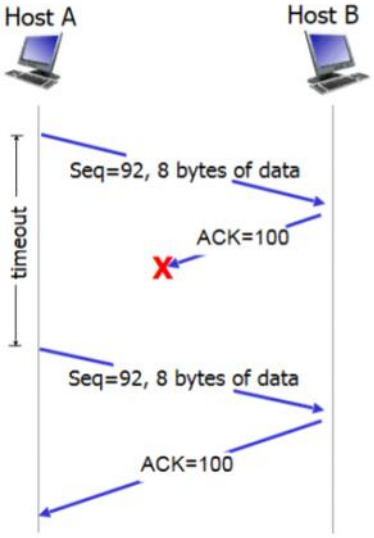
- if ACK acknowledges previously unACKed segments
  - update what is known to be ACKed
  - start timer if there are still unACKed segments

TCP Receiver: ACK generation [RFC 5681]

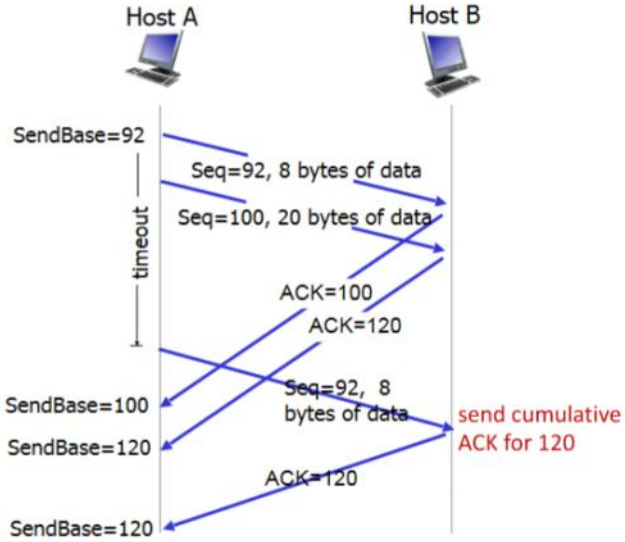
Event at receiver	*TCP receiver action*
arrival of in-order segment with expected seq #. All data up to expected seq # already ACKed	delayed ACK. Wait up to 500ms for next segment. If no next segment, send ACK (**reduce amount of unneed ACKs**)
arrival of in-order segment with expected seq #. One other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

-More on retransmission in TCP

TCP: retransmission scenarios

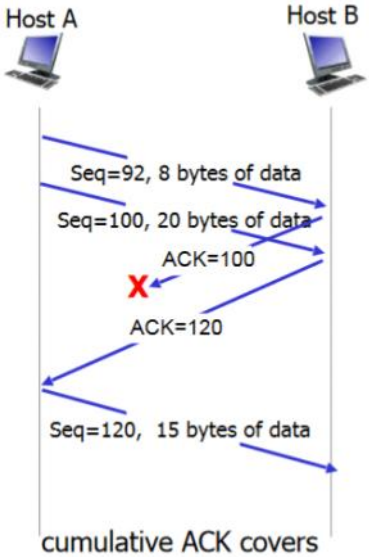


lost ACK scenario



premature timeout

TCP: retransmission scenarios



cumulative ACK covers for earlier lost ACK

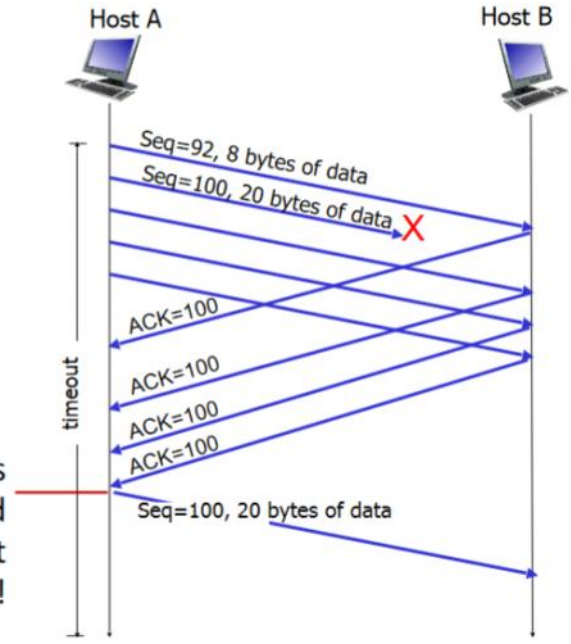
TCP fast retransmit

TCP fast retransmit

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

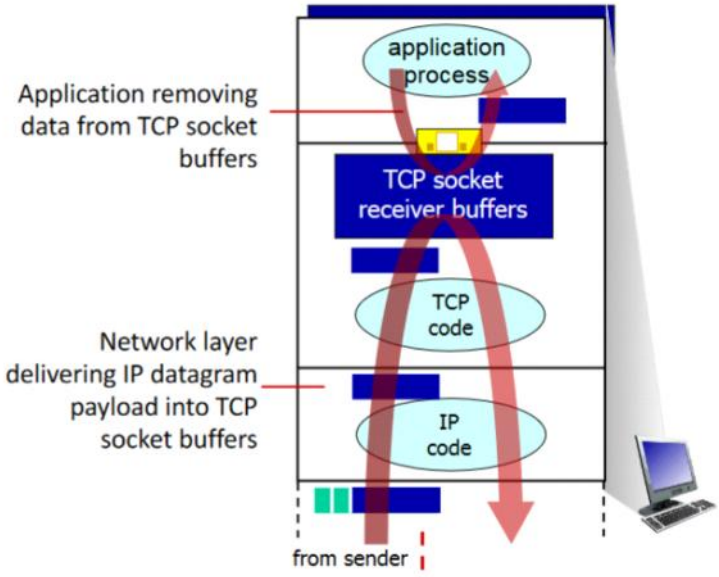
- likely that unACKed segment lost, so don't wait for timeout

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



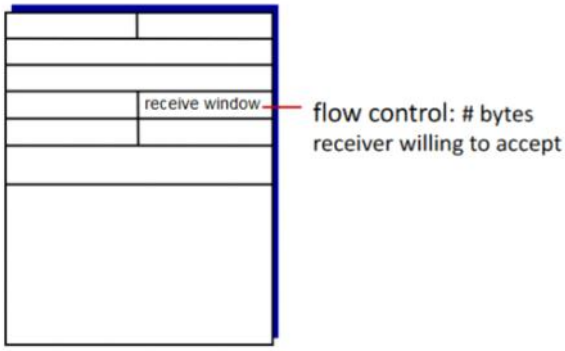
-If you receive three duplicate acknowledgements, that means the packet has definitely been lost somehow

-Flow control - don't overflow the server buffer with too many requests



receiver protocol stack

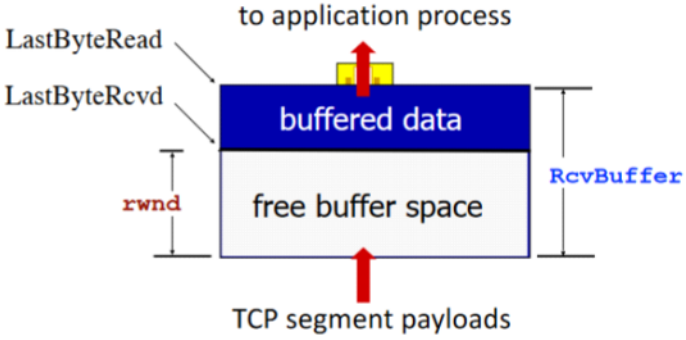
-Receive window - number of bytes the receiver is willing to accept, used in flow control (this is on TCP segment)



-How to calculate receive window (rwnd)

TCP flow control

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



TCP receiver-side buffering

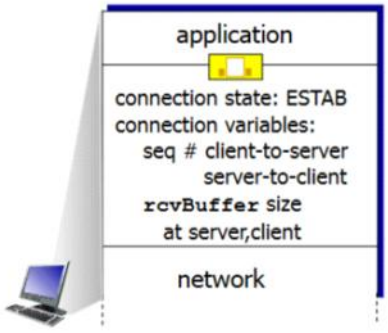
$$rwnd = RcvBuffer - (LastByteRcvd - LastByteRead)$$

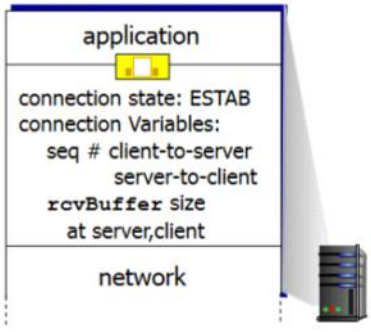
-TCP three-way handshake permits us to create a pipe - SYN, SYN-ACK, and ACK

TCP connection management

before exchanging data, sender/receiver "handshake":

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





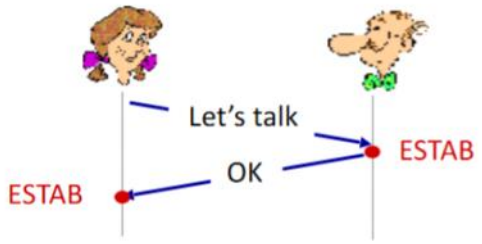

```
Socket clientSocket =
  newSocket("hostname","port number");
```

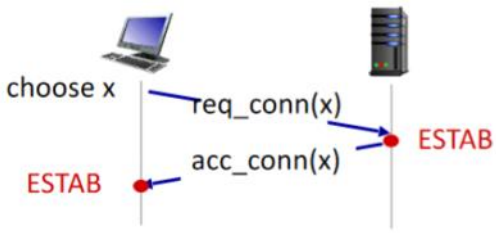
```
Socket connectionSocket =
  welcomeSocket.accept();
```

-Why not a 2-way handshake? (Why it doesn't work)

Agreeing to establish a connection

2-way handshake:

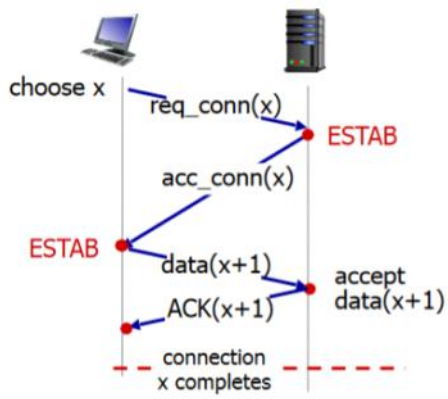




Q: will 2-way handshake always work in network?

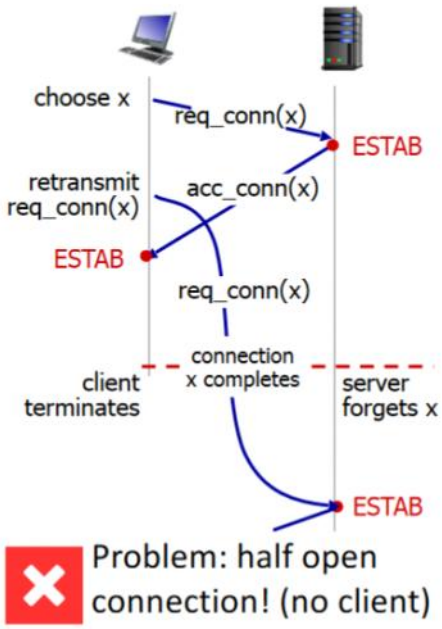
- variable delays
- retransmitted messages (e.g. req_conn(x)) due to message loss
- message reordering
- can't "see" other side

2-way handshake scenarios



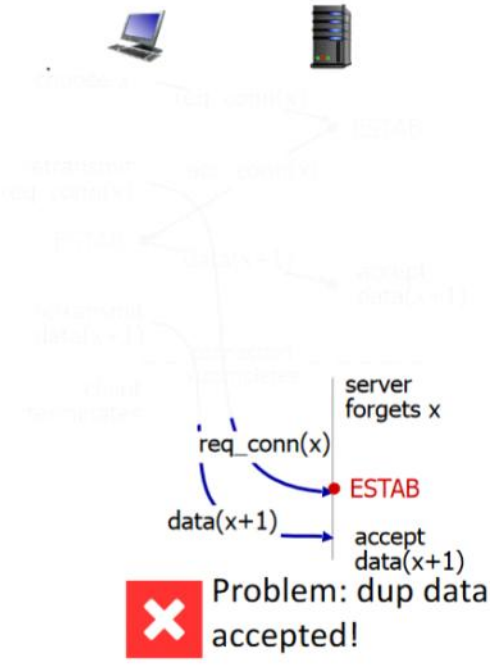
No problem!

2-way handshake scenarios



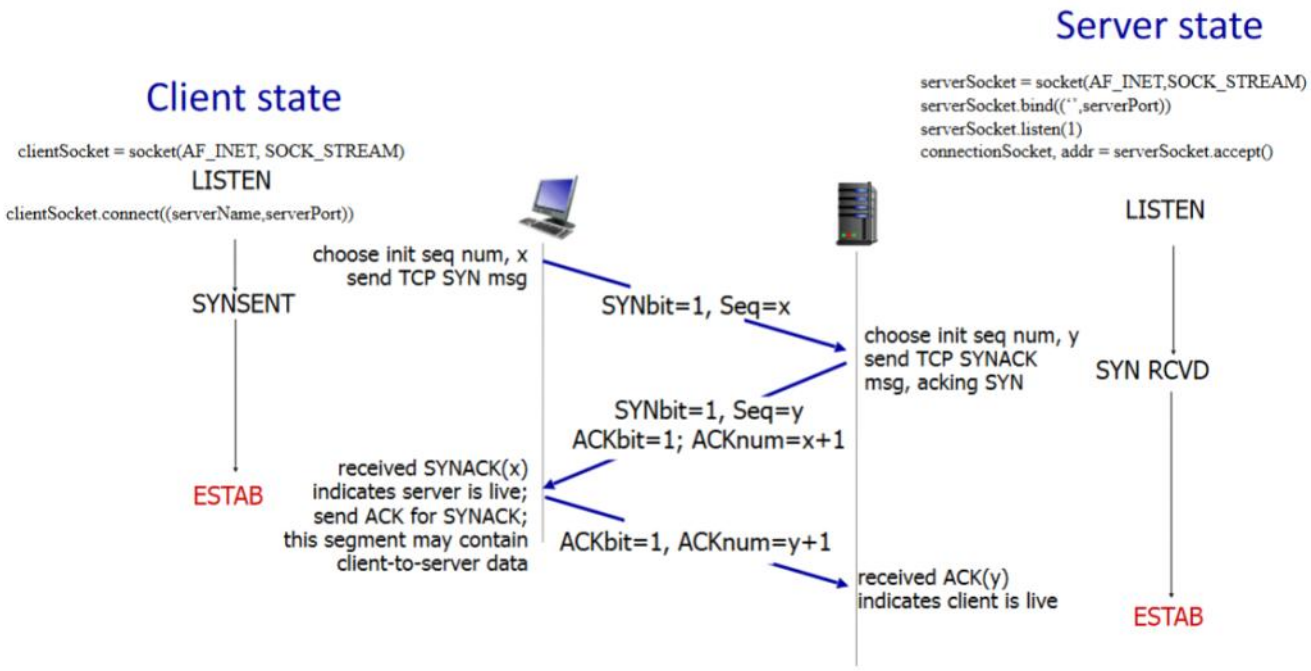
-"Half open connection" is the reason 2-way handshake doesn't work

2-way handshake scenarios



-How Three-Way Handshake works:

TCP 3-way handshake



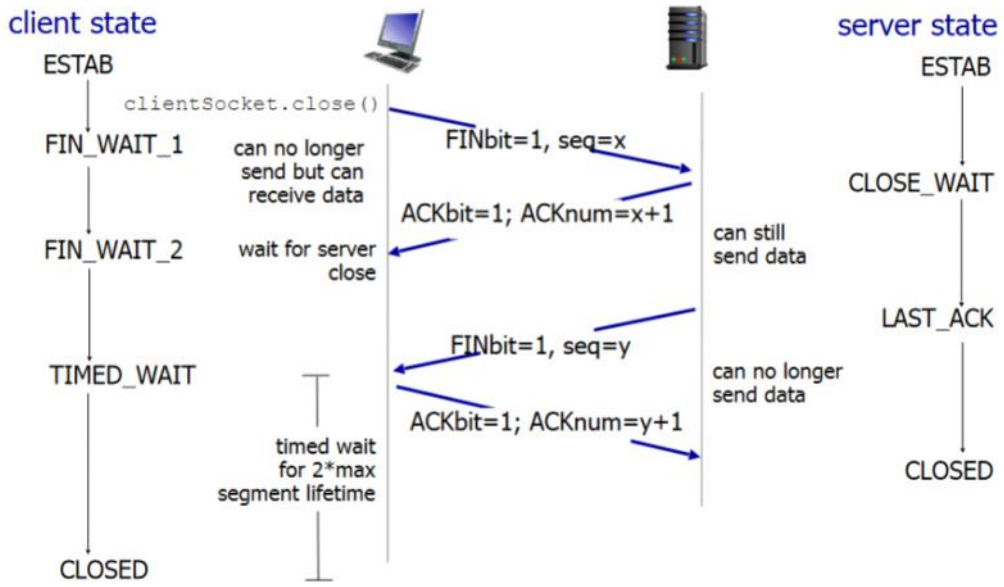
104

-Closing a TCP connection - setting the FIN bit to 1 on the TCP segment

Closing a TCP connection

- client, server each close their side of connection
  - send TCP segment with FIN bit = 1
- respond to received FIN with ACK
  - on receiving FIN, ACK can be combined with own FIN
- simultaneous FIN exchanges can be handled

Closing a TCP connection



-Timed wait being 2*max segment lifetime is because of sending ACKs over

10/15 - Congestion Control

Tuesday, October 15, 2024 5:06 PM

-Refresher:

-Flow control means don't overflow the <u>server</u> with many requests vs. congestion control means make sure the <u>router</u> buffer isn't full and having data loss

-The bigger the queue that fills up in the router, the longer the delay

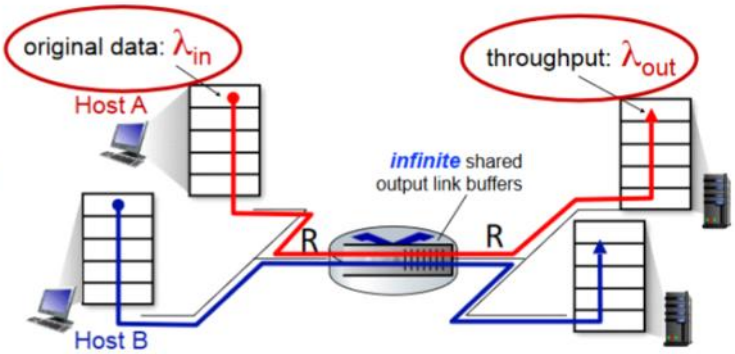
-Congestion control scenarios:

-One router with infinite buffers:

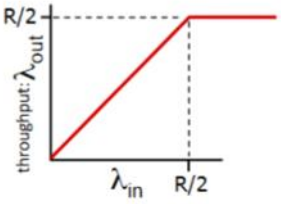
Causes/costs of congestion: scenario 1

Simplest scenario:

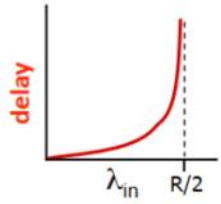
- one router, infinite buffers
- input, output link capacity: R
- two flows
- no retransmissions needed



Q: What happens as arrival rate λ_{in} approaches R/2?



maximum per-connection throughput: R/2

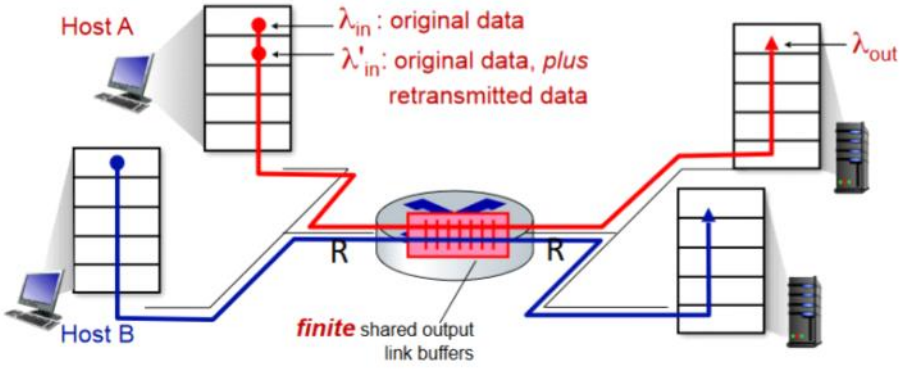


large delays as arrival rate λ_{in} approaches capacity

-One router with finite buffers:

Causes/costs of congestion: scenario 2

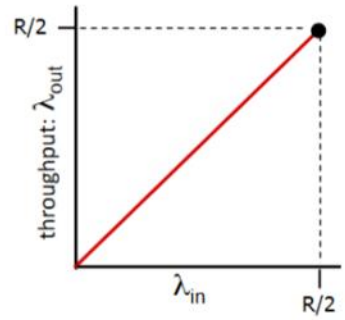
- one router, ***finite*** buffers
- sender retransmits lost, timed-out packet
  - application-layer input = application-layer output: $\lambda_{in} = \lambda_{out}$
  - transport-layer input includes *retransmissions* : $\lambda'_{in} \geq \lambda_{in}$

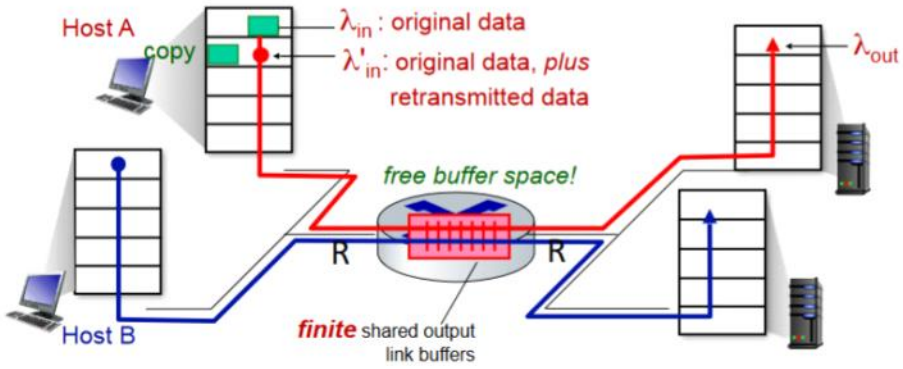


Causes/costs of congestion: scenario 2

Idealization: perfect knowledge

- sender sends only when router buffers available

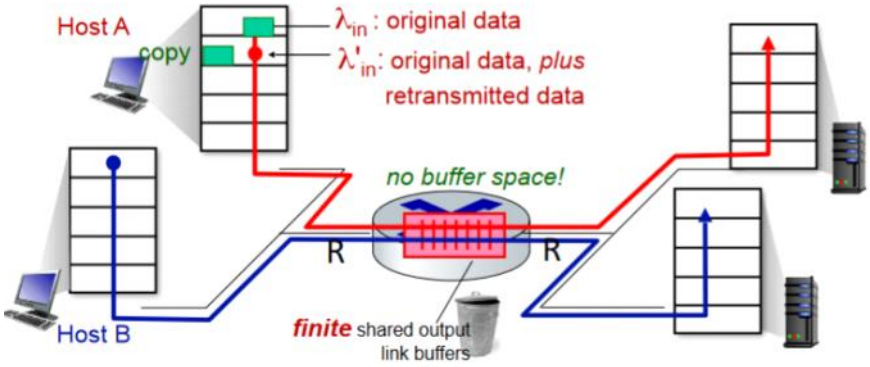




Causes/costs of congestion: scenario 2

Idealization: *some* perfect knowledge

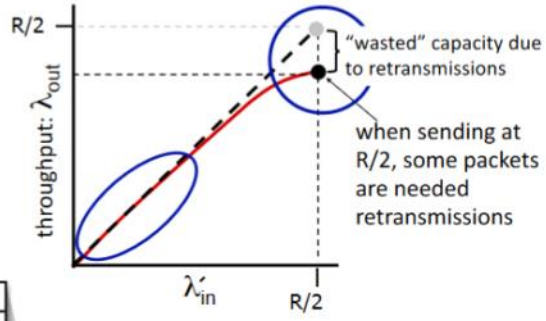
- packets can be lost (dropped at router) due to full buffers
- sender knows when packet has been dropped: only resends if packet *known* to be lost

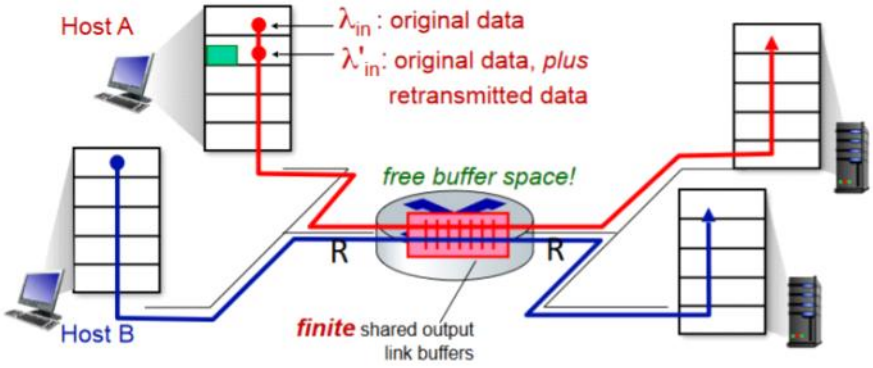


Causes/costs of congestion: scenario 2

Idealization: *some* perfect knowledge

- packets can be lost (dropped at router) due to full buffers
- sender knows when packet has been dropped: only resends if packet *known* to be lost

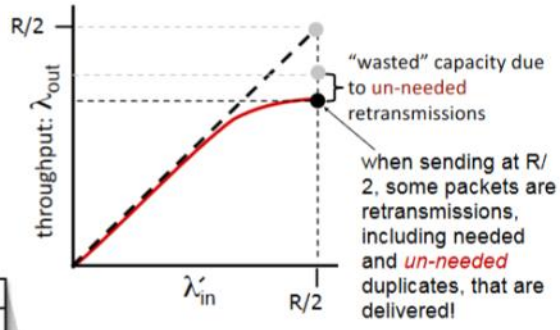


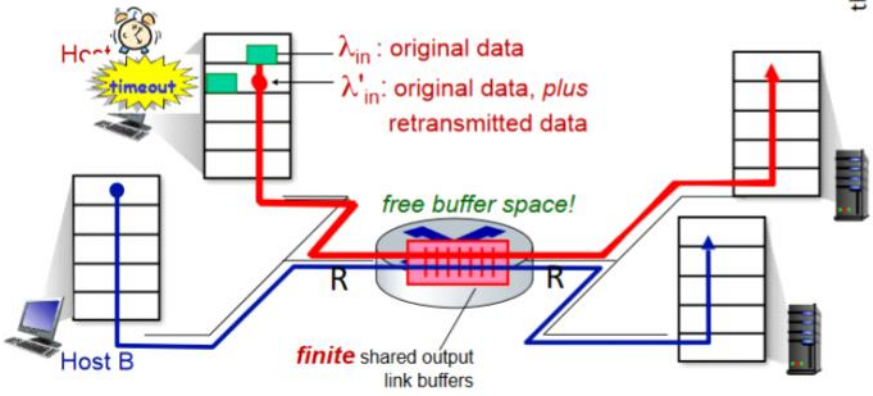


Causes/costs of congestion: scenario 2

Realistic scenario: *un-needed duplicates*

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

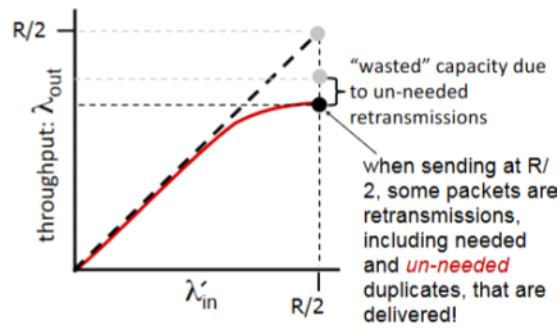




Causes/costs of congestion: scenario 2

Realistic scenario: *un-needed duplicates*

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



"costs" of congestion:

- more work (retransmission) for given receiver throughput
- unneeded retransmissions: link carries multiple copies of a packet
  - decreasing maximum achievable throughput

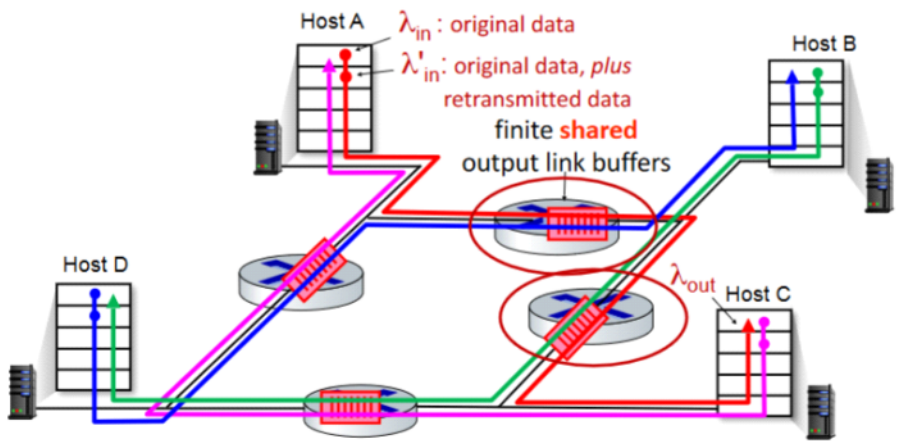
-Another scenario of why congestion is bad, this time with multiple routers:

Causes/costs of congestion: scenario 3

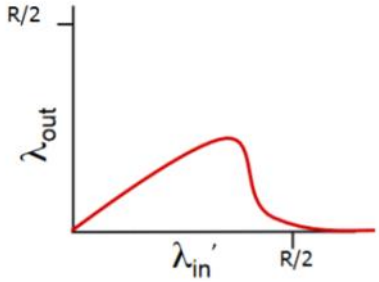
- *four* senders
- *multi-hop* paths
- timeout/retransmit

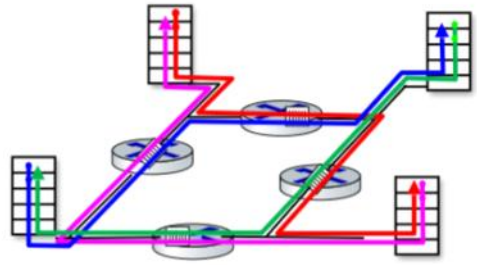
Q: what happens as λ_{in} and λ_{in}' increase ?

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



Causes/costs of congestion: scenario 3





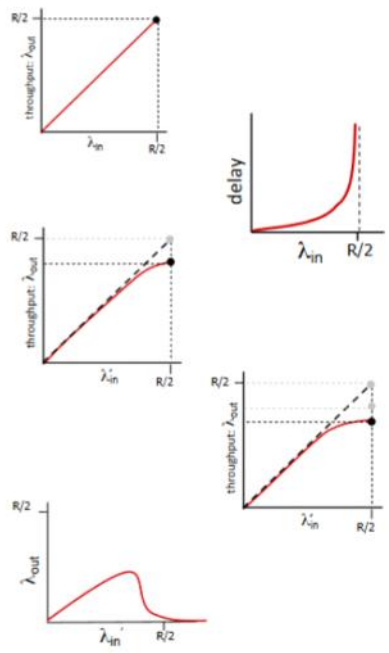
another "cost" of congestion:

- when packet dropped, any upstream transmission capacity and buffering used for that packet was wasted!

-Causes and costs of congestion:

Causes/costs of congestion: insights

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

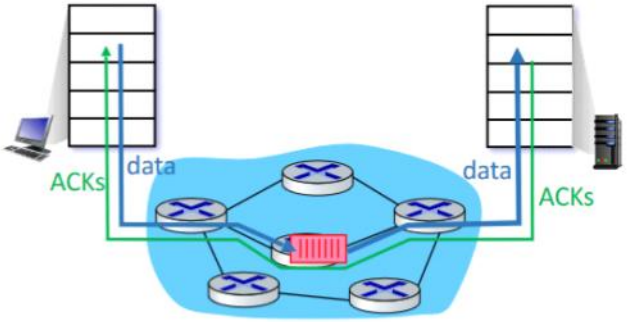


-Available approaches to congestion control:

Approaches towards congestion control

End-end congestion control:

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

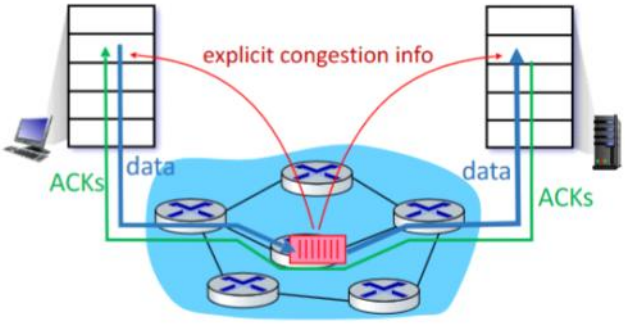


-End-to-end congestion - tells everyone on the Internet to slow down or speed up (the router will slow down the speed of data coming in to prevent an overflow)
-If we don't receive an ACK, we're either dropping the ACK or the original data at the router, this is the importance at looking at the full RTT (round trip time)

Approaches towards congestion control

Network-assisted congestion control:

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



-This might one day replace TCP or work alongside it
-This is a different protocol than TCP
-Router provides direct feedback through modifying specific congestion notification flags (e.g. ECN flag)

-How congestion control works for TCP
-**IMPORTANT FOR FINAL EXAM:** AIMD - Additive Increase, Multiplicative Decrease
-Sender sends at some speed, every time we get an acknowledgement back, we send more; when we stop getting acknowledgements back or we get premature timeouts (packet loss == congestion), we have to slow down

TCP congestion control: AIMD

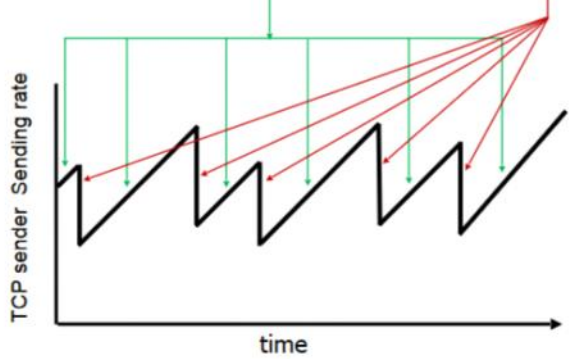
- *approach:* senders can increase sending rate until packet loss (congestion) occurs, then decrease sending rate on loss event

Additive Increase

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

Multiplicative Decrease

cut sending rate in half at each loss event



AIMD sawtooth behavior: *probing* for bandwidth

TCP AIMD: more

Multiplicative decrease detail: sending rate is

- Cut in half on loss detected by triple duplicate ACK (TCP Reno)
- Cut to 1 MSS (maximum segment size) when loss detected by timeout (TCP Tahoe)

Why AIMD?

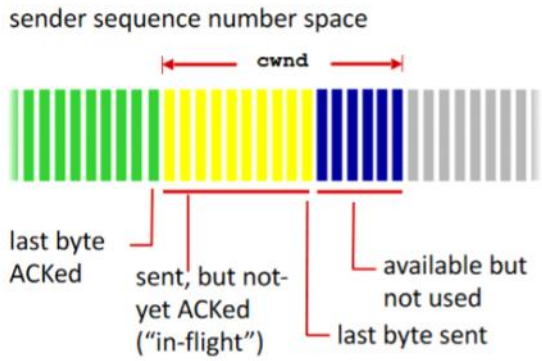
- AIMD – a distributed, asynchronous algorithm – has been shown to:
  - optimize congested flow rates network wide!
  - have desirable stability properties

-TCP Reno - cut the send rate in half if the network is minorly congested

-TCP Tahoe - if the network is completely congested, drop the sending rate to 1 MSS (1 packet at a time, 1500 bytes)

-TCP send rate formula, revisiting congestion window (cwnd):

TCP congestion control: details



TCP sending behavior:

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

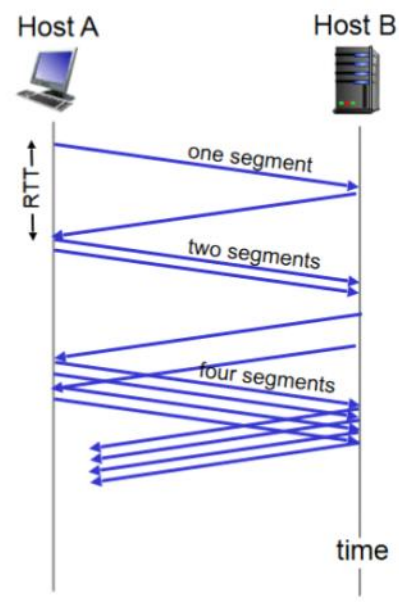
$$\text{TCP rate} \approx \frac{\texttt{cwnd}}{\text{RTT}} \text{ bytes/sec}$$

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

-TCP slow start - send 1 segment and double every RTT until packet loss, starts sending slow but ramps up exponentially fast:

TCP slow start

- when connection begins, increase rate **exponentially** until first loss event:
  - initially **cwnd** = 1 MSS
  - double **cwnd** every RTT
  - done by incrementing **cwnd** for every ACK received
- *summary:* initial rate is slow, but ramps up exponentially fast



-Congestion avoidance and the congestion control threshold:

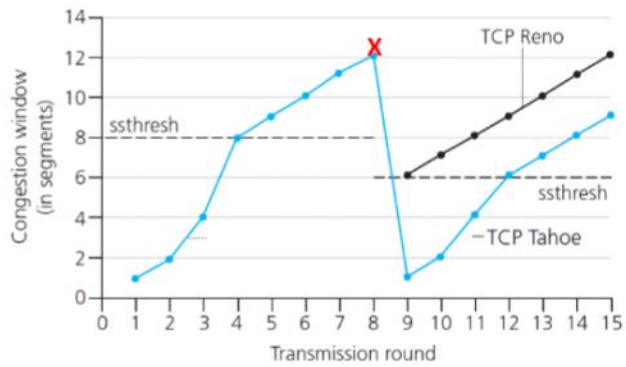
TCP: from slow start to congestion avoidance

Q: when should the exponential increase switch to linear?

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

Implementation:

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

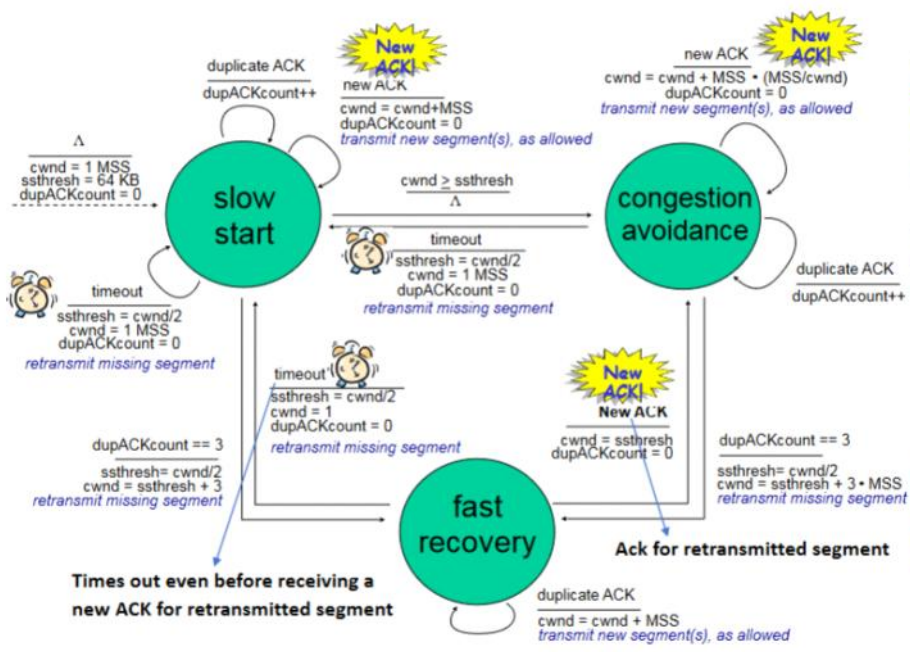


* Check out the online interactive exercises for more examples: http://gaia.cs.umass.edu/kurose_ross/interactive/

-**ON THE FINAL:** When we hit the threshold (after the slow start), we continue exponentially until we hit the threshold, then we have to change from "slow start" to "congestion avoidance" (where we just send 1 packet at a time, a linear growth rate) - eventually, a packet loss or timeout events occurs, which will reset us back to slow start AND the threshold is now half of what it was before

-TCP congestion control summary (with 4 minutes left in class /:):

Summary: TCP congestion control



Why MSS * (MSS/cwnd)?
One way to implement adding 1 segment at a time
e.g., MSS = 1460 bytes
cwnd = 14600 bytes
—> **10 segments** are being sent within one RTT.
Each new ACK increases cwnd by 1460 * 1460 / 14600 =
1460 * 1/10 = **146 bytes**
—> cwnd will increase by **1MSS after receiving all 10 ACKs for all 10 segments**

Why 3 * MSS?
(Implementation decision)
A good measure to account for the triple duplicate ACKs received

129

10/17 - Advanced TCP

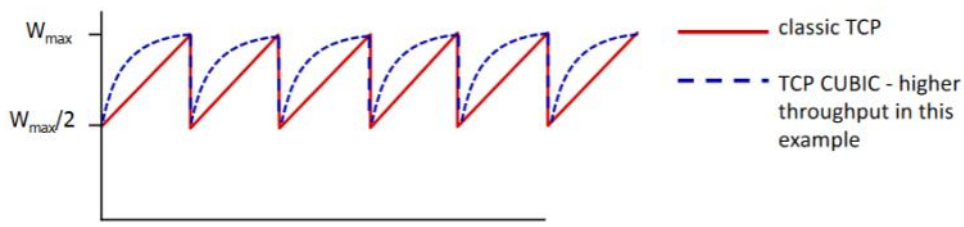
Thursday, October 17, 2024 5:57 PM

ALL INFORMATION FROM THIS POINT ONWARDS IS NOT ON EXAM, IT'S COMPLEMENTARY

-TCP CUBIC is an improvement on typical TCP - newer approach used by Linux and Google, derived from distance cubed

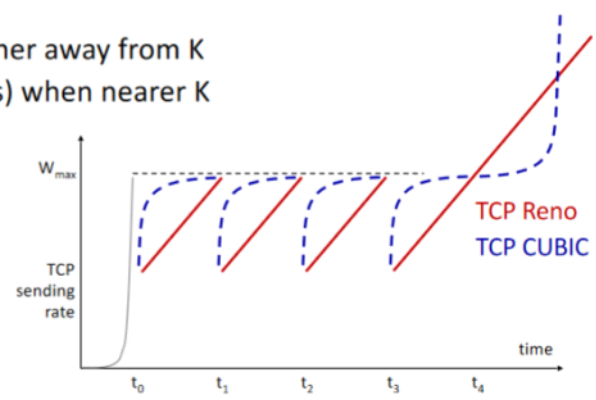
TCP CUBIC

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



TCP CUBIC

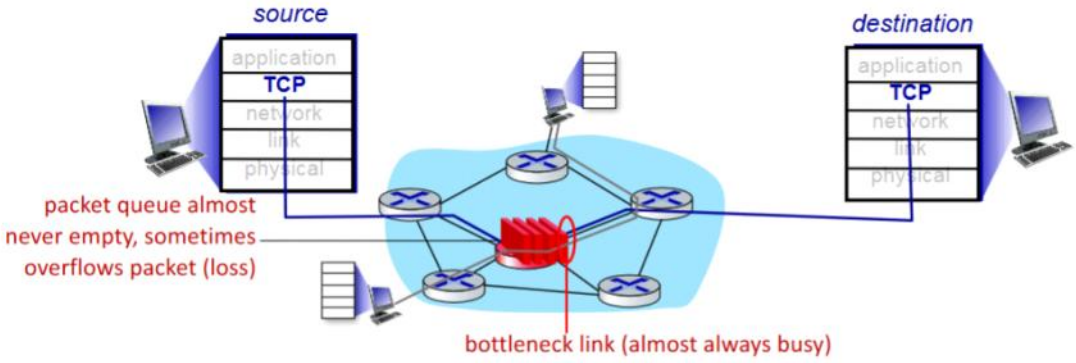
- **K (t_1, t_2, t_3, ...)**: point in **time when TCP window size will reach W_{max}**
  - K itself is tuneable
- increase W as a function of the ***cube*** of the **distance between current time and K**
  - larger increases when further away from K
  - smaller increases (cautious) when nearer K
- TCP CUBIC **default in Linux**, most popular TCP for popular Web servers



-TCP and congestion "bottleneck link" - making ways to ensure the router never gets full

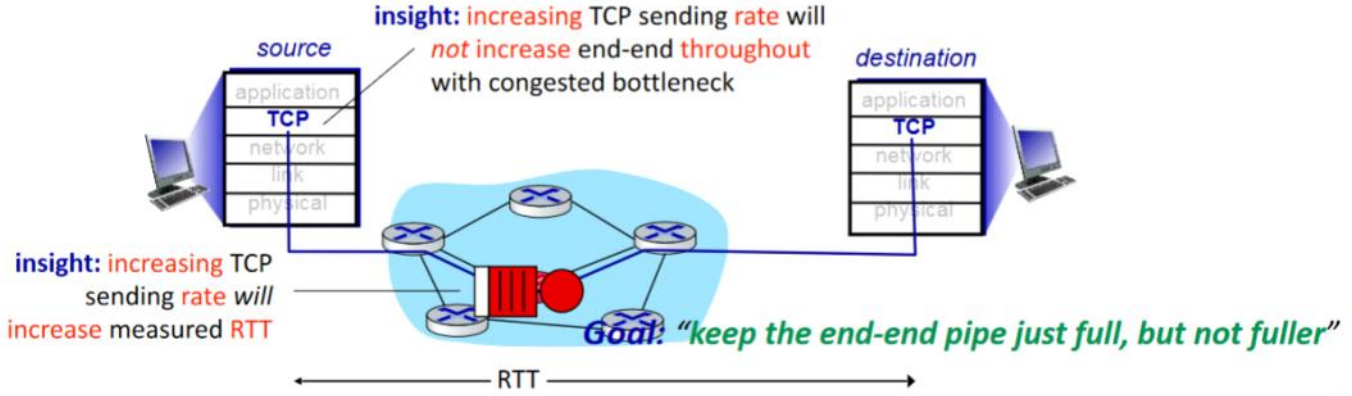
TCP and the congested "bottleneck link"

- TCP (classic or CUBIC) increase TCP's sending rate until packet loss occurs at some router's output: the *bottleneck link*



TCP and the congested "bottleneck link"

- TCP (classic or CUBIC) increase TCP's sending rate until packet loss occurs at some router's output: the *bottleneck link*
- understanding congestion: useful to focus on congested bottleneck link



133

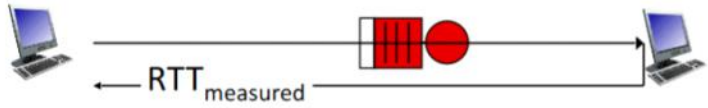
-Delay-based TCP congestion control:

Delay-based TCP congestion control

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

Delay-based TCP congestion control

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



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

Delay-based approach:

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

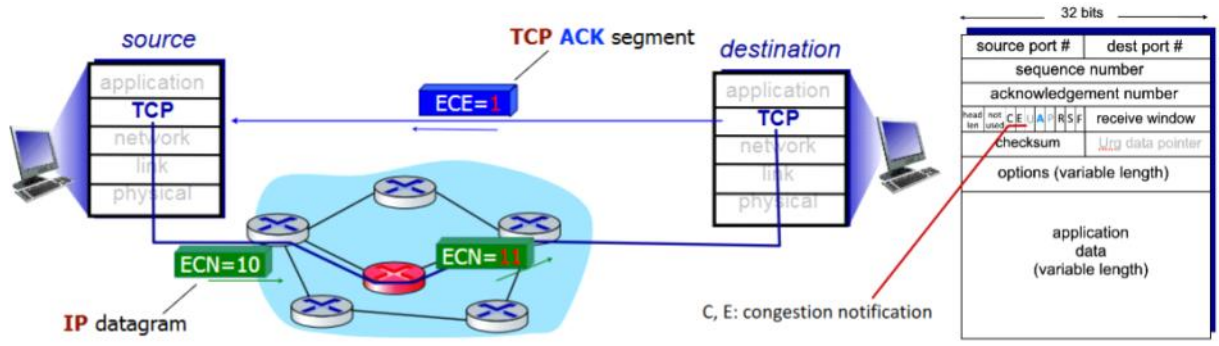
```
if measured throughput "very close" to uncongested throughput
    increase cwnd linearly          /* since path not congested */
else if measured throughput "far below" uncongested throughout
    decrease cwnd  linearly         /* since path is congested */
```

Delay-based TCP congestion control

- congestion control without inducing/forcing loss
- maximizing throughout ("keeping the pipe just full… ") while keeping delay low ("…but not fuller")
- a number of deployed TCPs take a delay-based approach
  - e.g., BBR deployed on Google's (internal) backbone network

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

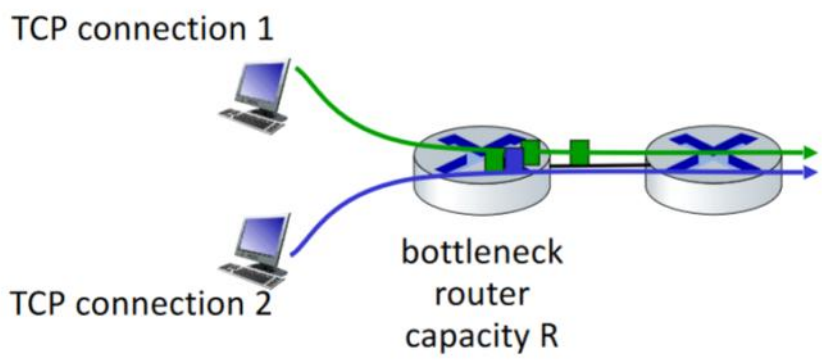
- two bits in **IP** header (**ToS field**) marked *by network router* to indicate congestion
  - *policy* to determine marking chosen by network operator
- congestion indication carried to destination
- destination sets **ECE bit** on **ACK** segment to notify sender of congestion
- involves both IP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



-TCP fairness

TCP fairness

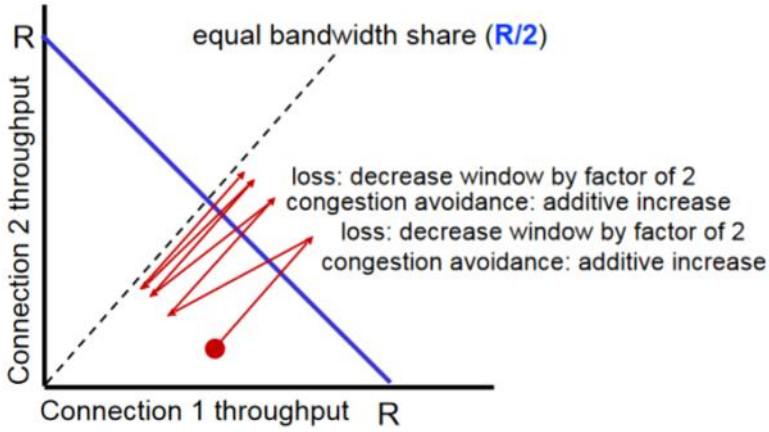
Fairness goal: if *K* TCP sessions **share** same **bottleneck link** of bandwidth *R*, **each should have average rate of *R/K***



Q: is TCP Fair?

Example: two competing TCP sessions:

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



Is TCP fair?

A: Yes, under idealized assumptions:

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

Fairness: must all network apps be "fair"?

Fairness and UDP

- multimedia apps often do not use TCP
  - do not want rate throttled by congestion control
- instead use UDP:
  - send audio/video at constant rate, tolerate packet loss
- there is no "Internet police" policing use of congestion control

Fairness, parallel TCP connections

- application can open *multiple* parallel connections between two hosts
- web browsers do this , e.g., link of rate R with 9 existing connections:
  - new app asks for 1 TCP, gets rate R/10
  - new app asks for 11 TCPs, gets R/2

-We're concerned about TCP fairness for the same reason we made HTTP/3, Web browsers can create unfairness in requests

Loss event (timeout or three duplicate ACKs)

Threshold is halved

Slow Start

10/17 - Port Forwarding

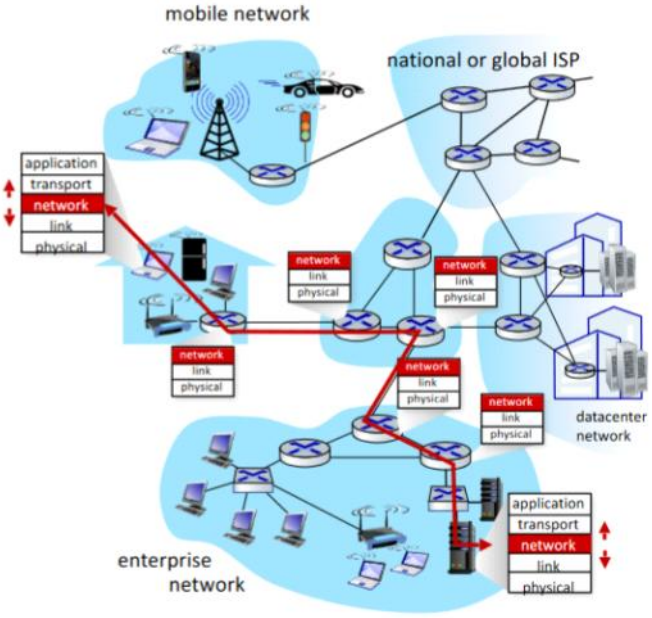
Tuesday, October 15, 2024 5:16 PM

-Goal of the network layer:
- -Routing (finding the best route to the destination)
- -Forwarding vs. routing
- -Host-to-host communication (IP addressing)
- -Protocols: IP, NAT

-A router handles scheduling (round-robin)
-Host-to-host communication
- -In the network layer, we have datagrams, not segments like with UDP/TCP

Network-layer services and protocols

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



4

-Routing uses some routing algorithm (Dijkstra's Shortest Path) to determine the route taken by packets from source to destination vs. forwarding which moves packets from a router's input link to its output link

Two key network-layer functions

network-layer functions:

- ***forwarding***: move packets from a router's input link to appropriate router output link
- ***routing***: determine route taken by packets from source to destination
  - *routing algorithms*

analogy: taking a trip

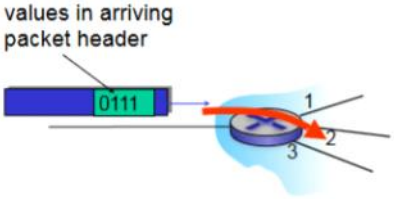
- ***forwarding***: process of getting through single interchange
- ***routing***: process of planning trip from source to destination

-Data plane vs control plane

Network layer: data plane, control plane

Data plane:

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



Control plane

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

-

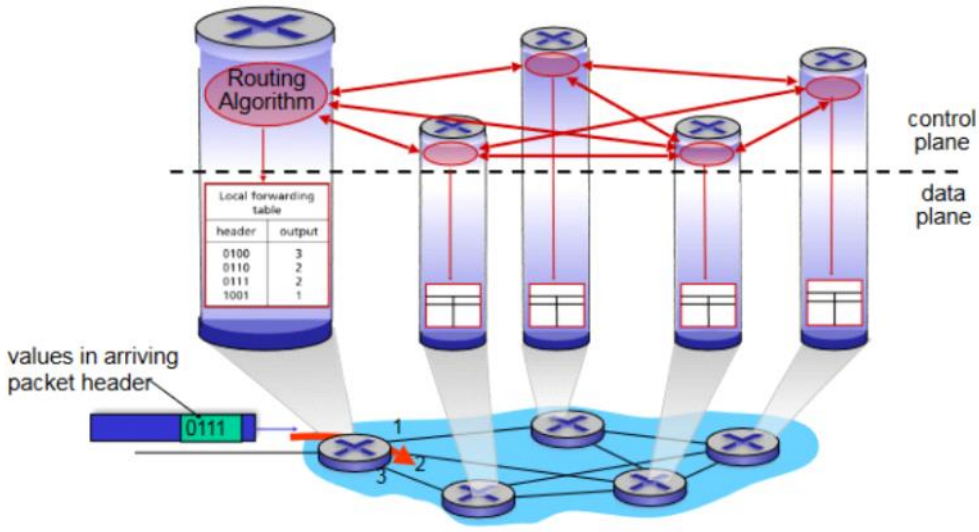
10/22 - Router Queuing

Tuesday, October 22, 2024 5:00 PM

-Routing is about finding the best path from source to destination across the network

Per-router control plane

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

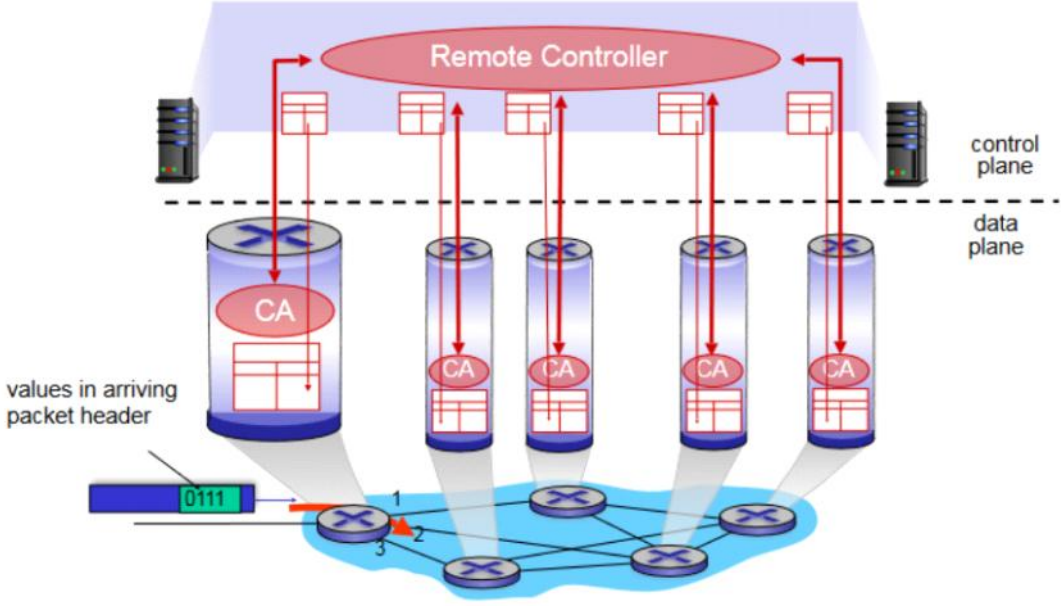


-Forwarding tables define the best path, every router has a forwarding table installed on it

-Software-Defined Networking (SDN) control plane - this is still in research, it's an algorithm to run offline and pass the forwarding table back to the router

Software-Defined Networking (SDN) control plane

Remote controller computes, installs forwarding tables in routers



-Refresher - each layer gives a service to the layer above it:

Network service model

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

example services for ***individual*** datagrams:

- guaranteed delivery
- guaranteed delivery with less than 40 msec delay

example services for a ***flow*** of datagrams:

- in-order datagram delivery
- guaranteed minimum bandwidth to flow
- restrictions on changes in inter-packet spacing

-TCP gives reliability and multiplexing
-The network layer gives services to the transport layer too, such as guaranteed delivery on individual segments and in-order delivery of datagrams
-IP doesn't really guarantee any of the services

Network-layer service model

Network Architecture	Service Model	Quality of Service (QoS) Guarantees ?			
		Bandwidth	Loss	Order	Timing
Internet	**best effort**	none	no	no	no

Internet "**best effort**" service model

No guarantees on:

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

-Networking protocols that don't matter anymore:

Network-layer service model

Network Architecture	Service Model	Quality of Service (QoS) Guarantees ?			
		Bandwidth	Loss	Order	Timing
Internet	best effort	none	no	no	no
ATM	Constant Bit Rate	Constant rate	yes	yes	yes
ATM	Available Bit Rate	Guaranteed min	no	yes	no
Internet	Intserv Guaranteed (RFC 1633)	yes	yes	yes	yes
Internet	Diffserv (RFC 2475)	possible	possibly	possibly	no

-Some experimentation and papers proved that IP is good enough for video and interactive voice applications, so it's not doing bad

Reflections on best-effort service:

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

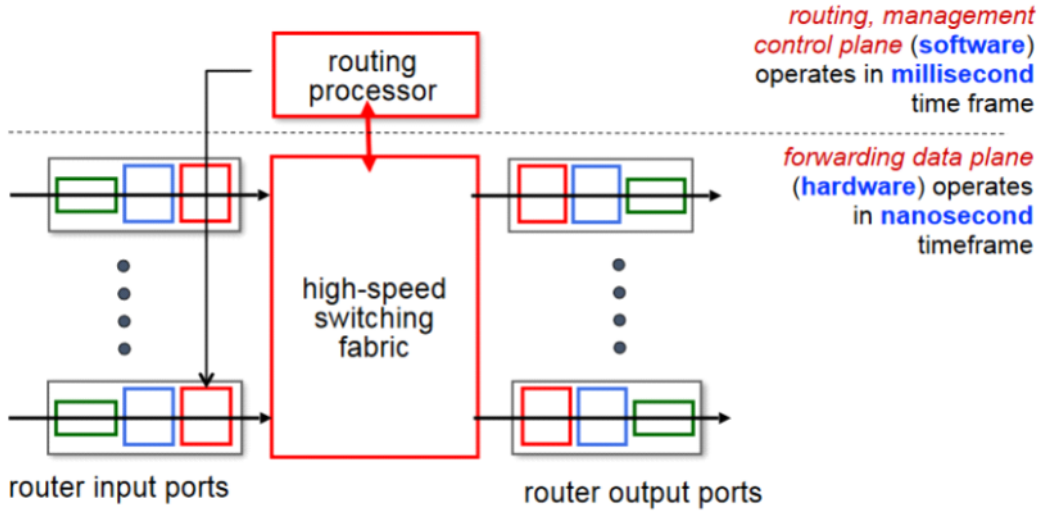
It's hard to argue with success of best-effort service model

-Content distribution networks (CDNs) - used by Netflix and streaming companies, it's about putting servers closer to people regionally (Europeans connect to European servers and Americans get routed to American servers)

-What's inside a router?

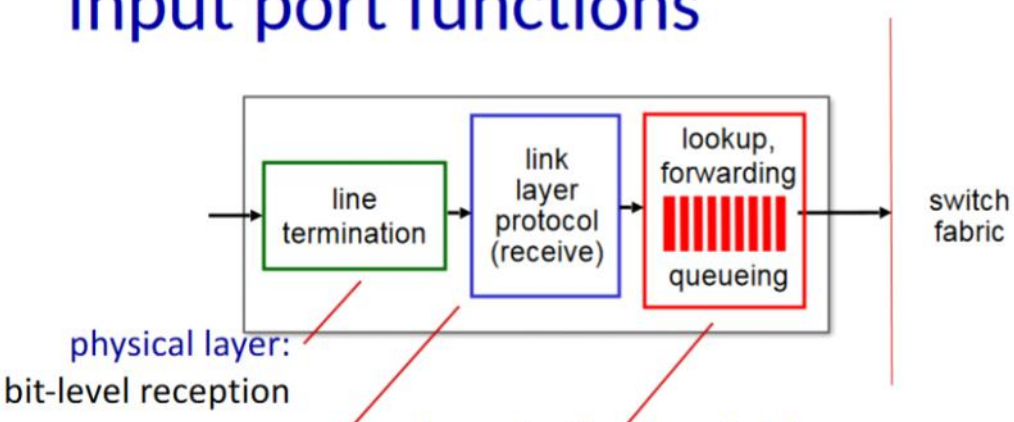
Router architecture overview

high-level view of generic router architecture:



-Circuitry in hardware to switch packets around between input ports and output ports in the router
-Routing processor, the brain of the router, calculates the forwarding table based on the routing algorithm being run and installs this forwarding table in memory
-IMPORTANT: routing happens in software but forwarding happens in hardware - it takes routing milliseconds but it takes forwarding nanoseconds
-A router sends data from one place to another; a switch connects computers over a network, so a switch doesn't forward data

Input port functions



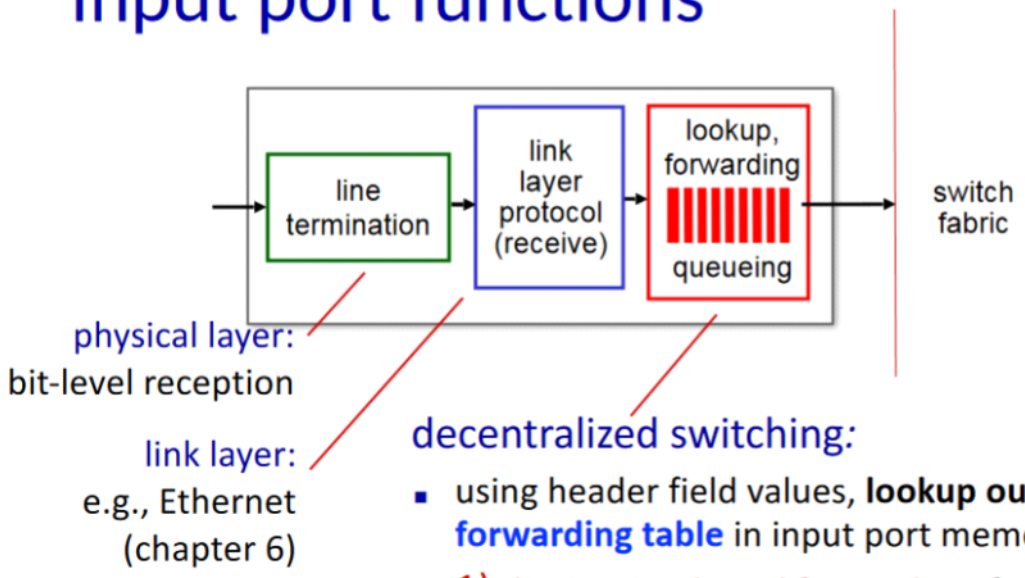
decentralized switching:

- using header field values, **lookup output port** using **forwarding table** in input port memory *("match plus action")*
- goal: complete input port processing at 'line speed'
- **input port queuing**: if datagrams arrive faster than forwarding rate into switch fabric

-Every computer has a network interface (NIC)
-Queuing happens at the memory of the input port (this is where loss an corruption occur)

Input port functions

Input port functions



decentralized switching:

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

-We forward based on IP

Destination-based forwarding

forwarding table

Destination Address Range	Link Interface
11001000 00010111 00010000 00000000 through	0
11001000 00010111 00010000 00000100 through 11001000 00010111 00010000 00000111	3
11001000 00010111 00011000 11111111	
11001000 00010111 00011001 00000000 through 11001000 00010111 00011111 11111111	2
otherwise	3

Q: but what happens if ranges don't divide up so nicely?

Longest prefix matching

longest prefix match

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

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

examples:

11001000 00010111 00010110 10100001 which interface?

11001000 00010111 00011000 10101010 which interface?

…Too much to copy, look in slides to see how longest prefix matching works
-Ternary content addressable memories (TCAMs) - using hardware (super fast in nanoseconds) to find content addressable memory with constant time lookup of one clock cycle regardless of table size (hash table), gets us our longest prefix matching

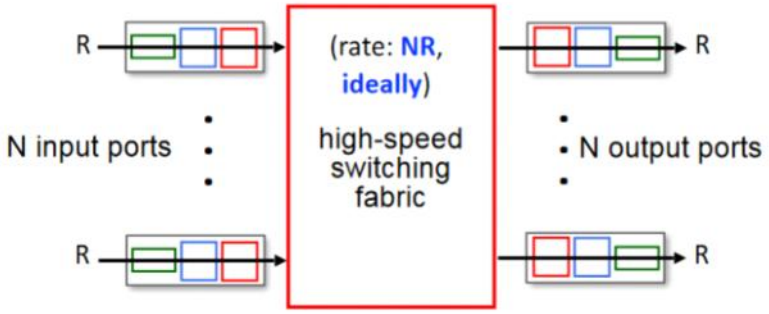
Longest prefix matching

- we'll see *why* longest prefix matching is used shortly, when we study addressing
- longest prefix matching: often performed using ternary content addressable memories (**TCAMs**)
  - *content addressable:* present address to TCAM: retrieve address in one clock cycle (**constant time lookup**), regardless of table size
  - Cisco Catalyst: ~1M routing table entries in TCAM

-Similar to FGPAs
-N*R (N inputs * line rate R) is the ideal rate of packet switching in the router, with near no delays

Switching fabrics

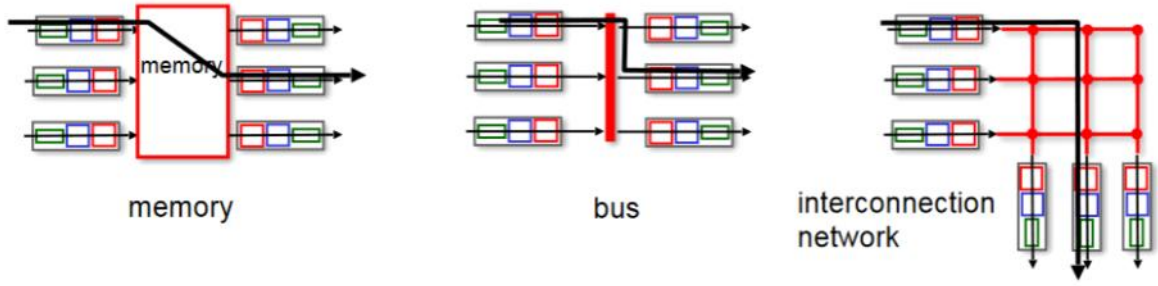
- transfer packet from input link to appropriate output link
- switching rate: rate at which packets can be transferred from inputs to outputs
  - often measured as multiple of input/output line rate
  - **N inputs**: desirable switching rate is N times **line rate R** (i.e., **NR**)



-Techniques for switching input port to output port

Switching fabrics

- transfer packet from input link to appropriate output link
- switching rate: rate at which packets can be transferred from inputs to outputs
  - often measured as multiple of input/output line rate
  - **N inputs**: desirable switching rate is N times **line rate R** (i.e., **NR**)
- **three** major **types** of switching fabrics:

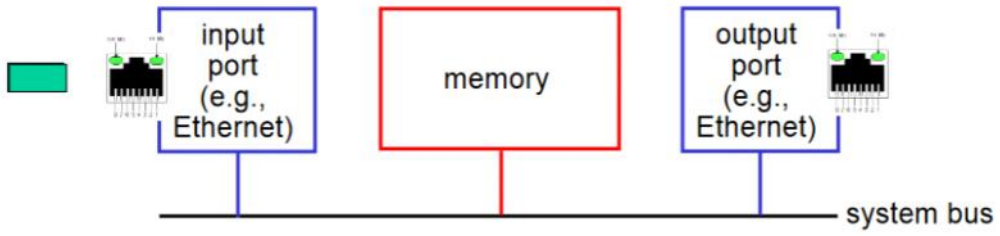


-Old school routers used shared memory for switching and it was slow (that's how you ended up with queued packets)
-Then they ended up with the "bus" technique
-Now we have nanoscale interconnected networks

Switching via memory

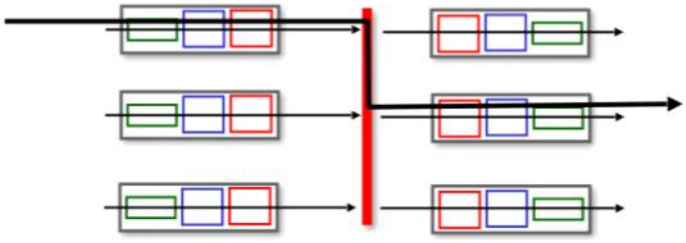
first generation routers:

- **traditional** computers with switching under direct control of **CPU**
- packet copied to system's memory
- speed limited by memory bandwidth (2 bus crossings per datagram)
  - if memory bandwidth is **B** packets/sec (i.e., max of B packets per second can be written into or read from memory)
    -> forwarding rate: **B/2** packets/sec
- Only **one** memory **read/write** can be done **at a time** over shared system bus



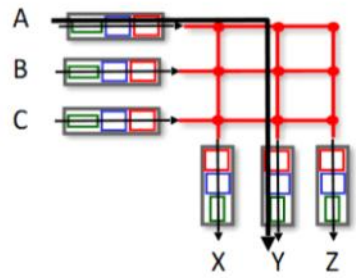
Switching via a bus

- datagram from input port memory to output port memory via a shared bus
- typical implementation: A **label/header** is appended to packet at input port, packet forwarded to bus, all output ports receive copy of packet but **only output port matching destination in label keeps it,** label is then removed
  - Packet crosses the bus only once rather than twice (via memory)
- *bus contention:* switching speed limited by bus bandwidth
- 32 Gbps bus, Cisco 5600: sufficient speed for **access routers**



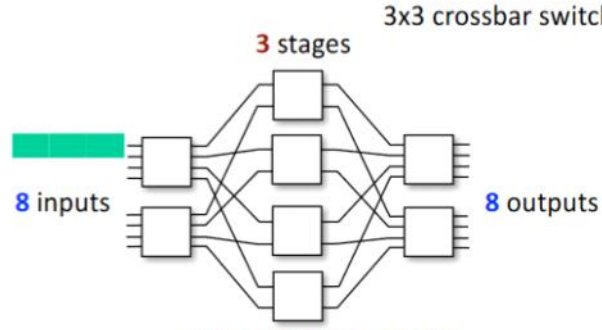
Switching via interconnection network

- Crossbar, Clos networks, other interconnection nets initially developed to connect processors in multiprocessor computer arch.
- **Capable of forwarding multiple packets at same time**, e.g., A -> Y, B -> X
- **Limitation**: Cannot have two packets from two input ports enter the bus at same time if destined to same output port (e.g., A -> Y, B -> Y)



3x3 crossbar switch

- multistage switch: *nxn* 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

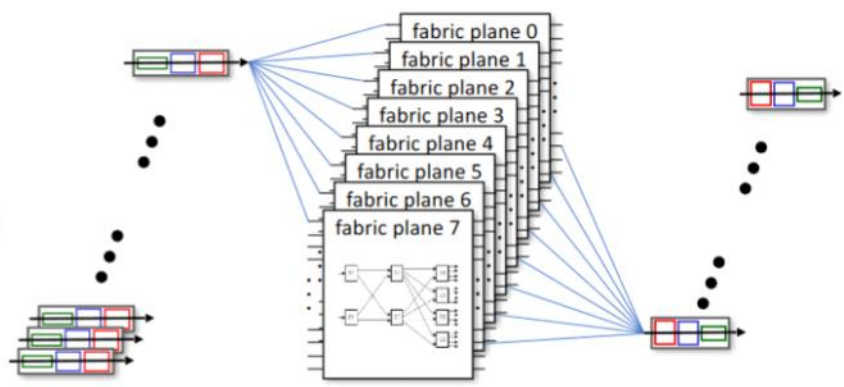


8x8 multistage switch
built from smaller-sized switches
(here **3**-stage)

27

Switching via interconnection network

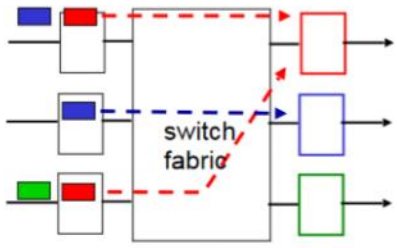
- scaling, using multiple switching fabrics/planes in parallel:
  - speedup, scaleup via parallelism
- Cisco CRS router:
  - basic unit: **8 switching fabrics**/planes
  - each plane: **3-stage** interconnection **network**
  - up to **100's Tbps** switching capacity



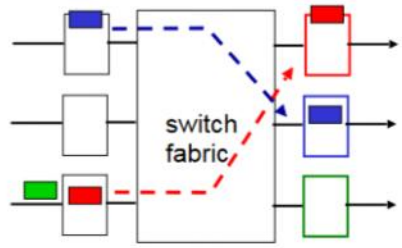
-Head-of-the-Line (HOL) blocking - you're in a queue, the queued item in the front of the line is taking too long and blocking everyone else from "ordering"

Input port queuing

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



output port contention: only one **red** datagram can be transferred. **lower red** packet is ***blocked*** (both reds are destined to same **red output port**)

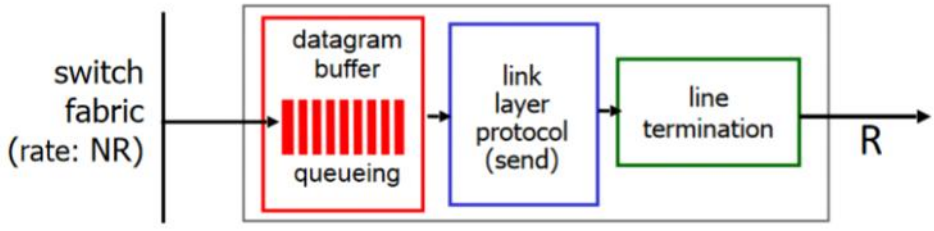


one packet time later: **green** packet experiences **HOL** blocking

-Scheduling discipline - round-robin and scheduling happens on the output port; also has some drop policy to drop data in the router

Output port queuing

This is a really important slide



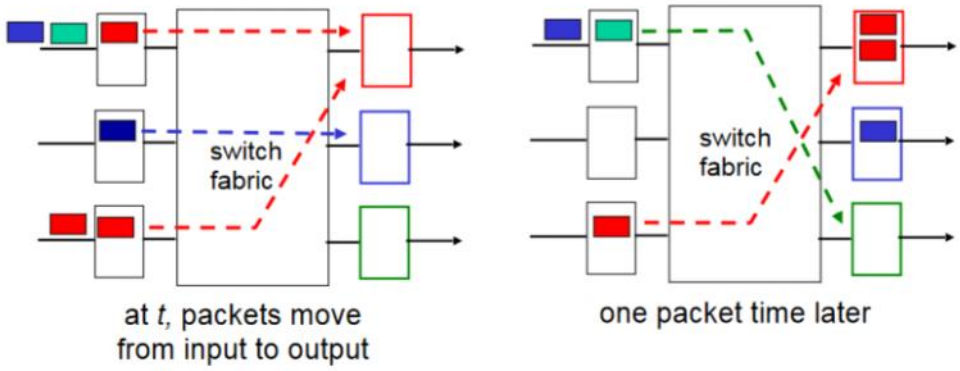
- *Buffering* required when datagrams arrive from fabric faster than link transmission rate.
  - *Drop policy:* which datagrams to drop if **no free buffers (i.e., buffer full)**? → Datagrams can be lost due to congestion, lack of buffers
- *Scheduling discipline* chooses among queued datagrams for transmission → **Priority scheduling** – who gets best performance, network neutrality

30

-Network neutrality has to do with the queue at the output port - net neutrality issue means big companies can pay to have their packets not dropped and prioritized in the output port in the router while others get dropped (this got petitioned against)

-Queuing delays and loss of packets due to output port buffer overflows

Output port queuing



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

-The input port is usually so fast (since it's hardware level) that that's not the bottleneck, the output port is typically the bottleneck because it has to handle data that the input port is sending over super fast, the output port is slower so the difference between how fast the input port is moving data through and how slow the output port is moving data out is what leads to congestion control and queuing delays and packet loss

10/24 - IPv4, Net Neutrality

Wednesday, October 23, 2024 12:26 PM

-Formula for buffering used in router

How much buffering?

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

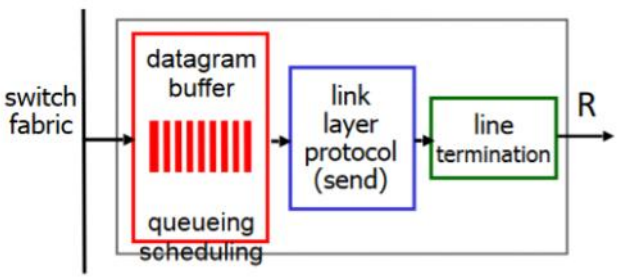
$$\frac{RTT \cdot C}{\sqrt{N}}$$

(larger number of TCP flows -> smaller buffer needed)

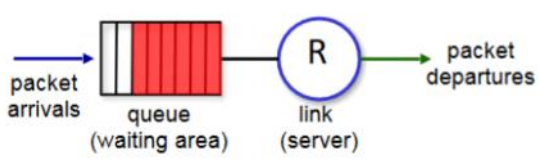
- *too* much buffering can increase delays (particularly in home routers)
  - long RTTs: poor performance for realtime apps (gaming), sluggish TCP resp.
  - **recall delay-based congestion control**: "**keep bottleneck link just full enough (busy) but no fuller**"

-There priority of which packets get dropped is determined by IP/diffserv (more advanced network layer protocol) at the Network Layer

Buffer Management



Abstraction: queue



buffer management:

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

-Scheduling like in OS but for networks

Packet Scheduling: FCFS

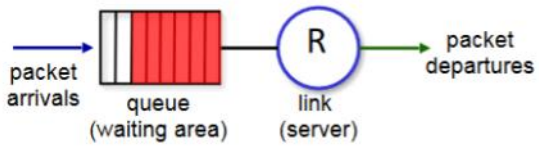
packet scheduling: deciding which packet to send next on link

- 1) first come, first served
- 2) priority
- 3) round robin
- 4) weighted fair queueing

FCFS: packets transmitted in order of arrival to output port

- also known as: First-in-first-out (**FIFO**)
- real world examples?

Abstraction: queue

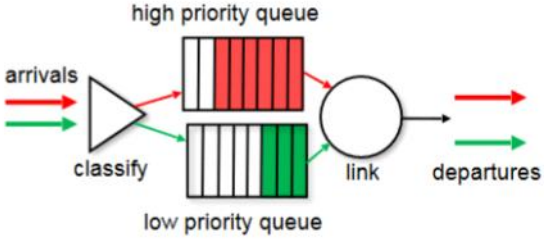


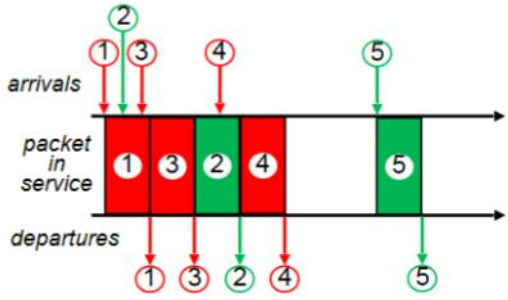
-Newer routers have been getting access to the destination TCP port (from the Transport Layer) on the Network Layer, routers used to only know about IP at the Network Layer

Scheduling policies: priority

Priority scheduling:

- arriving traffic classified, queued by class
  - any header fields can be used for classification (e.g., **dest TCP port** indicates a network management packet)
- send packet from highest priority queue that has buffered packets
  - **FCFS** within priority class



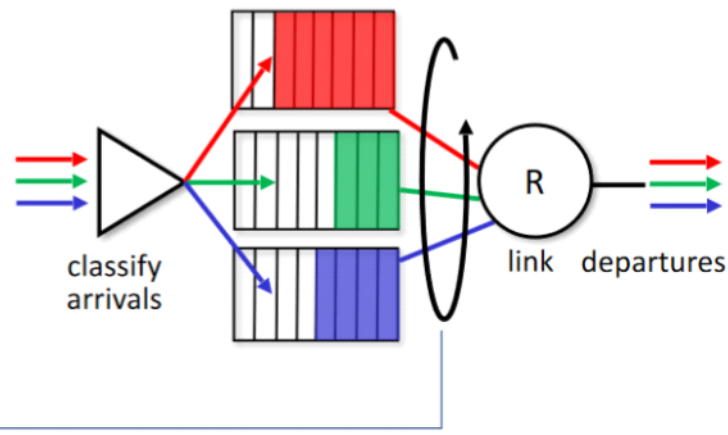


-Round robin (RR) scheduling is used in networks too

Scheduling policies: round robin

Round Robin (RR) scheduling:

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



-Weighted fair queuing (WFQ) is like a RR (carousel) where the weights are like priorities, so pretty much, we have a circular priority queue

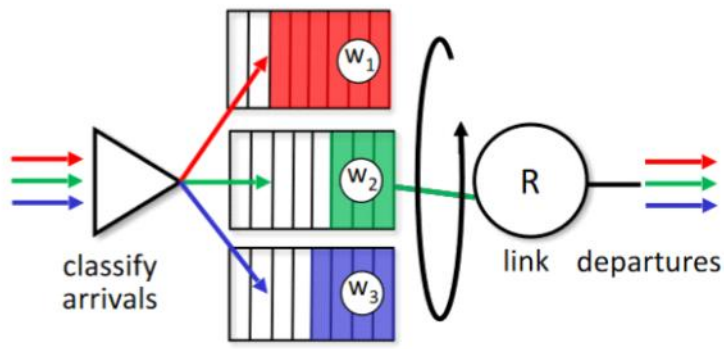
Scheduling policies: weighted fair queueing

Weighted Fair Queuing (WFQ):

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

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

- minimum bandwidth guarantee (per-traffic-class)



-Cool, we're talking about net neutrality

Sidebar: Network Neutrality

What is network neutrality?

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

Different countries have different "takes" on network neutrality

-Putting a limit on bandwidth would mean everyone would be watching 720p YouTube videos because it costs less

Sidebar: Network Neutrality

2015 US FCC *Order on Protecting and Promoting an Open Internet:* three "clear, bright line" rules:

- no blocking … "shall not block lawful content, applications, services, or non-harmful devices, subject to reasonable network management."
- no throttling … "shall not impair or degrade lawful Internet traffic on the basis of Internet content, application, or service, or use of a non-harmful device, subject to reasonable network management."
- **no paid prioritization. … "shall not engage in paid prioritization"**

-You cannot block traffic (e.g. an ISP can't block all of Google's traffic or all of Netflix's traffic)
-You can't throttle an application based on who's offering (e.g. can't have Google have better connection than Yahoo)
-No paid prioritization of networks

ISP: telecommunications or information service?

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

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

US Telecommunication Act of 1934 and 1996:

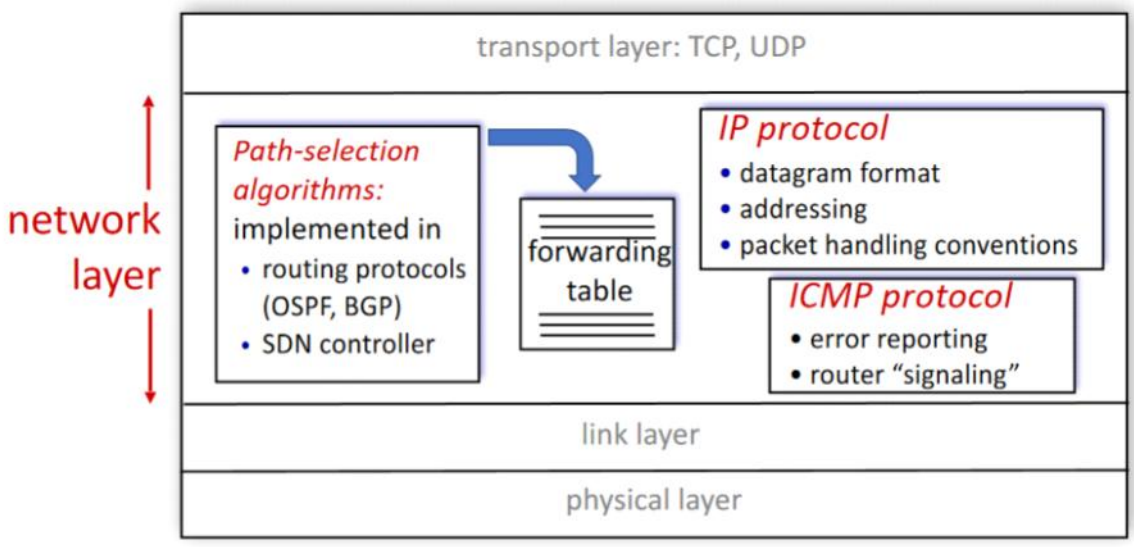
- *Title II:* imposes "common carrier duties" on *telecommunications services*: reasonable rates, non-discrimination and *requires regulation*
- *Title I:* applies to *information services:*
  - no common carrier duties (*not regulated*)
  - but grants FCC authority "… as may be necessary in the execution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

-Reading the US Telecommunication Act makes it sound like the Internet is an information service and not a telecommunication service

-Network Layer - the Internet

Network Layer: Internet

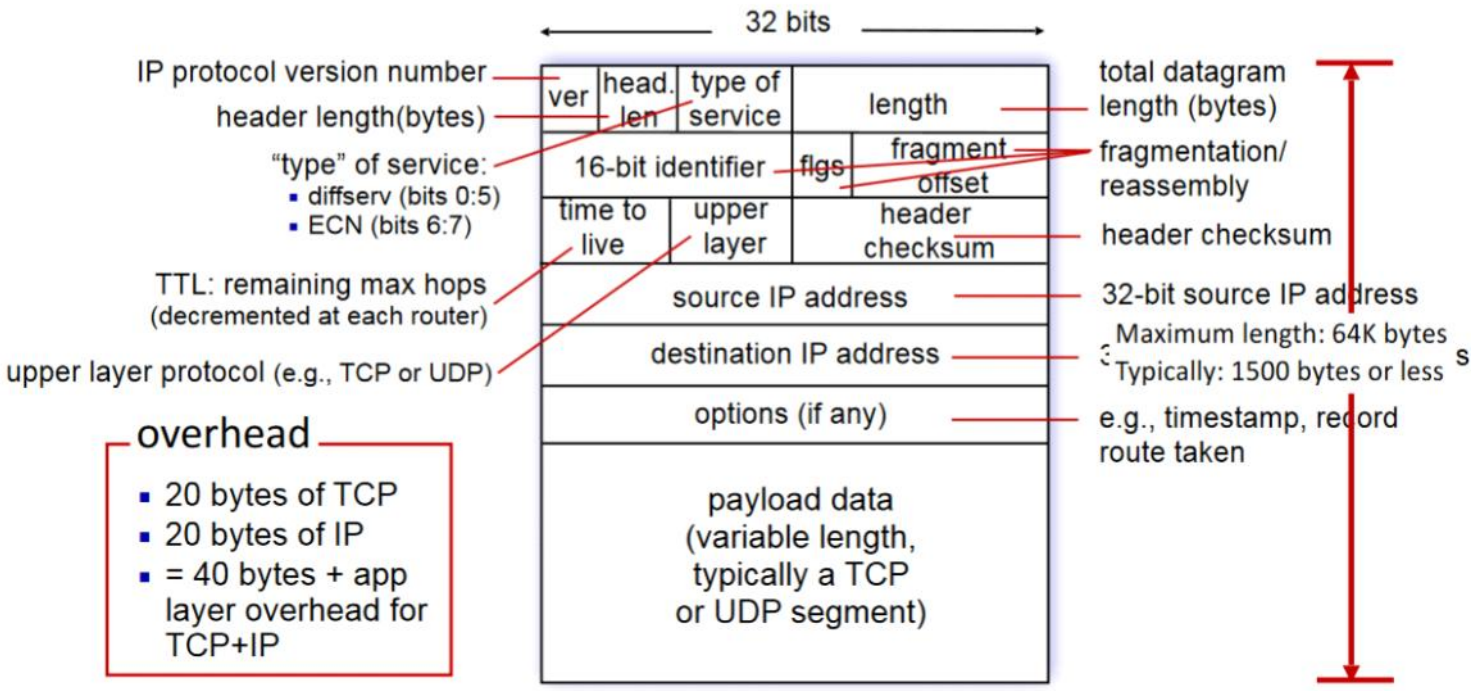
host, router network layer functions:



-Routing protocols we'll talk about: OSPF, BGP
-ICMP protocol, this is how the "ping" command works, using an ICMP datagram (not an IP datagram)

-IPv4 datagram format

IPv4 Datagram format

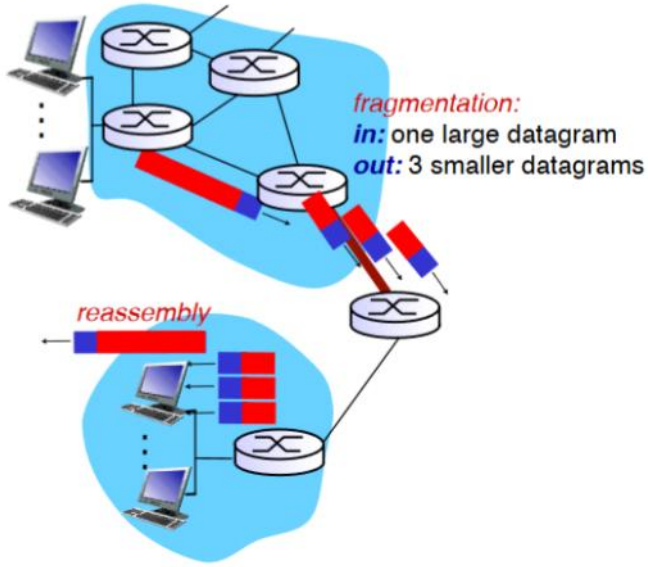


-We need to have the header length specified because it's variable due to the options field (same as TCP)
-Type of service (we won't talk about that in this class)
-Total length of the packet = header + payload = 20 bytes + 1480 bytes
-Header size = 32 bits (or 4 bytes) * 4 rows of data = 20 bytes
-A typical IP datagram has a payload of 1480 bytes, maximum of 64K bytes (2^16)
-Fragment offset (we'll talk about this later)
-Time to live (TTL) - same as with DNS cache, you need a TTL for a packet so you can drop it from the Internet to prevent it from continuing to circle forever (these are called routing loops); a router decrements this number by 1 until it hits 0 (remaining max hops)
-Upper layer - demultiplexing, UDP or TCP?
-Header checksum - there's an argument this is a duplicate functionality with the transport layer since it already has checksum, I guess we're just making extra sure that our data is sent correctly
-Source IP address and destination IP address (same as source port and destination port in UDP and TCP from the Network Layer)
-General idea: We need a field for every service
-Total overhead: 20 bytes from TCP header size + 20 bytes from IP header size + app layer overhead for TCP and IP = 40 bytes + app layer overhead for TCP and IP

-IPv4 fragmentation - maximum transmission unit (MTU) says the largest possible data/link-level "frame" has to be of a certain size, thus, the payload size of an IP "datagram" on the network layer above it has to also be a certain size, fragmented have you

IPv4 Fragmentation

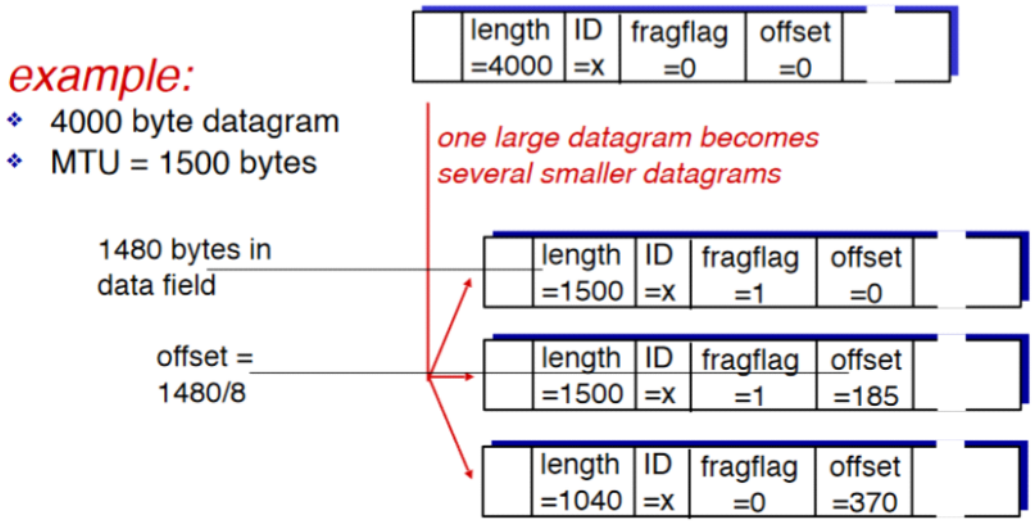
- network links have **MTU** (max.transfer size) - largest possible link-level frame
  - **different link** types, **different MTU**s
- large IP datagram divided ("**fragmented**") within net
  - one datagram becomes several datagrams
  - "reassembled" only at final destination
  - IP header bits used to identify, order related fragments



-We need a 16-bit identification field on an IP datagram to get the ordering of large IP datagrams splits into multiple IP fragments

-Example of IPv4 fragmentation:

IPv4 Fragmentation, reassembly



-If a fragment is dropped, we have to resend the whole IP datagram through TCP

-Our offset is 185 because it's a multiple of 8 (1480 bytes per IP datagram / 8 = 185)

-4000 bytes (including the header), 3980 payload and 20 bytes header

10/29 - Subnetting, DHCP

Tuesday, October 29, 2024 5:08 PM

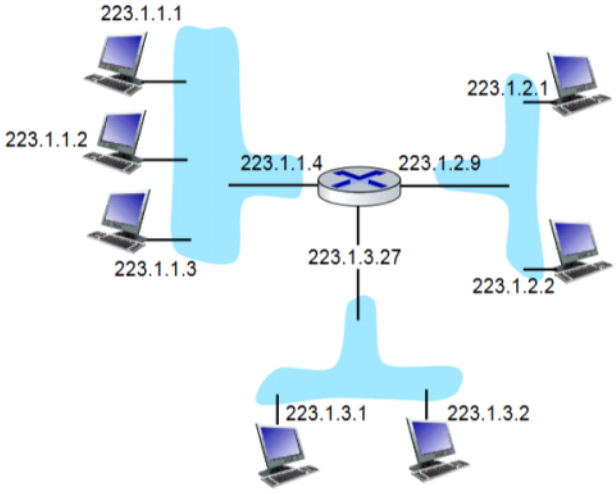
-IP address is a 32-bit identifier associated with each host or router interface (NIC)

-Each 8 bits is converted to decimal then put a dot between each 8-bits, this is how we represent IP addresses

-You can assign an IP address to every NIC (network interface card), containers and VMs also have virtual NICs so they have their own IP addresses too

IP addressing: introduction

- 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 (e.g., wired Ethernet, wireless 802.11)



dotted-decimal IP address notation:

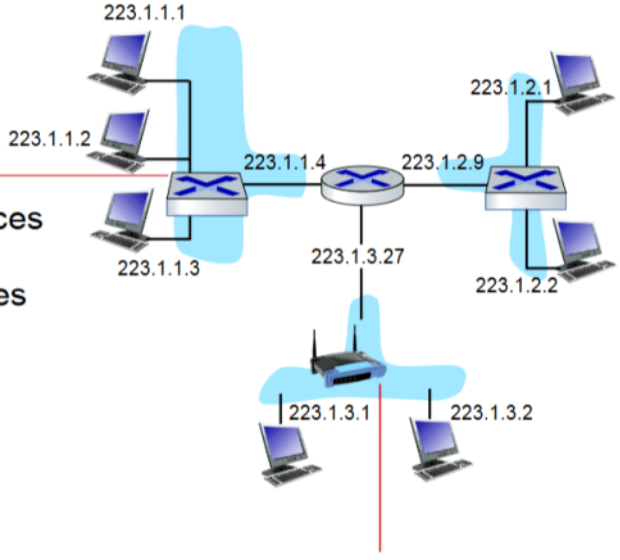
223.1.1.1 = 11011111 00000001 00000001 00000001

223 1 1 1

IP addressing: introduction

Q: how are interfaces actually connected?

A: we'll learn about that in chapters 6, 7



A: wired Ethernet interfaces connected by Ethernet switches

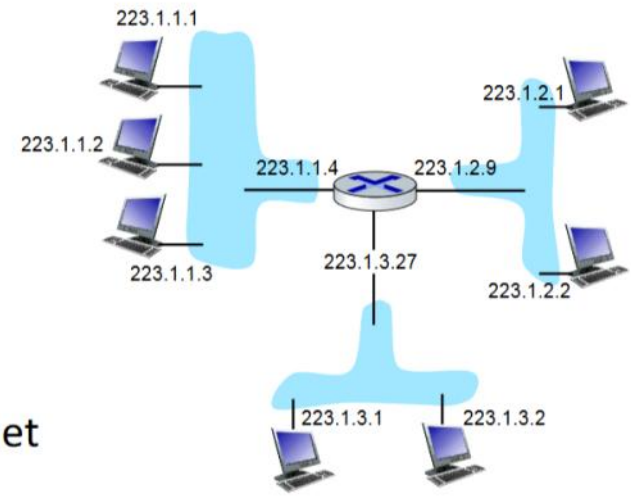
A: wireless WiFi interfaces connected by WiFi base station

For now: don't need to worry about how one interface is connected to another (with no intervening router)

-Subnetting - a subnet is a network that shares the same network address

Subnets

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



network consisting of 3 subnets

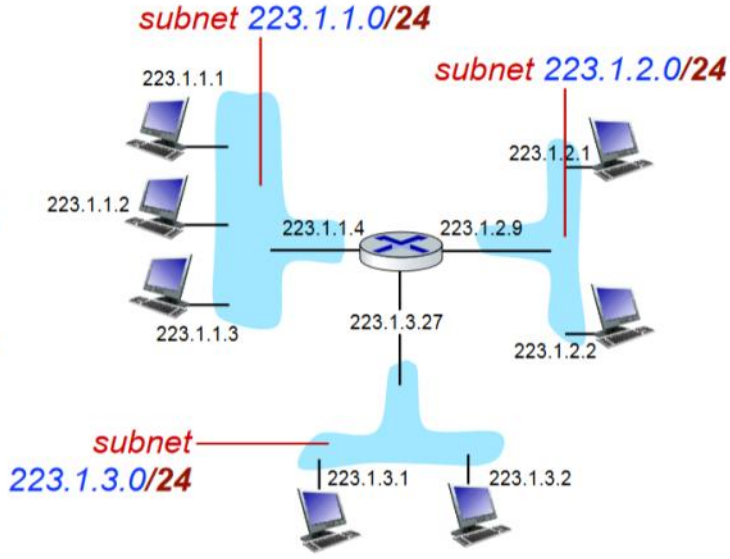
- IP addresses have structure:
  - subnet part: devices in same subnet have common high order bits
  - host part: remaining low order bits

-In the diagram, we have a router with 3 interfaces, 1 network for each interface
-The computers on the same network share the same first 24-bits (223.1.1) and the last 8-bits (223.1.1.1, 223.1.1.2, 223.1.1.3, 223.1.1.4) are different
-The subnetting part is here: 223.1.1.X, 223.1.2.X, 223.1.3.X
-Subnet mask - all IPs that share the same first 24-bits are on the same network

Subnets

Recipe for defining subnets:

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

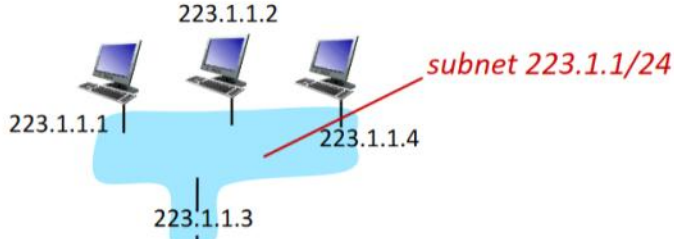


subnet mask: **/24**
(**high-order 24 bits**: subnet part of IP address)

-Subnet example
-In the example below, the two routers share a subnet 223.1.7.X

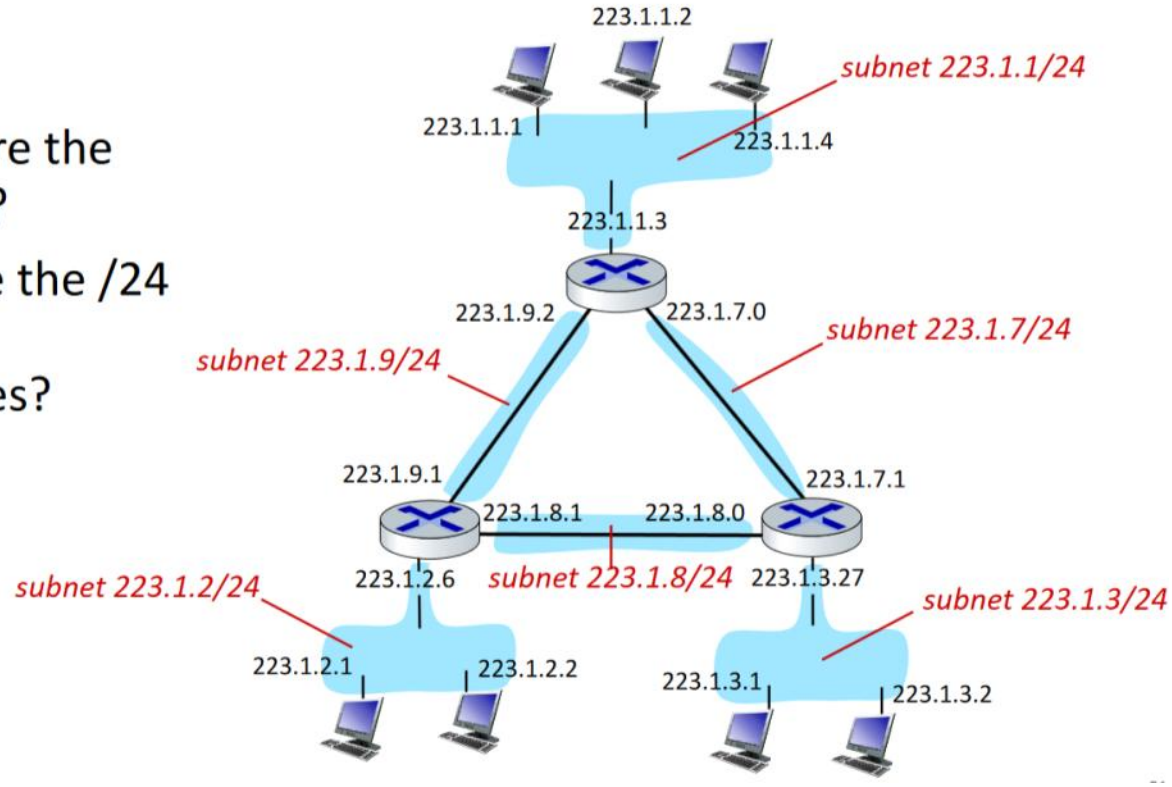
Subnets

- where are the subnets?



Subnets

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



-CIDR (Classless Interdomain Routing) - why we have the /24 part on the network
  -Example: **223.1.1**.0/24 - **network part** and host part
    -We put /24 because the first 24-bits are the subnet part
    -If the first 23 bits of the IP address identified the subnet/network and the remaining bits identified the host, then we'd have /23 at the end of our IP address

IP addressing: CIDR

CIDR: Classless InterDomain Routing (pronounced "cider")

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

  -The broadcast network for above is 200.23.17.255
-For the host part of the IP address:
  -If we have an IP address 200.23.16.0, that's the network address and cannot be a host, a broadcast network also cannot be assigned to a host part, 2^9 - 1 network for the network address (ends in .0) and - 1 network for broadcast network (ends in .255) = 512 - 2 = 510 number of usable hosts (ranging from ending in .1 to ending in .254)
-How do you get an IP address?
  -You can manually set your own IP address from settings (in Linux, go to /etc/rc.config)
  -DHCP - gives you the IP address that you need to use automatically

IP addresses: how to get one?

IP addresses: how to get one?

That's actually two questions:

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

How does *host* get IP address?

- **hard-coded** by sysadmin in config file (e.g., /etc/rc.config in UNIX)
- **DHCP**: **D**ynamic **H**ost **C**onfiguration **P**rotocol: dynamically get address from as server
  - "plug-and-play"

-DHCP (Dynamic Host Configuration Protocol) - is an application layer protocol that dynamically gets your IP address from your local IP server

-DHCP discover - client asks the server if it's a DHCP server
-DHCP offer - server response in confirmation that it's a DHCP server
-DHCP request - client requests a specific IP address from the DHCP server
-DHCP ack - server responds with the valid IP address
-NOTE: DHCP request and DHCP ack can be skipped if the IP address is cached between the client-server

DHCP: Dynamic Host Configuration Protocol

goal: host *dynamically* obtains IP address from network server when it "joins" network

- can renew its lease on address in use
- allows reuse of addresses (only hold address while connected/on)
- support for mobile users who join/leave network

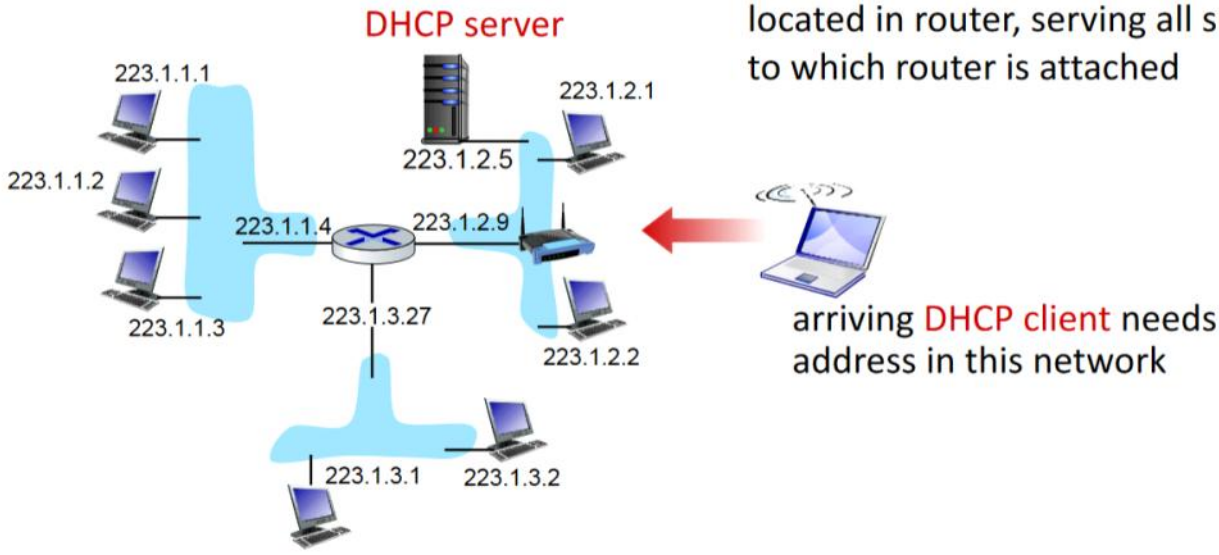
DHCP overview:

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

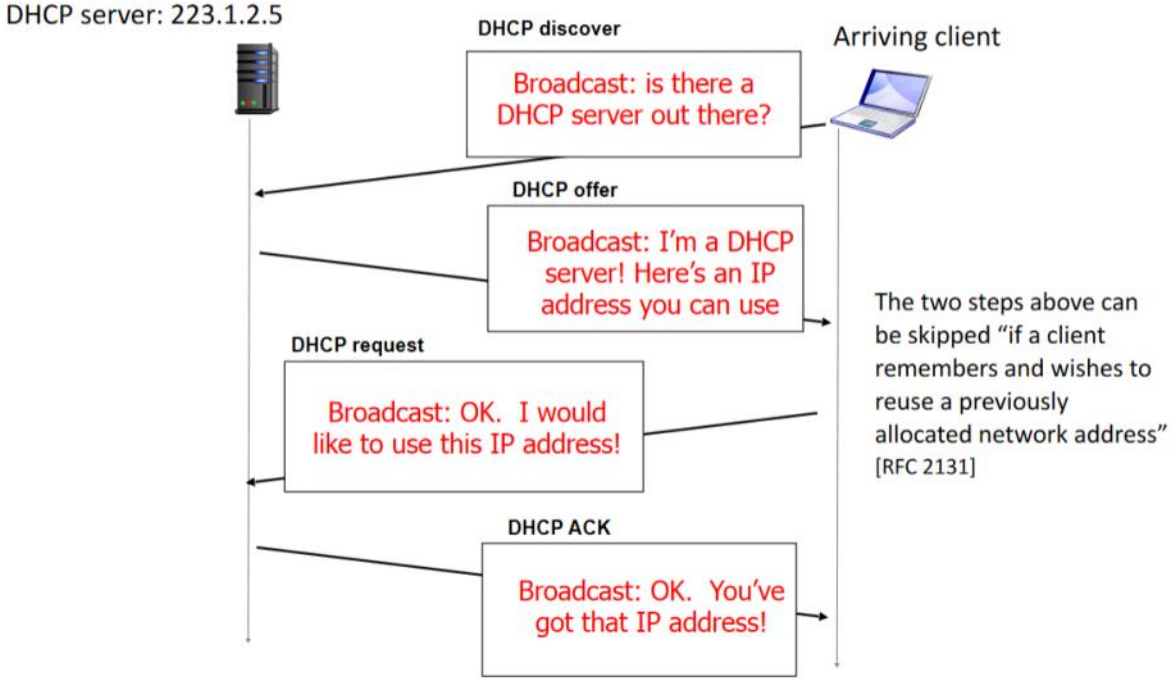
-Diagram of how DHCP works:

DHCP client-server scenario

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



DHCP client-server scenario



-DHCP runs on port 67
-DHCP client-server protocol has RFC

DHCP: client-server interactions (RFC 2131)

- DHCPREQUEST **MUST** include a 'server identifier' option to indicate which server it has selected (in case of multiple DHCP servers on subnet)
  - When a DHCP server receives a DHCP request with server identifier not being itself, it stops there
- DHCPREQUEST messages can be generated without prior DHCPDISCOVER/DHCPOFFER interaction
  - Upon renewing the lease (unicast), or rebinding (broadcast: try other DHCP servers for renewing)
- If the client receives DHCPACK and detects that the address is already in use (e.g., through the use of ARP), the client MUST send a DHCPDECLINE message to the server and restarts the configuration process
  - DHCP servers should NOT check for IP conflicts. However, some server implementations do so by using ICMP pings

-Typically the client is responsible for checking IP conflicts, not the server (this is different than most other network protocol servers where they check validity for you)

-DHCP as a service offers more than IP addresses

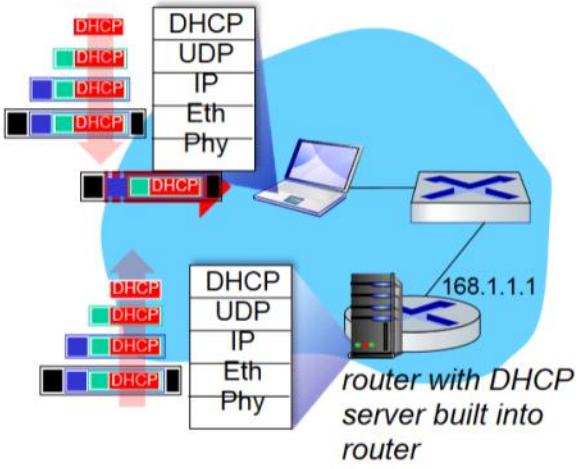
DHCP: more than IP addresses

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

- address of first-hop router (gateway) for client
- name and IP address of '**local**' DNS sever
- network mask (indicating network versus host portion of address)

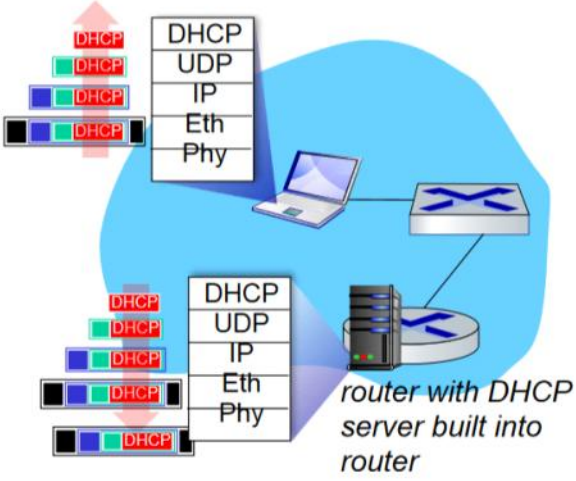
-How DHCP works

DHCP: example



- Connecting laptop will use DHCP to get IP address, address of first-hop router, address of DNS server.
- DHCP REQUEST message encapsulated in UDP, encapsulated in IP, encapsulated in Ethernet
- Ethernet frame broadcast (dest: FFFFFFFFFFFF) on LAN, received at router running DHCP server
- Ethernet demux'ed to IP demux'ed, UDP demux'ed to DHCP

DHCP: example



- DCP server formulates DHCP ACK containing client's IP address, IP address of first-hop router for client, name & IP address of DNS server
- encapsulated DHCP server reply forwarded to client, demuxing up to DHCP at client
- client now knows its IP address, name and IP address of DNS server, IP address of its first-hop router

-DHCP uses UDP instead of TCP because we don't want to send 250 requests to get the correct IP address with the overhead of a TCP connection

-"TCP/IP" is usually grouped together in settings as a catchall phrase for anything that's below the application layer in computer networks

-More on how you get your IP address

-Your ISP gives you some subnet (in this case below, /20), since 20 bits are subnet, we know the number of possible host IP addresses is 32 bits in IP - 20 bits used for subnet = 12 bits for host so 2 ^12 - 2 (subtract out network address and broadcast address) is the number of possible host IP addresses

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

-Even ISPs have to do their own subnetting to distribute ISPs to their organizations

-You want your gateways from your ISP to be as few as possible because IP addresses are expensive

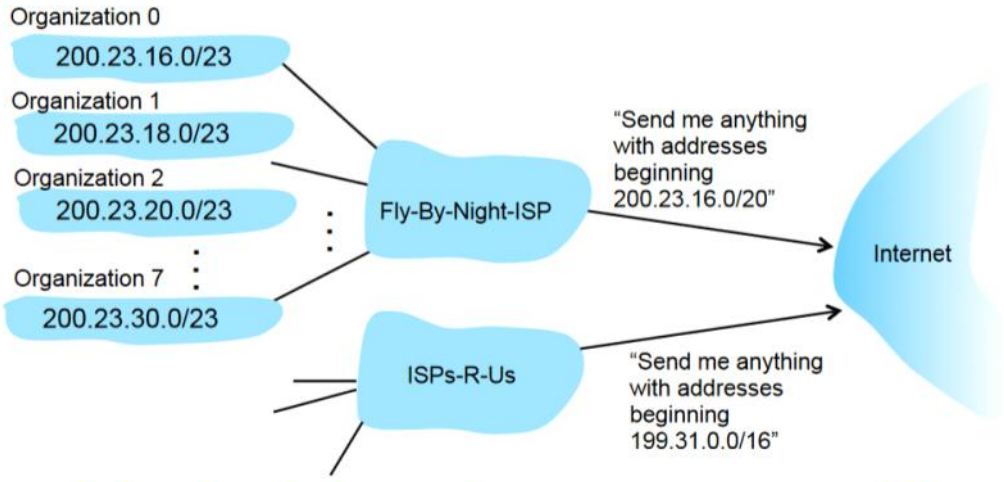
10/31 - NAT, IPv6

Tuesday, October 29, 2024 6:17 PM

-Hierarchical addressing

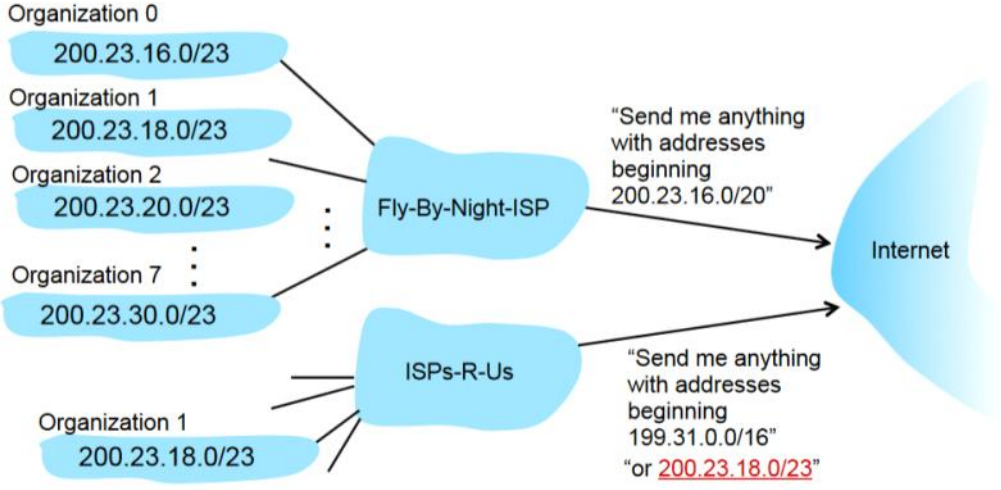
Hierarchical addressing: route aggregation

hierarchical addressing allows efficient advertisement of routing information:



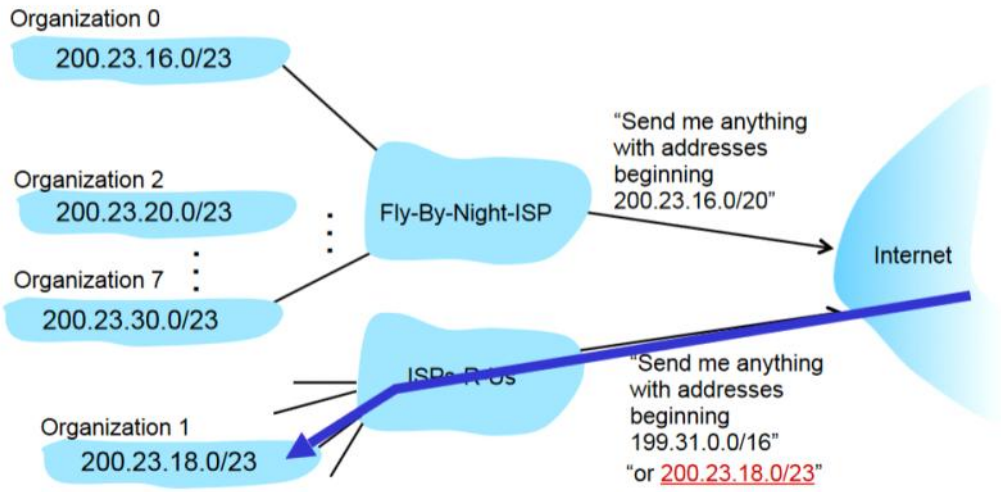
Hierarchical addressing: more specific routes

- **Organization 1 moves from Fly-By-Night-ISP to ISPs-R-Us**
- ISPs-R-Us now advertises a more specific route to Organization 1



Hierarchical addressing: more specific routes

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



-We don't have enough IP addresses with 2^32 IP addresses, that's why we have local IP addresses (NAT) and IPv6 (2^128 IP addresses), especially with IoT proponents

-ICANN (Internet Corporation for Assigned Names and Numbers) handles the registry of IP addresses

-Vint Cerf - the guy who invented the Internet, said "who the hell knew how much address space we needed" in retrospect

IP addressing: last words ...

Q: how does an ISP get block of addresses?

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

- allocates IP addresses, through 5 regional registries (RRs) (who may then allocate to local registries)
- manages DNS root zone, including delegation of individual TLD (.com, .edu , ...) management

Q: are there enough 32-bit IP addresses?

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

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

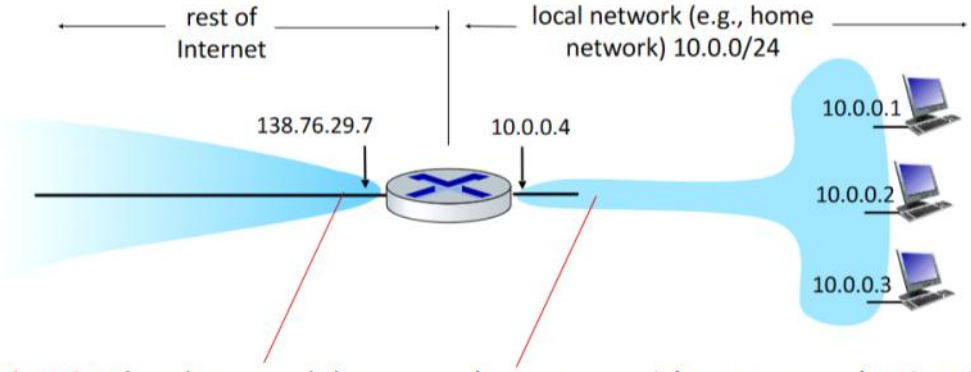
-NAT = Network Address Translation

-128.0.*, 192.168.*, 10.* are all designated as local addresses, all other addresses are facing the Internet

-All IP datagrams leaving your home use your router's global IP address, not your computer's IP address

NAT: network address translation

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



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

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

NAT: network address translation

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

-We use NAT to hide the existence of local devices also to not have to change your local IP address if you get a new global IP address from your ISP

NAT: network address translation

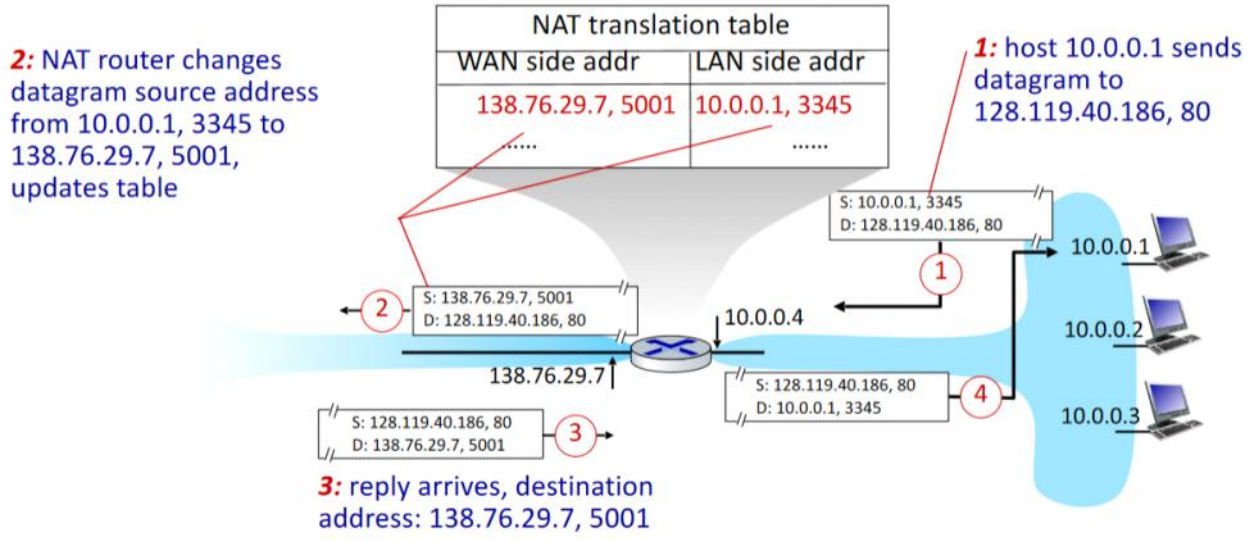
implementation: NAT router must (transparently):

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

-We have a NAT table

-Same IP address but different port number are used to connect over the Internet

NAT: network address translation



-An issue with NAT is that you have to do port forwarding, giving your router manual entries in the NAT table for other computers

-Another issue with NAT is that routers are dealing with port numbers (transport layer), violating separating of concern

NAT: network address translation

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

-IPv6

IPv6: motivation

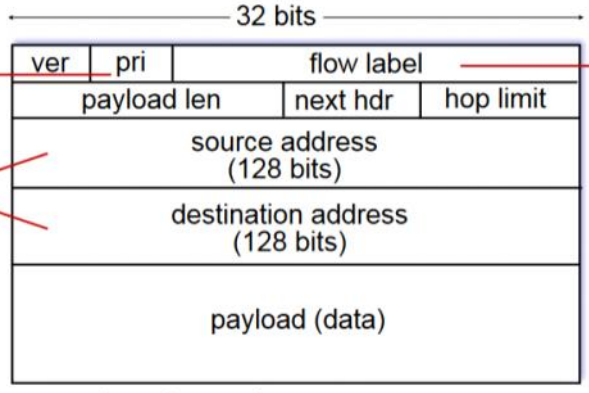
- initial motivation: 32-bit IPv4 address space would be completely allocated
- additional motivation:
  - speed processing/forwarding: 40-byte fixed length header
  - enable different network-layer treatment of "flows"

-40 byte fixed length header in IPv6 (more overhead than IPv4)
-IPv6 datagram

IPv6 datagram format

priority: identify priority among datagrams in flow

128-bit IPv6 addresses



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

What's missing (compared with IPv4):

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

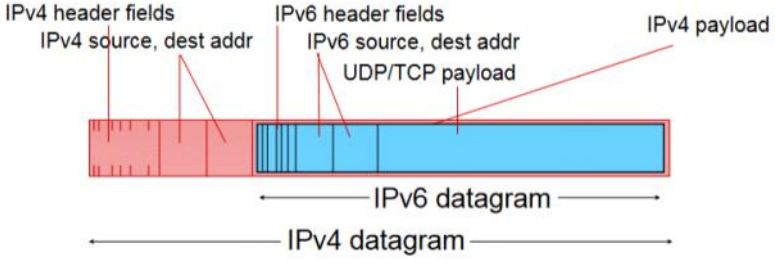
-Next hdr - equivalent to ULP (upper layer protocol) in IPv4, tells you if you're using UDP or TCP for demultiplexing
-Hop limit - equivalent to TTL (time to live) in IPv4, prevents routing loops
-Ver - same as IPv4 version header

-Pri - priority, related to network neutrality for which packet should be routed in the IPv6 flow, for more advanced courses in networks
-Flow label - they dropped all useless fields in IPv6 that were in IPv4, including fragmentation, fragmentation was replaced by "flow", but this is a more advanced topic
-No more options like in IPv4, our header size is fixed in IPv6
-No checksum field in IPv6 because it was deemed redundant since UDP and TCP already perform checksum

-Tunneling - an IPv6 datagram carried as a payload in an IPv4 datagram among IPv4 routers
-Not everyone uses IPv6, 60-70% of the Internet uses IPv6 and ISPs don't want to spend a lot of money to update hardware that only supports IPv4

Transition from IPv4 to IPv6

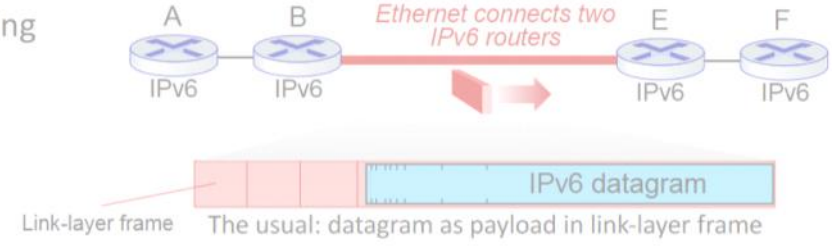
- not all routers can be upgraded simultaneously
  - no "flag days"
  - how will network operate with mixed IPv4 and IPv6 routers?
- tunneling: IPv6 datagram carried as *payload* in IPv4 datagram among IPv4 routers ("packet within a packet")
  - tunneling used extensively in other contexts (4G/5G)



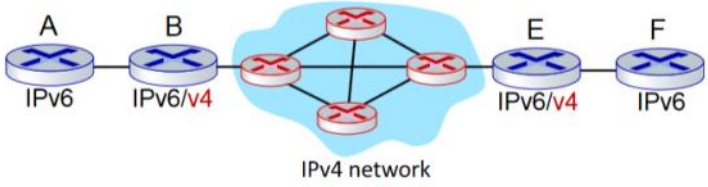
-More about tunneling

Tunneling and encapsulation

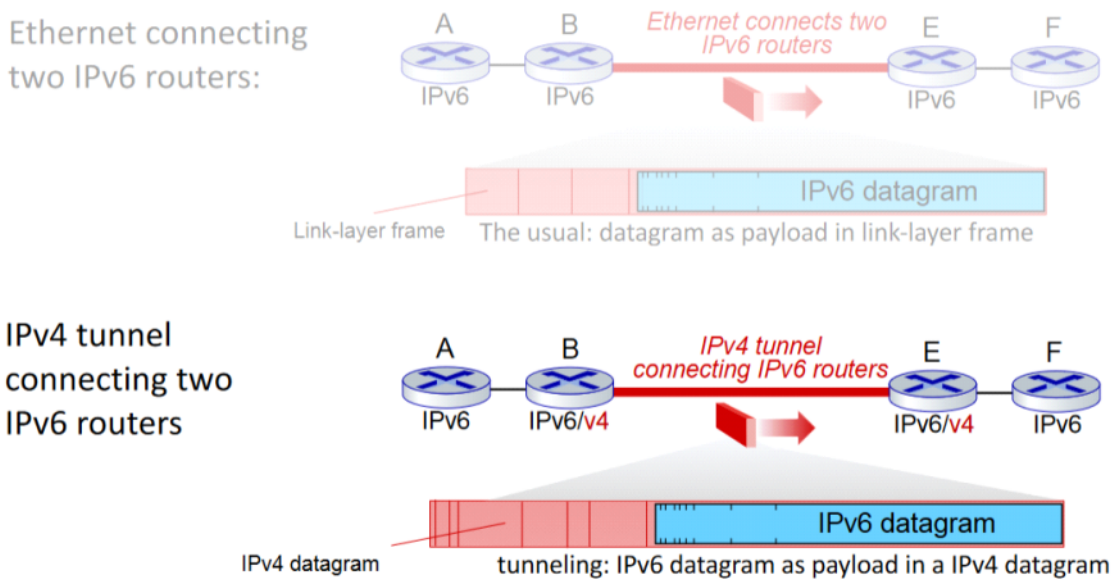
Ethernet connecting two IPv6 routers:



IPv4 network connecting two IPv6 routers

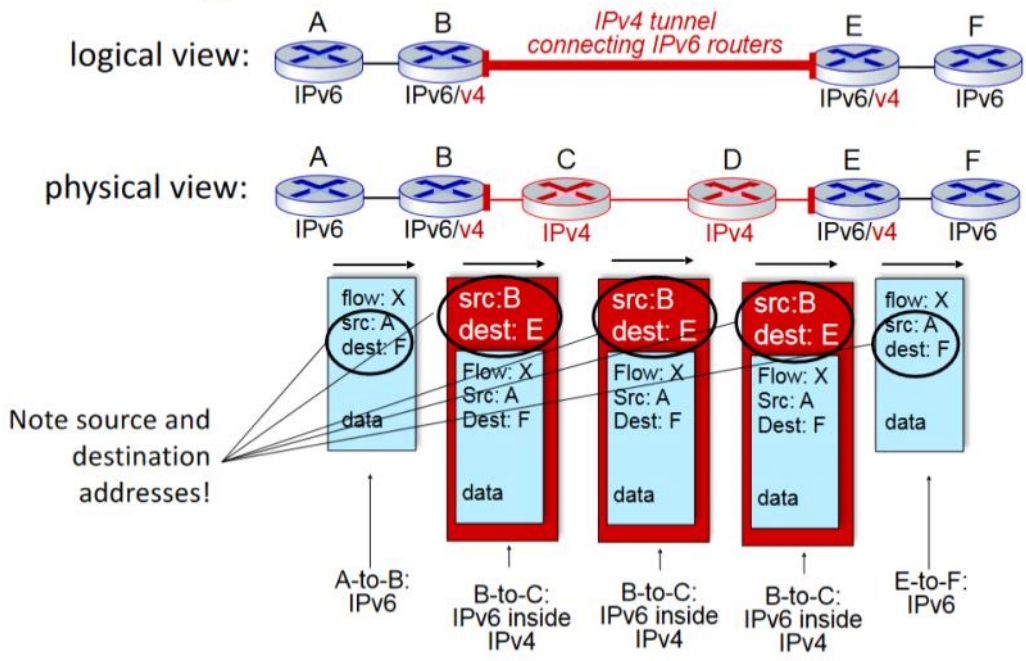


Tunneling and encapsulation



-Tunneling diagram example

Tunneling



-Use of IPv6 is growing

IPv6: adoption

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

[1] https://www.google.com/intl/en/ipv6/statistics.html

- Long (long!) time for deployment, use
  - 25 years and counting!
  - think of application-level changes in last 25 years: WWW, social media, streaming media, gaming, telepresence, …
  - *Why?*

10/31 - traceroute (ICMP)

Thursday, October 31, 2024 5:56 PM

-We just talked about forwarding in Chapter 4 with the data plane, now let's talk about routing in Chapter 5 with the control plane

Network-layer functions

- forwarding: move packets from router's input to appropriate router output — *data plane*
- routing: determine route taken by packets from source to destination — *control plane*

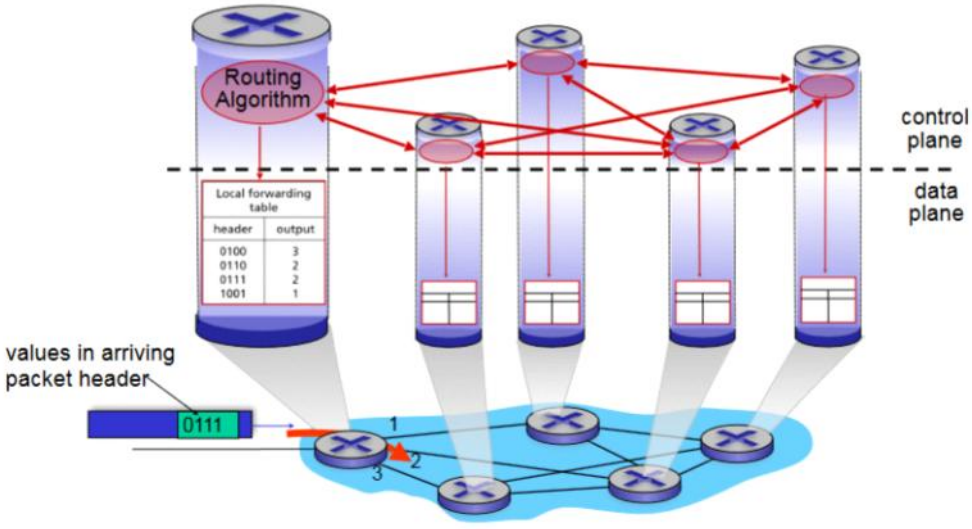
Two approaches to structuring network control plane:

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

-How the control plane works

Per-router control plane

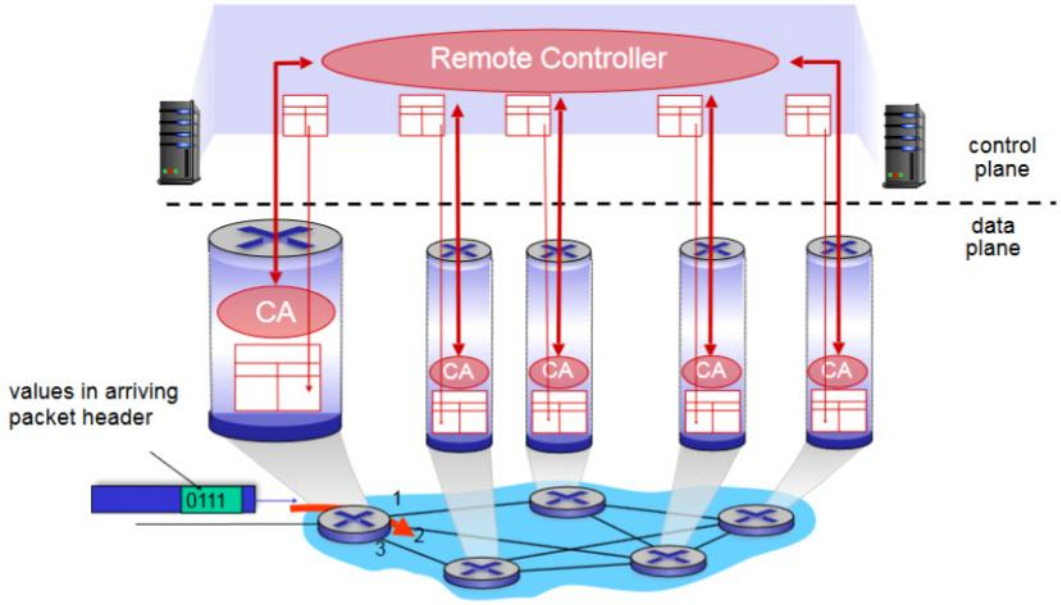
Individual routing algorithm components *in each and every router* interact in the control plane



-SDN - beyond the scope of this class

Software-Defined Networking (SDN) control plane

Remote controller computes, installs forwarding tables in routers



-ICMP - this is how "ping" works, ICMP is a network layer protocol (unlike DHCP, which is application layer but helpful to the network layer) an IP datagram carries it

-ICMP has a type and code in its header and that's it

ICMP: internet control message protocol

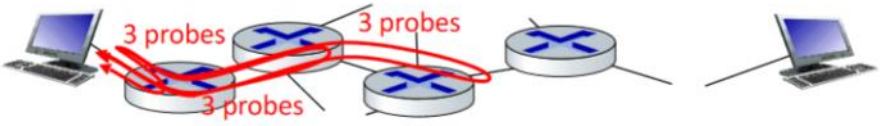
- used by hosts and routers to communicate network-level information
  - error reporting: unreachable host, network, port, protocol
  - echo request/reply (used by ping)
- network-layer "above" IP:
  - ICMP messages carried in IP datagrams
- *ICMP message:* type, code plus first 8 bytes of IP datagram causing error

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

-How "traceroute" works

Traceroute measures delays using ICMP

- what do "real" Internet delay & loss look like?
- **traceroute** program: provides delay measurement from source to router along end-end Internet path towards destination. For all *i:*
  - sends three packets that will reach router *i* on path towards destination (with time-to-live field value of *i*)
  - router *i* will return packets to sender
  - sender measures time interval between transmission and reply



Traceroute: Real Internet delays and routes

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

3 end-to-end delay measurements ($d_{proc}+d_{queue}+d_{trans}+d_{prop}$) from gaia.cs.umass.edu to cs-gw.cs.umass.edu

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

3 delay measurements to border1-rt-fa5-1-0.gw.umass.edu

trans-oceanic link

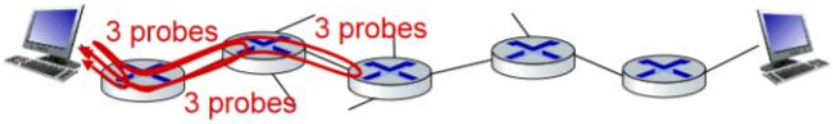
looks like delays *decrease*! Why?

—> Queuing delay larger for packets sent to router 10

* means no response (probe lost, router not replying)

* Do some traceroutes from exotic countries at www.traceroute.org

Traceroute and ICMP



- source sends sets of UDP segments to destination
  - 1st set has TTL =1, 2nd set has TTL=2, etc.
- datagram in *n*th set arrives to nth router:
  - router discards datagram and sends source ICMP message (**type 11, code 0**)
  - ICMP message possibly includes name of router & IP address
- when ICMP message arrives at source: record RTTs

stopping criteria:

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

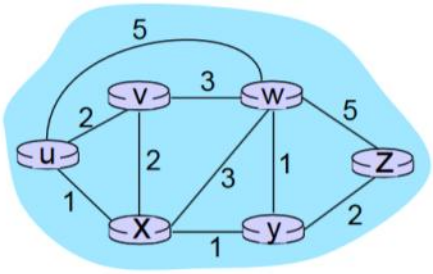
-"ping" checks if the server is up and running, "traceroute" checks the path (which routers you hit)

11/7 - Link-State

Thursday, October 31, 2024 6:17 PM

-Routing protocols
- -Routing protocol goal - determine "good" paths (AKA routes), from sending host to receiving host, through network of routers
- -Good route - decrease number of hops (number of times we hit a router); lowest cost path
- -Networks are graphs:

Graph abstraction: link costs



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

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

cost defined by network operator: could always be 1, or inversely related to bandwidth, or inversely related to congestion

graph: $G = (N,E)$

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

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

-Types of routing algorithms

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?

15

- -Global vs decentralized - what do their neighbors think about the path?
- -Global routing algorithm (AKA link state algorithm) gets the information of all routers on the network
- -Decentralized routing algorithm (distance vector algorithm) only gets the information of the neighboring routers
- -Static routing algorithm - if costs don't change often
- -Dynamic routing algorithm - costs change often

-Link state (global) routing algorithm
- -Lots of overhead, any time there's a change, we have to update a lot of paths

-Uses link state routing algorithm uses Dijkstra's shortest path algorithm

Dijkstra's link-state routing algorithm

- centralized: network topology, link costs known to ***all*** nodes
  - accomplished via "link state broadcast"
  - all nodes have same info
- computes least cost paths from one node ("source") to all other nodes
  - gives *forwarding table* for that node
- iterative: after *k* iterations, know least cost path to *k* destinations

notation

- $c_{x,y}$: direct link cost from node *x* to *y*; $= \infty$ if not direct neighbors
- *D(v): current* estimate of cost of least-cost-path from source to destination *v*
- *p(v):* predecessor node along path from source to *v*
- *N':* set of nodes whose least-cost-path *definitively* known

-In Dijkstra's algorithm, we need to know all the links/edges, that's why we call it the "link state" algorithm, accomplished via a "link state broadcast"
-Dijkstra's notation: $c_{x,y}$ is the cost of the links, D(v) is the estimate of cost or current distance, p(v) the previous node visited, N' is the set of nodes we've visited

Dijkstra's link-state routing algorithm

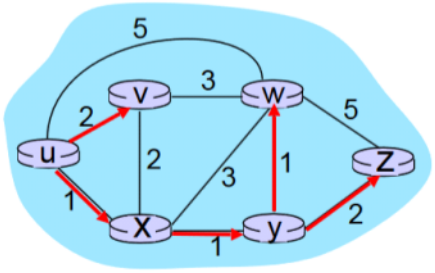
```
Initialization:
  N' = {u}                          /* compute least cost path from u to all other nodes */
  for all nodes v
    if v adjacent to u              /* u initially knows direct-path-cost only to direct neighbors */
      then D(v) = c_{u,v}           /* but may not be minimum cost! */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

-Dijkstra's algorithm example in networks

Dijkstra's algorithm: an example

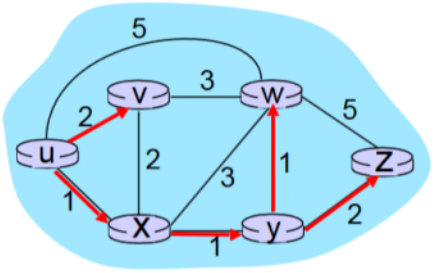
Step	N'	D(v),p(v)	D(w),p(w)	D(x),p(x)	D(y),p(y)	D(z),p(z)
0	u	2,u	5,u	1,u	∞	∞
1	ux	2,u	4,x		2,x	∞
2	uxy	2,u	3,y			4,y
3	uxyv		3,y			4,y
4	uxyvw					4,y
5	uxyvwz					



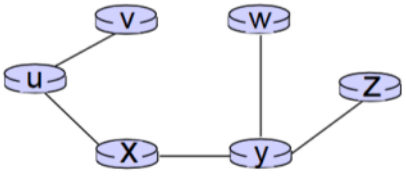
Initialization (step 0): For all a: if *a* adjacent to then $D(a) = c_{u,a}$

find *a* not in *N'* such that *D(a)* is a minimum
add *a* to *N'*
update *D(b)* for all *b* adjacent to *a* and not in *N'* :
$D(b) = \min (D(b), D(a) + c_{a,b})$

Dijkstra's algorithm: an example



resulting least-cost-path tree from u:



resulting forwarding table in u:

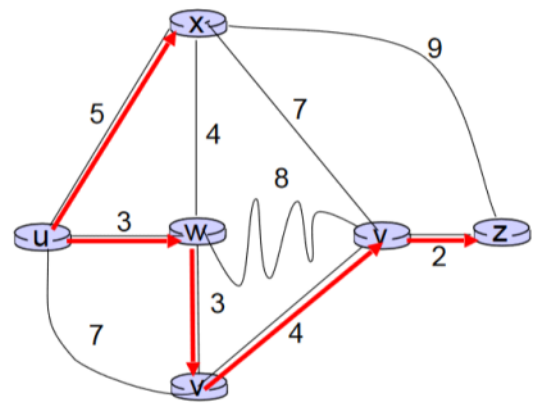
destination	outgoing link
v	(u,v)
x	(u,x)
y	(u,x)
w	(u,x)
x	(u,x)

route from *u* to *v* directly

route from u to all other destinations via *x*

Dijkstra's algorithm: another example

Step	N'	v D(v), p(v)	w D(w), p(w)	x D(x), p(x)	y D(y), p(y)	z D(z), p(z)
0	u	7,u	3,u	5,u	∞	∞
1	uw	6,w		5,u	11,w	∞
2	uwx	6,w			11,w	14,x
3	uwxv				10,v	14,x
4	uwxvy					12,y
5	uwxvyz					



notes:

- construct least-cost-path tree by tracing predecessor nodes
- ties can exist (can be broken arbitrarily)

-Aside: Dijkstra is not practical for representing the entire Internet, so we represent distance to some subnet

-Dijkstra's shortest path algorithm analysis

-Dijkstra's shortest path algorithm runs in $O(n^2)$, which is bad, because we have to find all nodes after each iteration

-You can make Dijkstra's shortest path algorithm faster using a heap data structure to get O(n log n), but this still isn't good

-Broadcasting algorithms run in O(1) so we need to catch up

-Since every router has to broadcast at least once O(n), then the overall message complexity is $O(n^2)$

Dijkstra's algorithm: discussion

algorithm complexity: How much computation needed to find path from source to all destinations?

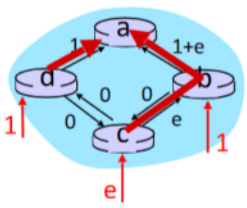
- **Each iteration**: need to check all nodes, *w*, not in *N (we have n nodes in total)*
- *1st iter: n checks, 2nd iter: n-1 checks, 3rd iter: n-2 checks …*
  - $n(n+1)/2$ comparisons: **$O(n^2)$** complexity
- more efficient implementations (e.g., using heap data structure) are possible: $O(n\log n)$

message complexity:

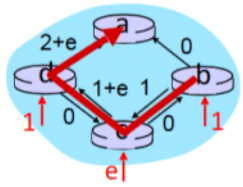
- each router must ***broadcast*** its link state information to other *n* routers
- efficient (and interesting!) broadcast algorithms: $O(n)$ link crossings to disseminate a broadcast message from one source
- **each router**'s message crosses **$O(n)$** links: **overall** message complexity: **$O(n^2)$**

Dijkstra's algorithm: oscillations possible

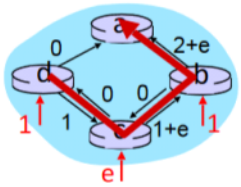
- when link costs depend on traffic volume, route oscillations possible
  - sample scenario:
    - routing to destination a, traffic entering at d, c, b with rates 1, e (<1), 1
      - Node d originates **1** unit of traffic **destined to node a**
      - Node c injects an amount of traffic equals to **e** and **destined to node a**
      - Node b originates **1** unit of traffic **destined to node a**
    - link costs are directional, and volume-dependent
- **Takeaway**: newer routing algorithms tend to avoid using short term congestion control metrics as link cost



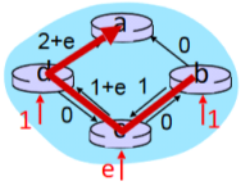
initially



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



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



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

-Oscillation problem happens in Dijkstra's shortest path algorithm because the cost of links keep changing, so a large amount of traffic causes router oscillation to infinity (this is why modern routers don't measure network traffic or short-term congestion control as a metric for link/edge cost)

HW4 - Question 5

Saturday, November 9, 2024 6:55 PM

Node z table

	cost to				
	z	v	x	y	u
z	0	6	2	∞	∞
v	∞	∞	∞	∞	∞
x	∞	∞	∞	∞	∞
y	∞	∞	∞	∞	∞
u	∞	∞	∞	∞	∞

		cost to				
		z	v	x	y	u
from	z	0	5	2	5	6
	v	6	0	3	∞	1
	x	2	3	0	3	∞
	y	∞	∞	3	0	2
	u	∞	1	∞	2	0

		cost to				
		z	v	x	y	u
from	z	0	5	2	5	6
	v	5	0	3	3	1
	x	2	3	0	3	4
	y	5	3	3	0	2
	u	6	1	4	2	0

Time

11/12 - Distance Vector

Tuesday, November 12, 2024 5:03 PM

-Distance vector (decentralized) routing algorithm
    -It uses the Bellman-Ford algorithm and dynamic programming

Distance vector algorithm

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

Bellman-Ford equation

Let $D_x(y)$: cost of least-cost path from x to y.
Then:

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

v's estimated least-cost-path cost to y

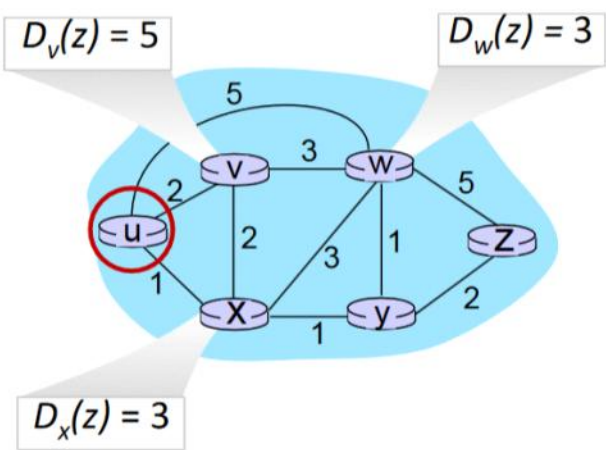
min taken over all **neighbors *v*** of *x*

direct cost of link from x to v

-With the distance vector algorithm, all we have to know is our neighbor's cost
    -Dynamic programming - trying to optimize paths using local optimizations
-Bellman-Ford / distance vector routing algorithm example

Bellman-Ford Example

Suppose that *u*'s neighboring nodes, *x,v,w,* know that for destination *z*:



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)

-Distance vector routing algorithm (Bellman Ford) overview

Distance vector algorithm

key idea:

- from time-to-time, each node sends its own distance vector estimate to neighbors
- when *x* receives new DV estimate from any neighbor, it updates its own DV using B-F equation:

$$D_x(y) \leftarrow min_v\{c_{x,v} + D_v(y)\} \text{ for each node } y \in N$$

- 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*
- If no notification received, no actions taken!

-Distance vector routing algorithm example

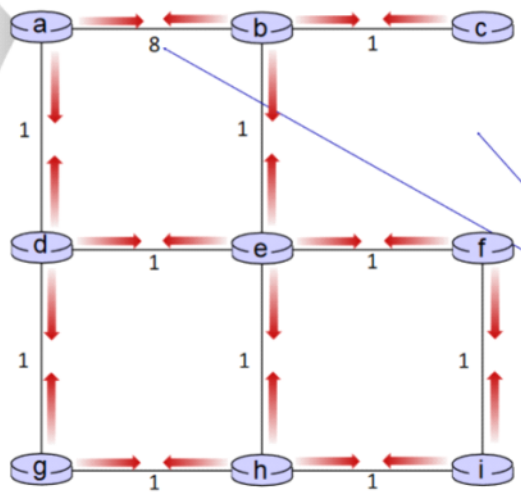
Distance vector: example

t=0

- All nodes have distance estimates to nearest neighbors (only)
- All nodes send their local distance vector to their neighbors

DV in a:

$D_a(a)=0$
$D_a(b) = 8$
$D_a(c) = \infty$
$D_a(d) = 1$
$D_a(e) = \infty$
$D_a(f) = \infty$
$D_a(g) = \infty$
$D_a(h) = \infty$
$D_a(i) = \infty$

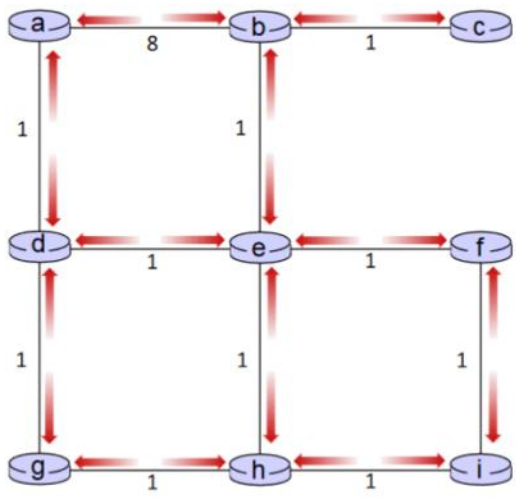


A few asymmetries:
- missing link
- larger cost

Distance vector example: iteration

t=1

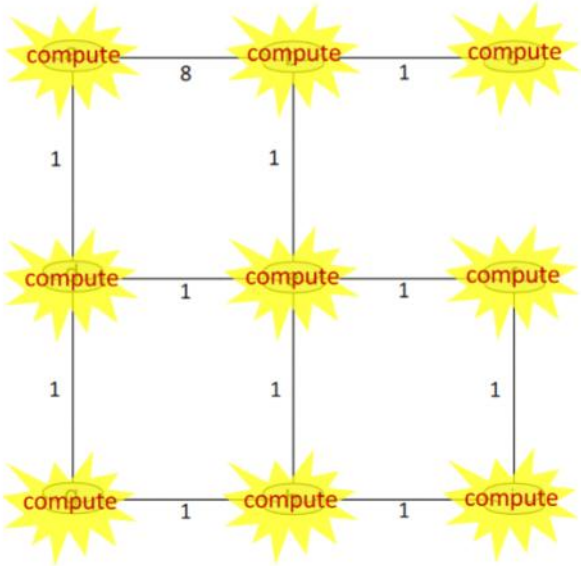
All nodes:
- receive distance vectors from neighbors
- compute their new local distance vector
- send their new local distance vector to neighbors



Distance vector example: iteration

t=1

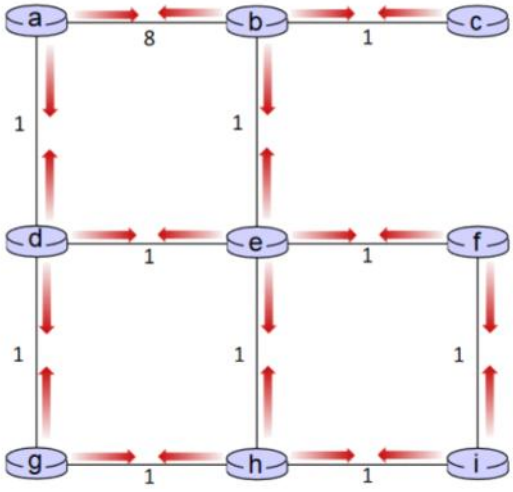
All nodes:
- receive distance vectors from neighbors
- compute their new local distance vector
- send their new local distance vector to neighbors



Distance vector example: iteration

t=1

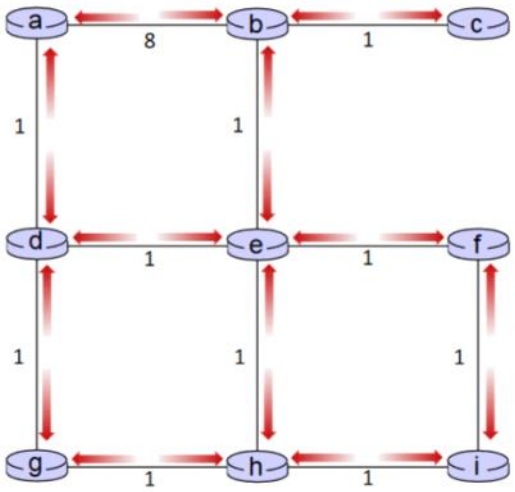
All nodes:
- receive distance vectors from neighbors
- compute their new local distance vector
- send their new local distance vector to neighbors



Distance vector example: iteration

t=2

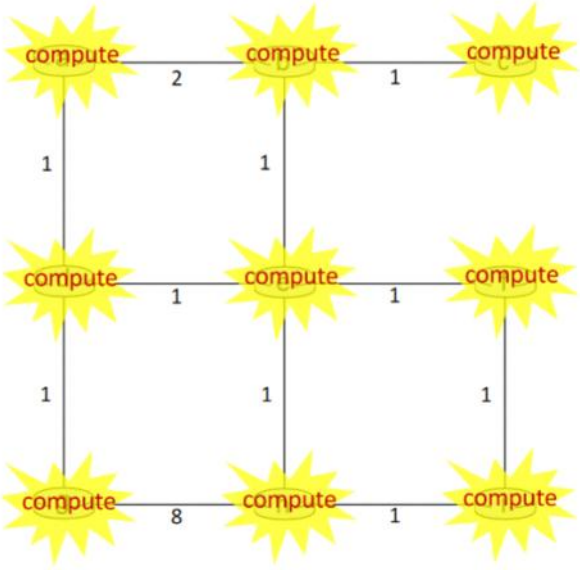
All nodes:
- receive distance vectors from neighbors
- compute their new local distance vector
- send their new local distance vector to neighbors



Distance vector example: iteration

t=2

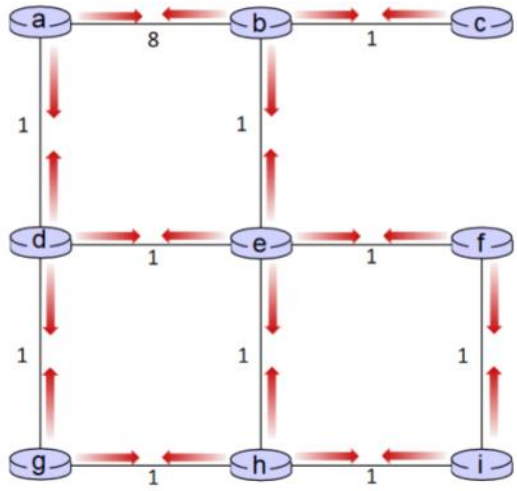
All nodes:
- receive distance vectors from neighbors
- compute their new local distance vector
- send their new local distance vector to neighbors



Distance vector example: iteration

t=2

All nodes:
- receive distance vectors from neighbors
- compute their new local distance vector
- send their new local distance vector to neighbors



Distance vector example: iteration

.... and so on

Let's next take a look at the iterative *computations* at nodes

Distance vector example:

t=1

- b receives DVs from a, c, e

DV in a:

$D_a(a)=0$
$D_a(b) = 8$
$D_a(c) = \infty$
$D_a(d) = 1$
$D_a(e) = \infty$
$D_a(f) = \infty$
$D_a(g) = \infty$
$D_a(h) = \infty$
$D_a(i) = \infty$

DV in b:

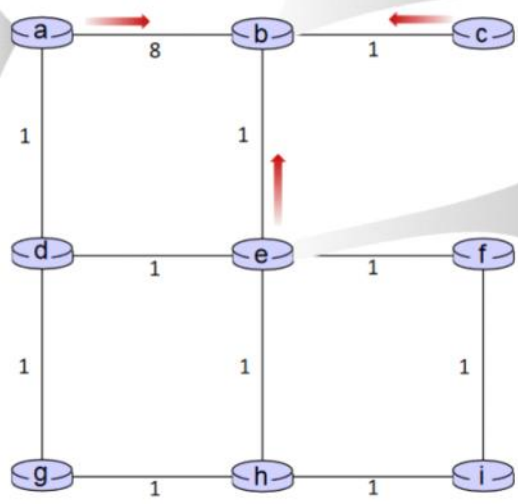
$D_b(a) = 8$ $D_b(f) = \infty$
$D_b(c) = 1$ $D_b(g) = \infty$
$D_b(d) = \infty$ $D_b(h) = \infty$
$D_b(e) = 1$ $D_b(i) = \infty$

DV in c:

$D_c(a) = \infty$
$D_c(b) = 1$
$D_c(c) = 0$
$D_c(d) = \infty$
$D_c(e) = \infty$
$D_c(f) = \infty$
$D_c(g) = \infty$
$D_c(h) = \infty$
$D_c(i) = \infty$

DV in e:

$D_e(a) = \infty$
$D_e(b) = 1$
$D_e(c) = \infty$
$D_e(d) = 1$
$D_e(e) = 0$
$D_e(f) = 1$
$D_e(g) = \infty$
$D_e(h) = 1$
$D_e(i) = \infty$



Distance vector example:

t=1

- b receives DVs from a, c, e, computes:

DV in a:

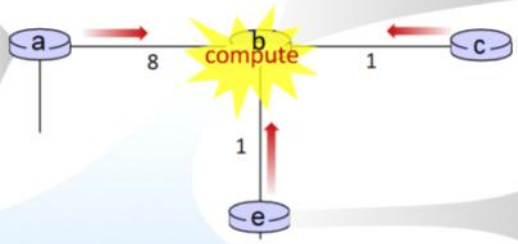
$D_a(a)=0$
$D_a(b) = 8$
$D_a(c) = \infty$
$D_a(d) = 1$
$D_a(e) = \infty$
$D_a(f) = \infty$
$D_a(g) = \infty$
$D_a(h) = \infty$
$D_a(i) = \infty$

DV in b:

$D_b(a) = 8$	$D_b(f) = \infty$
$D_b(c) = 1$	$D_b(g) = \infty$
$D_b(d) = \infty$	$D_b(h) = \infty$
$D_b(e) = 1$	$D_b(i) = \infty$

DV in c:

$D_c(a) = \infty$
$D_c(b) = 1$
$D_c(c) = 0$
$D_c(d) = \infty$
$D_c(e) = \infty$
$D_c(f) = \infty$
$D_c(g) = \infty$
$D_c(h) = \infty$
$D_c(i) = \infty$



DV in e:

$D_e(a) = \infty$
$D_e(b) = 1$
$D_e(c) = \infty$
$D_e(d) = 1$
$D_e(e) = 0$
$D_e(f) = 1$
$D_e(g) = \infty$
$D_e(h) = 1$
$D_e(i) = \infty$

$D_b(a) = \min\{c_{b,a}+D_a(a),\ c_{b,c}+D_c(a),\ c_{b,e}+D_e(a)\} = \min\{8,\infty,\infty\} = 8$

$D_b(c) = \min\{c_{b,a}+D_a(c),\ c_{b,c}+D_c(c),\ c_{b,e}+D_e(c)\} = \min\{\infty,1,\infty\} = 1$

$D_b(d) = \min\{c_{b,a}+D_a(d),\ c_{b,c}+D_c(d),\ c_{b,e}+D_e(d)\} = \min\{9,\infty,2\} = 2$

$D_b(e) = \min\{c_{b,a}+D_a(e),\ c_{b,c}+D_c(e),\ c_{b,e}+D_e(e)\} = \min\{\infty,\infty,1\} = 1$

$D_b(f) = \min\{c_{b,a}+D_a(f),\ c_{b,c}+D_c(f),\ c_{b,e}+D_e(f)\} = \min\{\infty,\infty,2\} = 2$

$D_b(g) = \min\{c_{b,a}+D_a(g),\ c_{b,c}+D_c(g),\ c_{b,e}+D_e(g)\} = \min\{\infty,\infty,\infty\} = \infty$

$D_b(h) = \min\{c_{b,a}+D_a(h),\ c_{b,c}+D_c(h),\ c_{b,e}+D_e(h)\} = \min\{\infty,\infty,2\} = 2$

$D_b(i) = \min\{c_{b,a}+D_a(i),\ c_{b,c}+D_c(i),\ c_{b,e}+D_e(i)\} = \min\{\infty,\infty,\infty\} = \infty$

DV in b:

$D_b(a) = 8$	$D_b(f) = 2$
$D_b(c) = 1$	$D_b(g) = \infty$
$D_b(d) = 2$	$D_b(h) = 2$
$D_b(e) = 1$	$D_b(i) = \infty$

Distance vector example:

t=1

- c receives DVs from b

DV in a:

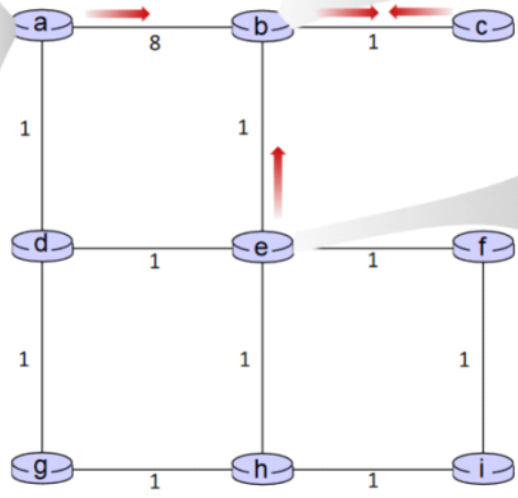
$D_a(a)=0$
$D_a(b) = 8$
$D_a(c) = \infty$
$D_a(d) = 1$
$D_a(e) = \infty$
$D_a(f) = \infty$
$D_a(g) = \infty$
$D_a(h) = \infty$
$D_a(i) = \infty$

DV in b:

$D_b(a) = 8$	$D_b(f) = \infty$
$D_b(c) = 1$	$D_b(g) = \infty$
$D_b(d) = \infty$	$D_b(h) = \infty$
$D_b(e) = 1$	$D_b(i) = \infty$

DV in c:

$D_c(a) = \infty$
$D_c(b) = 1$
$D_c(c) = 0$
$D_c(d) = \infty$
$D_c(e) = \infty$
$D_c(f) = \infty$
$D_c(g) = \infty$
$D_c(h) = \infty$
$D_c(i) = \infty$



DV in e:

$D_e(a) = \infty$
$D_e(b) = 1$
$D_e(c) = \infty$
$D_e(d) = 1$
$D_e(e) = 0$
$D_e(f) = 1$
$D_e(g) = \infty$
$D_e(h) = 1$
$D_e(i) = \infty$

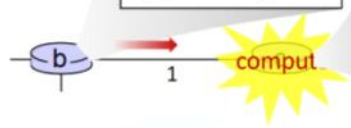
Distance vector example:

DV in b:

$D_b(a) = 8$	$D_b(f) = \infty$
$D_b(c) = 1$	$D_b(g) = \infty$
$D_b(d) = \infty$	$D_b(h) = \infty$
$D_b(e) = 1$	$D_b(i) = \infty$

DV in c:

$D_c(a) = \infty$
$D_c(b) = 1$
$D_c(c) = 0$
$D_c(d) = \infty$
$D_c(e) = \infty$
$D_c(f) = \infty$
$D_c(g) = \infty$
$D_c(h) = \infty$
$D_c(i) = \infty$



t=1

- c receives DVs from b computes:

$D_c(a) = \min\{c_{c,b}+D_b(a)\} = 1 + 8 = 9$
$D_c(b) = \min\{c_{c,b}+D_b(b)\} = 1 + 0 = 1$
$D_c(d) = \min\{c_{c,b}+D_b(d)\} = 1+ \infty = \infty$
$D_c(e) = \min\{c_{c,b}+D_b(e)\} = 1 + 1 = 2$
$D_c(f) = \min\{c_{c,b}+D_b(f)\} = 1+ \infty = \infty$
$D_c(g) = \min\{c_{c,b}+D_b(g)\} = 1+ \infty = \infty$
$D_c(h) = \min\{c_{bc,b}+D_b(h)\} = 1+ \infty = \infty$
$D_c(i) = \min\{c_{c,b}+D_b(i)\} = 1+ \infty = \infty$

DV in c:

$D_c(a) = 9$
$D_c(b) = 1$
$D_c(c) = 0$
$D_c(d) = \infty$
$D_c(e) = 2$
$D_c(f) = \infty$
$D_c(g) = \infty$
$D_c(h) = \infty$
$D_c(i) = \infty$

* Check out the online interactive exercises for more examples: http://gaia.cs.umass.edu/kurose_ross/interactive/

Distance vector example:

DV in b:

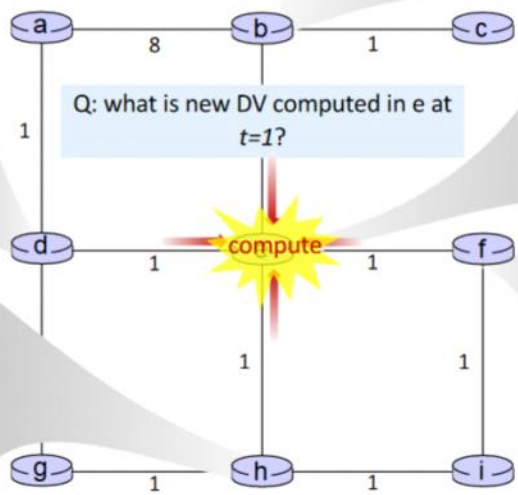
$D_b(a) = 8$	$D_b(f) = \infty$
$D_b(c) = 1$	$D_b(g) = \infty$
$D_b(d) = \infty$	$D_b(h) = \infty$
$D_b(e) = 1$	$D_b(i) = \infty$

DV in d:

$D_d(a) = 1$
$D_d(b) = \infty$
$D_d(c) = \infty$
$D_d(d) = 0$
$D_d(e) = 1$
$D_d(f) = \infty$
$D_d(g) = 1$
$D_d(h) = \infty$
$D_d(i) = \infty$

DV in e:

$D_e(a) = \infty$
$D_e(b) = 1$
$D_e(c) = \infty$
$D_e(d) = 1$
$D_e(e) = 0$
$D_e(f) = 1$
$D_e(g) = \infty$
$D_e(h) = 1$
$D_e(i) = \infty$



t=1

- e receives DVs from b, d, f, h

DV in h:

$D_h(a) = \infty$
$D_h(b) = \infty$
$D_h(c) = \infty$
$D_h(d) = \infty$
$D_h(e) = 1$
$D_h(f) = \infty$
$D_h(g) = 1$
$D_h(h) = 0$
$D_h(i) = 1$

DV in f:

$D_f(a) = \infty$
$D_f(b) = \infty$
$D_f(c) = \infty$
$D_f(d) = \infty$
$D_f(e) = 1$
$D_f(f) = 0$
$D_f(g) = \infty$
$D_f(h) = \infty$
$D_f(i) = 1$

-Efficient way to compute distance vector (Bellman-Ford), used in homework

$D_x(y) = \min\{c(x,y) + D_y(y),\ c(x,z) + D_z(y)\}$
$= \min\{2+0\ ,\ 7+1\} = 2$

$D_x(z) = \min\{c(x,y) + D_y(z),\ c(x,z) + D_z(z)\}$
$= \min\{2+1\ ,\ 7+0\} = 3$

node x table

from \ cost to	x	y	z
x	0	2	7
y	∞	∞	∞
z	∞	∞	∞

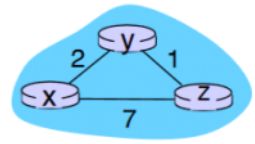
from \ cost to	x	y	z
x	0	2	3
y	2	0	1
z	7	1	0

node y table

from \ cost to	x	y	z
x	∞	∞	∞
y	2	0	1
z	∞	∞	∞

node z table

from \ cost to	x	y	z
x	∞	∞	∞
y	∞	∞	∞
z	7	1	0



time

42

$D_x(y) = \min\{c(x,y) + D_y(y),\ c(x,z) + D_z(y)\}$
$= \min\{2+0\ ,\ 7+1\} = 2$

$D_x(z) = \min\{c(x,y) + D_y(z),\ c(x,z) + D_z(z)\}$
$= \min\{2+1\ ,\ 7+0\} = 3$

node x table

from \ cost to	x	y	z
x	0	2	7
y	∞	∞	∞
z	∞	∞	∞

from \ cost to	x	y	z
x	0	2	3
y	2	0	1
z	7	1	0

from \ cost to	x	y	z
x	0	2	3
y	2	0	1
z	3	1	0

node y table

from \ cost to	x	y	z
x	∞	∞	∞
y	2	0	1
z	∞	∞	∞

from \ cost to	x	y	z
x	0	2	7
y	2	0	1
z	7	1	0

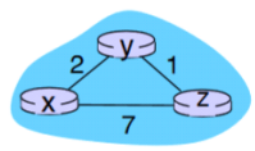
from \ cost to	x	y	z
x	0	2	3
y	2	0	1
z	3	1	0

node z table

from \ cost to	x	y	z
x	∞	∞	∞
y	∞	∞	∞
z	7	1	0

from \ cost to	x	y	z
x	0	2	7
y	2	0	1
z	3	1	0

from \ cost to	x	y	z
x	0	2	3
y	2	0	1
z	3	1	0



time

43

Distance vector: state information diffusion

Iterative communication, computation steps diffuses information through network:

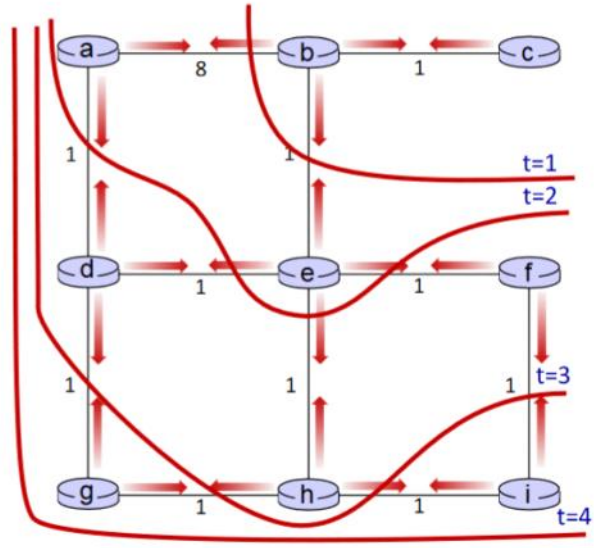
t=0 c's state at t=0 is at c only

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

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

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

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



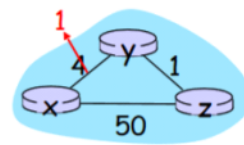
-We may have a count-to-infinity convergence problem

-Black hole / sink hole - intentionally advertising a link cost of 0 will cause all traffic to be sent to you

Distance vector: link cost changes

link cost changes:

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



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

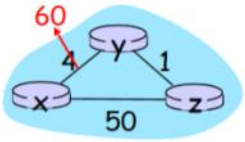
"good news travels fast"

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

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

Distance vector: link cost changes

link cost changes:



- node detects local link cost change
- "bad news travels slow" – count-to-infinity problem:
  - *y* sees direct link to *x* has new cost 60, but z has said it has a path at cost of 5. So *y* computes "my new cost to x will be 6, via z); notifies *z* of new cost of 6 to *x*.
  - *z* learns that path to *x* via *y* has new cost 6, so *z* computes "my new cost to *x* will be 7 via y), notifies *y* of new cost of 7 to *x*.
  - *y* learns that path to *x* via *z* has new cost 7, so *y* computes "my new cost to *x* will be 8 via y), notifies *z* of new cost of 8 to *x*.
  - *z* learns that path to *x* via *y* has new cost 8, so *z* computes "my new cost to *x* will be 9 via y), notifies *y* of new cost of 9 to *x*.
  ...
- see text for solutions. *Distributed algorithms are tricky!*

-Comparison of LS (link-state / Dijkstra's) and DV (distance vector / Bellman-Ford)

Comparison of LS and DV algorithms

message complexity

LS: *n* routers, $O(n^2)$ messages sent

DV: exchange between neighbors; convergence time varies (based on diameter of network)

speed of convergence

LS: $O(n^2)$ algorithm, $O(n^2)$ messages
- may have oscillations

DV: convergence time varies
- may have routing loops
- count-to-infinity problem

robustness: what happens if router malfunctions, or is compromised?

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

DV:
- DV router can advertise incorrect *path* cost ("I have a *really* low cost path to everywhere"): **black-holing**
- each router's table used by others: **error propagate thru network**

11/14 - OSPF, BGP

Thursday, November 14, 2024 5:03 PM

-Internet (intra-ISP) routing - OSPF

-Split the Internet into smaller pieces that you can run link-state (Dijkstra) or distance vector (Bellman-Ford) on, there's billions of destinations (can't have a routing table big enough to account for this)

Making routing scalable

our routing study thus far - idealized

- all routers identical
- network "flat"

... not true in practice

scale: billions of destinations:

- can't store all destinations in routing tables!
- routing table exchange would swamp links!

administrative autonomy:

- Internet: a network of networks
- each network admin may want to control routing in its own network

-Autonomous systems == AS == domains - aggregate routers into regions

-Intra-AS == intra-domains - this is routing within the same AS network

-OSPF (Open Shortest Path First, based on Dijkstra)

-Used to use RIP, defunct (Routing Information Protocol, based on Bellman-Ford)

-Inter-AS == inter-domains - this is routing among multiple AS's

-BGP (Border Gateway Protocol)

Internet approach to scalable routing

aggregate routers into regions known as "autonomous systems" (AS) (a.k.a. "domains")

intra-AS (aka "intra-domain"):
routing *within same AS ("network")*

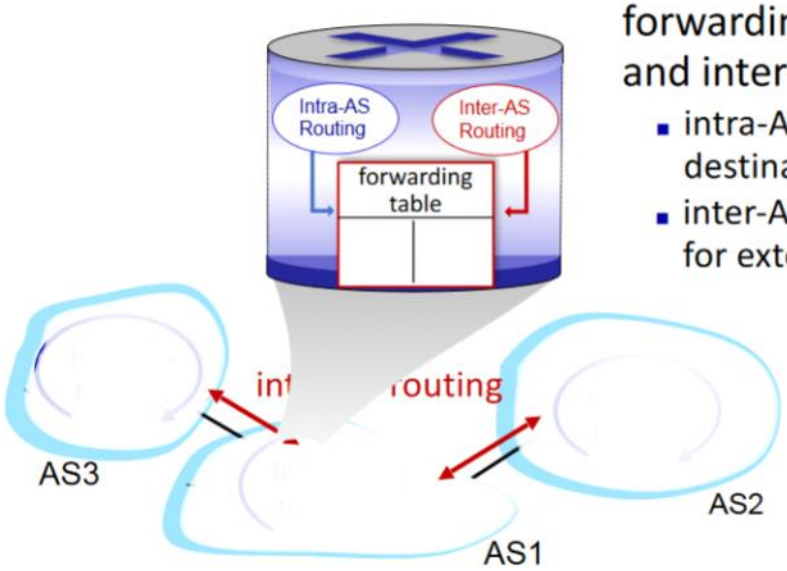
- all routers in AS must run same intra-domain protocol
- routers in different AS can run different intra-domain routing protocols
- gateway router: at "edge" of its own AS, has link(s) to router(s) in other AS'es

inter-AS (aka "inter-domain"):
routing *among* AS'es

- gateways perform inter-domain routing (as well as intra-domain routing)

-We're focusing on intra-AS which is routing within the same AS and uses OSPF (Open Shortest Path First, uses Dijkstra)

Interconnected ASes



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

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

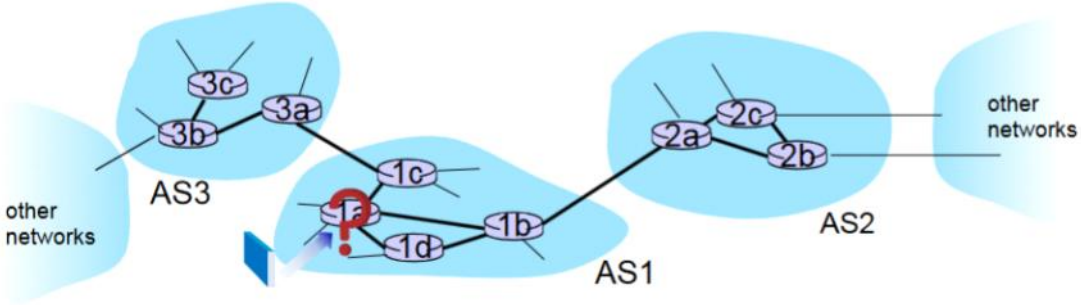
-Inter-AS example

Inter-AS routing: a role in intradomain forwarding

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

AS1 inter-domain routing must:

1. learn which destinations reachable through AS2, which through AS3
2. propagate this reachability info to all routers in AS1



-Intra-AS routing

Intra-AS routing: routing within an AS

most common intra-AS routing protocols:

- RIP: Routing Information Protocol [RFC 1723]
  - classic DV: DVs exchanged every **30 secs**
  - no longer widely used (was popular in 1980s and 1990s)
- EIGRP: Enhanced Interior Gateway Routing Protocol
  - DV based
  - formerly **Cisco-proprietary** for decades (became open in 2013 [RFC 7868])
- OSPF: Open Shortest Path First [RFC 2328]
  - Classic link-state routing
  - IS-IS protocol (ISO standard, not RFC standard) essentially same as OSPF

-We don't want a "flat network" (flat Internet), that's bad security, we don't have subnetworks which will have too many router entries in its forwarding table, we need smaller autonomous systems (AS's)

-OSPF - Open Shortest Path First

-Also called IS-IS routing algorithm as referred to be the ISO standards organization

-OSPF only lets you use metrics such as delay or bandwidth, not traffic volume because it uses Dijkstra's shortest path algorithm and this would lead to the same oscillation issues that the link-state routing algorithm has

-OSPF message authentication prevents malicious intrusion

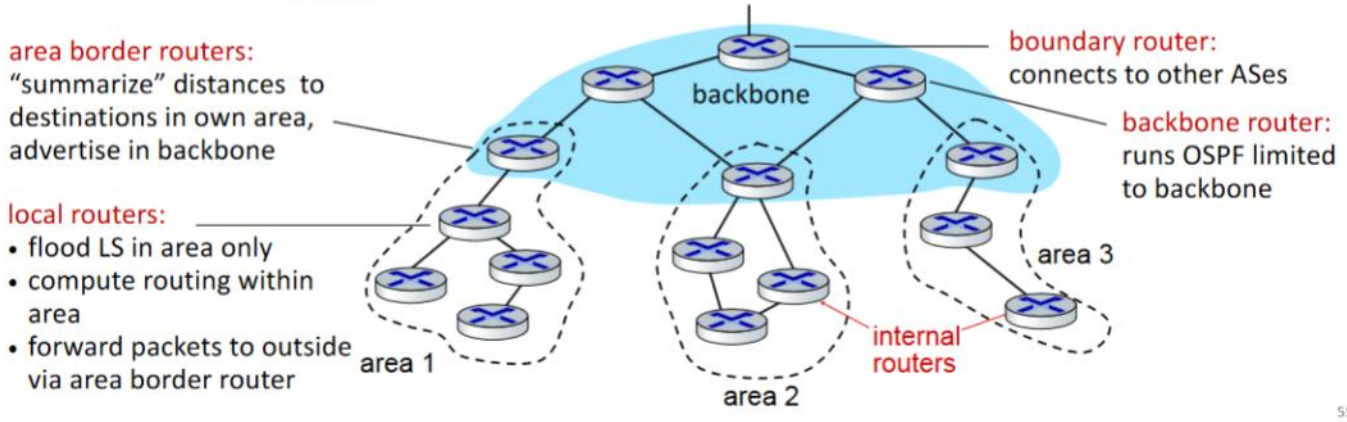
OSPF (Open Shortest Path First) routing

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

-Dijkstra floods specific areas supported by a backbone for OSPF on intra-domain routing

Hierarchical OSPF

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



55

-BGP (Border Gateway Protocol) - how we route among ISPs, very important for the Internet to exist, it's the "glue that holds the Internet together"

-eBGP (External BGP) - eBGP session, obtain subnet reachability between neighboring AS's

-iBGP (Internal BGP) - iBGP session, propagate reachability information to all AS's

Internet inter-AS routing: BGP

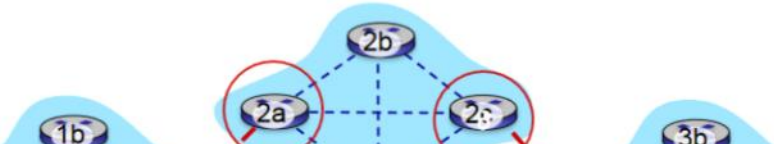
- BGP (Border Gateway Protocol): *the* de facto inter-domain routing protocol
  - "glue that holds the Internet together"
- allows subnet to advertise its existence, and the destinations it can reach, to rest of Internet: *"I am here, here is who I can reach, and how"*
- BGP provides each AS a means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

-eBGP and iBGP connections are actually TCP sessions

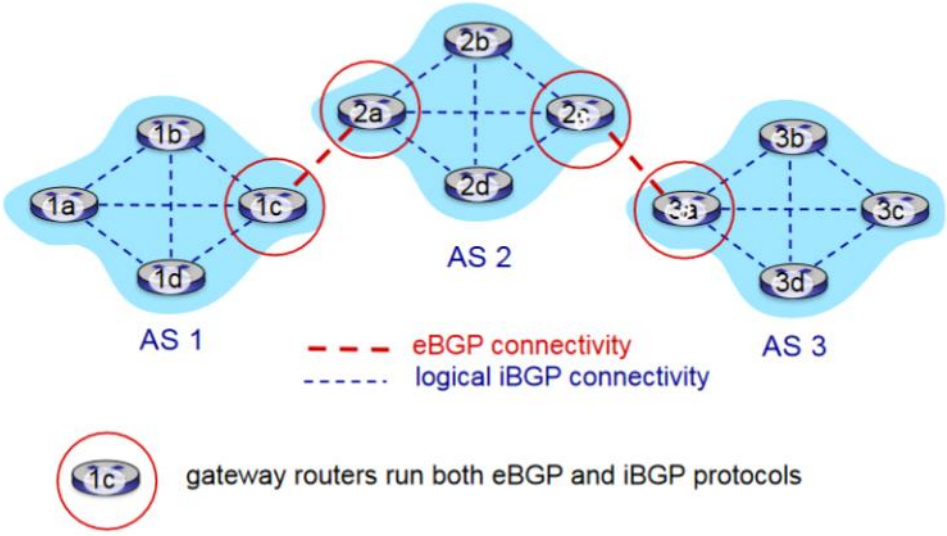
-iBGP session is a logical connection between routers

-eBGP session is used on edge routers (edge of a network) to communicate out to other networks, the routers doing this are gateway routers

eBGP, iBGP connections



eBGP, iBGP connections

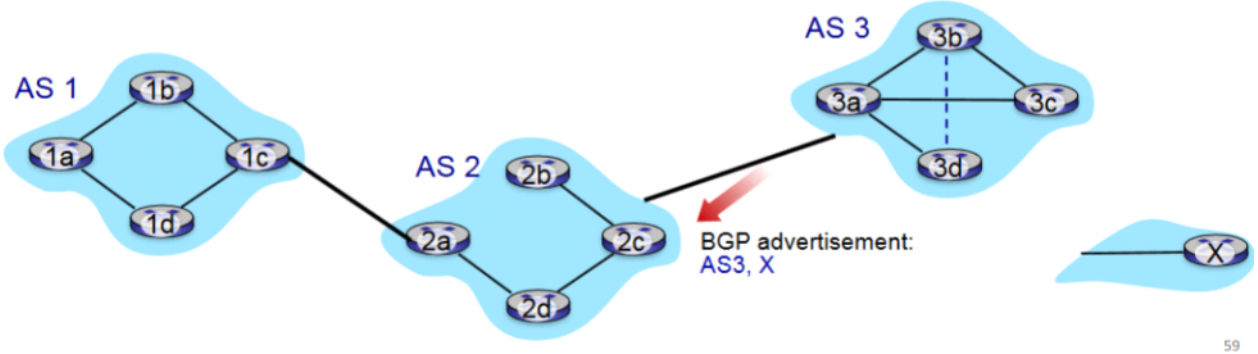


-Port 179 is the TCP port the router runs on its OS for a BGP session to message other routers, another router also has a BGP session connection open to receive a message

-BGP advertises as a promise that it will forward an IP datagram from one gateway to another

BGP basics

- BGP session: two BGP routers ("peers") exchange BGP messages over semi-permanent TCP connection (port 179):
  - advertising *paths* to different destination network prefixes , e.g., to a destination /16 network (BGP is a "path vector" protocol)
- when AS3 gateway 3a advertises path AS3,X to AS2 gateway 2c:
  - AS3 *promises* to AS2 it will forward datagrams towards X



59

-BGP messages over a TCP connection use specific messages such as OPEN, UPDATE, etc

BGP messages

- BGP messages exchanged between peers over TCP connection
- BGP messages (RFC 4271):
  - OPEN: ***opens TCP connection*** to remote BGP peer and authenticates sending BGP peer
  - UPDATE: ***advertises*** new **path** (or withdraws old)
  - KEEPALIVE: keeps connection alive in absence of UPDATES; also ACKs OPEN request
  - NOTIFICATION: reports errors in previous msg; also used to ***close connection***

-AS-PATH is the most important attribute of BGP

-Import policy is used by BGP to determine to accept or decline a path on a gateway

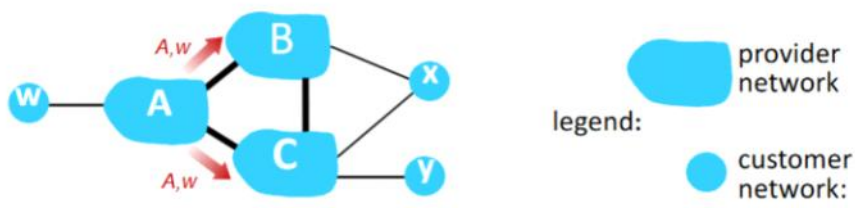
Path attributes and BGP routes

- BGP **advertised route/path**: **prefix** + **attributes (E.g., x, AS3 AS2 AS1 ...)**
  - **prefix**: destination being advertised (e.g., network x)
  - **two important attributes**:
    - 1) **AS-PATH**: **list of ASes** through which prefix advertisement has passed (e.g., AS3 AS2..)
    - 2) **NEXT-HOP**: indicates specific **internal-AS router to next-hop AS** (e.g., Gateway 1)
- policy-based routing:
  - gateway receiving route advertisement uses *import policy* to **accept/decline** path (e.g., never route through AS/ISP W, or country Y).
  - A router/gateway uses policy to **also** determine **whether to *advertise* path to neighboring AS** Z (does router really want to route traffic forwarded from Z destined to X?)

-ISPs only want to route traffic between its customers, they don't want to transmit traffic between customers from other ISPs

-In the real world, all ISPs are interconnected so they'll eventually find a path between each other (regional, international, local ISPs)

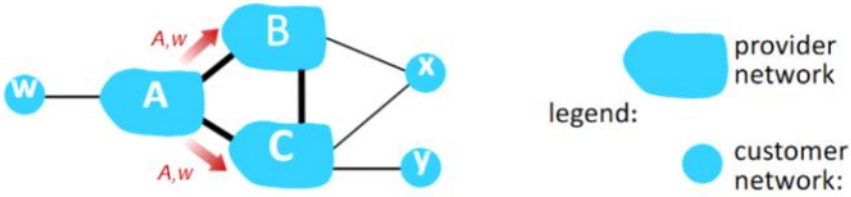
BGP: achieving policy via advertisements



ISP only wants to route traffic to/from its customer networks (does not want to carry ***transit traffic*** between other ISPs – a typical "real world" policy)

A advertises path Aw to B and to C (**ISP w is a customer of network provider A**)

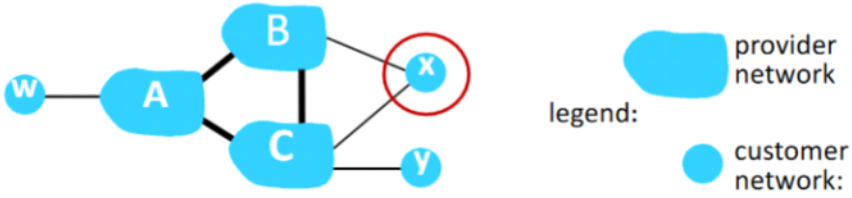
BGP: achieving policy via advertisements



ISP only wants to route traffic to/from its customer networks (does not want to carry ***transit traffic*** between other ISPs – a typical "real world" policy)

- A advertises path Aw to B and to C (**ISP w is a customer of network provider A**)
- B *chooses not to advertise* BAw to C!
  - B gets no "revenue" for routing CBAw, since none of C, A, w are B's customers
  - C does *not* learn about CBAw path
- C will route CAw (not using B) to get to w

BGP: achieving policy via advertisements (more)



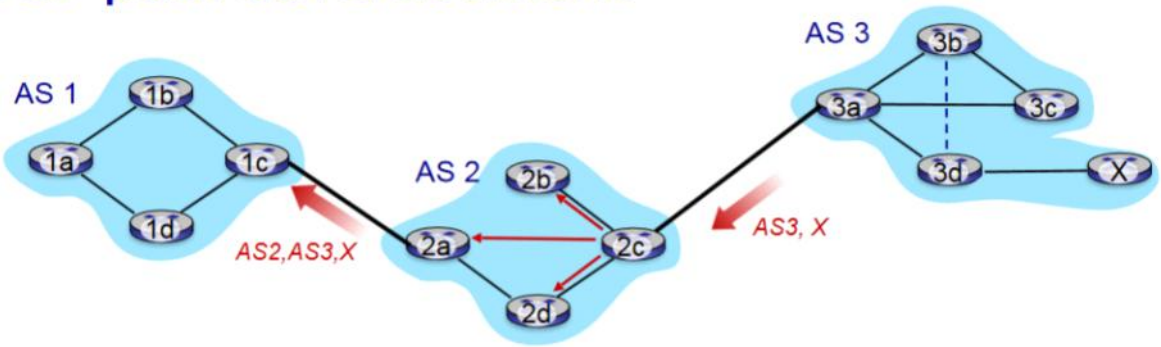
ISP only wants to route traffic to/from its customer networks (does not want to carry transit traffic between other ISPs – a typical "real world" policy)

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

-BGP routing example

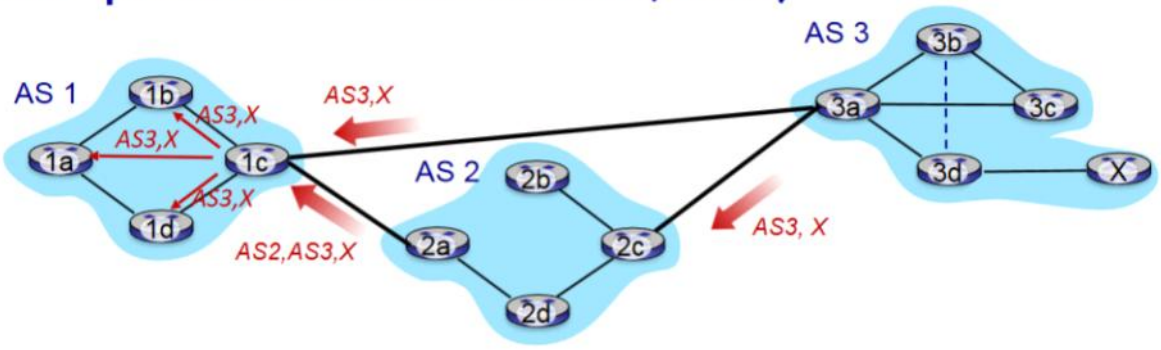
-BGP doesn't always choose the best path based on policy

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

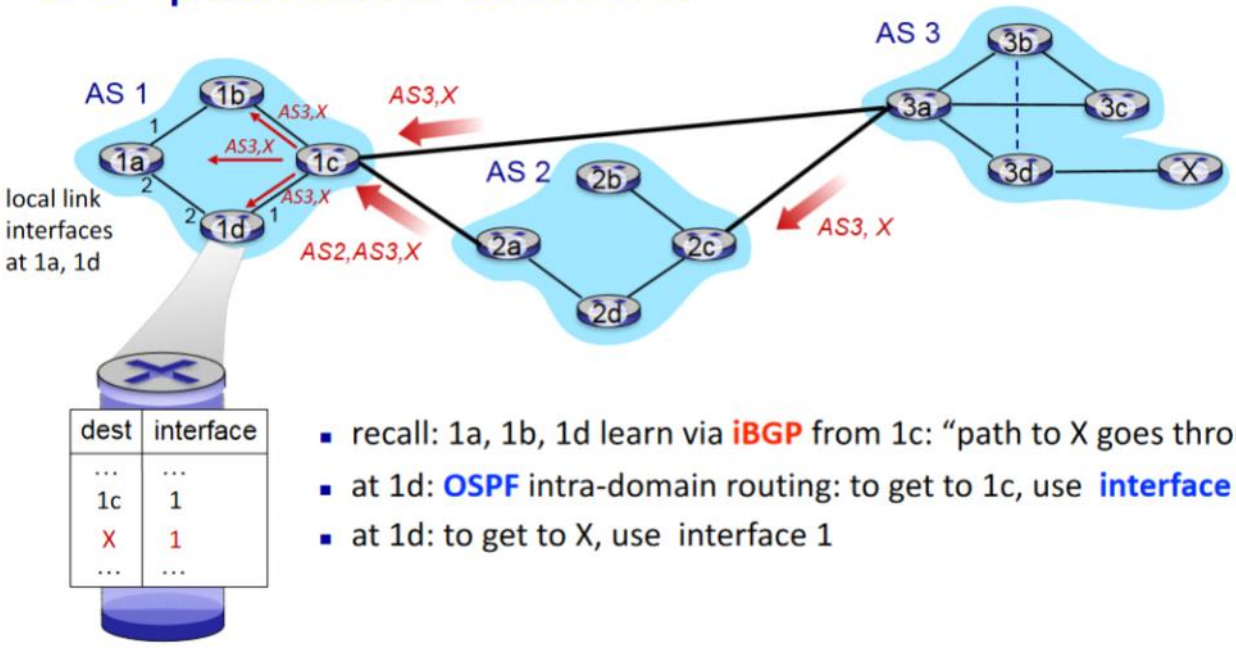
BGP path advertisement (more)



gateway router may learn about multiple paths to destination:

- AS1 gateway router 1c learns path *AS2,AS3,X* from 2a
- AS1 gateway router 1c learns path *AS3,X* from 3a
- based on *policy,* AS1 gateway router 1c **chooses** path *AS3,X* and advertises path within AS1 via iBGP

BGP path advertisement

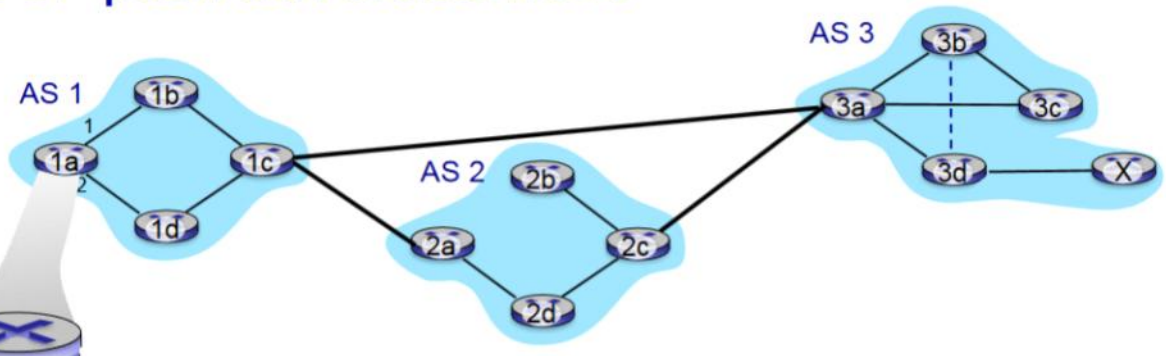


dest	interface
…	…
1c	1
X	1
…	…

- 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

66

BGP path advertisement



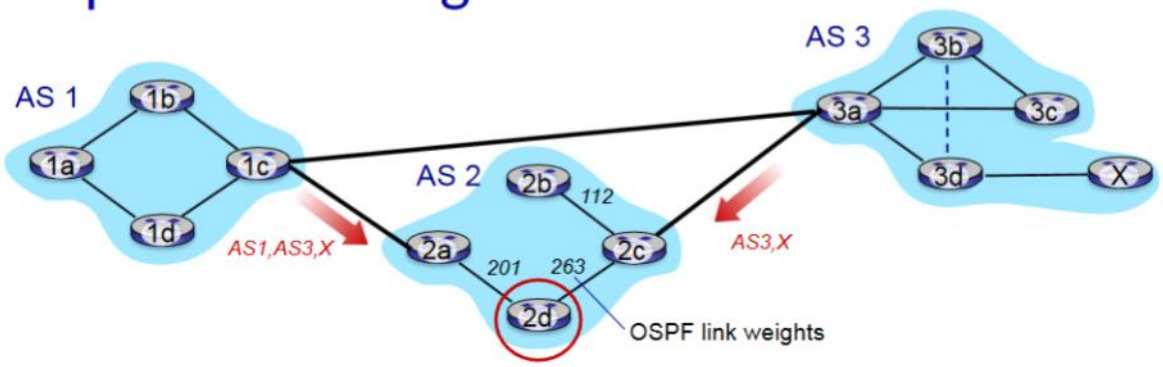
dest	interface
…	…
1c	2
X	2
…	…

- 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
- at 1a: **OSPF** intra-domain routing: ***suppose*** to get to 1c, use **interface 2**
- at 1a: to get to X, use interface 2

67

-Hot potato routing - if you have two advertisements from 2 different routers, you choose the local gateway that has the least intra-domain cost without concern for inter-domain costs; that is, you only care about the costs of edges/links within a network/domain, not costs outside of that domain

Hot potato routing



- 2d learns (via iBGP) it can route to X via 2a or 2c
- hot potato routing: choose local gateway that has **least *intra-domain* cost** (e.g., 2d chooses 2a, even though more AS hops to *X*): don't worry about inter-domain cost!

68

-Policy - if you're not making money passing traffic through your autonomous system (AS) as an ISP, then why would you pass that traffic through?

-Scale prevents flat networks which reduces the side of the forwarding table

-OSPF (Open Shortest Path First, based on Dijkstra), most popular and common router, used for Intra-AS

-BGP (Border Gateway Protocol, based on Distance Vector/Bellman-Ford), based on policy, Inter-AS

Why different Intra-, Inter-AS routing ?

policy:

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

scale:

- hierarchical routing saves table size, reduced update traffic

performance:

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

BGP route selection

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

11/19 - Error Detection

Tuesday, November 19, 2024 5:08 PM

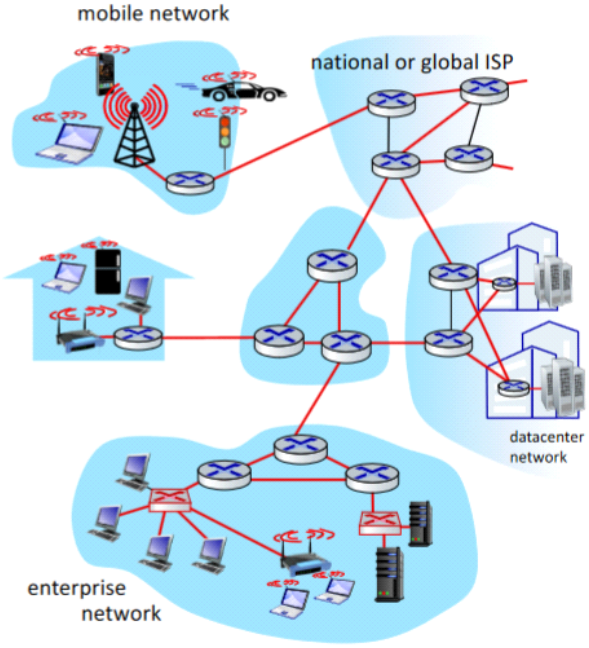
-Refresher of OSI model
- -Application Layer - used by developers to build application (all other layers used by the OS); ex. HTTP, DNS
- -Transport Layer - accessible over APIs on sockets that act as a door to the OS with port numbers, Transport Layer is responsible for process-to-process communication; examples: TCP, UDP where TCP is better with reliability and UDP is better with not needing to establish a connection (less overhead)
- -Network Layer - routing, forwarding, host-to-host communication, generating IP address
- -Data/Link Layer
  - -Error detection and correction
  - -Multiple access
  - -Link-layer addressing (MAC address)
  - -Ethernet and LANs
  - -Implementations (coaxial, fiber optics, wireless, Ethernet, WiFi 802.11)
- -Physical Layer

-Hosts and routers are called nodes; communication channels between nodes are called links

Link layer: introduction

terminology:

- hosts and routers: **nodes**
- communication channels that connect adjacent nodes along communication path: **links**
  - wired
  - wireless
  - LANs

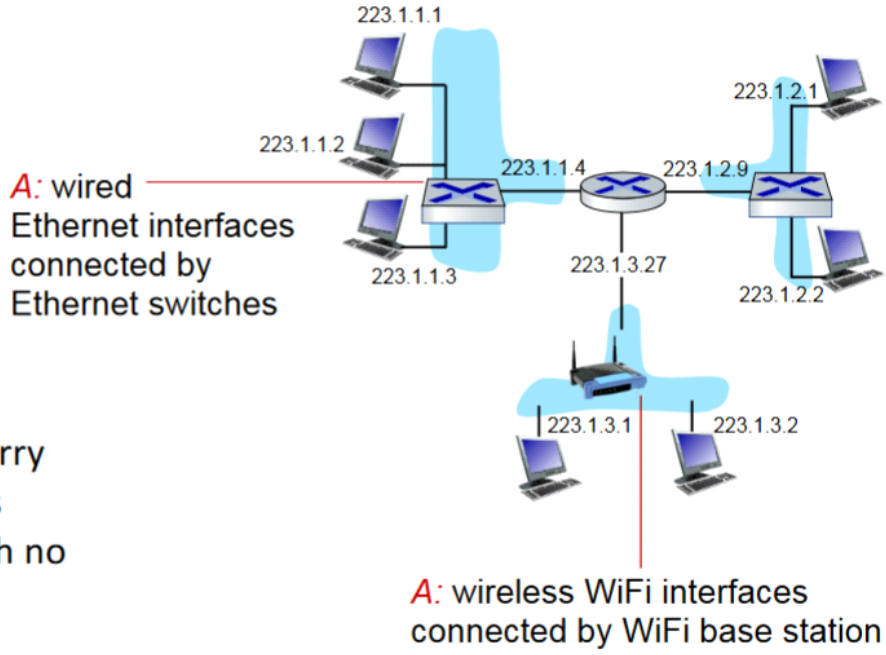


-Within a subnet, we connect devices together with a switch

Recall: IP addressing: introduction

Q: how are interfaces actually connected?

A: we'll learn about that in chapters 6, 7



For now: don't need to worry about how one interface is connected to another (with no intervening router)

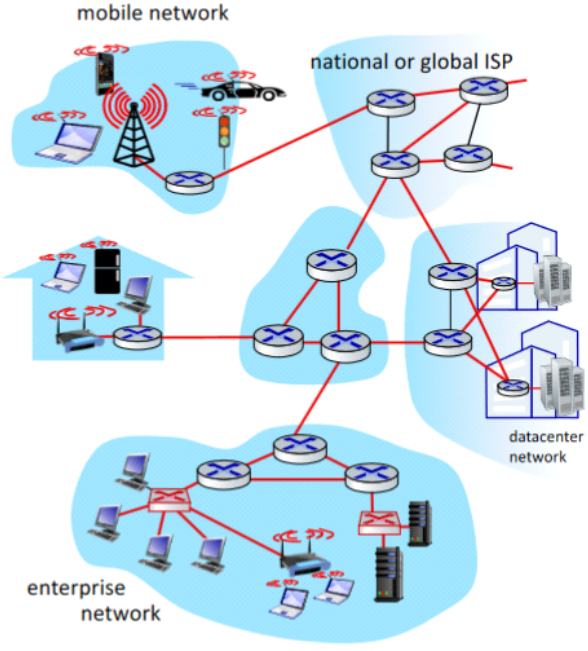
-On the Data/Link Layer, our data is called a Data/Link Frame

Link layer: introduction

terminology:

- hosts and routers: **nodes**
- communication channels that connect adjacent nodes along communication path: **links**
  - wired
  - wireless
  - LANs
- layer-2 packet: ***frame***, encapsulates datagram

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



6

-Network Layer IP Datagrams are transferred over different Data/Link Layer protocols over several different types of links (e.g. WiFi on first link, Ethernet on second link, etc)

Link layer: context

- datagram transferred by different link protocols over different links:
  - e.g., **WiFi** on **first link**, **Ethernet** on **next link**
- each link protocol provides different services
  - e.g., may or may not provide reliable data transfer over link

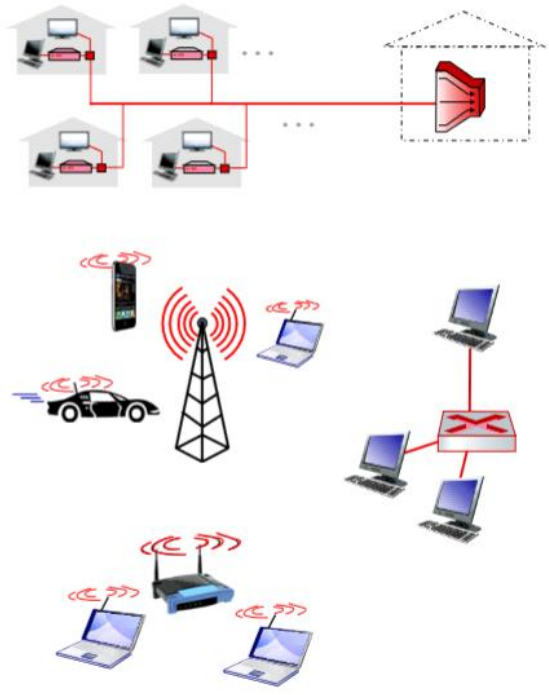
transportation analogy:

- trip from Princeton, NJ to Lausanne (Switzerland)
  - limo: Princeton to JFK
  - plane: JFK to Geneva (Switzerland)
  - train: Geneva to Lausanne
- tourist = datagram
- transport segment = communication link
- transportation mode (limo, plane, train, etc.) = link-layer protocol
- travel agent = routing algorithm

-Data/Link Layer frames have a header and trailer, MAC address needs to be in the frame header to identify source and destination node, TCP handles most reliability but Data/Link Layer does also handle bit-errors (we have both Transport Layer and Data/Link Layer reliability because we're not always using TCP at the Transport Layer so we need extra security)

Link layer: services

- framing, link access:
  - encapsulate datagram into frame, **adding header, trailer**
  - channel access **if shared medium**
  - **"MAC" addresses** in frame headers identify **source**, **destination** (different from IP address!)
- reliable delivery between adjacent nodes
  - we already know how to do this!
  - **seldom used on low bit-error links**
  - wireless links: high error rates
    - *Q:* why both link-level and end-end reliability?

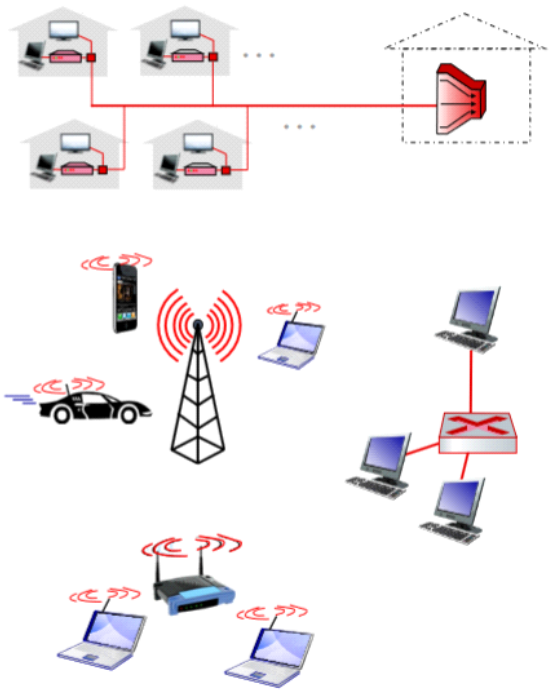


8

-Full-duplex is what a transceiver is (we can send and receive data at the same time), error detection in the Data/Link Layer lets us actually fix incorrect data on the spot (as opposed to just resending it on the Transport Layer)

Link layer: services (more)

- flow control:
  - pacing between adjacent sending and receiving nodes
- error detection:
  - errors caused by signal attenuation, noise.
  - receiver detects errors, signals retransmission, or drops frame
- **error correction:**
  - receiver identifies ***and corrects*** bit error(s) without retransmission
- half-duplex and full-duplex:
  - with half duplex, nodes at both ends of link can transmit, but not at same time



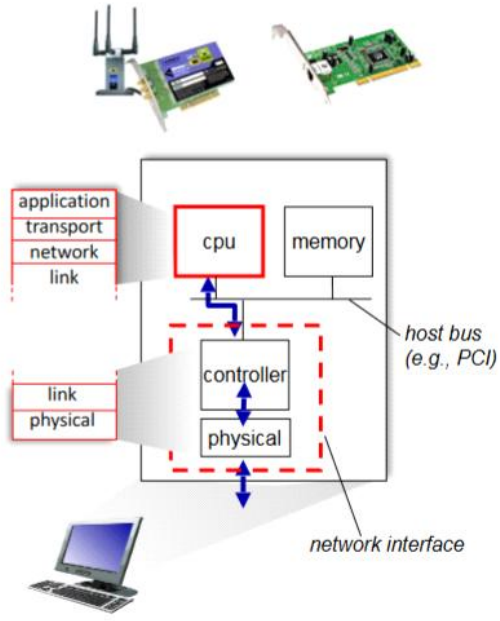
9

-Data/Link Layer is implemented on a Network Interface Card (NIC) that has its own MAC address

-Data/Link Layer is a combination of software and hardware and firmware (software that persists on BIOS on ROM)

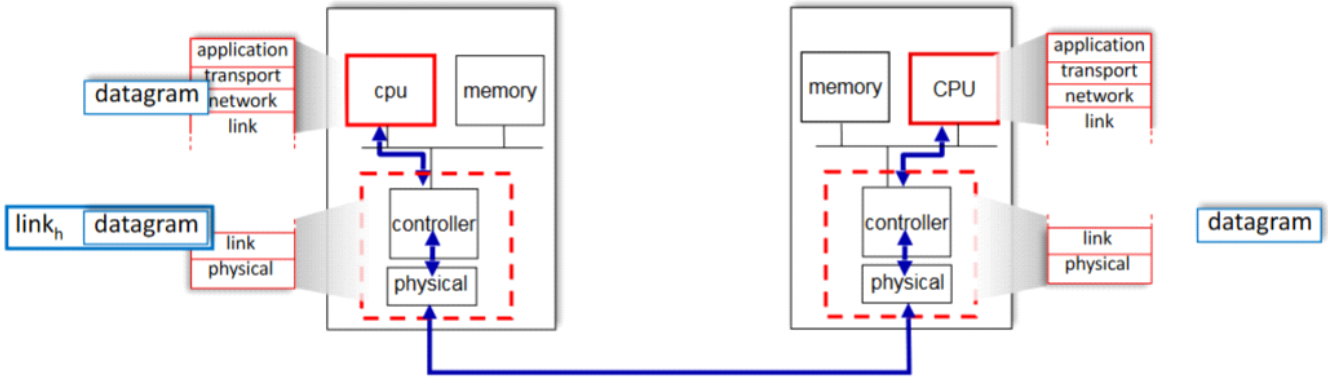
Where is the link layer implemented?

- in each-and-every host
- link layer implemented in ***network interface card*** (**NIC**) or on a chip
  - Ethernet, WiFi card or chip
  - implements link, physical layer
- attaches into host's system buses
- **combination of hardware, software, firmware**



-How encapsulation works

Interfaces communicating



sending side:

- encapsulates datagram in frame
- adds error checking bits, reliable data transfer, flow control, etc.

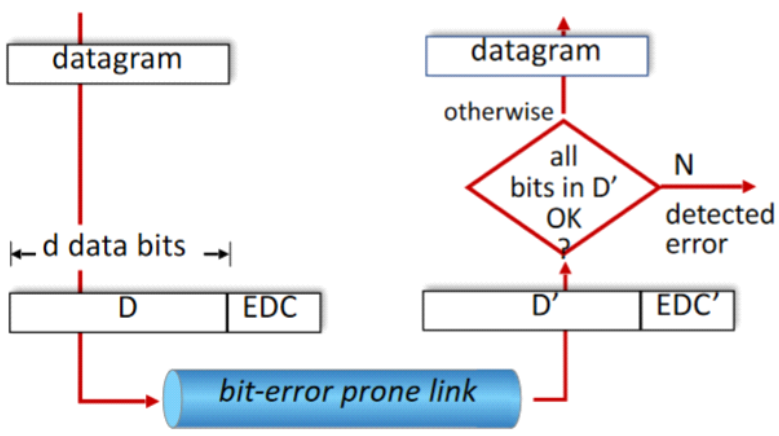
receiving side:

- looks for errors, reliable data transfer, flow control, etc.
- extracts datagram, passes to upper layer at receiving side

-EDC fields - Error Detection and Correction bits (AKA redundancy bits), its 8-bits
-D - data field that is protected by error checking (may include header fields)
-EDC' and D' are the bits checked by a "checksum" in the tail
-Actual checksum in the Data/Link Layer is a terrible idea, error detection is not 100% in the Data/Link Layer

Error detection

EDC: error detection and correction bits (e.g., redundancy)
D: data protected by error checking, may include header fields



Error detection **not 100% reliable**!

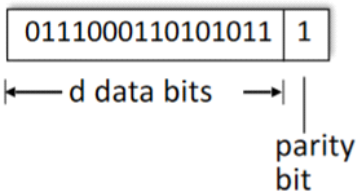
- protocol may miss some errors, but rarely
- larger EDC field yields better detection and correction

-Single bit parity is like checksum (it detects but does not correct), two-dimensional bit parity detects and corrects where you present the bits as rows and columns, example of how two-dimensional parity below

Parity checking

single bit parity:

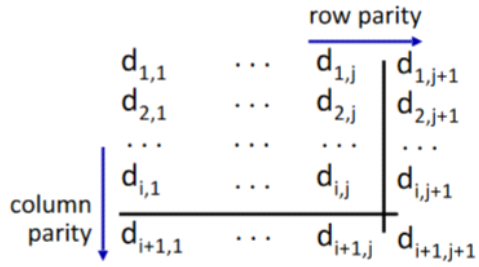
- detect single bit errors



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

two-dimensional bit parity:

- detect *and correct* single bit errors



no errors:

1	0	1	0	1	1
1	1	1	1	0	0
0	1	1	1	0	1
0	0	1	0	1	0

detected and correctable single-bit error:

1	0	1	0	1	1
1	0	1	1	0	0
0	1	1	1	0	1
0	0	1	0	1	0

parity error

parity error

* Check out the online interactive exercises for more examples: http://gaia.cs.umass.edu/kurose_ross/interactive/

-THIS WILL BE ON THE EXAM

-Cyclic Redundancy Check (CRC) - more powerful error-detection encoding (even better than a checksum), you add the CRC at the end of the frame (on the tail)

-CRC is the generator - 1 bits

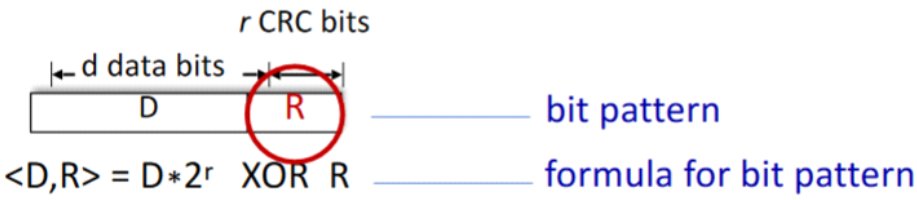
-<D,R> = D*2^r (XOR) R, shift D to the left by r number of bits then XOR'd by R yields <D,R> which should be exactly divisible by G (mod 2) where G is our generator bit pattern; we can detect all errors with this

-Zero remainder means there's DEFINITELY no error, and a non-zero remainder means there's DEFINITELY an error

-Generator unlikely to be corrupted since it's only 4 bits

Cyclic Redundancy Check (CRC)

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



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

- ***receiver knows G, divides <D,R> by G. If non-zero remainder: error detected!***
- can **detect** all burst errors less than r+1 bits
- widely used in practice (Ethernet, 802.11 WiFi)

-Example of CRC

Cyclic Redundancy Check (CRC): example

We want:

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

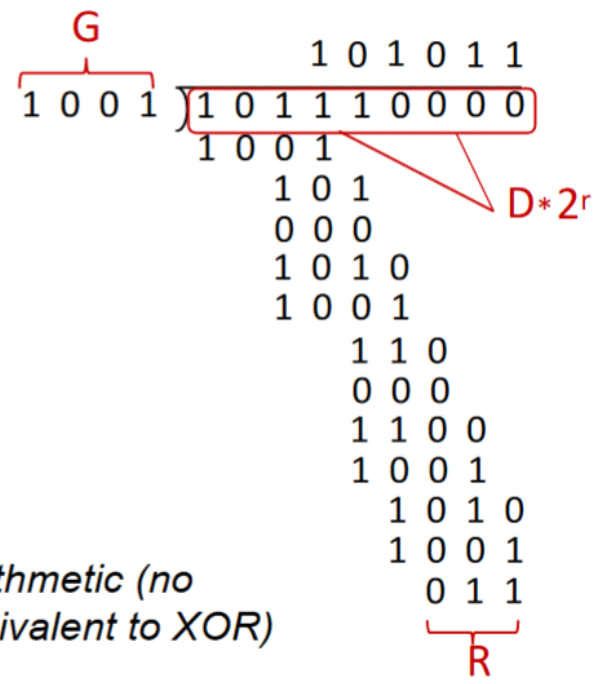
or equivalently:

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

or equivalently:

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

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



Note: All calculations done in mod-2 arithmetic (no carries in addition/subtraction, both equivalent to XOR)

* Check out the online interactive exercises for more examples: http://gaia.cs.umass.edu/kurose_ross/interactive/

11/21 - MAC Protocols

Thursday, November 21, 2024 4:57 PM

-Can't communicate at the same time over a single channel/medium or we have interference
-Distributed algorithms to implement how nodes share a channel are pretty tricky

Multiple access links, protocols

two types of "links":

- point-to-point
  - point-to-point link between Ethernet switch, host
  - PPP for dial-up access
- broadcast (shared wire or medium)
  - old-fashioned Ethernet (e.g., using a **Hub**)
  - upstream HFC (Hybrid Fiber Coaxial) in **cable**-based access network
  - 802.11 **wireless** LAN, **4G/5G**, **satellite**

Multiple access protocols

- single **shared broadcast channel**
- two or more simultaneous transmissions by nodes: **interference**
  - ***collision*** if node receives two or more signals at the same time

multiple access protocol

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

-An ideal multiple access protocol

An **ideal** multiple access protocol

given: multiple access channel (MAC) of rate *R* bps

desiderata:

1. when one node wants to transmit, it can send at rate *R*.
2. when M nodes want to transmit, each can send at **average rate *R/M***
3. fully **decentralized**:
   - no special node to coordinate transmissions
   - no synchronization of clocks, slots
4. **simple**

-MAC addresses are split into smaller pieces using time slots, frequency, and code

MAC protocols: taxonomy

three broad classes:

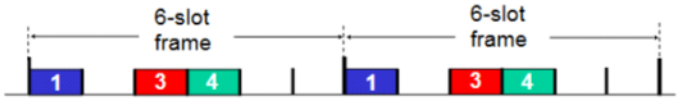
- channel partitioning
  - divide channel into smaller "pieces" (time slots, frequency, code)
  - allocate piece to node for exclusive use
- *random access*
  - channel not divided, allow collisions
  - "recover" from collisions
- "taking turns"
  - nodes take turns, but nodes with more to send can take longer turns

-How MAC addresses split these channels to allow multiple access without interference

Channel partitioning MAC protocols: TDMA

TDMA: time division multiple access

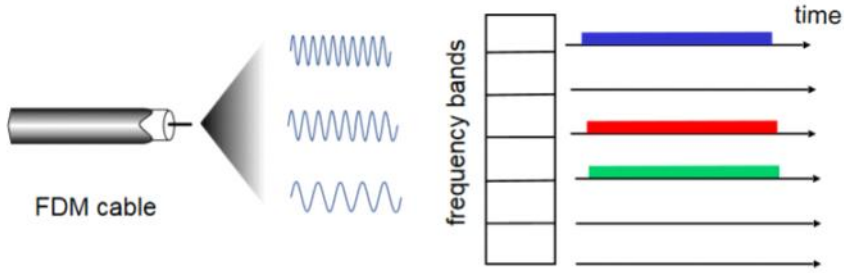
- access to channel in "rounds"
- each station gets fixed length slot (length = packet transmission time) in each round
- **unused slots** go **idle**
- example: 6-station LAN, 1,3,4 have packets to send, slots 2,5,6 idle



Channel partitioning MAC protocols: FDMA

FDMA: frequency division multiple access

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



-Random access allows us to detect collisions in MAC protocols
    -ALHOA is old, it was used to communicate with the Hawaiian islands
    -CSMA/CD is what NICs in your computer use
    -CSMA/CA is specifically for collision avoidance, it's wireless

Random access protocols

- when node has packet to send
  - transmit at full channel data rate R.
  - no *a priori* coordination among nodes
- two or more transmitting nodes: "**collision**"
- random access MAC protocol specifies:
  - When to send
  - how to detect collisions
  - how to recover from collisions (e.g., via delayed retransmissions)
- examples of random access MAC protocols:
  - ALOHA, slotted ALOHA
  - CSMA, CSMA/CD, CSMA/CA

-Random access MAC protocols used for collision detection
    -Slotted ALHOA

Slotted ALOHA

assumptions:

- all frames same size
- time divided into **equal size slots** (time to transmit 1 frame)
- nodes start to **transmit only at beginning of slot**
- nodes are **synchronized**
- if 2 or more nodes transmit in slot, all nodes detect **collision**

operation:

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

randomization – *why*?

Slotted ALOHA

C: collision
S: success
E: empty

Pros:

- single active node can continuously transmit at full rate of channel
- highly **decentralized**:
  - No need for central/special node
  - only slots in nodes need to be in sync
- **simple**

Cons:

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

Slotted ALOHA: efficiency

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

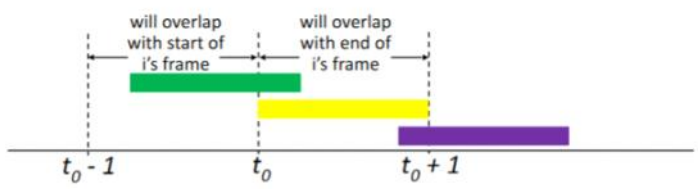
- *suppose:* ***N* nodes** with many frames to send, **each transmits in slot with probability *p***
  - prob that **given node** has **success** in a slot $= p(1-p)^{N-1}$
  - prob that ***any* node** has a **success** $= Np(1-p)^{N-1}$
  - **max** efficiency: find p^* that maximizes $Np(1-p)^{N-1}$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

  max efficiency = 1/e = .37
- ***at best:*** channel used for useful transmissions **only 37%** of time!

-Pure ALHOA

Pure ALOHA

- unslotted Aloha: 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 $[t_0-1, t_0+1]$



- pure Aloha **efficiency**: **18%** !

-CSMA

CSMA (carrier sense multiple access)

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

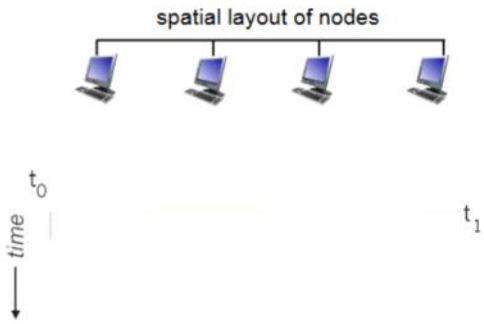
- human analogy: don't interrupt others!

CSMA/CD: CSMA with *collision detection*
- collisions *detected* within short time
- colliding transmissions **aborted**, reducing channel wastage
- collision detection **easy in wired**, **difficult with wireless**

- human analogy: the polite conversationalist

CSMA: collisions

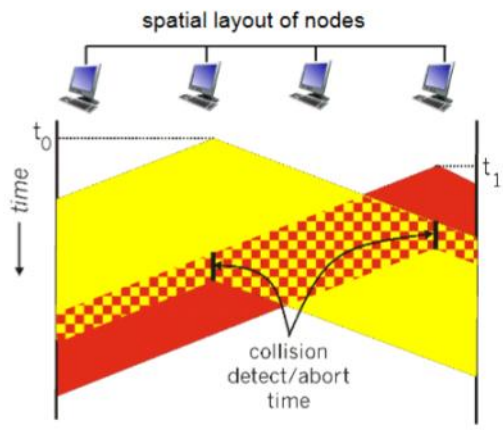
- collisions ***can* still occur** with carrier sensing:
  - propagation delay means two nodes may not hear each other's just-started transmission
- collision: entire packet transmission time wasted
  - distance & propagation delay play role in determining collision probability



-CSMA/CD

CSMA/CD:

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



-How the NIC works

Ethernet CSMA/CD algorithm

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

-How we calculate efficiency of CSMA/CD

CSMA/CD efficiency

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

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

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

-Channel partitioning MAC protocols vs. random access MAC protocols

"Taking turns" MAC protocols

channel partitioning MAC protocols:

- share channel ***efficiently* and *fairly* at high load**
- **inefficient at low load**: delay in channel access, 1/N bandwidth allocated even if only 1 active node!

random access MAC protocols

- **efficient at low load**: single node can fully utilize channel
- **high load: collision overhead**

"taking turns" protocols

- look for best of both worlds!
  - Channel allocated explicitly (no collisions)
  - Nodes won't hold channels for long if nothing to send
  - Two approaches (Polling and token passing)

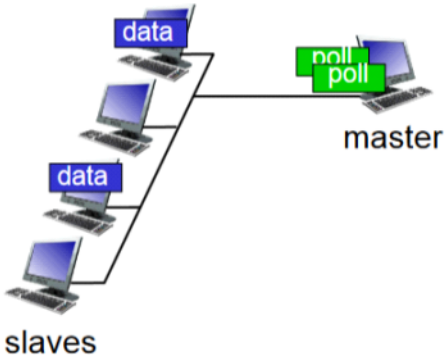
-"Taking turns" MAC protocols prevents collisions and has good efficiency

-Port polling "taking turns" MAC protocol - multiple devices transmit at the same time

"Taking turns" MAC protocols

1) polling:

- **master** node "invites" other nodes to transmit in turn
- typically used with "dumb" devices
- concerns:
  - join/leave protocol needed
  - polling overhead
  - latency (wait for turn)
  - single point of failure (master)

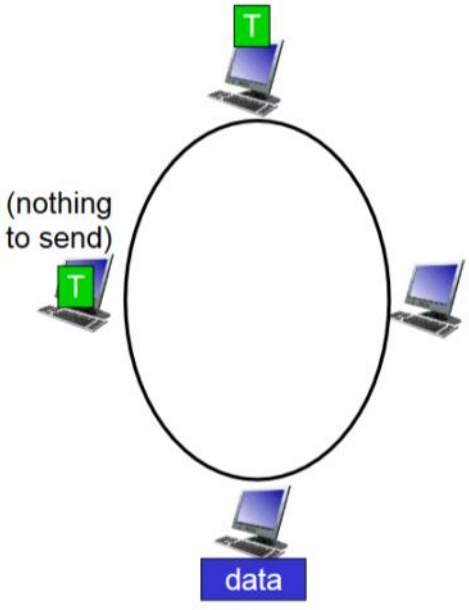


-Token passing "taking turns" MAC protocol - pretty much mutexes (Dining Philosopher's Problem)

"Taking turns" MAC protocols

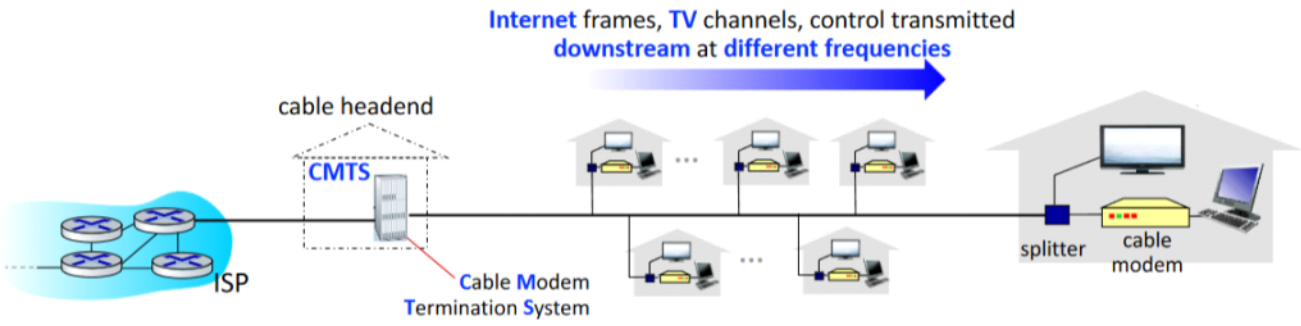
2) token passing:

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



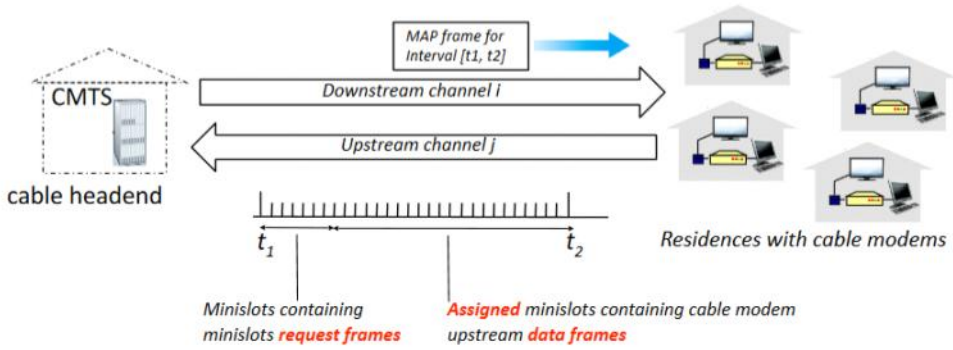
-More cable stuff

Cable access network: FDM, TDM *and* random access!



- multiple downstream (broadcast) **FDM** channels: up to 1.6 Gbps/channel
  - single CMTS transmits into channels
- multiple upstream **FDM** channels (up to 1 Gbps/channel)
  - multiple access: all users contend (**random access**) for **certain upstream channel time slots**; others assigned **TDM**

Cable access network:



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** (and data) transmitted **random access (binary backoff) in selected slots**

-Summary of MAC Protocols on a high level

Summary of MAC protocols

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

-Cable access network uses all of the above

11/26 - MAC Address, ARP

Tuesday, November 26, 2024 5:07 PM

-The MAC address is a physical address burned in the NIC ROM, it's 48 bits represented in hex (like an social security number for the hardware)

MAC addresses

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

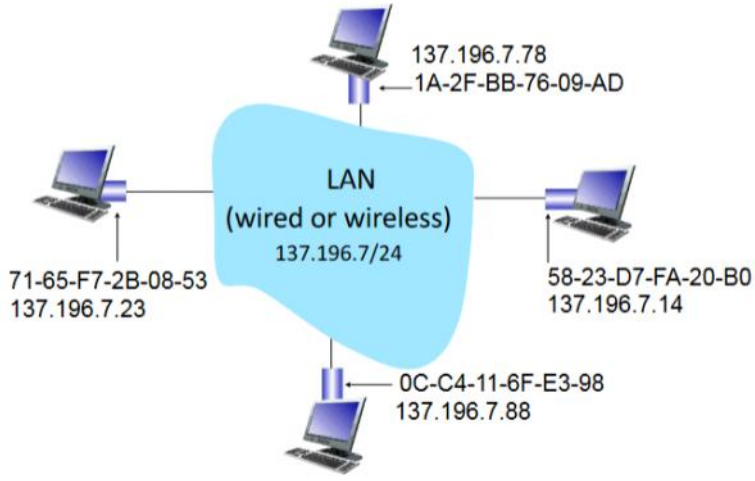
hexadecimal (base 16) notation (each "numeral" represents 4 bits)

-Each interface on a LAN gets a MAC address and a local IP address

MAC addresses

each interface on LAN

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



-MAC addresses are administered by the IEEE (they get paid for every NIC card made)

-The MAC address is a flat address, that is, it's portable, it does not change when you go from one LAN to another

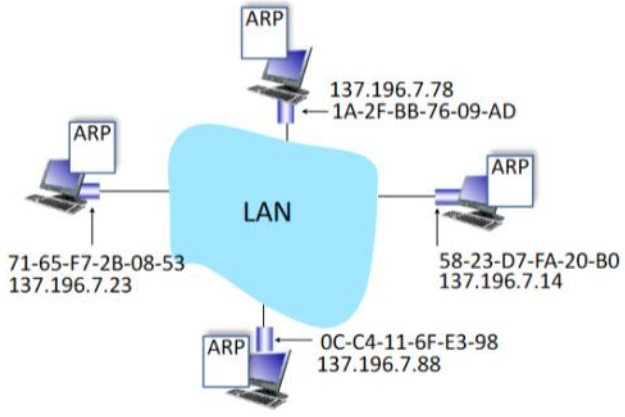
MAC addresses

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

-Address Resolution Protocol (ARP) - ARP broadcasts a message requesting a MAC address
-ARP lives in the Data/Link Layer
-Every computer/router has an ARP table (cache) that saves the MAC addresses

ARP: address resolution protocol

Question: how to determine interface's MAC address, knowing its IP address?



ARP table: each IP node (host, router) on LAN has table

- IP/MAC address mappings for some LAN nodes:

  < IP address; MAC address; TTL>
- **TTL** (Time To Live): time after which address mapping will be forgotten (typically **20 min**)

-How ARP works - IMPORTANT: need to know for homework and exam

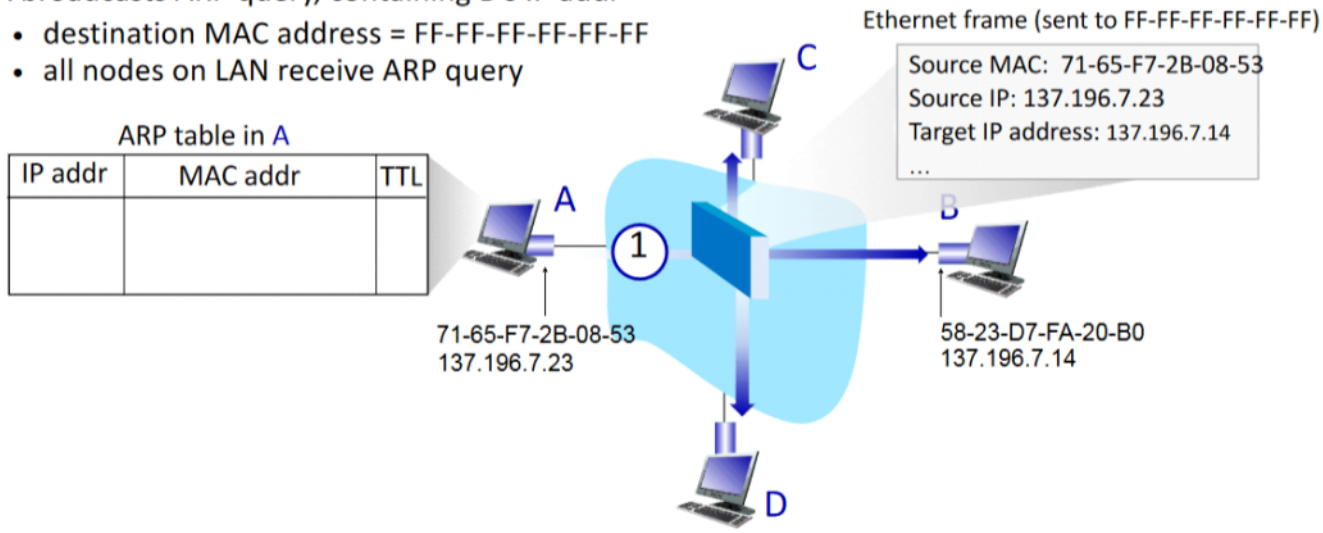
ARP protocol in action

example: A wants to send datagram to B

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

A broadcasts ARP query, containing B's IP addr

1
- destination MAC address = FF-FF-FF-FF-FF-FF
- all nodes on LAN receive ARP query

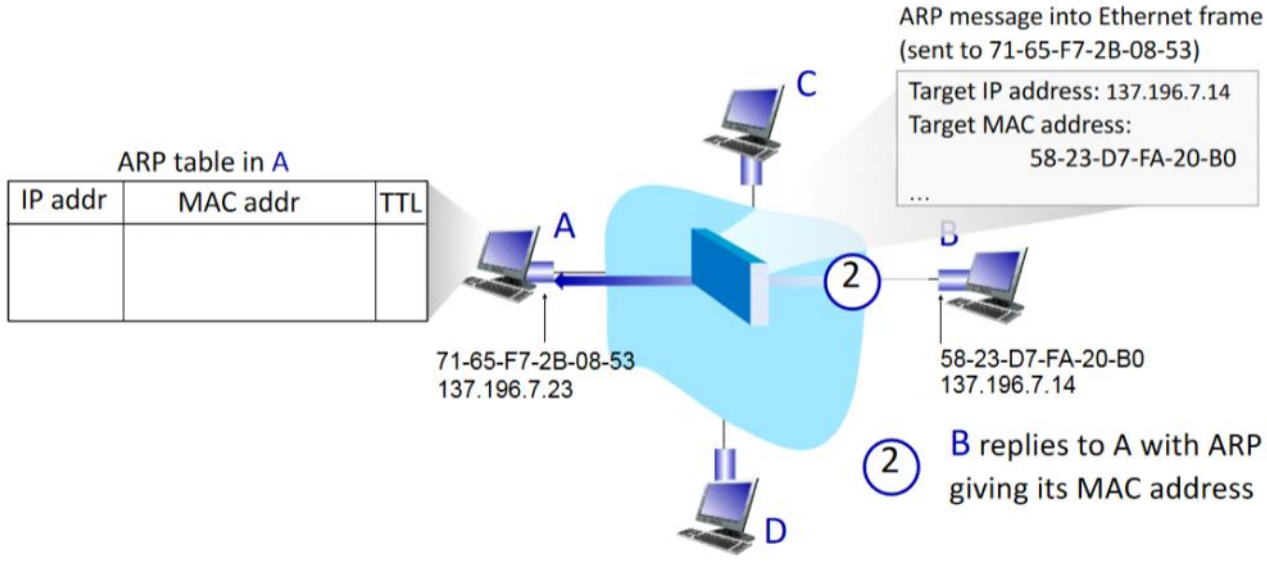


-You need to use 12 F's with the dashes on the exam

ARP protocol in action

example: A wants to send datagram to B

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



2 B replies to A with ARP response, giving its MAC address

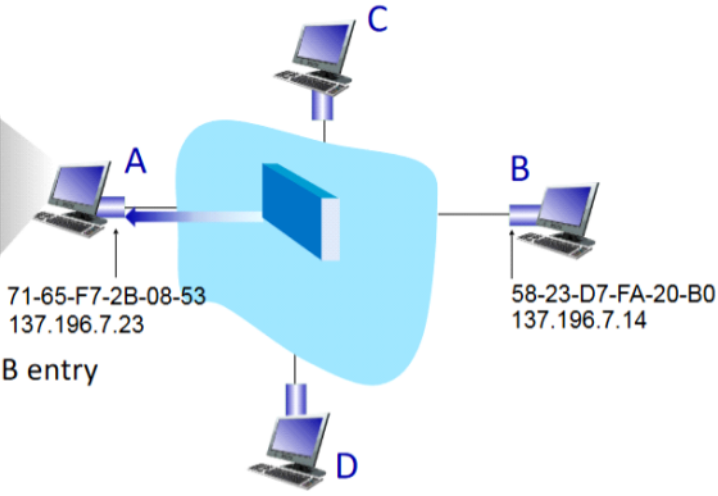
ARP protocol in action

example: A wants to send datagram to B

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

ARP table in A

IP addr	MAC addr	TTL
137.196.7.14	58-23-D7-FA-20-B0	500



3 A receives B's reply, adds B entry into its local ARP table

-How we find the right IP address, now factoring ARP into the equation

Determining Next Hop

- Destination on same subnet?
  - Local communication (Yes)
  - Remote Communication (No)
    - Forward frame to Gateway

49

-Finding an IP address if it's on the same subnet

Determining Next Hop (DestIP on Same Subnet)

- 10.0.0.1 255.0.0.0 wants to send a message to 10.0.0.2 255.0.0.0

	Host	Destination
IP	10.0.0.1	10.0.0.2
Subnet Mask	255.0.0.0	255.0.0.0
Network	10.0.0.0	10.0.0.0

0 0 0 0 1 0 1 0 . 0 0 0 0 0 0 0 0 . 0 0 0 0 0 0 0 0 . 0 0 0 0 0 0 0 1	Host IP
1 1 1 1 1 1 1 1 . 0 0 0 0 0 0 0 0 . 0 0 0 0 0 0 0 0 . 0 0 0 0 0 0 0 0	Host Mask
0 0 0 0 1 0 1 0 . 0 0 0 0 0 0 0 0 . 0 0 0 0 0 0 0 0 . 0 0 0 0 0 0 0 0	AND(IP, Mask)
0 0 0 0 1 0 1 0 . 0 0 0 0 0 0 0 0 . 0 0 0 0 0 0 0 0 . 0 0 0 0 0 0 1 0	DestIP
1 1 1 1 1 1 1 1 . 0 0 0 0 0 0 0 0 . 0 0 0 0 0 0 0 0 . 0 0 0 0 0 0 0 0	Dest Mask
0 0 0 0 1 0 1 0 . 0 0 0 0 0 0 0 0 . 0 0 0 0 0 0 0 0 . 0 0 0 0 0 0 0 0	AND(IP, Mask)

Destination IP on same subnet → Host Determines **Local** Communication

-Finding an IP address if it's not on the same subnet via remote communication over a gateway

Determining Next Hop (DestIP NOT on Same Subnet)

- 10.0.0.1 255.0.0.0 wants to send a message to 11.0.0.1 255.0.0.0

	Host	Destination
IP	10.0.0.1	11.0.0.1
Subnet Mask	255.0.0.0	255.0.0.0
Network	10.0.0.0	11.0.0.0

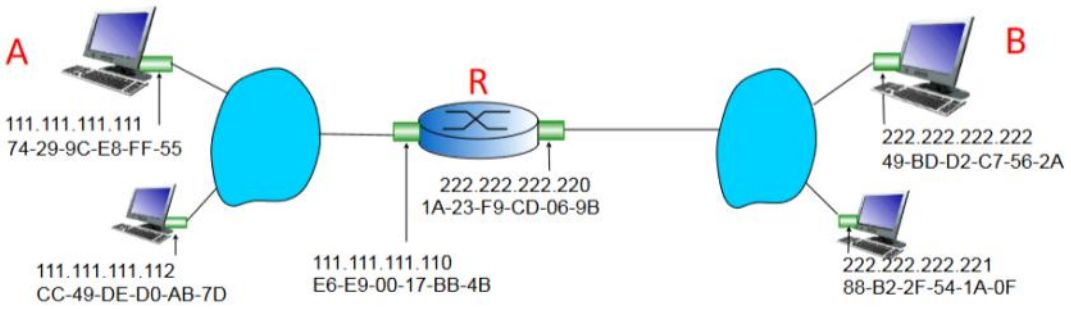
0 0 0 0 1 0 1 0 . 0 0 0 0 0 0 0 0 . 0 0 0 0 0 0 0 0 . 0 0 0 0 0 0 0 1	Host IP
1 1 1 1 1 1 1 1 . 0 0 0 0 0 0 0 0 . 0 0 0 0 0 0 0 0 . 0 0 0 0 0 0 0 0	Host Mask
0 0 0 0 1 0 1 0 . 0 0 0 0 0 0 0 0 . 0 0 0 0 0 0 0 0 . 0 0 0 0 0 0 0 0	AND(IP, Mask)
0 0 0 0 1 0 1 1 . 0 0 0 0 0 0 0 0 . 0 0 0 0 0 0 0 0 . 0 0 0 0 0 0 0 1	DestIP
1 1 1 1 1 1 1 1 . 0 0 0 0 0 0 0 0 . 0 0 0 0 0 0 0 0 . 0 0 0 0 0 0 0 0	Dest Mask
0 0 0 0 1 0 1 1 . 0 0 0 0 0 0 0 0 . 0 0 0 0 0 0 0 0 . 0 0 0 0 0 0 0 0	AND(IP, Mask)

Destination IP **NOT** on same subnet → Host Determines **Remote** Communication

Remote Communication (What happens at Gateway)

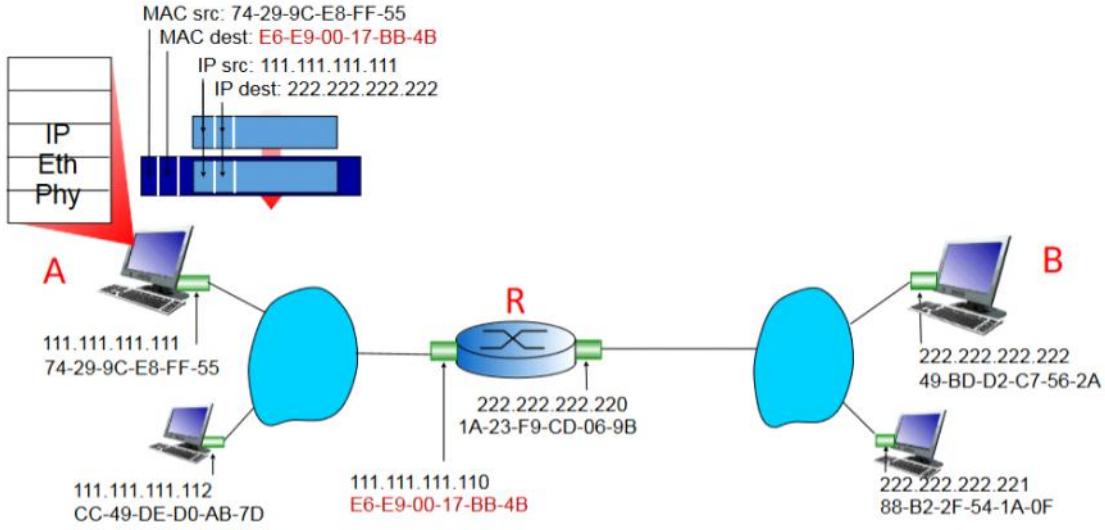
walkthrough: send IP datagram from A to B via R

- focus on addressing – at **IP / Networking layer (datagram)** and **MAC / Data Link layer (frame)**
- assume A knows B's IP address
- assume A knows **IP address** of first hop router, R (how?) → **ICMP router discovery message**
- assume A knows R's **MAC address** (how?) → **ARP**



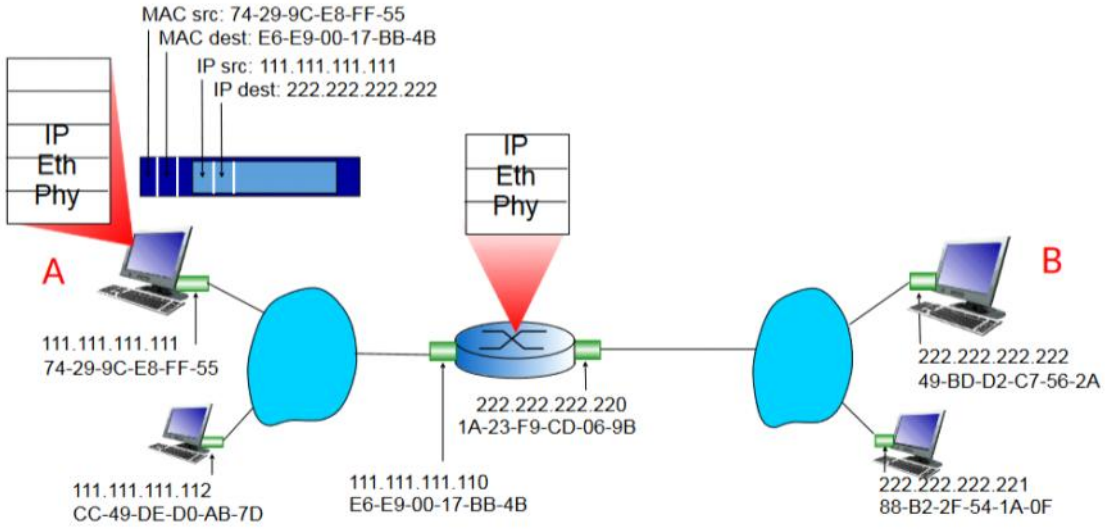
Remote Communication (What happens at Gateway)

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



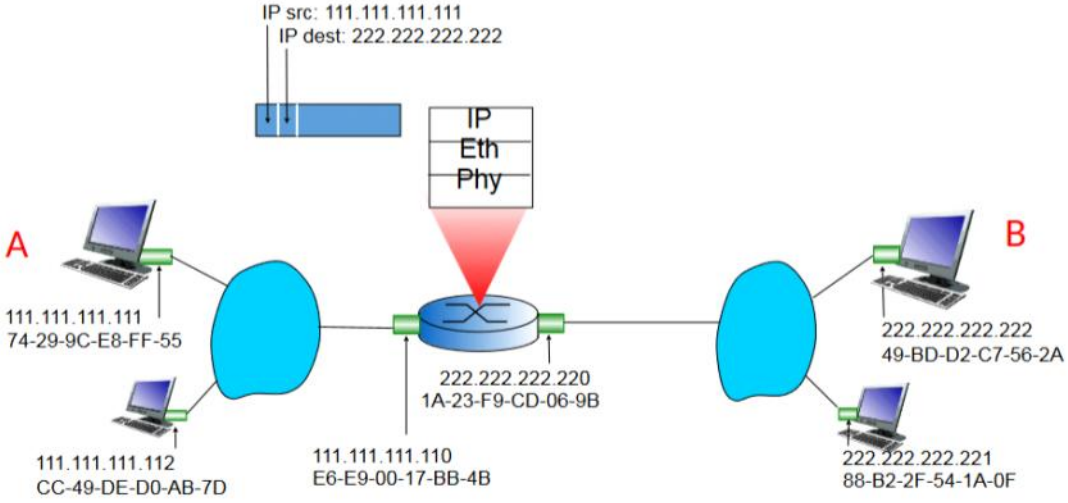
Remote Communication (What happens at Gateway)

- frame sent from A to R
- frame received at R



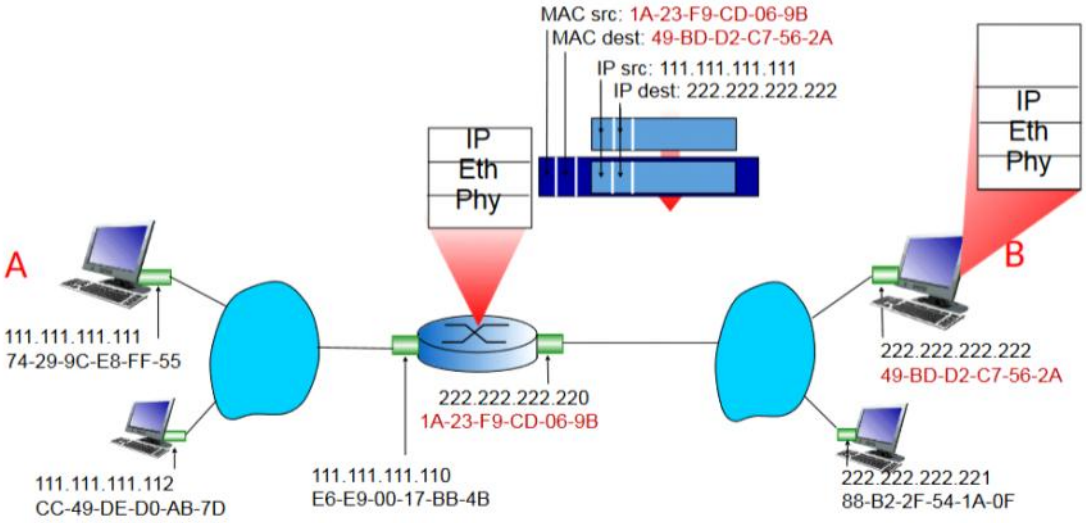
Remote Communication (What happens at Gateway)

- frame sent from A to R
- frame received at R, datagram removed, passed up to IP



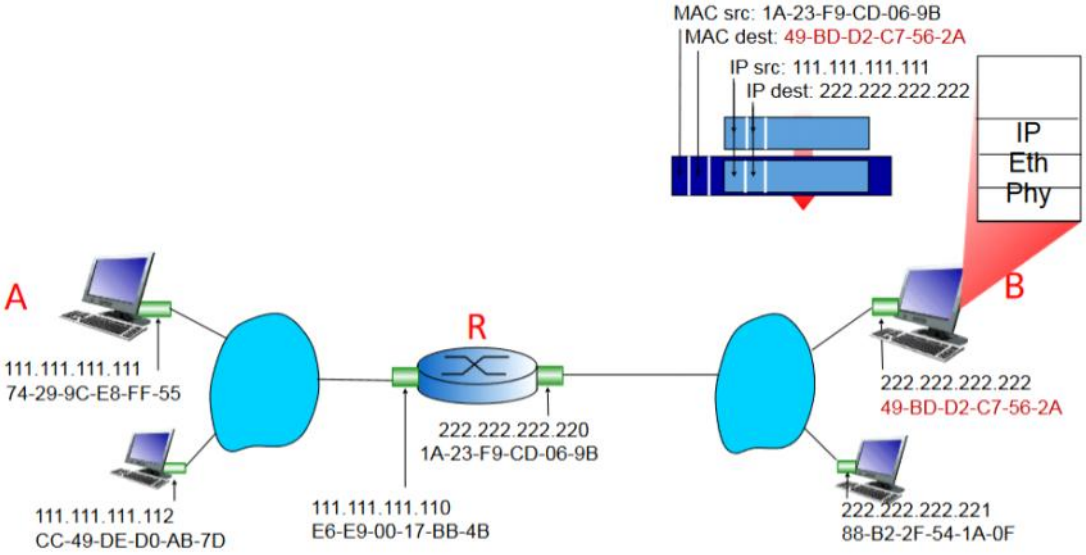
Remote Communication (What happens at Gateway)

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



Remote Communication (What happens at Gateway)

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



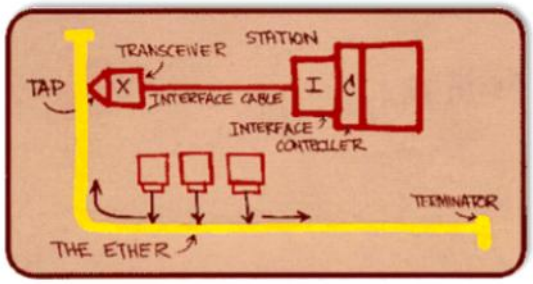
12/3 - Ethernet, Switches

Tuesday, December 3, 2024 5:15 PM

=== Ethernet ===
-What Ethernet is

Ethernet

"dominant" wired LAN technology:
- first widely used LAN technology
- simpler, cheap
- kept up with speed race: 10 Mbps – 400 Gbps
- single chip, multiple speeds (e.g., Broadcom BCM5761)



Metcalfe's Ethernet sketch

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

-How Ethernet connects computers together with buses and hubs

Ethernet: physical topology

- bus: popular through mid 90s
  - all nodes in **same collision domain** (can collide with each other)
- switched: prevails today
  - active link-layer 2 *switch* in center
  - each "spoke" runs a (separate) Ethernet protocol (nodes do not collide with each other)

bus: coaxial cable

switched

-Structure of an Ethernet frame

Ethernet frame structure

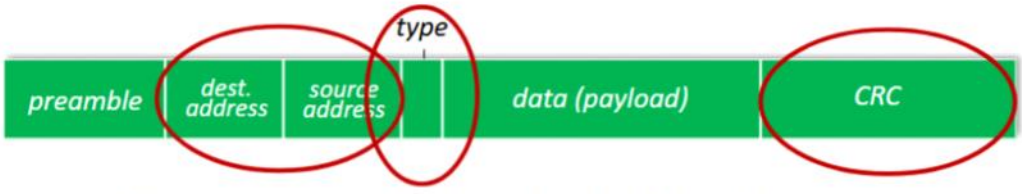
sending interface encapsulates IP datagram (or other network layer protocol packet) in Ethernet frame

preamble:

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

-ARP maps IP address to MAC address
-Type indicates upper layer protocol (ULP), this is done through demultiplexing

Ethernet frame structure (more)



- addresses: 6 byte source, destination MAC addresses
  - if adapter receives frame with **matching destination address, or with broadcast address** (e.g., ARP packet), it **passes data** in frame **to network layer** protocol
  - otherwise, adapter **discards** frame
- type: indicates higher layer protocol
  - mostly IP but others possible, e.g., Novell IPX, AppleTalk
  - used to demultiplex up at receiver
- CRC: cyclic redundancy check at receiver
  - error detected: frame is **dropped**

-Ethernet has no handshake and is unreliable (since TCP is already reliable), host-to-host connection is unreliable over Ethernet but end-to-end connection is reliable over TCP

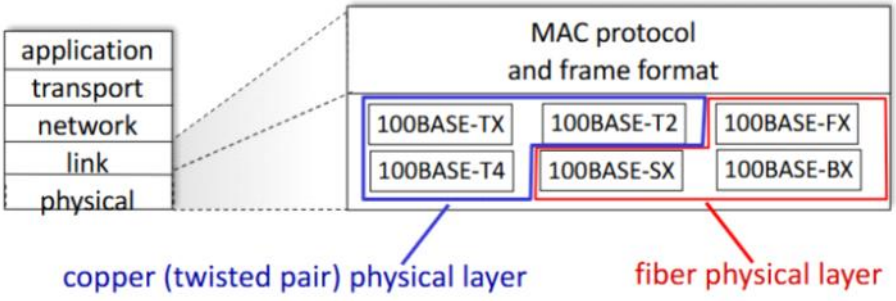
Ethernet: unreliable, connectionless

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

-Ethernet standard

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



=== SWITCHES ===

-Purpose of a switch is to store an Ethernet frame and forward it selectively to the right next hub or physically adjacent device

-Switches are plug-and-play, they do not need to be configured

Ethernet switch

- Switch is a link-layer device: takes an *active* role
  - store, forward Ethernet frames
  - examine incoming frame's MAC address, *selectively* forward frame

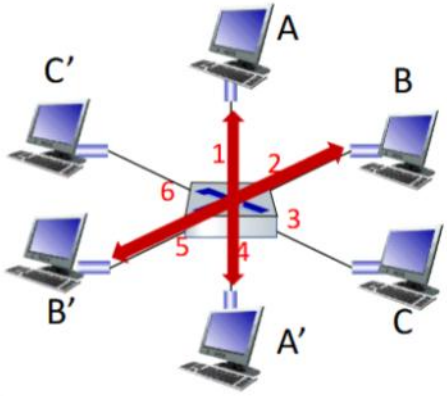
Ethernet switch

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

-Switches make it so each host gets its own broadcast domain with no collisions

Switch: multiple simultaneous transmissions

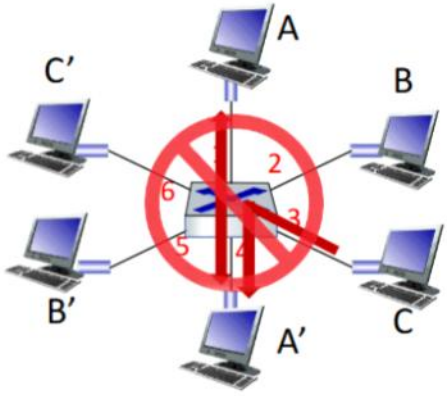
- hosts have dedicated, direct connection to switch
- switches buffer packets
- Ethernet protocol used on *each* incoming link, so:
  - **no collisions**; full duplex
  - each link is its own collision domain
- switching: A-to-A' and B-to-B' can transmit simultaneously, without collisions



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

Switch: multiple simultaneous transmissions

- hosts have dedicated, direct connection to switch
- switches buffer packets
- Ethernet protocol used on *each* incoming link, so:
  - no collisions; full duplex
  - each link is its own collision domain
- switching: A-to-A' and B-to-B' can transmit simultaneously, without collisions
  - but A-to-A' and C to A' can *not* happen simultaneously



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

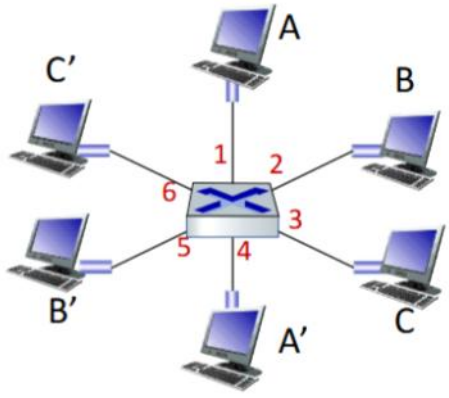
-Switch table (or switch forwarding table) is a cache that keeps track of MAC addresses with a TTL

Switch forwarding table

Q: how does switch **know A' reachable** via **interface 4**, B' reachable via interface 5?

A: each switch has a switch table, each entry:

- (MAC address of host, interface to reach host, time stamp)
- looks like a routing table!



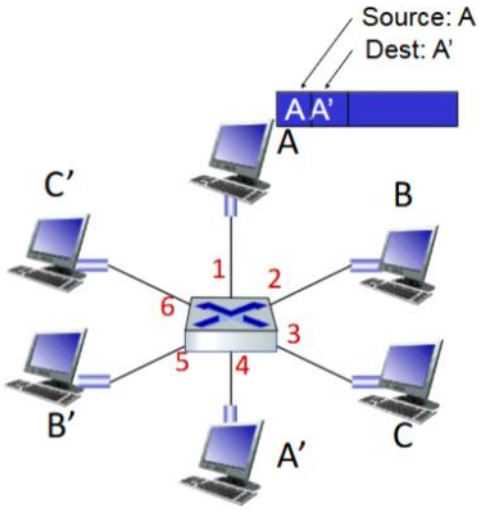
Q: how are entries created, maintained in switch table?

- something like a routing protocol?

-How the switch table works

Switch: self-learning

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



MAC addr	interface	TTL
A	*1*	*60*

Switch table (initially empty)

-Hub vs. switch

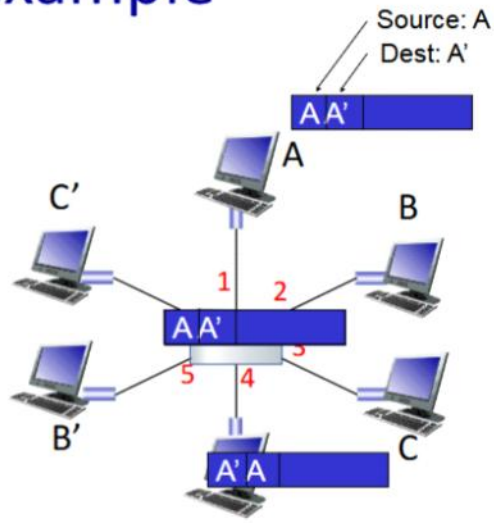
Switch: frame filtering/forwarding

when **frame received** at switch:

1. record incoming link, MAC address of sending host
2. index switch table using MAC destination address
3. if entry found for destination
   then {
   if destination on segment from which frame arrived
   then drop frame
   else forward frame on interface indicated by entry
   }
   else flood /* forward on all interfaces except arriving interface */

Self-learning, forwarding: example

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



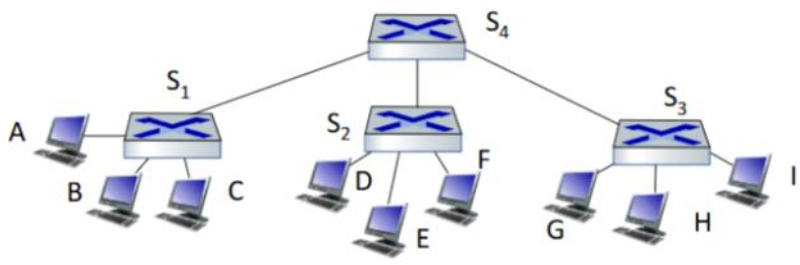
MAC addr	interface	TTL
A	1	60
A'	4	60

switch table (initially empty)

-You can connect switches to other switches and it still works

Interconnecting switches

self-learning switches can be connected together:



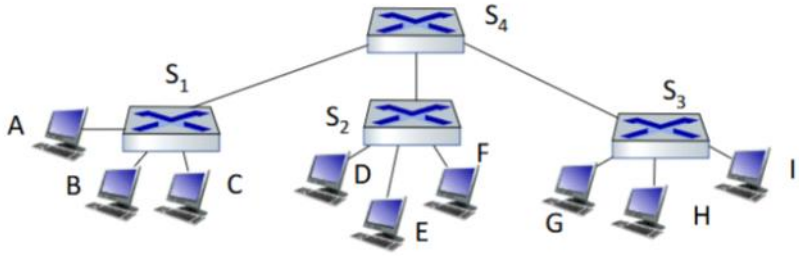
Q: sending from A to G - how does S_1 know to forward frame destined to G via S_4 and S_3?

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

-Example of interconnecting multiple switches together

Self-learning multi-switch example

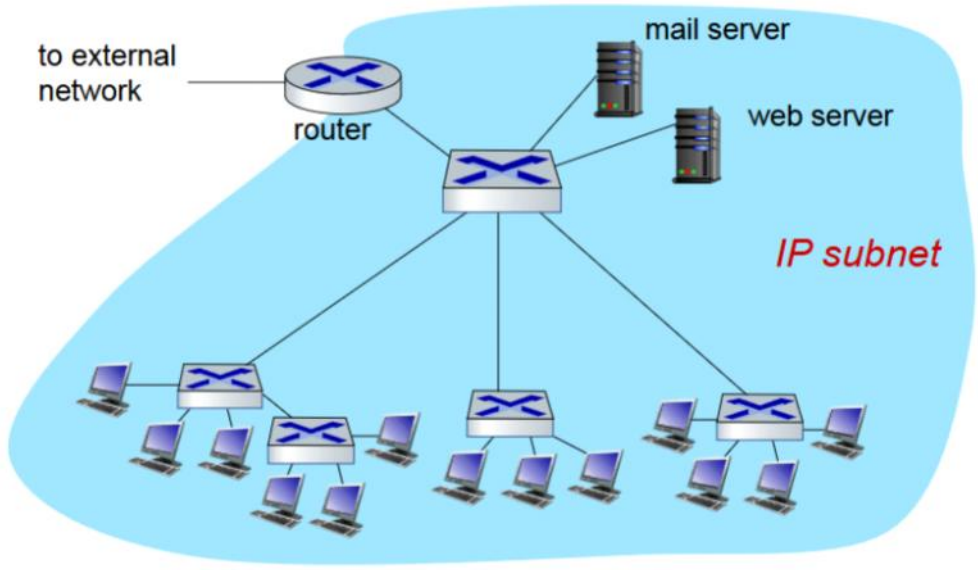
Suppose C sends frame to I, I responds to C



Q: show switch tables and packet forwarding in S_1, S_2, S_3, S_4

-A small institutional network is lab a computer lab worth of computers connected to a switch, several of these switches in various computer labs connected to a larger switch which connects to servers on the subnet and has an edge/gateway router at the end of the IP subnet

Small institutional network



-Switch vs router

-Both store-and-forward, but the router is on Network Layer but the switch is on the Link/Data Layer (which is why routers understand IP but switches do not)

-The switch is made by self-learn and the router is made by routing algorithms

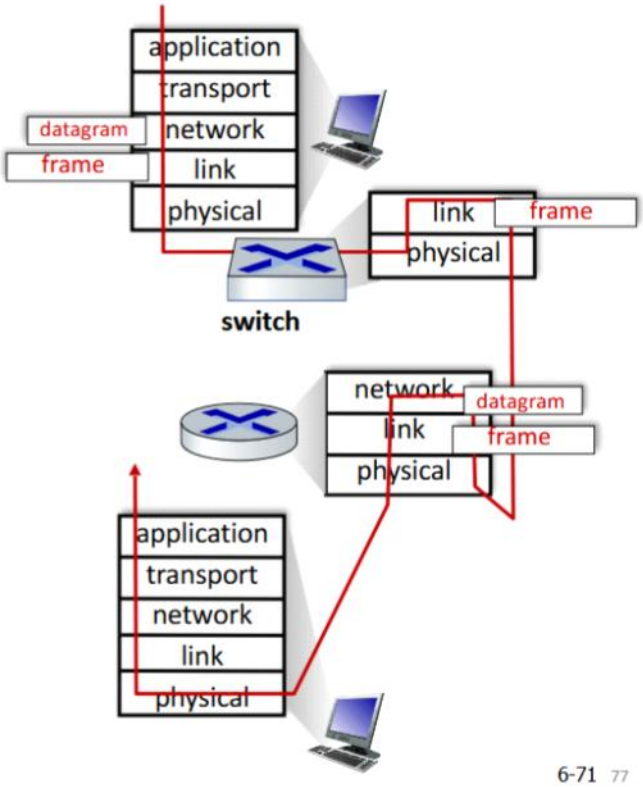
Switches vs. routers

both are store-and-forward:

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

both have forwarding tables:

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



6-71 77