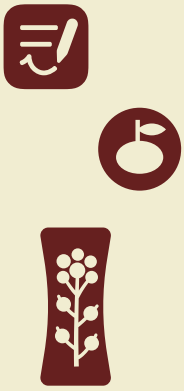


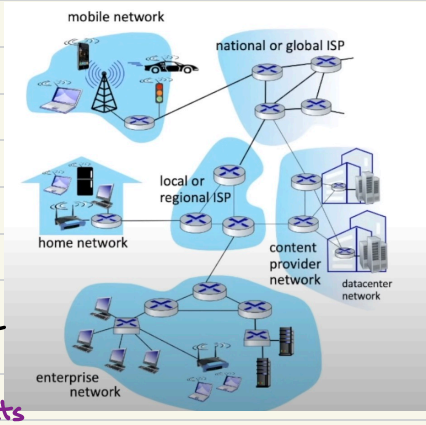
Network edge: Hosts, access network, physical media
Network core: packet / circuit switching, internet structure
performance: loss, delay, throughput

} overview

Internet: Billions of connected computing devices.
hosts → end systems
packet switches: forward packets (chunks of data) → routers, switches
Communication links: fiber, copper, radio, satellite — Transmission rate: bandwidth

---

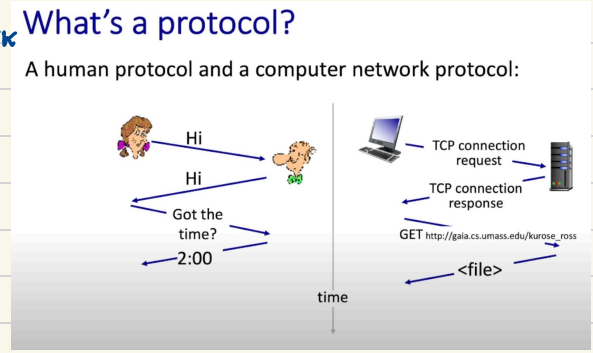
Internet "nuts and bolts" view:
Internet: "networks of networks" (Interconnected ISPs)
protocols: control sending, receiving of messages
↳ HTTP, streaming video, TCP, Skype, WiFi, 4G
protocols are everywhere
Because protocols describe a standard way of doing things, there needs to be a body that defines these standards → IETF: Internet engineering task force
Their standards are known as RFC: Request for comments



Internet "as a service" view:
Infrastructure that provides services to applications → web, streaming video, multimedia, email, games, ...
* provides programming Interface:
"hooks" allowing sending/receiving apps to connect to, use Internet transport service
provides service options, analogous to postal service

protocols define the format, order of messages sent and recieved among network entities, and actions taken on msg transmission, receipt.

All communications in internet are governed by protocols.

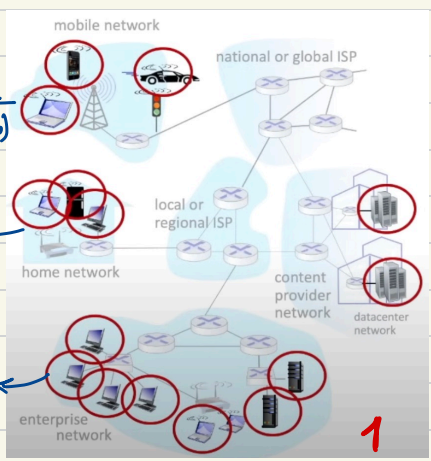


Internet structure:

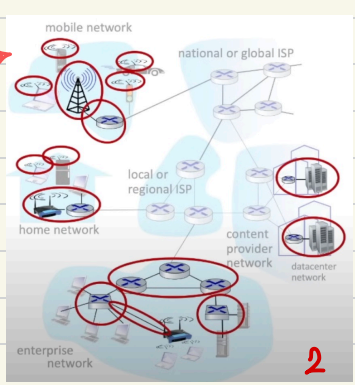
Network edge: Host: clients and servers → often in data centers

What to look for?

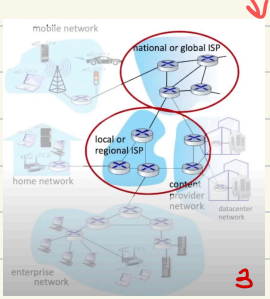
- Transmission rate (bits/sec) of access network?
- shared or dedicated access among users?



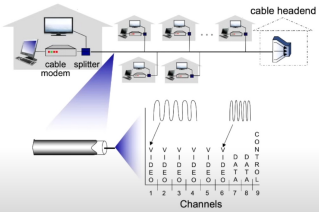
Access Networks: wired, wireless communication links →



Network core: Interconnected routers / networks of networks



Access networks: cable-based access

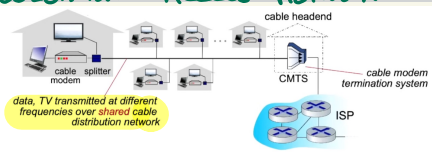


frequency division multiplexing (FDM): different channels transmitted in different frequency bands

Signals to/from different houses are sent on different frequencies → they don't interfere w/ each other

since there are limited no. of frequencies available cable users often share a cable frequency w/ their neighbors.

Residential Access network



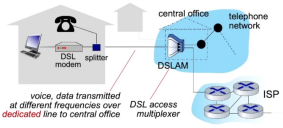
- HFC: hybrid fiber coax
  - asymmetric: up to 40 Mbps – 1.2 Gbs downstream transmission rate, 30-100 Mbps upstream transmission rate
- network of cable, fiber attaches homes to ISP router
  - homes *share access network* to cable headend

Usually asymmetric → higher downstream than upstream

users are mostly consumers of data than producers of data.

digital subscriber line

Access networks: digital subscriber line (DSL)

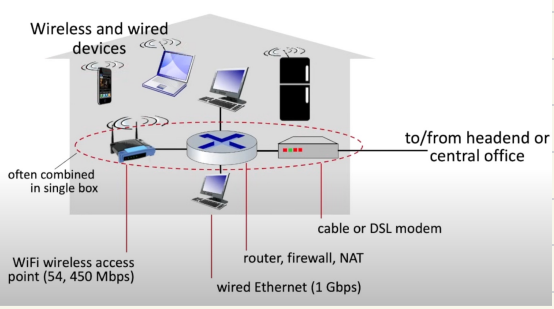


- use *existing* telephone line to central office DSLAM
  - data over DSL phone line goes to Internet
  - voice over DSL phone line goes to telephone net
- 24-52 Mbps dedicated downstream transmission rate
- 3.5-16 Mbps dedicated upstream transmission rate

→ in this access network, you don't share transmission capacity / bandwidth with neighbors

* Asymmetric (higher downstream)

Access networks: home networks



overview of a home network

Wireless access network:
- WLANs (wireless local area network)
- Wide-area cellular access network

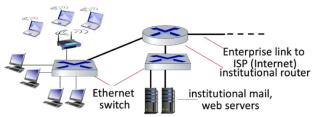
via base station aka "access point"

shared wireless access networks connect end users to router

WLANs are much shorter distances (35 meters)

Cellular support longer distances (~ 10 Km)

Access networks: enterprise networks

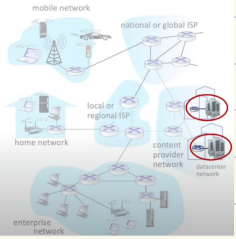


- companies, universities, etc.
- mix of wired, wireless link technologies, connecting a mix of switches and routers (we'll cover differences shortly)
  - Ethernet: wired access at 100Mbps, 1Gbps, 10Gbps
  - WiFi: wireless access points at 11, 54, 450 Mbps

Access networks: data center networks

- high-bandwidth links (10s to 100s Gbps) connect hundreds to thousands of servers together, and to Internet

Courtesy: Massachusetts Green High Performance Computing Center (mghpcc.org)



Sending data: Host want to send a file. They break the file into smaller chunks (packets) of length L bits.

example: File size: 2L → $\frac{2L}{L}$ = 2 packets

The host transmits the data into access network at transmission rate R

R → link bandwidth

packet transmission delay = time needed to transmit L-bit packet into link

$= \frac{L}{R}$

Physical media:

- Guided media → signals propagate in solid media: copper, fiber, coax
- Unguided media → signals propagate freely, e.g., radio

Coaxial cable:

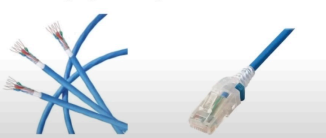
- two concentric copper conductors
- bidirectional
- broadband:
  - multiple frequency channels on cable
  - 100's Mbps per channel

Fiber optic cable:

- glass fiber carrying light pulses, each pulse a bit
- high-speed operation:
  - high-speed point-to-point transmission (10's-100's Gbps)
- low error rate:
  - repeaters spaced far apart
  - immune to electromagnetic noise

Twisted pair (TP)

- two insulated copper wires
  - Category 5: 100 Mbps, 1 Gbps Ethernet
  - Category 6: 10Gbps Ethernet



Wireless is less safe → might allow eavesdropping

Links: physical media

Wireless radio

- signal carried in various "bands" in electromagnetic spectrum
- no physical "wire"
- broadcast, "half-duplex" (sender to receiver)
- propagation environment effects:
  - reflection
  - obstruction by objects
  - Interference/noise

Radio link types:

- Wireless LAN (WiFi)
  - 10-100's Mbps; 10's of meters
- wide-area (e.g., 4G cellular)
  - 10's Mbps over ~10 Km
- Bluetooth: cable replacement
  - short distances, limited rates
- terrestrial microwave
  - point-to-point; 45 Mbps channels
- satellite
  - up to 45 Mbps per channel
  - 270 msec end-end delay

Two key functions → Forwarding & Routing

Forwarding (switching): local action, moves arriving packets from router's input link to appropriate router output link.

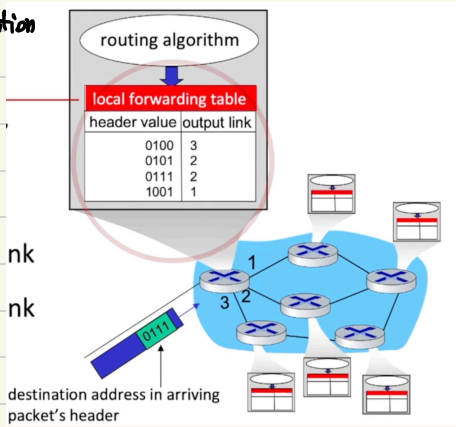
*There is a forwarding table in every router connected to the internet.

The network core

- mesh of interconnected routers
- 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

Routing: global action, determines source-destination paths taken by packets.

* Routing Algorithms compute these paths and update the tables.



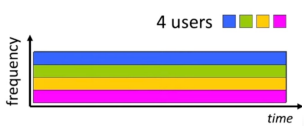
Alternative to packet switching

↳ Circuit switching: end-end resources allocated to, reserved for "call" between source and destination → no more queues

Two ways to do circuit switching

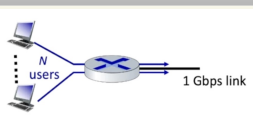
Circuit switching: FDM and TDM

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



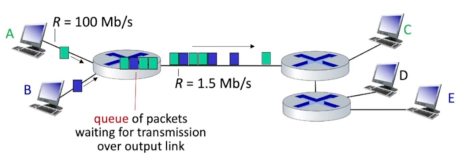
Time Division Multiplexing (TDM)
- time divided into slots
- each call allocated periodic slot(s), can transmit at maximum rate of (wider) frequency band (only) during its time slot(s)

example:
- 1 Gb/s link
- each user:
  - 100 Mb/s when "active"
  - active 10% of time



Q: how many users can use this network under circuit-switching and packet switching?

Packet-switching: queueing



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

Circuit switching → max users x 100 Mb/s = 1 Gb/s → N = 10 (reserved)

Packet switching → p = 0.1 (probability of being active)

a users active → N = 35 for a > 10 → $P_{N/a} < 0.0004$

↳ we assume it's zero

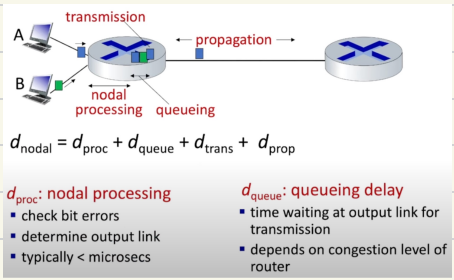
Is packet switching always better? It is good with "bursty data" meaning → sometimes has to send, but at other times not

Pros → resource sharing, simpler, no call setup

Cons → excessive congestion possible, packet delay and loss due to buffer overflow, protocols needed for reliable data transfer and congestion control

Question: How to improve circuit-like behavior with packet-switching?

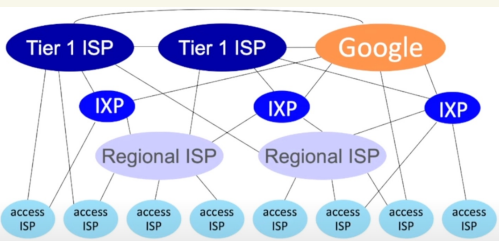
→ try to make packet switching as "circuit-like" as possible.



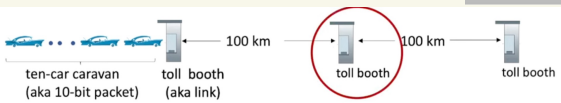
$d_{nodal} = d_{proc} + d_{queue} + d_{trans} + d_{prop}$

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



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



- 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?

12 x 10 sec + 1 h => 62 minutes

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)
- $d_{prop} = d/s$

near speed of light, but quite noticeable

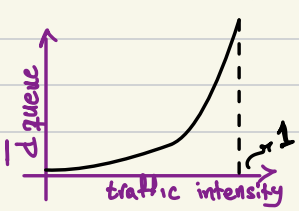
a: average packet arrival rate

L: packet length (bits)

R: link bandwidth (bit transmission rate)

$\frac{L \cdot a}{R}$ → average bits/sec → bits/sec } → traffic intensity

* when $\frac{La}{R} > 1$ → "work" arriving is more than what can be serviced



$\frac{La}{R} \approx 0$: d_{queue} is small

$\frac{La}{R} \approx 1$: d_{queue} is large

$\frac{La}{R} \approx > 1$: d_{queue} is ∞

Real Internet delays and routes

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

traceroute sends three packets

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

trans-oceanic link → propagation delay

looks like delays *decrease*! Why? queuing delay & congestion level changes over time

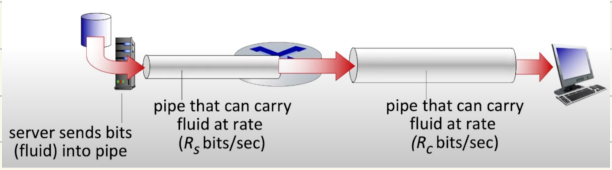
no response (lost or router refuses to talk)

Packet loss: queue (buffer) has finite capacity. → packets arriving to full queue, get dropped (aka lost)

dropped packets might get retransmitted

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



$R_s < R_c$: average throughput is R_s

$R_s > R_c$: av. throughput is R_c

$\min(R_s, R_c)$ is the bottleneck link

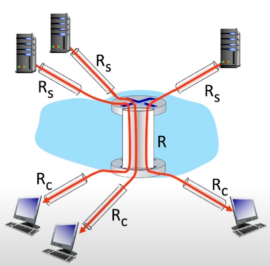
10 connections go through $R \Rightarrow \frac{R}{10}$ per connection

bottleneck $\left(\frac{R}{10}, R_c, R_s\right)$

smallest R_c

smallest R_s

Throughput: network scenario

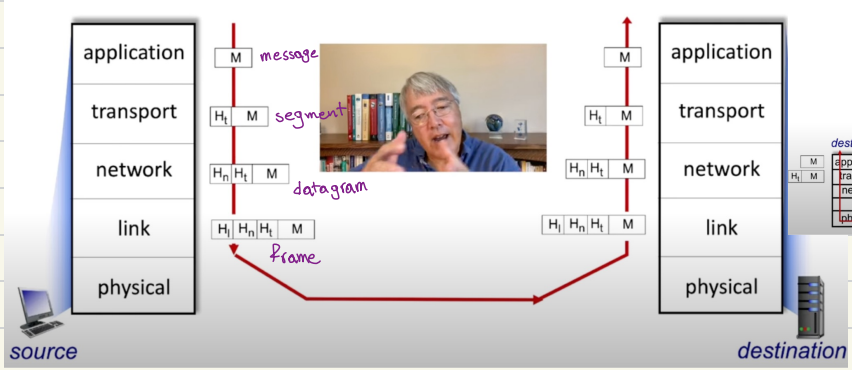


10 connections (fairly) share backbone bottleneck link *R* bits/sec

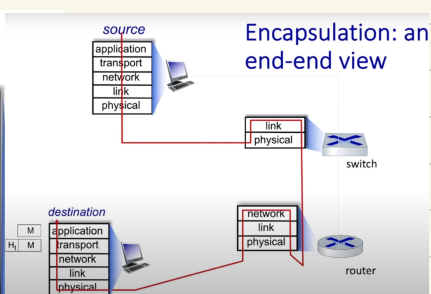
- per-connection end-end throughput: $min(R_c, R_s, R/10)$
- in practice: R_c or R_s is smaller than R/N
  - bottleneck link is at network edge

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

Services, Layering and Encapsulation



Encapsulation: an end-end view



Application layer: where most of our use of the internet is located.

Two Architectures → client-server, P2P

- social networking
- Web
- text messaging
- e-mail
- multi-user network games
- streaming stored video (YouTube, Hulu, Netflix)
- P2P file sharing
- voice over IP (e.g., Skype)
- real-time video conferencing (e.g., Zoom)
- Internet search
- remote login
- …

Client-server model:

Server → always-on host, permanent IP address, often in data centers

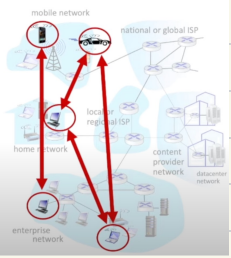
Clients → communicates with server, may be intermittently connected, may have dynamic IP addresses, do not communicate directly with each other

Examples: HTTP, IMAP, FTP

Peer-peer architecture

- *no* always-on server
- arbitrary end systems directly communicate
- peers request service from other peers, provide service in return to other peers
  - *self scalability* – new peers bring new service capacity, as well as new service demands
- peers are intermittently connected and change IP addresses
  - complex management

Example: P2P File Sharing



Process: Program running within a host

→ within same host, two processes communicate using inter-process communication (defined by the OS)

→ processes in different hosts communicate by exchanging messages.

Client process → process that initiates communication

Server process → process that waits to be contacted

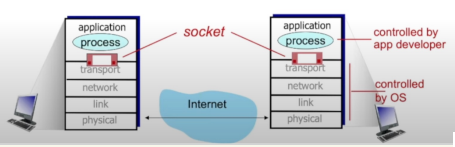
} → application with P2P arch have client AND server processes

Sockets

Socket → door

note: → door 1 → door 2 →

- 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



Addressing processes

- to receive messages, process must have *identifier*
- host device has unique 32-bit IP address
- *Q:* does IP address of host on which process runs suffice for identifying the process?
  - *A:* no, *many* processes can be running on same host
- *identifier* includes both IP address and port numbers associated with process on host.
- example port numbers:
  - HTTP server: 80
  - mail server: 25

  } → protocols for some ports; web server → 80
- to send HTTP message to gaia.cs.umass.edu web server:
  - IP address: 128.119.245.12
  - port number: 80
- more shortly…

We need ways of identification → Address

How to define a network protocol?

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

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

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

security
- encryption, data integrity, ...

Transport service requirements: common apps

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

Internet transport protocols services

TCP service:
- *reliable transport* between sending and receiving process
- *flow control:* sender won't overwhelm receiver
- *congestion control:* throttle sender when network overloaded
- *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?

we can add the services that are not provided by UDP in the Application layer

Internet applications, and transport protocols

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

TCP is much more widely used

Securing TCP

Vanilla TCP & UDP sockets:
- no encryption
- cleartext passwords sent into socket traverse Internet in cleartext (!)

Transport Layer Security (TLS) → Implement in user application space, on top of TCP sockets
- provides encrypted TCP connections
- data integrity
- end-point authentication

In the begining there was no encryption for TCP

it sits between application layer and transport layer

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: 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 protocol
→ client initiates TCP connection to server (port 80)
→ server accepts TCP connection from client
→ HTTP messages start to exchange
→ TCP connection closed

HTTP is "stateless" → server maintains no information about past client requests

why? for its simplicity

maintaining state is complex → if server, client crashes data may be inconsistent

HTTP connections: two types

Non-persistent HTTP

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

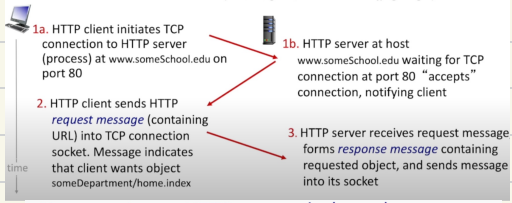
downloading multiple objects required multiple connections

Persistent HTTP → HTTP 1.1

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

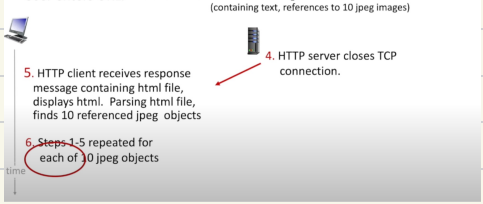
Non-persistent HTTP: example

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



Non-persistent HTTP: example (cont.)

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

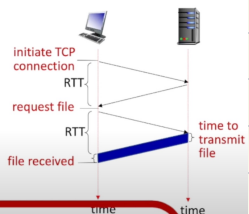


Non-persistent HTTP: response time

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
- obect/file transmission time



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

Non-persistent HTTP issues:

- requires 2 RTTs per object
- OS overhead for each TCP connection
- Browser 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)

carriage return character
line-feed character

POST,

header lines

```
GET /index.html HTTP/1.1\r\n
Host: www-net.cs.umass.edu\r\n
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X
  10.15; rv:80.0) Gecko/20100101 Firefox/80.0 \r\n
Accept: text/html,application/xhtml+xml\r\n
Accept-Language: en-us,en;q=0.5\r\n
Accept-Encoding: gzip,deflate\r\n
Connection: keep-alive\r\n
\r\n
```

ne feed →

Other HTTP request messages

GET request without a body

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

HTTP response status codes

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

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

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

400 Bad Request
- request msg not understood by server

404 Not Found
- requested document not found on this server

505 HTTP Version Not Supported

Maintaining user/server state: cookies

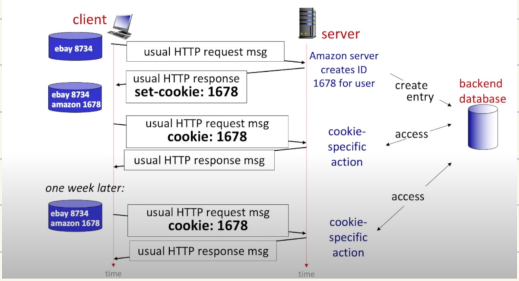
Web sites and client browser use *cookies* to maintain some state between transactions

Adding state to HTTP

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

Maintaining user/server state: cookies



If we delete cookies locally, the web servers will give us new cookies (unless we sign in)

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 inHTTP messages carry state

aside

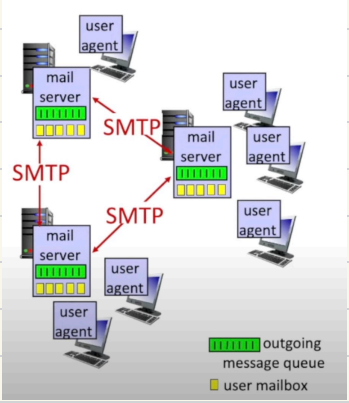
cookies and privacy:
- cookies permit sites to *learn* a lot about you on their site.
- third party persistent cookies (tracking cookies) allow common identity (cookie value) to be tracked across multiple web sites

---

Email: three components → user agents, mail servers, simple mail transfer protocol SMTP

User Agent: mail reader, composing and editing emails
e.g → Outlook, iPhone mail client

Mail servers: mailbox: contains incoming messages for user } Message queue of outgoing (to be sent) mail messages

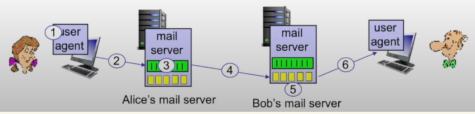


SMTP: Between mail servers to send emails.
"client" → sending mail server → gmail
"server" → receiving mail server → yahoo

SMTP uses TCP to reliably transfer email message from client

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 to HTTP:

HTTP: client pull

SMTP: client push

both have ASCII comand/response interactions, status codes

HTTP → each object encapsulated in its own response message

SMTP → multiple objects sent in multipart message. Also, it uses persistent connections (multiple emails transferred over a single SMTP connection).

Sample SMTP interaction

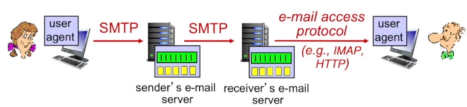
very human readable

message

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

SMTP is for sending/pushing emails. In order to receive/pull the message, we need to use IMAP or HTTP

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

---

DNS: built on top of the Application layer

Internet hosts → IP Address. used for addressing datagrams

↳ name: used by humans

Domain Name system:

Distributed database implemented in hierarchy of many name servers

Application-layer protocol: hosts, DNS servers communicate to resolve names

* Complexity at networks "edge" → making the network core simple and the edge, complex

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

Local DNS name server:

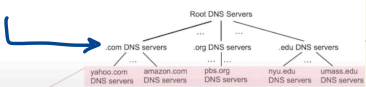
when host makes DNS query, it is sent to its local DNS server. local DNS replies by:

→ from its local cache of recent name to address translation pairs (possibly out of date)

→ forwarding request into DNS hierarchy for resolution

Top-Level Domain (TLD) servers:

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



authoritative DNS servers:

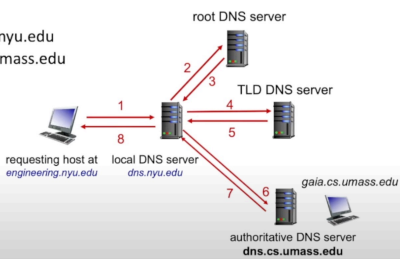
- organization's own DNS server(s), providing authoritative hostname to IP mappings for organization's named hosts
- can be maintained by organization or service provider

DNS name resolution: iterated query

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

Iterated query:

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

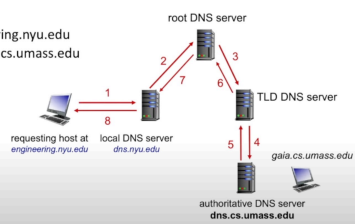


DNS name resolution: recursive query

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

Recursive query:

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



Two methods for querying DNS servers. we mostly use iterative because in recursive, the heavy load is on the upper level of hierarchy (we want more compute on lower levels)

Caching DNS Information

- once (any) name server learns mapping, it *caches* mapping, and *immediately* 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!*

→ if a entry changes its ip, the DNS would have inaccurate information until all of the TTLs get to zero.

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`

DNS security

DDoS attacks
- bombard root servers with traffic
  - not successful to date
  - traffic filtering
  - local DNS servers cache IPs of TLD servers, allowing root server bypass
- bombard TLD servers
  - potentially more dangerous

Spoofing attacks
- intercept DNS queries, returning bogus replies
  - DNS cache poisoning
  - RFC 4033: DNSSEC authentication services

Peer to peer (P2P) architecture:

no always-on server — Peers request service from other peers, provide service in return to other peers

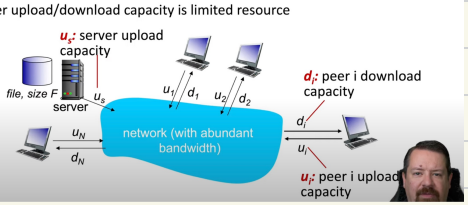
arbitrary systems directly communicate

→ self scalability: new peers bring new service capacity, and new demands.

peers are intermittently connected and change IP addresses

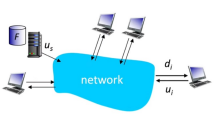
examples → P2P file sharing (BitTorrent), streaming, VoIP (Skype)

Q: how much time to distribute file (size *F*) from one server to *N* peers?
- peer upload/download capacity is limited resource



File distribution time: client-server

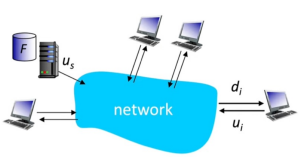
- *server transmission:* must sequentially send (upload) *N* file copies:
  - time to send one copy: F/u_s
  - time to send *N* copies: NF/u_s
- *client:* each client must download file copy
  - d_{min} = min client download rate
  - min client download time: F/d_{min}



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

File distribution time: P2P

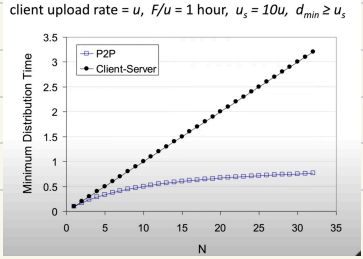
- *server transmission:* must upload at least one copy:
  - time to send one copy: F/u_s
- *client:* each client must download file copy
  - min client download time: F/d_{min}
- *clients:* as aggregate must download *NF* bits
  - max upload rate (limiting max download rate) is $u_s + \Sigma u_i$



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

as N increases, more u_i's also get added which helps with scaling.

client upload rate = u, F/u = 1 hour, $u_s = 10u$, $d_{min} \geq u_s$



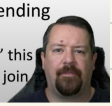
BitTorrent: requesting, sending file chunks

Requesting chunks:

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

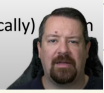
Sending chunks: tit-for-tat

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



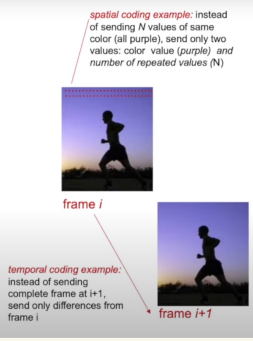
P2P file distribution: BitTorrent

- peer joining torrent:
  - has no chunks, but will accumulate them over time from other peers
  - registers with tracker to get list of peers, connects to subset of peers ("neighbors")
- while downloading, peer uploads chunks to other peers
- peer may change peers with whom it exchanges chunks
- *churn:* peers may come and go
- once peer has entire file, it may (selfishly) leave or (altruistically) leave in torrent



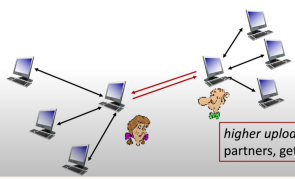
Multimedia: video

- CBR: (constant bit rate): video encoding rate fixed
- VBR: (variable bit rate): video encoding rate changes as amount of spatial, temporal coding changes
- examples:
  - MPEG 1 (CD-ROM) 1.5 Mbps
  - MPEG2 (DVD) 3-6 Mbps
  - MPEG4 (often used in Internet, 64Kbps – 12 Mbps)



BitTorrent: tit-for-tat

(1) Alice "optimistically unchokes" Bob
(2) Alice becomes one of Bob's top-four providers; Bob reciprocates
(3) Bob becomes one of Alice's top-four providers



challenges in video streaming → server-to-client bandwidth will vary over time, with changing network congestion levels (in house, access network, network core, ...)
packet loss, delay due to congestion will delay payout, or result in poor video quality

Streaming multimedia: DASH

*D*ynamic, *A*daptive *S*treaming over *H*TTP

server:

- divides video file into multiple chunks
- each chunk encoded at multiple different rates
- different rate encodings stored in different files
- files replicated in various CDN nodes
- *manifest file:* provides URLs for different chunks

client:

- periodically estimates server-to-client bandwidth
- consulting manifest, requests one chunk at a time
  - chooses maximum coding rate sustainable given current bandwidth
  - can choose different coding rates at different points in time (depending on available bandwidth at time), and from different servers

Streaming multimedia: DASH

- *"intelligence"* at client: client determines
  - *when* to request chunk (so that buffer starvation, or overflow does not occur)
  - *what encoding rate* to request (higher quality when more bandwidth available)
  - *where* to request chunk (can request from URL server that is "close" to client or has high available bandwidth)

Content distribution networks (CDNs)

challenge: how to stream content (selected from millions of videos) to hundreds of thousands of *simultaneous* users?

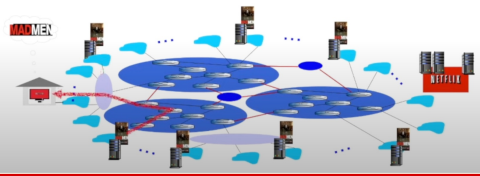
- *option 2:* store/serve multiple copies of videos at multiple geographically distributed sites *(CDN)*
  - *enter deep:* push CDN servers deep into many access networks
    - close to users
    - Akamai: 240,000 servers deployed in > 120 countries (2015)
  - *bring home:* smaller number (10's) of larger clusters in POPs near access nets
    - used by Limelight

Akamai

Limelight NETWORKS

Content distribution networks (CDNs)

- CDN: stores copies of content (e.g. MADMEN) at CDN nodes
- subscriber requests content, service provider returns manifest
  - using manifest, client retrieves content at highest supportable rate
  - may choose different rate or copy if network path congested



Socket programming: one and only API that sits between the application layer and transport layer. → door between application & transport layer

Socket programming with UDP

UDP: no "connection" between client and server:

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

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

Application viewpoint:

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

Transport layer: provides logical communication between application processes running on different hosts → sender breaks the message into segments, passes to network layer

receiver reassembles segments into messages, passes to application layer

Transport vs. network layer services and protocols

- network layer: logical communication between *hosts*
- transport layer: logical communication between *processes*
  - relies on, enhances, network layer services

household analogy:

12 kids in Ann's house sending letters to 12 kids in Bill's house:

- hosts = houses
- processes = kids
- app messages = letters in envelopes
- transport protocol = Ann and Bill who demux to in-house siblings
- network-layer protocol = postal service

Network layer: post service

Transport layer: inside the house

→ logical communications

Two principal Internet transport protocols

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

neither TCP or UDP