CS 4390: Application Layer

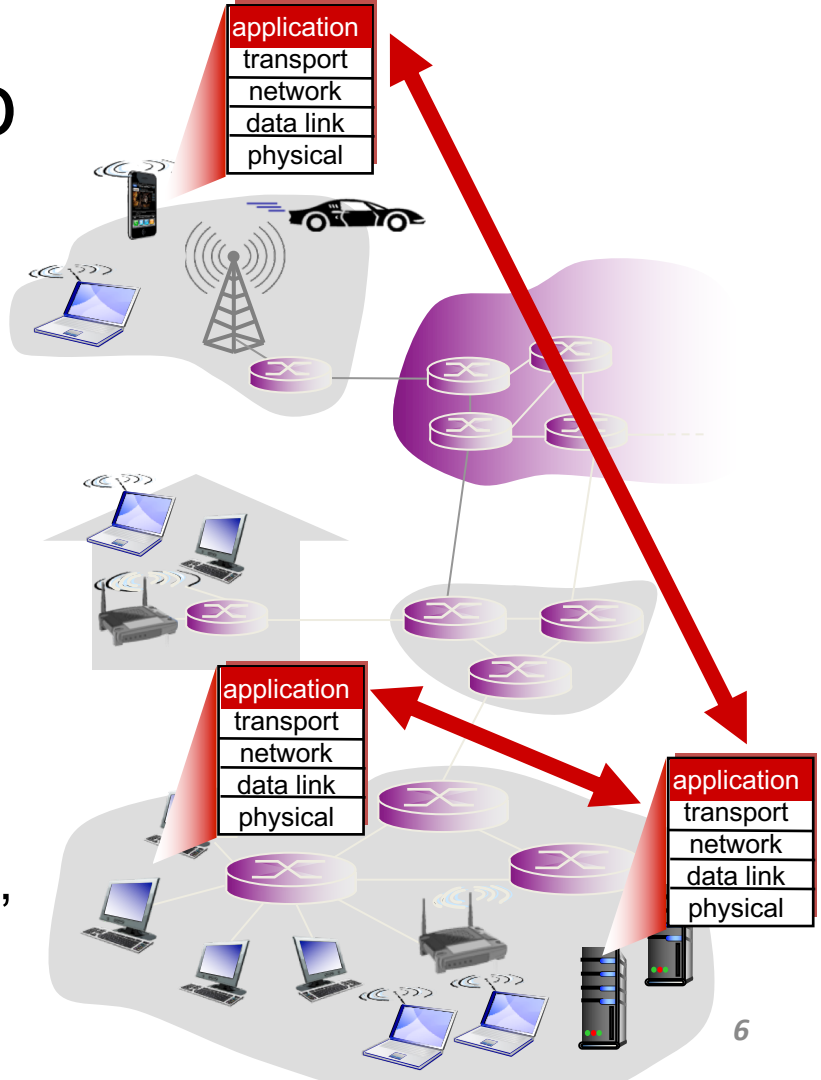
Shuang Hao
University of Texas at Dallas
Fall 2023

Application Layer

- Learning about protocols by examining popular application-level protocols
- Example application protocols
  - HTTP
  - FTP
  - SMTP / POP3 / IMAP
  - DNS
- Creating network applications
  - socket API

Creating A Network App

- Write programs that:
  - run on (different) end systems
  - communicate over network
  - e.g., web server software communicates with browser software
- No need to write software for network-core devices
  - Network-core devices do not run user applications
  - Applications on end systems allows for rapid app development, propagation



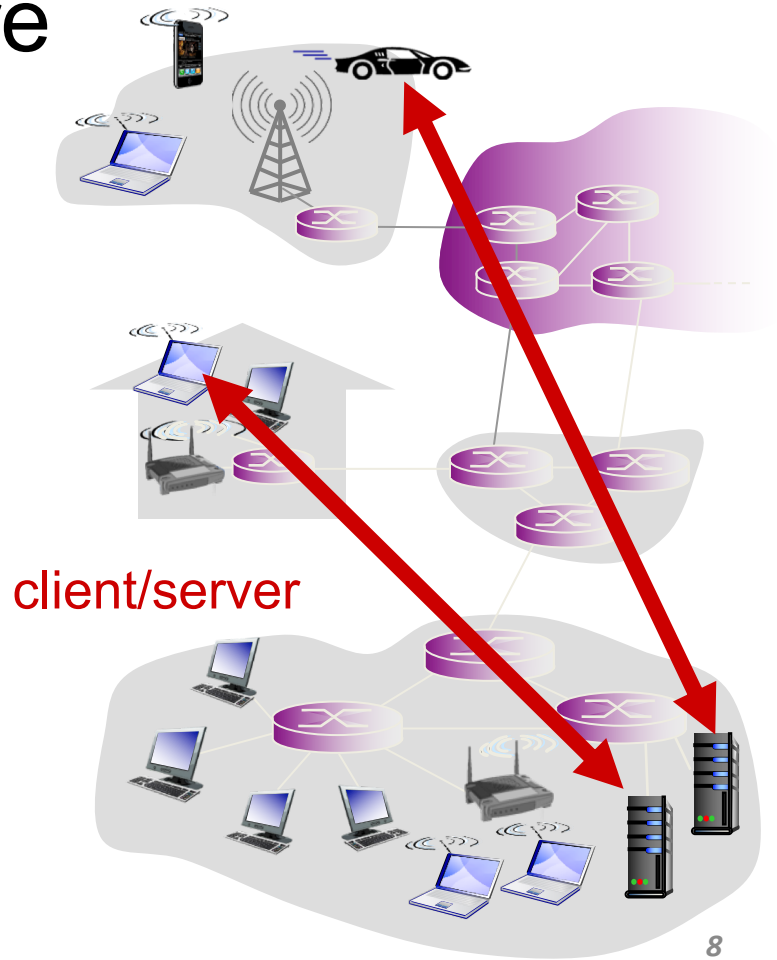
Application Architectures

- Possible structure of applications:
  - Client-server
  - Peer-to-peer (P2P)

Client-server Architecture

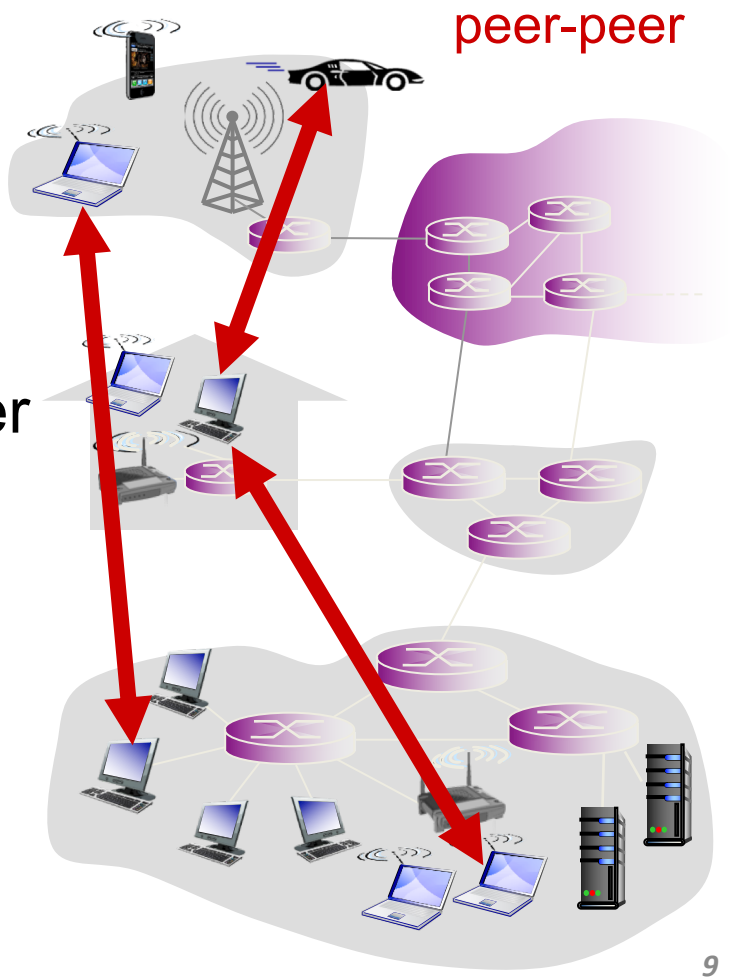
- Server:
  - always-on host
  - permanent IP address
  - data centers for scaling

- Clients:
  - communicate with server
  - may be intermittently connected
  - may have dynamic IP addresses
  - do not communicate directly with each other



P2P Architecture

- No always-on server
- Arbitrary end systems directly communicate
- Peers request service from other peers, provide service in return to other peers
  - Peers are intermittently connected and change IP addresses
  - Complex management



Recap Where We’re At

- Protocol Layers
- Application layer

Outline

- Process Communicating
- HTTP

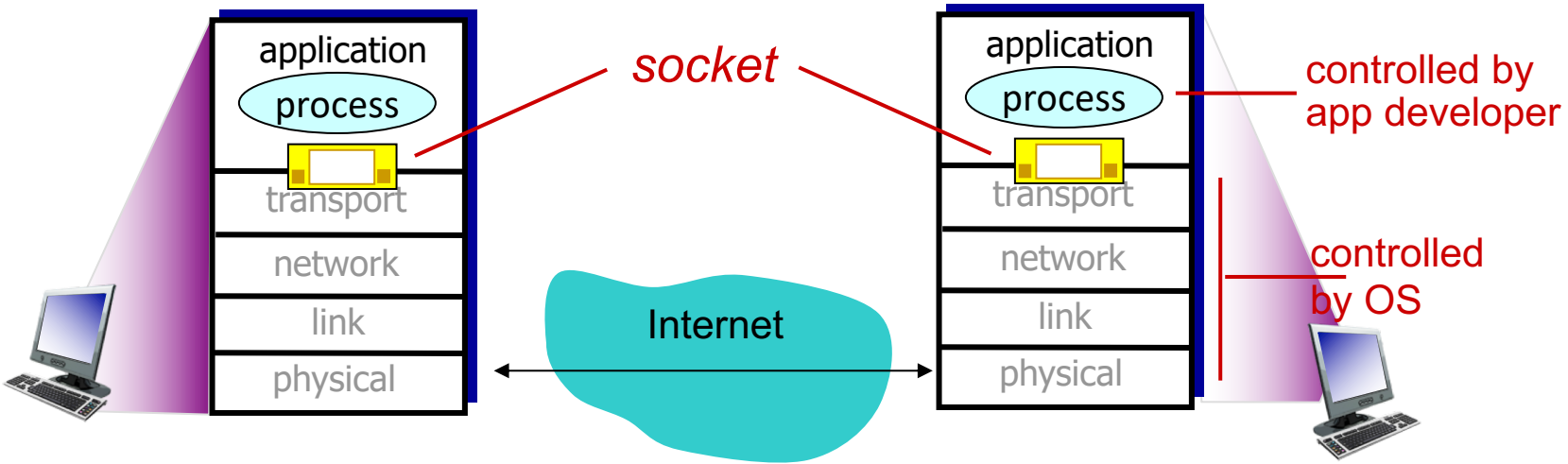
Processes Communicating

- Clients, servers
  - Client process: process that initiates communication
  - Server process: process that waits to be contacted

- Process: program running within a host
  - Within same host, two processes communicate using inter-process communication (defined by OS)
  - Processes in different hosts communicate by exchanging messages

Sockets

- Process sends/receives messages to/from its socket



Addressing Processes

- To receive messages, process must have identifier
- Identifier includes both IP address and port numbers associated with process on host.
  - Host device has unique 32-bit IP address
    - 172.16.254.1
  - Example port numbers:
    - HTTP server: 80
    - mail server: 25

Internet Apps

application	application layer protocol	underlying transport protocol
e-mail	SMTP [RFC 2821]	TCP
remote terminal access	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
file transfer	FTP [RFC 959]	TCP
streaming multimedia	HTTP (e.g., YouTube), RTP [RFC 1889]	TCP or UDP
Internet telephony	SIP, RTP, proprietary (e.g., Skype)	TCP or UDP

Web and HTTP

- The World-Wide Web was originally conceived as a geographically distributed document retrieval system with a hypertext structure
- Web page consists of objects
  - Object can be HTML file, JPEG image, audio file,…
  - Web page consists of base HTML-file which includes several referenced objects
  - Each object is addressable by a URL

URL Syntax

- Uniform Resource Locator
  - an identifier that contains enough information to access the resource
- The general URL syntax is specified in RFC 2396
- Syntax: <scheme>://<authority><path>?<query>
- Examples:
  - ftp://ftp.ietf.org/rfc/rfc1808.txt
  - http://cs.utdallas.edu/?s=2023

HyperText Transfer Protocol

- HTTP: Hypertext Transfer Protocol
  - Web's application layer protocol
  - Client/server model
    - Client: browser that requests, receives, (using HTTP protocol) and "displays" Web objects
    - Server: Web server sends (using HTTP protocol) objects in response to requests

HyperText Transfer Protocol

- Based on TCP, uses port 80
- HTTP is “stateless”
  - Server maintains no information about past client requests
- Version 1.1 is defined in RFC 2616

HTTP Procedure

- suppose user enters URL:
  www.someSchool.edu/someDepartment/home.index

(contains text, references to 10 jpeg images)

1a. HTTP client initiates TCP connection to HTTP server (process) at www.someSchool.edu on port 80

1b. HTTP server at host www.someSchool.edu waiting for TCP connection at port 80. "accepts" connection, notifying client

2. HTTP client sends HTTP *request message* (containing URL) into TCP connection socket. Message indicates that client wants object someDepartment/home.index

3. HTTP server receives request message, forms *response message* containing requested object, and sends message into its socket

time

HTTP Procedure

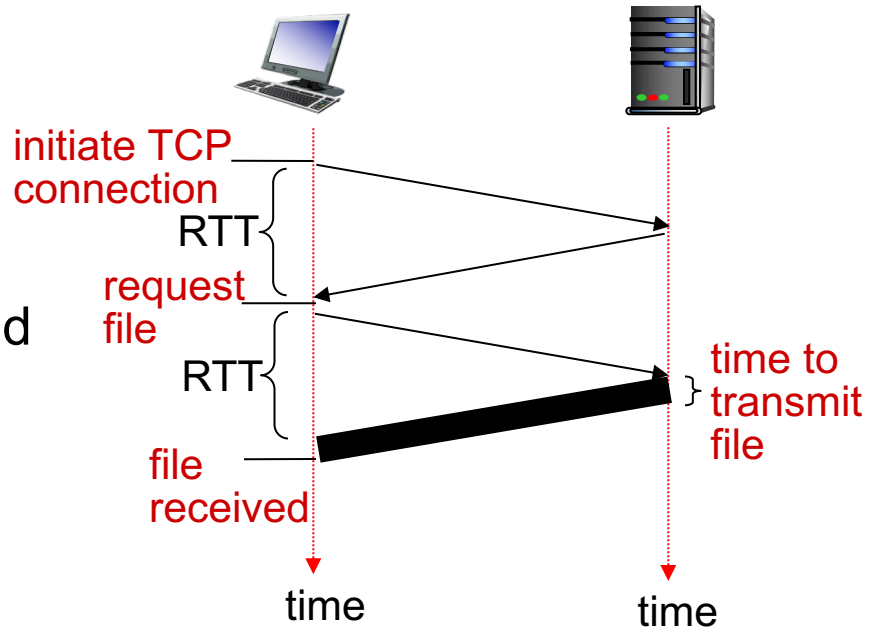
4. HTTP server closes TCP connection.

5. HTTP client receives response message containing html file, displays html. Parsing html file, finds 10 referenced jpeg objects

6. Steps 1-5 repeated for each of 10 jpeg objects

HTTP: Response Time

- RTT (definition): time for a small packet to travel from client to server and back
- HTTP response time:
  - one RTT to initiate TCP connection (an assumption)
  - one RTT for HTTP request and first few bytes of HTTP response to return
  - file transmission time
- HTTP response time =
  - 2RTT+ file transmission time



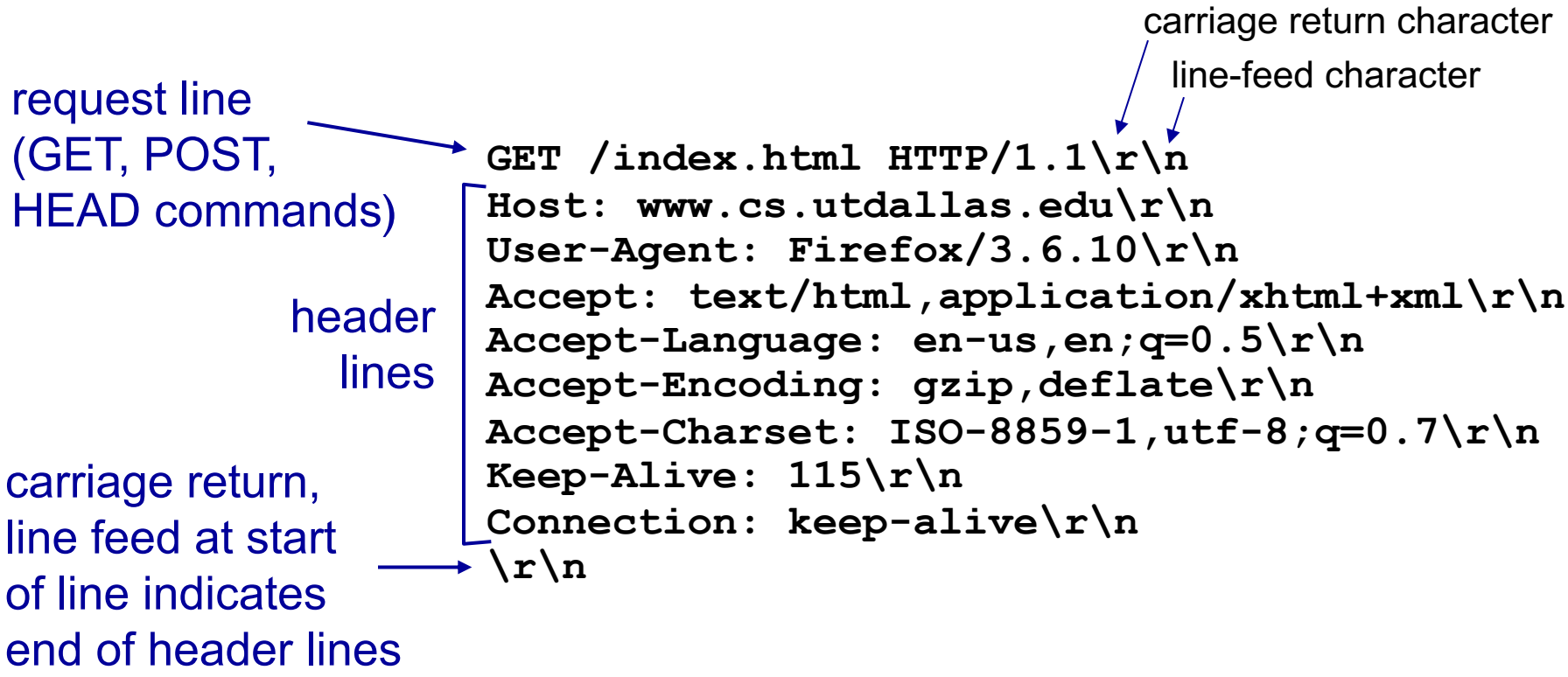
Requests

- A request is composed of a header and a body (optional) separated by an empty line (CR LF)
  - ASCII (human-readable format)
- The header specifies:
  - Method (GET, HEAD, POST)
  - Resource (e.g., /hypertext/doc.html)
  - Protocol version (HTTP/1.1)
- The body is considered as a byte stream

Methods

- GET requests the transfer of the entity referred by the URL
- HEAD requests the transfer of header meta-information only
- POST asks the server to process the included entity as “data” associated with the resource identified by the URL
  - Message posting (newsgroups and mailing list)
  - Form data submission
  - Database input

HTTP Request Example



Recap Where We're At

- Process Communicating

- HTTP
  - URL syntax, HTTP procedure
  - Request and GET/POST methods

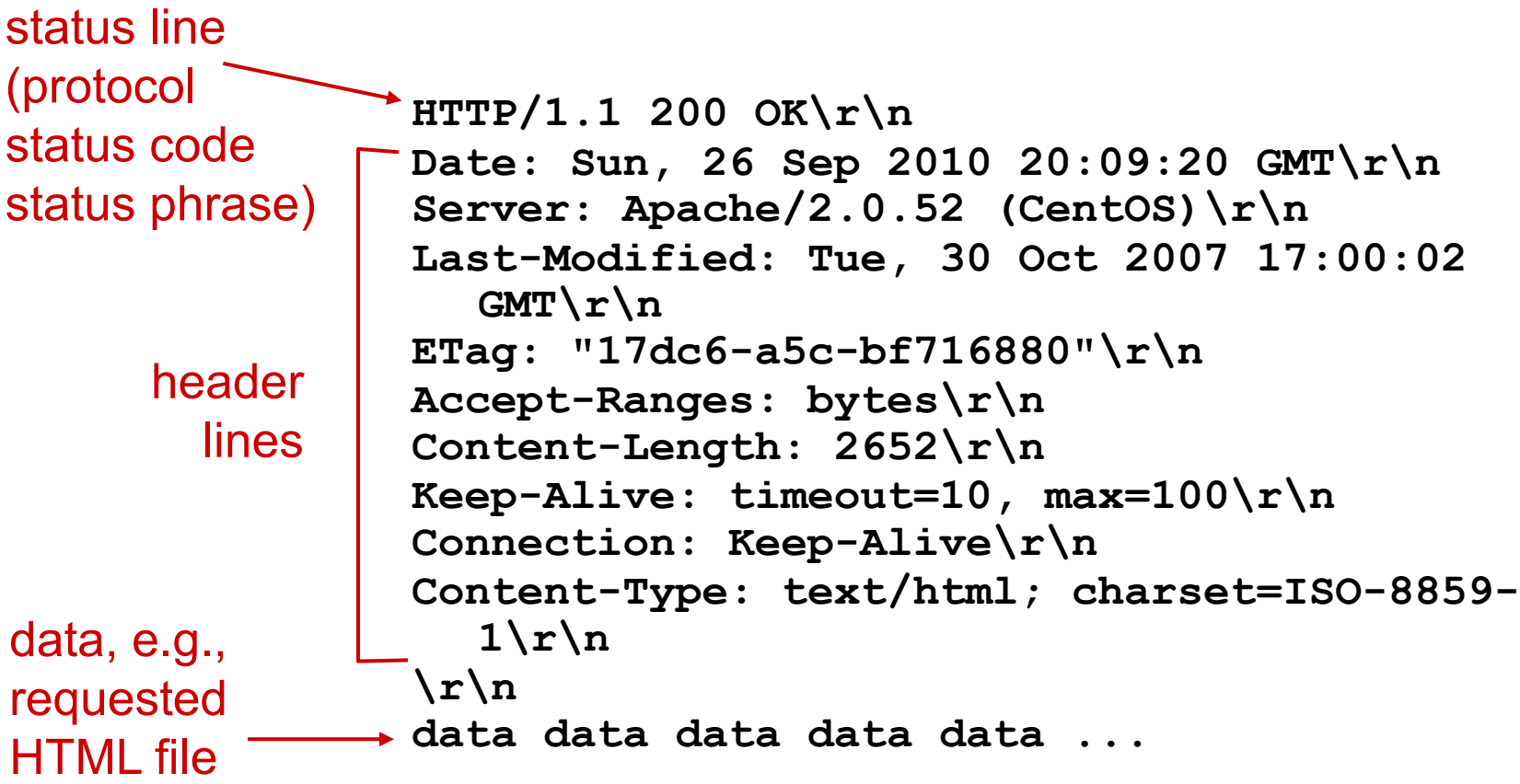
Outline

- HTTP

Replies

- Replies are composed of a header and a body separated by a empty line (CR LF)
- The header contains:
  - Protocol version (e.g., HTTP/1.0 or HTTP/1.1)
  - Status code
  - Diagnostic text
  - Other info
- The body is a byte stream

HTTP Response Example



Status Codes

- 1xx: Informational - Request received, continuing process
- 2xx: Success - The action was successfully received, understood, and accepted
- 3xx: Redirection - Further action must be taken in order to complete the request
- 4xx: Client Error - The request contains bad syntax or cannot be fulfilled
- 5xx: Server Error - The server failed to fulfil an apparently valid request

Examples

- "200" ; OK
- "201" ; Created
- "202" ; Accepted
- "204" ; No Content
- “301" ; Moved Permanently
- "307" ; Temporary Redirect

- "400" ; Bad Request
- "401" ; Unauthorized
- "403" ; Forbidden
- "404" ; Not Found
- "500" ; Internal Server Error
- "501" ; Not Implemented
- "502" ; Bad Gateway
- "503" ; Service Unavailable

User-server State: Cookies

- Cookies are small information containers that a web server can store on a web client
- They are set by the server by including the “Set-Cookie” header field in a reply:

  Set-Cookie: USER=foo; SHIPPING=fedex; path=/
- Cookies are passed (as part of the “Cookie” header field) in every further transaction with the site that set the cookie

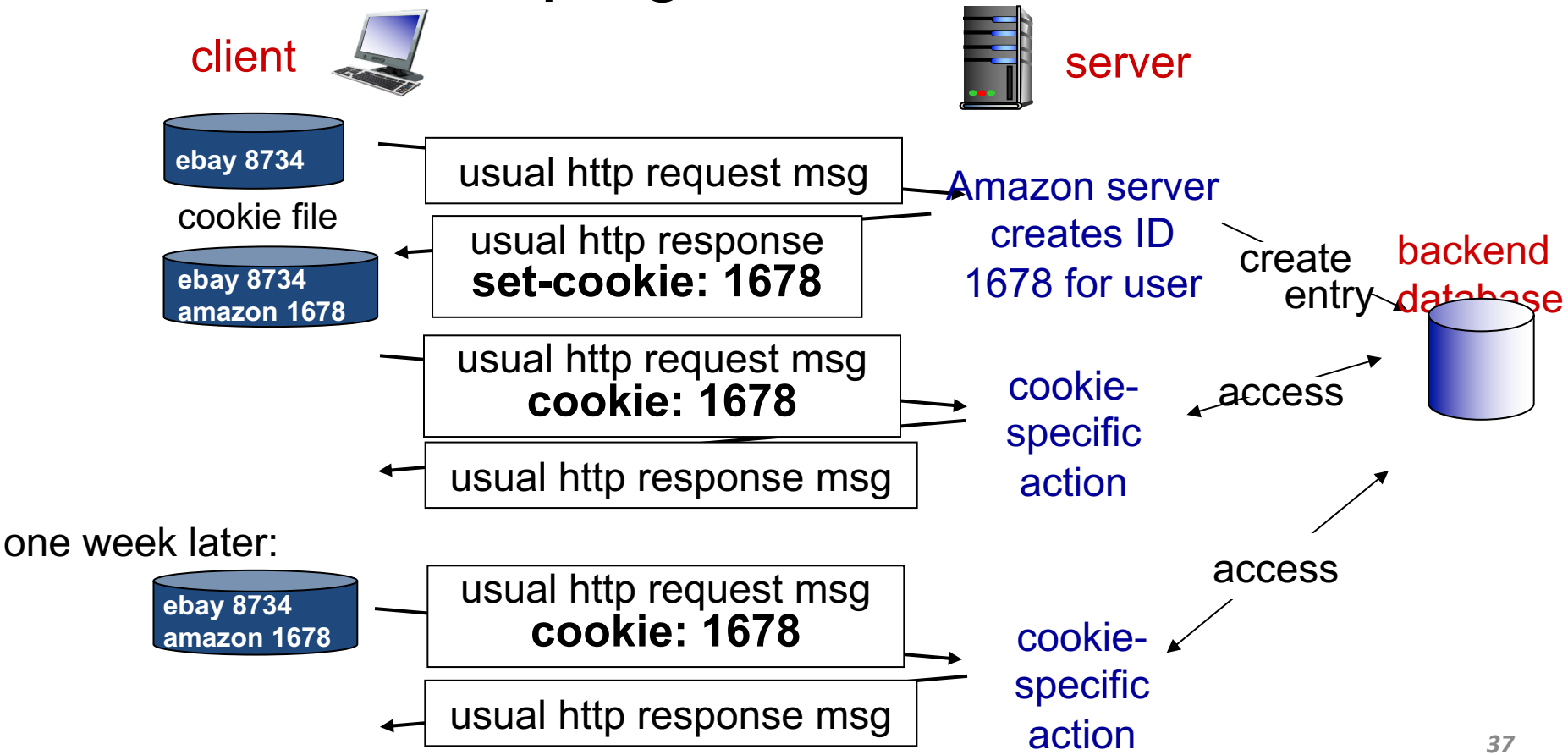
  Cookie: USER=foo; SHIPPING=fedex;

Cookies

- What cookies can be used for:
  - authorization
  - shopping carts
  - recommendations

- Cookies and privacy
  - Cookies permit sites to learn a lot about you

Cookies: Keeping "State"

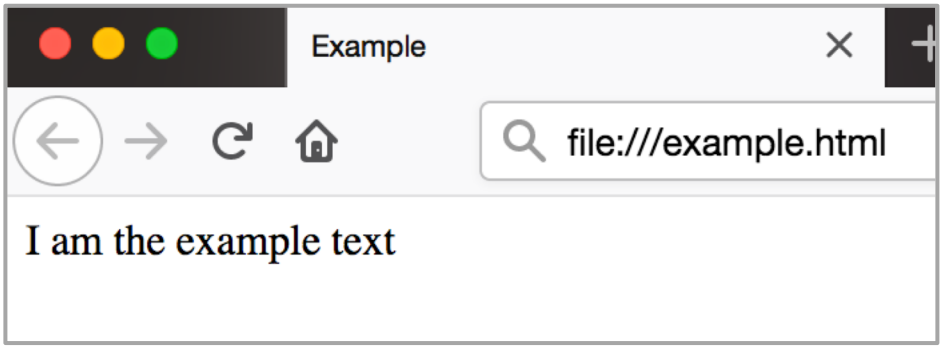


HTML

- Basic idea is to “markup” document with tags, which add meaning to raw text
  - Start tag: <foo>
  - Followed by text
  - End tag: </foo>
  - Self-closing tag: <bar />
- Tag are hierarchical

HTML – Example

```
<html>
  <head>
    <title>Example</title>
  </head>
  <body>
    <p>I am the example text</p>
  </body>
</html>
```



HTML – Hyperlink

- The anchor tag is used to create a hyperlink
- href attribute is used provide the URL
- Text inside the anchor tag is the text of the hyperlink

```
<a href="http://google.com">Google</a>
```

CSS (Cascading Style Sheets)

- Language used for describing the presentation of a document

- index.css

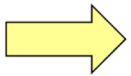
```
p.serif
 {
font-family: "Times New Roman", Times, serif;
}
p.Sansserif
{
font-family: Arial, Helvetica, sans-serif;
}
```

DOM (Document Object Model)

- Cross-platform model for representing and interacting with objects in HTML

HTML

```
<html>
    <body>
        <div>
            foo
        </div>
        <form>
            <input type="text" />
            <input type="radio" />
            <input type="checkbox" />
        </form>
    </body>
</html>
```



DOM Tree

```
|-> Document
   |-> Element (<html>)
      |-> Element (<body>)
         |-> Element (<div>)
            |-> text node
         |-> Form
               |-> Text-box
               |-> Radio Button
               |-> Check Box
```

Executing Code on the Server

- The server-side component of an application executes code in reaction to an HTTP request
- This simple mechanism allows for the creation of web-based portal to database and other applications

PHP

- The “PHP Hypertext Processor” is a scripting language that can be embedded in HTML pages to generate dynamic content
- PHP code is executed on the server side when the page containing the code is requested
- A common setup is a LAMP system, which is the composition of
  - Linux
  - Apache
  - MySQL
  - PHP

Client-side Scripting

- Scripting languages used to implement dynamic behavior in web pages
- JavaScript
  - Code is included using external references

```
<script src="http://www.foocom/somecode.js"></script>
```

  - Code is embedded into HTML pages using the SCRIPT tag

```
<script LANGUAGE="JavaScript">
 var name = prompt ('Please Enter your name below.','')
 if ( name == null ) {
       document.write ('Welcome to my site!')
 }
 else {
       document.write ('Welcome to my site '+name+'!')
 }
</script>
```

The Power of JavaScript

- Almost anything you want to the DOM
  - Can change HTML content
  - Can change images
  - Can chance style of elements
  - Can hide elements
  - Can access cookies
  - ….

Recap Where We're At

- HTTP
  - Response and Cookie
  - HTML and CSS
  - Scripting Languages and JavaScript

Outline

- HTTP
  - Same Origin Policy
  - HTTPS
- DNS

Same Origin Policy

- "Same origin" policy
  - JavaScript code can access only resources (e.g., cookies) that are associated with the same origin
- An origin is defined as a combination of URL scheme, host name, and port number (exact match)

Same Origin Policy

- Checks against the URL

  http://www.example.com/dir/page.html

✓ http://www.example.com/dir/page2.html

✓ http://www.example.com/dir2/other.html

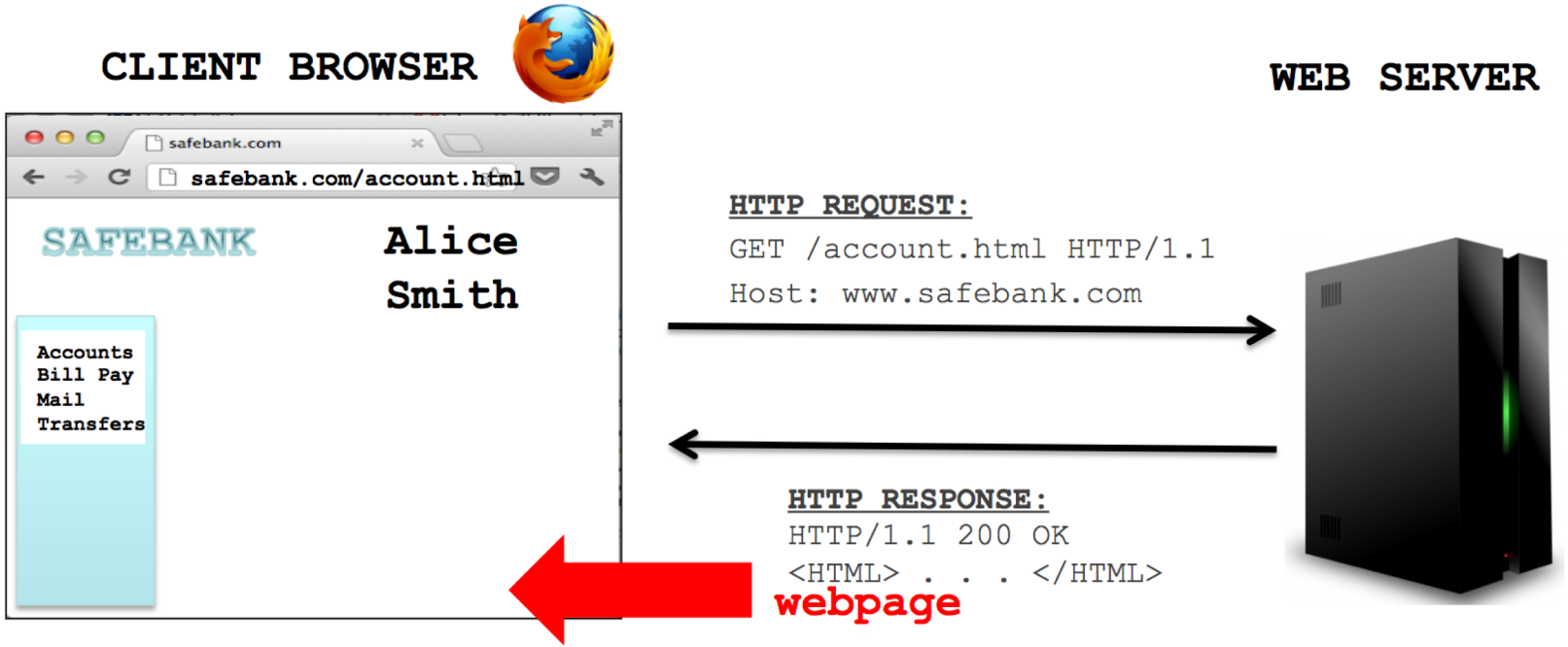
✓ http://username:password@www.example.com/dir2/other.html

✗ http://www.example.com:81/dir/other.html Same protocol and host but different port

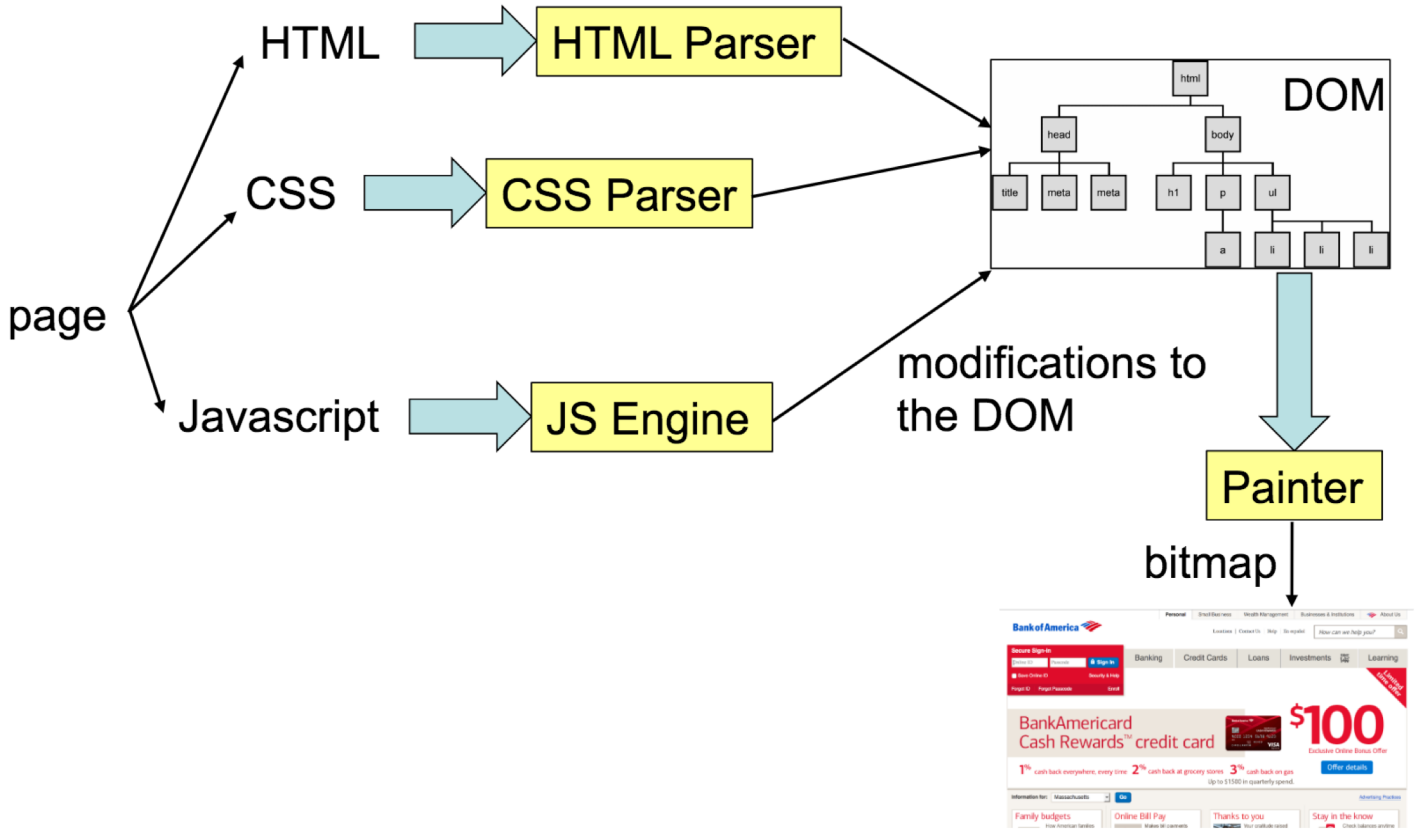
✗ https://www.example.com/dir/other.html Different protocol

✗ http://en.example.com/dir/other.html Different host

HTTP Workflow



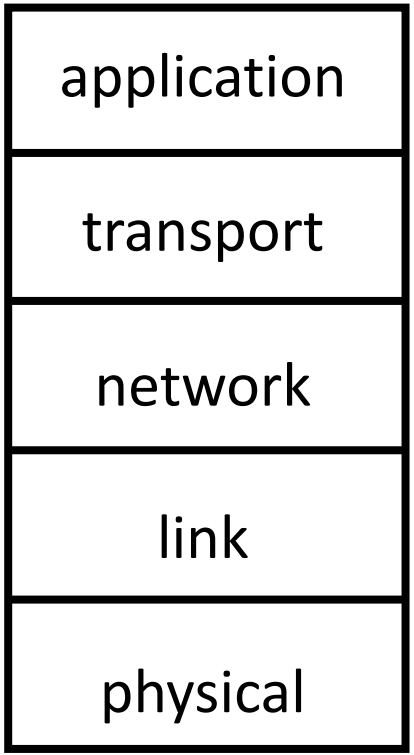
Page Rendering

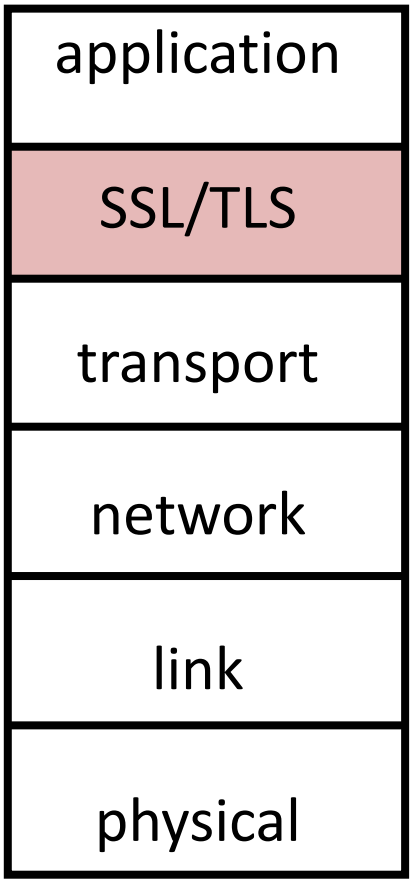


HTTPS

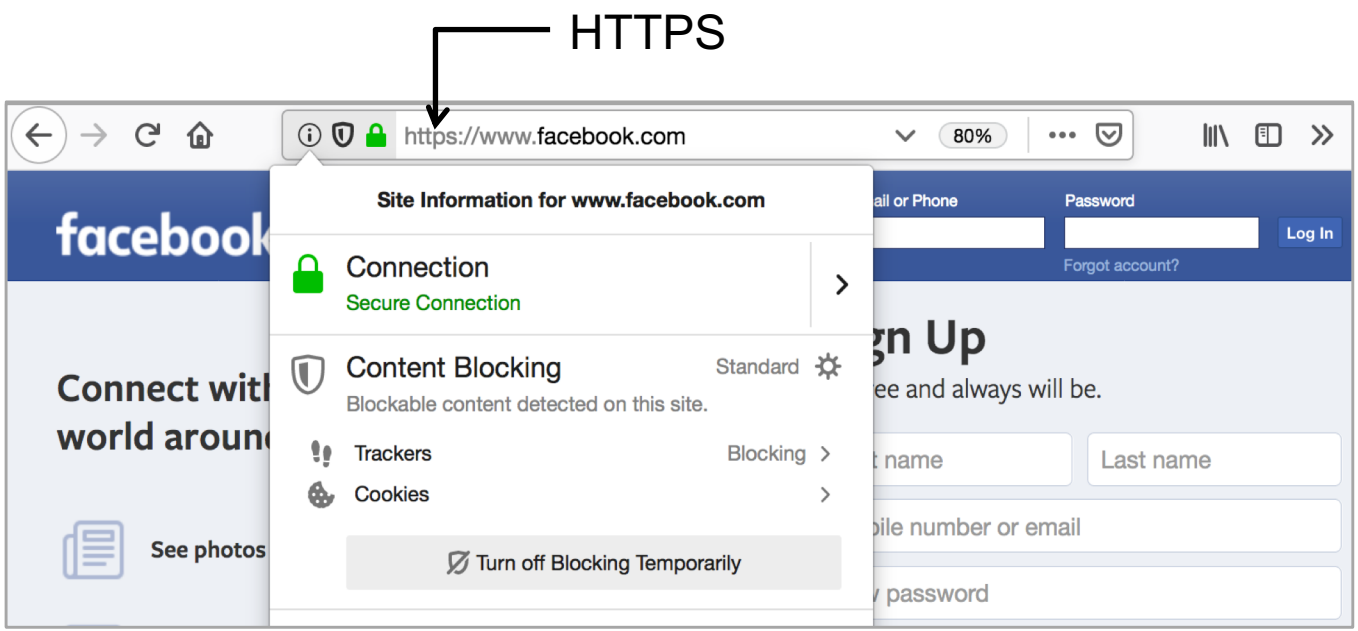
- Hypertext Transfer Protocol Secure (HTTPS)
  - Authentication of the accessed website and protection of the confidentiality and integrity of the exchanged data while in transit
  - HTTPS URLs begin with "https://"
  - Use TCP port 443 by default
- In HTTPS, the communication protocol is encrypted using Transport Layer Security (TLS)
  - Formerly, its predecessor, Secure Sockets Layer (SSL)

SSL/TLS in Network Layering

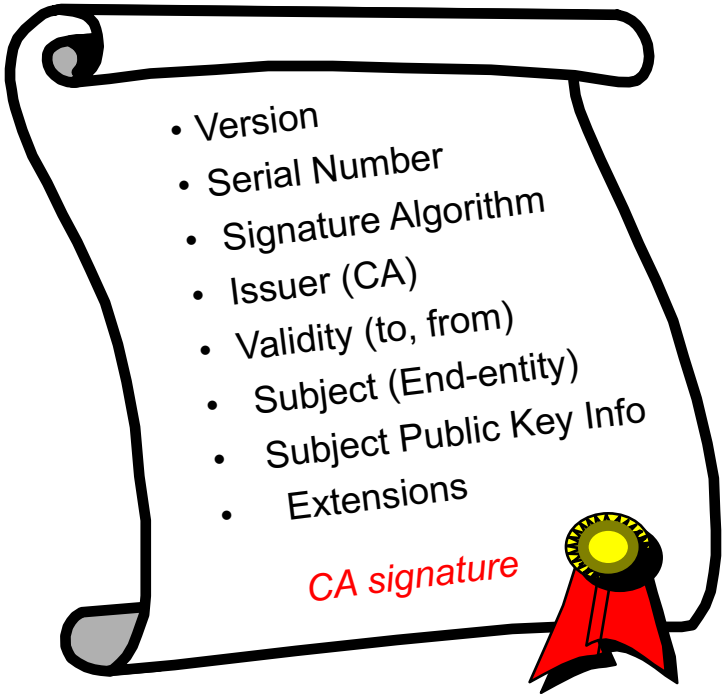




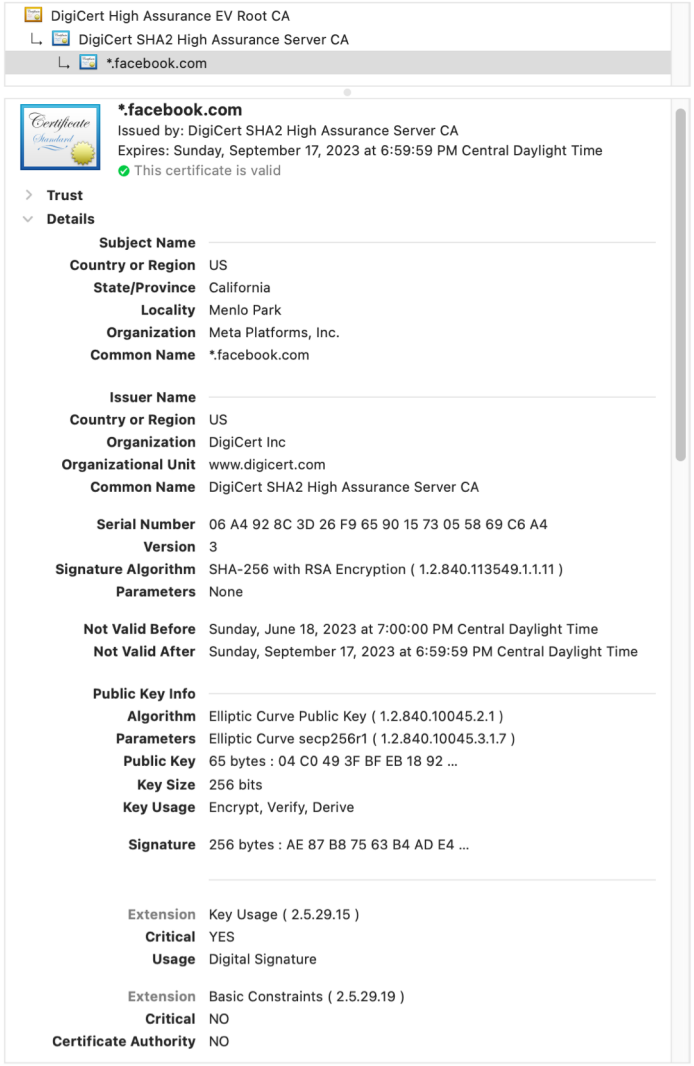
HTTPS (Instead of HTTP)



Certificate



Certificates are used in many Internet protocols, including TLS/SSL


DigiCert High Assurance EV Root CA
DigiCert SHA2 High Assurance Server CA
*.facebook.com
*.facebook.com
Issued by: DigiCert SHA2 High Assurance Server CA
Expires: Sunday, September 17, 2023 at 6:59:59 PM Central Daylight Time
This certificate is valid
Trust
Details
Subject Name
Country or Region US
State/Province California
Locality Menlo Park
Organization Meta Platforms, Inc.
Common Name *.facebook.com
Issuer Name
Country or Region US
Organization DigiCert Inc
Organizational Unit www.digicert.com
Common Name DigiCert SHA2 High Assurance Server CA
Serial Number 06 A4 92 8C 3D 26 F9 65 90 15 73 05 58 69 C6 A4
Version 3
Signature Algorithm SHA-256 with RSA Encryption (1.2.840.113549.1.1.11)
Parameters None
Not Valid Before Sunday, June 18, 2023 at 7:00:00 PM Central Daylight Time
Not Valid After Sunday, September 17, 2023 at 6:59:59 PM Central Daylight Time
Public Key Info
Algorithm Elliptic Curve Public Key (1.2.840.10045.2.1)
Parameters Elliptic Curve secp256r1 (1.2.840.10045.3.1.7)
Public Key 65 bytes : 04 C0 49 3F BF EB 18 92 ...
Key Size 256 bits
Key Usage Encrypt, Verify, Derive
Signature 256 bytes : AE 87 B8 75 63 B4 AD E4 ...
Extension Key Usage (2.5.29.15)
Critical YES
Usage Digital Signature
Extension Basic Constraints (2.5.29.19)
Critical NO
Certificate Authority NO

DNS: Domain Name System

- People: many identifiers
  - SSN, name, passport #
- Internet hosts, routers:
  - IP address (32 bit) - used for addressing datagrams
  - "name", e.g., www.google.com - used by humans
- Q: how to map between IP address and name, and vice versa ?

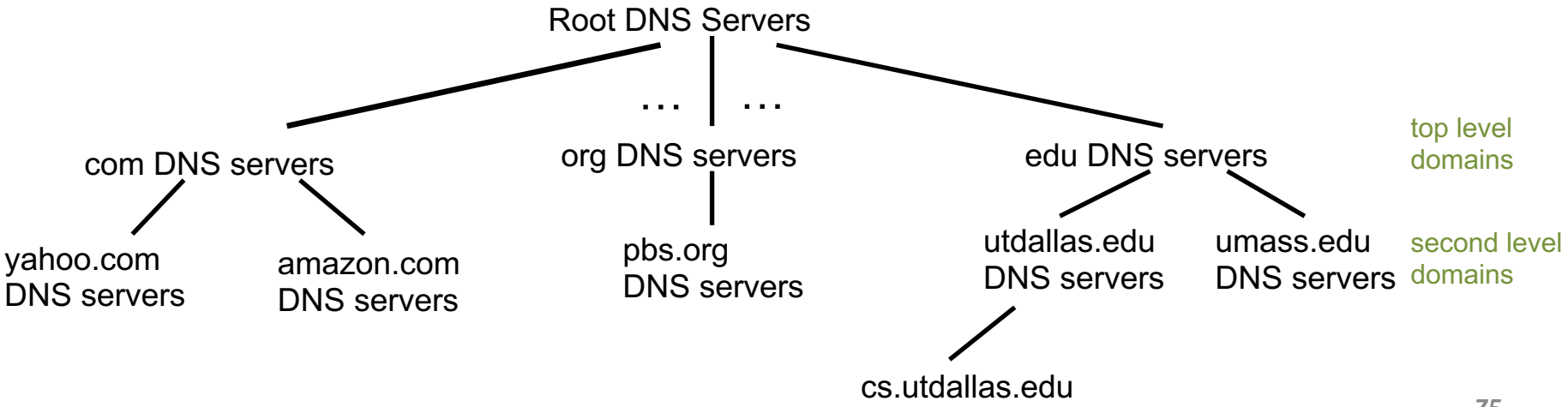
- Domain Name System
  - Distributed database implemented in hierarchy of many name servers
  - Application-layer protocol: address/name translation
    - Using UDP, port 53
    - RFC 1034
  - Core Internet function, implemented as application-layer protocol

DNS: Services, Structure

- DNS services
  - Hostname to IP address translation
  - Host aliasing
  - Mail server aliasing
  - Load distribution
- Why not centralize DNS?
  - Single point of failure
  - Traffic volume
  - Maintenance
  - Doesn’t scale

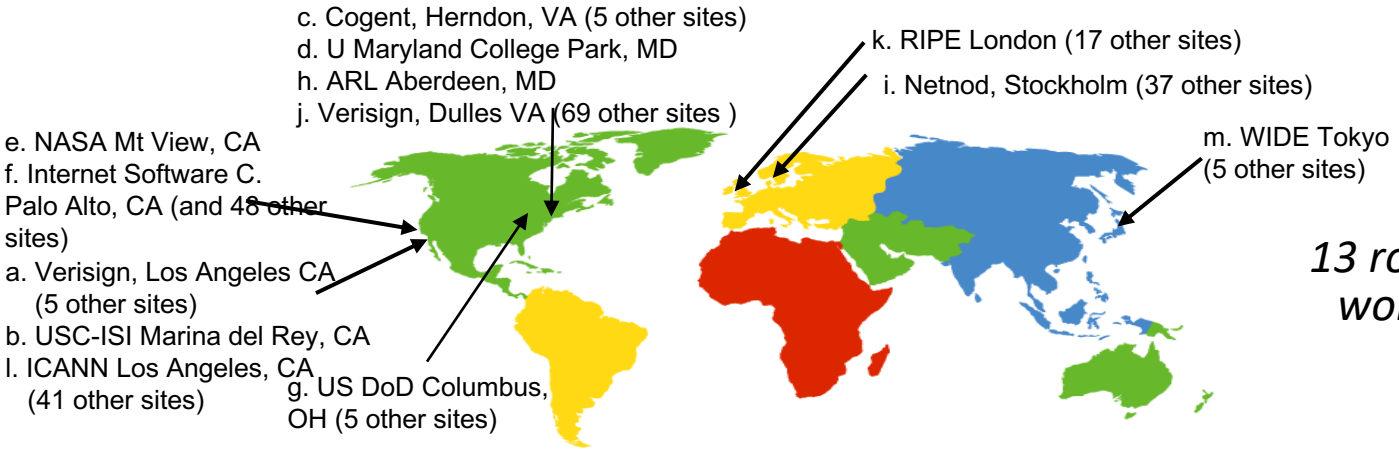
DNS: A Distributed, Hierarchical Database

- Domain name format
  - cs.utdallas.edu.
  - Each label consists of ASCII letters, digits, and hyphens
  - Case insensitive



DNS: Root Name Servers

- Root name server
  - Contacted by clients if name mapping not known
  - Returns mapping to the next-level name server



13 root name "servers" worldwide

TLD, Authoritative Servers

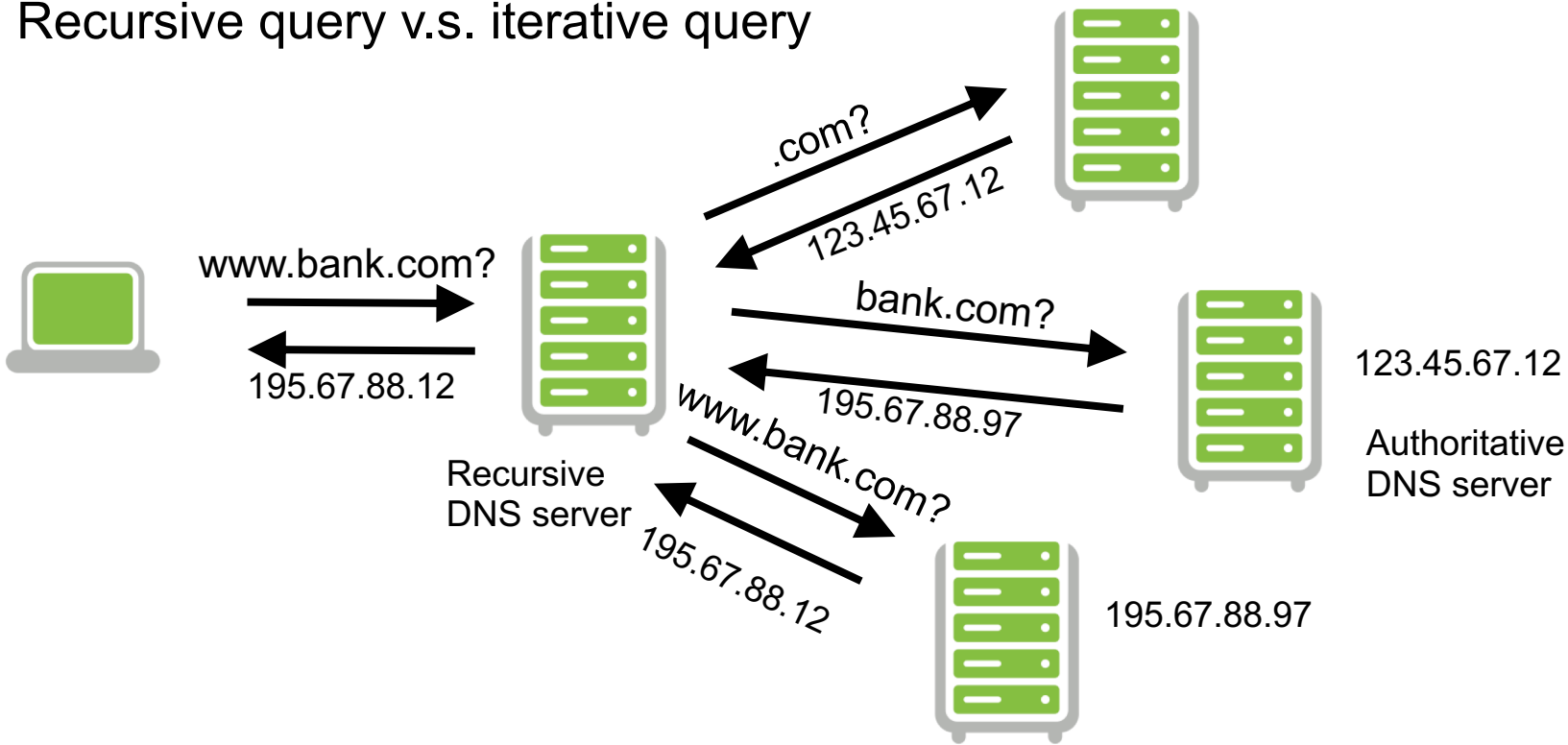
- Top-level domain (TLD) servers:
  - Responsible for com, org, net, edu, aero, jobs, museums, and all top-level country domains, e.g.: uk, fr, ca, jp
- Authoritative DNS servers:
  - Organization's own DNS server(s), providing authoritative hostname to IP mappings for organization's named hosts

DNS Queries

- Recursive queries: require a name server to find the answer to the query itself
- Iterative queries: require a name server to provide the information requested or a reference to another server that may have the information

DNS Recursive Resolution

- Recursive query v.s. iterative query



DNS Caching

- 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

DNS Resource Records

DNS: distributed db 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

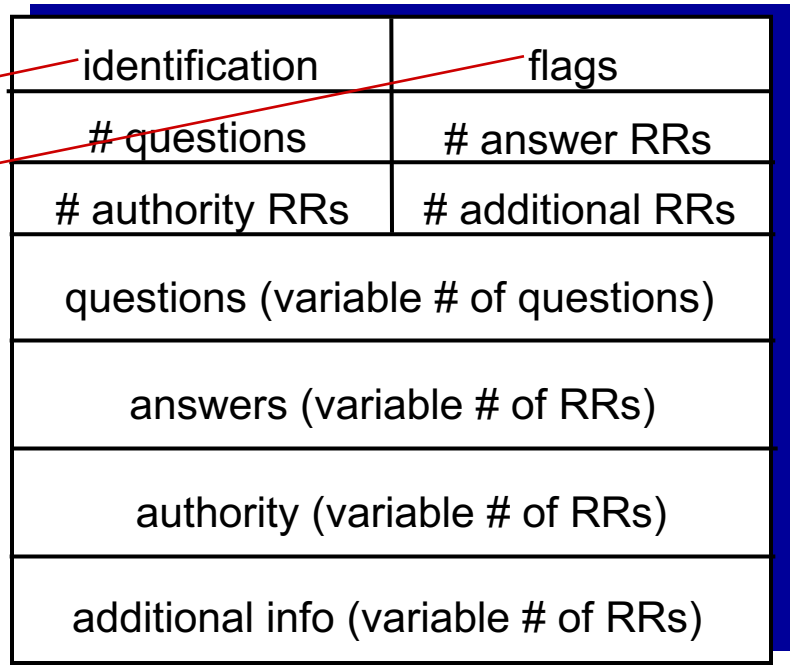
- **`value`** is name of mailserver associated with **`name`**

DNS Messages

- Query and reply messages, both with same message format

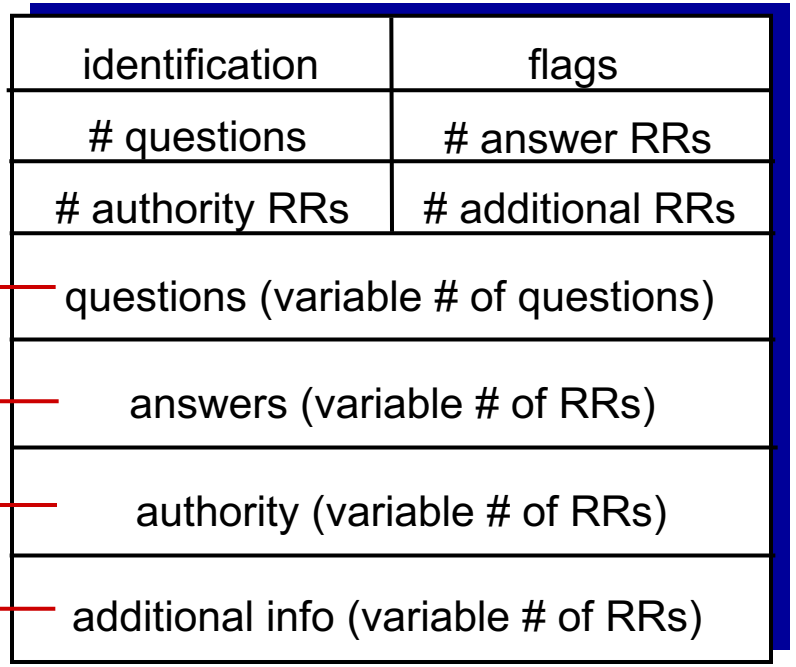
Message header

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



DNS Messages

- name, type fields for a query
- records in response to query
- records for authoritative servers
- additional "helpful" info that may be used



Dig

```
$ dig A cs.utdallas.edu

; <<>> DiG 9.8.3-P1 <<>> A cs.utdallas.edu
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 11134
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;cs.utdallas.edu.           IN    A

;; ANSWER SECTION:
cs.utdallas.edu. 16984 IN    A     162.144.57.72
```

Dig

```
$ dig +trace A cs.utdallas.edu

; <<>> DiG 9.8.3-P1 <<>> +trace A cs.utdallas.edu
;; global options: +cmd
.               74474   IN      NS      j.root-servers.net.
.               74474   IN      NS      d.root-servers.net.
.               74474   IN      NS      k.root-servers.net.
.               74474   IN      NS      h.root-servers.net.
.               74474   IN      NS      b.root-servers.net.
.               74474   IN      NS      l.root-servers.net.
.               74474   IN      NS      e.root-servers.net.
;; Received 228 bytes from 209.18.47.62#53(209.18.47.62) in 40 ms

edu.            172800  IN      NS      f.edu-servers.net.
edu.            172800  IN      NS      g.edu-servers.net.
edu.            172800  IN      NS      a.edu-servers.net.
;; Received 268 bytes from 192.228.79.201#53(b.root-servers.net) in 51 ms

utdallas.edu.           172800  IN      NS      ns2.ots.utsystem.edu.
utdallas.edu.           172800  IN      NS      ibext3.utdallas.edu.
utdallas.edu.           172800  IN      NS      ibext4.utdallas.edu.
;; Received 154 bytes from 192.26.92.30#53(g.edu-servers.net) in 38 ms

cs.utdallas.edu.        28800   IN      A       162.144.57.72
;; Received 49 bytes from 129.110.182.24#53(ibext3.utdallas.edu) in 19 ms
```

Recap Where We're At

- HTTP
  - Same Origin Policy
  - HTTPS
- DNS

Outline

- DNS
  - DNSSEC
- SMTP
- FTP

DNS Cache Poisoning

- This attack exploits a bug in some DNS implementations
- A recursive server stores in the cache anything that is contained in a DNS reply
- A malicious authoritative server returns additional answers that will poison the cache
- Some old implementations will even accept answer records in DNS request, caching the information
- DNS cache poisoning can:
  - add a record for a specific host
  - add a record for the authoritative server for a whole domain

DNSSEC

- DNSSEC: DNS Security Extensions
  - Protects against data spoofing and corruption
  - Authenticates servers, who sign records with their private keys

DNSSEC Keys

```
$ dig +trace +dnssec com -t DNSKEY

com.            86400 IN DNSKEY256 3 8 (
                        AQPLmJ3YIDDZPh9wYbc76pe1azHsCyIiDzBgf7fz+k4p
                        oMWHzE6f4My3VcPX2SIn/v0jPGBIEVZl2pPdY/IqOd5B
                        fCsjrE+bBUYz6uxWiaT3wMuhfN1LNOazLQRlgU8RrEOg
                        G1L4yFD8m9FVAisWsXLY+WQ8qIyrK/UIUwDmiFC5vQ==
                        ) ; ZSK; alg = RSASHA256; key id = 46967
com.            86400 IN DNSKEY257 3 8 (
                        AQPDzldNmMvZFX4NcNJ0uEnKDg7tmv/F3MyQR0lpBmVc
                        NcsIszxNFxsBfKNW9JYCYqpik8366LE7VbIcNRzfp2h9
                        OO8HRl+H+E08zauK8k7evWEmu/6od+2boggPoiEfGNyv
                        NPaSI7FOIroDsnw/taggzHRX1Z7SOiOiPWPNIwSUyWOZ
                        79VmcQ1GLkC6NlYvG3HwYmynQv6oFwGv/KELSw7ZSdrb
                        TQ0HXvZbqMUI7BaMskmvgm1G7oKZ1YiF7O9ioVNc0+7A
                        SbqmZN7Z98EGU/Qh2K/BgUe8Hs0XVcdPKrtyYnoQHd2y
                        nKPcMMlTEih2/2HDHjRPJ2aywIpKNnv4oPo/
                        ) ; KSK; alg = RSASHA256; key id = 30909
com.            86400 IN RRSIG DNSKEY 8 1 86400 (
                        20180221192533 20180206192033 30909 com.
                        BLBLOXoUPsduhEpnWktynDSmChraVfnkj9quz6YQW2tZ
                        3u2naIkFDHM+72F1OwBzUDo2FosLOP2R3xV4A1skuMSK
                        cywX6WOS2FqLoD5OIGnvRyI3gmtZFVUPBviAIe0TfnBo
                        eJvYzHOwR+tvBLcGSj1G9twdf3RhRBaTaR8P4KcuCZQS
                        zoCEFDk8tIEoZ6U4VhC8PRZiXQc6SY9dgyZxaVs11Ase
                        oRU04BcMgSI/d5WZ5CWGH4ZN+c6+WqzGGoTEiAqzIikz
                        pWM0mDIzI+nxcDiZ4lJCosuFNrwfAwyldXzT8kxopuZ9
                        xmHvonzgYLM1f7Lak9syOyGO4UsPYd3gKw== )
```

DNSSEC Signing

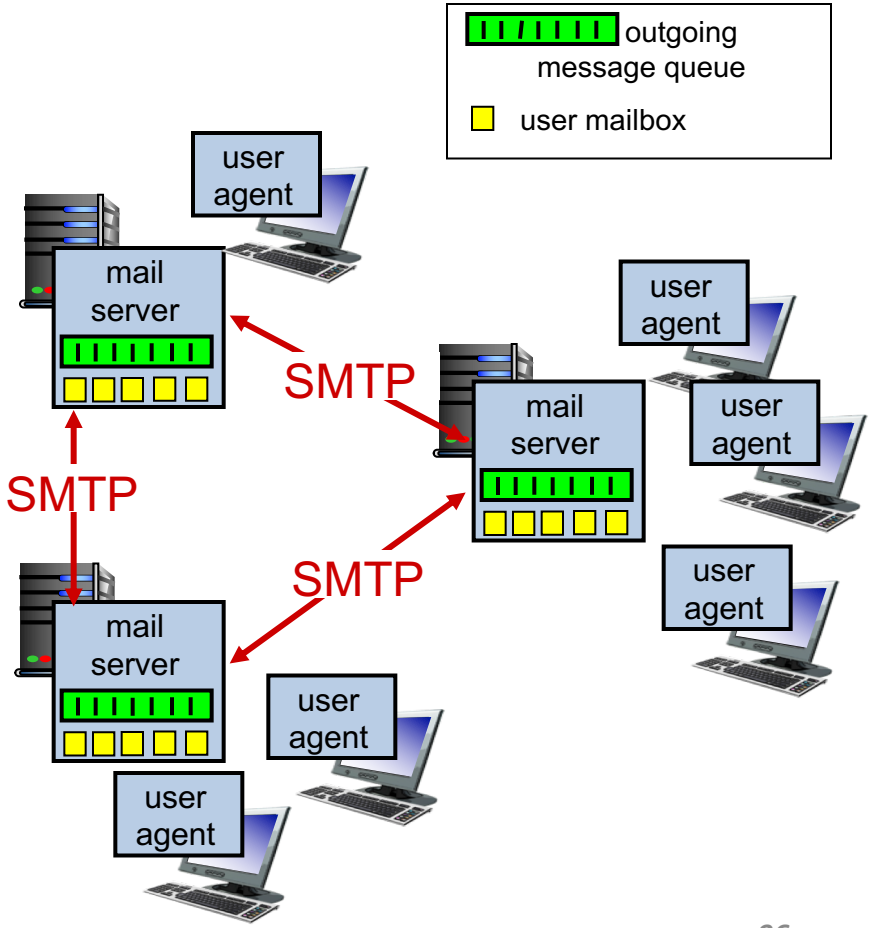
```
$ dig +trace +dnssec -t A upenn.edu

upenn.edu.          300 IN A 130.91.210.133
upenn.edu.          300 IN RRSIG A 5 2 300 (
                    20180315035331 20180213031038 50475 upenn.edu.
                    OvSzBzMPlPZI/2CbybFIykTDEkX4Td3IgCYLI1lCq93X
                    77adBQM/9WO+7GMsdW0c6EdOcd3n6A9ngbVEzzDjV1s6
                    0uL+Ttd5SzNssUuQ+v21Ns0joHzxK0eWbTHPqVR8oNzO
                    sCFMTFcY4rrQ+mFQ8XIZFObjLZLXqmzXWYGTczU= )
```

Electronic Mail

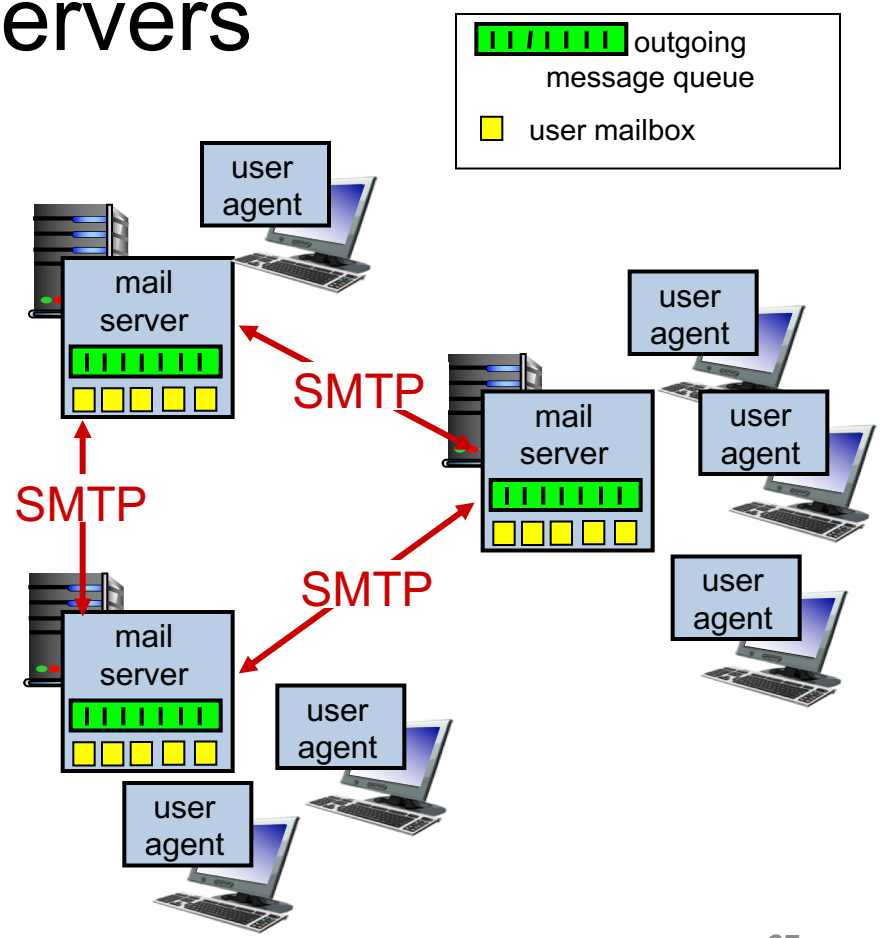
- Three major components
  - User agents
  - Mail servers
  - Simple Mail Transfer Protocol (SMTP)

- User Agent
  - a.k.a. “mail reader”
  - e.g., Outlook, Thunderbird, iPhone mail client
  - Composing, editing, reading mail messages



Electronic 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



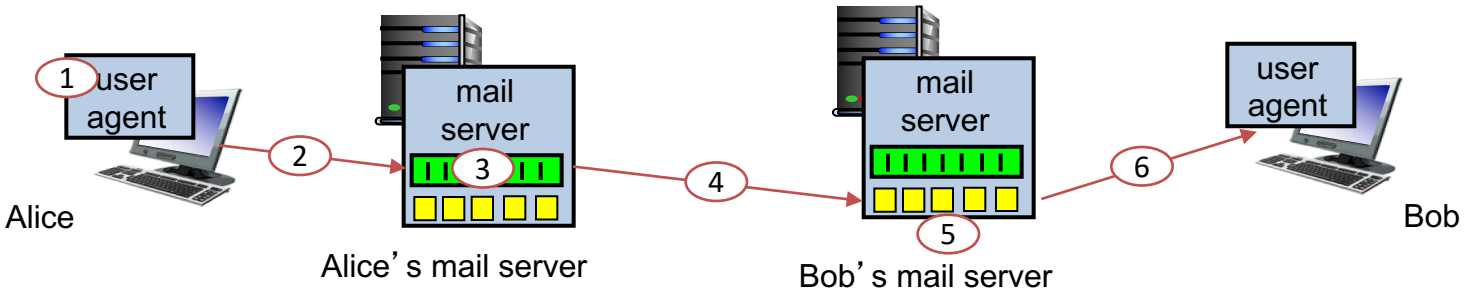
SMTP (RFC 2821)

- Uses TCP to reliably transfer email message from client to server, port 25

- Command/response interaction (like HTTP)
  - Commands: ASCII text
  - Response: status code and phrase

Scenario: Alice Sends Message to Bob

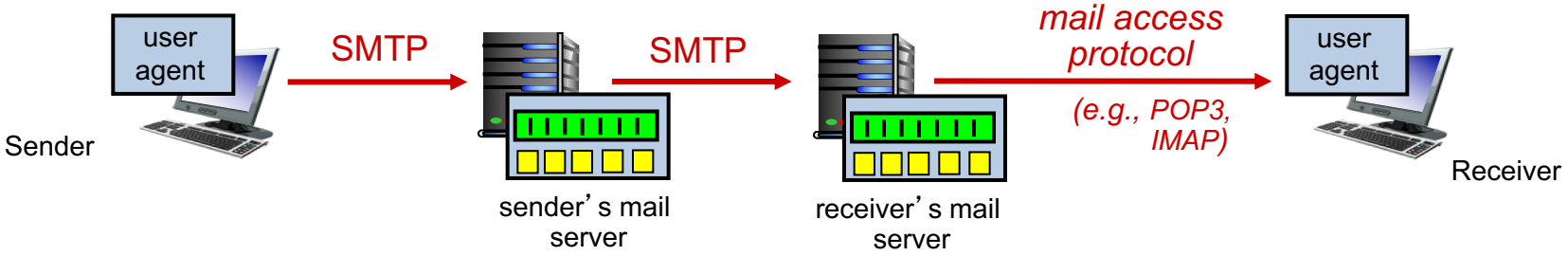
1) Alice uses UA to compose message “to” `bob@someschool.edu`
2) Alice’s UA sends message to her mail server; message placed in message queue
3) client side of SMTP opens TCP connection with Bob’s mail server
4) SMTP client sends Alice’s message over the TCP connection
5) Bob’s mail server places the message in Bob’s mailbox
6) Bob invokes his user agent to read message



Sample SMTP Interaction

```
S: 220 hamburger.edu
C: HELO crepes.fr
S: 250  Hello crepes.fr
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
```

Mail Access Protocols



- SMTP: delivery/storage to receiver's server
- Mail access protocol: retrieval from server
  - POP3: Post Office Protocol [RFC 1939]: authorization, download (TCP port number 110)
  - IMAP: Internet Mail Access Protocol [RFC 1730]: more features, including manipulation of stored msgs on server (TCP port number 143)
  - HTTP: gmail, Hotmail, Yahoo! Mail, etc.

POP3 Protocol

- Authorization phase
  - Client commands:
    - `user`: declare username
    - `pass`: password
  - Server responses
    - `+OK`
    - `-ERR`
- Transaction phase, client:
  - `list`: list message numbers
  - `retr`: retrieve message by number
  - `dele`: delete
  - `quit`

```
S: +OK POP3 server ready
C: user bob
S: +OK
C: pass hungry
S: +OK user successfully logged on
```

```
C: list
S: 1 498
S: 2 912
S: .
C: retr 1
S: <message 1 contents>
S: .
C: dele 1
C: retr 2
S: <message 1 contents>
S: .
C: dele 2
C: quit
S: +OK POP3 server signing off
```

FTP: File Transfer Protocol

- FTP provides a file transfer service
  - Client/server model, using TCP
  - RFC 959
  - The client (ftp) sends a connection request to the server (ftpd), listening on port 21
  - The user provides username and password to authenticate himself/herself

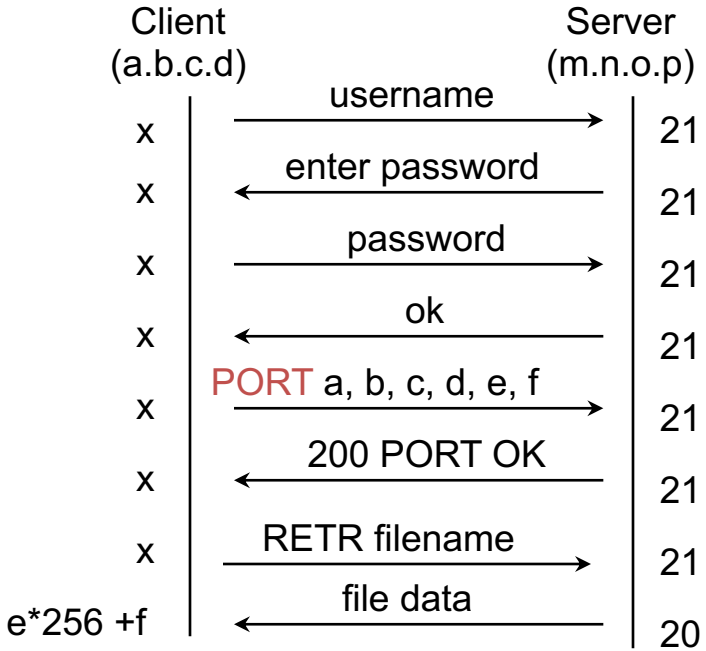
FTP Commands, Responses

- Sample commands:
  - Sent as ASCII text over control channel
  - USER username
  - PASS password
  - LIST return list of file in current directory
  - RETR filename retrieves (gets) file
  - STOR filename stores (puts) file onto remote host
- Sample return codes
  - Status code and phrase
  - 331 Username OK, password required
  - 125 data connection already open; transfer starting
  - 425 Can't open data connection
  - 452 Error writing file

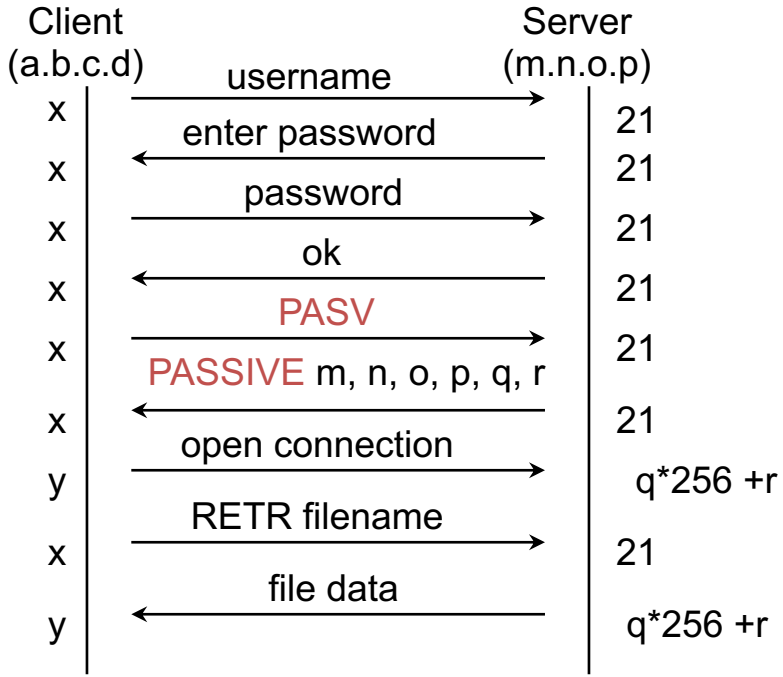
FTP: Separate Control, Data Connections

- FTP client contacts FTP server at port 21
- When server receives file transfer command, server opens 2nd TCP data connection (for file) to client
  - e.g. port 20
- After transferring one file, server closes data connection
- Control connection: "out of band"

FTP Protocol



FTP Protocol (Passive Mode)



Recap Where We're At

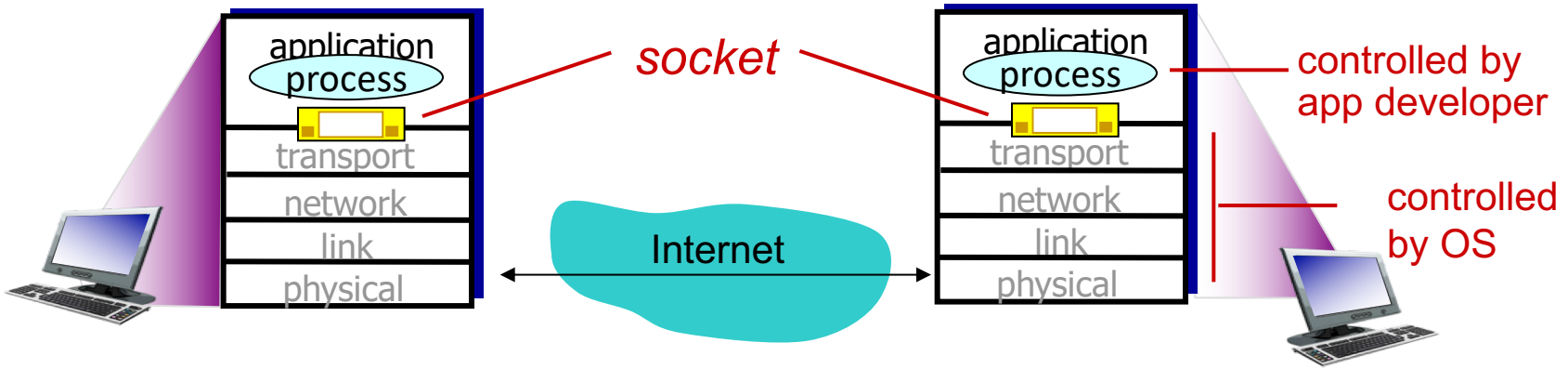
- DNS
  - DNSSEC
- SMTP
- FTP

Outline

- Socket Programming

Socket Programming

- Goal: learn how to build client/server applications that communicate using sockets
- Socket: door between application process and end-end-transport protocol



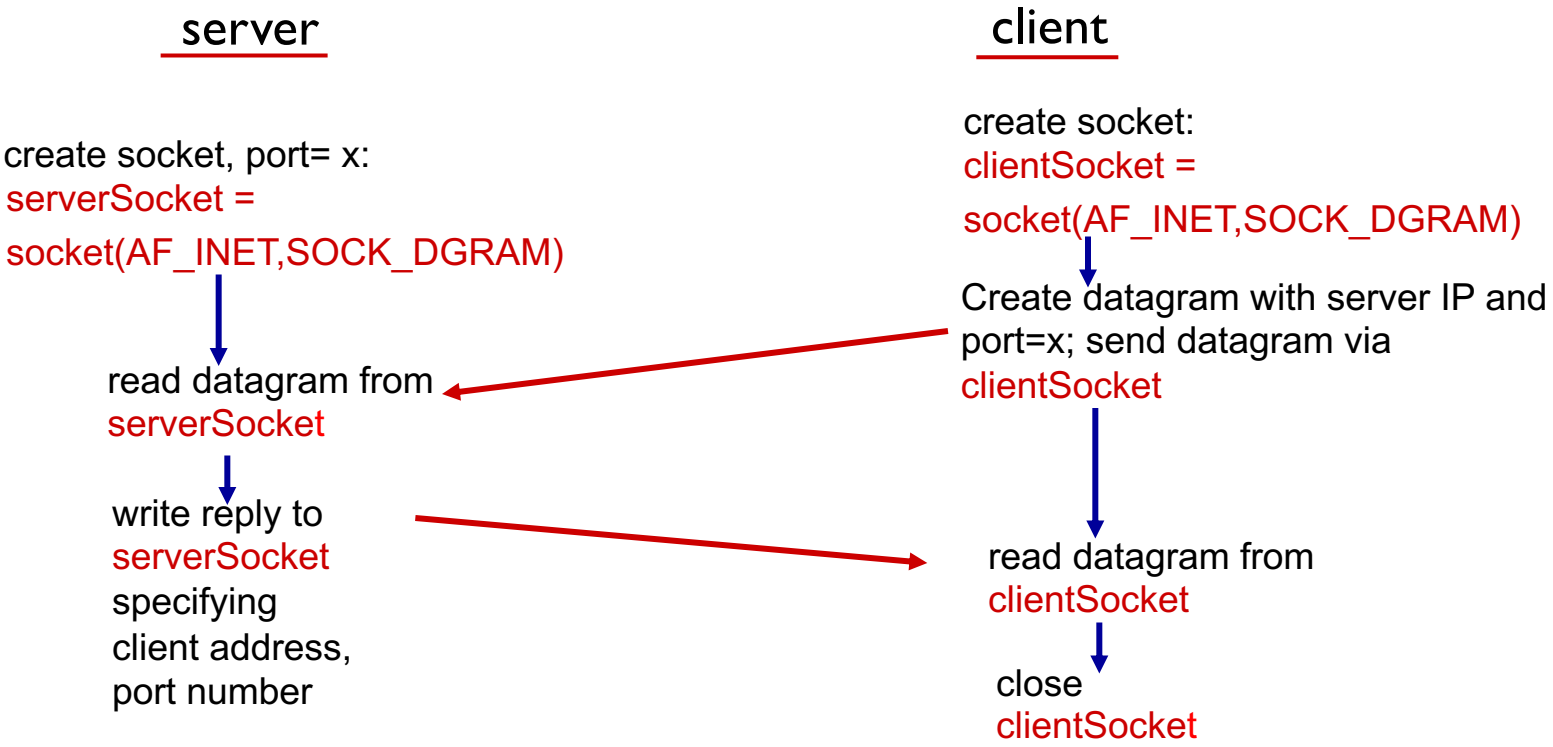
Socket Programming

- 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 the data to the server.
  2. The server receives the data and converts characters to uppercase.
  3. The server sends the modified data to the client.
  4. The client receives the modified data and displays the line on its screen.

Socket Programming with UDP

- 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



Pseudo App: UDP Server

Python UDPServer

include Python's socket library

```
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 1:
```

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

```
    message, clientAddress = serverSocket.recvfrom(2048)
    modifiedMessage = message.upper()
```

send upper case string back to this client

```
    serverSocket.sendto(modifiedMessage, clientAddress)
```

Pseudo App: UDP Client

Python UDPClient

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

create UDP socket for client

get user keyboard input

Attach server name, port to message; send into socket

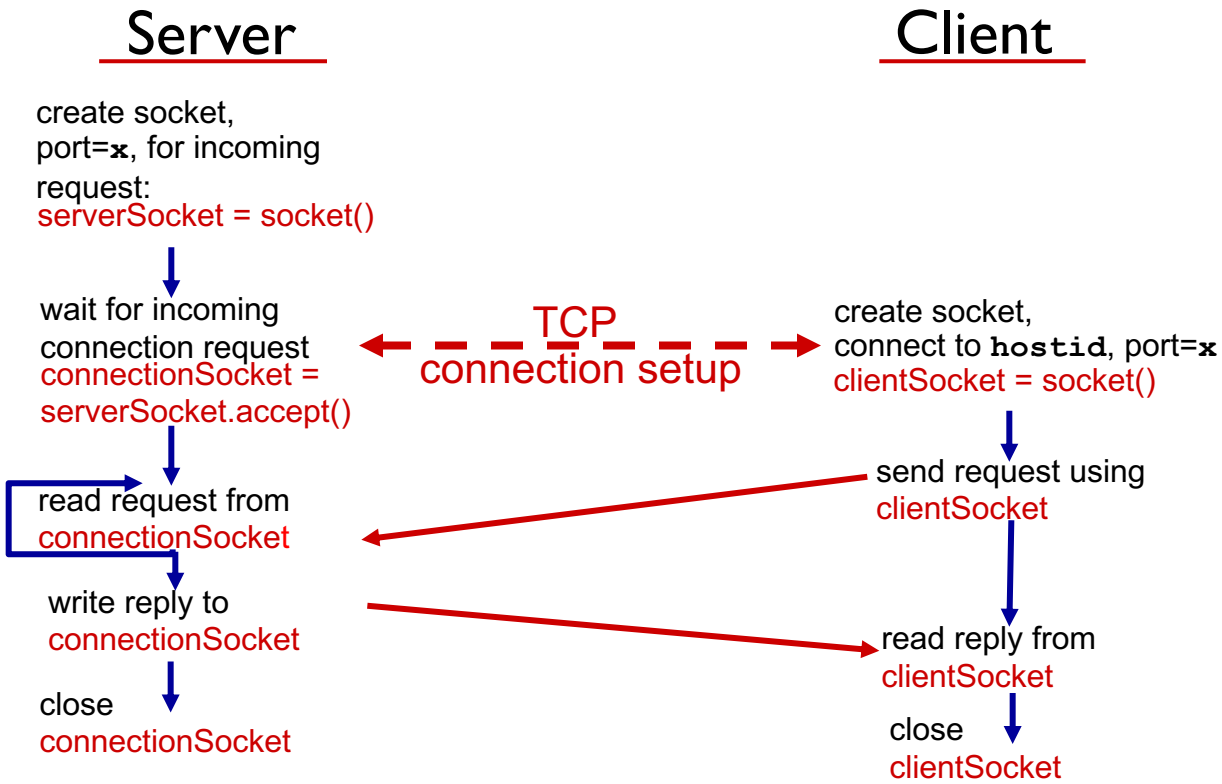
read reply characters from socket into string

print out received string and close socket

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

Client/server Socket Interaction: TCP



Pseudo App: TCP Server

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 1:
    connectionSocket, addr = serverSocket.accept()

    sentence = connectionSocket.recv(1024)
    capitalizedSentence = sentence.upper()
    connectionSocket.send(capitalizedSentence)
    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)

Pseudo App: TCP Client

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)
modifiedSentence = clientSocket.recv(1024)
print 'From Server:', modifiedSentence
clientSocket.close()
```

create TCP socket for server, remote port 12000

No need to attach server name, port