

Lecture 2: Application

SMTP DNS Others

05506015 Data Communication and Computer Networks

Dr. Rungrat Wiangsripanawan

Outline

- Tip: What is netmask?
- E-mail, SMTP, IMAP
- The Domain Name System
- Lab: DNS
- Group Assignment (2 weeks time)
UDP/TCP Socket

เนื้อหาใน *Slide* นำมาจาก *Slide* บทที่ 2 ของ หนังสือ
*Computer Networking: A Top-Down
Approach*

8th edition

Jim Kurose, Keith Ross

Pearson, 2020

First Question

Email, SMTP, IMAP

Sample SMTP interaction

```
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
```

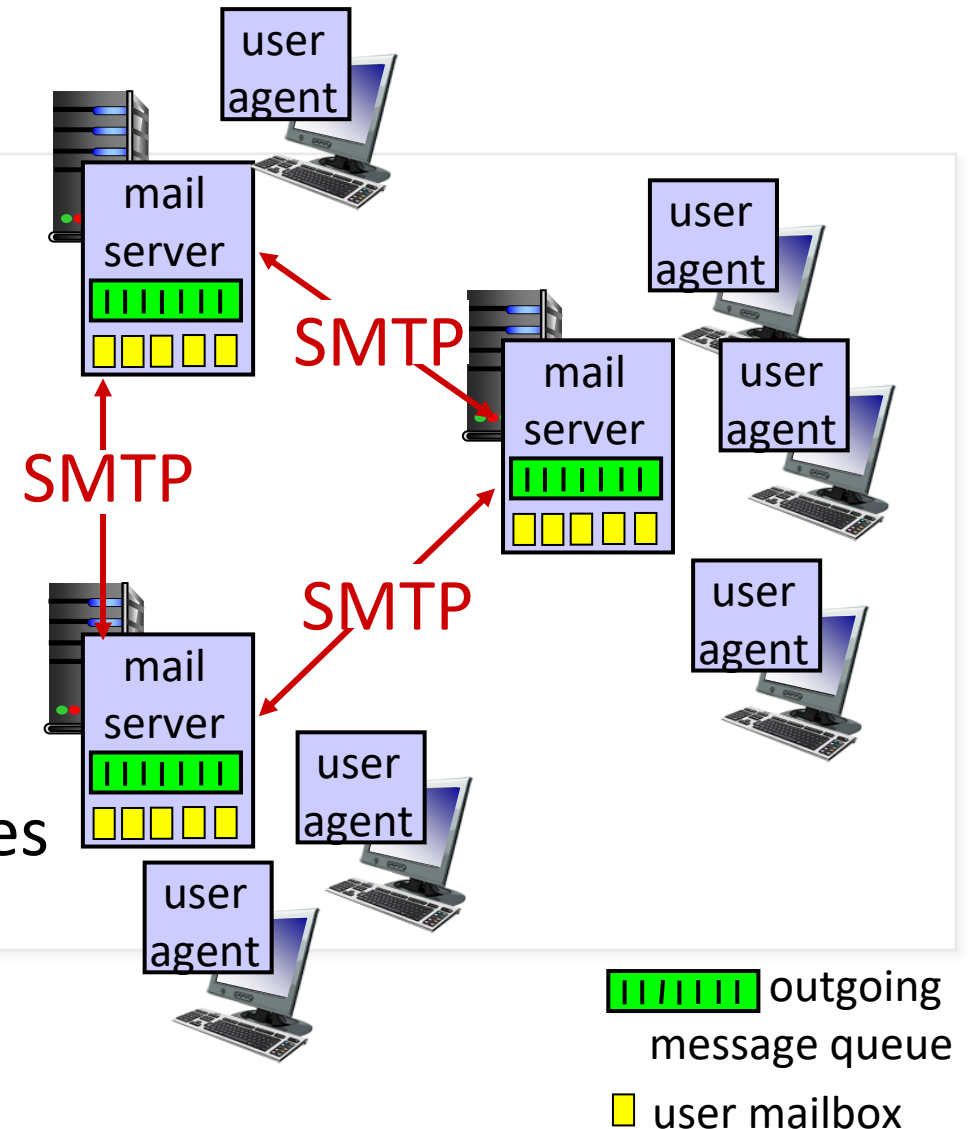
E-mail

Three major components:

- user agents
- mail servers
- simple mail transfer protocol: SMTP

User Agent

- a.k.a. “mail reader”
- composing, editing, reading mail messages
- e.g., Outlook, iPhone mail client
- outgoing, incoming messages stored on server



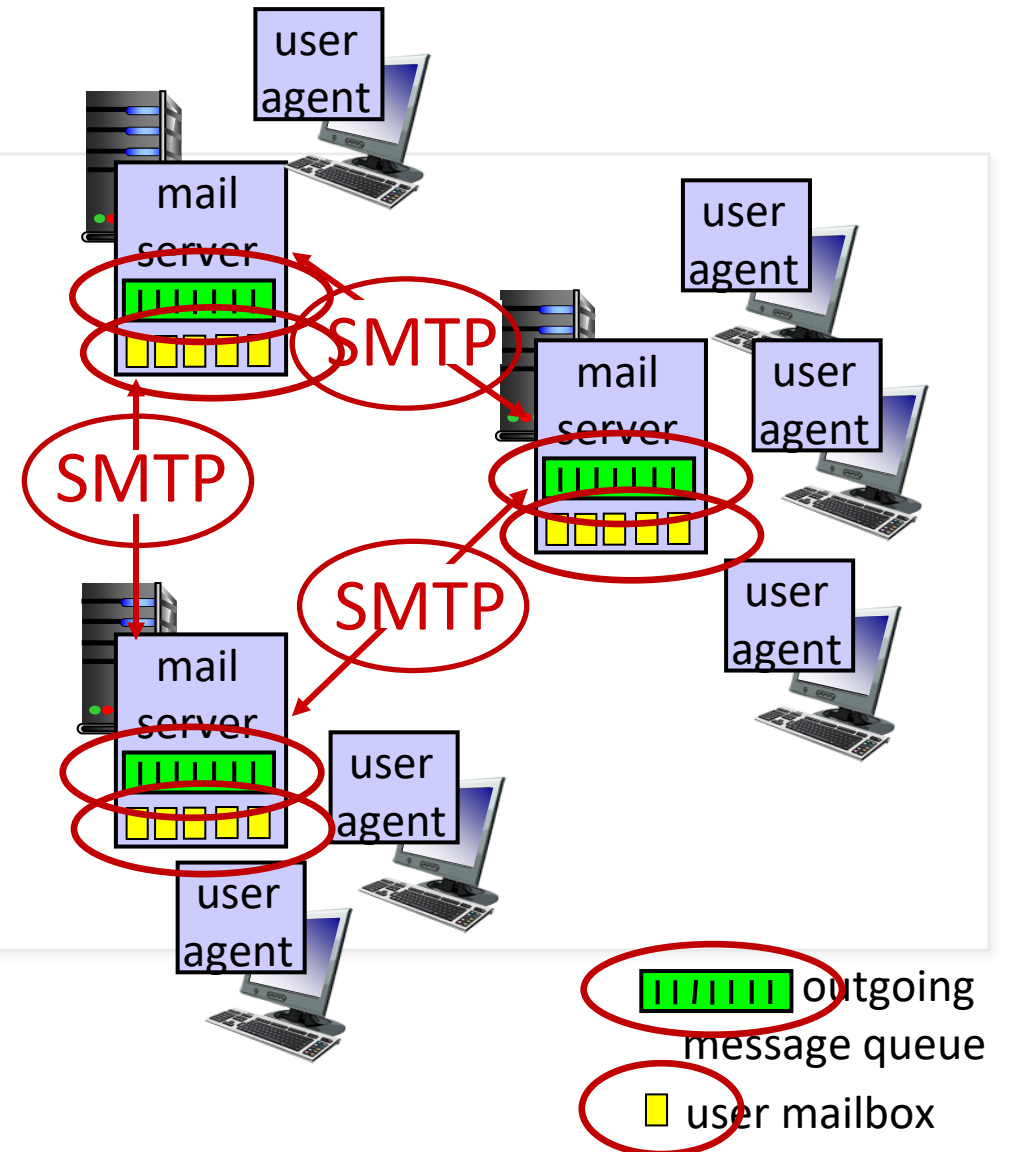
E-mail: mail servers

mail servers:

- *mailbox* contains incoming messages for user
- *message queue* of outgoing (to be sent) mail messages

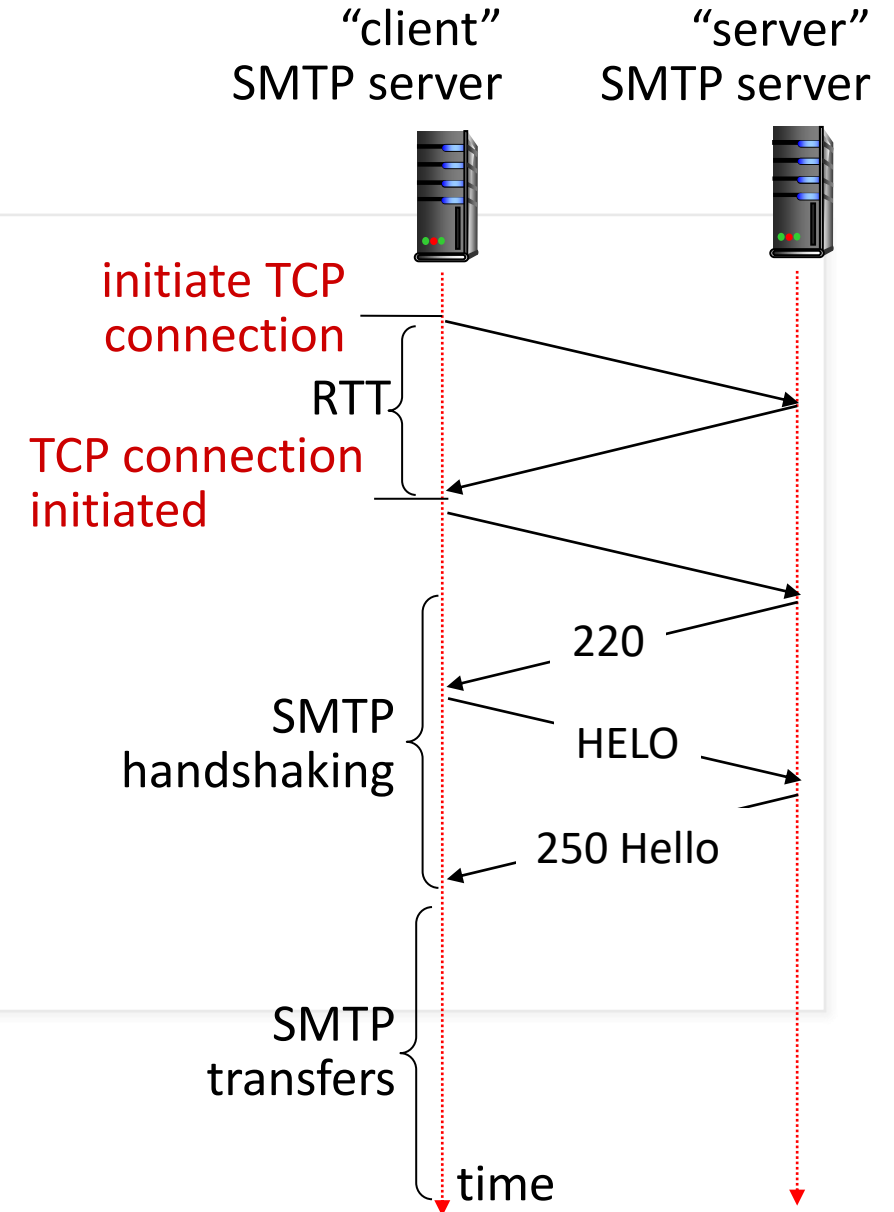
SMTP protocol between mail servers to send email messages

- **client**: sending mail server
- **“server”**: receiving mail server



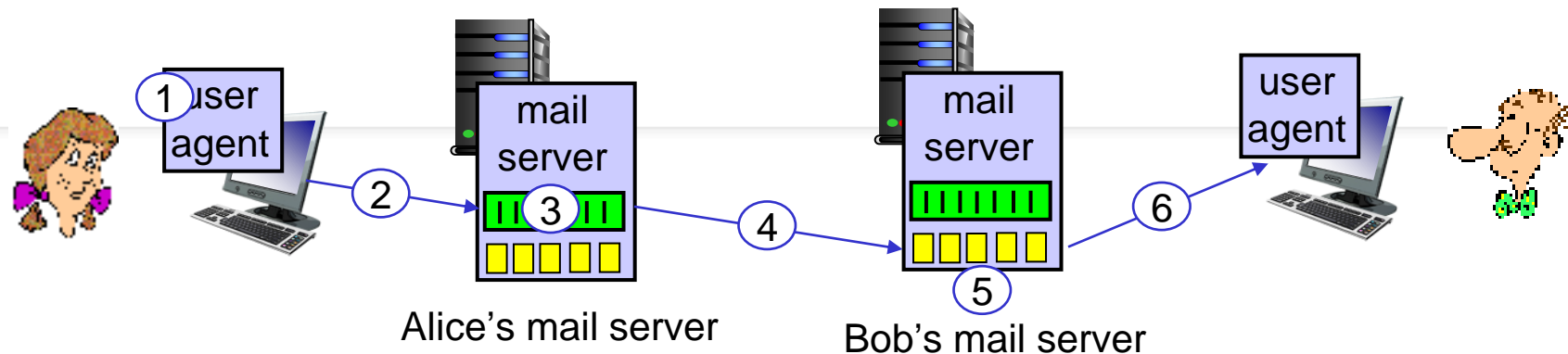
SMTP RFC (5321)

- uses TCP to reliably transfer email message from client (mail server initiating connection) to server, port 25
 - direct transfer: sending server (acting like client) to receiving server
- three phases of transfer
 - SMTP handshaking (greeting)
 - SMTP transfer of messages
 - SMTP closure
- command/response interaction (like HTTP)
 - **commands:** ASCII text
 - **response:** status code and phrase



Scenario: Alice sends e-mail to Bob

- 1) Alice uses UA to compose e-mail message "to" bob@some school.edu
- 2) Alice's UA sends message to her mail server using SMTP; message placed in message queue
- 3) client side of SMTP at mail server opens TCP connection with Bob's mail server
- 4) SMTP client sends Alice's message over the TCP connection
- 5) Bob's mail server places the message in Bob's mailbox
- 6) Bob invokes his user agent to read message



Sample SMTP interaction

S: 220 hamburger.edu

SMTP: observations

comparison with HTTP:

- HTTP: client pull
- SMTP: client push
- both have ASCII command/response interaction, status codes
- HTTP: each object encapsulated in its own response message
- SMTP: multiple objects sent in multipart message
- SMTP uses persistent connections
- SMTP requires message (header & body) to be in 7-bit ASCII
- SMTP server uses CRLF.CRLF to determine end of message

Mail message format

SMTP: protocol for exchanging e-mail messages, defined in RFC 5321 (like RFC 7231 defines HTTP)

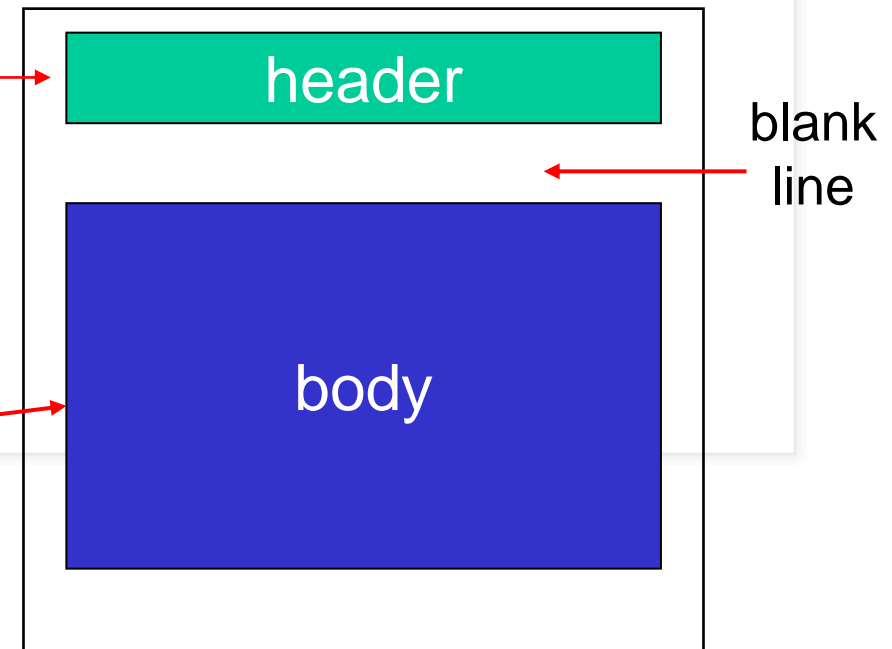
RFC 2822 defines *syntax* for e-mail message itself (like HTML defines syntax for web documents)

- header lines, e.g.,

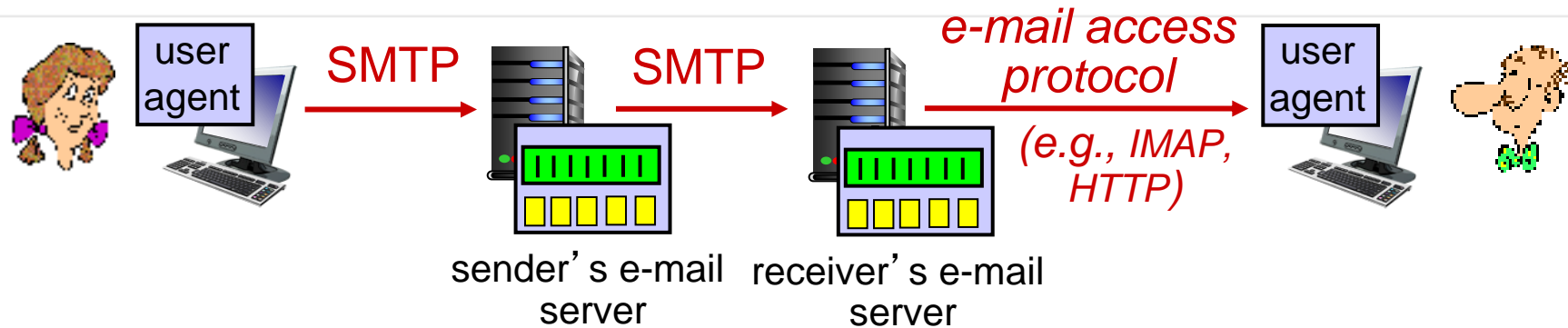
- To:
- From:
- Subject:

these lines, within the body of the email message area different from SMTP MAIL FROM:, RCPT TO: commands!

- Body: the “message” , ASCII characters only



Retrieving email: mail access protocols



- **SMTP:** delivery/storage of e-mail messages to receiver's server
- mail access protocol: retrieval from server
 - **IMAP:** Internet Mail Access Protocol [RFC 3501]: messages stored on server, IMAP provides retrieval, deletion, folders of stored messages on server
- **HTTP:** gmail, Hotmail, Yahoo!Mail, etc. provides web-based interface on top of SMTP (to send), IMAP (or POP) to retrieve e-mail messages



First Question

The Domain Name System (DNS)

DNS: Domain Name System

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 (DNS):

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



DNS: Note

DNS: services, structure

DNS services:

- hostname-to-IP-address translation
- host aliasing
 - canonical, alias names
- mail server aliasing
- load distribution
 - replicated Web servers: many IP addresses correspond to one name

Q: Why not centralize DNS?

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

A: doesn't scale!

- Comcast DNS servers alone: 600B DNS queries/day
- Akamai DNS servers alone: 2.2T DNS queries/day

Thinking about the DNS

humongous distributed database:

- ~ billion records, each simple

handles many *trillions* of queries/day:

- *many* more reads than writes
- *performance matters*: almost every Internet transaction interacts with DNS - msec count!

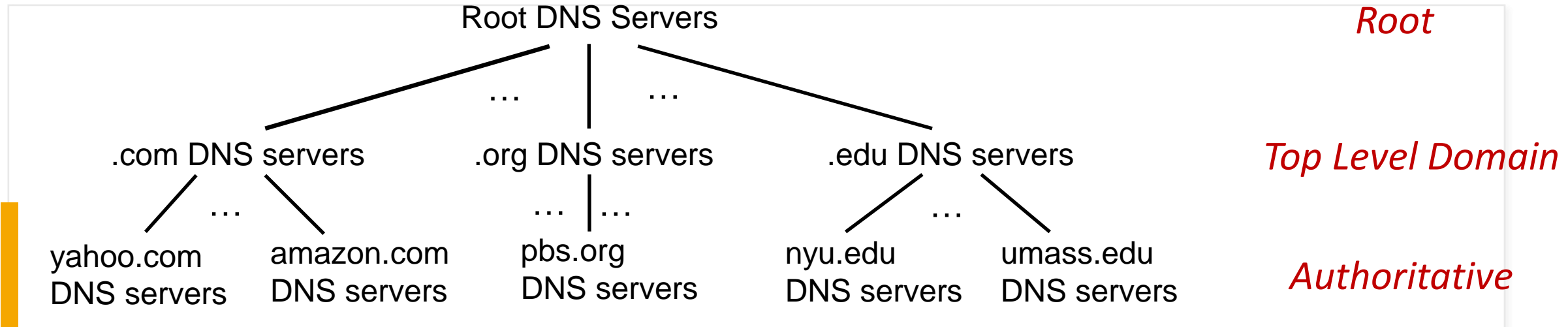
organizationally, physically decentralized:

- millions of different organizations responsible for their records

“bulletproof”: reliability, security



DNS: a distributed, hierarchical database

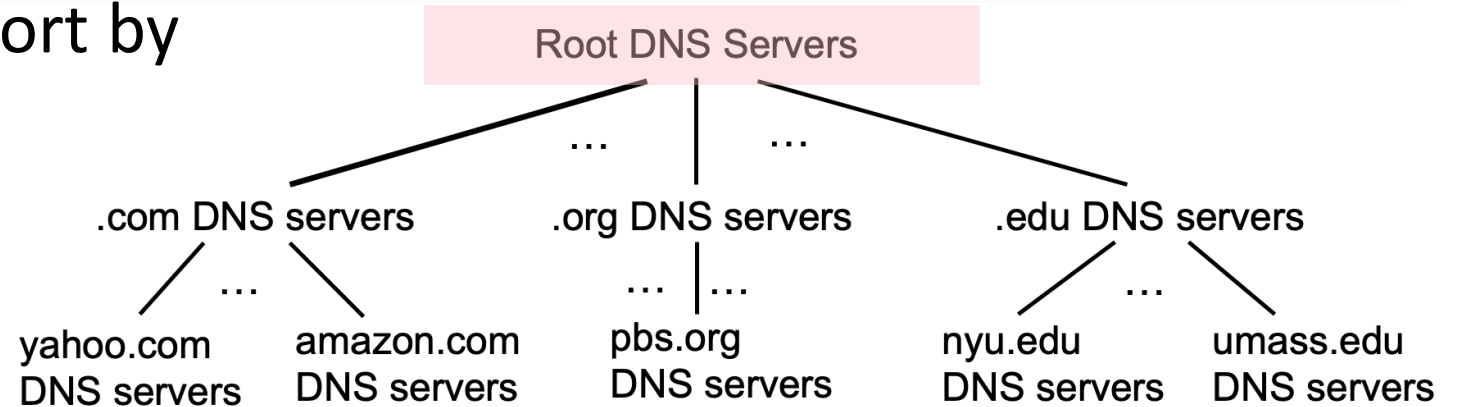


Client wants IP address for www.amazon.com; 1st approximation:

- 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: root Name Servers

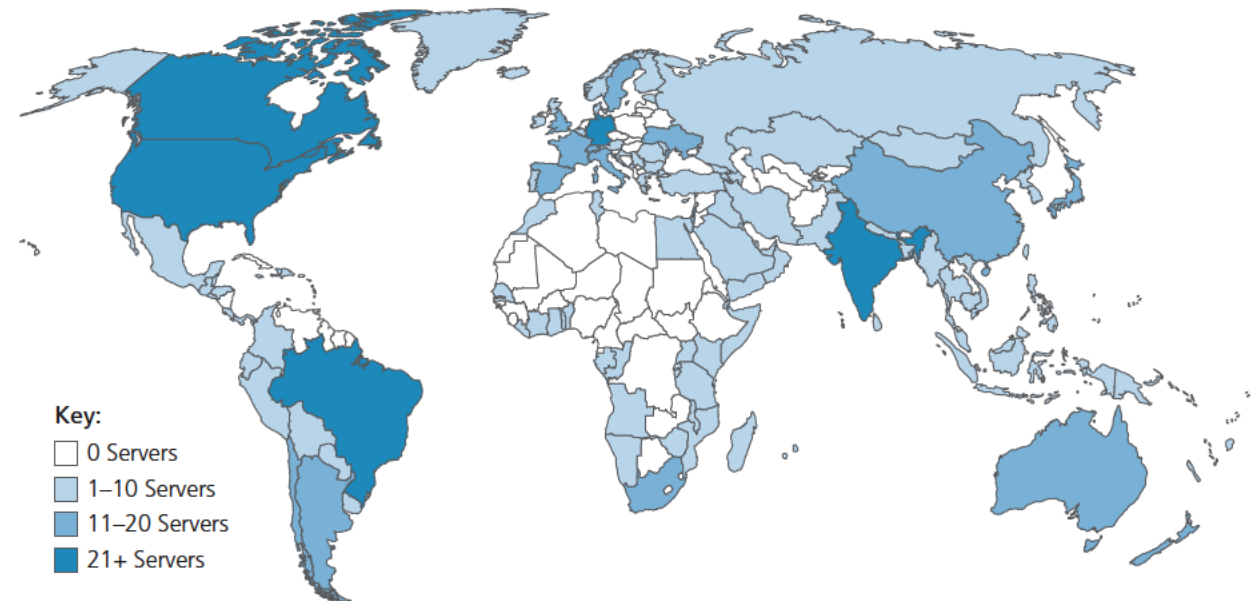
- official, contact-of-last-resort by name servers that can not resolve name



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, message integrity)
- ICANN (Internet Corporation for Assigned Names and Numbers) manages root DNS domain

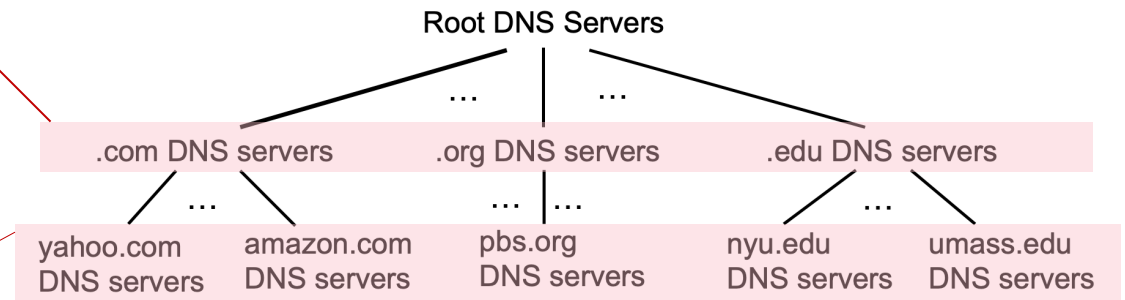
13 logical root name “servers” worldwide each “server” replicated many times (~200 servers in US)



Top-Level Domain, and 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

- when host makes DNS query, it is sent to its *local* DNS server
 - Local DNS server returns reply, answering:
 - from its local cache of recent name-to-address translation pairs (possibly out of date!)
 - forwarding request into DNS hierarchy for resolution
 - each ISP has local DNS name server; to find yours:
 - MacOS: `% scutil --dns`
 - Windows: `>ipconfig /all`
- local DNS server doesn't strictly belong to hierarchy

Exercise: Capture your local DNS Server and send Chat:

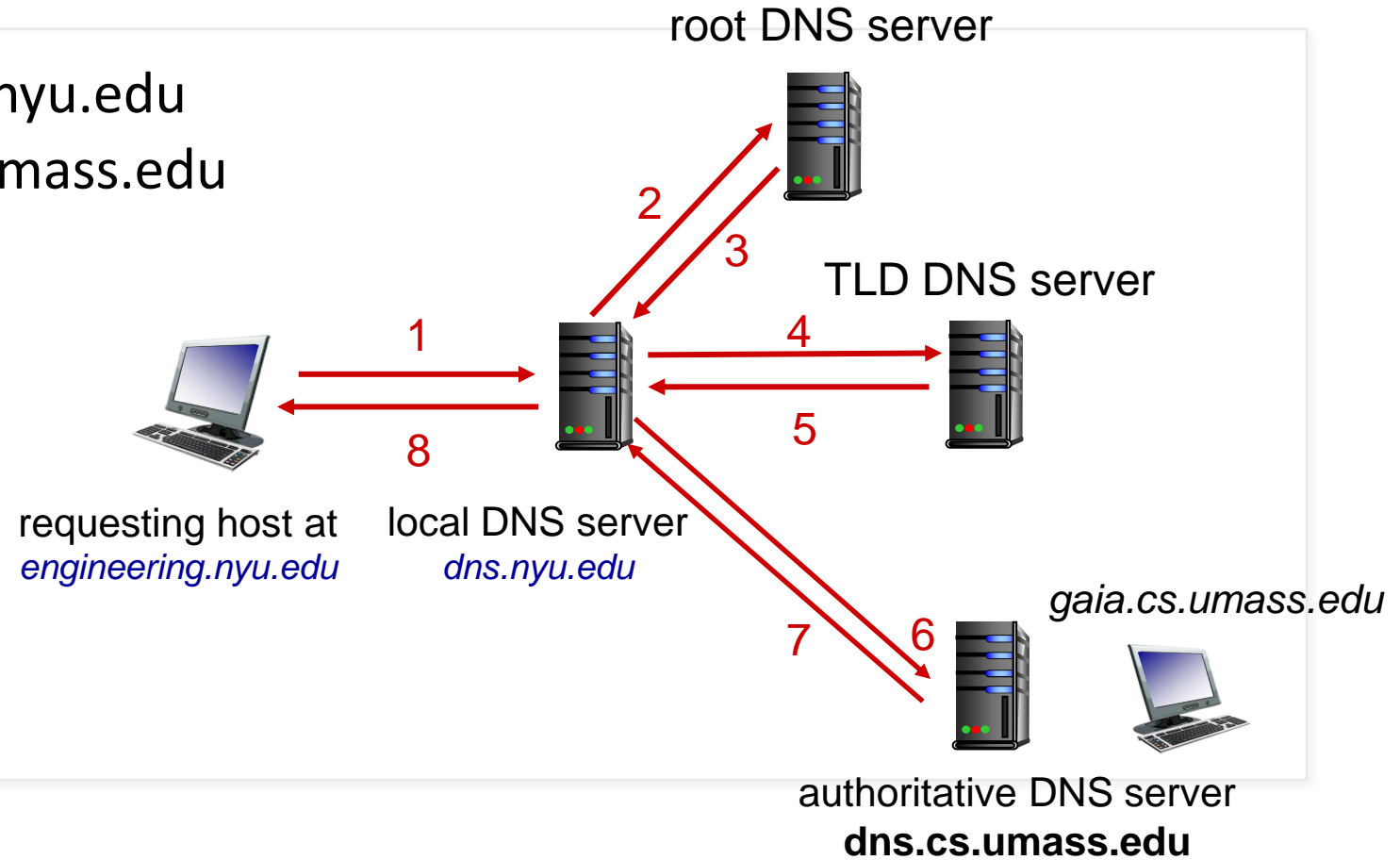
1. Capture screen to show what is your DNS Server IP.
2. If the Local DNS Server is in the same network as your IP Address. How do you know?

DNS name resolution: iterated query

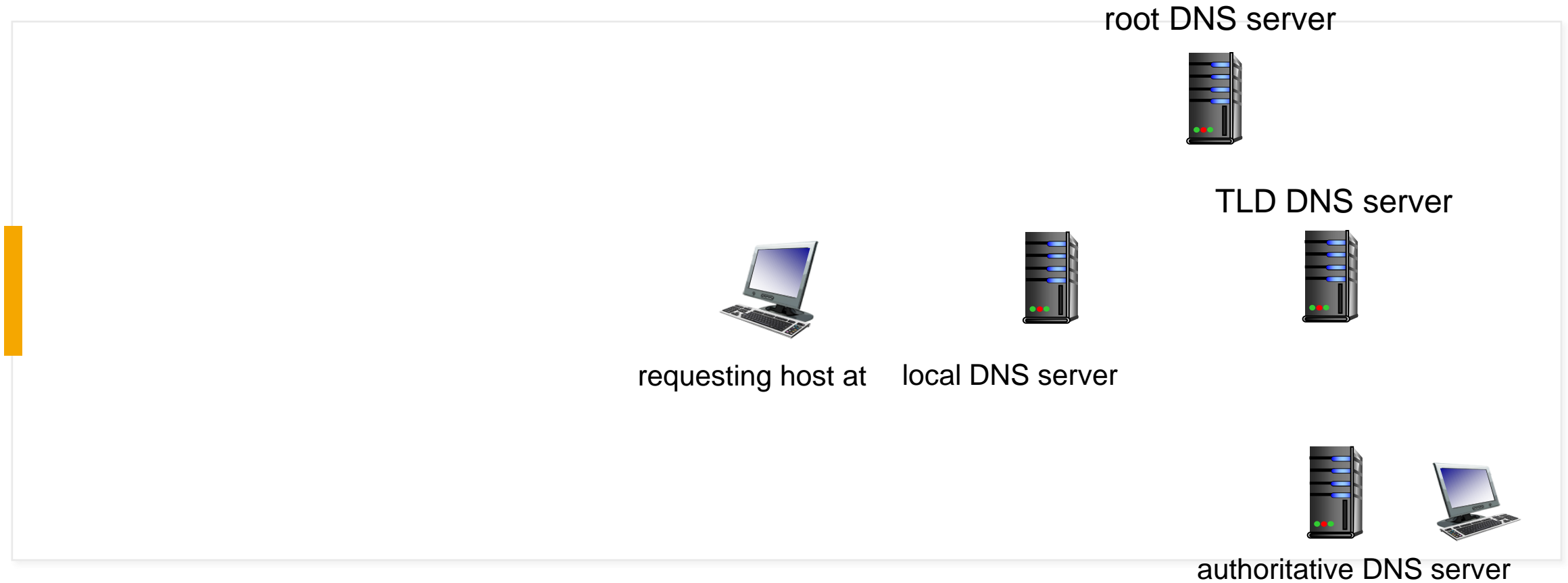
Example: host at `engineering.nyu.edu` wants IP address for `gaia.cs.umass.edu`

Iterated query:

- contacted server replies with name of server to contact
- “I don’t know this name, but ask this server”



DNS name resolution: iterated query

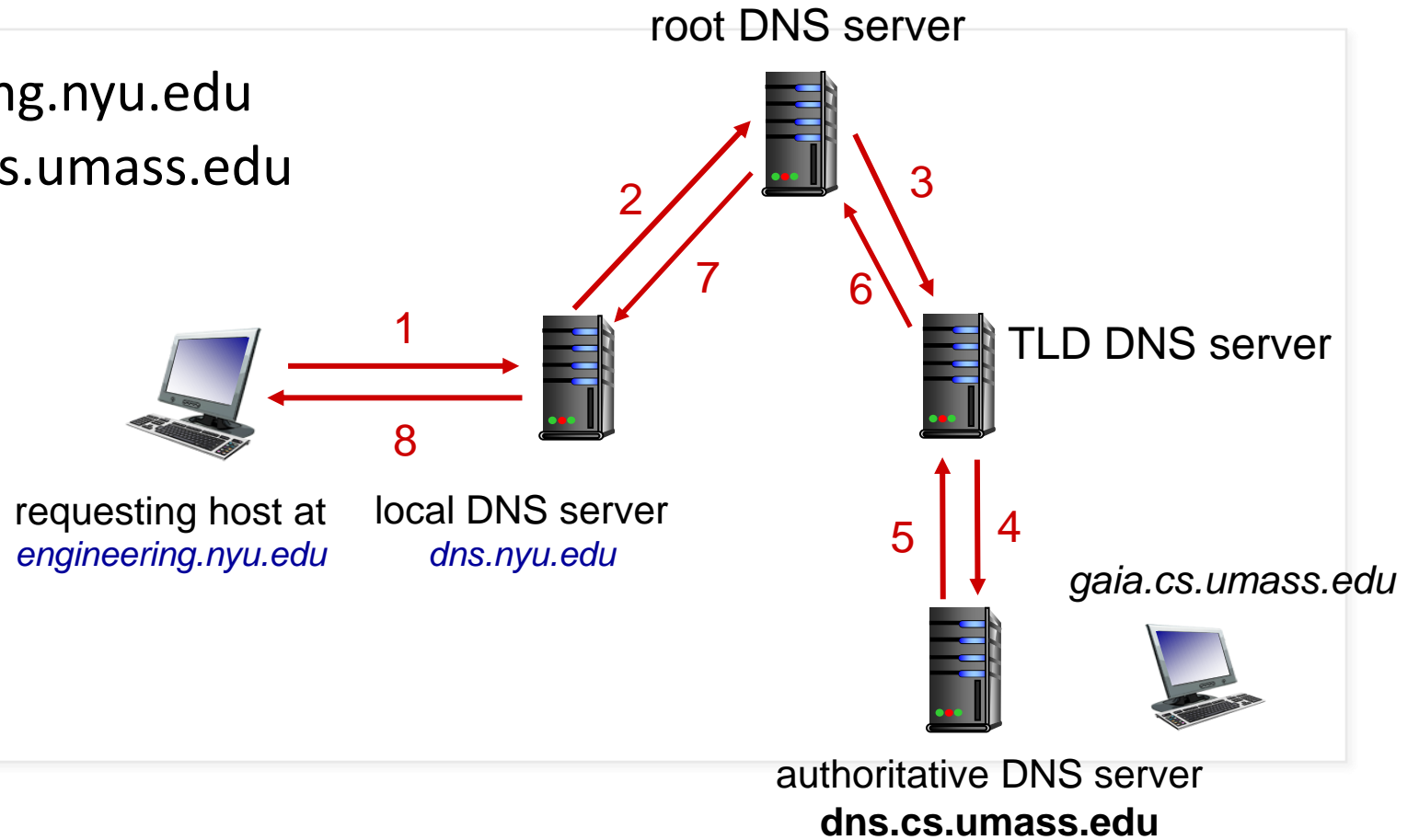


DNS name resolution: recursive query

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 DNS Information

- once (any) name server learns mapping, it *cached* mapping, and *immediately* returns a cached mapping in response to a query
 - caching improves response time
 - cache entries timeout (disappear) after some time (TTL)
 - TLD servers typically cached in local name servers
- cached entries may be *out-of-date*
 - if named host changes IP address, may not be known Internet-wide until all TTLs expire!
 - *best-effort name-to-address translation!*

DNS records

DNS: distributed database storing resource records (RR)

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

type=A

- name is hostname
- value is IP address

type=NS

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

type=CNAME

- name is alias name for some “canonical” (the real) name
- www.ibm.com is really servereast.backup2.ibm.com
- value is canonical name

type=MX

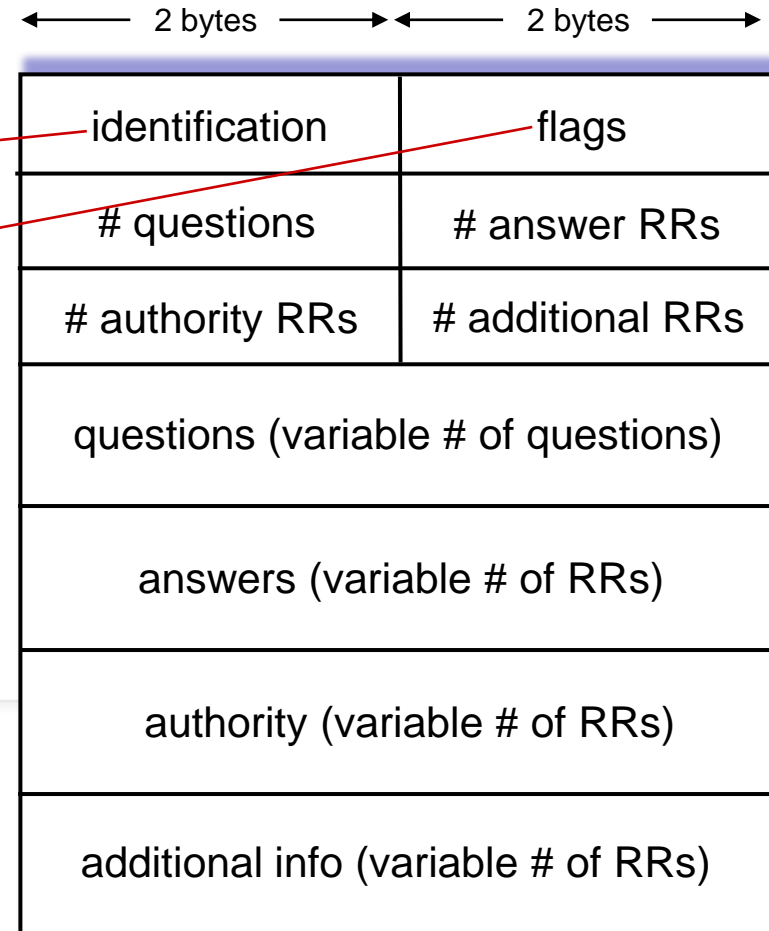
- value is name of SMTP mail server associated with name

DNS protocol messages

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
 - recursion desired
 - recursion available
 - reply is authoritative



DNS protocol messages

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

← 2 bytes → ← 2 bytes →

identification	flags
# questions	# answer RRs
# authority RRs	# additional RRs
questions (variable # of questions)	
answers (variable # of RRs)	
authority (variable # of RRs)	
additional info (variable # of RRs)	

name, type fields for a query

RRs in response to query

records for authoritative servers

additional “helpful” info that may
be used

Getting your info into the DNS

example: new startup “Network Utopia”

- register name networkutopia.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.networkutopia.com`
 - type MX record for `networkutopia.com`

DNS security

DDoS attacks

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

Spoofing attacks

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

DNS Lab exercise



- Go to http://www-net.cs.umass.edu/wireshark-labs/Wireshark_DNS_v8.0.pdf
- Type
 - nslookup [ชื่อ Host+Domain name]
 - nslookup -type=NS ชื่อ Domain
 - nslookup [ชื่อ Host+Domain name] Specific DNS Host

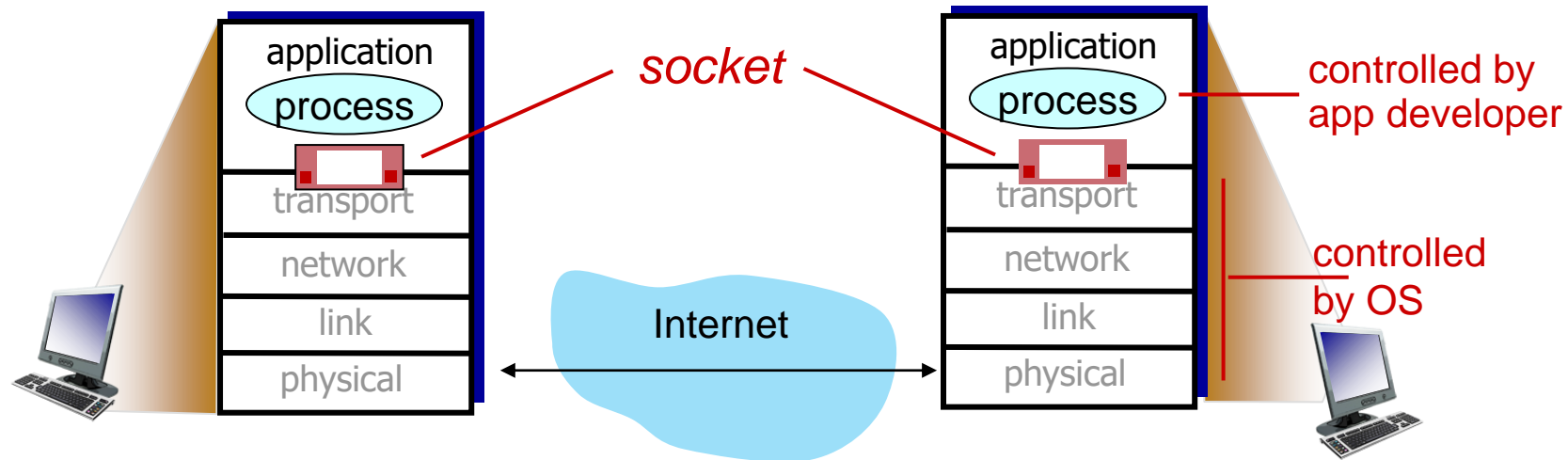
UDP/TCP Socket



Socket programming

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

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



Socket programming

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

UDP: no “connection” between client and server:

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

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

- receiver extracts sender IP address

Application viewpoint:

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

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 serverIP address
And port=x; send datagram via
clientSocket

read datagram from
clientSocket

close
clientSocket

Example app: UDP client

Python UDPClient

include Python's socket library → `from socket import *`
`serverName = 'hostname'`
`serverPort = 12000`
create UDP socket for server → `clientSocket = socket(AF_INET,
SOCK_DGRAM)`
get user keyboard input → `message = raw_input('Input lowercase sentence:')`
attach server name, port to message; send into socket → `clientSocket.sendto(message.encode(),
(serverName, serverPort))`
read reply characters from socket into string → `modifiedMessage, serverAddress =
clientSocket.recvfrom(2048)`
print out received string and close socket → `print modifiedMessage.decode()
clientSocket.close()`

Example app: UDP server

Python UDPServer

```
from socket import *
serverPort = 12000
create UDP socket → serverSocket = socket(AF_INET, SOCK_DGRAM)
bind socket to local port number 12000 → serverSocket.bind(('', serverPort))
print ("The server is ready to receive")
loop forever → while True:
    Read from UDP socket into message, getting client's address (client IP and port) → message, clientAddress = serverSocket.recvfrom(2048)
    modifiedMessage = message.decode().upper()
    send upper case string back to this client → 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

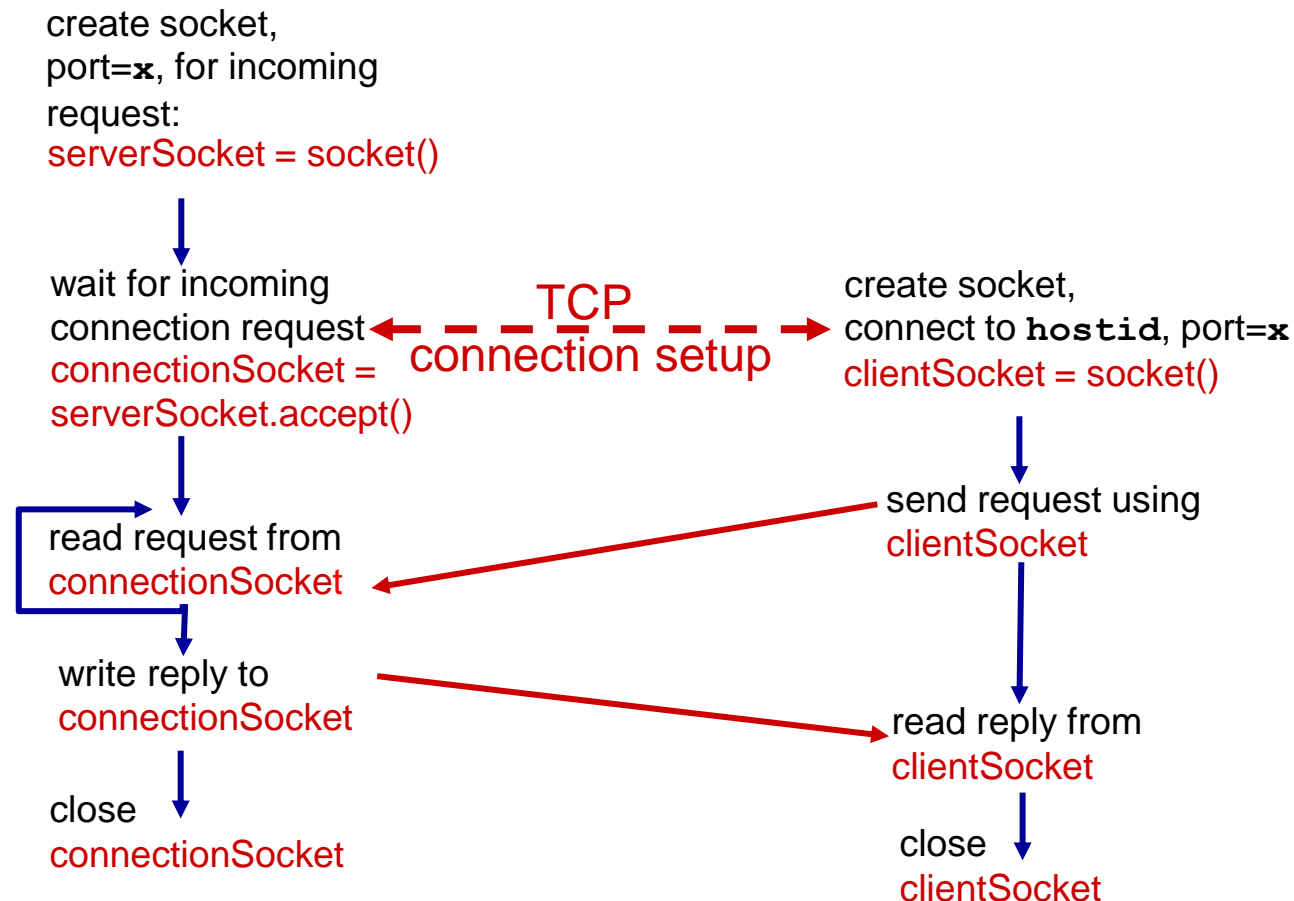
TCP provides reliable, in-order byte-stream transfer ("pipe") between client and server processes

Client/server socket interaction: TCP



server (running on `hostid`)

client



Example app: TCP client

Python TCPClient

create TCP socket for server,
remote port 12000

```
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()
```

No need to attach server name, port

Example app: TCP server

Python TCPServer

		<pre>from socket import *</pre>
		<pre>serverPort = 12000</pre>
create TCP welcoming socket	→	<pre>serverSocket = socket(AF_INET,SOCK_STREAM)</pre>
		<pre>serverSocket.bind(('',serverPort))</pre>
server begins listening for incoming TCP requests	→	<pre>serverSocket.listen(1)</pre>
		<pre>print 'The server is ready to receive'</pre>
loop forever	→	<pre>while True:</pre>
server waits on accept() for incoming requests, new socket created on return	→	<pre> connectionSocket, addr = serverSocket.accept()</pre>
		<pre> sentence = connectionSocket.recv(1024).decode()</pre>
read bytes from socket (but not address as in UDP)	→	<pre> capitalizedSentence = sentence.upper()</pre>
		<pre> connectionSocket.send(capitalizedSentence.encode())</pre>
close connection to this client (but <i>not</i> welcoming socket)	→	<pre> connectionSocket.close()</pre>

Group Project Assignment

- Group Project ไปทำให้ run ระหว่างเครื่องนักศึกษา 2 เครื่องที่อยู่คนละที่ run ให้ได้ค่ะ
 - UDP Socket
 - TCP Socket