COMPUTER NETWORKS

UNIT-2

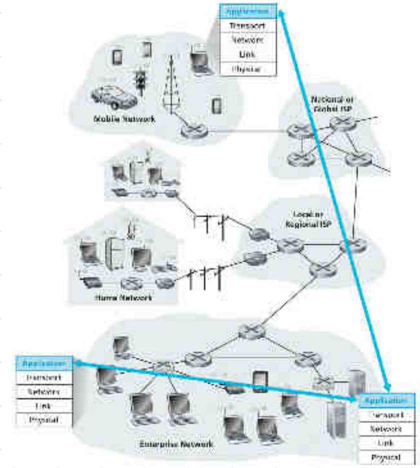
application layer

feedback/corrections: vibha@pesu.pes.edu

VIBHA MASTI

Application Layer

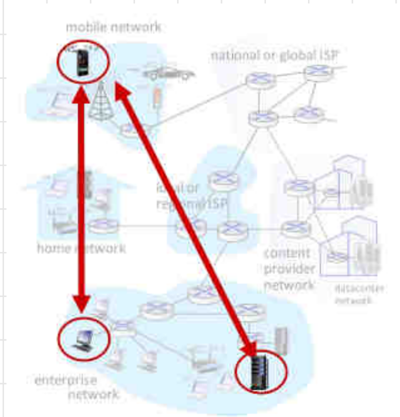
- network applications
- social network, VoIP, games, streaming, P2P file sharing, text, web, real-time conferencing
- client-server, peer-to-peer
- network core devices: do not run user apps



Client-Server Paradigm

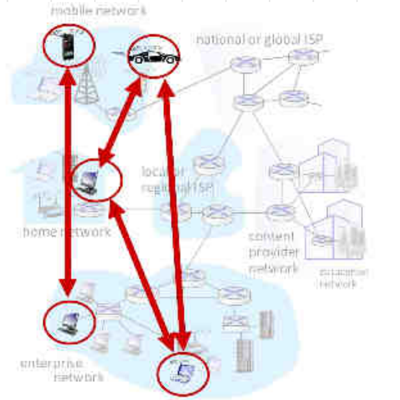
⟶ Server
- always on
- permanent IP address

⟶ Client
- static/dynamic IP address
- contact servers, not clients



- HTTP, IMAP, FTP

Peer-to-Peer Architecture



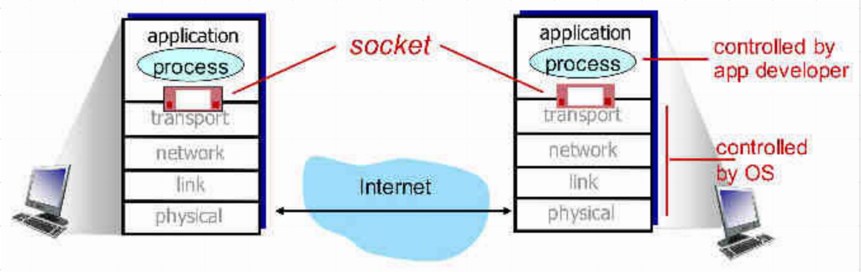
- no always-on server
- end systems communicate (arbitrary)
- peers act as both clients and servers
- self-scalability: new users bring capacity and demands
- peers intermittently connected and change IP addresses
- peers request service from others peers and provide service for other peers
- eg: Skype, P2P file sharing

Process Communicating

- process: program running on a host
- two processes on same host: inter-process communication — managed by OS
- different hosts: send/receive messages across network
- client process: process that initiates communication
  server process: process waiting to be contacted

Sockets

- processes send / receive messages from sockets (to/from transport layer)
- analogous to doors
- one socket on each side (client, server)



- interface between application layer and transport layer

Addressing Processes

- process sending message has an identifier (32-bit IP address)
- many processes on same host; host IP address not sufficient
- each process has a port number
- HTTP server: 80
  mail server: 25

Application Layer Protocol

- type of message: request, response
- message syntax: message fields
- message semantics: meaning of field information
- rules: how processes send/receive messages
- open protocols: defined in RFCs (Request for Comments), protocol definition
  - eg: HTTP, SMTP
- proprietary protocols: skype

Transport Services

- data integrity: 100% reliability on transactions, files; videos, audio do not require (checksum)
- timing: delay for video chat, conferencing, gaming should be low
- throughput: real-time bit flow; multimedia require certain min to be effective, elastic use available throughput
- security: encryption

Transport Service Requirement

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

TCP Service Transmission Control Protocol

- flow control: agreement between sender and receiver such that receiver buffer is not overwhelmed (handshake)
- congestion control: throttle sender in overwhelmed network
- reliable transport (send / receive)
- connection setup between client and server

UDP Service User Datagram Protocol

- unreliable data transfer
- no flow control, congestion control, timing, throughput guarantee
- no connection setup

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 (Skype)	TCP or UDP
streaming audio/video	DASH	TCP
interactive games	WOW, FPS (proprietary)	UDP or TCP

the WEB

- Tim Berners-Lee : CERN – invented www
- collection of resources interconnected via hyperlinks
- webpages consist of objects (stored on servers)
- objects: image, applet, audio, HTML etc
- objects: base HTML + images, audio etc
- URL: uniform resource locator } same
  URI: uniform resource identifier }
- www.somesite.com/somepage/pic.gif
  (www.somesite.com = host name; somepage/pic.gif = path name)

HTTP PROTOCOL

- Application layer protocol for web
- client/server model
- HyperText Transfer Protocol

Defined in RFC 1945; RFC 2616

HTTP uses TCP

- client creates socket - initiates TCP connection to server (port 80 - default)
- connects app and transport layers on client side
- server accepts TCP connection from client
- server creates socket as doorway that connects app and transport layers on server side
- HTTP request-response messages exchanged between browser and server
- TCP connection closed

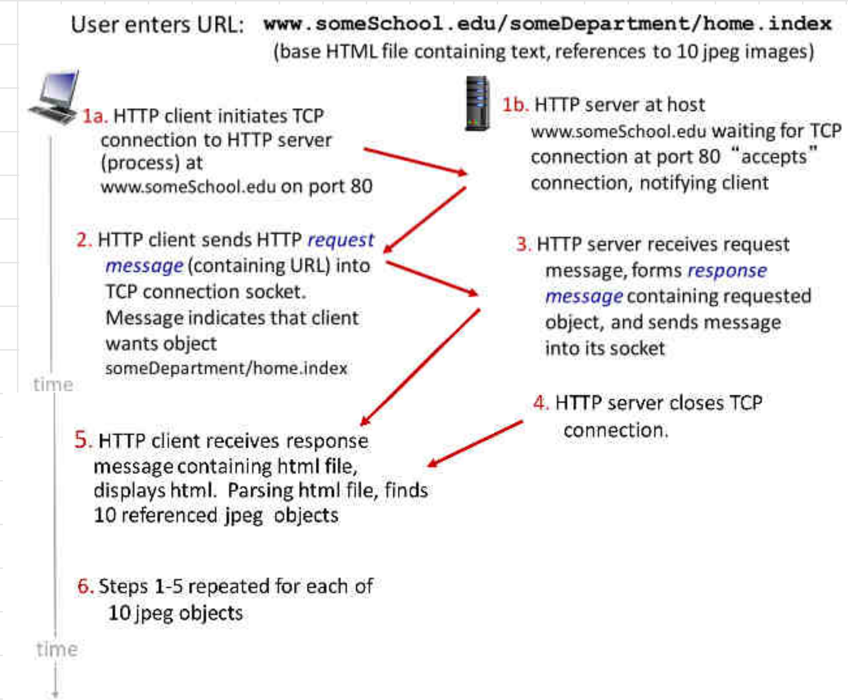
http is stateless

- server does not retain information about past client requests
- state maintenance is complete

non-Persistent & Persistent

NON-PERSISTENT HTTP	PERSISTENT HTTP
1. TCP connection opened	1. TCP connection opened
2. One object per TCP conn	2. Multiple objects per conn
3. TCP conn closed	3. TCP conn closed

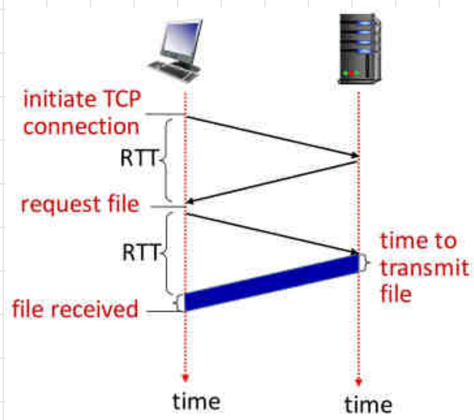
— NON PERSISTENT



RTT: Response Time

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

HTTP Response Time (per object)



- one RTT to initiate TCP conn
- one RTT for HTTP req and first few bytes of res to return
- object/file transmission time

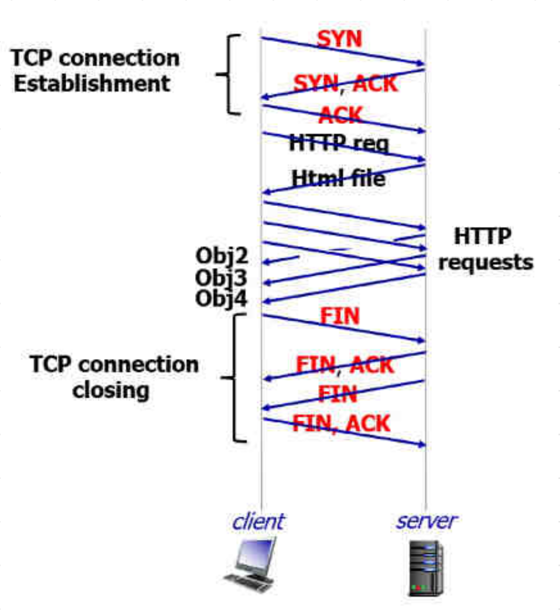
response time → **2 RTT + file transmission time**

Issues

- 2 RTTs per object
- OS overhead for each TCP conn
- solution: parallelism

PERSISTENT

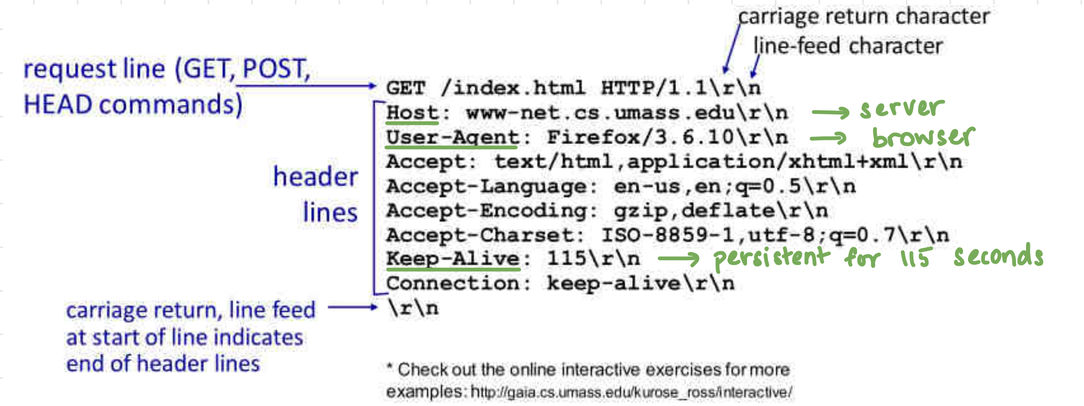
- HTTP 1.1
- server leaves connection open after sending response
- only one RTT



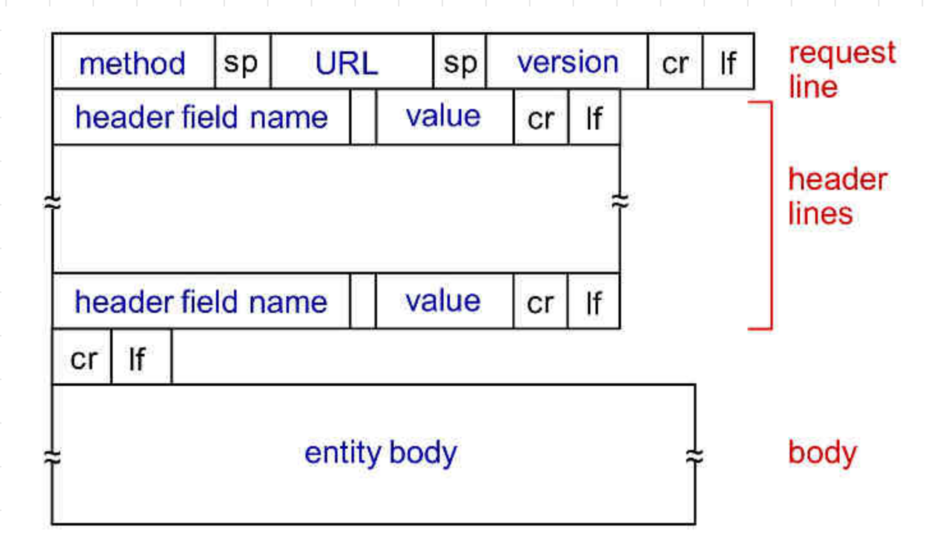
HTTP

HTTP Request Message

observe on wireshark



General Format



HTTP specifications [RFC 1945; RFC 2616; RFC 7540]

POST

- form input
- body of request: user input

GET

- user data in URL field (? query)

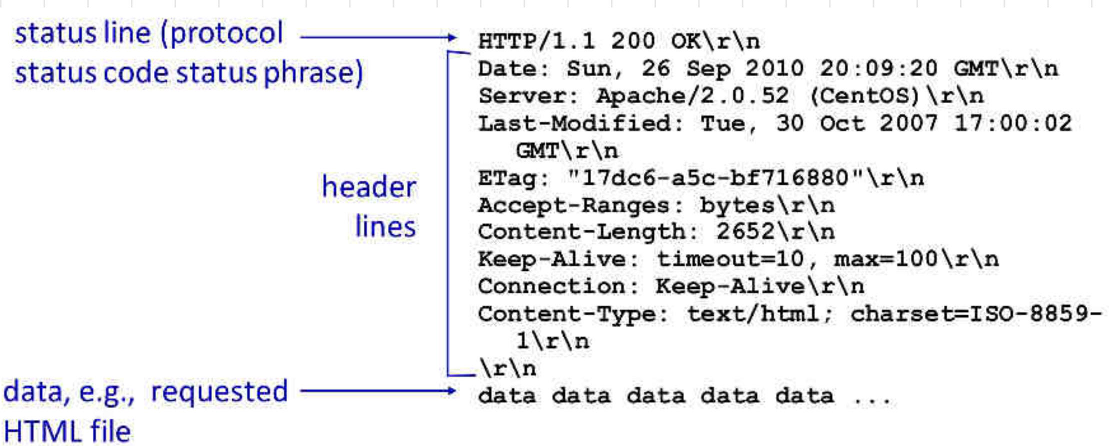
HEAD

- request for only headers
- similar to get

PUT

- uploads new file to server
- replaces existing file on server
- content in body of req

HTTP Response Message



statuy 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)

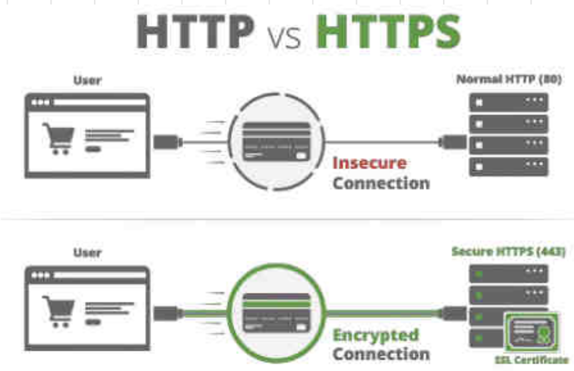
400 Bad Request
- request msg not understood by server

404 Not Found
- requested document not found on this server

505 HTTP Version Not Supported

HTTPS

- HTTPS - secure
- encrypted communication between browser and server
- uses TLS - transport layer security - based on SSL - Secure Socket Layer - to encrypt normal http req-res



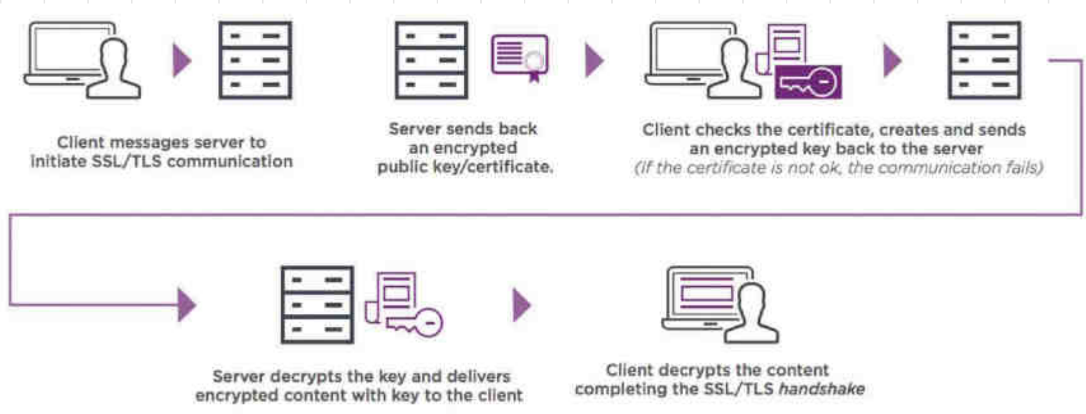
HTTP vs HTTPS

- HTTP + TLS → encrypted
- port : 443 for data communication (not 80)
- public-private key cryptography
- SSL certificate: web server's digital certificate issued by third party CA
- SSL or TLS → SSL 4.0 is TLS 1.0

working of SSL

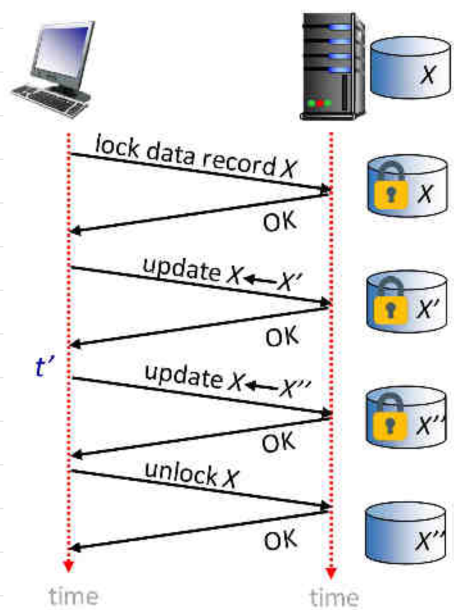
- Step 1: Browser requests secure pages (HTTPS) from a server.
- Step 2: Server sends its public key with its SSL certificate (digitally signed by a third party – CA).
- Step 3: On receipt of certificate, browser verifies issuer's digital signature. (green padlock key)
- Step 4: Browser creates a symmetric key (shared key), keeps one and gives a copy to server. Encrypts it using server's public key.
- Step 5: On receipt of encrypted secret key, server decrypts it using its private key and gets browser's secret key.

asymmetric encryption



COOKIES

- Piece of data from specific website
- Stored on user's computer
- keep track of users
- HTTP / HTTPS is stateless
- cookies prevent incomplete transactions from occuring
- maintains state
- sent by server to client in response message



- copy of cookie in db server; cookie stored in browser history

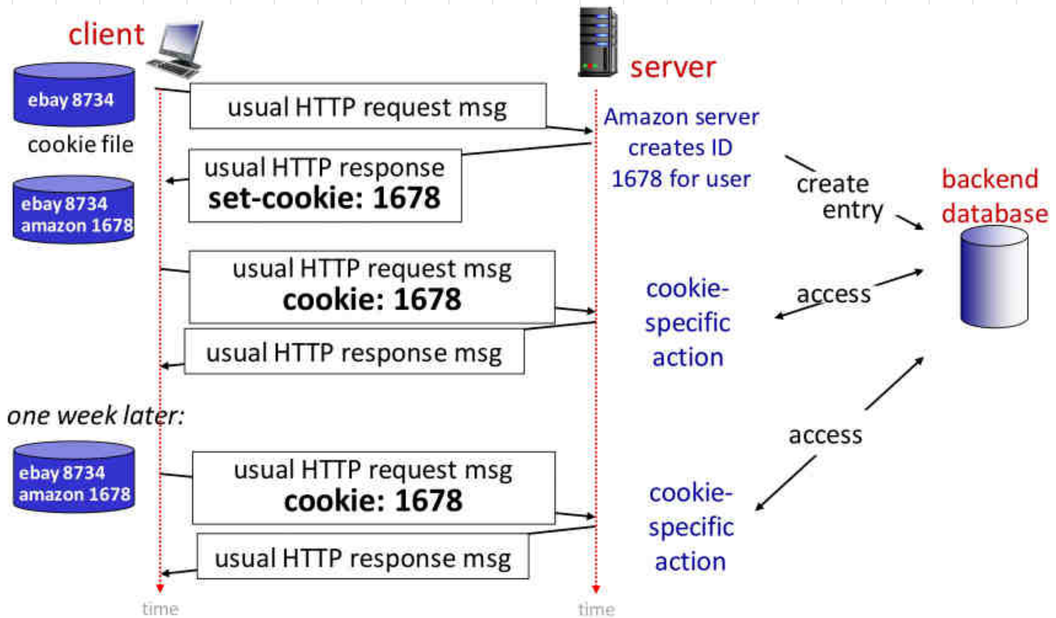
four components

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) Backend database at website

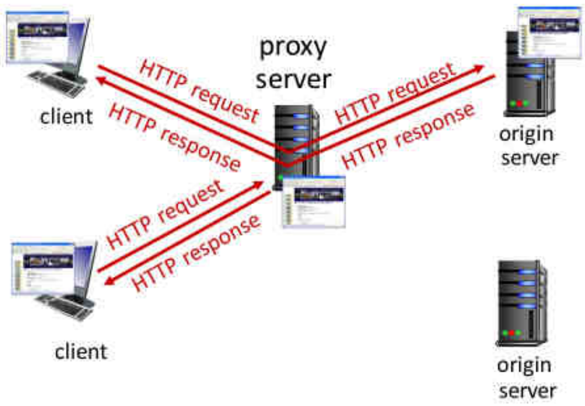


Uses of cookies

- tracking user's browsing history
- remembering login details
- shopping carts
- recommendations

WEB CACHING

- Proxy servers
- User configures browser to point to a web cache
- Browser sends all http messages to cache
- If object in cache: cache returns object to client
- Else cache requests object from origin server, receives object and returns object to client

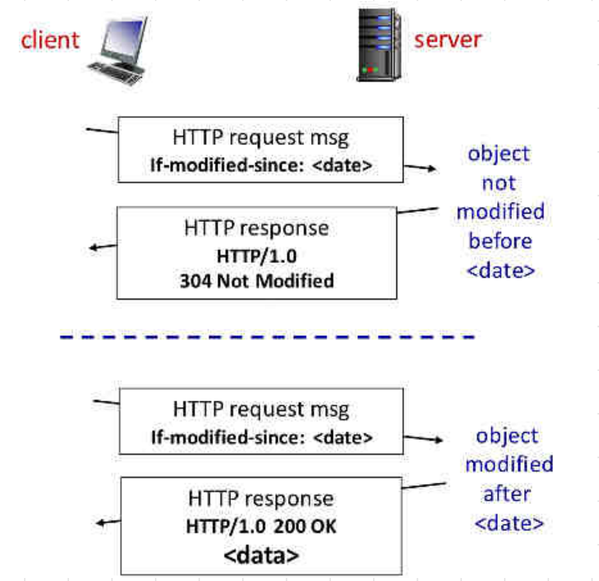


- Proxy server acts as both client and server

- Cache typically installed by ISP
- Reduce response time for client request
- Reduce traffic on institution's access link
- Privacy: surf anonymously (IP address hidden)

CONDITIONAL GET

- Idea: not to send object if cache has up-to-date version
- Cache: specify date/time of cached copy in HTTP request
  if-modified-since: <date>
- Server: response contains no object if cached copy is up-to-date
  HTTP/1.0 304 Not Modified



DOMAIN NAME SYSTEM

- Domain name assigned to IP addresses
- Domain name resolution
- Distributed database implemented in hierarchy of many name servers (tree)
- Application layer protocol: hosts, name servers communicate to resolve names
- core internet function
- Runs over UDP: port 53

DNS Services

- hostname to IP address translation
- host aliasing – host machine names (canonical names) may not be as mnemonic

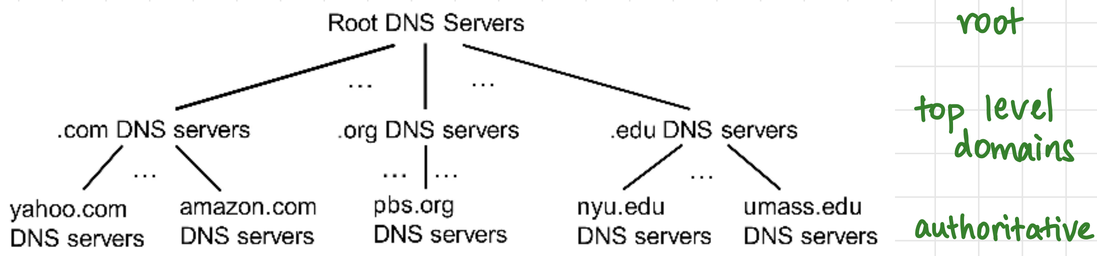
    www.abc.example.com ⟶ Canonical host name
    www.example.com ⟶ alias name

- load distribution : replicated web servers

DNS Not Centralised

- Single point of failure
- Traffic volume
- Distant centralised database
- Maintenance
- Scalability

DNS : Hierarchical Database

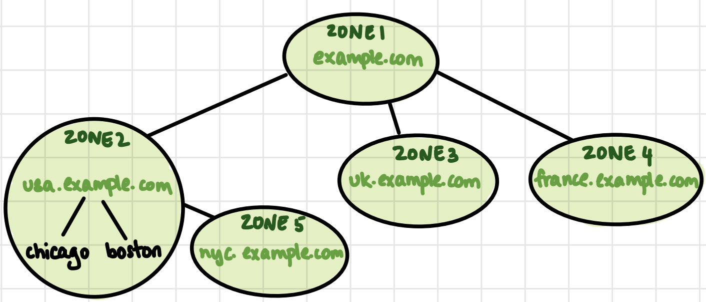


Client wants IP address for www.amazon.com; 1st approximation:

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

DNS ZONES

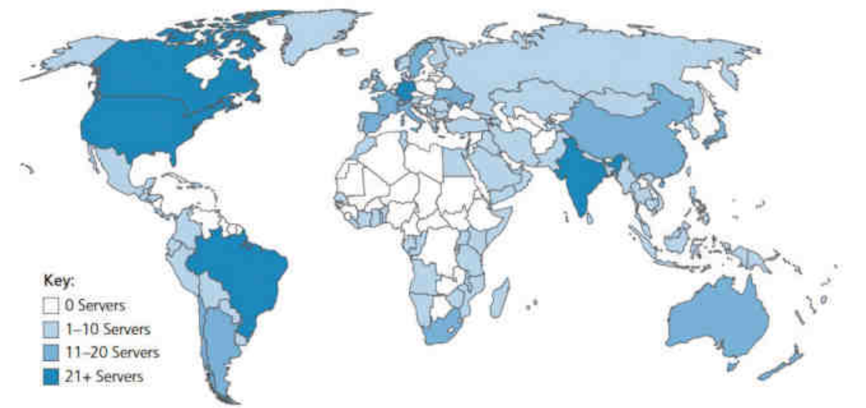
- Groups contiguous domains and subdomains on the domain tree
- Multiple DNS zones ; one for each country
- Zone keeps records of who the authority is for each subdomain



HIERARCHY

Root Name Servers

- ICANN : Internet Corporation for Assigned Names and Numbers manages root DNS domain
- 13 logical root name servers worldwide ; each server replicated many times (13 organisations)



- DNSSEC - security
- Essential for functioning of internet

Top-Level Domain (TLD) Servers

- .com, .net, .edu, .org, .aero, .jobs, .cn, .uk, .in, .fr etc (country domains as well)

Authoritative DNS servers

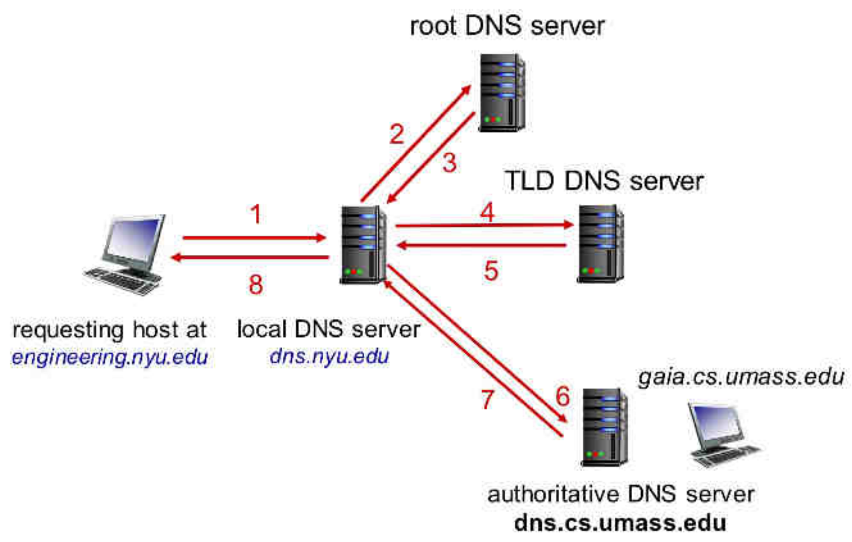
- organisation's own DNS servers
- maintained by organisation or service provider

Local DNS Name Server

- Not strictly belonging to hierarchy
- Each ISP has one (residential, university etc)
- When host makes DNS query, query sent to local DNS server (Airtel, BSNL etc)
- Has local cache of recent name-to-address pairs, but may be outdated
- Acts as a proxy server and forwards query to hierarchy

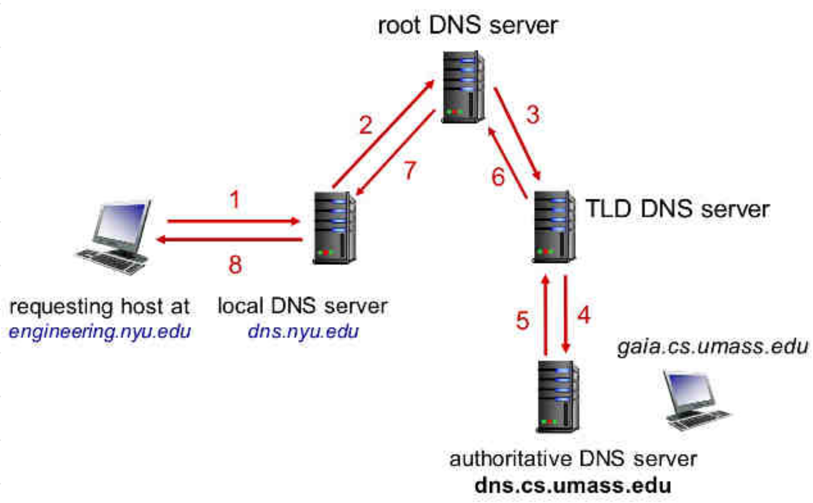
DNS Name Resolution

- Iterated query - contacted server replies with name of server to contact



- Recursive query - burden of name resolution on contacted name server

- Heavy load at upper levels of hierarchy



CACHING & UPDATING DNS RECORDS

- Two hosts query DNS server for same hostname, second query served cached mapping

- Cache entries timeout after some time (TTL - time to live)

- TLD servers typically cached in local name servers ; root name servers not visited often

- If name host changes IP address, may not be known internet-wide until TTLs expire

- Update / notify mechanisms proposed IETF standard RFC 2136

DNS records

- Distributed database storing resource records (RR)
- RR format: (name, value, type, ttl) — four tuple
  - ttl: when resource to be removed from cache

TYPES

1. type = A
   - name is hostname
   - value is IP address
   - standard hostname-to-IP address mapping
   - (relay1.bar.foo.com, 145.37.93.126, A)

2. type = CNAME
   - name is alias name for some canonical (real) name
   - www.ibm.com is really servereast.backup2.ibm.com
   - value is canonical name
   - (ibm.com, servereast.backup2.ibm.com, CNAME)

3. type = NS
   - name is domain (eg: foo.com)
   - value is hostname of authoritative name server that knows to obtain IP addresses for hosts in this domain
   - (foo.com, dns.foo.com, NS)

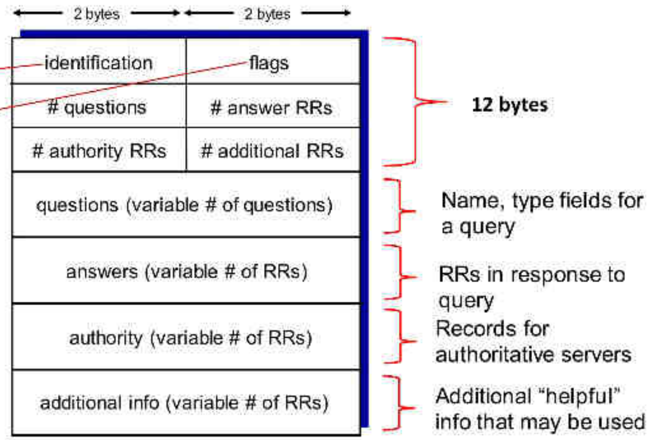
4. type = MX
   - value is canonical name of a mailserver associated with alias hostname name
   - (example.com, mail.example.com, MX)

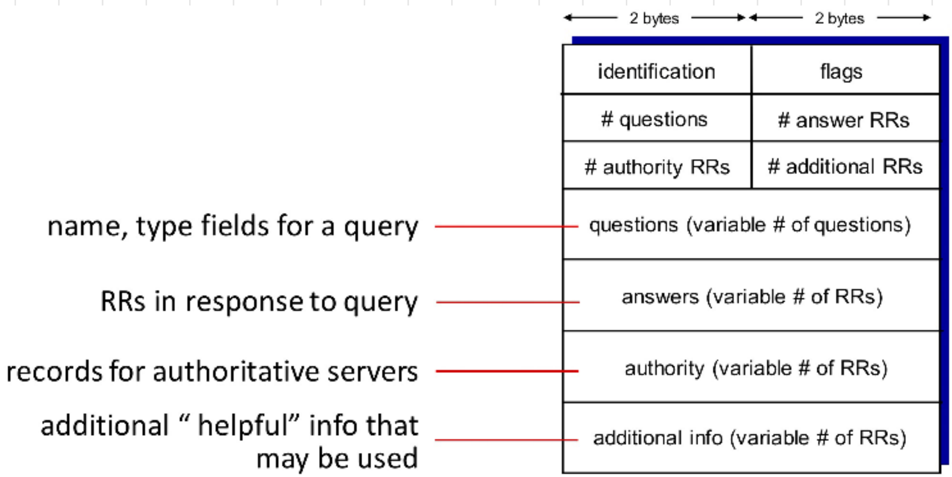
DNS Protocol Messages

- DNS query and reply messages (same format)

message header:

- identification: 16 bit # for query, reply to query uses same #
- flags:
  - query or reply (1-bit) — 0 / 1
  - recursion desired
  - recursion available
  - reply is authoritative





Type (A, MX, NS, CNAME)

Terminal

```
[→ ~ dig @a.root-servers.net www.example.net

; <<>> DiG 9.10.6 <<>> @a.root-servers.net www.example.net
; (2 servers found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 23449
;; flags: qr rd; QUERY: 1, ANSWER: 0, AUTHORITY: 13, ADDITIONAL: 27
;; WARNING: recursion requested but not available

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1472
;; QUESTION SECTION:
;www.example.net.               IN      A

;; AUTHORITY SECTION:
net.                    172800  IN      NS      a.gtld-servers.net.
net.                    172800  IN      NS      b.gtld-servers.net.
net.                    172800  IN      NS      c.gtld-servers.net.
net.                    172800  IN      NS      d.gtld-servers.net.
net.                    172800  IN      NS      e.gtld-servers.net.
net.                    172800  IN      NS      f.gtld-servers.net.
net.                    172800  IN      NS      g.gtld-servers.net.
net.                    172800  IN      NS      h.gtld-servers.net.
net.                    172800  IN      NS      i.gtld-servers.net.
net.                    172800  IN      NS      j.gtld-servers.net.
net.                    172800  IN      NS      k.gtld-servers.net.
net.                    172800  IN      NS      l.gtld-servers.net.
net.                    172800  IN      NS      m.gtld-servers.net.

;; ADDITIONAL SECTION:
a.gtld-servers.net.     172800  IN      A       192.5.6.30
b.gtld-servers.net.     172800  IN      A       192.33.14.30
c.gtld-servers.net.     172800  IN      A       192.26.92.30
d.gtld-servers.net.     172800  IN      A       192.31.80.30
e.gtld-servers.net.     172800  IN      A       192.12.94.30
f.gtld-servers.net.     172800  IN      A       192.35.51.30
g.gtld-servers.net.     172800  IN      A       192.42.93.30
h.gtld-servers.net.     172800  IN      A       192.54.112.30
i.gtld-servers.net.     172800  IN      A       192.43.172.30
j.gtld-servers.net.     172800  IN      A       192.48.79.30
k.gtld-servers.net.     172800  IN      A       192.52.178.30
l.gtld-servers.net.     172800  IN      A       192.41.162.30
m.gtld-servers.net.     172800  IN      A       192.55.83.30
a.gtld-servers.net.     172800  IN      AAAA    2001:503:a83e::2:30
b.gtld-servers.net.     172800  IN      AAAA    2001:503:231d::2:30
c.gtld-servers.net.     172800  IN      AAAA    2001:503:83eb::30
d.gtld-servers.net.     172800  IN      AAAA    2001:500:856e::30
e.gtld-servers.net.     172800  IN      AAAA    2001:502:1ca1::30
f.gtld-servers.net.     172800  IN      AAAA    2001:503:d414::30
g.gtld-servers.net.     172800  IN      AAAA    2001:503:eea3::30
h.gtld-servers.net.     172800  IN      AAAA    2001:502:8cc::30
i.gtld-servers.net.     172800  IN      AAAA    2001:503:39c1::30
j.gtld-servers.net.     172800  IN      AAAA    2001:502:7094::30
k.gtld-servers.net.     172800  IN      AAAA    2001:503:d2d::30
l.gtld-servers.net.     172800  IN      AAAA    2001:500:d937::30
m.gtld-servers.net.     172800  IN      AAAA    2001:501:b1f9::30

;; Query time: 115 msec
;; SERVER: 198.41.0.4#53(198.41.0.4)
;; WHEN: Tue Feb 16 07:04:06 UTC 2021
;; MSG SIZE  rcvd: 837
```

go ask them (name servers)

```
➜  ~ dig @m.gtld-servers.net www.example.net

; <<>> DiG 9.10.6 <<>> @m.gtld-servers.net www.example.net
; (2 servers found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 2631
;; flags: qr rd; QUERY: 1, ANSWER: 0, AUTHORITY: 2, ADDITIONAL: 5
;; WARNING: recursion requested but not available

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;www.example.net.               IN      A

;; AUTHORITY SECTION:
example.net.            172800  IN      NS      a.iana-servers.net.
example.net.            172800  IN      NS      b.iana-servers.net.

;; ADDITIONAL SECTION:
a.iana-servers.net.     172800  IN      A       199.43.135.53
a.iana-servers.net.     172800  IN      AAAA    2001:500:8f::53
b.iana-servers.net.     172800  IN      A       199.43.133.53
b.iana-servers.net.     172800  IN      AAAA    2001:500:8d::53

;; Query time: 110 msec
;; SERVER: 192.55.83.30#53(192.55.83.30)
;; WHEN: Tue Feb 16 07:08:40 UTC 2021
;; MSG SIZE  rcvd: 177
```

go ask them (name servers)

```
➜  ~ dig @a.iana-servers.net www.example.net

; <<>> DiG 9.10.6 <<>> @a.iana-servers.net www.example.net
; (2 servers found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 23131
;; flags: qr aa rd; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1
;; WARNING: recursion requested but not available

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;www.example.net.               IN      A

;; ANSWER SECTION:
www.example.net.        86400   IN      A       93.184.216.34

;; Query time: 233 msec
;; SERVER: 199.43.135.53#53(199.43.135.53)
;; WHEN: Tue Feb 16 07:10:11 UTC 2021
;; MSG SIZE  rcvd: 60
```

answer

Insert Records into DNS

Example: new startup "Network Utopia"

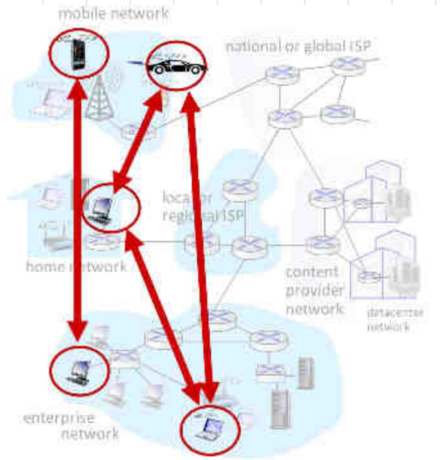
- register name **networkuptopia.com** at *DNS registrar* (e.g., Network Solutions)
  - provide names, IP addresses of authoritative name server (primary and secondary)
  - registrar inserts NS, A RRs into .com TLD server:

    (networkutopia.com, dns1.networkutopia.com, NS)

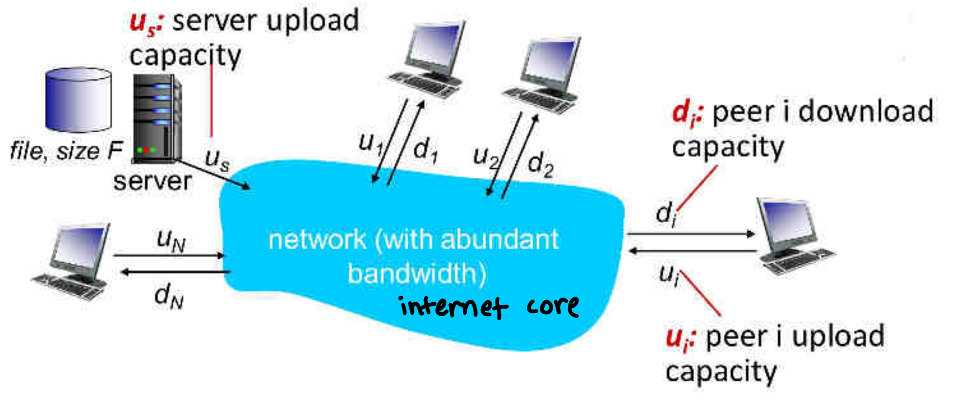
    (dns1.networkutopia.com, 212.212.212.1, A)
- create authoritative server locally with IP address 212.212.212.1
  - type A record for www.networkuptopia.com
  - type MX record for networkutopia.com

peer-to-peer application

- No always on server
- Arbitrary end systems directly communicate
- Self-scalability
- More complex
- Peers intermittently connected and constantly change IP address
- eg: BitTorrent, VoIP (Skype)



Q: How much time to distribute file (size F) from one server to n peers?



- bottlenecks: access networks
- peer upload/download capacity is limited resource
- distribution time: time taken to get a copy of file to all N peers

File Distribution Time: Client-Server

- server transmission: must sequentially transmit N file copies
- time taken for one file = F/u_s
  time taken for N files = NF/u_s
- client: each client must download file copy
- d_{min} = min download rate of any client
- min time = F/d_{min}

- time to distribute F to N clients using client-server approach

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

linearly increases w N

File Distribution Time: P2P

- server: must upload at least one copy (F/u_s)
- each client downloads and uploads file

  min client download time = F/d_{min}

- clients: as aggregrate must download NF bits

  max upload rate (limiting max download rate)

  $$u_s + \sum u_i$$

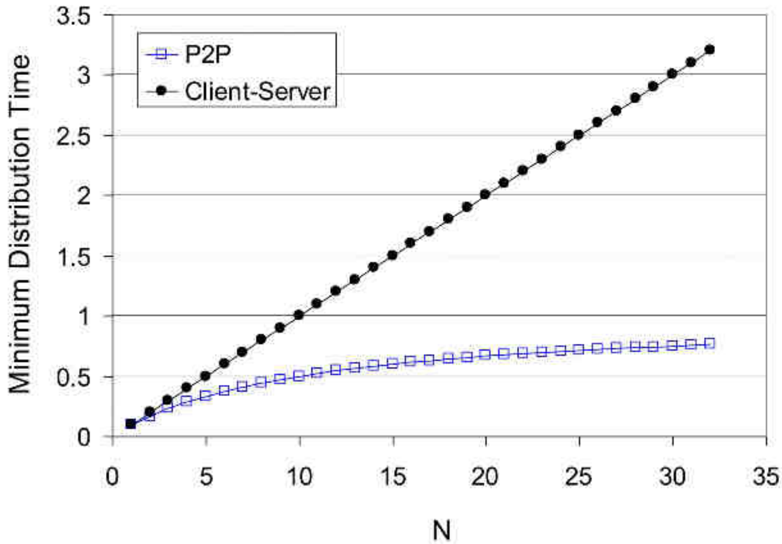
- time to distribute F to N clients using P2P approach

$$D_{P2P} > \max\left\{\frac{F}{u_s}, \frac{F}{d_{min}}, \frac{NF}{(u+\sum u_i)}\right\}$$

linearly increases w N

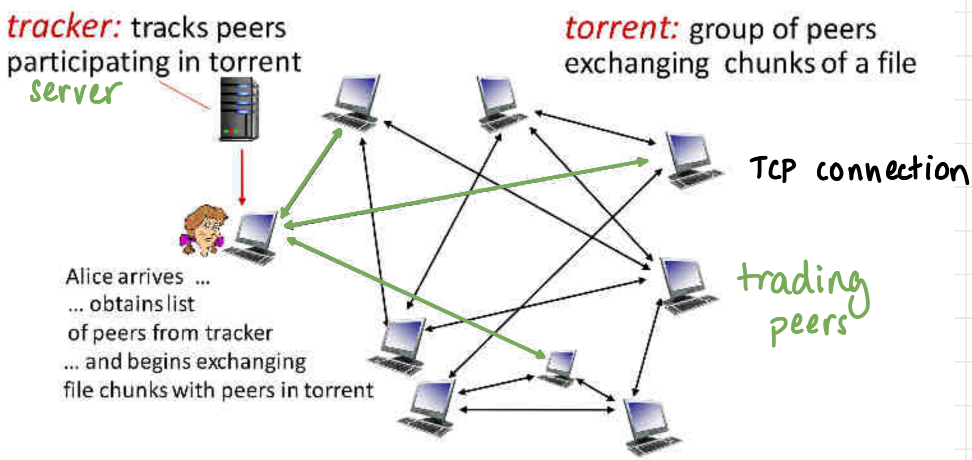
C-S vs P2P

Client (all peers) upload rate = u, F/u = 1 hour, us = 10u, dmin ≥ us



BitTorrent

- P2P file distribution
- Files divided into 256 kb chunks
- Peers in torrent send and receive file chunks



- New peer joining torrent: has no chunks but accumulates over time
- Registers with tracker to get list of peers, connects to a subset of peers (neighbours)
- While downloading, peer also uploads to other peers
- Churn: peers change neighbours for exchange
- Once peer has file entirely, it may leave (selfish) or remain in torrent (altruistically)

Requesting Chunks

- At any time, different peers have different subsets of file chunks
- Periodically, one peer asks every other peer for the list of chunks they have (neighbours)
- Requests for missing chunks (rarest first) less available in torrent

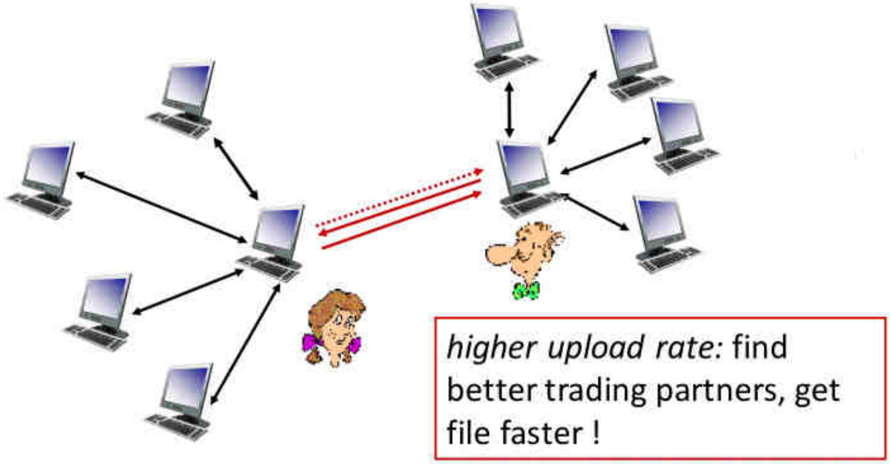
Sending Chunks: tit-for-tat

- One peer sends chunks to four peers currently sending chunks to said peer at highest rate (upload rate)
- Other peers are choked (do not receive chunks from said peer)
- Top 4 reevaluated every 10 seconds
- Randomly select another peer every 30 seconds and starts sending chunks to it (optimistically unchoke)

(1) Alice "optimistically unchokes" Bob

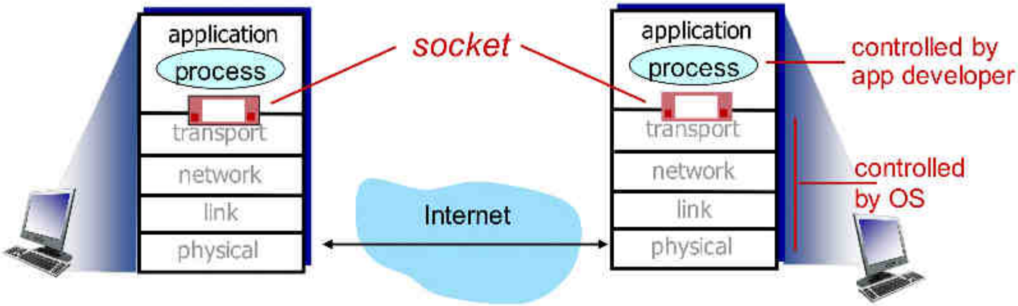
(2) Alice becomes one of Bob's top-four providers; Bob reciprocates

(3) Bob becomes one of Alice's top-four providers



SOCKET PROGRAMMING

- TCP and UDP
- Socket: door between application process and end-to-end transport protocol
- Client-server applications that communicate using sockets
- Process to process communication

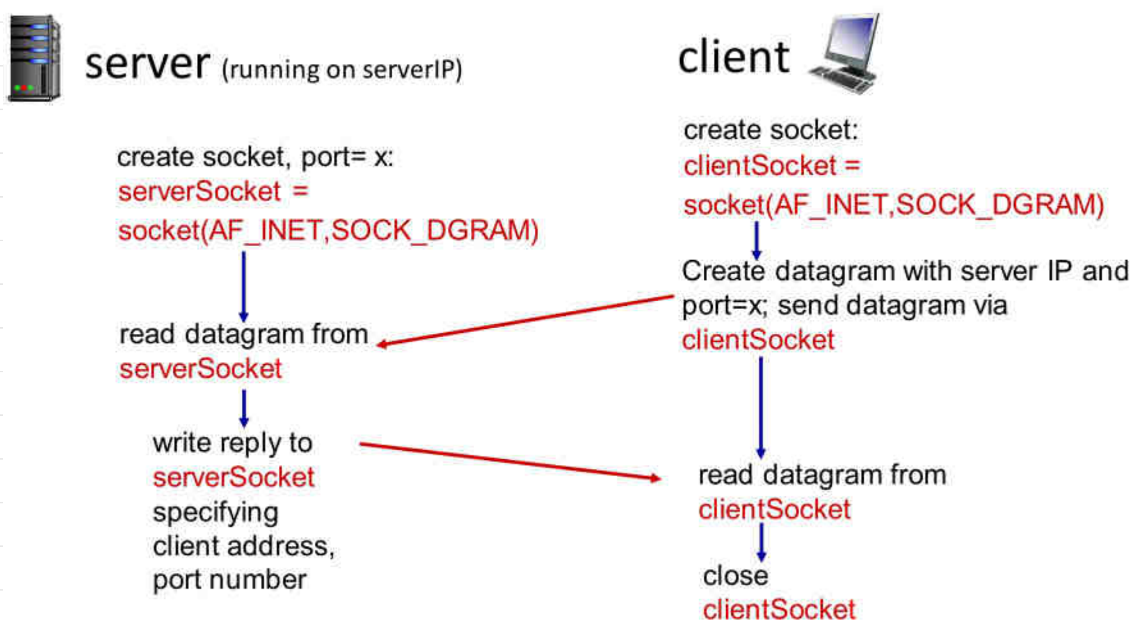


Socket types

- UDP: unreliable datagram
- TCP: reliable, byte-stream oriented

SOCKET PROGRAMMING WITH UDP

- No 'connection' between client and server (no handshake)
- Sender explicitly attaches IP destination address and port number to every packet
- Receiver extracts sender IP address and port number from received packet
- Data maybe lost or received out of order



Python UDPClient

create socket →

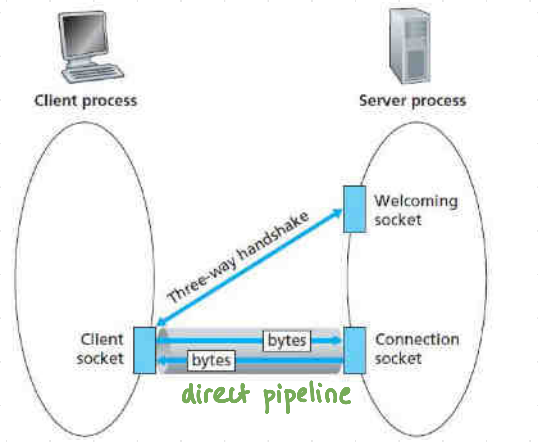
```
#!/usr/bin/python2

from socket import *

serverName = 'localhost'
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_DGRAM)
message = raw_input('Input lowercase sentence: ')
clientSocket.sendto(message,(serverName, serverPort))
modifiedMessage, serverAddress = clientSocket.recvfrom(2048)

print modifiedMessage
clientSocket.close()
```

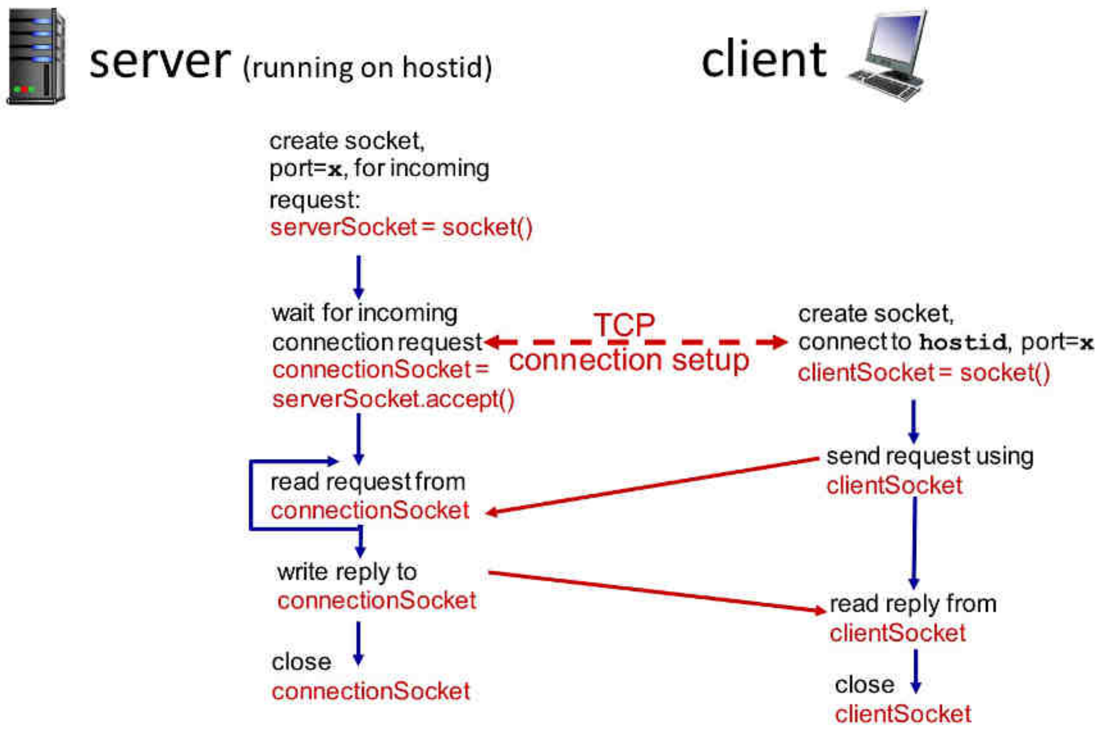
Python UDPServer

```
#!/usr/bin/python2

from socket import *

serverPort = 12000
serverSocket = socket(AF_INET, SOCK_DGRAM)
serverSocket.bind(('', serverPort))
print "The server is ready to receive"

while 1:
    message, clientAddress = serverSocket.recvfrom(2048)
    modifiedMessage = message.upper()
    serverSocket.sendto(modifiedMessage, clientAddress)
```

SOCKET PROGRAMMING WITH TCP

- Client-server handshake
- Server must be running and must have a socket that welcomes client's contact
- Client contacts server:
  - Create TCP socket (IP address, port no. of server process)
  - Client TCP establishes connection to server TCP
- Server TCP creates new socket (different from welcome socket) for server process to communicate with that particular client
- Allows server to communicate with multiple clients (new socket for each client)
- Source port numbers used to distinguish clients (client does not pick port number at sender's side; automatically assigned by OS)
- Reliable, byte-stream transfer (pipe)





Python TCPClient

```
#!/usr/bin/python2

from socket import *

serverName = 'localhost'
serverPort = 12000

clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName, serverPort))
sentence = raw_input('Input lowercase sentence: ')
clientSocket.send(sentence)
modifiedSentence = clientSocket.recv(1024)
print 'From Server: ', modifiedSentence
clientSocket.close()
```

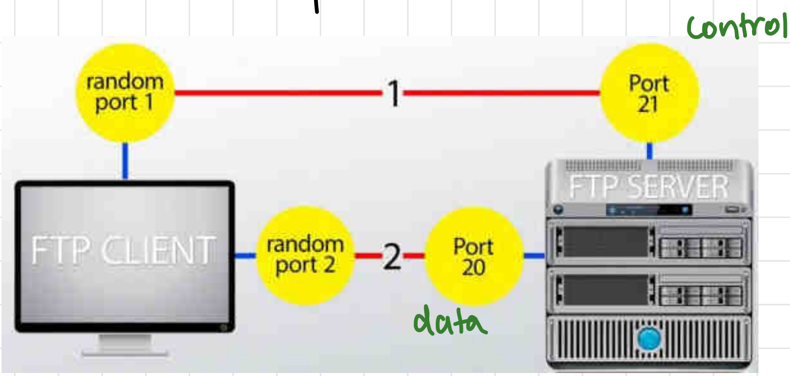
Python TCPServer

```python
#!/usr/bin/python2

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 1:
    connectionSocket, addr = serverSocket.accept()
    sentence = connectionSocket.recv(1024)
    capitalizedSentence = sentence.upper()
    connectionSocket.send(capitalizedSentence)
connectionSocket.close()
```

Other application-layer protocols

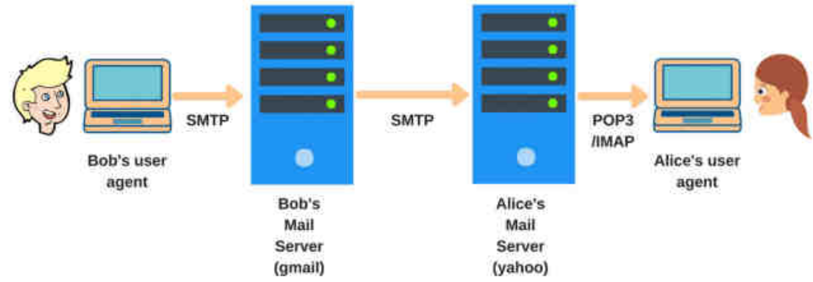
1. File Transfer Protocol (FTP)

- exchange large files on internet TCP
- invoked from cmd or gui
- allows to (delete, rename, move, copy) files at a server
- data connection - port 20
- control connection - port 21



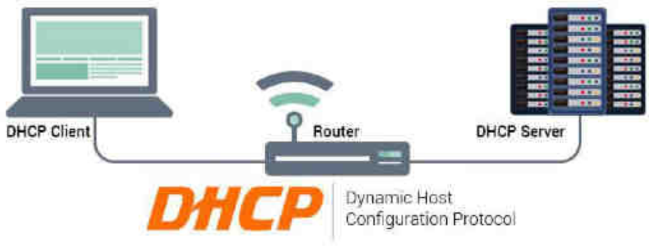
2. Simple Mail Transfer Protocol (SMTP)

- email transmission
- connections secured with SSL (secure socket layer)
- messages stored and then forwarded to destination (relay)
- SMTP – port 25 of TCP



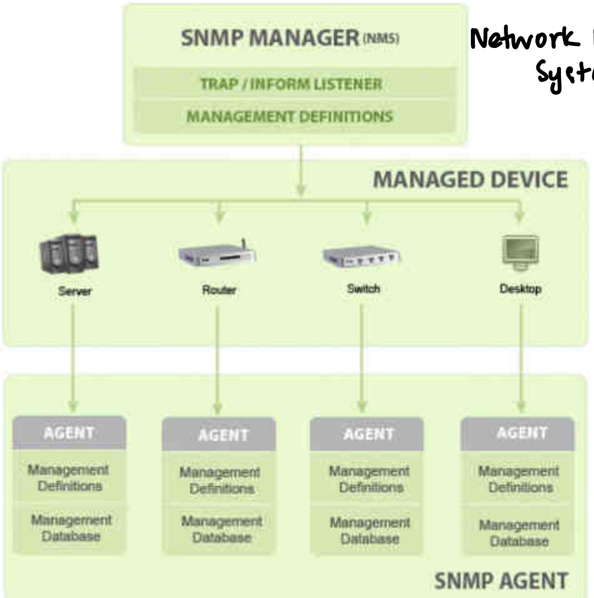
3. Dynamic Host Configuration Protocol (DHCP)

- assign IP addresses to computers in a network dynamically
- IP addresses can change even when hosts are in network (DHCP leases)
- DHCP server: port 67
  DHCP client: port 68
- Client-server model
- Based on discovery, offer, request, ACK
- Subnet mask, DNS server address, default gateway



4. Simple Network Management Protocol (SNMP)

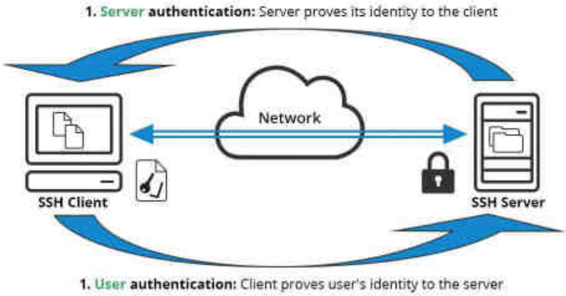
- exchange management information between network devices
- basic component and functionalities
  - SNMP Manager
  - Managed Devices
  - SNMP Agents
  - MIB (Management Information Base)



Network Manager System

5. Telnet & SSH

- communicate with remote device
- used by network admins to access devices/manage devices
- Telnet client & telnet server
- Telnet : port 23
- SSH : public/private encryption : TCP port 22



SUMMARY OF APPLICATION LAYER PROTOCOLS

Port #	Application Layer Protocol	Type	Description
20	FTP	TCP	File Transfer Protocol - data
21	FTP	TCP	File Transfer Protocol - control
22	SSH	TCP/UDP	Secure Shell for secure login
23	Telnet	TCP	Unencrypted login
25	SMTP	TCP	Simple Mail Transfer Protocol
53	DNS	TCP/UDP	Domain Name Server
67/68	DHCP	UDP	Dynamic Host
80	HTTP	TCP	HyperText Transfer Protocol
123	NTP	UDP	Network Time Protocol
161,162	SNMP	TCP/UDP	Simple Network Management Protocol
389	LDAP	TCP/UDP	Lightweight Directory Authentication Protocol
443	HTTPS	TCP/UDP	HTTP with Secure Socket Layer