Chapter 2 Application Layer

Monday, March 1, 2021 9:01 PM

2.1 Principles of Network Applications

- Software run on multiple end systems
- no need to write software that runs on network core devices (routers, switches, work at network layer and below)
- confined application software to end systems

2.1.1 Network Application Architectures

- Client-server architecture or Peer to peer P2P architecture
- **Client server**
  - always on host - server, services requests from many other hosts - clients
    - send requested object to client host
    - clients do not directly communicate with each other
  - server has fixed IP address, client can contact server by sending packet to IP address
  - ex. Web, FTP, Telnet, e-mail
  - data center has large number of hosts - powerful virtual server
  - service providers pay recurring interconnection and bandwidth costs
- **P2P**
  - minimum/no reliance on dedicated servers in data centers
  - direct comm bw hosts - peers, not owned by service provider
  - hosts communicate without passing through server
  - ex. BitTorrent
  - self-scalability: each peer adds service capacity to the system by distributing files to other peers
  - cost effective: don't require significant server infrastructure and bandwidth
  - challenges security, performance, reliability - decentralized

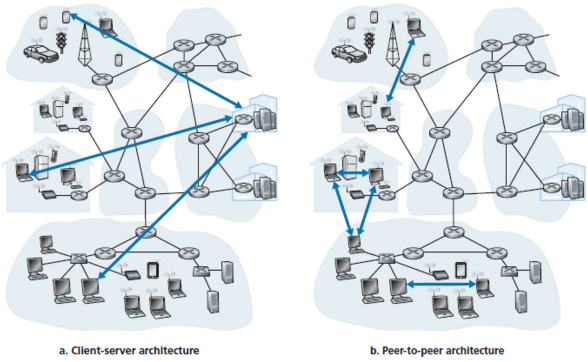


Figure 2.2 ♦ (a) Client-server architecture; (b) P2P architecture

2.1.2 Processes Communicating

- process: program running within end system, can communicate with each other with interprocess comm on same end system
- different end system processes exchange messages across computer network
- **Client Server Processes**
  - A process can be both client and server - P2P
  - Web: browser process (client) contacts web server process (server)
  - P2P: peer A (client) asks peer B(server) to send file
- **Interface Between Process and Computer Network**
  - process sends messages into and receives messages from software socket
  - transportation infrastructure on other side of socket to transport message to socket of destination process
  - message passes through process door

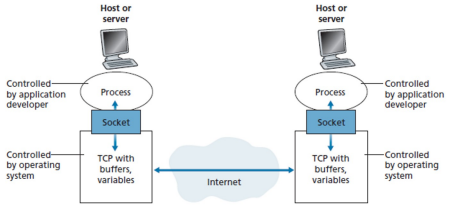


Figure 2.3 ♦ Application processes, sockets, and underlying transport protocol

- **Addressing Processes**
  - to identify receivign process, need to specify:

        - address of host: IP Address
        - identifier that specifies receiving process in destination host: receiving socket running in host - destination port number

2.1.3 Transport Services Available to Applications

- application sending pushes messages through socket
- transport layer protocol gets messages to socket of receiving process
- **Services offered by transport layer protocol:**
    - **Reliable Data Transfer**
        - guarantee that data sent is delivered to other end of app
        - sending process pass data into socket, data will arrive without errors
        - if not reliable transfer, some data may never arrive - acceptable for multimedia apps (glitch in audio/video)
    - **Throughput**
        - communication session bw 2 processes along network path, rate at which sending process can deliver bits to receiving process
        - guaranteed aailable throughput at specified rate
        - app can requrest guaranteed throughput of r bits/sec
        - if transport protocol cannot provide throughput, app need to encode at lower rate or give up
            - bandwidth sensitive applications ex. mail, file transfer, elastic apps
    - **Timing**
        - timing guarantee
        - every bit sender pumps into socket arrives at receiver socket no more than # msec later
        - ex. multiplayer games, virtual env, conference, tight timing constraints
    - **Security**
        - sending host transport protocol encrypt all data transmitted
        - receiving host, transport layer protocol decrypt data before delivering data to receiving process

2.1.4 Transport Services Provided by Internet

Application	Data Loss	Throughput	Time-Sensitive
File transfer/download	No loss	Elastic	No
E-mail	No loss	Elastic	No
Web documents	No loss	Elastic (few kbps)	No
Internet telephony/ Video conferencing	Loss-tolerant	Audio: few kbps–1 Mbps Video: 10 kbps–5 Mbps	Yes: 100s of msec
Streaming stored audio/video	Loss-tolerant	Same as above	Yes: few seconds
Interactive games	Loss-tolerant	Few kbps–10 kbps	Yes: 100s of msec
Smartphone messaging	No loss	Elastic	Yes and no

- **TCP**
    - reliable data transfer
    - Connection oriented
        - Handshake procedure: client and server exchange transport layer control info before application level messages flow
        - TCP connection now exists between sockets of two processes
        - processes can send messages to eachother at same time
        - connection is torn down when done
    - Reliable data transfer service
        - data is transferred with no missing or duplicate bytes
        - congestion control
        - limit each TCP connection to share of network bandwidth
    - TLS provide security services for encryption
- **UDP**
    - unreliable data transfer service, no guarantee that message reaches receiving process
    - messages may arrive out of order
    - no congestion control mechanism
    - can pump data into the layer at any rate
- **Services Not Provided by Internet Transport Protocols**
    - Internet can provide service to time sensitive applications but no timing and throughput guarantee

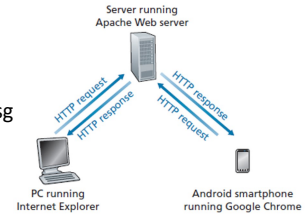
Application	Application-Layer Protocol	Underlying Transport Protocol
Electronic mail	SMTP [RFC 5321]	TCP
Remote terminal access	Telnet [RFC 854]	TCP
Web	HTTP 1.1 [RFC 7230]	TCP
File transfer	FTP [RFC 959]	TCP
Streaming multimedia	HTTP (e.g., YouTube), DASH	TCP
Internet telephony	SIP [RFC 3261], RTP [RFC 3550], or proprietary (e.g., Skype)	UDP or TCP

2.1.5 Application Layer Protocols

- Application layer protocol defines how app processes pass messages to eachoter
  - type of messages exchagned ex. request messages and response messages
  - syntax of various message types, fields
  - semantics of fields, meaning of info
  - rules for determining when and how process sends/responds msg
- some specified in RFCs public domain ex. HTTP

2.2 Web and HTTP

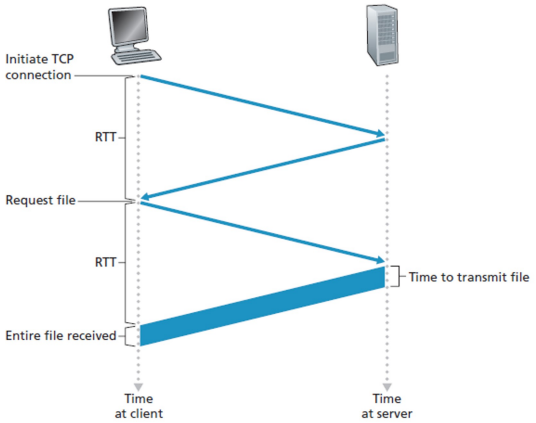
2.1.1 HTTP



- HyperText Transfer Protocol, defined in RFC
- client and server program executing on different end systems, exchange HTTP msg
- Web page consists of objects, ex. images, HTML file
- URL has hostname and houses object and object path name
- Uses TCP as transport protocol
  - TCP connection with server, socket interface
  - client sends HTTP request message into socket, recieves HTTP response
- Stateless Protocol: does not store state info about client
  - if client asks for same object twice, server resends object

2.2.2 Non-Persistent and Persistent Connections

- back to back requests, sent over separate TCP connections or same
- **HTTP and Non-persistent Connections**
  - ex. http://www.someSchool.edu/someDepartment/home.index
  - HTTP client TCP connection to server with socket
  - client sends request message to server with path name /someDepartment/home .index
  - server receives request message, retrieves object, ecapsulates in HTTP response, sends in socket
  - HTTP server tells TCP to close
  - HTTP client receives response, TCP connection closes, client extracts file from response,ecamines HTML file, finds references to 10 JPEG objects
  - repeat for each JPEG objects
  - Total response time: 2 RTTs + transmission time at server of HTML file
  - Cons:
    - new connection established and maintained for each requested object
    - TCP buffers allocated and TCP vars kept in client and server, burden on Web server
    - each object delivery delay of 2 RTTs - establish TCP and request/receive object

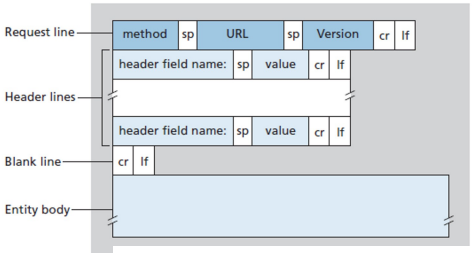


- **HTTP with Persistent Conections**
  - HTTP/1.1 server leaves TCP connection open after sending response
  - entier web page sent over single TCP connection
  - requests back to back without waiting for replies to pending requests (pipelining)
  - server sends objects back to back

2.2.3 HTTP Message Format

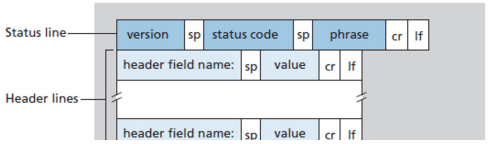
- **HTTP Request Message**

```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
Connection: close
User-agent: Mozilla/5.0
Accept-language: fr
```



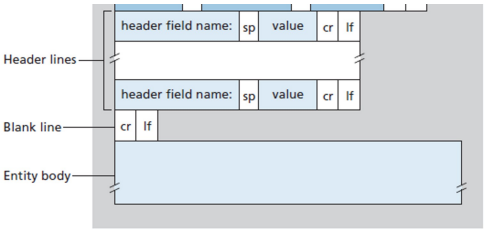
- **HTTP Response Message**

```
HTTP/1.1 200 OK
Connection: close
Date: Tue, 18 Aug 2015 15:44:04 GMT
Server: Apache/2.2.3 (CentOS)
Last-Modified: Tue, 18 Aug 2015 15:11:03 GMT
Content-Length: 6821
```



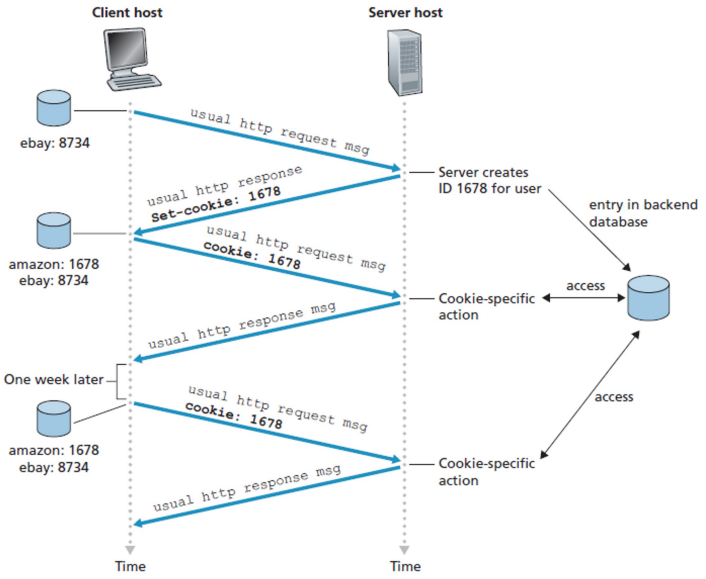
```
Date: Tue, 18 Aug 2015 15:44:04 GMT
Server: Apache/2.2.3 (CentOS)
Last-Modified: Tue, 18 Aug 2015 15:11:03 GMT
Content-Length: 6821
Content-Type: text/html

(data data data data data ...)
```



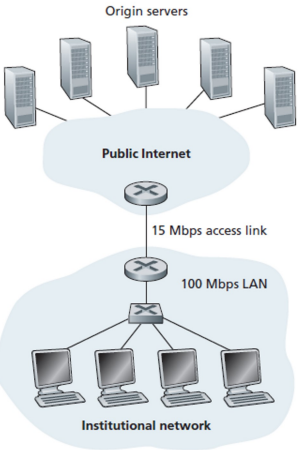
2.2.4 User-Server Interaction: Cookies

- HTTP server stateless
- Website wants to identify users, user cookies, allow sites to keep track of users
  - cookie header line in HTTP response msg
  - cookie header line in HTTP request msg
  - cookie file kept on users end system managed by user browser
  - back end database at web site
- Set-Cookie: # header contains id number HTTP response and request



2.2.5 Web Caching

- proxy server, satisfies HTTP requests on behalf of origin Web server
- own disk storage, keeps copies of recently requested obj
- When browser request object
  - browser establish TCP connection, sends HTTP request for obj in web cache
  - if copy of obj stored, cache returns obj to client browser
  - else cache opens TCP connection to server, sends HTTP request for obj into cache to server TCP, server sends obj to cache
  - cache receives obj, stores copy, sends copy to client browser
- cache is server and client at same time
- purchased and installed by ISP
- reduce response time for client request
- reduce traffic on institution access link to internet, no need to upgrade bandwidth as quickly
- reduce web traffic as a whole, improve performance for apps



Key:
Cookie file

a) The average time required to send an object over the access link Δ can be determined by:

$$\Delta = \frac{750{,}000 \text{ bits}}{15000000 \text{ bits/sec}} = 0.05 \text{ sec}$$

The traffic intensity on the link is given by $\Delta\beta = 0.05\ NK$. Therefore, the average access delay is $\frac{0.05}{1-0.05NK}$. The total average response time is $\left(\frac{0.05}{1-0.05NK} + 3\right)$ sec.

b) When a cache is installed, the traffic intensity on the link is given by $\Delta\beta = 0.05\ (1-p)NK$. Therefore, the average access delay is $\frac{0.05}{1-0.05(1-p)NK}$.

Note that, the probability that a request can be satisfied by the cache is p, which is the hit rate. Hence, the average response time when a cache is installed can be determined by:

$$p \cdot 0 + (1-p) \cdot \left(3 + \frac{0.05}{1-0.05(1-p)NK}\right) = (1-p) \cdot \left(3 + \frac{0.05}{1-0.05(1-p)NK}\right).$$

- **Conditional GET**
  - copy of obj in cache may be stale, obj in server may have been modified
  - HTTP allows cache to verify that obj are up to date
  - If-Modified-Since: header line

```
GET /fruit/kiwi.gif HTTP/1.1
Host: www.exotiquecuisine.com
If-modified-since: Wed, 9 Sep 2015 09:23:24
```

  - telling server to send obj only if obj modified since specified date
  - sends response msg without requested obj, Not modified

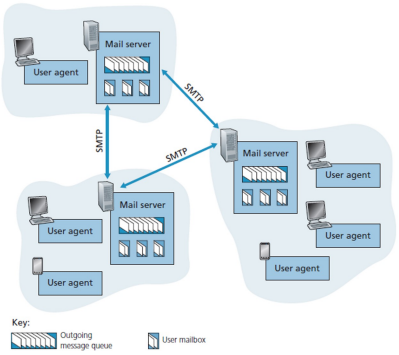
2.2.6 HTTP/2

- goal to reduce latency by enabling request and response multiplexing over single TCP connection
- request prioritization and server push
- efficient compression of HTTP header fields
- reduce number of parallel TCP connections for single Web page, reduce number of sockets
- allow TCP congestion control to operate as intended
- need carefully designed mechanism to avoid HOL blocking
  - Head of Line HOL

    - ex. large video clip, many small obj below, low-medium speed bottleneck link
    - using single TCP connection, video clip takes long time to pass through, small obj delayed
    - HTTP/1.1 work around by opening multiple parallel TCP
- **HTTP/2 Framing**
    - HOL solution: break msg into small frames, req and res on same TCP
    - ex. 1 large video, 8 small obj
        - 9 requests, video 1000 frames same length, small obj 2 frames
        - frame interleaving, one frame from video, one frame from each small obj
    - ability to break down HTTP msg into independent frames, interleave them, then reassemble on other end
    - framing sublayer binary encode frames, more efficient to parse
- **Response Msg Prioritization and Server Pushing**
    - customize rel priority of req to optimize app perf
    - client prioritize res it is req, assign weight 1-256 to each msg, 256 high
    - server send first frames for high priority res, dependency stated
    - send multiple res for single client req, server can push additional obj to client in adddition to original res
        - HTML base page indicates obj needed to fully render, can send to client before receiving explicit req
- **HTTP/3**
    - QUIC new transport protocol over UDP
    - msg multiplexing/interleaving, per stream flow control, low latency connection

2.3 Electronic Mail in the Internet

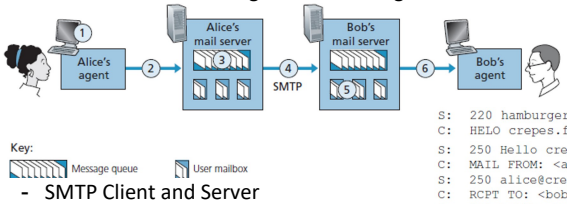
- User agents, mail servers, Simple Mail Transfer Protocol SMTP



- use reliable data transfer service TCP to transfer sender mail to receiver mail server

2.3.1 SMTP

- restricts body of all msg to 7 bit ASCII
- encoded and decoded back to binary
- reliable data transfer TCP
- Sending Email
    - A provide B email address, tells user agent to send msg
    - A user agent sends msg to mail server, msg queue
    - client of SMTP on A mail server opens TCP connection to SMTP server running on B mail server
    - SMTP handshaking, SMTP client sends A msg in TCP connection
    - B mail server SMTP receives msg, B mail server places msg in B mailbox
    - B invokes user agent to read msg



- SMTP Client and Server
    - ex. C: crepes.fr, S: hamburger.edu
    - TCP connection established
- SMTP users persistent connection
- user Telnet for direct dialogue
    - telnet serverName 25

```
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
```

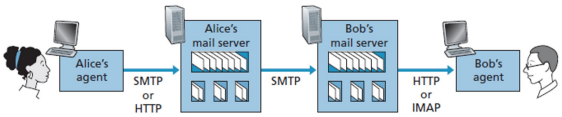
2.3.2 Mail Message Formats

- From, To, Subject header line

```
From: alice@crepes.fr
To: bob@hamburger.edu
Subject: Searching for the meaning of life.
```

2.3.3 Mail Access Protocols

- B host would have to always remain on and connected to internet to arrive new mail that could arrive, impractical
- user runs user agent on local host, but accesses mailbox stored on always on shared mail server, shared with other users



- Retrieve msg from mail server
  - using web based email, using HTTP to retreive, and SMTP to communicate with A server
  - using IMAP Internet Mail Access Protocol
    - manage folders maintained in B mail server

2.4 DNS Internet's Directory Service

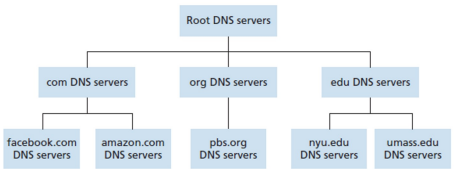
- hosts identified by hostname - mnemonic
- host also identified by IP addresses

2.4.1 Services Provided by DNS

- domain name system DNs
  - distributed db implemented in hierarchy of DNS servers
  - application layer protocol allowing hosts to query distributed db
- Obtain IP address of hostname
  - user machine runs client side of DNS app
  - browser extracts hostname from URL and pass to client side of DNS
  - DNS client sends query containing hostname to DNS server
  - DNS client receives reply, includes IP address for hostname
  - browser initiates TCP connection to HTTP server process at port 80 of IP
- IP address often cached in DNS server, reduce DNS traffic and delay
- **Host aliasing**
  - host with complex hostname can have 1+ alias names - canonical hostname
- **Mail server aliasing**
  - email address simplified even if yahoo mail server has a canonical hostname of relay1.west-coast.yahoo.com
- **Load distribution**
  - DNS perform load distribution among replicated servers/webservers
  - busy sites are replicated over multiple servers with each running on diff end system and IP address
  - set of IP associated with one alias hostname
  - DNS db contains set of IP addresses, server responds with entire set of IP to query for a name
  - DNS rotation distributes traffic among replicated servers
  - used for email so that many mail servers can have same alias name

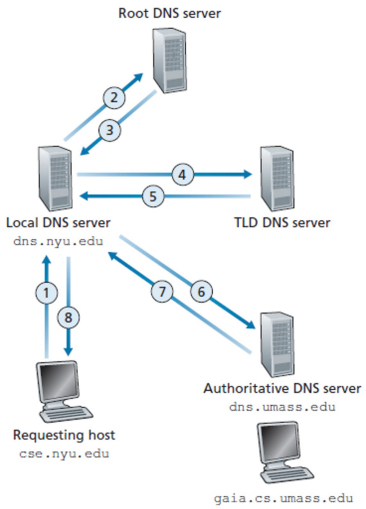
2.4.2 DNS Overview

- hostname to IP address translation
- Centralized DNS server design problems
  - single point of failure
  - traffic volume
  - distant centralized database
    - not close to all querying clients
  - Maintenance
  - doesn't scale
- **Distributed Hierarchical Database**
  - DNS has many servers around the world
  - mapping for all hosts in internet, destributed across DNS servers
  - 3 classes of DNS servers: root DNS, Top level domain TLD, authoritative DNS

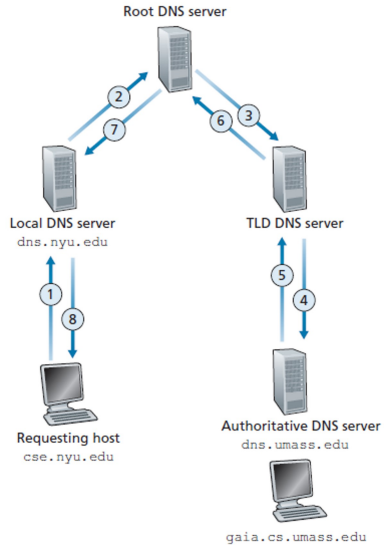


  - ex. finding IP for www.amazon.com

        - client contacts root servers, return IP address for TLD server for .com domain
        - client contacts TLD server, returns IP of authoritative server for amazon.com
        - client contacts authoritative server for amazon.com, returns IP for hostname www.amazon.com
    - **Root DNS servers**
        - more than 1000 instances over world
        - copies of 13 diff root servers managed by 12 organizations
        - provide IP addresses of TLD servers
    - **Top-level domain TLD servers**
        - com, org, net, edu, gov, country top lvl domains uk, fr, ca, jp
        - provide IP addresses for authoritative DNS servers
    - **Authoritative DNS servers**
        - DNS records that map hostnames to IP addresses
        - organization's authoritative DNS server houses authorative DNS records
        - can also pay to store records in another DNS server of service provider



- **Local DNS Server**
    - not part of hierarchy
    - each ISP has local DNS server
    - when host connects to ISP, provides host with IP address of 1+ local DNS servers
    - close to host, may be on same LAN
    - forwards host DNS queries into DNS server hierarchy
- TLD server does not always know authoritative DNS server for hostname, may only know of an intermediate DNS server leading to desired mapping
- **recursive query:** asks to obtain mapping on behalf of other server
- **iterative query:** all of replies are directly returned to client server
- **DNS Caching**
    - reduce delays and number of DNS messages
    - when DNS server receives DNS reply, can cache mapping in local memory
    - if other query asks for same hostname, DNS server can provide IP address even if not authoritative for hostname
    - can also cache IP of TLD servers, bypass root DNS servers on query chain



2.4.3 DNS Records and Messages

- The DNS servers store resource records RRs that provide hostname to IP mapping

```
(Name, Value, Type, TTL)
```

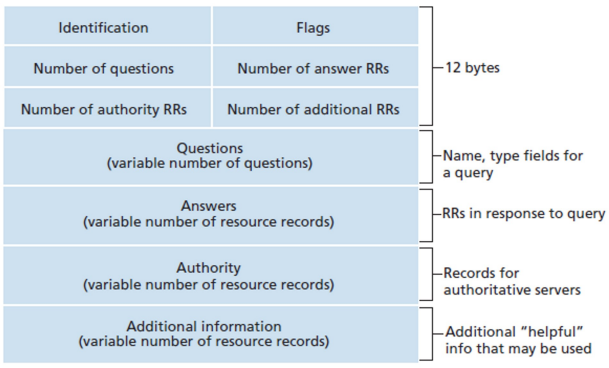
- **TTL**
    - time to live, when resource should be removed from cache
- **Type**
    - **A**
        - name = hostname, value = IP address for hostname

        `(relay1.bar.foo.com, 145.37.93.126, A)`
    - **NS**
        - name = domain, value = hostname of authoritative dNS

        `(foo.com, dns.foo.com, NS)`
    - **CNAME**
        - value = canonical hostname for alias hostname Name

        `(foo.com, relay1.bar.foo.com, CNAME)`
    - **MX**
        - Value = canonical name of mail server with alias hostname Name

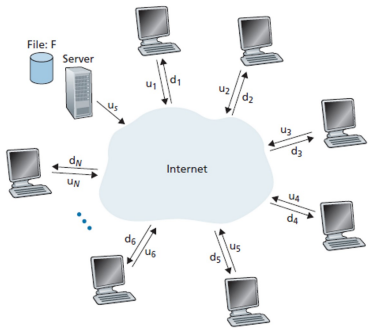
        `(foo.com, mail.bar.foo.com, MX)`
- **DNS Messages**
    - 12 bytes header section
    - 16 bit number identifying query, copied into reply msg to query
    - 1 bit flag indicating query/flag
    - 1 bit authoritative flag in reply msg when DNs server is authoritative
    - 1 bit recursion-desired flag set when DNS server perform recursion if no record
    - 1 bit recursion-available set in reply if DNS supports recursion
    - 4 number field in header for num of occurences in 4 types of data sections under header
    - question section: info about query, includes name field, ype field
    - answer section: reply from DNS server return RRs
    - authority section contains records of other authoritative servers
    - additional section contains other helpful records
    - use nslookup program to send DNS query from host to DNS



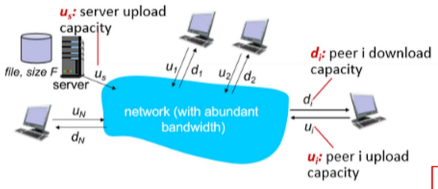
- **Inserting Records into DNS Database**
    - register domain name at registrar
    - provide names and IP addresses of primary and secondary authoritative DNS servers
    - Type A resource record for web server

- register domain name at registrar
- provide names and IP addresses of primary and secondary authoritative DNS servers
- Type A resource record for web server

2.5 Peer-to-Peer File Distribution



- little to no reliance on alwayrs on infrastructure servers
- pairs of connected hosts, peers communicate directly
- peers exchange IP addresses, request service from others
- peers may come and go
- ex. BitTorrent
- **Scalability of P2P Architecture**
  - self scalability: new peers bring new service capacity and service demands
- **File Distribution: Client-Server vs P2P**
  - Distributing file sie F from one server to N peers



  - **Client-Server:**
    - must send N file copies N*F/u_s
    - min client download time F/d_min
      - d_min client download rate

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

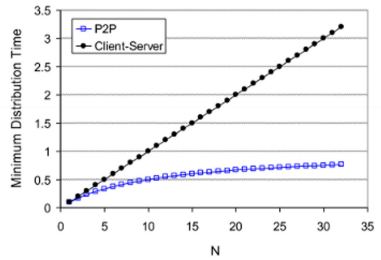
increases linearly in N

  - **P2P:**
    - time to send one copy F/u_s
    - client must download file copy F/d_min
    - clients aggregate must download N*F bits

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

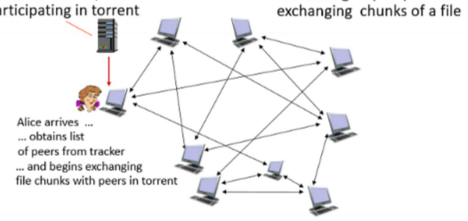
increases linearly in N ...
... but so does this, as each peer brings service capacity

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



tracker: tracks peers participating in torrent

torrent: group of peers exchanging chunks of a file



- **Requesting, Sending File Chunks**
  - peers have different subsets of chunks
  - host asks each peer for list of chunks they have and requests rarest first
  - **Rarest first**
    - send and request chunks
    - determine among the missing chunks, request from neighbours the rarest chunks
    - redistributed quicker to equalize number of copies
  - host gives priority to peers sending her chunks at highest rate
  - other peers are choked, do not receive chunks from her
  - randomly select another peer, unchoke and send chunks

2.6 Video Streaming and Content Distribution Networks

- Distributed application level infrastructure

2.6.1 Internet Video

- sequence of images 24-30 images/second, uncompressed digitally encoded image consists of array of pixels, each encoded into number of bits to represent luminance and color
- important performance measure for streaming is avg end-to-end throughput
- network must provide an avg throughput to streaming application that is as large as bit rate of compressed video

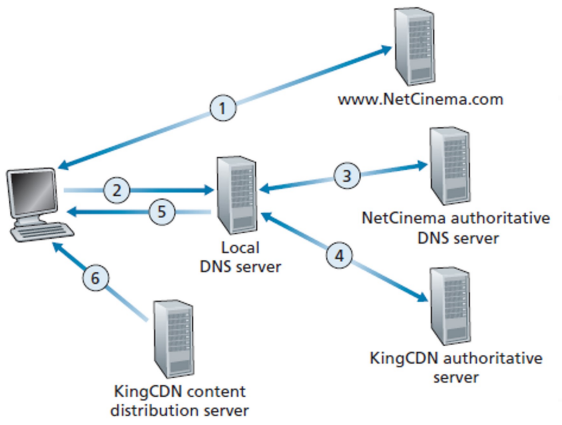
2.6.2 HTTP Streaming and DASH

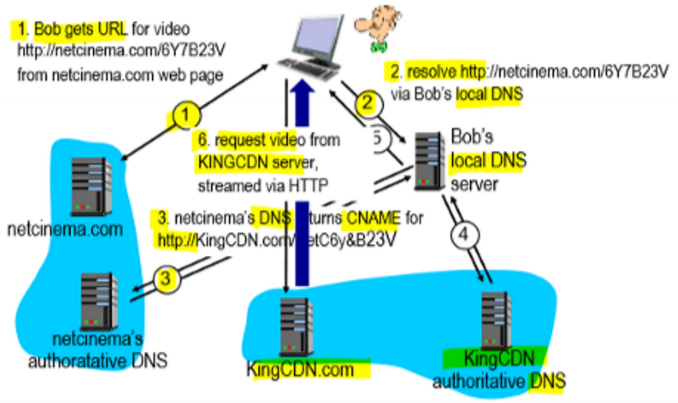
- video stored at HTTP server as file with URL
- server sends video file within HTTP response message as quickly as underlying network protocols and traffic conditions allow
- bytes collected in client app buffer, once number of bytes in buffer exceeds threshold, client app begins playback, grabbing video frames from buffer

- Con: all clients receive same encoding of video despite variations in bandwidth
- **Dynamic Adaptive Streaming over HTTP (DASH)**
    - video encoded into several different versions w diff bit rate, diff quality level
    - client requests chunks of video of a few sec in length, when bandwidth is high, client selects chunks from high rate version
    - allows clients with different internet access rates to stream at different encoding reates
    - lots of buffered video, receive high bitrate chunks

2.6.3 Content Distribution Networks

- How to stream content to simultaneous users
- 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
- **bring home:** bring home smaller number of larger clusters
- stores copies of content at CDN nodes
- subscriber requests content from CDN, retrieves content or choose different copy if congested
- DNS redirects user request to CDN server



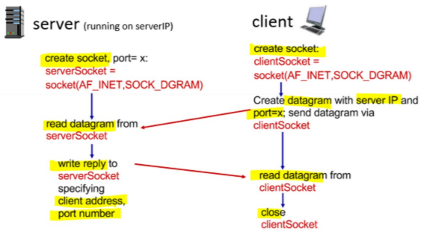


2.7 Socket Programming: Creating Network Applications

- socket: door bw application process and end-to-end transport protocol
- UDP: unreliable datagram
- TCP: reliable, byte stream oriented

2.7.1 Socket Programming UDP

- no handshaking or connection bw client and server
- sender attaches IP address and port # to each packet
- receiver extracts sender IP address and port # of packet
- transmitted data may be lost or out of order

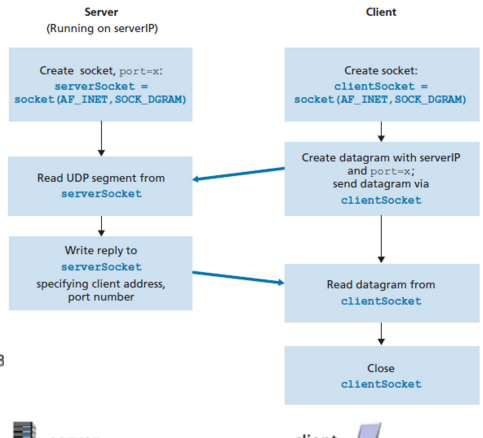


Python UDPClient

```
from socket import *
serverName = 'hostname'
serverPort = 12000
clientSocket = socket(AF_INET,
                      SOCK_DGRAM)
message = raw_input('Input lowercase sentence:')
clientSocket.sendto(message.encode(),
                    (serverName, serverPort))
modifiedMessage, serverAddress =
                    clientSocket.recvfrom(2048)
print modifiedMessage.decode()
clientSocket.close()
```

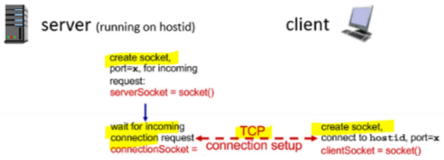
Python UDPServer

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET, SOCK_DGRAM)
serverSocket.bind(('', serverPort))
print ("The server is ready to receive")
while True:
    message, clientAddress = serverSocket.recvfrom(2048
    modifiedMessage = message.decode().upper()
    serverSocket.sendto(modifiedMessage.encode(),
                        clientAddress)
```



2.7.2 Socket Programming TCP

- client must contact server
- server process is running and created socket
- client creates TCP socket, specifying IP and port number of server
- client creates TCP connection



- client must contact server
- server process is running and created socket
- client creates TCP socket, specifying IP and port number of server
- client creates TCP connection
- server TCP creates new socket to communicate with client
- can have multiple sockets and clients

Python TCPClient

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

create TCP socket for server, remote port 12000

No need to attach server name, port

Python TCPServer

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

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

create TCP welcoming socket

server begins listening for incoming TCP requests

loop forever

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

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

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