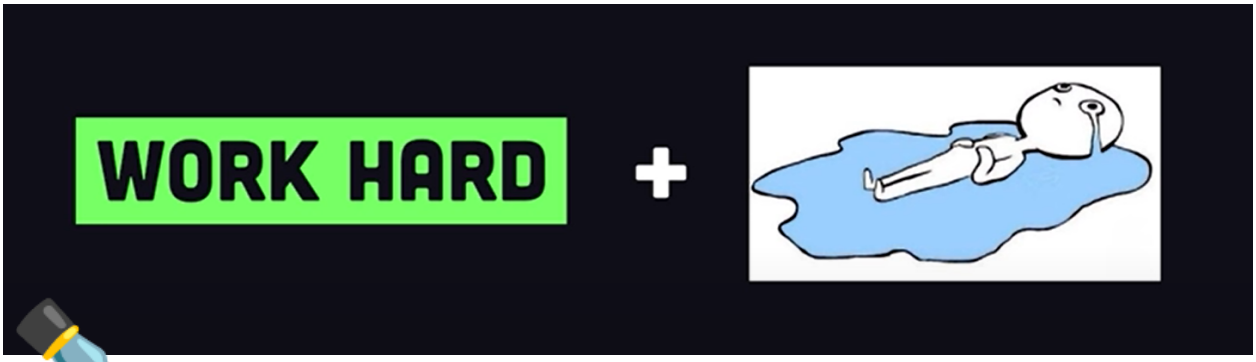


Cheat Sheet CN

Created	@September 27, 2023 4:19 PM
Tags	Cheat Sheet Computer Networks
Created by	Muhammad Ali Zahid

Key points

- 1 byte=8 bits
- throughput= file-size/ delay

Formula

$$m/s = L/R$$

$$d_{p.p} = m/s$$

$$BandwidthDela = R * d_{p.p}$$

$$BandwidthDela = R * d_{p.p}$$

▼ Ch 1: Computer Networks and the internet

- Network edge: hosts, access network, physical media
- Network core: packet/circuit switching, internet structure

- Performance: loss, delay, throughput
- **a “nuts and bolts” view**
  - Billions of connected computing *devices*:

```
hosts = end systems
```

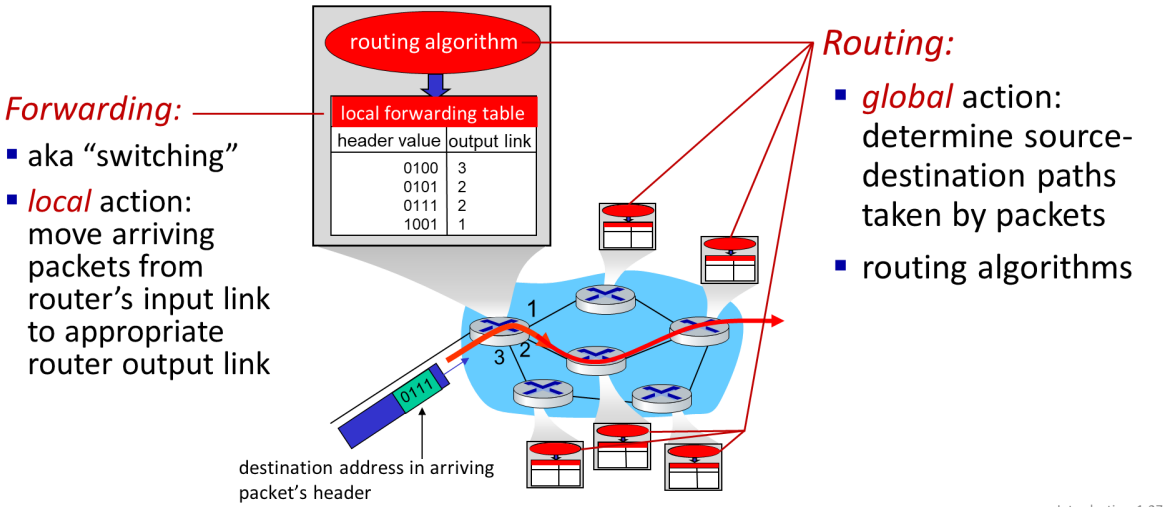
  - running *network apps* at Internet’s “edge”
- *Protocols define the format, order of messages sent and received among network entities, and actions taken on message transmission, receipt*
- **Network edge:**
  - hosts: clients and servers
  - servers often in data centers
- **Access networks, physical media:**
  - wired, wireless communication links
- **Network core:**
  - interconnected routers
  - network of networks
- ***How to connect end systems to edge router?***
  - residential access nets
  - institutional access networks (school, company)
  - mobile access networks (WiFi, 4G/5G)
- ***frequency division multiplexing (FDM):*** different channels transmitted in different frequency bands
- **HFC: hybrid fiber coax**
  - asymmetric: up to 40 Mbps – 1.2 Gbps downstream transmission rate, 30-100 Mbps upstream transmission rate

- **host sending function:**
  - takes application message
  - breaks into smaller chunks, known as *packets*, of length *L* bits
  - transmits packet into access network at *transmission rate R*
  - link transmission rate, aka link *capacity, aka link bandwidth*

$$\text{packet transmission delay} = \text{time needed to transmit } L\text{-bit packet into link} = \frac{L \text{ (bits)}}{R \text{ (bits/sec)}}$$

- **packet-switching:** hosts break application-layer messages into *packets*
  - network forwards packets from one router to the next, across links on path from source to destination

Two key network-core functions

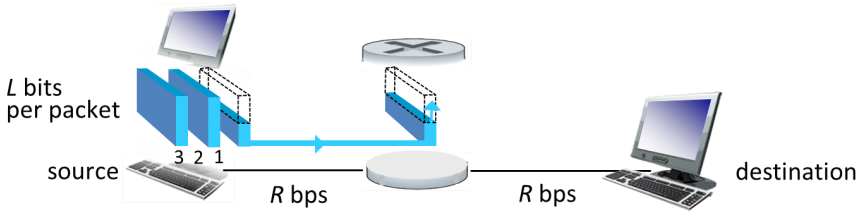


Forwarding:
- aka "switching"
- *local* action: move arriving packets from router's input link to appropriate router output link

Routing:
- *global* action: determine source-destination paths taken by packets
- routing algorithms

- **Routing:** Planning the best path for data packets from source to destination in a network.
- **Forwarding:** Executing the chosen path, physically moving data packets along that route.

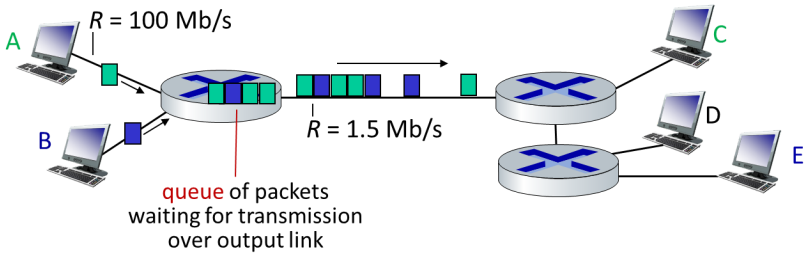
Packet-switching: store-and-forward



- packet transmission delay: takes L/R seconds to transmit (push out) L-bit packet into link at R bps
- *store and forward: entire* packet must arrive at router before it can be transmitted on next link

One-hop numerical example:
- L = 10 Kbits
- R = 100 Mbps
- one-hop transmission delay = 0.1 msec

Packet-switching: queueing



Queueing occurs when work arrives faster than it can be serviced:

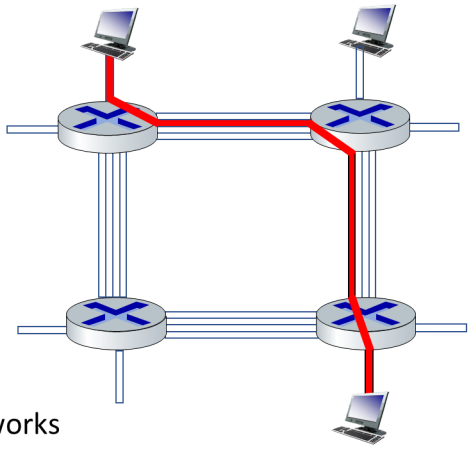


- ***Packet queuing and loss:*** if arrival rate (in bps) to link exceeds transmission rate (bps) of link for some period of time:
  - packets will queue, waiting to be transmitted on output link
  - packets can be dropped (lost) if memory (buffer) in router fills up

Alternative to packet switching: circuit switching

end-end resources allocated to, reserved for "call" between source and destination

- in diagram, each link has four circuits.
  - call gets 2nd circuit in top link and 1st circuit in right link.
- dedicated resources: no sharing
  - circuit-like (guaranteed) performance
- circuit segment idle if not used by call (no sharing)
- commonly used in traditional telephone networks

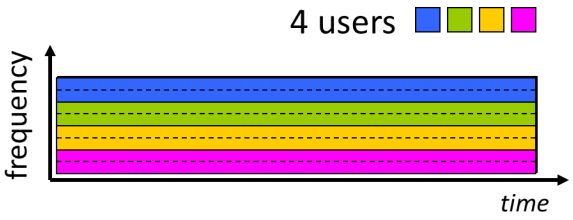


* Check out the online interactive exercises for more examples: http://gaia.cs.umass.edu/kurose_ross/interactive

Circuit switching: FDM and TDM

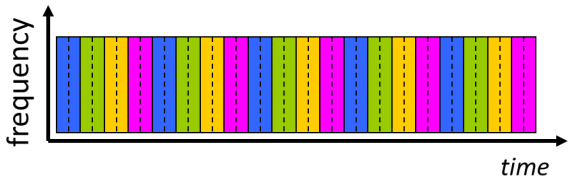
Frequency Division Multiplexing (FDM)

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



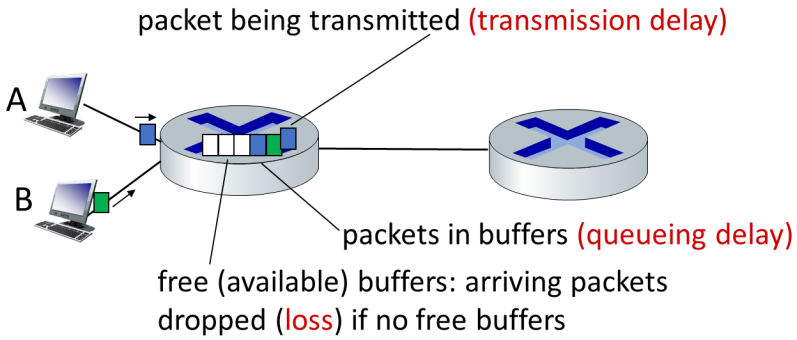
Time Division Multiplexing (TDM)

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

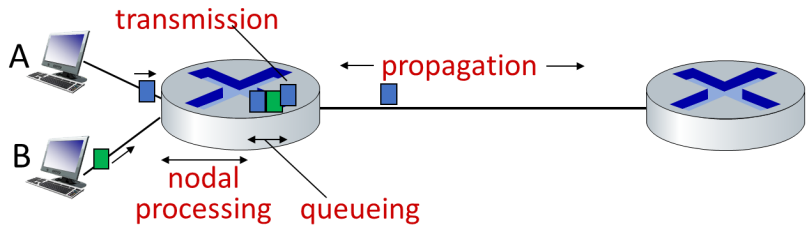


How do packet delay and loss occur?

- packets *queue* in router buffers, waiting for turn for transmission
  - queue length grows when arrival rate to link (temporarily) exceeds output link capacity
- packet *loss* occurs when memory to hold queued packets fills up



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 < microsecs

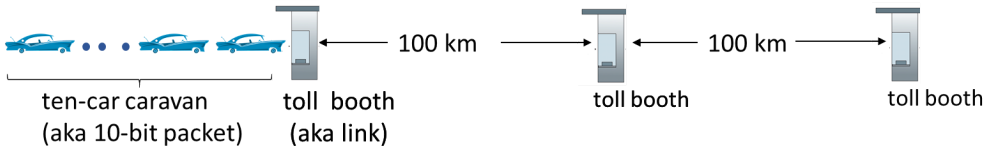
d_{queue}: queueing delay
- time waiting at output link for transmission
- depends on congestion level of router

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

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

Caravan analogy



- car ~ bit; caravan ~ packet; toll service ~ link transmission
- toll booth takes 12 sec to service car (bit transmission time)
- "propagate" at 100 km/hr
- *Q:* How long until caravan is lined up before 2nd toll booth?
- time to "push" entire caravan through toll booth onto highway = 12*10 = 120 sec
- time for last car to propagate from 1st to 2nd toll both: 100km/(100km/hr) = 1 hr
- *A:* 62 minutes

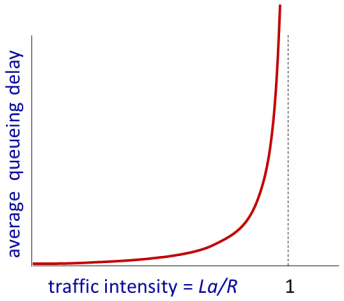
- suppose cars now "propagate" at 1000 km/hr
- and suppose toll booth now takes one min to service a car
- *Q:* Will cars arrive to 2nd booth before all cars serviced at first booth?

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

Packet queueing delay (revisited)

- *a:* average packet arrival rate
- *L:* packet length (bits)
- *R:* link bandwidth (bit transmission rate)

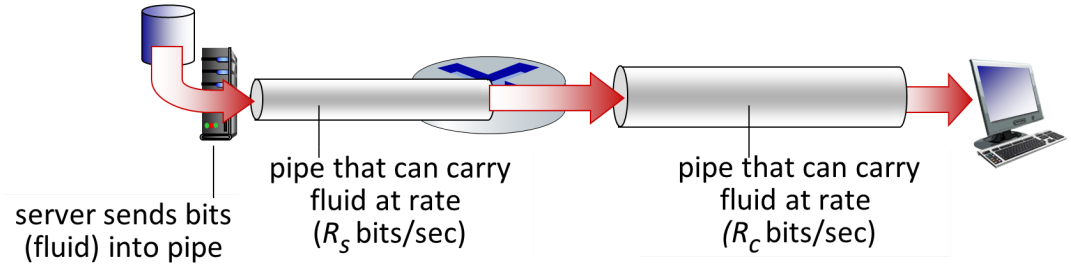
$$\frac{L \cdot a}{R} : \frac{\text{arrival rate of bits}}{\text{service rate of bits}}$$ *"traffic intensity"*



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

Throughput

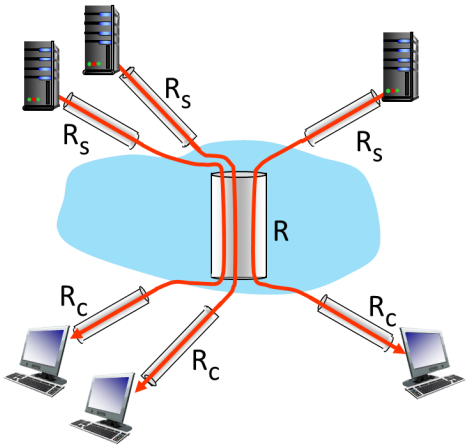
- *throughput:* rate (bits/time unit) at which bits are being sent from sender to receiver
  - *instantaneous:* rate at given point in time
  - *average:* rate over longer period of time



bottleneck link

link on end-end path that constrains end-end throughput

Throughput: network scenario



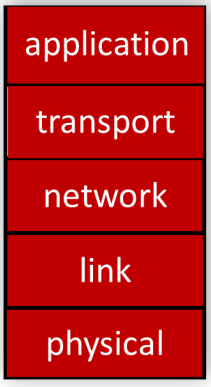
10 connections (fairly) share backbone bottleneck link *R* bits/sec

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

Layered Internet protocol stack

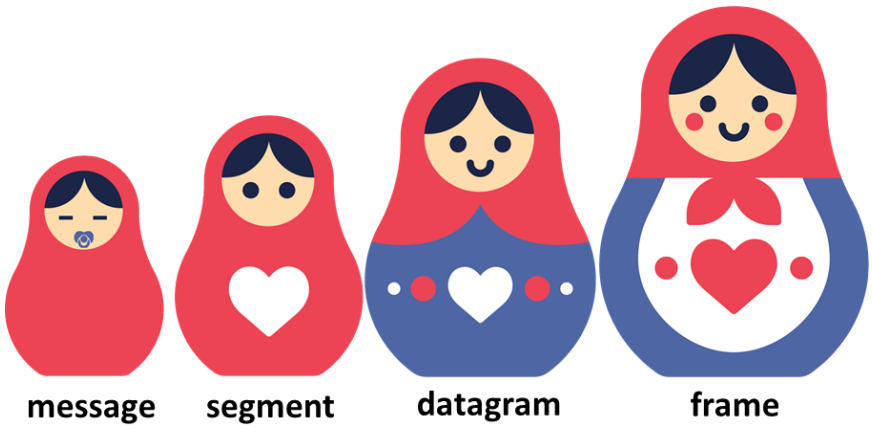
- *application:* supporting network applications
  - HTTP, IMAP, SMTP, DNS
- *transport:* process-process data transfer
  - TCP, UDP
- *network:* routing of datagrams from source to destination
  - IP, routing protocols
- *link:* data transfer between neighboring network elements
  - Ethernet, 802.11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



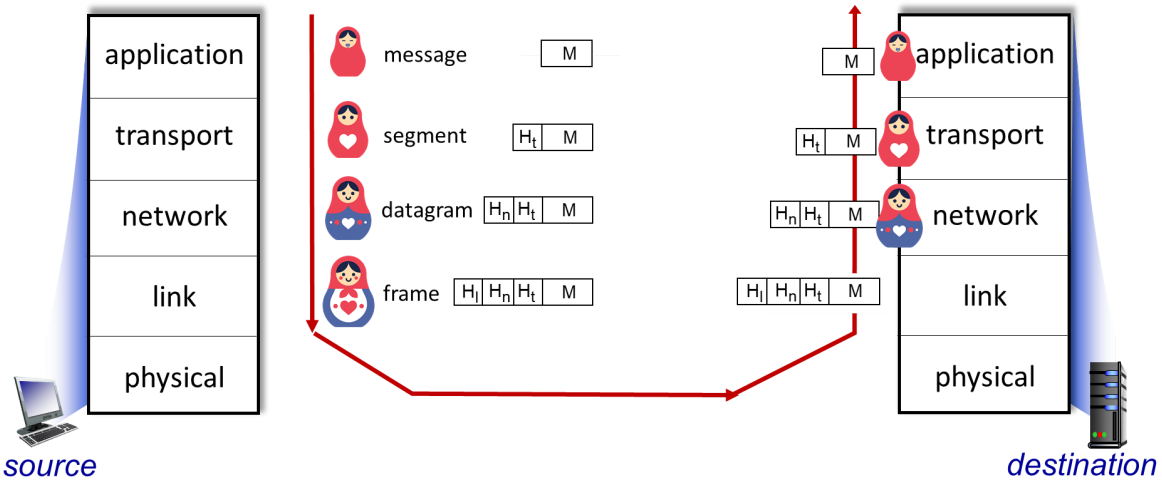
- Application exchanges **msg** to implement some application service using service of transport layer
- transport-layer protocol encapsulates application-layer message, with *transport* layer-layer header to create a

transport-layer **segment**

- network-layer protocol encapsulates transport-layer segment with network layer-layer header to create a network-layer **datagram**
- link-layer protocol encapsulates network datagram with link-layer header to create a link-layer **frame**



Services, Layering and Encapsulation



Introduction: 1-76

▼ Ch 2: Application Layer

- server:

- always-on host
- permanent IP address
- often in data centers, for scaling

- clients:
  - contact, communicate with server
  - may be intermittently connected
  - may have dynamic IP addresses
  - do *not* communicate directly with each other
  - examples: HTTP, IMAP, FTP
- process sends/receives messages to/from its socket
- socket analogous to door
  - sending process shoves message out door
  - sending process relies on transport infrastructure on other side of door to deliver message to socket at receiving process
  - two sockets involved: one on each side

- to receive messages, process must have *identifier*
- host device has unique 32-bit IP address
- *Q:* does IP address of host on which process runs suffice for identifying the process?

- *identifier* includes both IP address and port numbers associated with process on host.
- example port numbers:
  - HTTP server: 80
  - mail server: 25
- to send HTTP message to gaia.cs.umass.edu web server:
  - IP address: 128.119.245.12
  - port number: 80
- more shortly…

An application-layer protocol defines:

- types of messages exchanged,
  - e.g., request, response
- message syntax:
  - what fields in messages & how fields are delineated
- message semantics
  - meaning of information in fields
- rules for when and how processes send & respond to messages

open protocols:

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

proprietary protocols:

- e.g., Skype, Zoom

What transport service does an app need?

data integrity

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

timing

- some apps (e.g., Internet telephony, interactive games) require low delay to be "effective"

throughput

- some apps (e.g., multimedia) require minimum amount of throughput to be "effective"
- other apps ("elastic apps") make use of whatever throughput they get

security

- encryption, data integrity, …

TCP service:

- *reliable transport* between sending and receiving process
- *flow control:* sender won't overwhelm receiver
- *congestion control:* throttle sender when network overloaded
- *connection-oriented:* setup required between client and server processes
- *does not provide:* timing, minimum throughput guarantee, security

UDP service:

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

Q: why bother? *Why* is there a UDP?

Transport service requirements: common apps

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

Internet 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 [RFC 7230, 9110]	TCP
Internet telephony	SIP [RFC 3261], RTP [RFC 3550], or proprietary	TCP or UDP
streaming audio/video	HTTP [RFC 7230], DASH	TCP
interactive games	WOW, FPS (proprietary)	UDP or TCP

- Vanilla TCP & UDP sockets:
  - no encryption
  - cleartext passwords sent into socket traverse Internet in cleartext (!)
- Transport Layer Security (TLS)
  - provides encrypted TCP connections
  - data integrity
  - end-point authentication

Web and HTTP

First, a quick review…

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

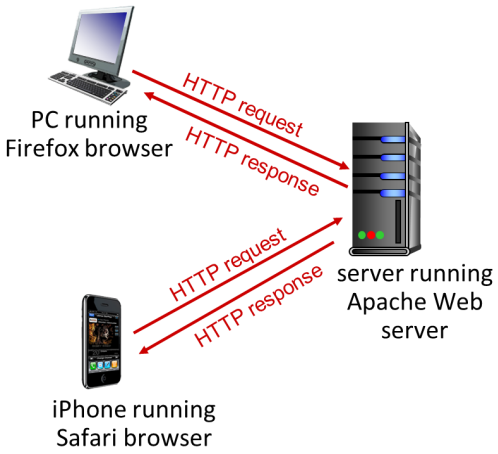
`www.someschool.edu/someDept/pic.gif`

host name | path name

HTTP overview

HTTP: hypertext transfer protocol

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



- *HTTP uses TCP:*
  - client initiates TCP connection (creates socket) to server, port 80
  - server accepts TCP connection from client

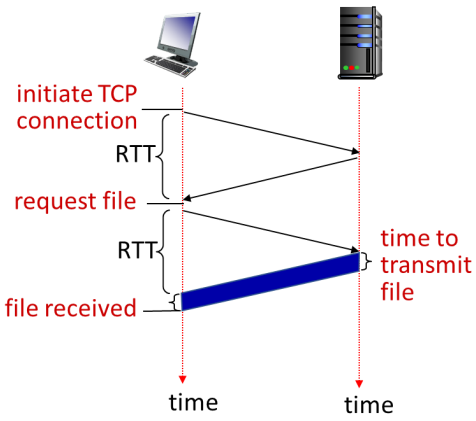
    - HTTP messages (application-layer protocol messages) exchanged between browser (HTTP client) and Web server (HTTP server)
    - TCP connection closed
- *HTTP is “stateless”*
    - server maintains *no* information about past client requests
- **HTTP connections: two types**
    - *Non-persistent HTTP*
        - TCP connection opened
        - at most one object sent over TCP connection
        - TCP connection closed
        - downloading multiple objects required multiple connections
    - *Persistent HTTP*
        - TCP connection opened to a server
        - multiple objects can be sent over *single* TCP connection between client, and that server
        - TCP connection closed

Non-persistent HTTP: response time

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

HTTP response time (per object):

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



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

Persistent HTTP (HTTP 1.1)

Non-persistent HTTP issues:

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

Persistent HTTP (HTTP1.1):

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

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.

PUT method:

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

- status code appears in 1st line in server-to-client response message.

  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

Maintaining user/server state: cookies

Web sites and client browser use *cookies* to maintain some state between transactions

four components:

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

Example:

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

HTTP cookies: comments

What cookies can be used for:

- authorization
- shopping carts
- recommendations
- user session state (Web e-mail)

Challenge: How to keep state?

- *at protocol endpoints:* maintain state at sender/receiver over multiple transactions
- *in messages:* cookies in HTTP messages carry state

aside

cookies and privacy:

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

- Cookies can be used to:
  - track user behavior on a given website (first party cookies)

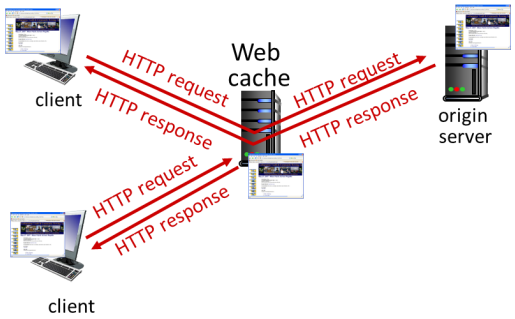
- track user behavior across multiple websites (third party cookies) without user ever choosing to visit tracker site (!)
- tracking may be *invisible* to user:
- rather than displayed ad triggering HTTP GET to tracker, could be an invisible link

- third party tracking via cookies:
  - disabled by default in Firefox, Safari browsers
  - to be disabled in Chrome browser in 2023

Web caches

Goal: satisfy client requests without involving origin server

- user configures browser to point to a (local) *Web cache*
- browser sends all HTTP requests to cache
  - *if* object in cache: cache returns object to client
  - *else* cache requests object from origin server, caches received object, then returns object to client



Web caches (aka proxy servers)

- Web cache acts as both client and server
  - server for original requesting client
  - client to origin server
- server tells cache about object's allowable caching in response header:

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

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

Why Web caching?

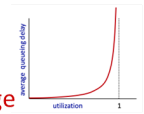
- reduce response time for client request
  - cache is closer to client
- reduce traffic on an institution's access link
- Internet is dense with caches
  - enables "poor" content providers to more effectively deliver content

- `Example`

Caching example

Scenario:

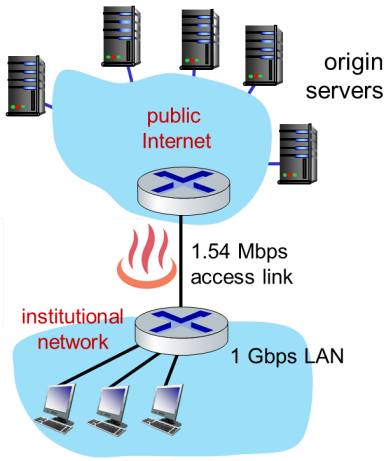
- access link rate: 1.54 Mbps
- RTT from institutional router to server: 2 sec
- web object size: 100K bits
- average request rate from browsers to origin servers: 15/sec
  - avg data rate to browsers: 1.50 Mbps



Performance:

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

problem: large queueing delays at high utilization!



Option 1: buy a faster access link

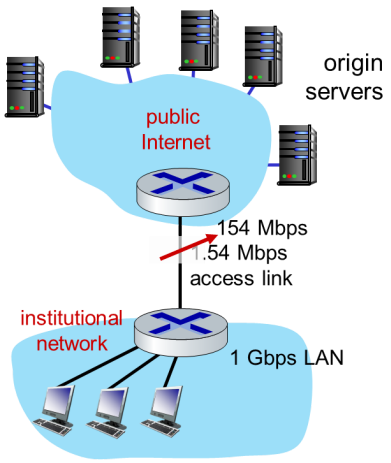
Scenario:

- access link rate: 1.54 Mbps → 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:

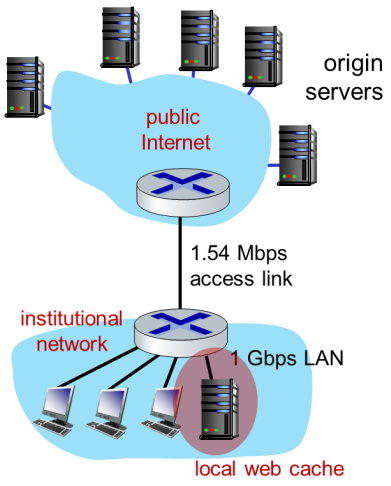
- access link rate: 1.54 Mbps
- RTT from institutional router to server: 2 sec
- web object size: 100K bits
- average request rate from browsers to origin servers: 15/sec
  - avg data rate to browsers: 1.50 Mbps

Cost: web cache (cheap!)

Performance:

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

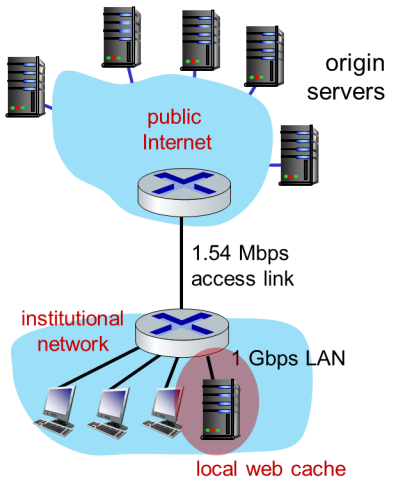
How to compute link utilization, delay?



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



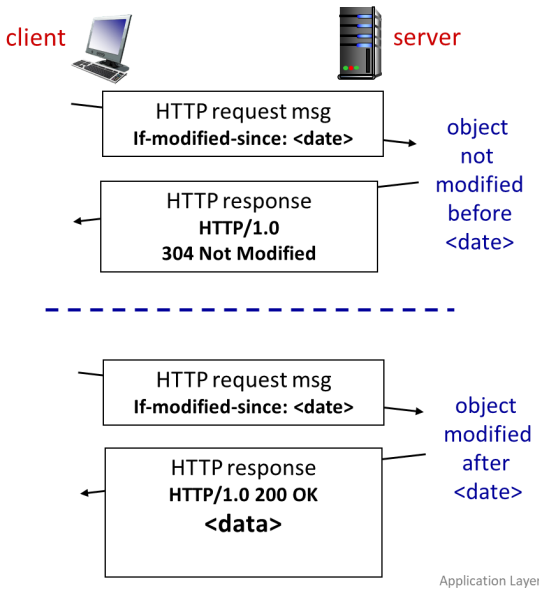
lower average end-end delay than with 154 Mbps link (and cheaper too!)

Browser caching: Conditional GET

Goal: don't send object if browser has up-to-date cached version

- no object transmission delay (or use of network resources)

- *client:* specify date of browser-cached copy in HTTP request
  If-modified-since: <date>
- *server:* response contains no object if browser-cached copy is up-to-date:
  HTTP/1.0 304 Not Modified



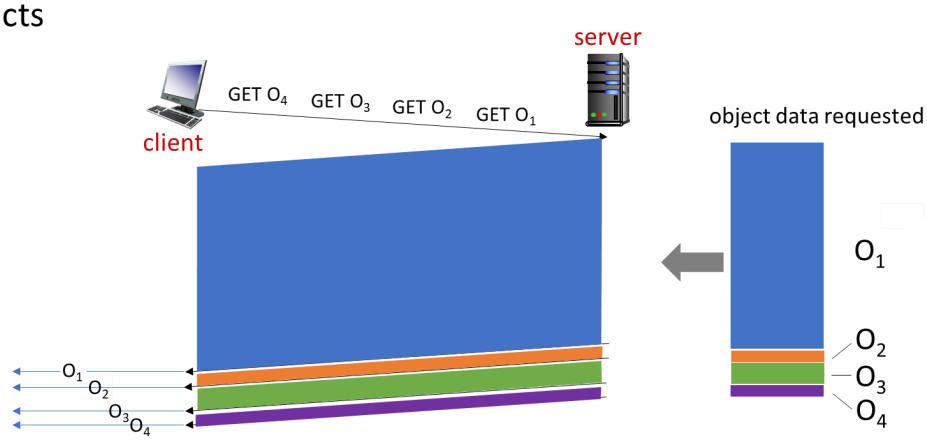
- ***HTTP1.1:*** introduced multiple, pipelined GETs over single TCP connection
  - server responds *in-order* (FCFS: first-come-first-served scheduling) to GET requests

- with FCFS, small object may have to wait for transmission (head-of-line (HOL) blocking) behind large object(s)
- loss recovery (retransmitting lost TCP segments) stalls object transmission

- ***HTTP/2:*** increased flexibility at *server* in sending objects to client:
  - methods, status codes, most header fields unchanged from HTTP 1.1
  - transmission order of requested objects based on client-specified object priority (not necessarily FCFS)
  - *push* unrequested objects to client
  - divide objects into frames, schedule frames to mitigate HOL blocking

HTTP/2: mitigating HOL blocking

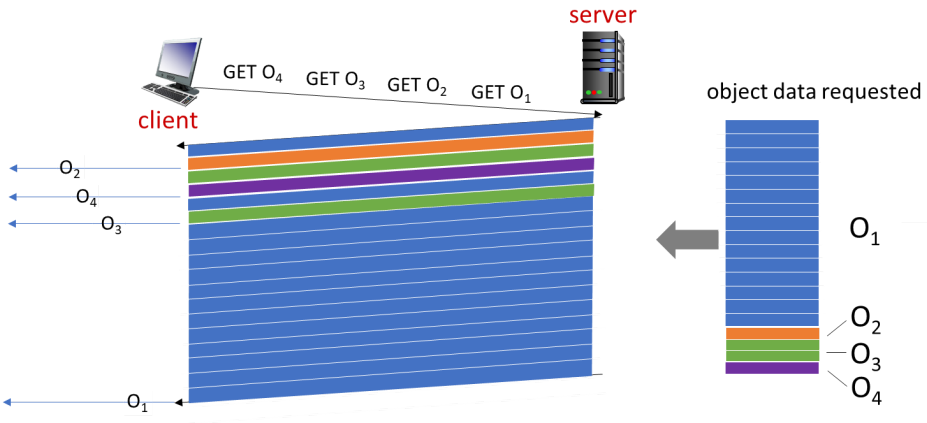
HTTP 1.1: client requests 1 large object (e.g., video file) and 3 smaller objects



objects delivered in order requested: O_2, O_3, O_4 wait behind O_1

HTTP/2: mitigating HOL blocking

HTTP/2: objects divided into frames, frame transmission interleaved



O_2, O_3, O_4 delivered quickly, O_1 slightly delayed

- **HTTP/2** over single TCP connection means:
  - recovery from packet loss still stalls all object transmissions
  - as in HTTP 1.1, browsers have incentive to open multiple parallel TCP connections to reduce stalling, increase overall throughput
  - no security over vanilla TCP connection
- **HTTP/3:** adds security, per object error- and congestion-control (more pipelining) over UDP
  - more on HTTP/3 in transport layer
- Three major components:
  - user agents
  - mail servers
  - **simple mail transfer protocol: SMTP**
- User Agent
  - a.k.a. “mail reader”

    - composing, editing, reading mail messages
    - e.g., Outlook, iPhone mail client
    - outgoing, incoming messages stored on server
- mail servers:
    - *mailbox* contains incoming messages for user
    - *message queue* of outgoing (to be sent) mail messages
- SMTP protocol between mail servers to send email messages
    - client: sending mail server
    - “server”: receiving mail server
- uses TCP to reliably transfer email message from client (mail server initiating connection) to server, port 25
    - direct transfer: sending server (acting like client) to receiving server
- three phases of transfer
    - SMTP handshaking (greeting)
    - SMTP transfer of messages
    - SMTP closure
- command/response interaction (like HTTP)
    - commands: ASCII text
    - response: status code and phrase

Scenario: Alice sends e-mail to Bob

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

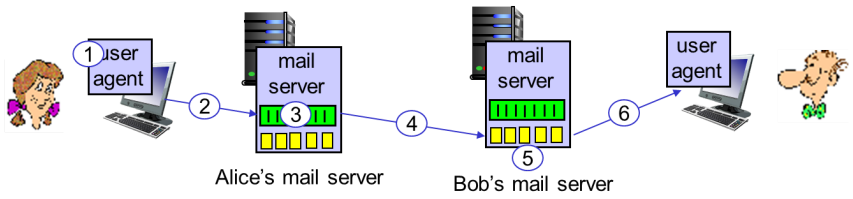
2) Alice's UA sends message to her mail server using SMTP; message placed in message queue

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

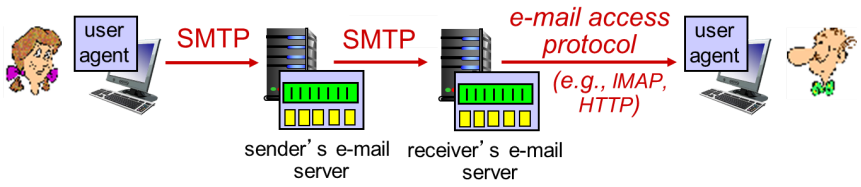
5) Bob's mail server places the message in Bob's mailbox

6) Bob invokes his user agent to read message



- *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
  - 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

Retrieving email: mail access protocols

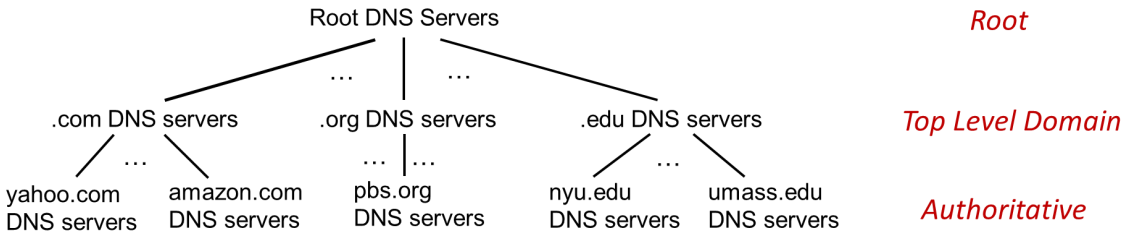


- SMTP: delivery/storage of e-mail messages to receiver's server
- mail access protocol: retrieval from server
  - IMAP: Internet Mail Access Protocol [RFC 3501]: messages stored on server, IMAP provides retrieval, deletion, folders of stored messages on server
- HTTP: gmail, Hotmail, Yahoo!Mail, etc. provides web-based interface on top of STMP (to send), IMAP (or POP) to retrieve e-mail messages

- **Domain Name System (DNS):**
  - *distributed database* implemented in hierarchy of many *name servers*
  - *application-layer protocol:* hosts, DNS servers communicate to *resolve* names (address/name translation)
  - *note:* core Internet function, implemented as application-layer protocol
  - complexity at network's "edge"
- DNS services:
  - hostname-to-IP-address translation
  - host aliasing
  - canonical, alias names
  - mail server aliasing
  - load distribution
  - replicated Web servers: many IP addresses correspond to one name

- *Q: Why not centralize DNS?*
  - single point of failure
  - traffic volume
  - distant centralized database
  - maintenance
- *A: doesn't scale!*
  - Comcast DNS servers alone: 600B DNS queries/day
  - Akamai DNS servers alone: 2.2T DNS queries/day

DNS: a distributed, hierarchical database



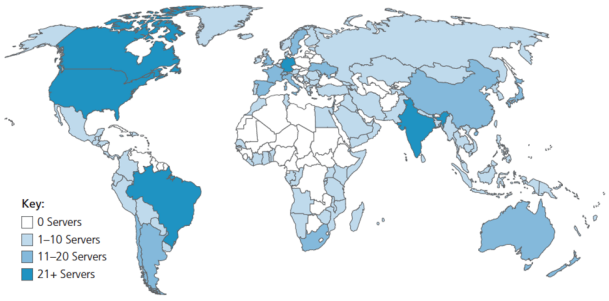
Client wants IP address for www.amazon.com; 1st approximation:

- client queries root server to find .com DNS server
- client queries .com DNS server to get amazon.com DNS server
- client queries amazon.com DNS server to get IP address for www.amazon.com

DNS: root name servers

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

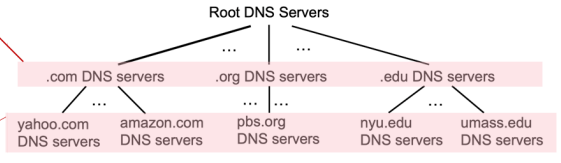
13 logical root name "servers" worldwide each "server" replicated many times (~200 servers in US)



Top-Level Domain, and authoritative servers

Top-Level Domain (TLD) servers:

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



authoritative DNS servers:

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

Local DNS name servers

- when host makes DNS query, it is sent to its *local* DNS server
  - Local DNS server returns reply, answering:
    - from its local cache of recent name-to-address translation pairs (possibly out of date!)
    - forwarding request into DNS hierarchy for resolution
  - each ISP has local DNS name server; to find yours:
    - MacOS: `% scutil --dns`
    - Windows: `>ipconfig /all`
- local DNS server doesn't strictly belong to hierarchy

- once (any) name server learns mapping, it *caches* mapping, and i*mmediately* returns a cached mapping in response to a query
  - caching improves response time
  - cache entries timeout (disappear) after some time (TTL)
  - TLD servers typically cached in local name servers
- cached entries may be *out-of-date*
  - if named host changes IP address, may not be known Internet-wide until all TTLs expire!
  - *best-effort name-to-address translation!*

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`

- ▼ **mid 1(ch1+ch2)**
  - End systems, packet switches, and other pieces of the Internet run **protocols** that control the sending and receiving of information within the Internet.
  - The **Transmission Control Protocol (TCP)** and the **Internet Protocol (IP)** are two of the most important protocols in the Internet. The IP protocol specifies the format of the packets that are sent and received among routers and end systems.
  - The Internet's principal protocols are collectively known as **TCP/IP**
  - End systems are connected together by a network of communication links and packet switches
  - different types of physical media, including coaxial cable, copper wire, optical fiber, and radio spectrum
  - The resulting packages of information, known as **packets**

- The two most prominent types in today’s Internet are **routers** and **link-layer switches**. Both types of switches forward packets toward their ultimate destinations
- **Link-layer switches** are typically used in access networks, while routers are typically used in the network core.
- The sequence of communication links and packet switches traversed by a packet from the sending end system to the receiving end system is known as a **route or path** through the network
- End systems access the Internet through **Internet Service Providers (ISPs)**, including residential ISPs
- Network protocols are **a set of rules outlining how connected devices communicate across a network to exchange information easily and safely**. Protocols serve as a common language for devices to enable communication irrespective of differences in software, hardware, or internal processes.
- **Packet switching** is **the transfer of small pieces of data across various networks**. These data chunks or “packets” allow for faster, more efficient data transfer.
    - Most packet switches use **store-and-forward transmission**. Store and forward is **a telecommunications technique in which information is sent to an intermediate station where it is kept and sent at a later time to the final destination or to another intermediate station**.
- The maximum queuing delay is proportional to buffer size. The longer the line of packets waiting to be transmitted, the longer the average waiting time is. **The router queue of packets waiting to be sent also introduces a potential cause of packet loss**.
- The routing table contains the path routing information, while the forwarding table contains the port information.

The main difference between the routing and forwarding tables is that **routing is used for routing the traffic, while forwarding is used for forwarding the data to the appropriate destination port**.

- **Circuit switching** is **a type of network configuration in which a physical path is obtained and dedicated to a single connection between two endpoints in the network for the duration of a dedicated connection**. Ordinary voice phone service uses circuit switching. This reserved circuit is used for the duration of a call.
- **Multiplexing in Circuit-Switched Networks**
  - Frequency Division Multiplexing or FDM is **used when multiple data signals are combined for simultaneous transmission via a shared communication medium** .It is a technique by which the total bandwidth is divided into a series of non-overlapping frequency sub-bands, where each sub-band carry different signal
- **Packet Switching V/S Circuit Switching**
- An **Internet exchange point (IXP)** is a physical location through which Internet infrastructure companies such as Internet Service Providers (ISPs) and CDNs connect with each other.
- Nodal Delay

🧮 Transmission Delay= Packet size/Rate of transfer

$$d_t = L/R,$$

- A circuit-switched network relies on a physical connection between two nodes, which requires the link to be set up before the nodes can communicate. In contrast, a packet-switched network is a digital network that manages data transfer in the form of small and optimized packets, an improvement from older network types.

- Nodal Delay

🧮 If we let processing, queuing, transmission, and propagation delays

$$d_n = d_p + d_q + d_t + d_{p \cdot p}$$

- End-to-End Delay
  - is **measured from the moment the packet leaves the source application to the moment the same packet arrives at the destination application**.

🧮 If we let processing, transmission, and propagation delays, there are N - 1 routers

$$d = n(d_p + d_t + d_{p \cdot p})$$

- A traceroute **works by sending Internet Control Message Protocol (ICMP) packets**, and every router involved in transferring the data gets these packets. The ICMP packets provide information about whether the routers used in the transmission are able to effectively transfer the data.
- If the file consists of F bits and the transfer takes T seconds for Host B to receive all F bits, then the **average throughput** of the file transfer is F/T bits/sec
- a **bottleneck link** for a given data flow is a link that is fully utilized (is *saturated*) and of all the flows sharing this link, the given data flow achieves maximum data rate network-wide
- **Five-layer Internet protocol stack**

Application Layer

The application layer is where network applications and their application-layer protocols reside. The Internet’s application layer includes many protocols, such as the
HTTP protocol (which provides for Web document request and transfer), SMTP
(which provides for the transfer of e-mail messages), and FTP (which provides for
the transfer of files between two end systems

Transport Layer

The Internet’s transport layer transports application-layer messages between application endpoints. In the Internet, there are two transport protocols, TCP and UDP, either of which can transport application-layer messages. TCP provides a connection-oriented service to its applications. This service includes guaranteed delivery of application-layer messages to the destination and flow control (that is, sender/receiver speed matching).

Network Layer

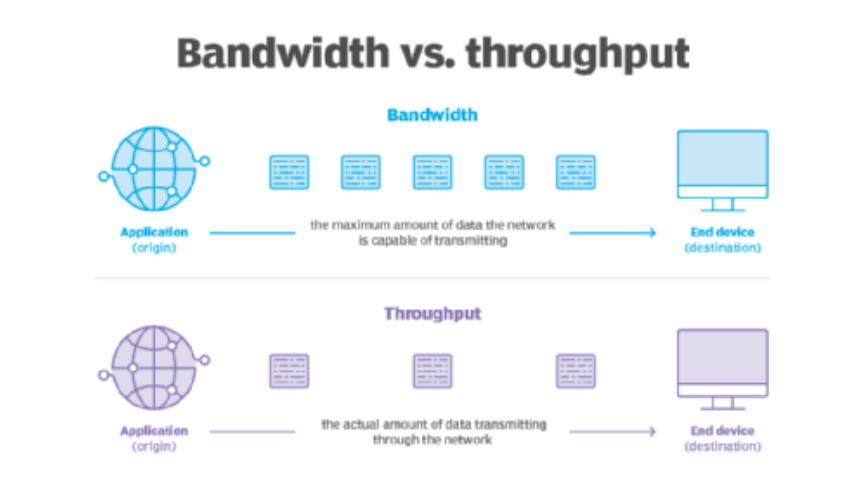
The Internet’s network layer is responsible for moving network-layer packets known as datagrams from one host to another. The Internet transport-layer protocol (TCP or UDP) in a source host passes a transport-layer segment and a destination address to the network layer, just as you would give the postal service a letter with a destination address. The network layer then provides the service of delivering the segment to the transport layer in the destination host. The Internet’s network layer includes the celebrated IP protocol

Link Layer

The Internet’s network layer routes a datagram through a series of routers between the source and destination. To move a packet from one node (host or router) to the next node in the route, the network layer relies on the services of the link layer. In particular, at each node, the network layer passes the datagram down to the link layer, which delivers the datagram to the next node along the route. At this next node, the link layer passes the datagram up to the network layer

Physical Layer

- In the context of a communication session between a pair of processes, the process that initiates the communication (that is, initially contacts the other process at the beginning of the session) is labeled as the client. The process that waits to be contacted to begin the session is the server

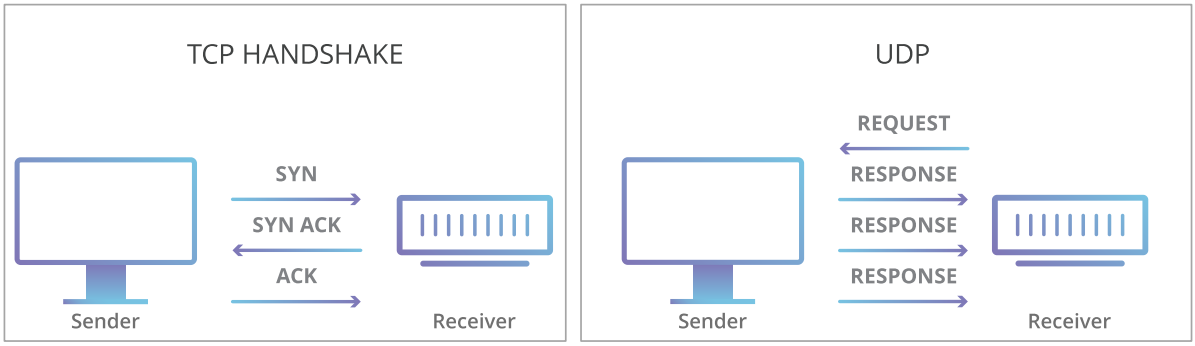


- TCP is a stream-oriented protocol. It enables the sending process to deliver data as a stream of bytes and the receiving process to acquire data as a stream of bytes.
  - The sending and receiving processes cannot produce and receive data at the same speed. Hence, TCP needs a buffer for storage.
  - Buffering is used to handle the difference between the speed of data transmission and data consumption. But only buffering is not enough.
  - one more step before sending the data on the Internet Protocol (IP) layer as a TCP service provider. It needs to send data in the form of packets and not as a stream of bytes.
  - At the transport layer, TCP groups several bytes into a packet and this is called a segment. A header is added to each segment to exercise control.
  - The segment is encapsulated in an IP diagram and then transmitted. The entire operation is transparent to the receiving process. The segment may be deceived out of order, lost or corrupted when it receives the receiving end.

- TCP offers a full-duplex service where the data can flow in both directions simultaneously. Each TCP will then have a sending buffer and receiving buffer. The TCP segments are sent in both directions.
- TCP is a reliable transport protocol. It uses an acknowledgment mechanism for checking the safe and sound arrival of data.
- **TCP connections that are kept open after transactions complete are called persistent connections**.
- Non persistent connections are closed after each transaction. Persistent connections stay open across transactions, until either the client or the server decides to close them.

- UDP Services
  - The User Datagram Protocol, or UDP, is a communication protocol used across the Internet for especially time-sensitive transmissions such as video playback or DNS lookups. It speeds up communications by not formally establishing a connection before data is transferred. This allows data to be transferred very quickly, but it can also cause packets to become lost in transit
  - UDP is a standardized method for transferring data between two computers in a network. Compared to other protocols, UDP accomplishes this process in a simple fashion: it sends packets (units of data transmission) directly to a target computer, without establishing a connection first, indicating the order of said packets, or checking whether they arrived as intended.
  - UDP is commonly used in time-sensitive communications where occasionally dropping packets is better than waiting. Voice and video traffic are often sent using this protocol because they are both time-sensitive and designed to handle some level of loss.

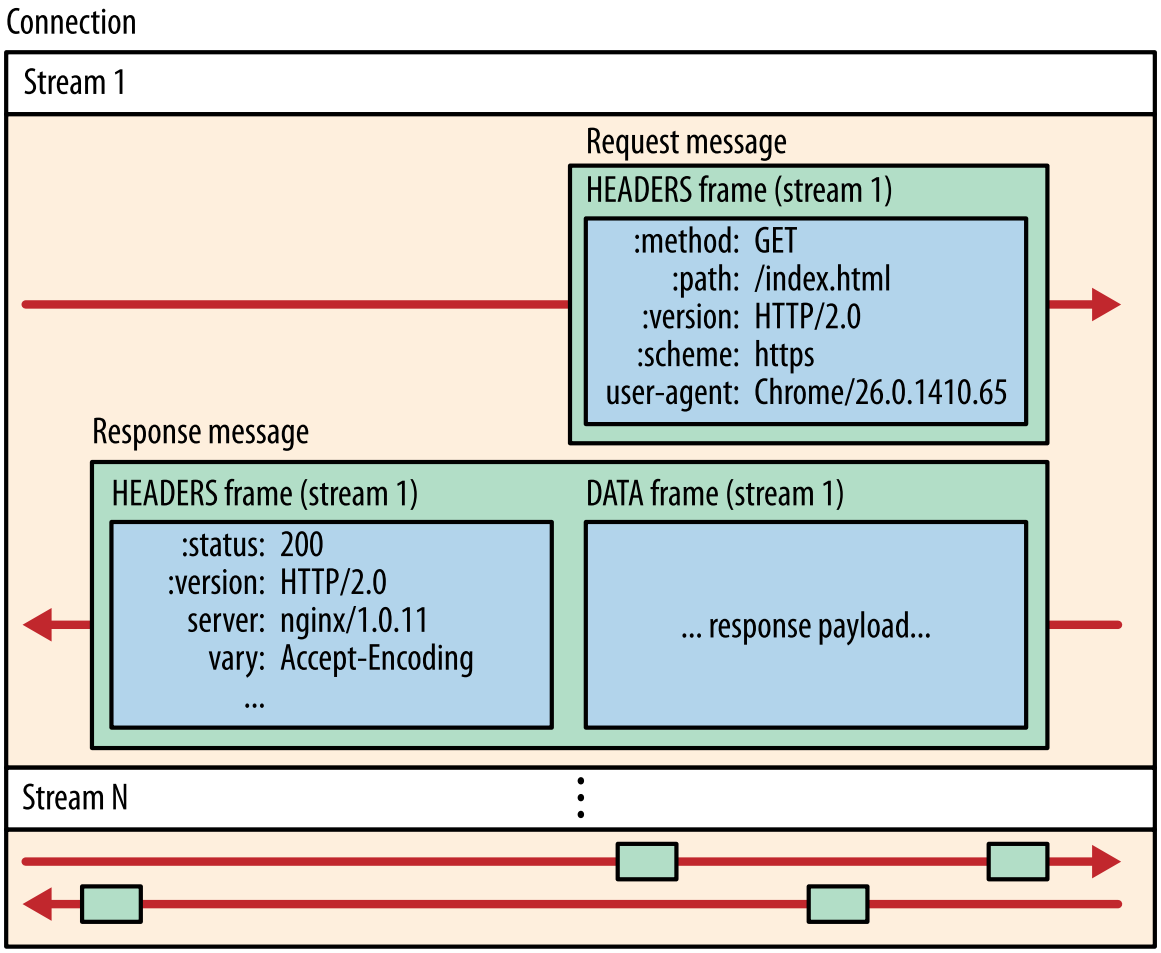
TCP vs UDP Communication



- A Web cache—also called a proxy server—is a network entity that satisfies HTTP requests on the behalf of an origin Web server. The Web cache has its own disk storage and keeps copies of recently requested objects in this storage
- **HTTP/1.1** loads resources one after the other, so if one resource cannot be loaded, it blocks all the other resources behind it. In contrast, **HTTP/2** is able to use a single TCP connection to send multiple streams of data at once so that no one resource blocks any other resource.
- sending all the objects in a Web page over a single TCP connection has a Head of Line (HOL) blocking problem
- HTTP/2 terminology:
  - *Stream*: A bidirectional flow of bytes within an established connection, which may carry one or more messages.
  - *Message*: A complete sequence of frames that map to a logical request or response message.
  - *Frame*: The smallest unit of communication in HTTP/2, each containing a frame
    header, which at a minimum identifies the stream to which the frame
    belongs.

  The relation of these terms can be summarized as follows:

- All communication is performed over a single TCP connection that can carry any number of bidirectional streams.
- Each stream has a unique identifier and optional priority information that is used to carry bidirectional messages.
- Each message is a logical HTTP message, such as a request, or response, which consists of one or more frames.
- The frame is the smallest unit of communication that carries a specific type of data—e.g., HTTP headers, message payload, and so on. Frames from different streams may be interleaved and then reassembled via the embedded stream identifier in the header of each frame.



- The Simple Mail Transfer Protocol is an Internet standard communication protocol for electronic mail transmission. Mail servers and other message transfer agents use **SMTP** to send and receive mail messages.
- The **Domain Name System (DNS)** is the phonebook of the Internet. Humans access information online through domain names, like nytimes.com or espn.com. Web browsers interact through Internet Protocol (IP) addresses. DNS translates domain names to IP addresses so browsers can load Internet resources.
    - The process of DNS resolution involves converting a hostname (such as www.example.com) into a computer-friendly IP address (such as 192.168.1.1). An IP address

is given to each device on the Internet, and that address is necessary to find the appropriate Internet device - like a street address is used to find a particular home. When a user wants to load a webpage, a translation must occur between what a user types into their web browser (example.com) and the machine-friendly address necessary to locate the example.com webpage.

- The DNS servers that together implement the DNS distributed database store **resource records (RRs)**, including RRs that provide hostname-to-IP address mappings. Each DNS reply message carries one or more resource records.
- A resource record is a four-tuple that contains the following fields:

$$(Name, Value, Type, TTL)$$

- TTL is the time to live of the resource record; it determines when a resource should
  be removed from a cache. In the example records given below, we ignore the TTL
  field. The meaning of Name and Value depend on Type:
  - If **Type=A**, then Name is a hostname and Value is the IP address for the host name. Thus, a Type A record provides the standard hostname-to-IP address mapping. As an example, (relay1.bar.foo.com, 145.37.93.126, A) is a Type A record.
  - If **Type=NS**, then Name is a domain (such as foo.com) and Value is the host name of an authoritative DNS server that knows how to obtain the IP addresses for hosts in the domain. This record is used to route DNS queries further along in the query chain. As an example, (, dns.foo.com, NS) is a Type NS record.
  - If **Type=CNAME**, then Value is a canonical hostname for the alias hostname Name. This record can provide

querying hosts the canonical name for a host name. As an example, (foo.com, relay1.bar.foo.com, CNAME) is a CNAME record.

- If **Type=MX**, then Value is the canonical name of a mail server that has an alias hostname Name. As an example, (foo.com, mail.bar.foo.com, MX) is an MX record. MX records allow the hostnames of mail servers to have simple aliases. Note that by using the MX record, a company can have the same aliased name for its mail server and for one of its other servers (such as its Web server). To obtain the canonical name for the mail server, a DNS client would query for an MX record; to obtain the canonical name for the other server, the DNS client would query for the CNAME record.

▼ Ch 3:transport Layer

▼ intro

- there are two internet transport layer protocols
  - UDP: connectionless, best effort service
  - tCP: reliable, flow and congestion controlled connection oriented transport
- provides logical communication between application process running on different hosts
- transport protocols actions in end systems
  - sender: breaks application messages into segments, passes to network layer
  - reciever: reassembles segments into messages, passes to application laier
- Sender:
  - is passed an application layer message
  - determines segment header fields val

    - creates segment
    - passes segment to IP
  - Receiver
    - receives segment from IP
    - checks header val
    - extracts applications layer message
    - demultiplexes message up to application via socket

▼ multiplexing and demultiplexing

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

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

▼ **How it works (UDP)**

- host receives IP datagrams
  - each datagram has source IP address, destination IP address
  - each datagram carries one transport-layer segment
  - each segment has source, destination port number
- host uses IP addresses & port numbers to direct segment to appropriate socket
- when creating socket, must specific host-local port #

```
DatagramSocket dum=new DatagramSocket(12345);
```

- when creating datagram to send to udp socket, must specify
  - destination IP Address
  - destination port #

- when receiving host receives UDP segment
  - checks destination port # in segment
  - direct UDP Segment to socket with that port #

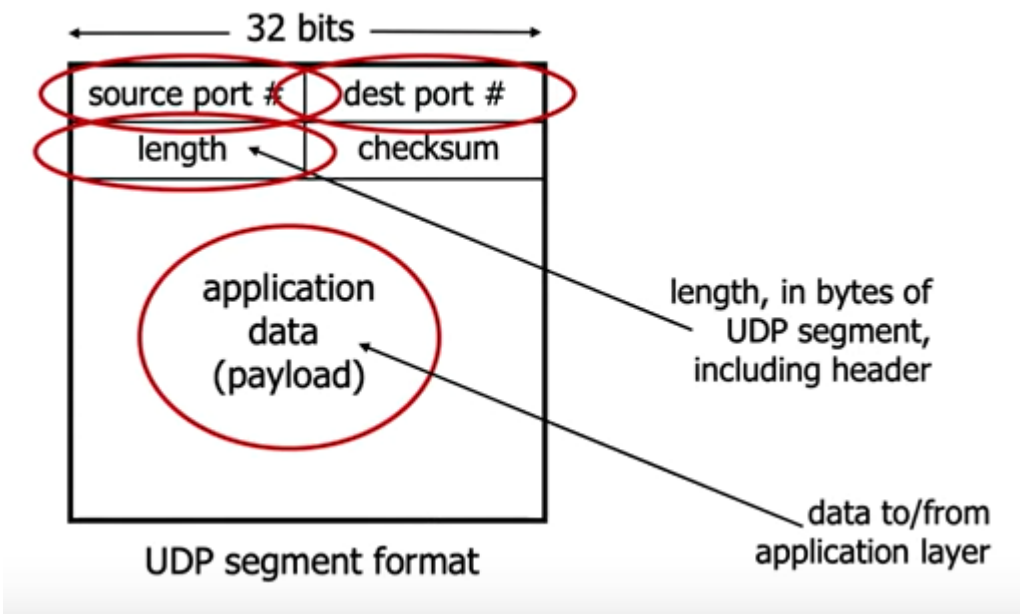
▼ **How it Works (tcp)**

- tcp socket identified bi 4-tuple
  - source ip address
  - source port number
  - destination ip address
  - destination port number
- on receiving end, receiver uses all these to direct segment to appropriate socket

▼ connectionless transport: UDP

- “best effort” service, UDP segment mai be:
  - lost
  - out of order delivery
- connectionless
  - no handshaking between UDP sender and receiver
  - each UPD segment handled independently
- why UDP?
  - no connection establishment (which can add rtt delay)
  - simple: no connection state at sender, receiver
  - small header size
  - no congestion control
    - udp can function in congestion
- UDP is used in

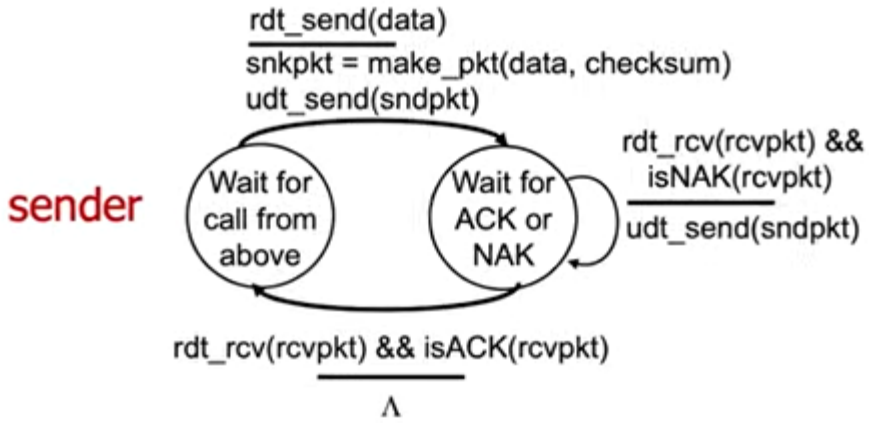
    - streaming multimedia
    - DNS
    - SNMP
    - http/3
- if relible transfer needed over UDP
    - add needed reliability at application layer
    - acc congestion control at application layer
- UDP sender actions
    - is passed an application layer message
    - determines UDP segment header fields values
    - creates UDP segment
    - passes segment to IP
- UDP receiver actions
    - receives segment from IP
    - checks UDP checksum header value
    - extracts application-layer message
    - demultiplexes message up to application via socket

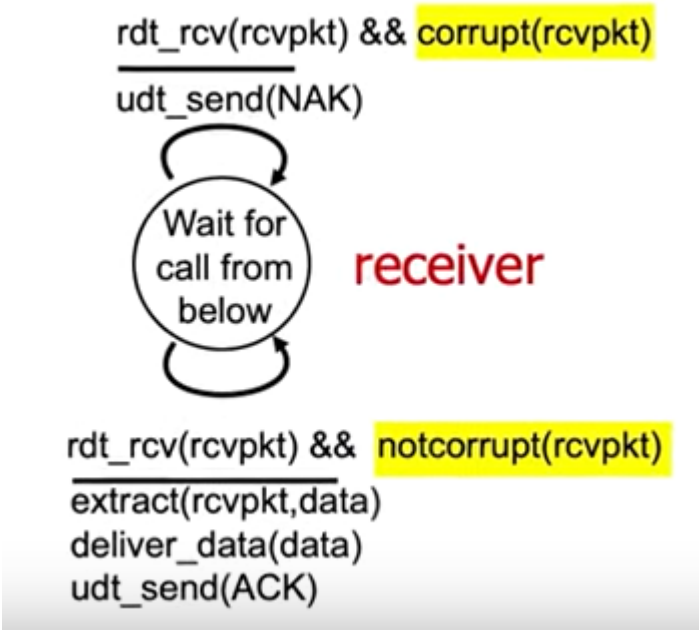


▼ Principles of Reliable data transfer

- Complexity of reliable data transfer protocol depends upon characteristics of channel used
- reliable data transfer == rdt
- rdt 1.0
  - perfectli reliable (delusional)

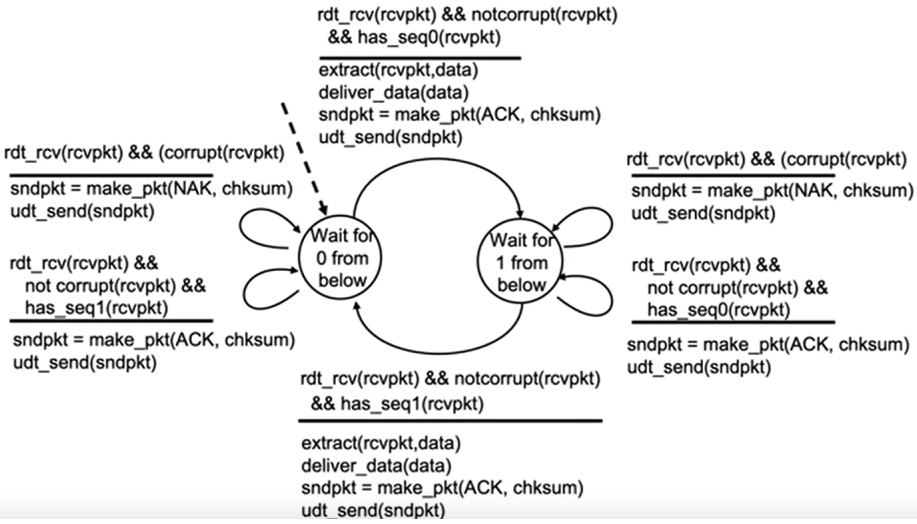
- rdt 2.0 - stop and wait
  - sends one packet and then waits for ack
  - cant tell if ack/nak is corrupted





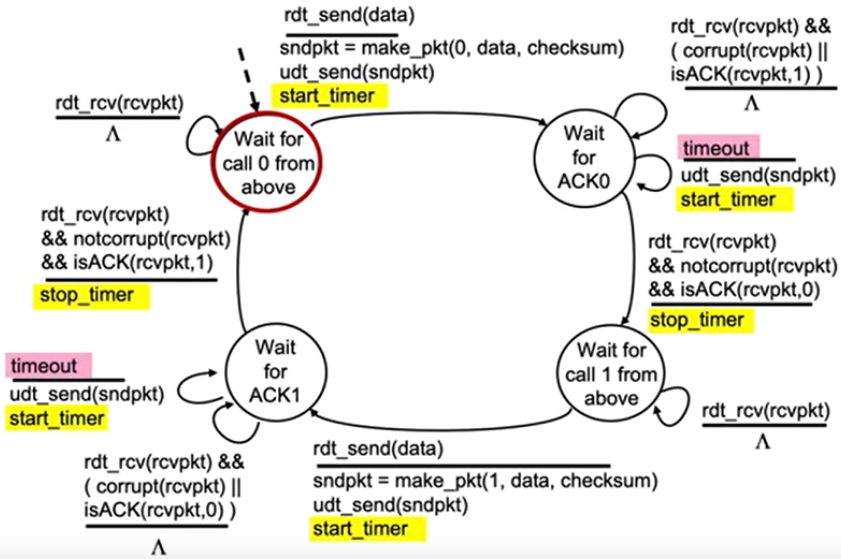
- `rdt 2.1 - handles corrupted acks`

rdt2.1: receiver, handling garbled ACK/NAKs

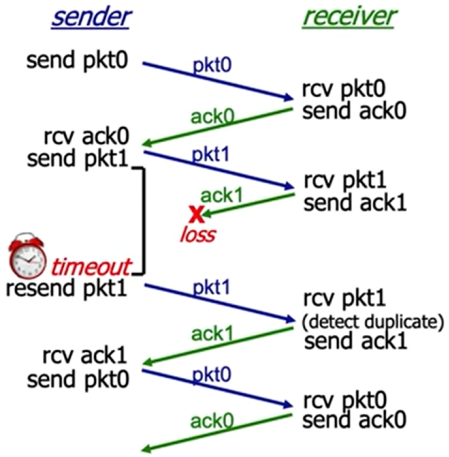


- rdt 3.0 - introduces timer

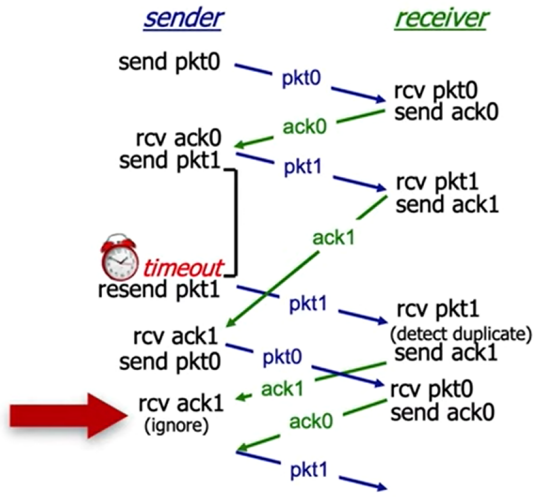
rdt3.0 sender



rdt3.0 in action



(c) ACK loss



(d) premature timeout/ delayed ACK

- `Go Back N`

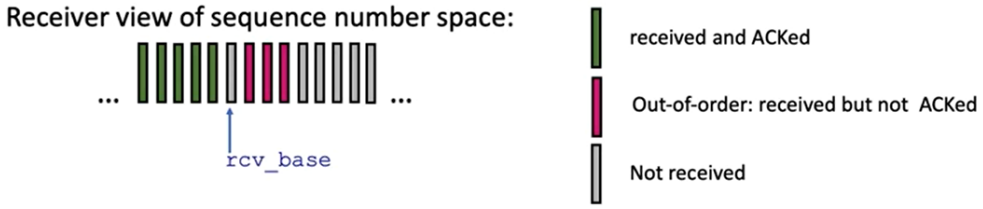
Go-Back-N: sender

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

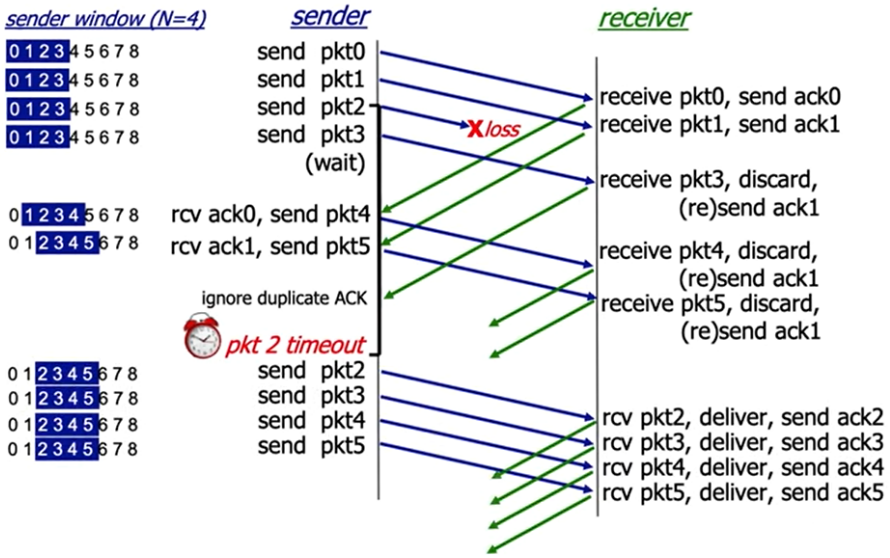
- *cumulative ACK:* ACK(*n*): ACKs all packets up to, including seq # *n*
  - on receiving ACK(*n*): move window forward to begin at *n+1*
- timer for oldest in-flight packet
- *timeout(n):* retransmit packet n and all higher seq # packets in window

Go-Back-N: receiver

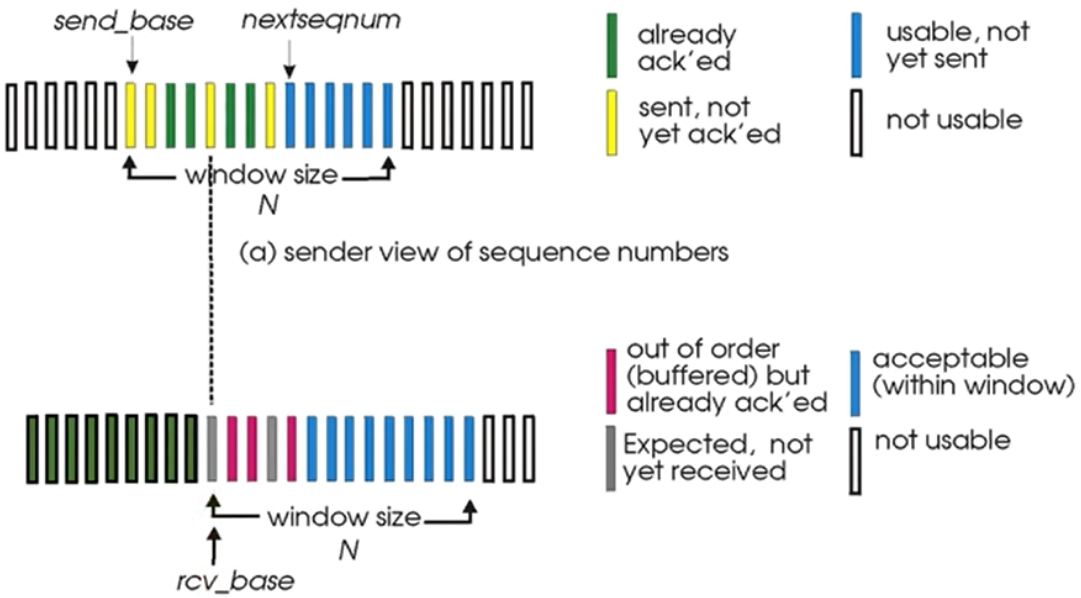
- ACK-only: always send ACK for correctly-received packet so far, with highest *in-order* seq #
  - may generate duplicate ACKs
  - need only remember `rcv_base`
- on receipt of out-of-order packet:
  - can discard (don't buffer) or buffer: an implementation decision
  - re-ACK pkt with highest in-order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



Go-Back-N in action



- Selective repeat
  - sends out individual unacked packets



sender

data from above:

- if next available seq # in window, send packet

timeout(*n*):

- resend packet *n*, restart timer

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

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

receiver

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

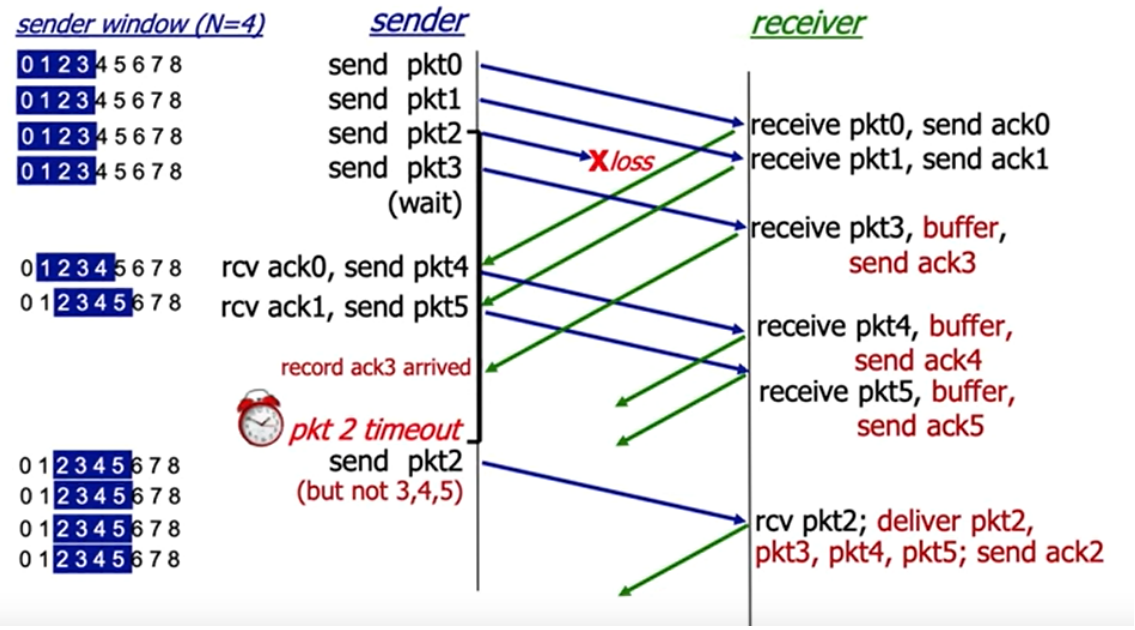
- send ACK(*n*)
- out-of-order: buffer
- in-order: deliver (also deliver buffered, in-order packets), advance window to next not-yet-received packet

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

- ACK(*n*)

otherwise:

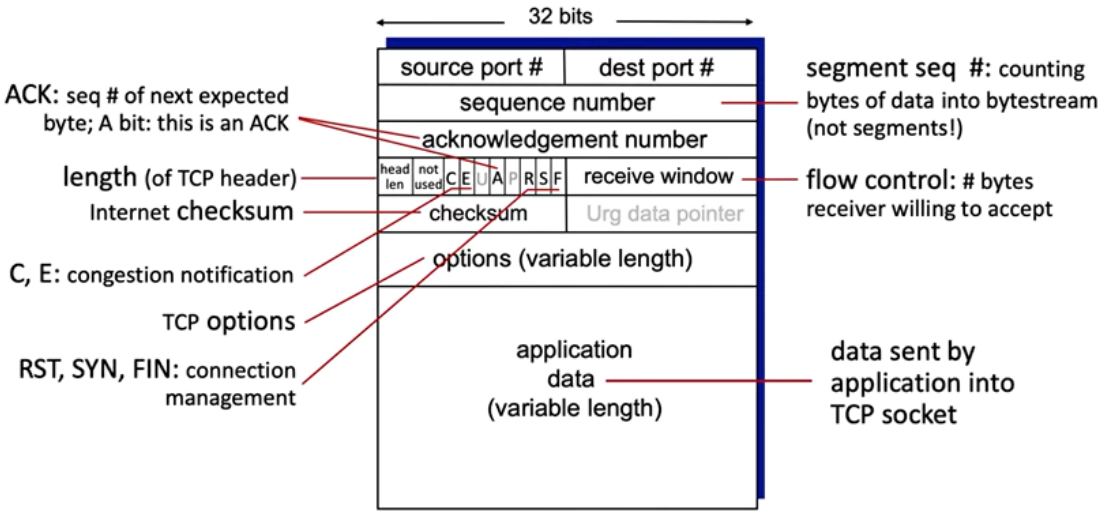
- ignore



▼ Connection orientated transport ~ tcp

- point to point (one sender, one receiver)
- reliable, in order transmission
- full duplex
  - bi directional data flow
- Maximum segment size
- cumulative ACKs
- Pipelining
  - tcp congestion and flow control set window size
- connection oriented
  - handshaking initializes sender and receiver state before data exchange
- flow control
  - sender will not overwhelm reciever

TCP segment structure

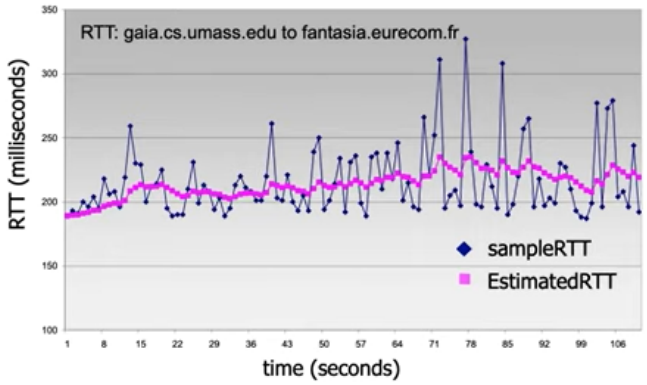


- `timeout`

TCP round trip time, timeout

$$\text{EstimatedRTT} = (1-\alpha)*\text{EstimatedRTT} + \alpha*\text{SampleRTT}$$

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



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, β = 0.25)

- sender

TCP Sender (simplified)

event: data received from application

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

event: timeout

- retransmit segment that caused timeout
- restart timer

event: ACK received

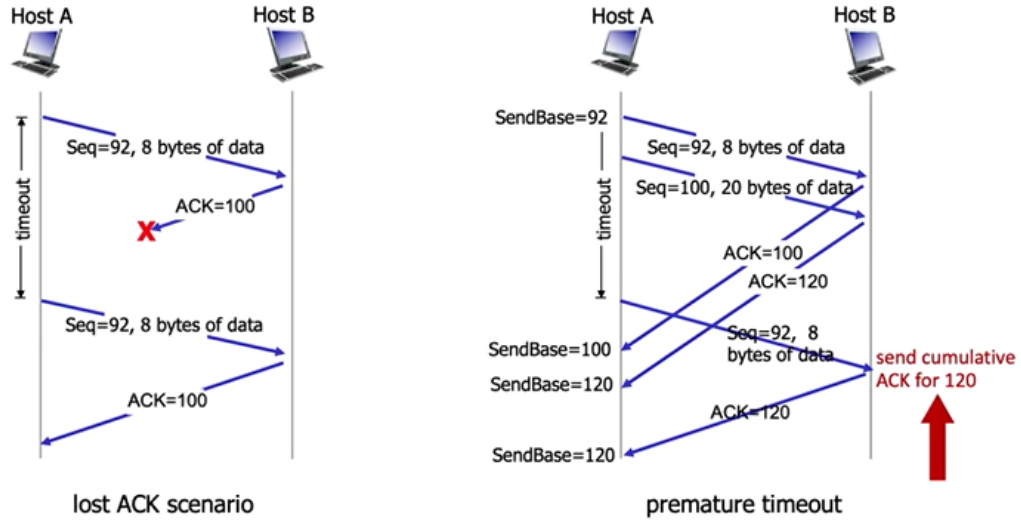
- if ACK acknowledges previously unACKed segments
  - update what is known to be ACKed
  - start timer if there are still unACKed segments

- receiver

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

- re

TCP: retransmission scenarios



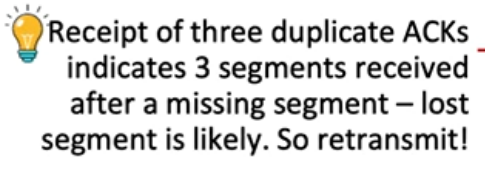
- fas
-

TCP fast retransmit

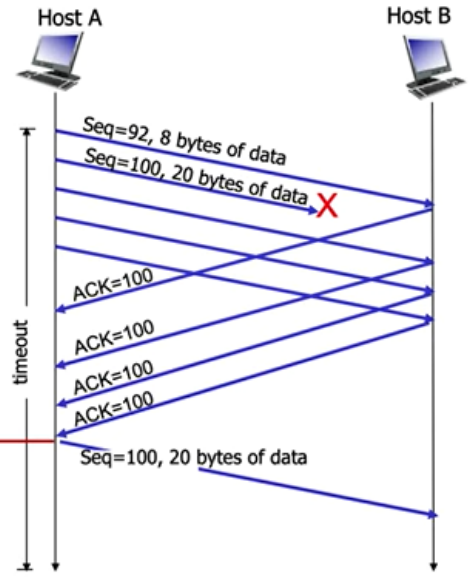
TCP fast retransmit

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

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



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

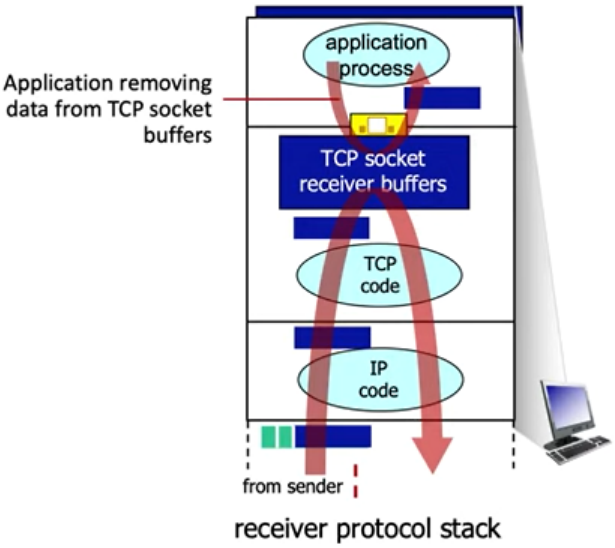


TCP flow control

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

flow control

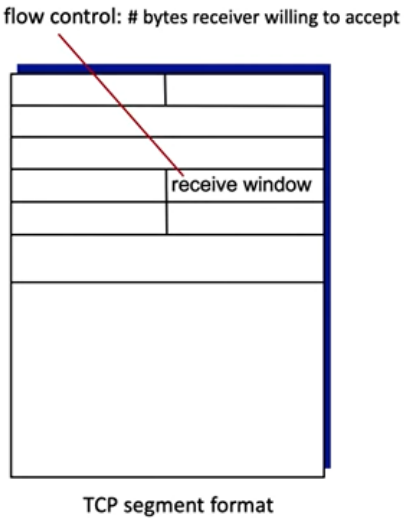
receiver controls sender, so sender won't overflow receiver's buffer by transmitting too much, too fast



receiver protocol stack

TCP flow control

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

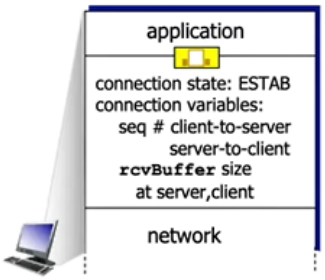


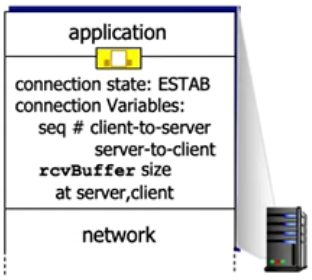
TCP segment format

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)

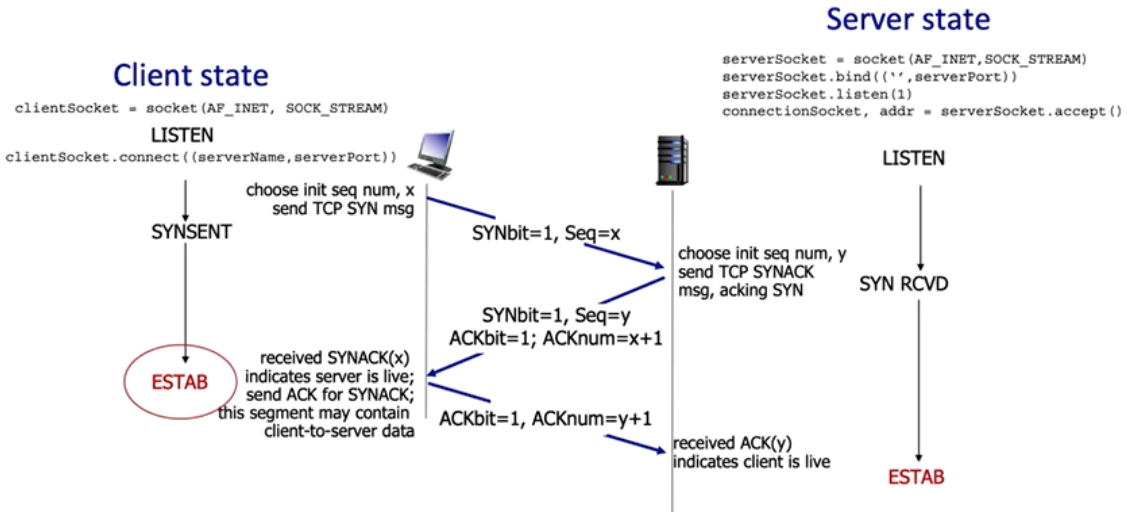





```
Socket clientSocket =
  newSocket("hostname","port number");
```

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

TCP 3-way handshake



- Principles of congestion control
  - **congestion:**
    - too many sources sending too much data too fast for network to handle
    - causes
      - longer delays
      - packet loss
  -

- **Ch 4:network layer**

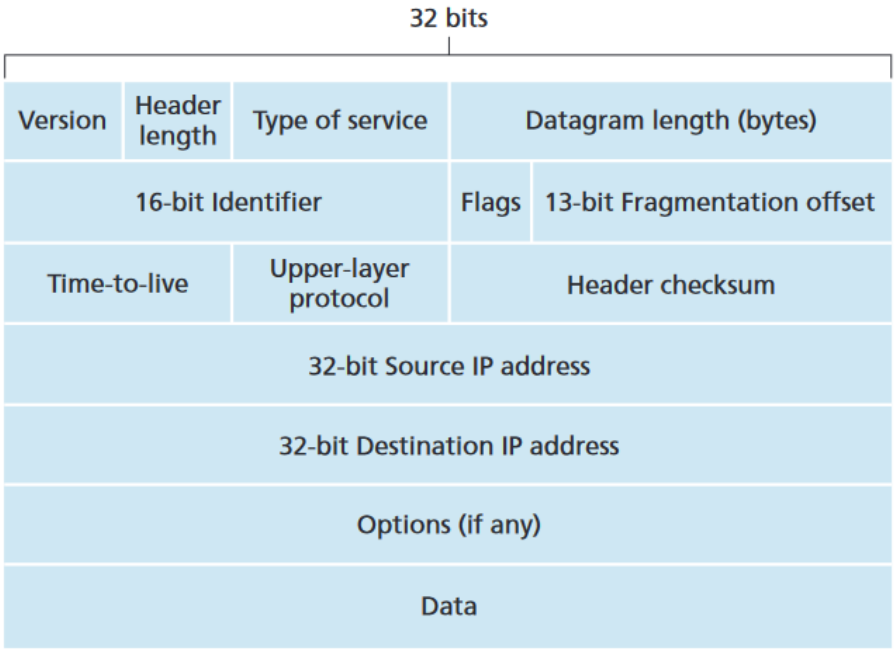


Figure 4.17 ♦ IPv4 datagram format

▼ **numericals(3,4)**

- **Subnet Mask**: 255.255.254.0
- **Network Address**: 132.100.24.0
- **Broadcast Address**: 132.100.25.255
- **First Host Address**: 132.100.24.1
- **Last Host Address**: 132.100.25.254

To calculate the network address, we need to set all the host bits to 0. Therefore, the network address is 132.100.24.0 .

To calculate the broadcast address, we need to set all the host bits to 1. Therefore, the broadcast address is 132.100.25.255 .

To calculate the first host address, we need to set the rightmost bit in the host portion to 1. Therefore, the first host address is 132.100.24.1

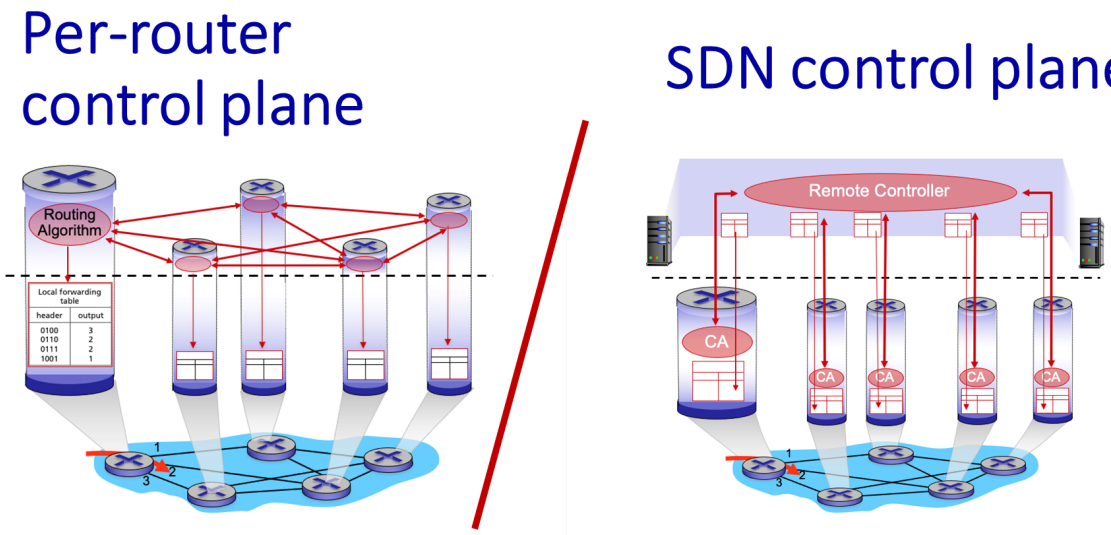
To calculate the last host address, we need to set the rightmost bit in the host portion to 0. Therefore, the last

host address is 132.100.25.254

- verifing recieved data is valid or not
  - divide data segment in 16 bits segments
  - add together
  - wrap around if carry
  - take complement at end
  - if equal to checksum, true

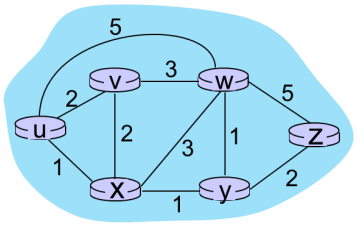
▼ Ch 5: Network Layer

- basic 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*
- **Per-router control plane**
  - Individual routing algorithm components *in each and every router* interact in the control plane
- **Software-Defined Networking (SDN) control plane**
  - Remote controller computes, installs forwarding tables in routers



- **Routing protocols**
  - **Routing protocol goal:** determine “good” paths (equivalently, routes), from sending hosts to receiving host, through network of routers
    - path: sequence of routers packets traverse from given initial source host to final destination host
    - “good”: least “cost”, “fastest”, “least congested”
    - routing: a “top-10” networking challenge!

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)* }

Routing algorithm classification

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

How fast do routes change?

static: routes change slowly over time

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

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

global or decentralized information?

Dijkstra's link-state 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

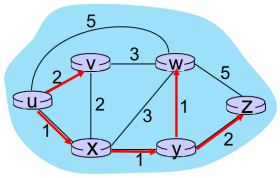
notation

- $c_{x,y}$: direct link cost from node *x* to *y*; = ∞ 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


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

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'* :
$D(b) = \min (D(b), D(a) + c_{a,b})$

D(w),p(w)
5,u
4,x
3,y
3,y

algorithm complexity: *n* nodes

- each of *n* iteration: need to check all nodes, *w*, not in *N*
- $n(n+1)/2$ comparisons: $O(n^2)$ complexity
- more efficient implementations 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: $O(n^2)$

algorithm complexity: *n* nodes

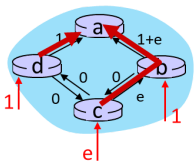
- each of *n* iteration: need to check all nodes, *w*, not in *N*
- $n(n+1)/2$ comparisons: $O(n^2)$ complexity
- more efficient implementations possible: $O(n\log n)$

message complexity:

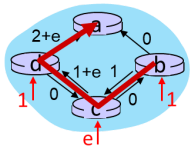
- each router must *broadcast* its link state information to other *n* routers
- efficient (and interesting!) broadcast algorithms: $O(n)$ link crossings to disseminate a broadcast message from one source
- each router's message crosses $O(n)$ links: overall message complexity: $O(n^2)$

Dijkstra's algorithm: oscillations possible

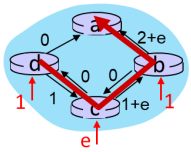
- when link costs depend on traffic volume, route oscillations possible
- sample scenario:
  - routing to destination a, traffic entering at d, c, e with rates 1, e (<1), 1
  - link costs are directional, and volume-dependent



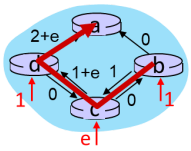
initially



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



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



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

Distance vector algorithm

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

Bellman-Ford equation

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

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

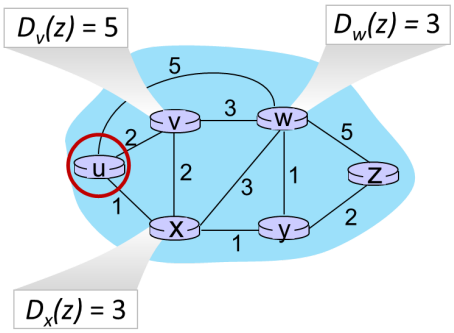
v's estimated least-cost-path cost to y

min taken over all neighbors v of x

direct cost of link from x to v

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)

Comparison of LS and DV algorithms

message complexity

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

DV: exchange between neighbors; convergence time varies

speed of convergence

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

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

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

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

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

Comparison of LS and DV algorithms

message complexity

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

DV: exchange between neighbors; convergence time varies

speed of convergence

LS: $O(n^2)$ algorithm, $O(n^2)$ messages

- may have oscillations

DV: convergence time varies

- may have routing loops
- count-to-infinity problem

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

LS:

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

DV:

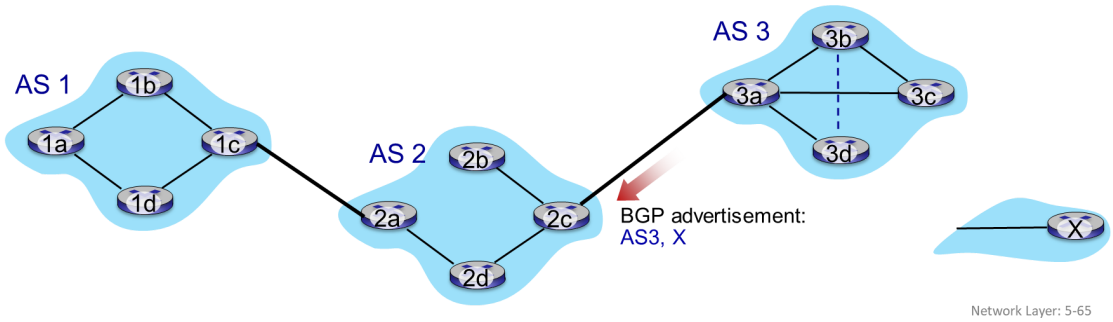
- DV router can advertise incorrect *path* cost ("I have a *really* low-cost path to everywhere"): *black-holing*
- each router's DV is used by others: error propagate thru network

- **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
- **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:
    - obtain destination network reachability info from neighboring ASes (eBGP)

- determine routes to other networks based on reachability information and *policy*
- propagate reachability information to all AS-internal routers (iBGP)
- advertise (to neighboring networks) destination reachability info

BGP basics

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

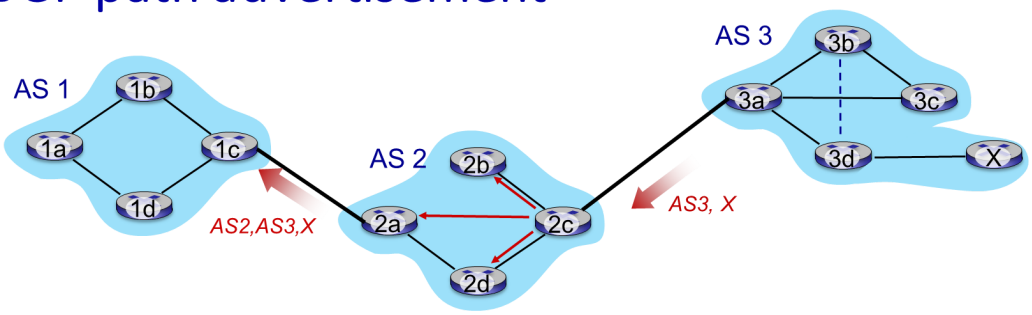


- BGP messages exchanged between peers over TCP connection
- BGP messages [RFC 4371]:
  - OPEN: opens TCP connection to remote BGP peer and authenticates sending BGP peer
  - **UPDATE:** advertises new path (or withdraws old)
  - KEEPALIVE: keeps connection alive in absence of UPDATES; also ACKs OPEN request
  - NOTIFICATION: reports errors in previous msg; also used to close connection

Path attributes and BGP routes

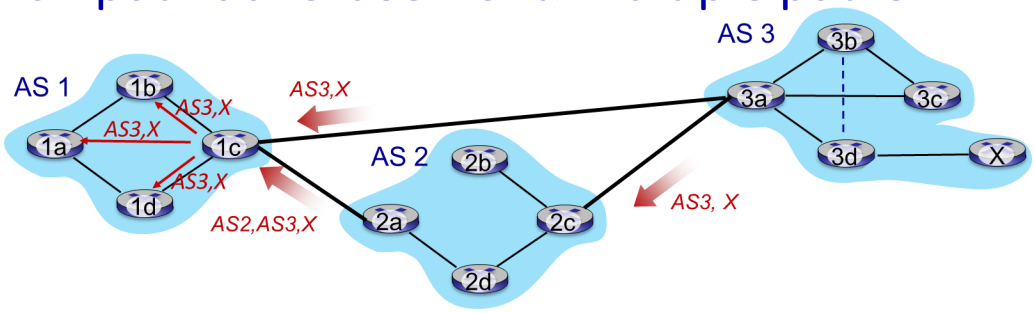
- BGP advertised route: prefix + attributes
  - prefix: destination being advertised
  - two important attributes:
    - AS-PATH: list of ASes through which prefix advertisement has passed
    - NEXT-HOP: indicates specific internal-AS router to next-hop AS
- policy-based routing:
  - gateway receiving route advertisement uses *import policy* to accept/decline path (e.g., never route through AS Y).
  - AS policy also determines whether to *advertise* path to other other neighboring ASes

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

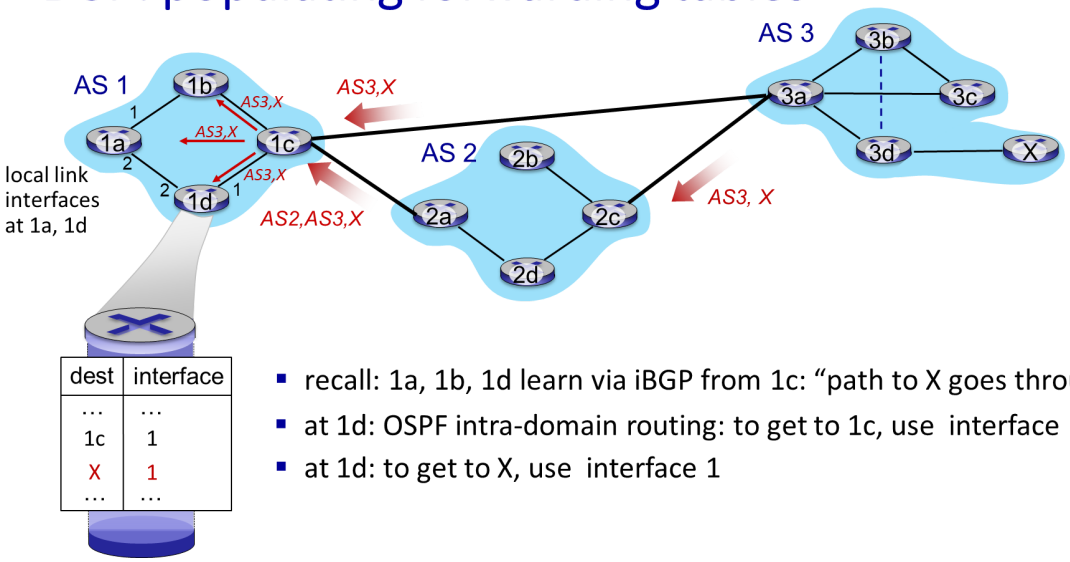
BGP path advertisement: multiple paths



gateway router may learn about multiple paths to destination:

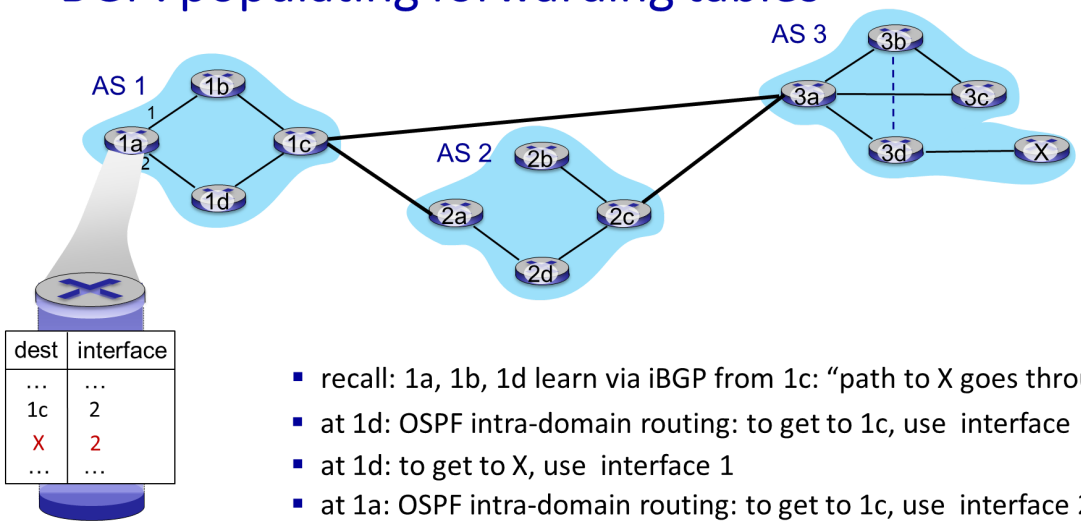
- AS1 gateway router 1c learns path *AS2,AS3,X* from 2a
- AS1 gateway router 1c learns path *AS3,X* from 3a
- based on *policy,* AS1 gateway router 1c chooses path *AS3,X* and advertises path within AS1 via iBGP

BGP: populating forwarding tables



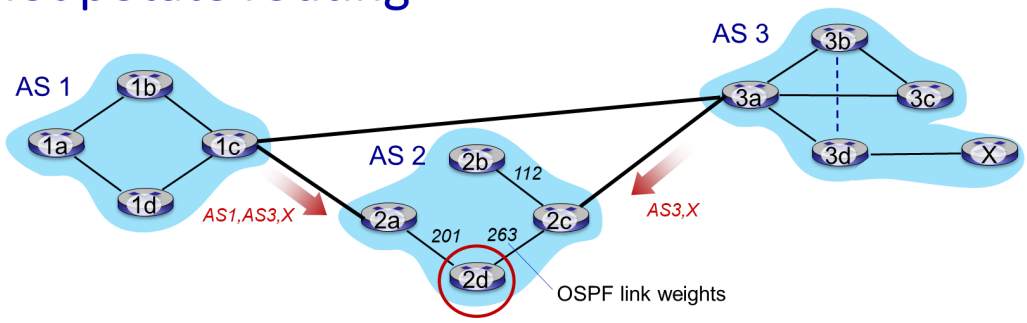
- recall: 1a, 1b, 1d learn via iBGP from 1c: "path to X goes through 1c"
- at 1d: OSPF intra-domain routing: to get to 1c, use interface 1
- at 1d: to get to X, use interface 1

BGP: populating forwarding tables



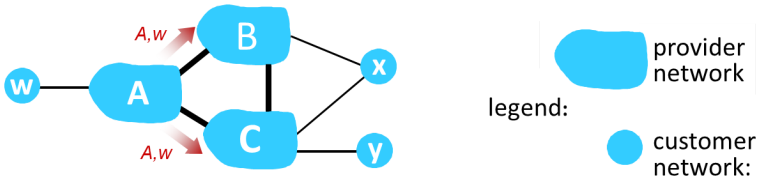
- recall: 1a, 1b, 1d learn via iBGP from 1c: "path to X goes through 1c"
- at 1d: OSPF intra-domain routing: to get to 1c, use interface 1
- at 1d: to get to X, use interface 1
- at 1a: OSPF intra-domain routing: to get to 1c, use interface 2
- at 1a: to get to X, use interface 2

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!

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
- 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

- 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 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

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

- **SDN**
  - Internet network layer: historically implemented via distributed, per-router control approach:
    - *monolithic* router contains switching hardware, runs proprietary implementation of Internet standard

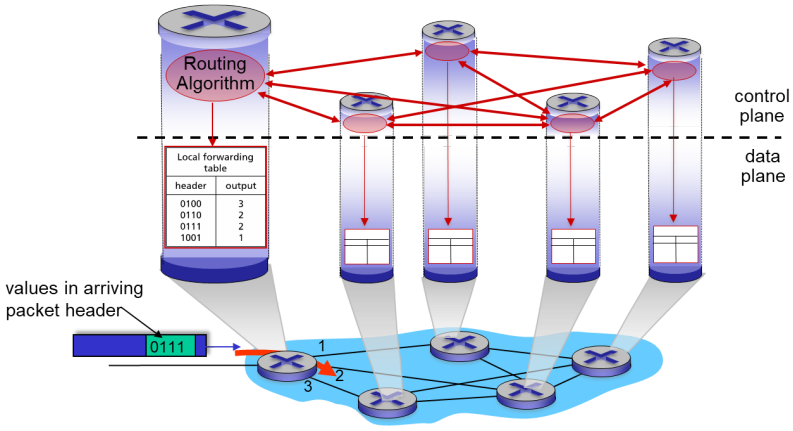
protocols (IP, RIP, IS-IS, OSPF, BGP) in proprietary router OS (e.g., Cisco IOS)

- different “middleboxes” for different network layer functions: firewalls, load balancers, NAT boxes, ..

- ~2005: renewed interest in rethinking network control plane

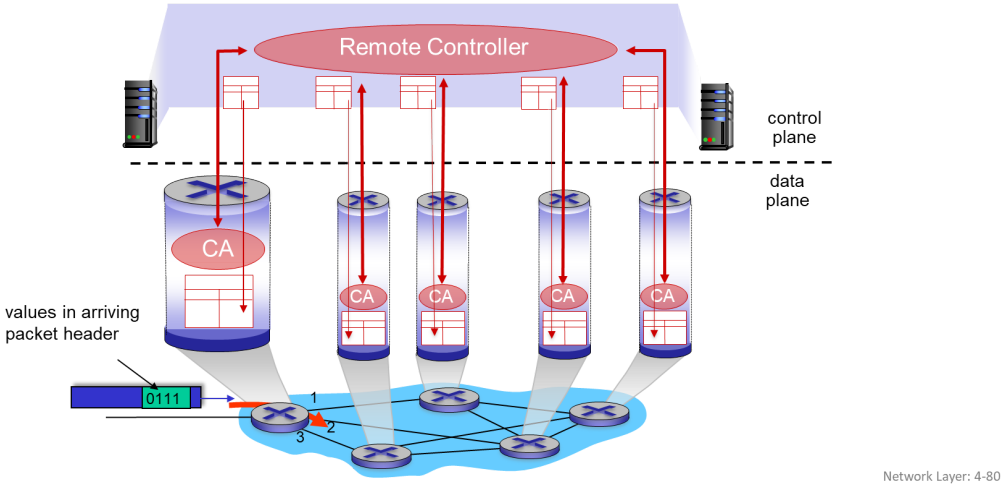
Per-router control plane

Individual routing algorithm components *in each and every router* interact in the control plane to computer forwarding tables



Software-Defined Networking (SDN) control plane

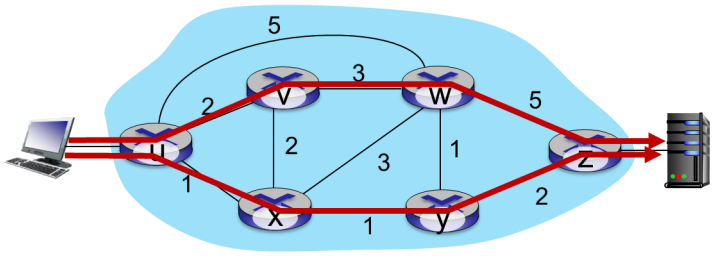
Remote controller computes, installs forwarding tables in routers



Why a *logically centralized* control plane?

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

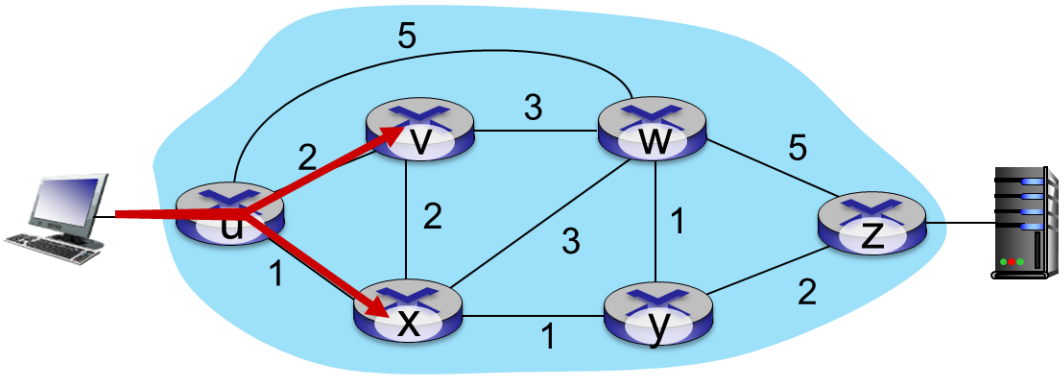
Traffic engineering: difficult with traditional routing



Q: what if network operator wants u-to-z traffic to flow along *uvwz*, rather than *uxyz*?

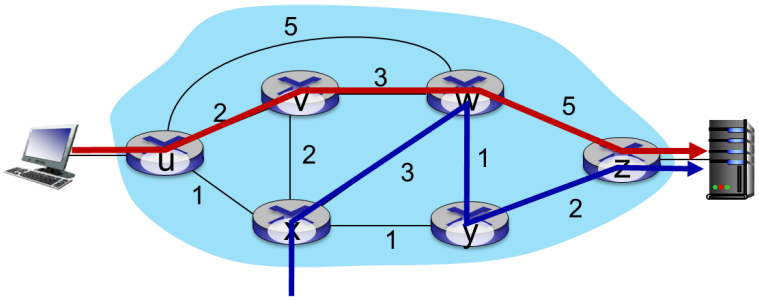
A: need to re-define link weights so traffic routing algorithm computes routes accordingly (or need a new routing algorithm)!

link weights are only control "knobs": not much control!



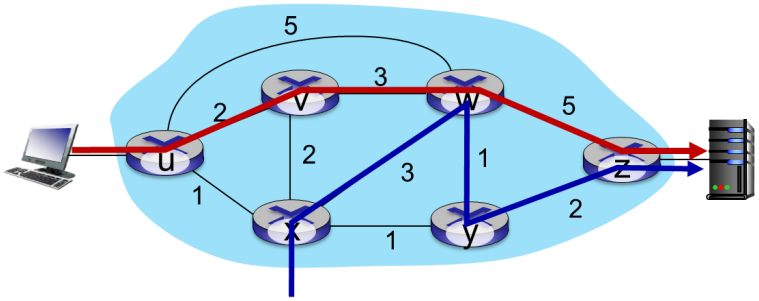
Q: what if network operator wants to split u-to-z traffic along uvwz *and* uxyz (load balancing)?

A: can't do it (or need a new routing algorithm)



Q: what if w wants to route blue and red traffic differently from w to z?

A: can't do it (with destination-based forwarding, and LS, DV routing)



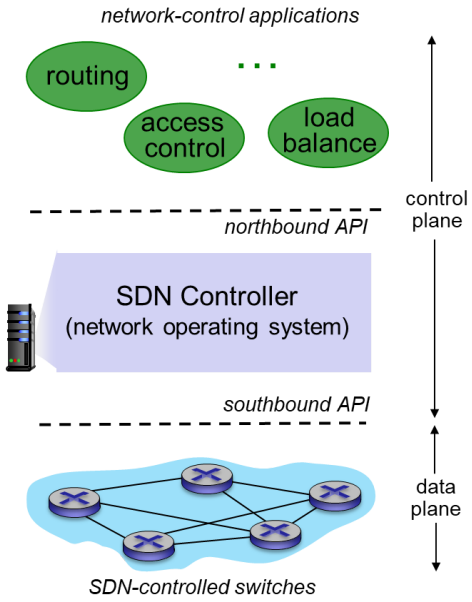
Q: what if w wants to route blue and red traffic differently from w to z?

A: can't do it (with destination-based forwarding, and LS, DV routing)

- `generalized forwarding and SDN can be used to achieve` *`any`* `routing desired`

Data-plane switches:

- fast, simple, commodity switches implementing generalized data-plane forwarding (Section 4.4) in hardware
- flow (forwarding) table computed, installed under controller supervision
- API for table-based switch control (e.g., OpenFlow)
  - defines what is controllable, what is not
- protocol for communicating with controller (e.g., OpenFlow)



SDN controller (network OS):

- maintain network state information
- interacts with network control applications “above” via northbound API
- interacts with network switches “below” via southbound API
- implemented as distributed system for performance, scalability, fault-tolerance, robustness

network-control apps:

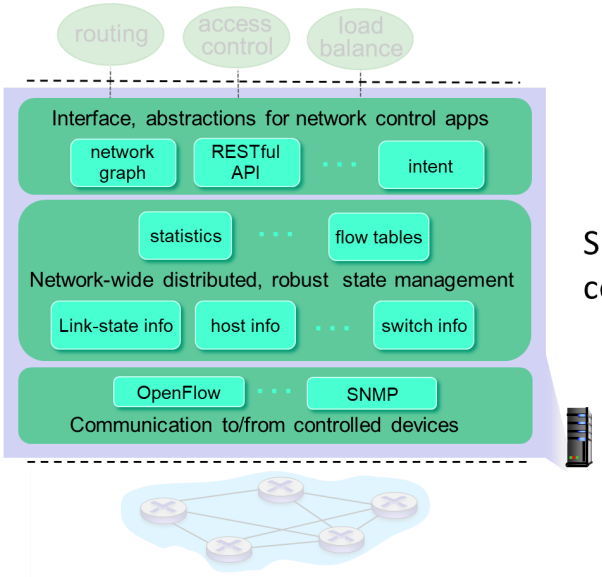
- "brains" of control: implement control functions using lower-level services, API provided by SDN controller
- *unbundled:* can be provided by 3rd party: distinct from routing vendor, or SDN controller

Components of SDN controller

interface layer to network control apps: abstractions API

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

communication: communicate between SDN controller and controlled switches



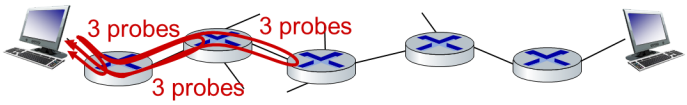
SDN controller

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

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

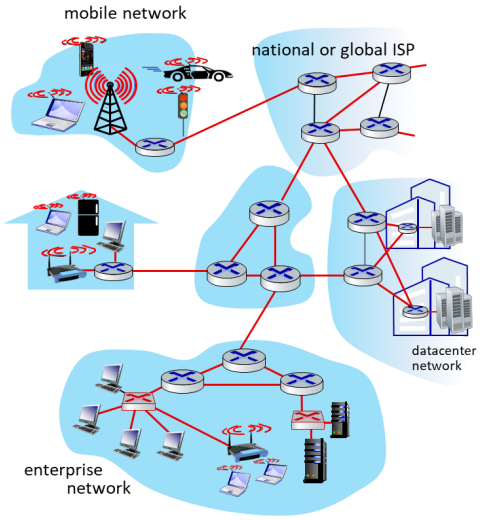
▼ **Ch 6: The Link Layer and LANs**

Link layer: introduction

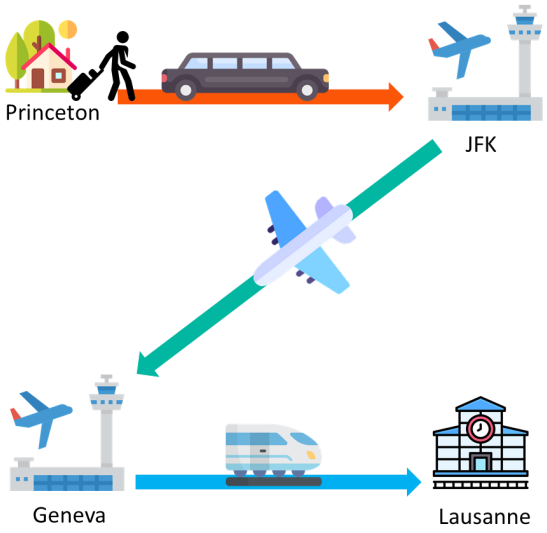
terminology:

- hosts, 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



Transportation analogy

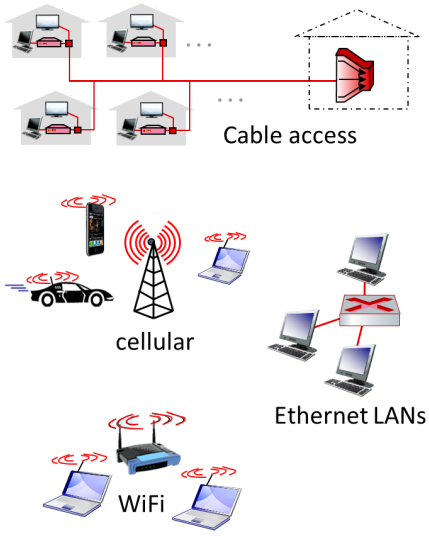


transportation analogy:

- trip from Princeton to Lausanne
  - limo: Princeton to JFK
  - plane: JFK to Geneva
  - train: Geneva to Lausanne
- tourist = datagram
- transport segment = communication link
- transportation mode = link-layer protocol
- travel agent = routing algorithm

Link layer: services

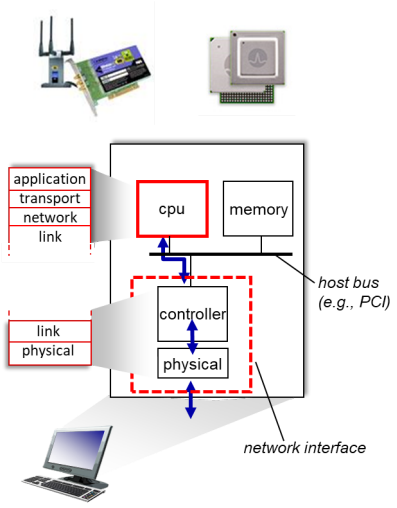
- framing, link access:
  - encapsulate datagram into frame, adding header, trailer
  - channel access if shared medium
  - "MAC" addresses in frame headers identify source, destination (different from IP address!)
- reliable delivery between adjacent nodes
  - we already know how to do this!
  - seldom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



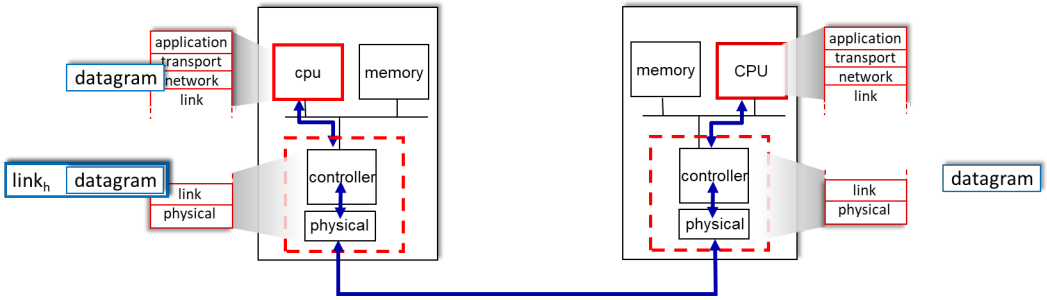
- flow control:
  - pacing between adjacent sending and receiving nodes
- error detection:
  - errors caused by signal attenuation, noise.
  - receiver detects errors, signals retransmission, or drops frame
- error correction:
  - receiver identifies *and corrects* bit error(s) without retransmission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

Host link-layer implementation

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



Interfaces communicating



sending side:

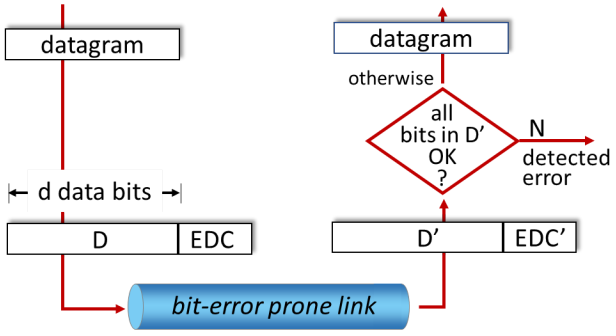
- encapsulates datagram in frame
- adds error checking bits, reliable data transfer, flow control, etc.

receiving side:

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

Error detection

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



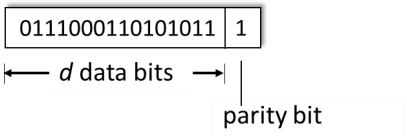
Error detection not 100% reliable!

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

Parity checking

single bit parity:

- detect single bit errors



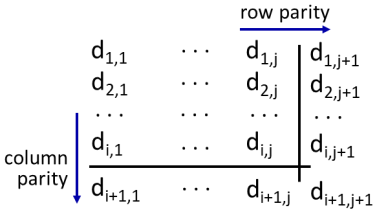
Even/odd parity: set parity bit so there is an even/odd number of 1's

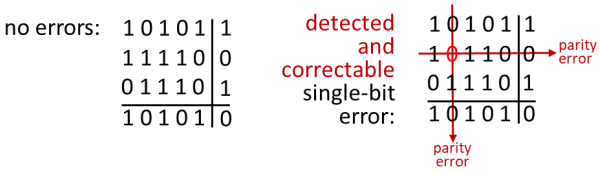
At receiver:

- compute parity of *d* received bits
- compare with received parity bit – if different than error detected

Can detect *and* correct errors (without retransmission!)

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





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

Internet checksum (review, see section 3.3)

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

sender:

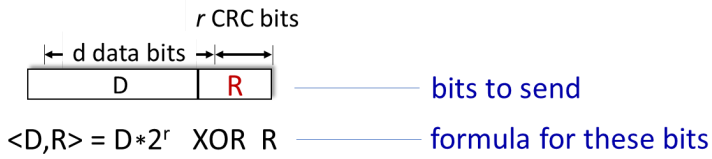
- treat contents of UDP segment (including UDP header fields and IP addresses) as sequence of 16-bit integers
- checksum: addition (one's complement sum) of segment content
- checksum value put into UDP checksum field

receiver:

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

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, specified in CRC standard)



sender: compute *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)

Cyclic Redundancy Check (CRC): example

Sender wants to compute R such that:

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

... or equivalently (XOR R both sides):

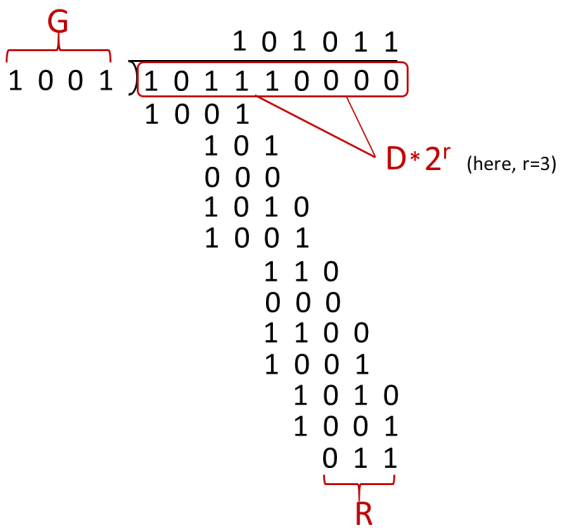
$D \cdot 2^r = nG \ XOR \ R$

... which says:

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

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

algorithm for computing R



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

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-school Ethernet
  - upstream HFC in cable-based access network
  - 802.11 wireless LAN, 4G/4G. 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

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 protocols: taxonomy

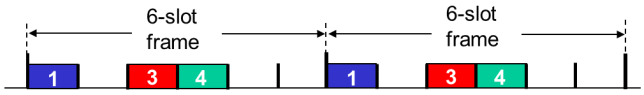
three broad classes:

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

Channel partitioning MAC protocols: TDMA

TDMA: time division multiple access

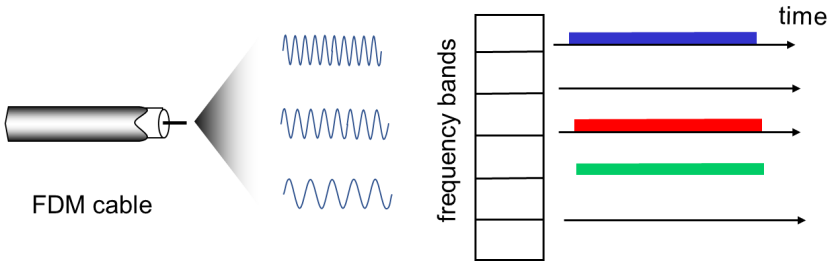
- access to channel in “rounds”
- each station gets fixed length slot (length = packet transmission time) in each round
- unused slots go idle
- example: 6-station LAN, 1,3,4 have packets to send, slots 2,5,6 idle



Channel partitioning MAC protocols: FDMA

FDMA: frequency division multiple access

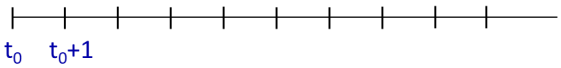
- channel spectrum divided into frequency bands
- each station assigned fixed frequency band
- unused transmission time in frequency bands go idle
- example: 6-station LAN, 1,3,4 have packet to send, frequency bands 2,5,6 idle



Random access 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 protocol specifies:
  - 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

Slotted ALOHA



assumptions:

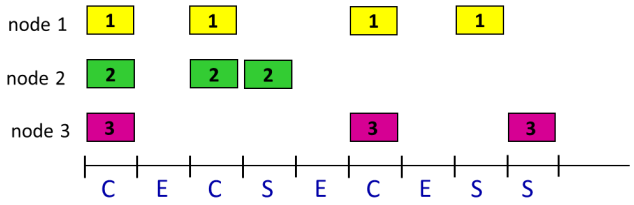
- all frames same size
- time divided into equal size slots (time to transmit 1 frame)
- nodes start to transmit only slot beginning
- nodes are synchronized
- if 2 or more nodes transmit in slot, all nodes detect collision

operation:

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

randomization – *why*?

Slotted ALOHA



C: collision
S: success
E: empty

Pros:

- single active node can continuously transmit at full rate of channel
- highly decentralized: only slots in nodes need to be in sync
- simple

Cons:

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

Slotted ALOHA: efficiency

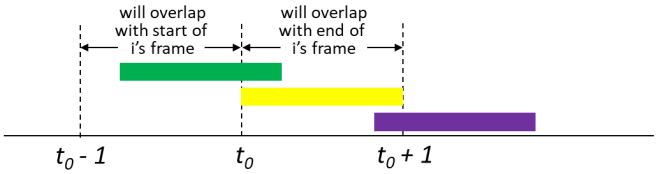
efficiency: long-run fraction of successful slots (many nodes, all with many frames to send)

- *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 37% of time!

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 (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 in determining collision probability

CSMA/CD:

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

Ethernet CSMA/CD algorithm

1. Ethernet receives datagram from network layer, creates frame
2. If Ethernet senses channel:
   if idle: start frame transmission.
   if busy: wait until channel idle, then transmit
3. If entire frame transmitted without collision - done!
4. If another transmission detected while sending: abort, send jam signal
5. After aborting, enter *binary (exponential) backoff:*
   - after *m*th collision, chooses *K* at random from $\{0,1,2, \ldots, 2^m-1\}$. Ethernet waits $K \cdot 512$ bit times, returns to Step 2
   - more collisions: longer backoff interval

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!

"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!

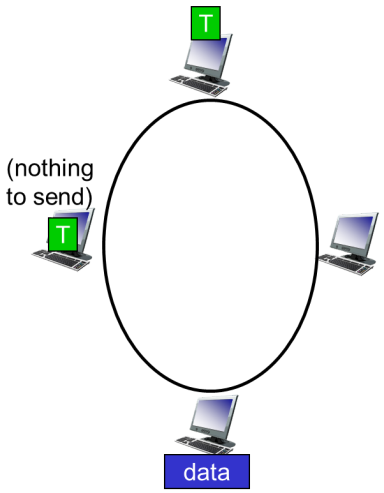
polling:

- centralized controller “invites” other nodes to transmit in turn
- typically used with “dumb” devices
- concerns:
  - polling overhead
  - latency
  - single point of failure (master)
- Bluetooth uses polling

“Taking turns” MAC protocols

token passing:

- control *token* message explicitly passed from one node to next, sequentially
  - transmit while holding token
- concerns:
  - token overhead
  - latency
  - single point of failure (token)



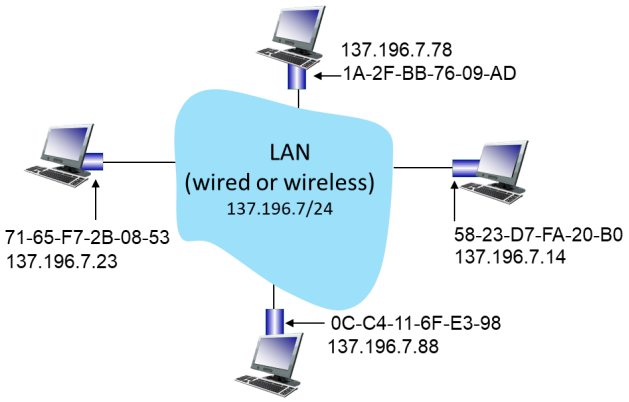
MAC addresses

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

hexadecimal (base 16) notation (each “numeral” represents 4 bits)

each interface on LAN

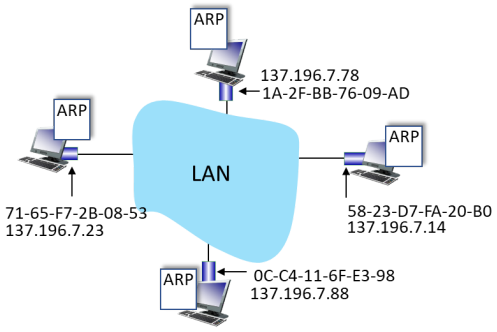
- has unique 48-bit MAC address
- has a locally unique 32-bit IP address (as we've seen)



- 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

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)

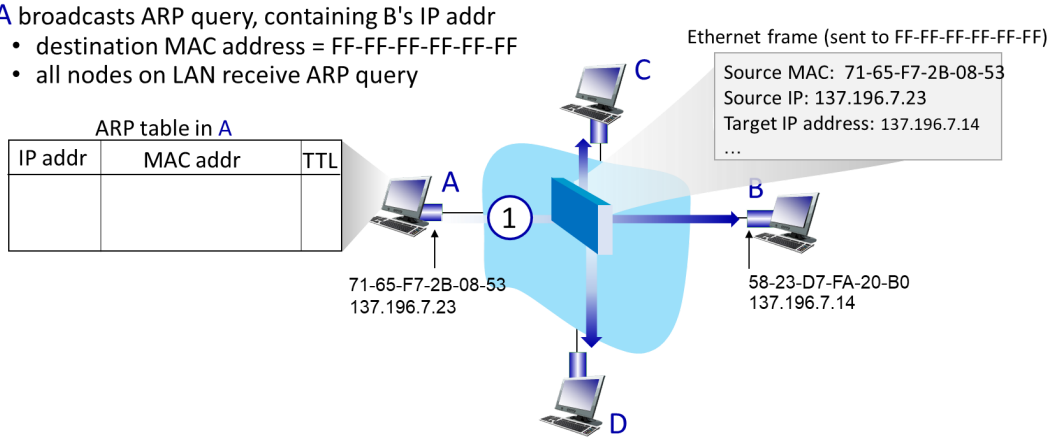
ARP protocol in action

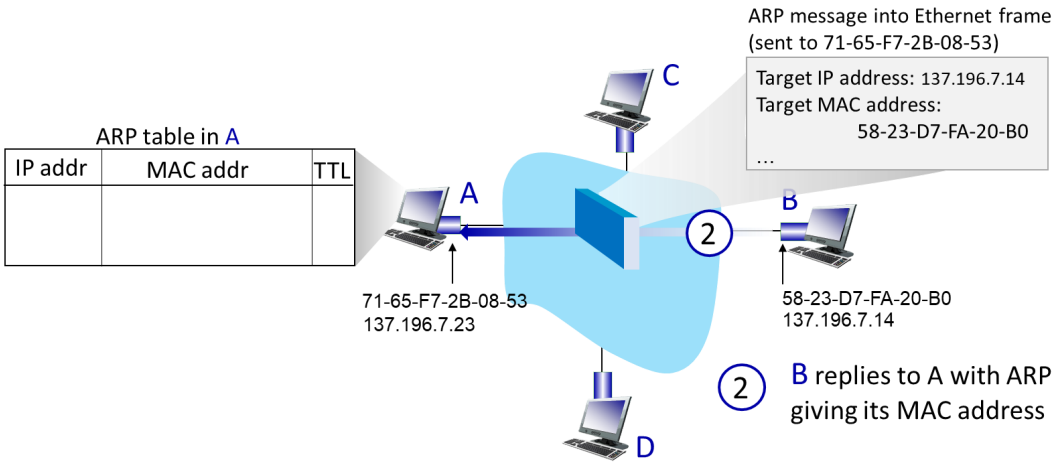
example: A wants to send datagram to B

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

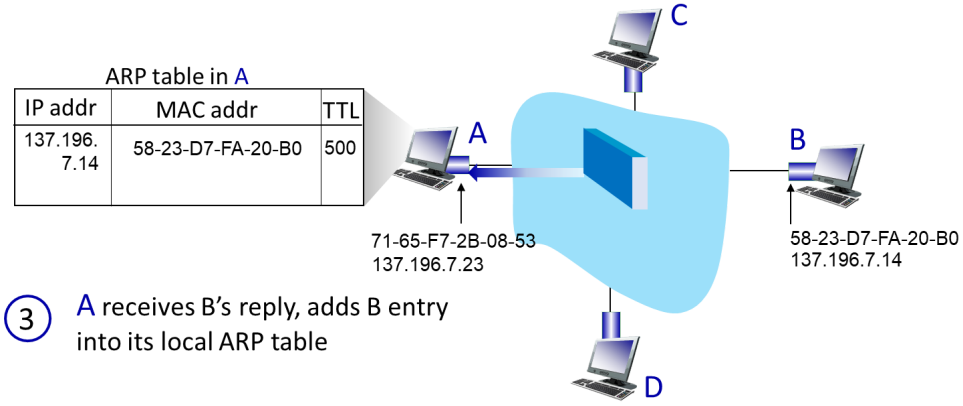
A broadcasts ARP query, containing B's IP addr

1. destination MAC address = FF-FF-FF-FF-FF-FF
   all nodes on LAN receive ARP query





2 B replies to A with ARP response, giving its MAC address

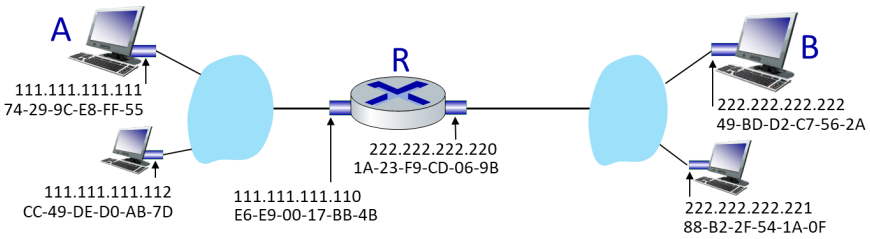


3 A receives B's reply, adds B entry into its local ARP table

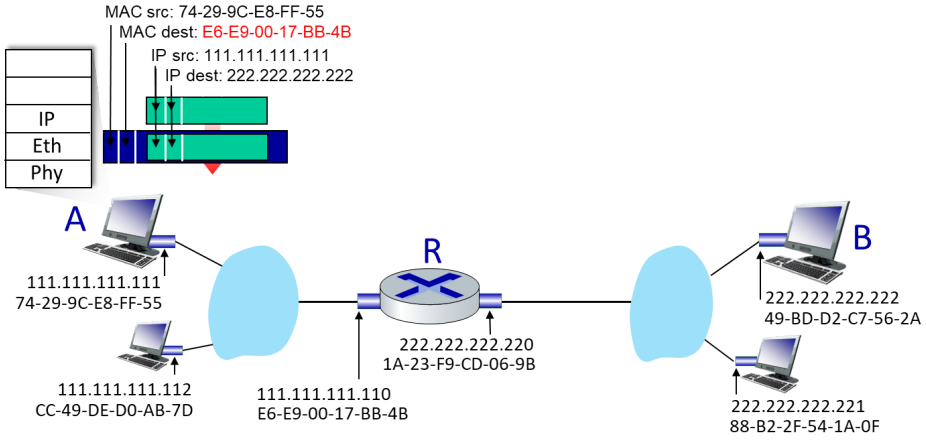
Routing to another subnet: addressing

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

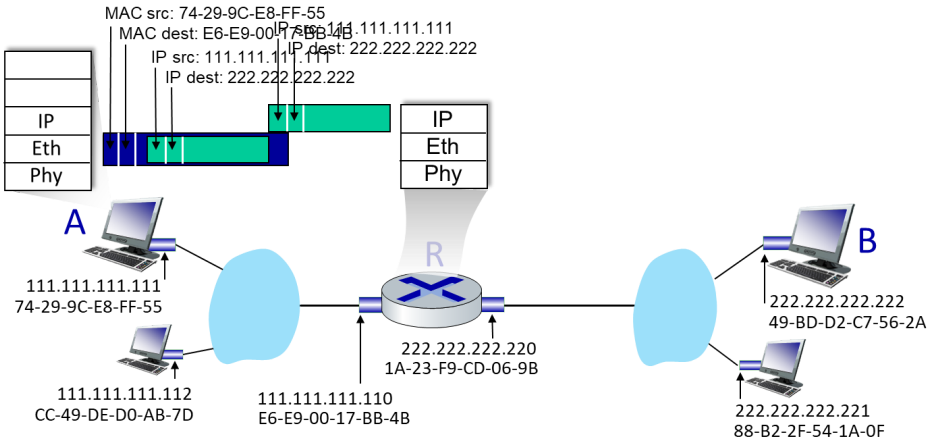
- focus on addressing – at IP (datagram) and MAC layer (frame) levels
- assume that:
  - A knows B's IP address
  - A knows IP address of first hop router, R (how?)
  - A knows R's MAC address (how?)



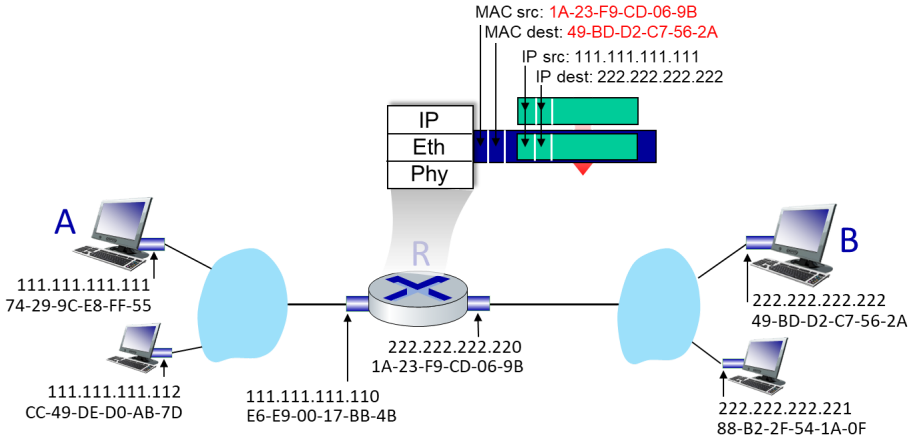
- A creates IP datagram with IP source A, destination B
- A creates link-layer frame containing A-to-B IP datagram
  - R's MAC address is frame's destination



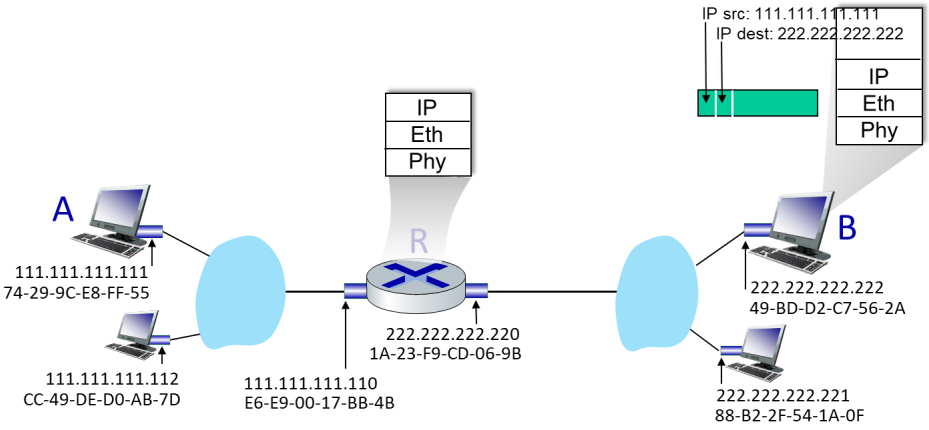
- frame sent from A to R
- frame received at R, datagram removed, passed up to IP



- R determines outgoing interface, passes datagram with IP source A, destination B to link layer
- R creates link-layer frame containing A-to-B IP datagram. Frame destination address: B's MAC address



- B receives frame, extracts IP datagram destination B
- B passes datagram up protocol stack to IP



Ethernet: physical topology

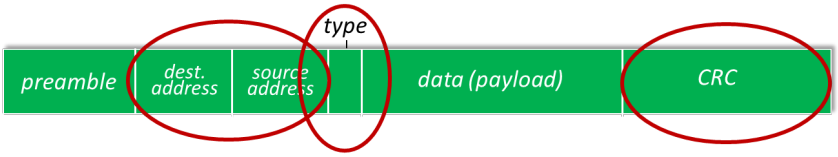
- bus: popular through mid 90s
  - all nodes in same collision domain (can collide with each other)
- switched: prevails today
  - active link-layer 2 *switch* in center
  - each "spoke" runs a (separate) Ethernet protocol (nodes do not collide with each other)

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



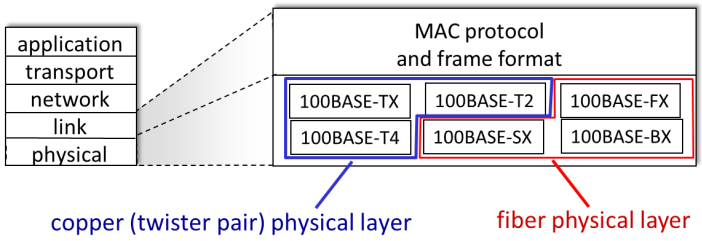
- addresses: 6 byte source, destination MAC addresses
  - if adapter receives frame with matching destination address, or with broadcast address (e.g., ARP packet), it passes data in frame to network layer protocol
  - otherwise, adapter discards frame
- type: indicates higher layer protocol
  - mostly 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: 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

802.3 Ethernet standards: link & physical layers

- *many* different Ethernet standards
  - common MAC protocol and frame format
  - different speeds: 2 Mbps, ... 100 Mbps, 1Gbps, 10 Gbps, 40 Gbps, 80 Gbps
    - different physical layer media: fiber, cable

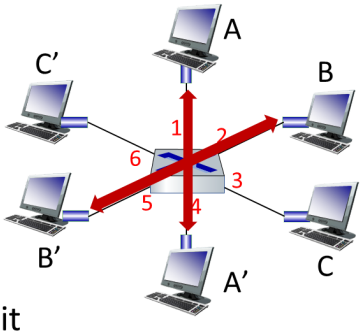


Ethernet switch

- Switch is a link-layer device: takes an *active* role
  - store, forward Ethernet (or other type of) 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

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)

- but A-to-A' and C to A' can *not* happen simultaneously

Switch forwarding table

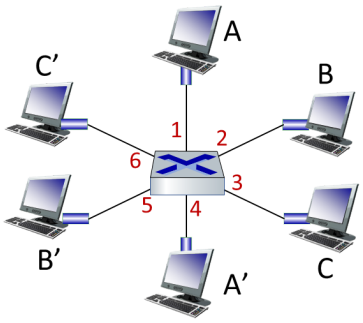
Q: how does switch know A' reachable via interface 4, B' reachable via interface 5?

A: each switch has a switch table, each entry:

- (MAC address of host, interface to reach host, time stamp)
- looks like a routing table!

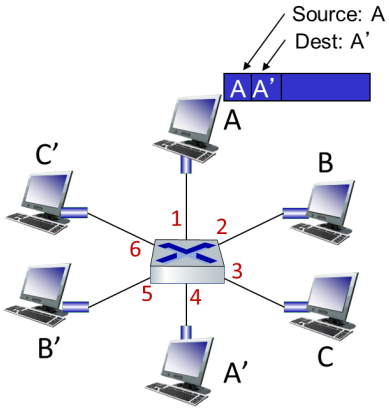
Q: how are entries created, maintained in switch table?

- something like a routing protocol?



Switch: self-learning

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



MAC addr	interface	TTL
A	*1*	*60*

Switch table (initially empty)

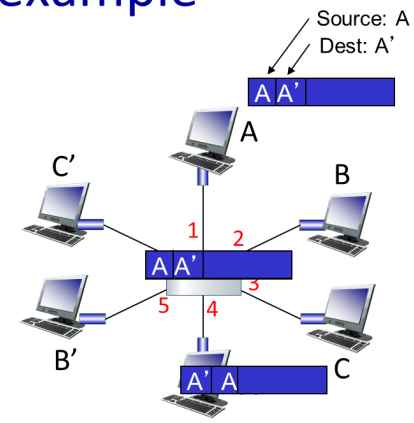
Switch: frame filtering/forwarding

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

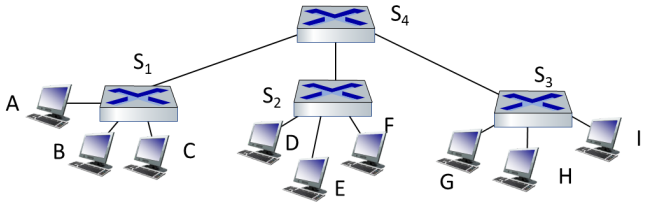


MAC addr	interface	TTL
A	1	60
A'	4	60

switch table (initially empty)

Interconnecting switches

self-learning switches can be connected together:

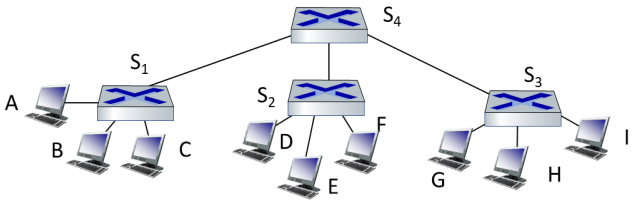


Q: sending from A to G - how does S_1 know to forward frame destined to G via S_4 and S_3?

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

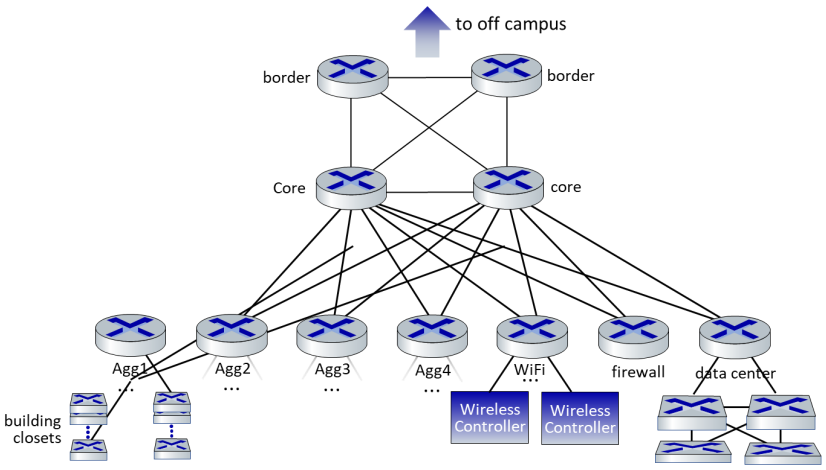
Self-learning multi-switch example

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

UMass Campus Network - Detail



UMass network:

- 4 firewalls
- 10 routers
- 2000+ network switches
- 6000 wireless access points
- 30000 active wired network jacks
- 55000 active end-user wireless devices

… all built, operated, maintained by ~15 people

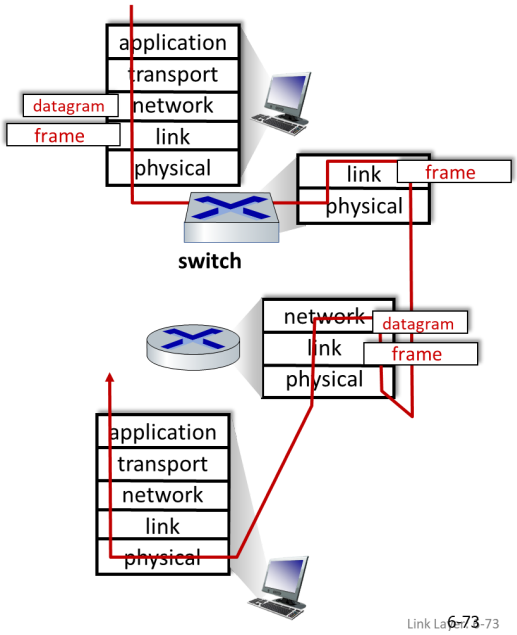
Switches vs. routers

both are store-and-forward:

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

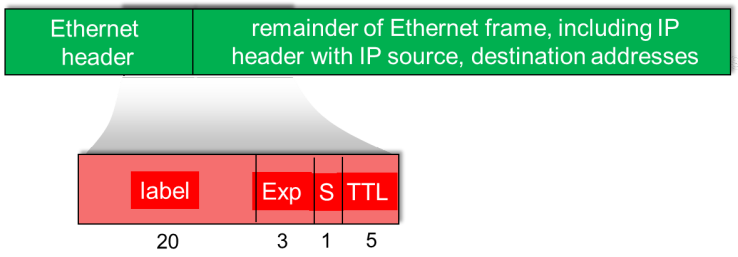
both have forwarding tables:

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



Multiprotocol label switching (MPLS)

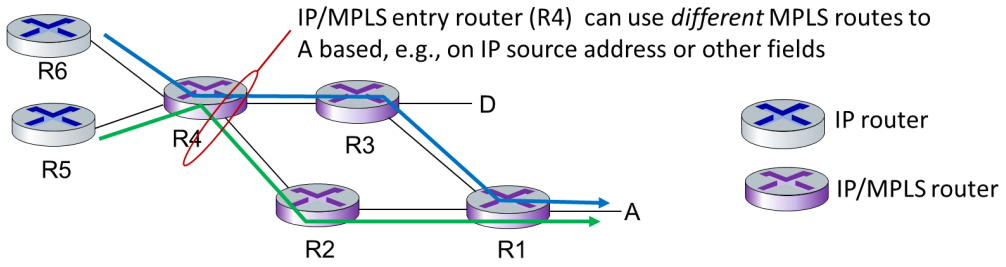
- goal: high-speed IP forwarding among network of MPLS-capable routers, using fixed length label (instead of shortest prefix matching)
  - faster lookup using fixed length identifier
  - borrowing ideas from Virtual Circuit (VC) approach
  - but IP datagram still keeps IP address!



MPLS capable routers

- a.k.a. label-switched router
- forward packets to outgoing interface based only on label value (*don't inspect IP address*)
  - MPLS forwarding table distinct from IP forwarding tables
- *flexibility:* MPLS forwarding decisions can *differ* from those of IP
  - use destination *and* source addresses to route flows to same destination differently (traffic engineering)
  - re-route flows quickly if link fails: pre-computed backup paths

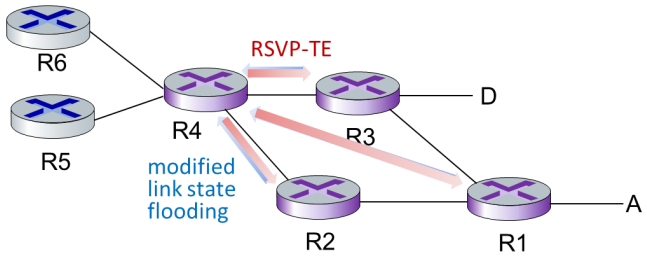
MPLS versus IP paths



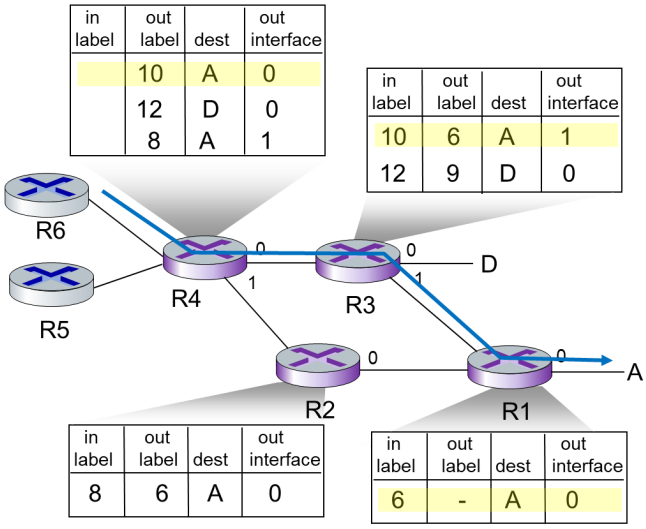
- IP routing: path to destination determined by destination address alone
- MPLS routing: path to destination can be based on source *and* destination address
  - flavor of generalized forwarding (MPLS 10 years earlier)
  - *fast reroute:* precompute backup routes in case of link failure

MPLS signaling

- modify OSPF, IS-IS link-state flooding protocols to carry info used by MPLS routing:
  - e.g., link bandwidth, amount of "reserved" link bandwidth
- entry MPLS router uses RSVP-TE signaling protocol to set up MPLS forwarding at downstream routers



MPLS forwarding tables



-