COMPUTER NETWORKS

TEAM NETWORKS
Department of Computer Science and Engineering

COMPUTER NETWORKS

Application Layer

Department of Computer Science and Engineering

Unit – 2 Application Layer

2.3 The Domain Name System

2.4 P2P Applications

2.5 Socket Programming with TCP & UDP

2.6 Other Application Layer Protocols

people: many identifiers:

- SSN, name, passport #

Internet hosts, routers:

- IP address (32 bit) - used for addressing datagrams
- "name", e.g., cs.umass.edu - used by humans

Q: how to map between IP address and name, and vice versa ?

Domain Name System:

- *distributed database* implemented in hierarchy of many *name servers*
- *application-layer protocol:* hosts, name servers communicate to *resolve* names (address/name translation)
  - note: core Internet function, *implemented as application-layer protocol*
  - complexity at network's "edge"

www.abc.example.com -> **Canonical Host Name**
www.example.com -> **Alias Name**

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?

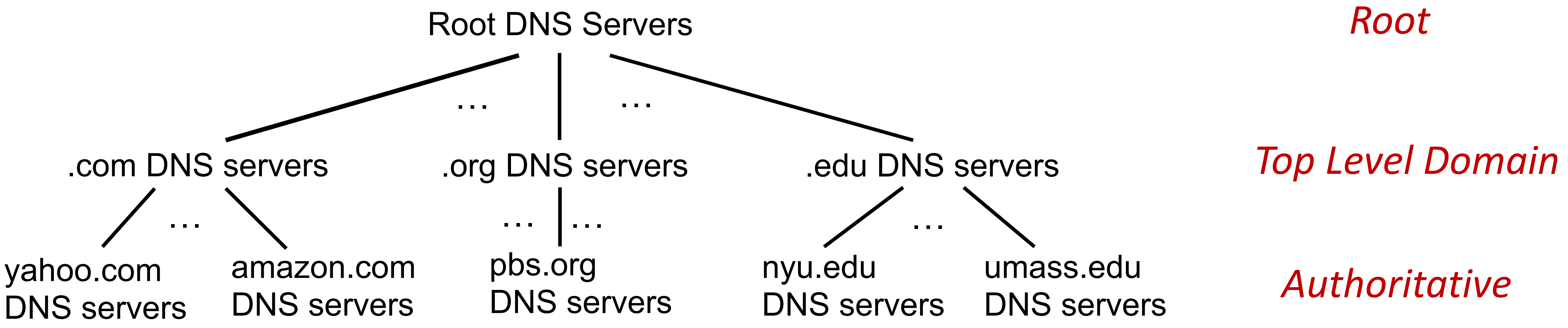
- single point of failure
- traffic volume
- distant centralized database
- maintenance

A: doesn't scale!

- Comcast DNS servers alone: 600B DNS queries per day

www.abc.example.com -> **Canonical Host Name**
bob@example.com -> **Alias Name**

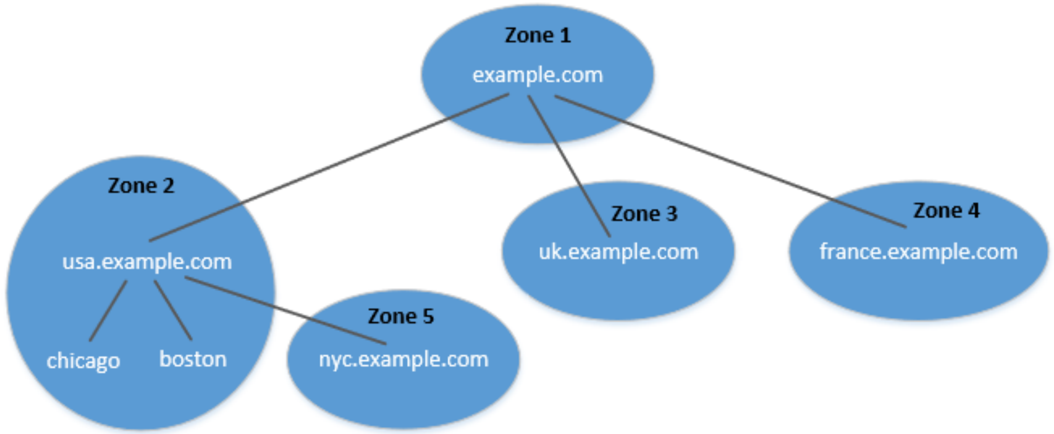
DNS: a distributed, hierarchical database



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

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

DNS Zone vs Domain



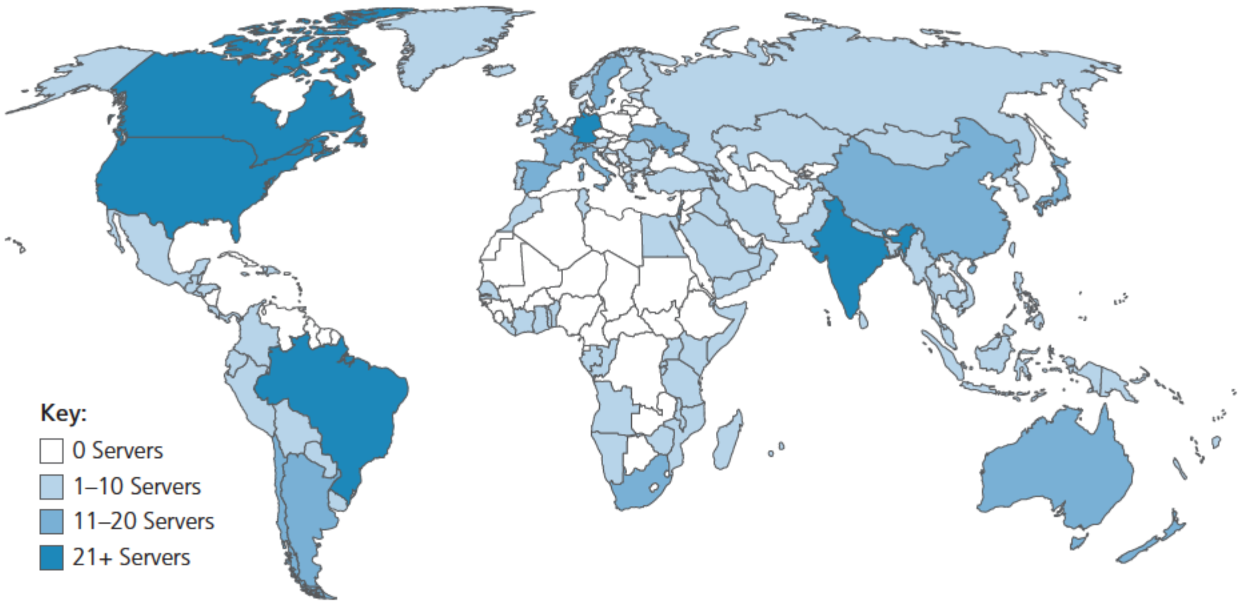
- DNS is organized according to zones.
- A zone groups contiguous domains and subdomains on the domain tree.
- Assign management authority to an entity.

- The tree structure depicts subdomains within example.com domain.
- Multiple DNS zones one for each country. The zone keeps records of who the authority is for each of its subdomains.
- The zone for example.com contains only the DNS records for the hostnames that do not belong to any subdomain like mail.example.com

DNS: root name servers

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

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



TLD: authoritative servers

Top-Level Domain (TLD) servers:

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

Authoritative DNS servers:

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

Local DNS Name Servers

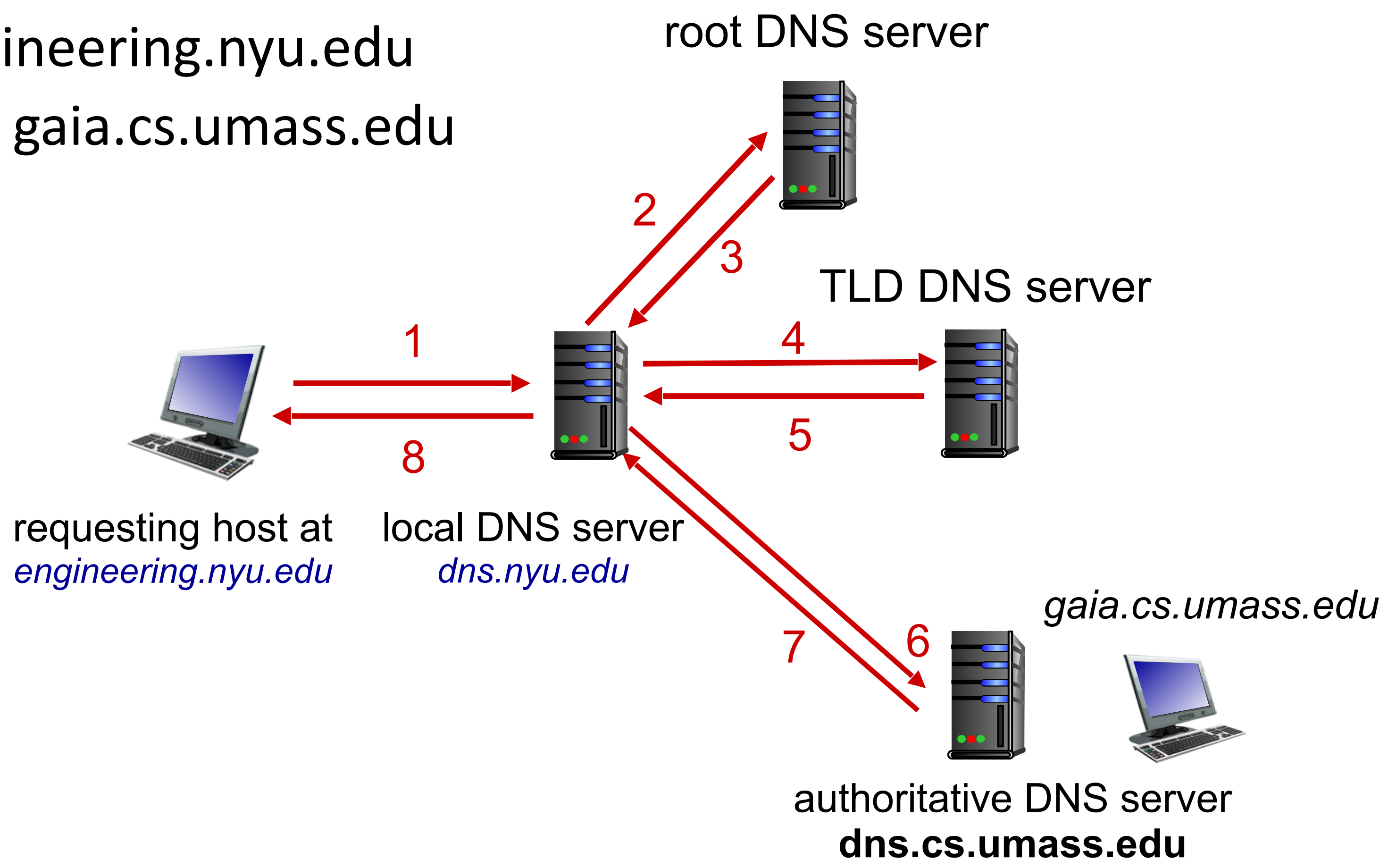
- does not strictly belong to hierarchy
- each ISP (residential ISP, company, university) has one
  - also called "default name server"
- when host makes DNS query, query is sent to its local DNS server
  - has local cache of recent name-to-address translation pairs (but may be out of date!)
  - acts as proxy, forwards query into hierarchy

DNS name resolution: iterated query

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

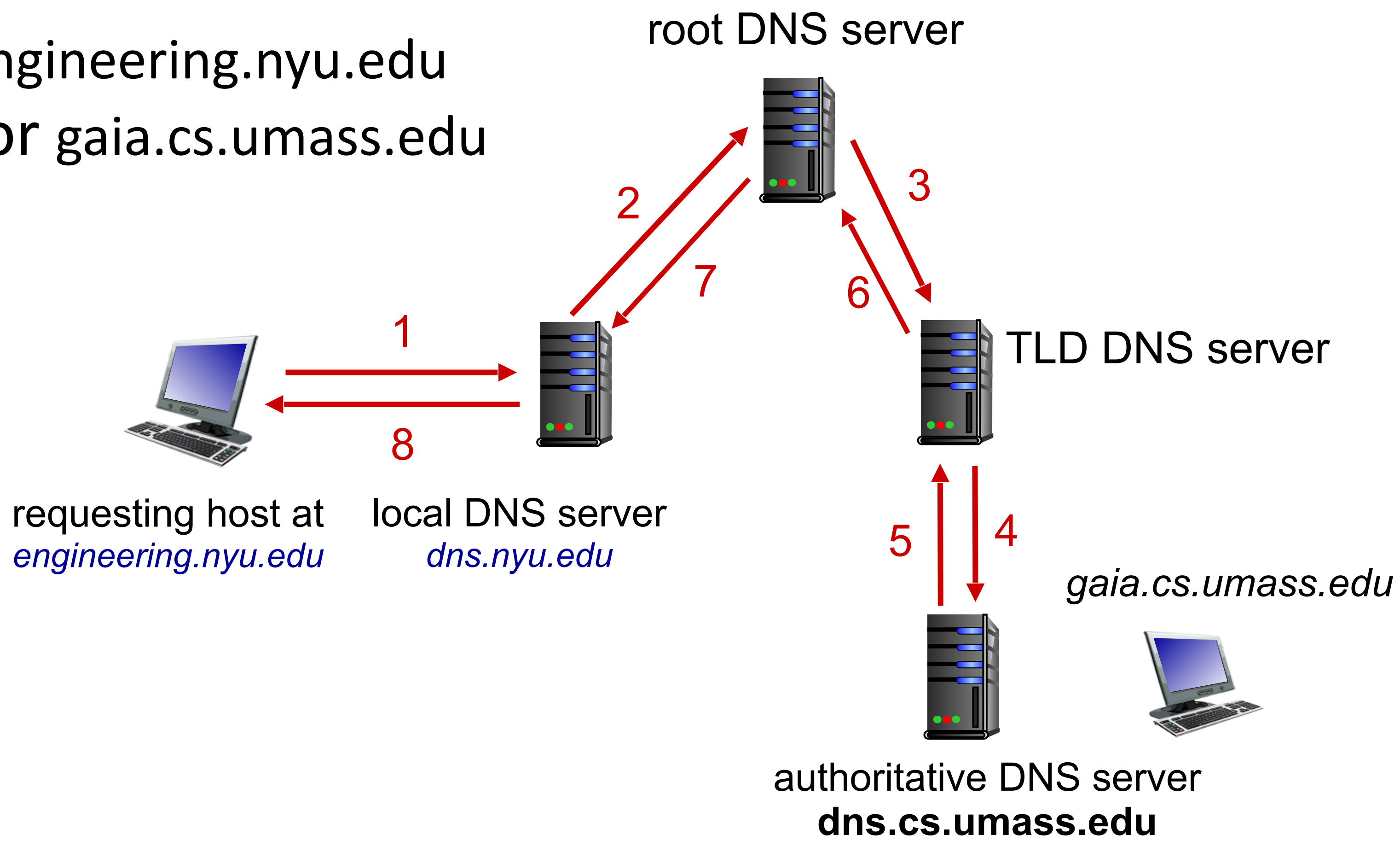
Iterated query:

- 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?

Caching and Updating DNS Records

- Suppose that a host **apricot.nyu.edu** queries **dns.nyu.edu** for the IP address for the hostname **cnn.com**. After an hour later, another NYU host, say, **kiwi.nyu.edu**, also queries **dns.nyu.edu.**
- once (any) name server learns mapping, it *caches* mapping
  - cache entries timeout (disappear) after some time (TTL)
  - TLD servers typically cached in local name servers
    - thus root name servers not often visited
- cached entries may be *out-of-date* (best-effort name-to-address translation!)
  - if name host changes IP address, may not be known Internet-wide until all TTLs expire!
- update/notify mechanisms proposed IETF standard
  - RFC 2136

DNS: distributed database storing resource records (RR)

RR format: (`name, value, type, ttl`)

type=A

- `name` is hostname
- `value` is IP address

relayl.bar.foo.com, 145.37.93.126, A

type=NS

- `name` is domain (e.g., foo.com)
- `value` is hostname of authoritative name server for this domain

foo.com, dns.foo.com, NS

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

ibm.com, servereast.backup2.ibm.com, CNAME

type=MX

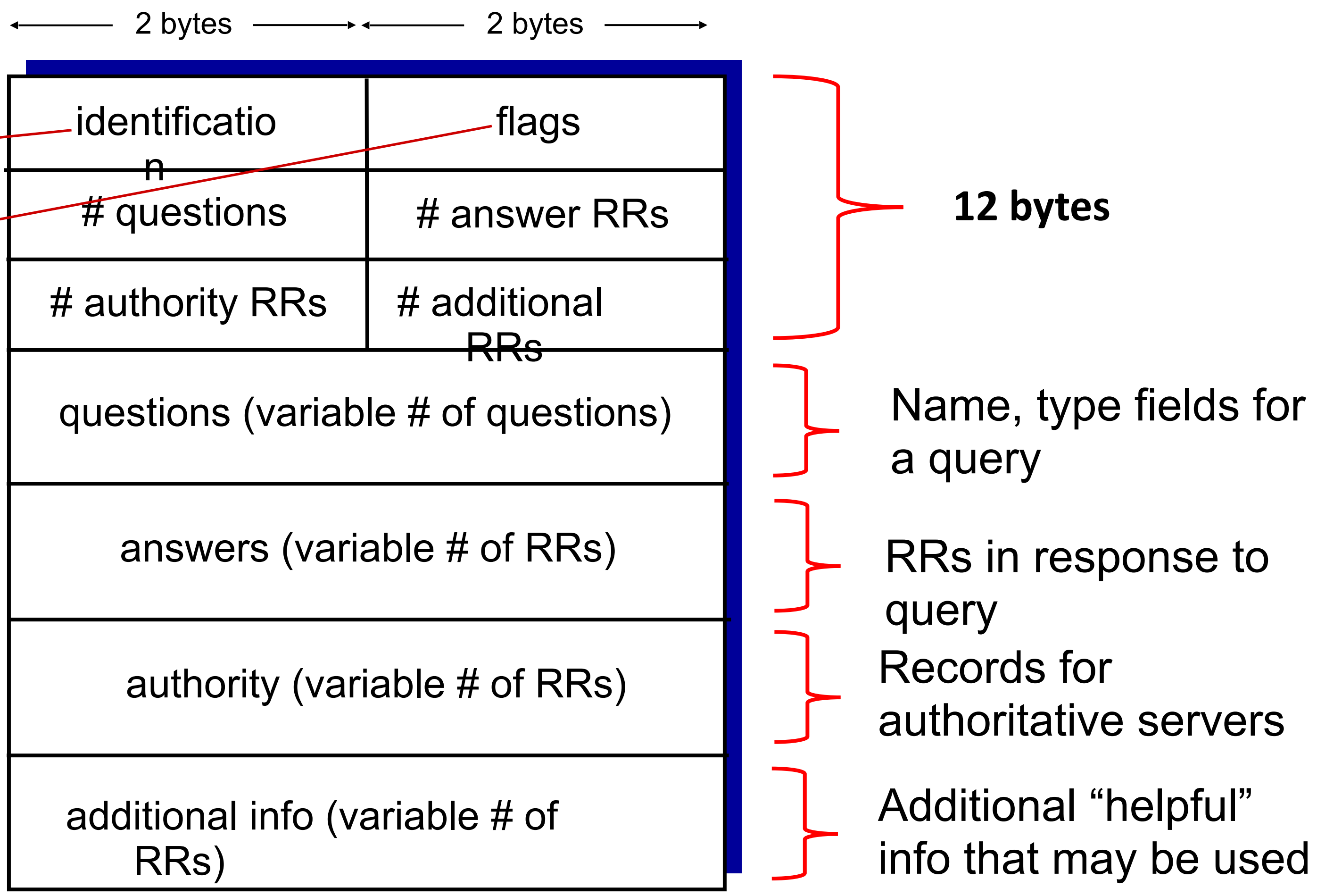
- `value` is canonical name of a mailserver associated with alias hostname `name`

example.com, mail.example.com, MX

DNS *query* and *reply* messages, both have same *format:*

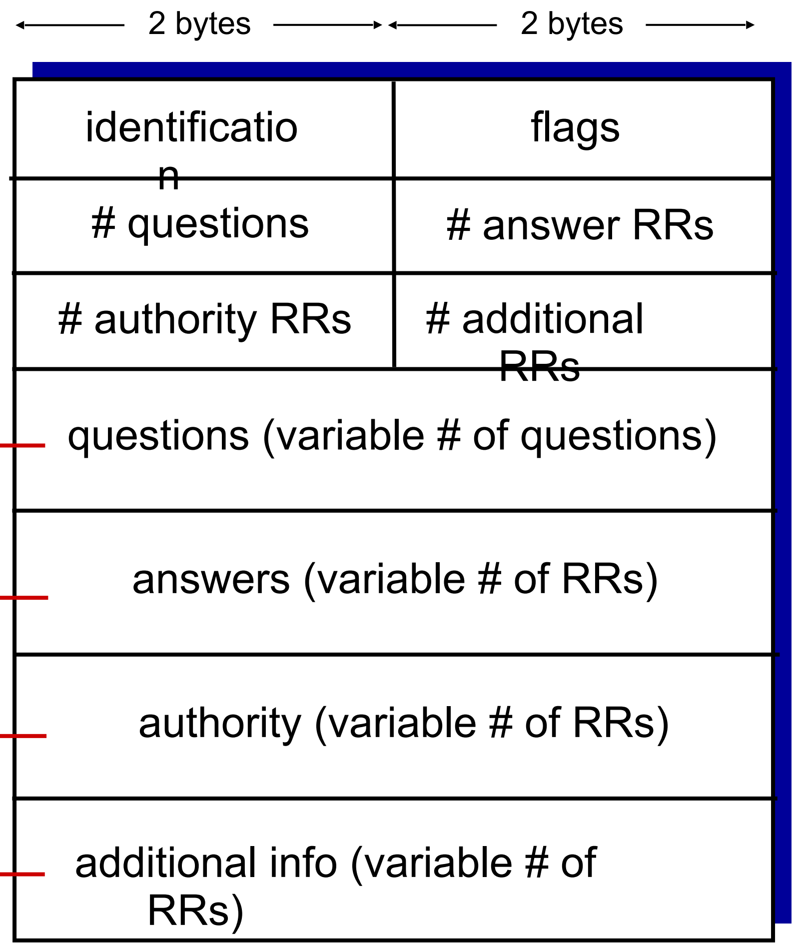
message header:

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



DNS Protocol Messages

DNS *query* and *reply* messages, both have same *format:*



name, type fields for a query

RRs in response to query

records for authoritative servers

additional " helpful" info that may be used

Type (for example, A, NS, CNAME, and MX), the Value, and the TTL.

Emulating Local DNS Server (Step 1: Ask Root)

Directly send the query to this server.

```
seed@ubuntu:~$ dig @a.root-servers.net www.example.net

(Only a portion of the reply is shown here)
;; QUESTION SECTION:
;www.example.net.                IN      A

;; AUTHORITY SECTION:
net.                    172800   IN      NS      m.gtld-servers.net.
net.                    172800   IN      NS      l.gtld-servers.net.
net.                    172800   IN      NS      k.gtld-servers.net.

;; ADDITIONAL SECTION:
m.gtld-servers.net.     172800   IN      A       192.55.83.30
l.gtld-servers.net.     172800   IN      A       192.41.162.30
k.gtld-servers.net.     172800   IN      A       192.52.178.30
```

No answer (the root does not know the answer)

Go ask them!

Steps 2-3: Ask .net & example.net servers

```
seed@ubuntu:~$ dig @m.gtld-servers.net www.example.net

;; 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       199.43.132.53
b.iana-servers.net.    172800  IN      A       199.43.133.53
```

Go ask them!

```
seed@ubuntu:$ dig @a.iana-servers.net www.example.net

;; QUESTION SECTION:
;www.example.net.               IN      A

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

- Ask an example.net nameservers.

Finally got the answer

Inserting records into DNS

Example: new startup "Network Utopia"

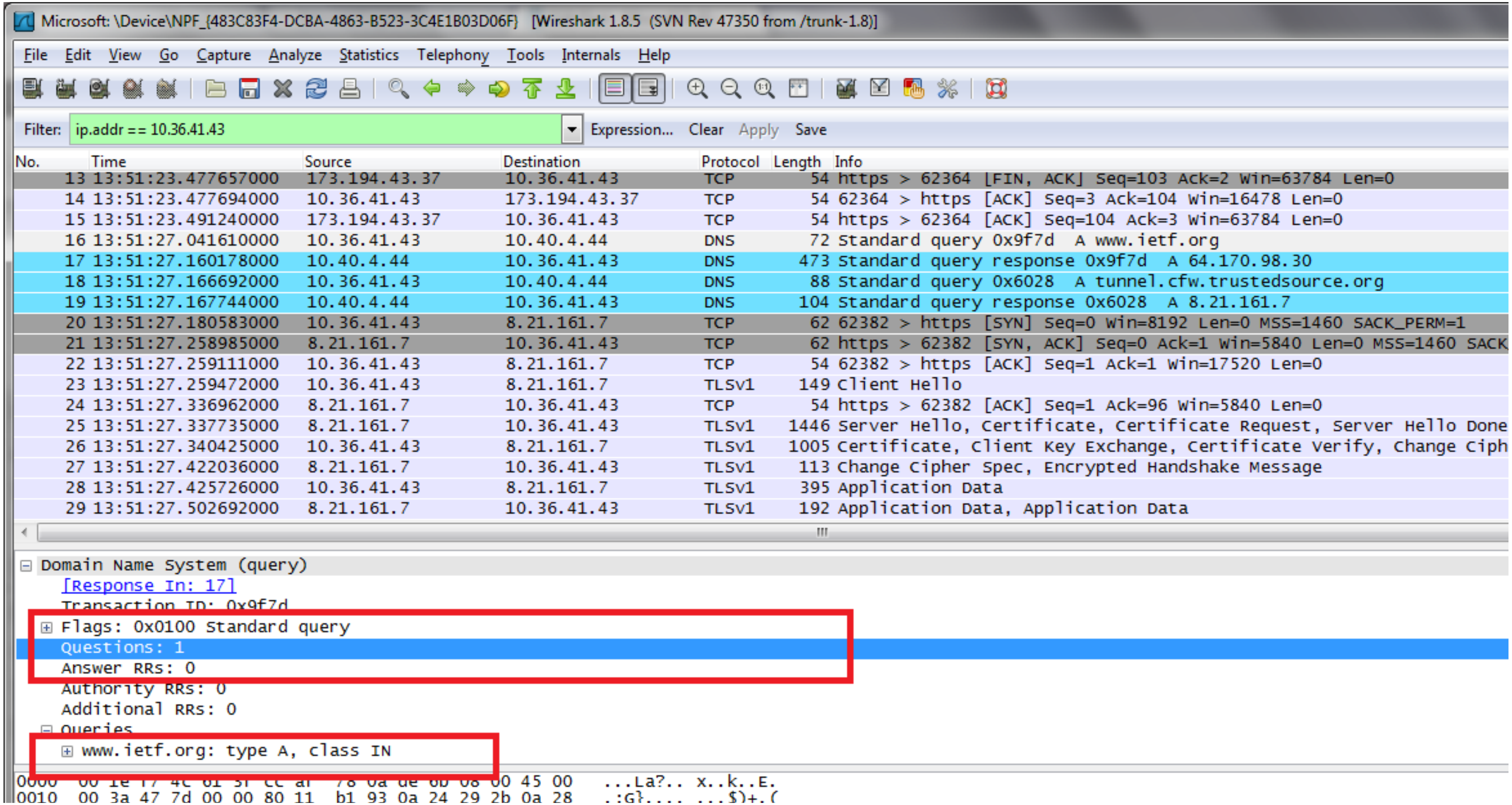
- register name **networkuptopia.com** at *DNS registrar* (e.g., Network Solutions)
  - provide names, IP addresses of authoritative name server (primary and secondary)
  - registrar inserts NS, A RRs into .com TLD server:

    `(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

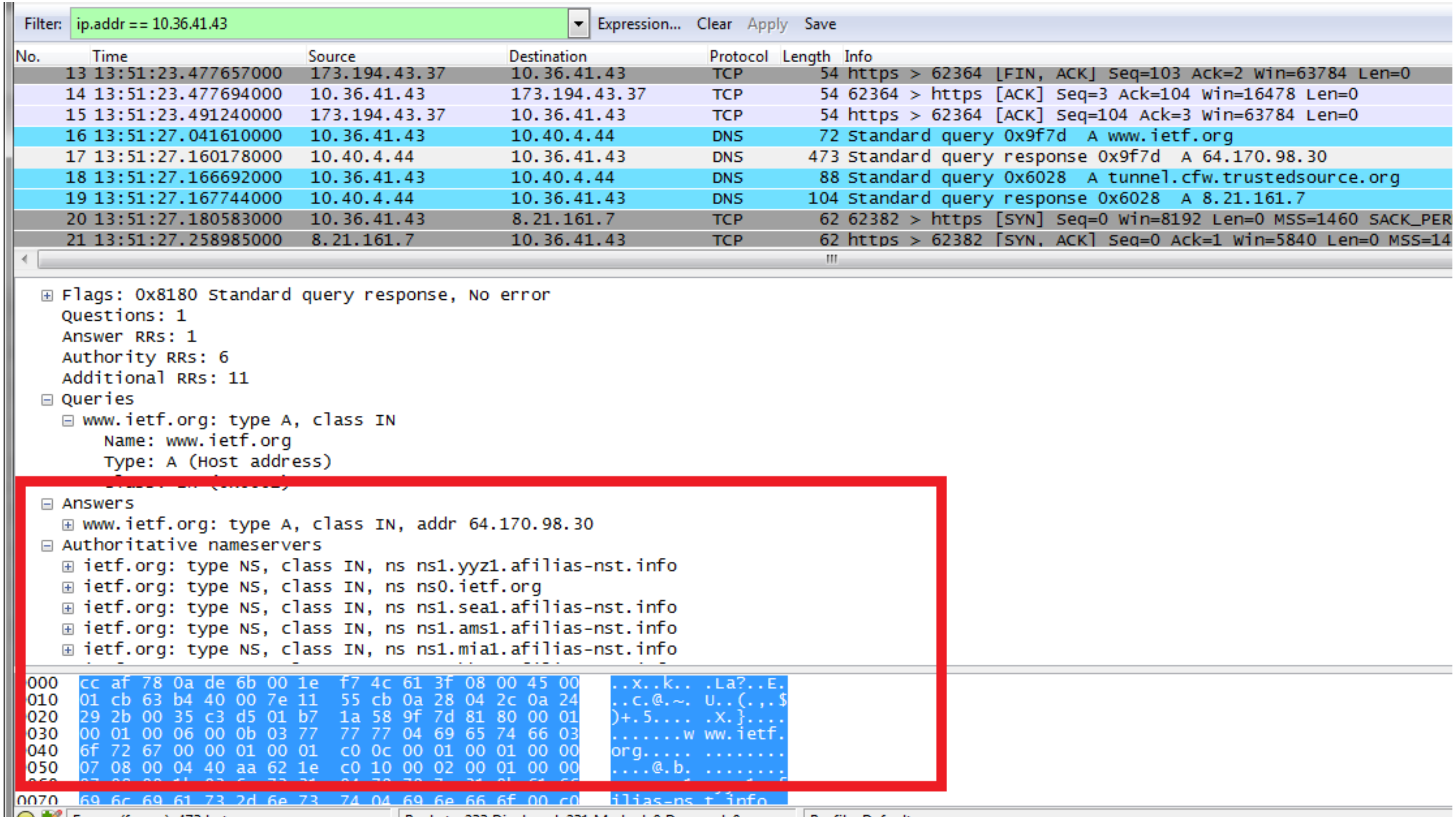
COMPUTER NETWORKS

DNS Request - Wireshark Packet Capture



COMPUTER NETWORKS

DNS Response - Wireshark Packet Capture



Suggested Readings

- DNS (Domain Name System) – Explained – https://youtu.be/JkEYOt08-rU
- How a DNS Server (Domain Name System) works – https://youtu.be/rdVPflECed8
- Wireshark Lab: DNS v7.0 – http://www-net.cs.umass.edu/wireshark-labs/Wireshark_DNS_v7.0.pdf

Q&A

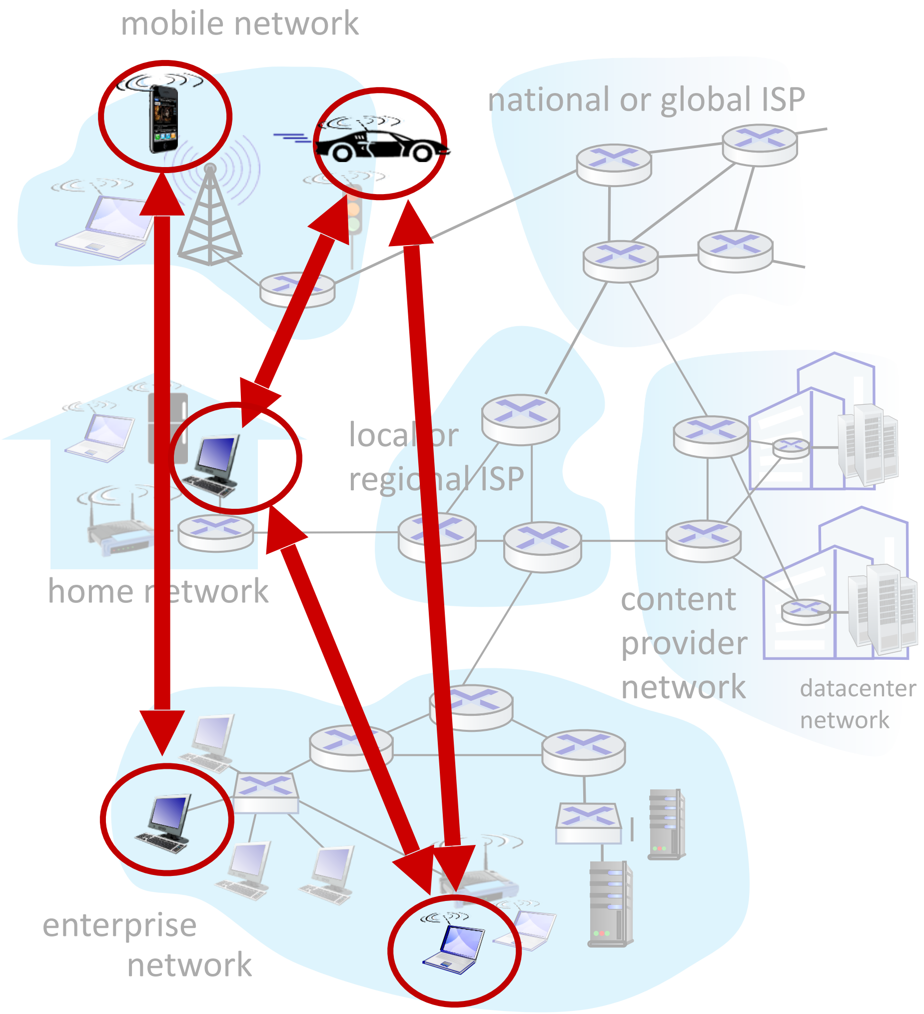
Thank You
For Your Attention

Unit – 2 Application Layer

- 2.3 The Domain Name System
- 2.4 P2P Applications
- 2.5 Socket Programming with TCP & UDP
- 2.6 Other Application Layer Protocols

Peer-to-peer (P2P) 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, and new service demands
- peers are intermittently connected and change IP addresses
  - complex management
- examples: P2P file sharing (BitTorrent), media streaming (Spotify), VoIP (Skype)

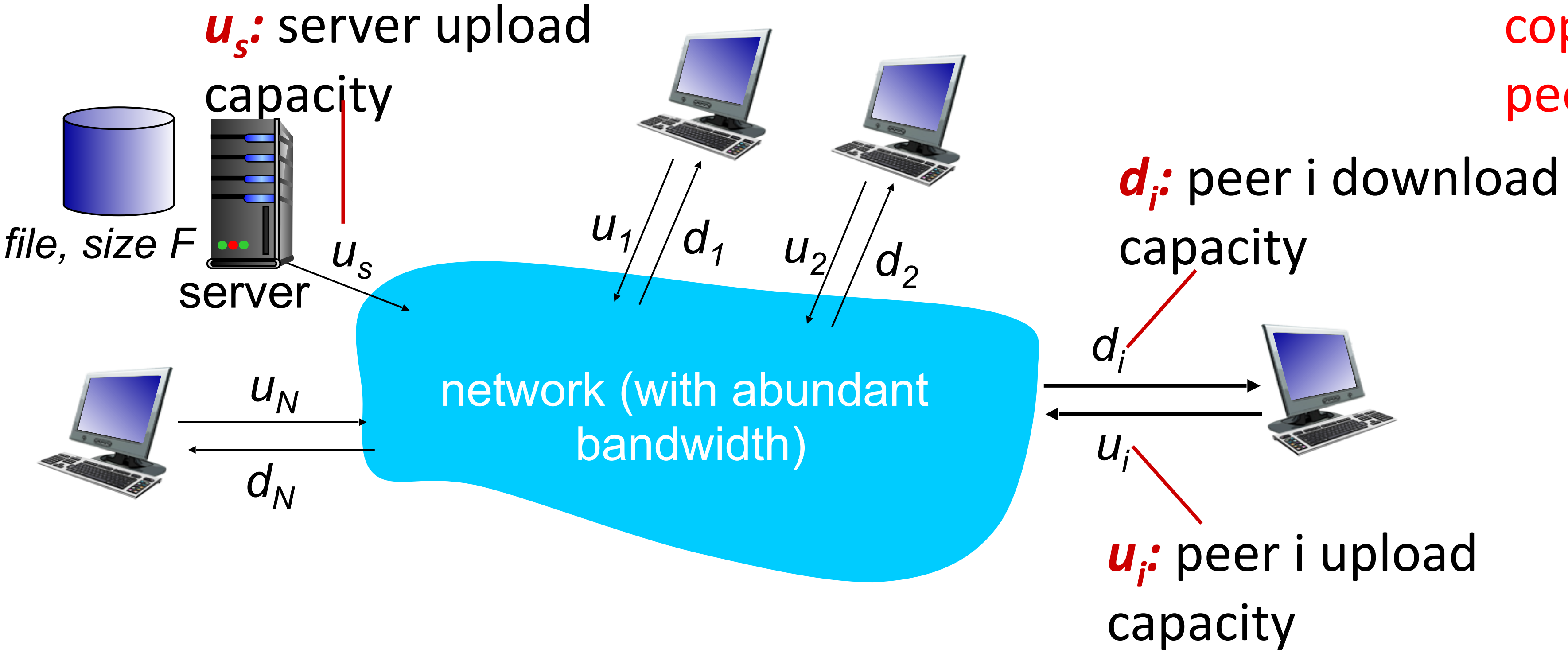


File distribution: client-server vs P2P

Q: how much time to distribute file (size *F*) from one server to *N peers*?

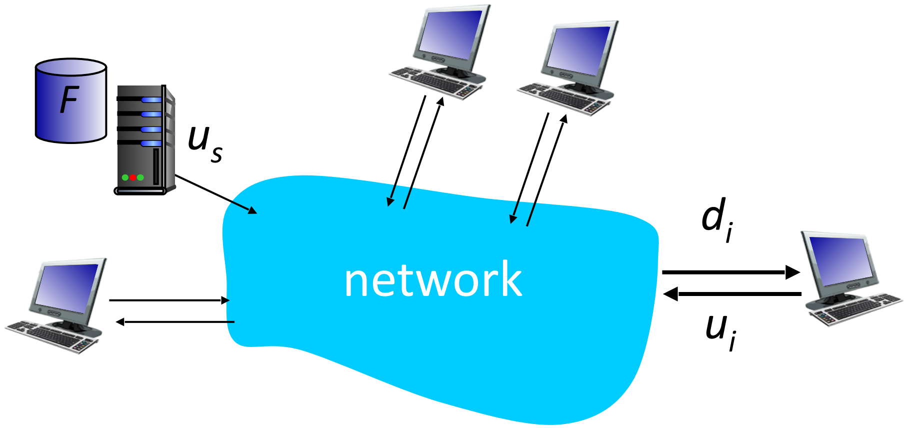
- peer upload/download capacity is limited resource

The **distribution time** is the time it takes to get a copy of the file to all *N* peers.



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 = i.e., d_{min} = min {d1, dp,……, dN }
  - min client download time: F/d_{min}



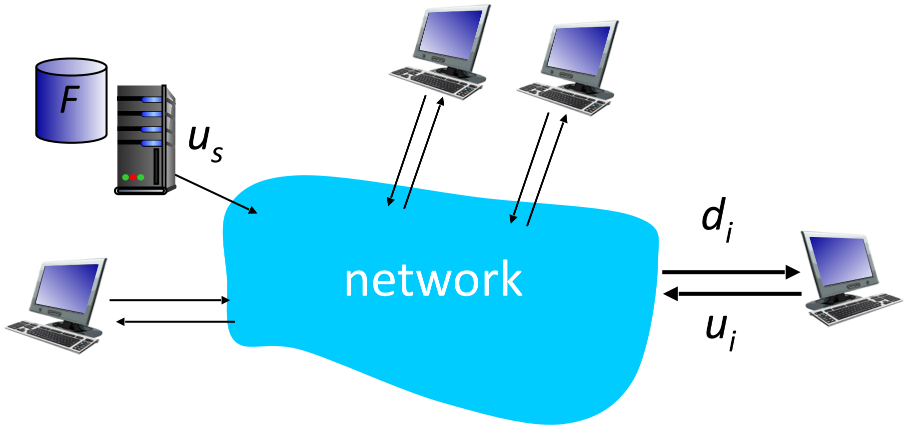
time to distribute F to N clients using client-server approach

$$D_{c\text{-}s} > max\{NF/u_{s,},F/d_{min}\}$$

increases linearly in N

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 $u_s + \Sigma u_i$



Total upload capacity of the system as a whole

Eqtn - provides a lower bound for the minimum distribution time for the P2P architecture.

time to distribute F to N clients using P2P approach

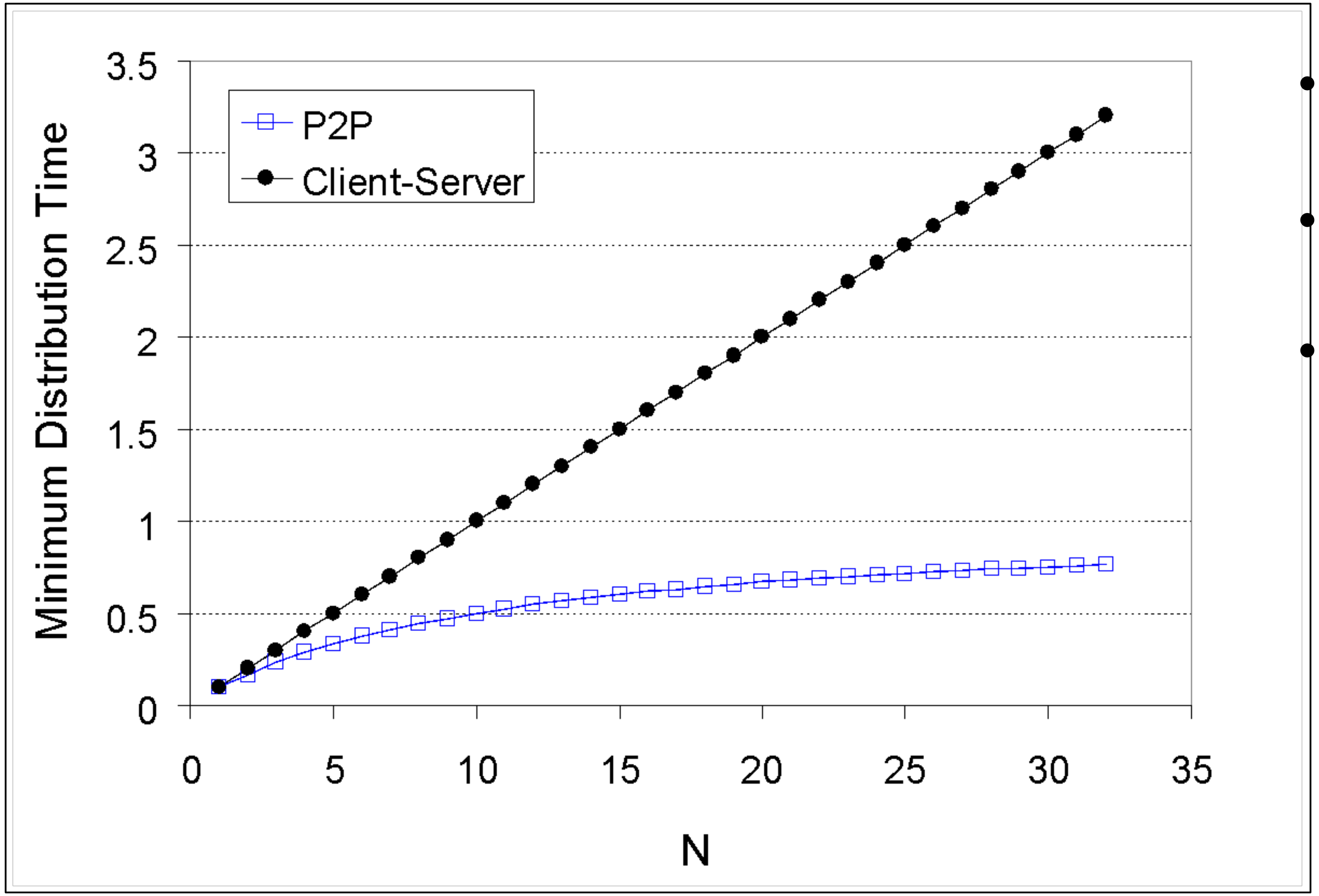
$$D_{P2P} > \max\{F/u_s, F/d_{min}, NF/(u_s + \Sigma u_i)\}$$

increases linearly in *N* …

… but so does this, as each peer brings service capacity

Client-server vs. P2P: example

Client (all peers) upload rate = u, F/u = 1 hour, $u_s = 10u$, $d_{min} \geq u_s$



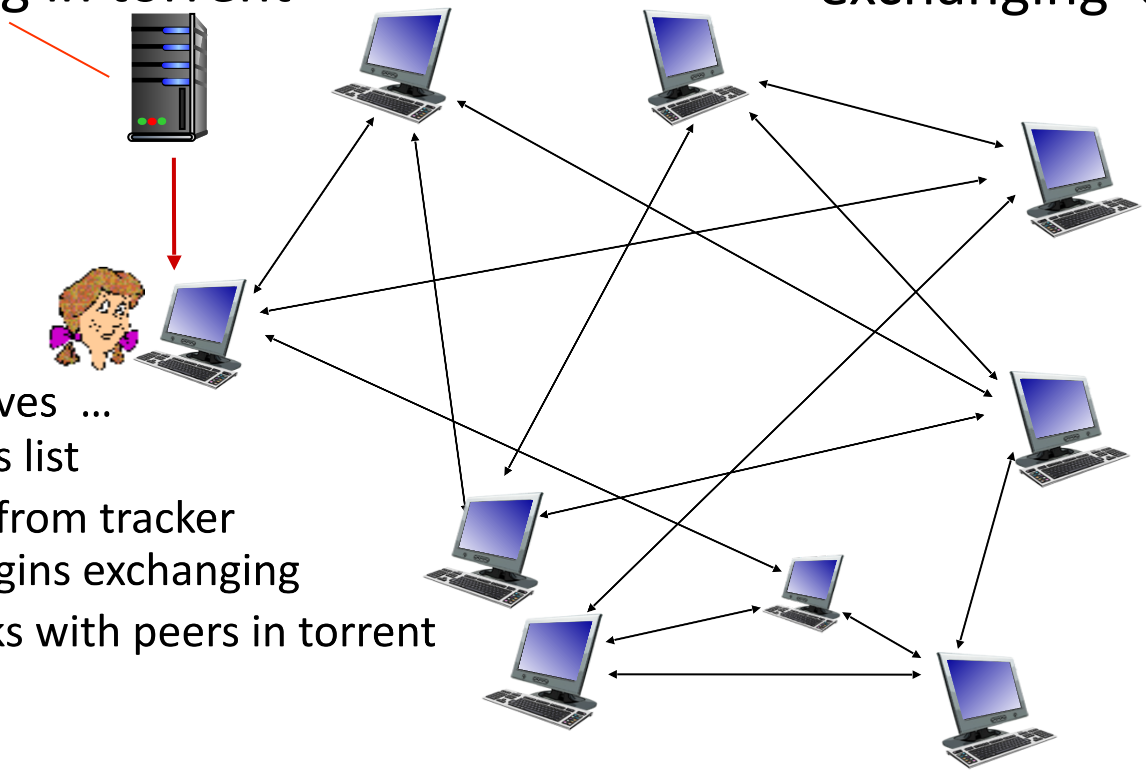
- A peer can transmit the entire file in one hour.
- The server transmission rate is 10 times the peer upload rate.
- Peer download rates are set large enough so as not to have an effect.

P2P file distribution: BitTorrent

- file divided into 256Kb chunks
- peers in torrent send/receive file chunks

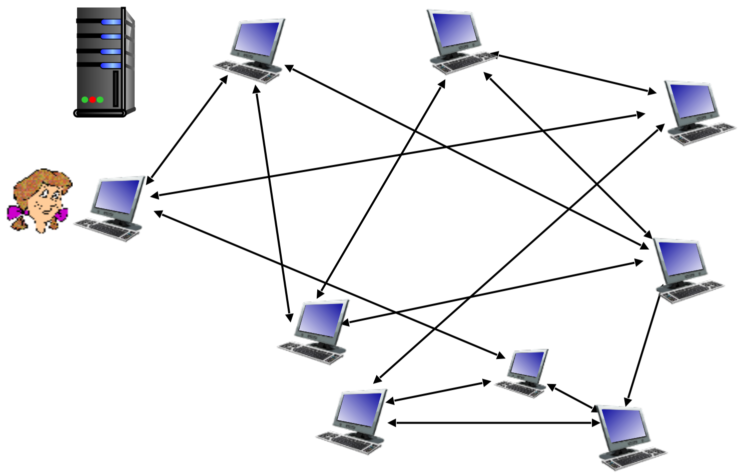
tracker: tracks peers participating in torrent

torrent: group of peers exchanging chunks of a file



Alice arrives …
… obtains list
of peers from tracker
… and begins exchanging
file chunks with peers in torrent

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) remain in torrent

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 every 10 secs
- every 30 secs: randomly select another peer, starts sending chunks
  - "**optimistically unchoke**" this peer
  - newly chosen peer may join top 4

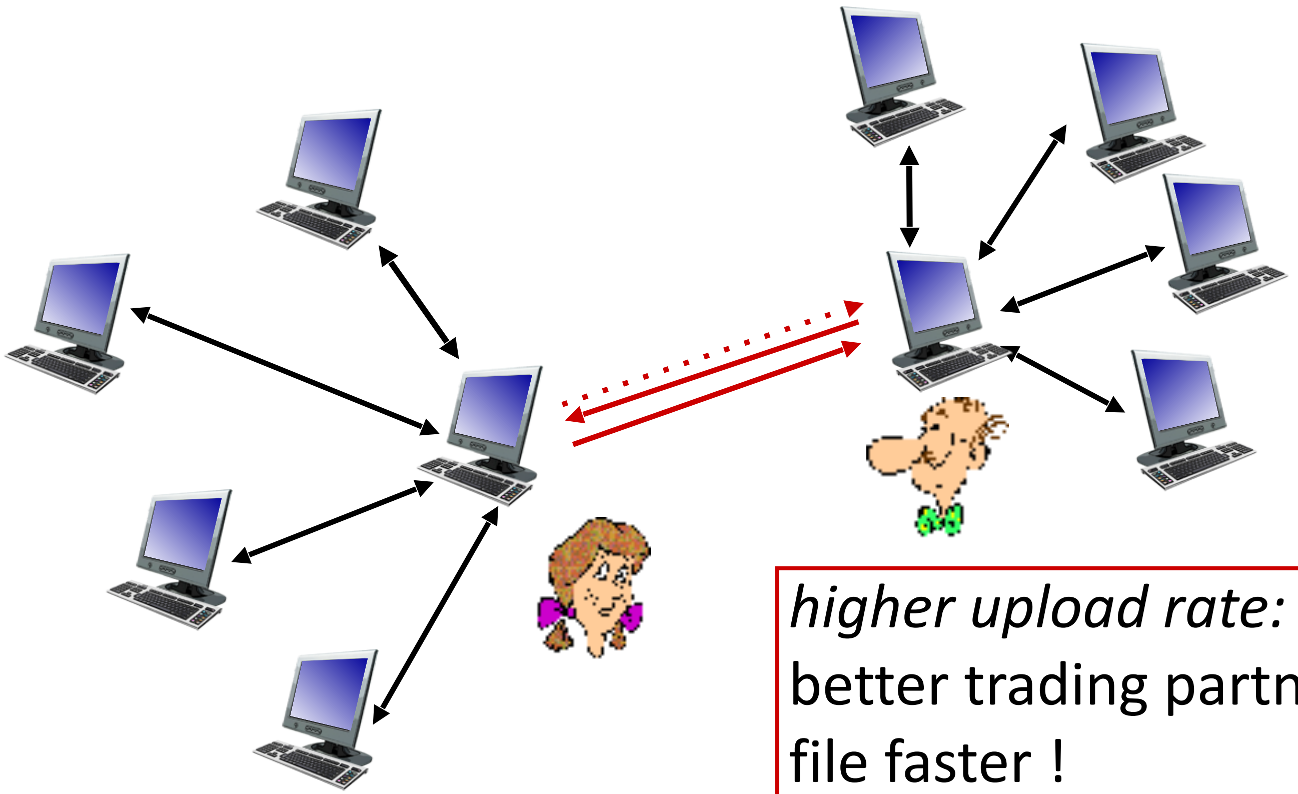
BitTorrent: tit-for-tat

(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

Pieces (mini-chunks), pipelining, random first selection, endgame mode, and anti-snubbing



higher upload rate: find better trading partners, get file faster !

Suggested Readings

- BitTorrent (BTT) White Paper – https://www.bittorrent.com/btt/btt-docs/BitTorrent_(BTT)_White_Paper_v0.8.7_Feb_2019.pdf
- Peer-to-peer networking with BitTorrent – http://web.cs.ucla.edu/classes/cs217/05BitTorrent.pdf
- Torrents Explained: How BitTorrent Works – https://youtu.be/urzQeD7ftbI

Q&A

Thank You
For Your Attention

Unit – 2 Application Layer

2.3 The Domain Name System

2.4 P2P Applications

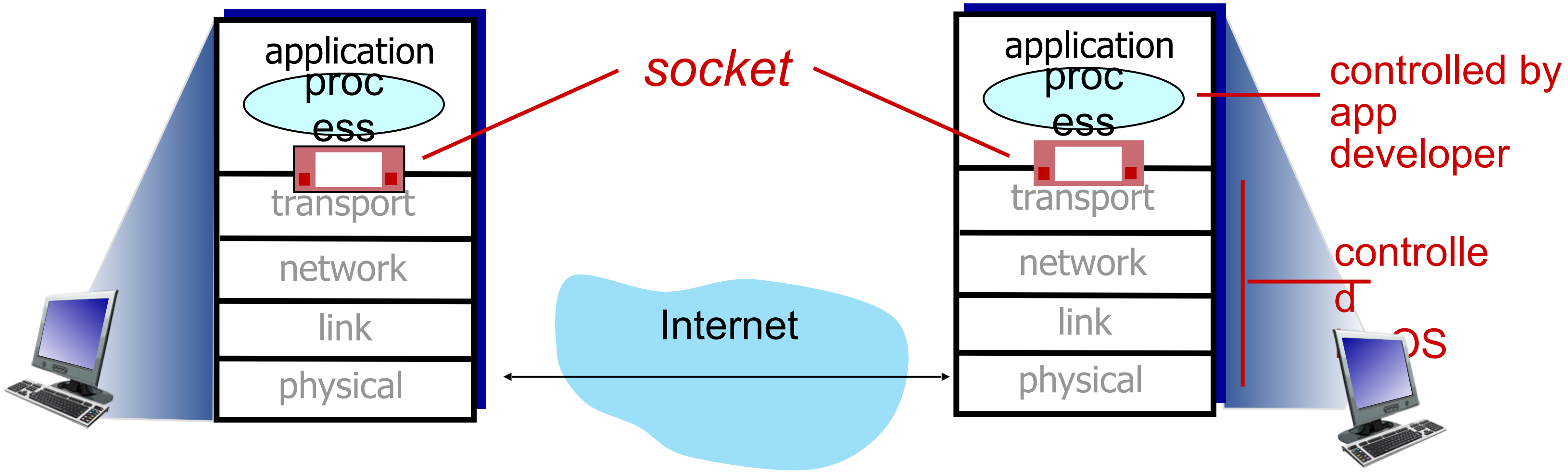
2.5 Socket Programming with TCP & UDP

2.6 Other Application Layer Protocols

Socket Programming

goal: learn how to build client/server applications that communicate using sockets

socket: door between application process and end-end-transport protocol



Two socket types for two transport services:

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

Application Example:

1. client reads a line of characters (data) from its keyboard and sends data to server
2. server receives the data and converts characters to uppercase
3. server sends modified data to client
4. client receives modified data and displays line on its screen

Socket Programming with UDP

PES UNIVERSITY
CELEBRATING 50 YEARS

UDP: no "connection" between client & server

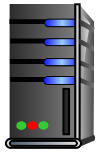
- no handshaking before sending data
- sender explicitly attaches IP destination address and port # to each packet
- receiver extracts sender IP address and port# from received packet

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

Client/Server socket interaction: UDP



server (running on serverIP)

create socket, port= x:
serverSocket =
socket(AF_INET,SOCK_DGRAM)

read datagram from
serverSocket

write reply to
serverSocket
specifying
client address,
port number

client

create socket:
clientSocket =
socket(AF_INET,SOCK_DGRAM)

Create datagram with server IP and
port=x; send datagram via
clientSocket

read datagram from
clientSocket

close
clientSocket

Example app: UDP client

Python UDPClient

include Python's socket library →
create UDP socket for server →
get user keyboard input →
attach server name, port to message; send into socket →
read reply characters from socket into string →
print out received string and close socket →

```
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()
```

Example app: UDP server

Python UDPServer

create UDP socket →

bind socket to local port number 12000 →

loop forever →

Read from UDP socket into message, getting client's address (client IP and port) →

send upper case string back to this client →

```
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)
```

Socket programming with TCP

Client must contact server

- server process must first be running
- server must have created socket (door) that welcomes client's contact

Client contacts server by:

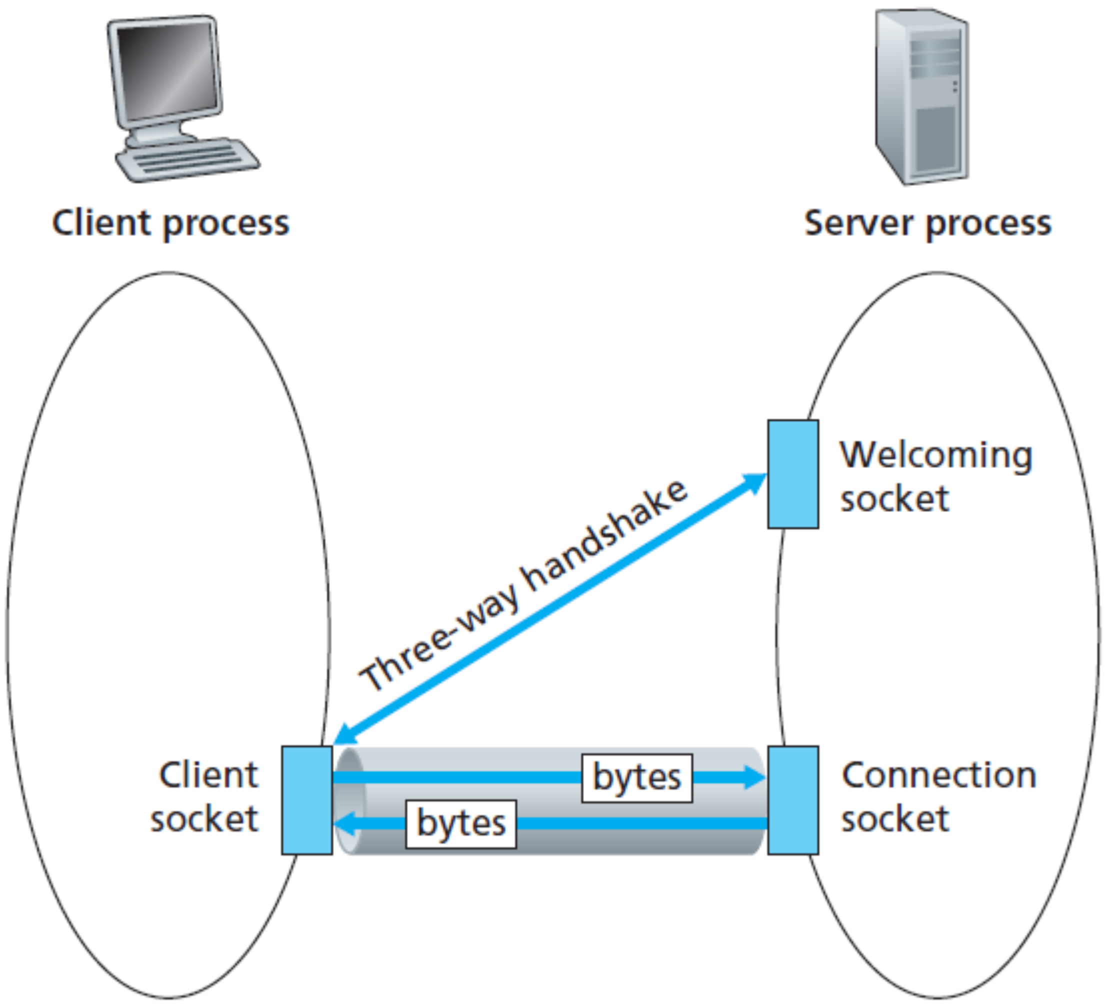
- Creating TCP socket, specifying IP address, port number of server process
- *when client creates socket:* client TCP establishes connection to server TCP

- when contacted by client, *server TCP creates new socket* for server process to communicate with that particular client
  - allows server to talk with multiple clients
  - source port numbers used to distinguish clients (more in Chap 3)

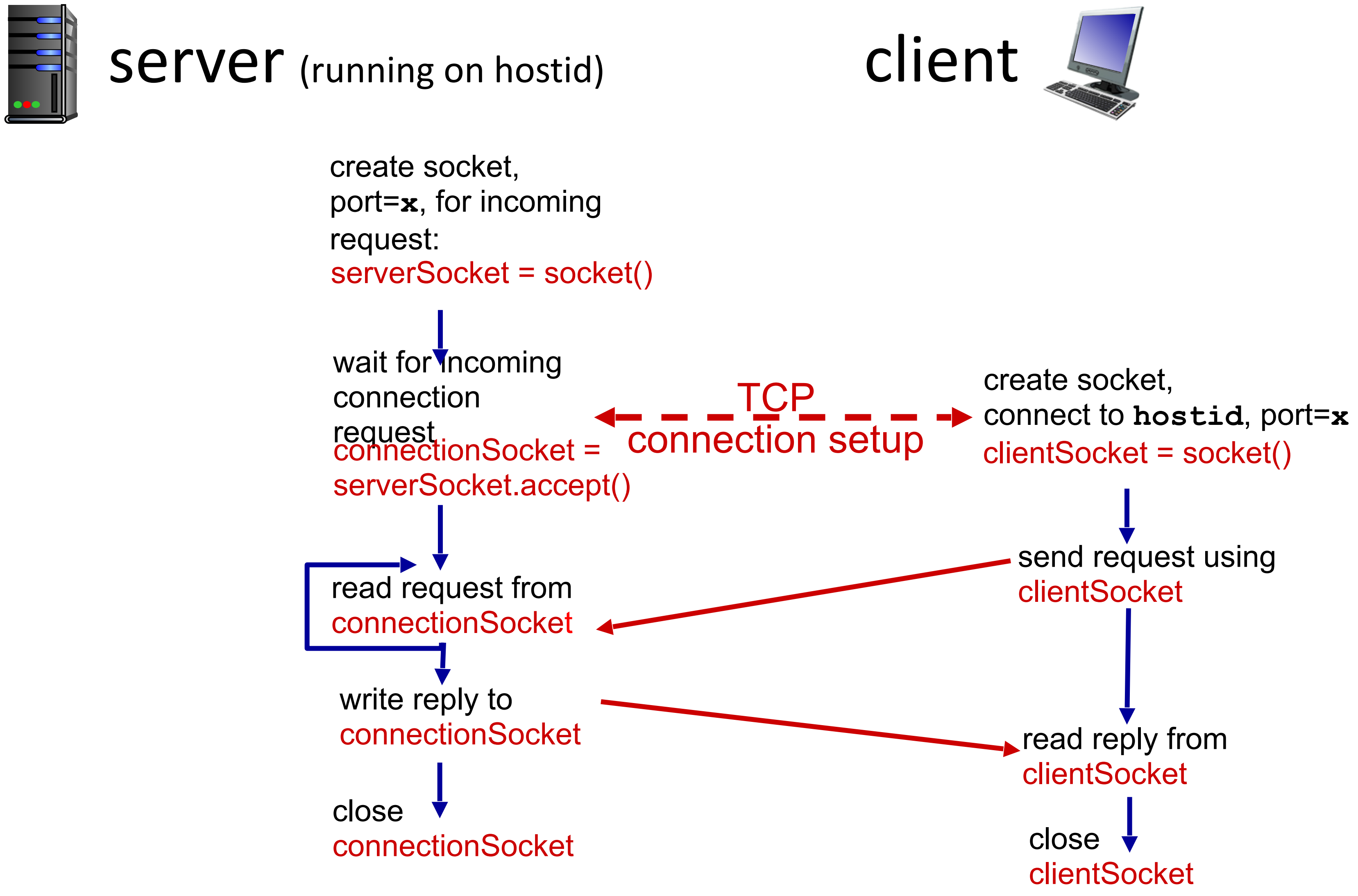
Application viewpoint

TCP provides reliable, in-order byte-stream transfer ("pipe") between client and server

The TCPServer Process has Two Sockets



Client/server socket interaction: TCP



Example app: TCP client

PES UNIVERSITY
CELEBRATING 50 YEARS

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

Example app: TCP server

Python TCPServer

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)

```
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()
```

Unit – 2 Application Layer

2.3 The Domain Name System

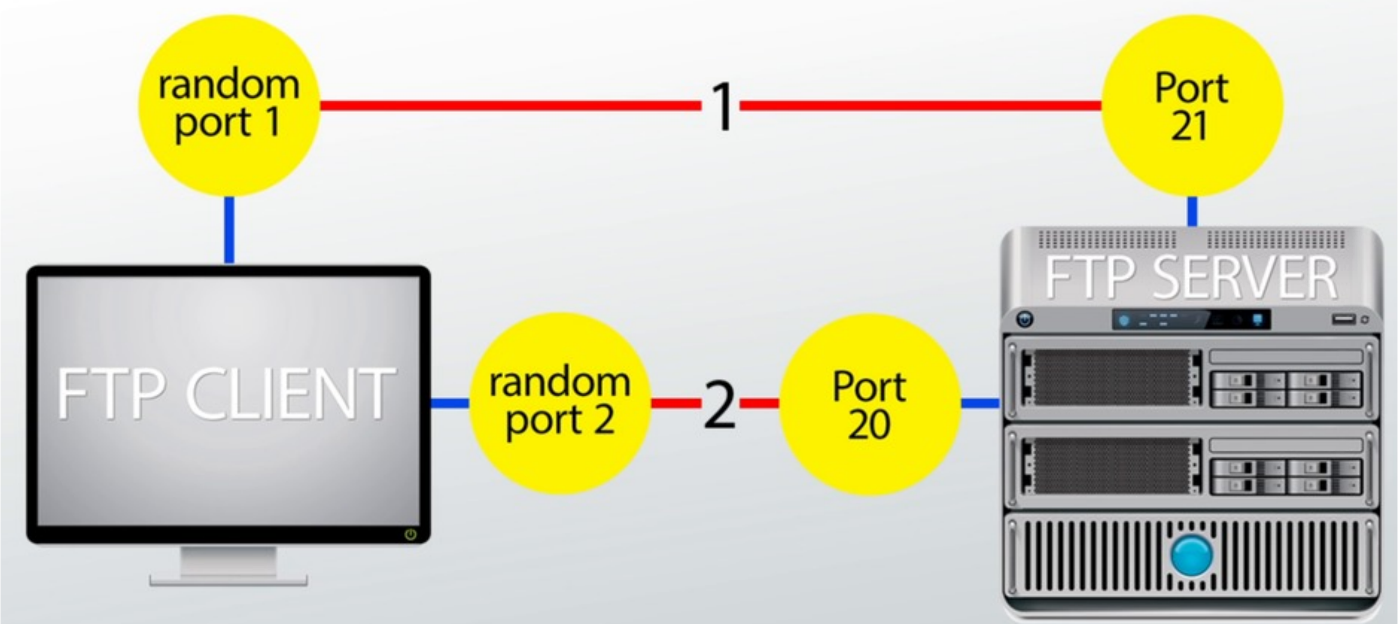
2.4 P2P Applications

2.5 Socket Programming with TCP & UDP

2.6 Other Application Layer Protocols

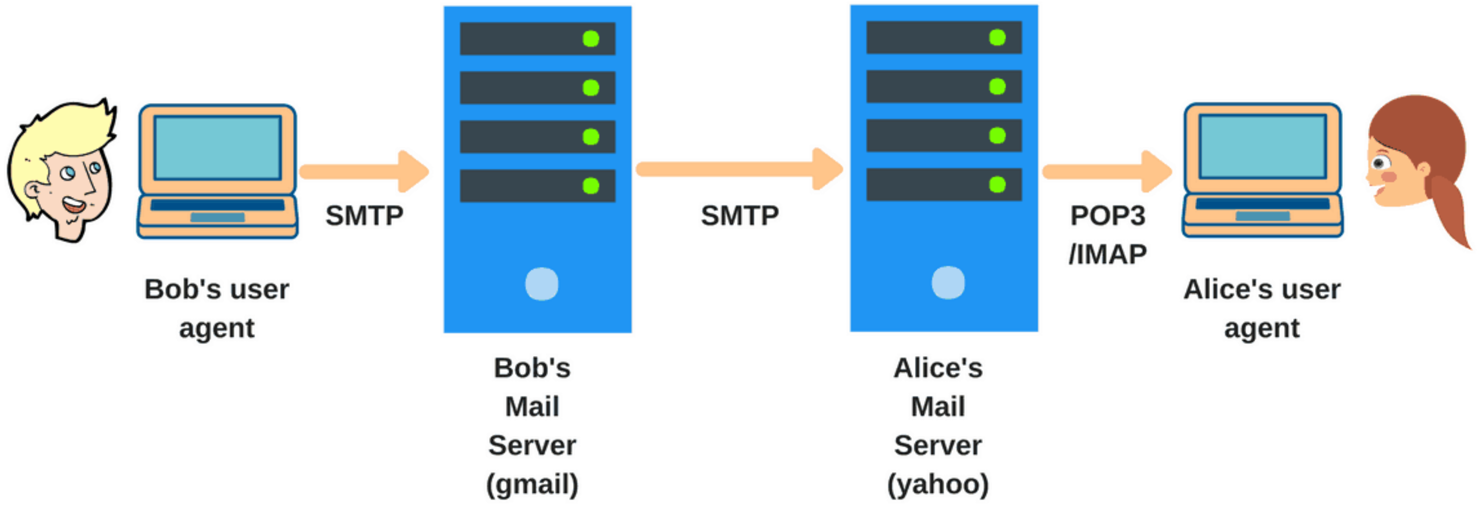
Other Application Layer Protocols - FTP

- File Transfer Protocol (FTP) - used to exchange large files on the internet TCP
- Invoked from the command prompt or some GUI.
- Allows to update (delete, rename, move, and copy) files at a server.
- Data connection (Port No. 20) & Control connection (Port No. 21)



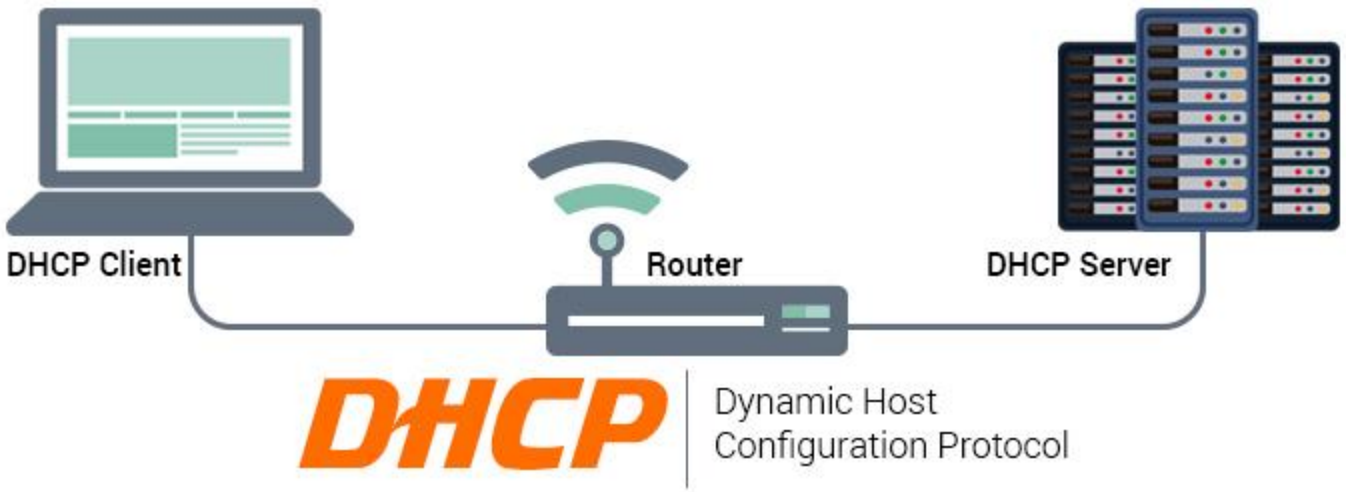
Other Application Layer Protocols - SMTP

- **Simple Mail Transfer Protocol** - an internet standard for e-mail Transmission.
- Connections are secured with SSL (Secure Socket Layer).
- Messages are stored and then forwarded to the destination (relay).
- SMTP uses a port number 25 of TCP.



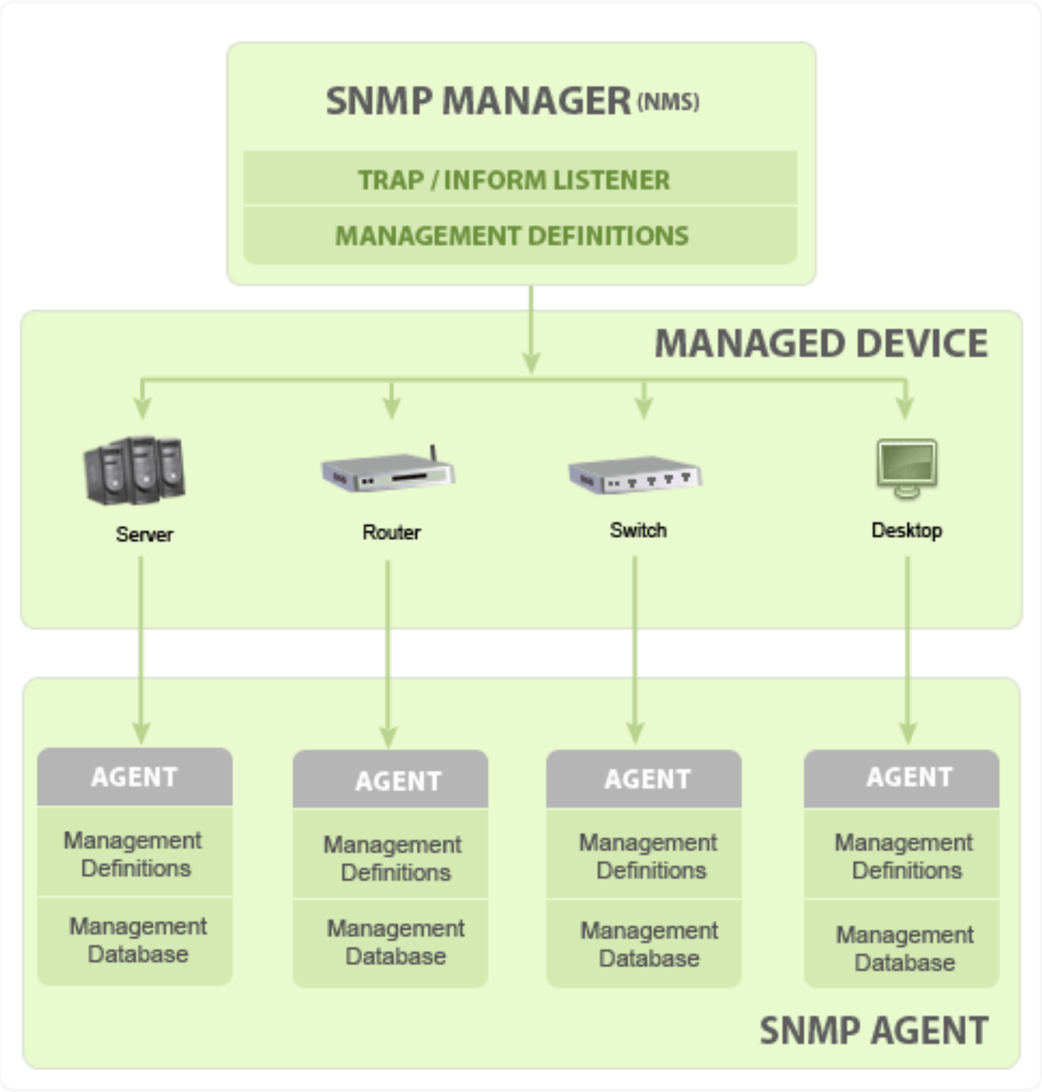
Other Application Layer Protocols - DHCP

- **Dynamic Host Configuration Protocol** - assign IP addresses to computers in a network dynamically.
- IP addresses may change even when computer is in network (DHCP leases).
- DHCP port number for server is 67 and for the client is 68.
- A client-server model & based on **discovery, offer, request, and ACK**.
- Includes subnet mask, DNS server address, default gateway



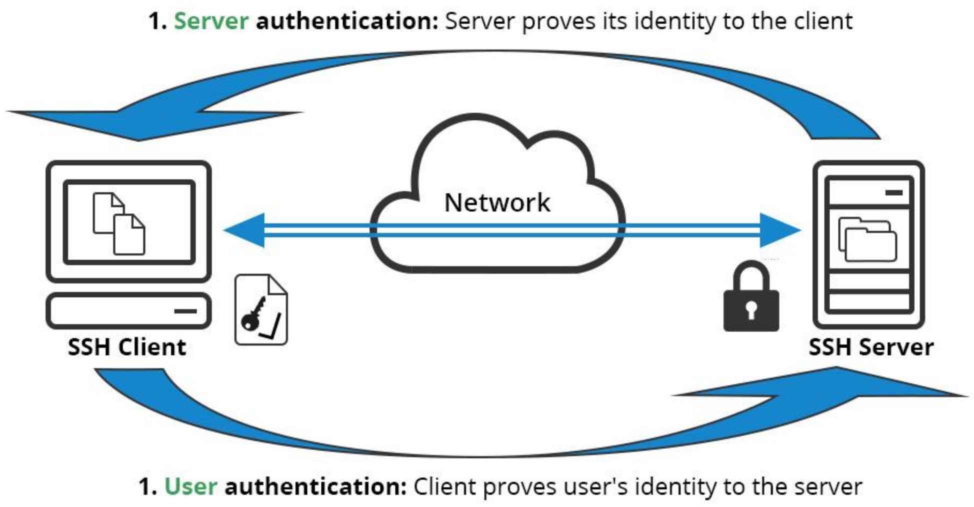
Other Application Layer Protocols - SNMP

- **Simple Network Management Protocol** - exchange management information between network devices.
- Basic components & functionalities
  - SNMP Manager
  - Managed Devices
  - SNMP Agents
  - MIB (Management Information Base)



Other Application Layer Protocols – Telnet & SSH

- Allows a user to communicate with a remote device.
- Used mostly by network admin to remotely access and manage devices.
- Telnet client and server installed – uses TCP port no. 23
- SSH – uses public key **encryption** & TCP port 22 by default.



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

PES UNIVERSITY
CELEBRATING 50 YEARS

Summary

our study of network application layer is now complete!

- application architectures
  - client-server
  - P2P
- application service requirements:
  - reliability, bandwidth, delay
- Internet transport service model
  - connection-oriented, reliable: TCP
  - unreliable, datagrams: UDP
- specific protocols:
  - HTTP
  - DNS
  - P2P: BitTorrent
- socket programming: TCP, UDP sockets

Summary (more)

Most importantly: learned about *protocols*!

- typical request/reply message exchange:
  - client requests info or service
  - server responds with data, status code
- message formats:
  - *headers*: fields giving info about data
  - *data:* info(payload) being communicated

important themes:

- centralized vs. decentralized
- stateless vs. stateful
- scalability
- reliable vs. unreliable message transfer
- “complexity at network edge”

THANK YOU

TEAM NETWORKS

Department of Computer Science and Engineering