Chapter 2 - Application Layer

Wednesday, January 20, 2016 2:41 PM

Network applications are the reason for the existence of computer networks

70s-80s - email, remote access, file transfer
90s - WWW, search, instant messaging, P2P
modern - Voice-over-IP, skype, online gaming, Netflix

- run of different end systems
- communicate over network (ex. browser ↔ server)

Application Architectures

CLIENT-SERVER

server → always on
→ permanent IP
→ data centers for scaling

clients → communicate with server
→ intermittently connected
→ dynamic IP
→ do not communicate directly with e/o

PEER-TO-PEER (ex. bittorrent)

- no always-on server
- end systems directly communicate
- peers request and provide eachother with services
- self-scalability

- complex
  → peers intermittently connected & change IP addresses

- not great for security because don't really know who you're dealing with
- downloads generally much faster

process - program running within a host
- must be uniquely identified on network, ip:port (device (unique 32-bit IP))

↳ process on device

(default 80 on HTTP, default 25 on SMTP)

Processes communicating

- within same host, two processes communicate
- exchange messages

server process - waits to be contacted
client process - initiates communication

Sockets

↳ INTERFACE BETWEEN APPLICATION & TRANSPORT LAYER

- how a process sends/recieves messages
- analogous to a door
  - ↳ sending process puts message out door & relies on transport infrastructure on the other side of door to deliver message to recieving process' socket
- supported by all major programming languages

TRANSPORT LAYER SERVICES FOR APPLICATIONS

1. DATA INTEGRITY/RELIABILITY
2. THROUGHPUT (bandwidth guarantees)
3. TIMING (delay limits)
4. SECURITY (encryption & decryption, point-to-point authentication)

Transport layer protocols

TCP
- → reliable/guaranteed transport (promises all data delivered without error & in order)
- → flow control (sender won't overwhelm reciever)
- → congestion control (throttle sender when network congested)
- → connection oriented (transport layer handshake before sending application messages)

UDP
- → lightweight, "minimal", "no-frills"
- → unreliable data transfer (no guarantee message will be recieved, also might be out-of-order)
- → connectionless (no handshaking before sending messages)
- → no congestion-control (sender will send at whatever rate it wants)

OOTB, neither TCP or UDP provide
- → Timing
- → Throughput
- → Security
  - ↳ although SSL (Secure Sockets Layer) is an application-layer

enhancement of TCP that provides some encryption for passwords
HTTPS is HTTP w/ SSL

APPLICATION & TRANSPORT PROTOCOL USES

application	application layer protocol	underlying transport protocol
email	SMTP	TCP
remote terminal access	Telnet	TCP
web	HTTP	TCP
file transfer	FTP	TCP
streaming-media	HTTP, RTP	TCP, UDP
internet telephony	SIP, RTP, Proprietary	TCP, UDP

APPLICATION LAYER PROTOCOLS (just some examples)

defines types of messages exchanged, syntax of messages, semantics of the fields, and when/how actions are taken.

HTTP The Web

→ client-server model
- has a client program and a server program
- "pull protocol"
- defines how clients request web-pages & how servers transfer web-pages

→ stateless protocol
- server maintains no history of past requests
- if you request the same file 10 times, it will be sent 10 times
  ↳ allows many simultaneous connections, we'll see that FTP isn't stateless and therefore has a limited # of connections
  ↳ we get around this with the use of cookies & caches

→ uses TCP
- HTTP doesn't have to worry about lost data or details of reordering data

NON-PERSISTENT CONNECTIONS

↳ for every TCP connection, only send one object is sent, 2RTT per object
↳ one request message, one response message, then connection closed
↳ can be configured with parallelism
(5 to 10 parallel TCP connections, each non-persistent)
↳ places significant burden on web server & causes delay

PERSISTENT CONNECTIONS

↳ TCP establishes handshake then keeps connection open
↳ requests can be sent back-to-back without waiting for a response (pipelining)

↳ connection closes if not used for set interval (timeout)

Default HTTP uses persistent connections with pipelining

Message format

↳ two types of messages, requests & responses

REQUEST

method URL version

Request line → `GET /index.html HTTP/1.1`

Header lines:
- `Host: www-net.cs.umass.edu` ← used for proxy caches
- `User-Agent: Firefox/3.6.10`
- `Accept: text/html` } many possible content negotiation headers
- `Accept-Language: en-us`
- `Keep-Alive: 115` } creating persistent HTTP
- `Connection: keep-alive`

Blank line →

Body: `Body data if needed` ← empty for GET method, but used with POST method

METHODS GET, POST, HEAD, PUT, DELETE

- HEAD: for checking if cache should be updated
- PUT: for uploading files
- POST: ↳ for when user fills out a form
- DELETE: for deleting files

↳ alternatively, could use GET with URL like someside.com/search?.fruit=kiwi

RESPONSE

status line → `HTTP/1.1 200 OK`

header lines:
- `Date: Sun, 26 Sep 2010 20:09:20 GMT`
- `Server: Apache/2.0.52 (CentOS)`
- `Last-Modified: Tue, 30 Oct 2007 17:00:02 GMT`
- `Content-Length: 6821`
- `Content-Type: text/html`

entity body, data → `<html><title> ....`

Some example codes

- 200 OK
- 301 Moved Permanently ← usually never see
- 400 Bad request ← request not understood
- 404 Not Found
- 505 HTTP Version Not Supported

403 Forbidden
451 Unavailable for Legal Reasons ← censorship

COOKIES

→ HTTP is stateless ∴ no memory
  ↳ means can handle thousands of simultaneous connections
→ cookies are used to allow sites to identify users
→ have timeout lifetime

Components:
1) initial Set-cookie: header line in HTTP response
2) cookie header line in subsequent HTTP requests
3) cookie file kept and managed by browser on user end-system
4) a cookie back-end database at the web site

cookies can be used to create user sessions on top of stateless HTTP

WEB CACHING

→ satisfies client request without involving origin server
→ keeps copies of recently requested objects in storage of proxy server
→ faster, uses less bandwidth on access links → don't have to install new infrastructure
→ typically purchased & installed by ISP (campus, company, residential)

must ensure cache contains an up-to-date copy of object
  ↳ can use Date-modified header in HTTP request
  ↳ can also use the conditional GET

makes a small request asking for object if-modified-since the date it was cached
  if not, sends small response w/o object (304 Not Modified)

FTP File transfer

↳ basically obsolete, very insecure (better to use HTTPS or FTPS)
↳ user provides hostname to establish TCP connection with FTP server
↳ then user provides password & username

OUT-OF-BAND

→ two different parallel TCP connections, one for data, one for control

Control band
- used to send username / password / commands (in plain-text)
- open for entirety of communication session

data band
- used to transmit files

- new data connection opened & closed for each file transfered

STATEFUL
→ unlike HTTP, maintains a state about the user
→ associates a control connection with a specific user and keeps track of user's current directory
↳ constrains # of simultaneous connections

Some commands: USER, PASS, LIST, RETR, STOR
↳ upload
USER, PASS: send server credentials
LIST: list files in directory
RETR: download

man, I wish I could do this in HTTP

Some Errors
331 Username ok, password required
125 Data connection already open; transfer starting
425 Can't open data connection
452 Error writing file

SMTP Electronic mail
→ asynchronous communication medium, can send/read when convenient for you
→ unlike HTTP, SMTP is a push protocol → connection initiated by machine that wants to send file
→ uses TCP, reliable but no promise it is immediate, port 25 unless added security
→ persistent connection, handshakes

Some commands HELO, MAIL FROM, RCPT TO, DATA, QUIT, CRLF.CRLF

Components
User-agents → outlook, phone app, browser
mail servers
SMTP
↳ mailbox contains incoming messages
↳ message queue contains outgoing messages

MAIL ACCESS PROTOCOL
→ used to transfer mail from recipient's mail server to recipient's user agent
POP3
↳ "download and delete" vs "download and keep"
IMAP
↳ allows mail folder heirarchy in IMAP server
↳ keeps everything in server, keeps user state

DNS

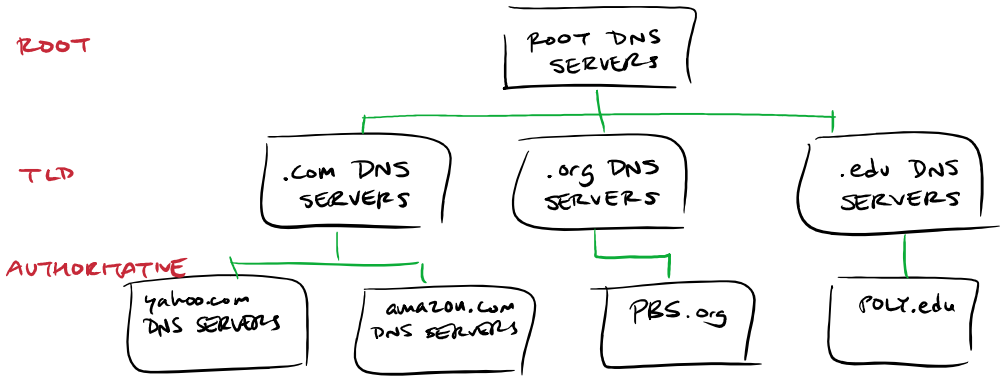
↳ keeps everything in server, keeps user state

DNS

→ translates hostname to IP address
→ adds additional delay
→ uses UDP
→ application-layer → not interacted with directly by user
→ provides host aliasing & mail server aliasing

4 bytes separated by decimal
0-255

DISTRIBUTED HIERARCHICAL DATABASES



Root DNS servers, 13 in the world
contacted by client that cannot resolve hostname
returns IP address of TLD

TLD server (top-level domain) responsible for host extensions
returns IP address of authoritative server

Authoritative DNS server, organization's own server, must provide publicly accessible DNS records

LOCAL DNS SERVER

→ each ISP has own local DNS server
→ keeps cache of recent mappings

- queries sent to local DNS server
- local requests from Root, TLD, and Authoritative servers for you (if iterative)
- local request from root, which requests from TLD, which requests from Authoritative (if recursive)

P2P APPLICATIONS

ex. BitTorrent

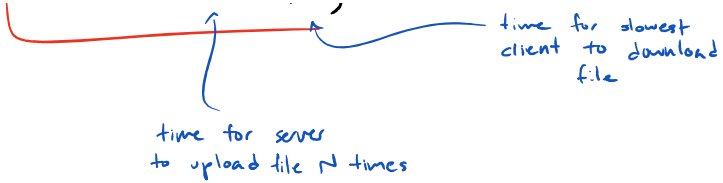
→ self-scalability

Distribution Time of Client-Server Model

↳ time for server to transmit one file of size F to N peers

$$D_{cs} \geq \max\left\{\frac{NF}{u_s}, \frac{F}{d_{min}}\right\}$$

time for slowest client to download file



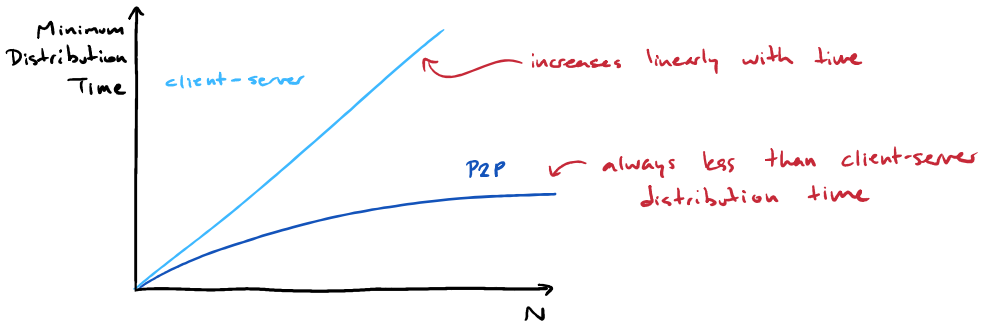
Distribution Time for P2P Model

$$D_{P2P} \geq \max\left\{\frac{F}{u_s}, \frac{F}{d_{min}}, \frac{NF}{u_s + \sum_{i=1}^{N} u_i}\right\}$$

time for server to upload file once

time for slowest client to download

all files over total upload capacity of whole system



BitTorrent

- → file divided into 256kb chunks
- → tracker monitors what peers have torrent
- → the reciever ask neighboring peers to list chunks they have and request chunks it doesn't have
  - ↳ using rarest first technique, requests least common chunks first so they get distributed most quickly in order to equalize # chunks in each torrent
- → incentivized request responses "tit-for-tat"
  - → priority given to peers that send to you at highest rate, "unchoked"
  - → randomly select a peer to "optimistically unchoke"
    - ↳ helps find roughly equal trading partners

Distributed Hash Table (DHT)

→ a distributed P2P database
→ key-value pairs for peers to query to find torrents