

Application Layer

- Reference
 - Chapter 02, *Computer Networking: A Top Down Approach*, 6/E, Jim Kurose, Keith Ross, Addison-Wesley
 - Adapted from part of the slides provided by the authors

CS3700 - Instructor: Dr. Zhu

1

Some network apps

- e-mail
- web
- text messaging
- remote login
- P2P file sharing
- multi-user network games
- streaming stored video
 - (YouTube, Hulu, Netflix)
- voice over IP (e.g., Skype)
- real-time video conferencing
- social networking
- search
- ...

CS3700 - Instructor: Dr. Zhu

2

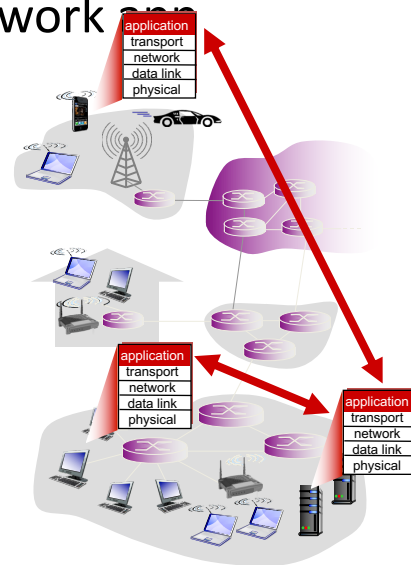
Creating a network

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



CS3700 - Instructor: Dr. Zhu

3

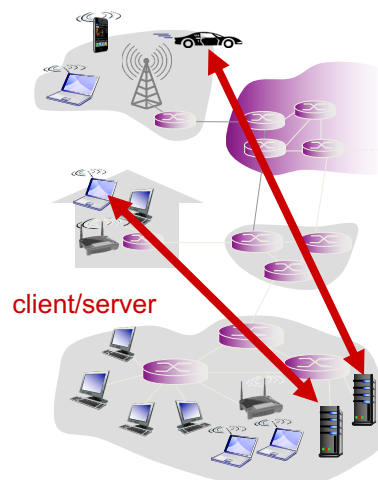
Client-server architecture

• 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

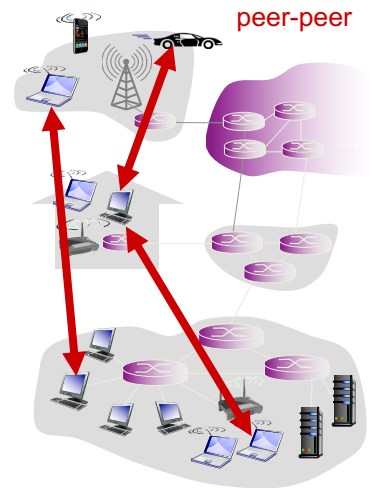


CS3700 - Instructor: Dr. Zhu

4

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, as well as new service demands
- peers are intermittently connected and change IP addresses
 - complex management



CS3700 - Instructor: Dr. Zhu

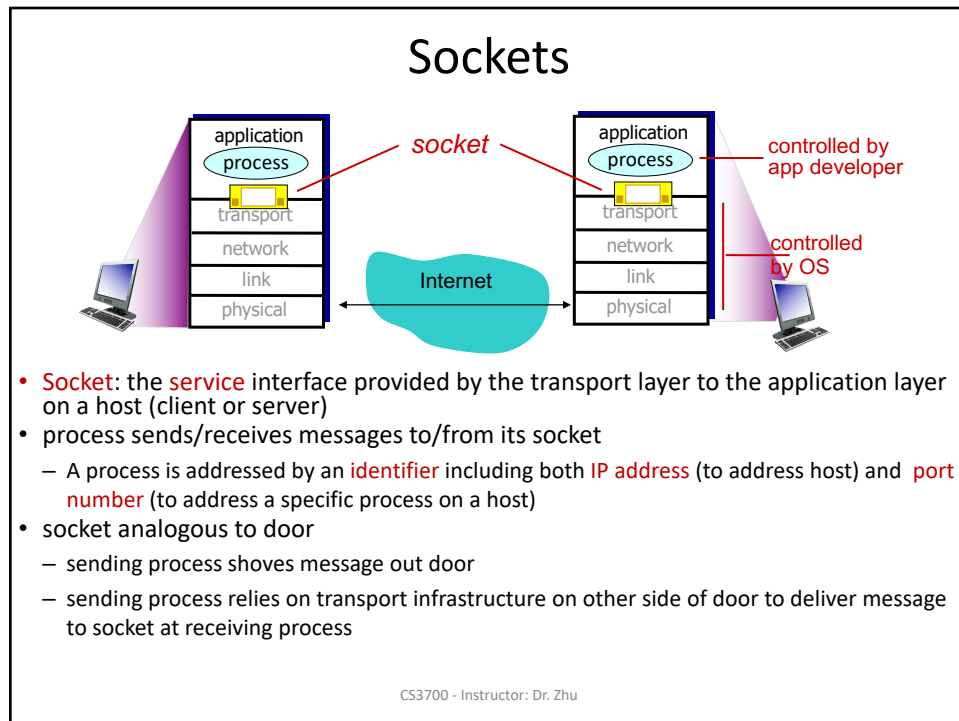
5

Processes

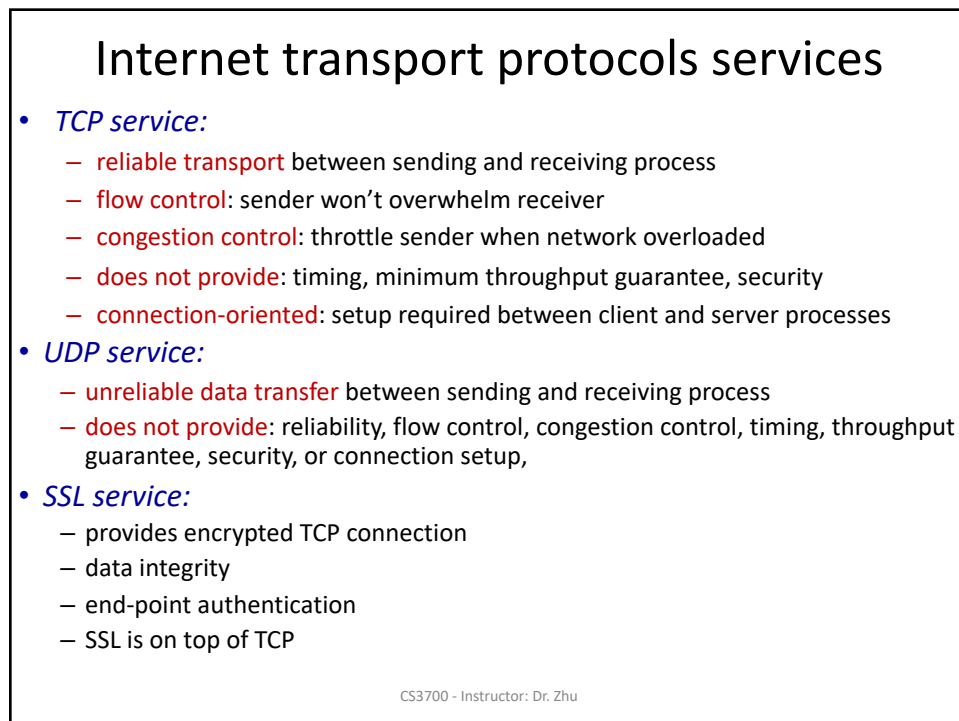
- *process*: program running within a host
 - *Message*: the unit/format of data at the application layer
 - processes in different hosts communicate by exchanging *messages*
- Clients and servers
 - *client process*: process that initiates communication
 - *server process*: process that waits to be contacted
- applications with P2P architectures have client processes & server processes

CS3700 - Instructor: Dr. Zhu

6



7



8

Socket programming in Java

- **TCP:**
 - Socket class used on server side: class ServerSocket
 - Socket class used on client side: class Socket
- **UDP:** Socket class used on both client and server side: DatagramSocket

Application Example:

1. Client reads a line of characters (data) from its keyboard, displays the data on its screen, 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.
5. Repeat Steps 1 – 4 until “Bye.” is read by the client.

CS3700 - Instructor: Dr. Zhu

9

Socket programming *with UDP*

- **Application viewpoint:**
 - UDP provides unreliable transfer of groups of bytes (“datagrams”) between client and server
 - **UDP: no “connection” between client & server**
 - no handshaking before sending data
 - sender explicitly attaches IP destination address and port # to each packet
 - rcvr extracts sender IP address and port# from received packet
 - **UDP: transmitted data may be lost or received out-of-order**
- See source codes after reading the next slide:
- UDP_ToUpperCase/UDPClient.java
 - UDP_ToUpperCase/UDPServer.java

CS3700 - Instructor: Dr. Zhu

10

Client/server socket interaction: UDP

server (running on serverIP)

```
import java.net.*;
DatagramSocket udpServerSocket;
int port = x;
udpServerSocket =
    new DatagramSocket(port);
```

↓
read datagram from
udpServerSocket

↓
write reply to
udpServerSocket
specifying client address and
port number, which are included
in the received datagram

Client

```
import java.net.*;
DatagramSocket udpSocket;
udpSocket = new DatagramSocket();
```

↓
Create datagram with serverIP and
port=x; send datagram via
udpSocket

↓
read datagram from
udpSocket

↓
close
udpSocket

CS3700 - Instructor: Dr. Zhu

Application 2-11

11

Socket programming *with TCP*

- **application viewpoint:**
 - TCP provides reliable, in-order byte-stream transfer (“pipe”) between client and server
 - **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
- ➔ See source codes after reading the next slide:
- TCP_ToUpperCase/TCPClient.java
 - TCP_ToUpperCase/TCPServer.java

CS3700 - Instructor: Dr. Zhu

12

Client/server socket interaction: TCP

server (running on `hostid`)

```
import java.net.*;
ServerSocket serverTcpSocket ;
int port=x; create and listen to
socket for incoming request:
serverTcpSocket = new ServerSocket(x);
```

wait for and create a socket for
incoming connection request

```
Socket clientTcpSocket
= serverTcpSocket.accept();
```

read request from
`clientTcpSocket`

write reply to
`clientTcpSocket`

close
`clientTcpSocket`

client

```
import java.net.*;
```

declare a client socket,
`Socket tcpSocket = null`;
connect to `serverid`, port=`x`
`tcpSocket = new Socket(serverid, x);`

send request using
`tcpSocket`

read reply from
`tcpSocket`

close
`tcpSocket`

TCP
connection setup

CS3700 - Instructor: Dr. Zhu

13

TCP Server using Multiple Threads

- Multiple threads are used by TCP server to support multiple clients
- Upon accepting a connection request from a client, TCP Server starts a new thread designated to handle the TCP connection with this client.

```
new TCPServerThread(serverTCPSocket.accept()).start();
```

- One of the two ways to start a new thread in Java

- Extend the Thread class:

```
public class TCPServerThread extends Thread { ... }
```

- o TCPServerThread: subclass or extending class, Thread: super class

- The extending class must override the `run()` method of the Thread class, which is the entry point for the new thread

```
public void run() { ... }
```

- To start a new thread, create an instance of the extending class and call `start()`

```
new TCPServerThread(serverTCPSocket.accept()).start();
```

- This new thread stops at the end of the `run()` method

- Each thread may have a name, in the constructor of the extending class,
`super("TCPServerThread")`

➔ See `TCP_toUpperCase_MultiThread/*.java`

CS3700 - Instructor: Dr. Zhu

14

Internet apps: application, transport protocols

<u>application</u>	<u>application layer protocol</u>	<u>underlying transport protocol</u>
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

CS3700 - Instructor: Dr. Zhu

15

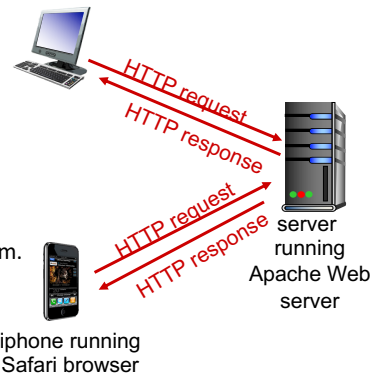
HTTP Overview

- **web page** consists of **objects**
 - **Objects** can be HTML file, JPEG image, Java applet, audio file,...
 - web page consists of **base HTML-file** which includes **several referenced objects**
 - each **object** is addressable by a URL, e.g.,
www.someschool.edu / someDept/pic.gif

host name

path name

- **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
- **HTTP specifications ([RFC 1945] and [RFC 2616])**
 - define only the communication protocol between the client HTTP program and the server HTTP program.
 - have nothing to do with how a client interprets a Web page



CS3700 - Instructor: Dr. Zhu

16

HTTP Connections

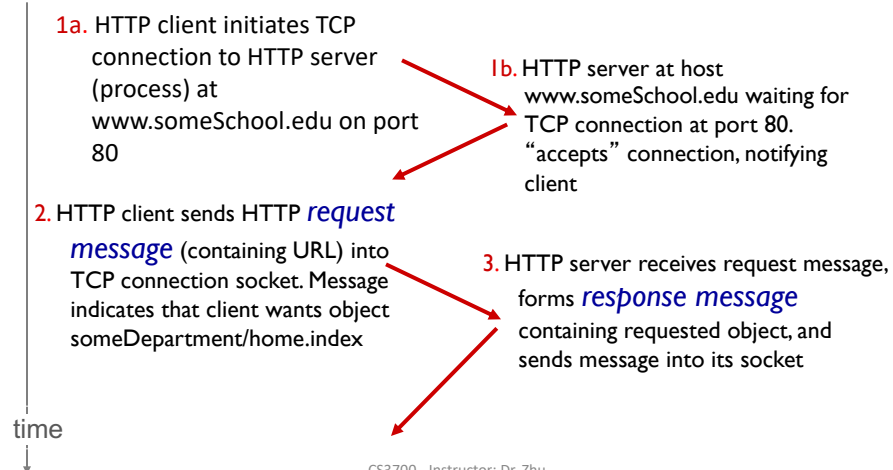
- **HTTP uses TCP:**
 - client initiates TCP connection (creates socket) to server, port 80
 - server accepts TCP connection from client
 - HTTP messages (application-layer protocol messages) exchanged between browser (HTTP client) and Web server (HTTP server)
 - TCP connection closed
- **HTTP is “stateless”**
 - server maintains no information about past client requests
- **non-persistent HTTP v.s. persistent HTTP**
 - **non-persistent HTTP:** at most one object sent over TCP connection; connection is then closed
 - downloading multiple objects required multiple connections
 - **persistent HTTP:** multiple objects can be sent over single TCP connection between client, server

CS3700 - Instructor: Dr. Zhu

17

Non-persistent HTTP

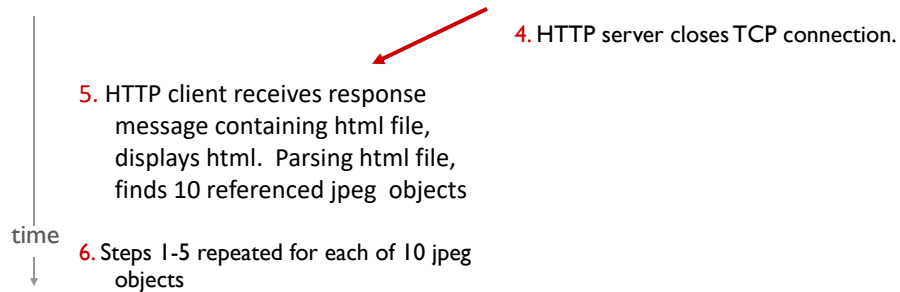
suppose user enters URL: `www.someSchool.edu/someDepartment/home.index` (contains text, references to 10 jpeg images)



CS3700 - Instructor: Dr. Zhu

18

Non-persistent HTTP (cont.)

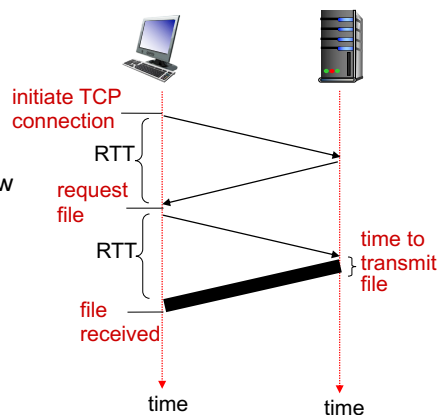


CS3700 - Instructor: Dr. Zhu

19

Non-persistent 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
 - one RTT for HTTP request and first few bytes of HTTP response to return
 - file transmission time
 - non-persistent HTTP response time = $2\text{RTT} + \text{file transmission time}$



CS3700 - Instructor: Dr. Zhu

20

Persistent HTTP

- **non-persistent HTTP issues:**
 - requires 2 RTTs per object
 - OS overhead for each TCP connection
 - browsers often open parallel TCP connections to fetch referenced objects
- **persistent HTTP:**
 - server leaves connection open after sending response
 - subsequent HTTP messages between same client/server sent over open connection
 - client sends requests as soon as it encounters a referenced object
 - as little as one RTT for all the referenced objects

CS3700 - Instructor: Dr. Zhu

21

HTTP request message

- two types of HTTP messages: *request, response*
- **HTTP request message:** from HTTP client to HTTP server
 - ASCII (human-readable format)

The diagram shows an HTTP request message in ASCII format. Annotations with arrows point to specific parts of the message:

- request line (GET, POST, HEAD commands):** Points to the first line: `GET /index.html HTTP/1.1\r\n`
- header lines:** A bracket groups the following lines: `Host: www-net.cs.umass.edu\r\n`, `User-Agent: Firefox/3.6.10\r\n`, `Accept: text/html,application/xhtml+xml\r\n`, `Accept-Language: en-us,en;q=0.5\r\n`, `Accept-Encoding: gzip,deflate\r\n`, `Accept-Charset: ISO-8859-1,utf-8;q=0.7\r\n`, `Keep-Alive: 115\r\n`, and `Connection: keep-alive\r\n`
- carriage return, line feed at start of line indicates end of header lines:** Points to the `\r\n` at the end of the last header line.
- carriage return character line-feed character:** Two arrows point to the `\r` and `\n` characters at the end of the request line.

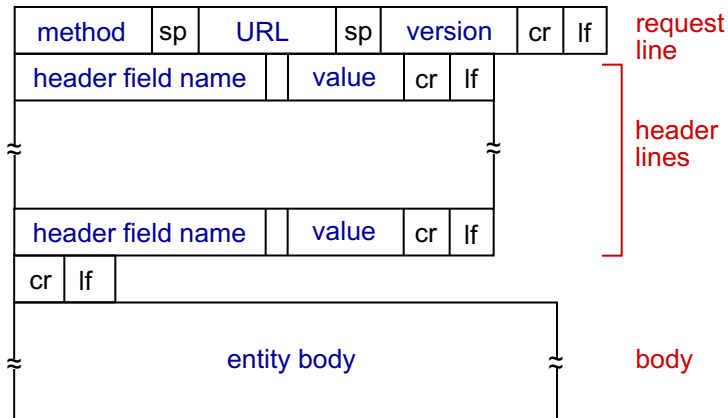
```

GET /index.html HTTP/1.1\r\n
Host: www-net.cs.umass.edu\r\n
User-Agent: Firefox/3.6.10\r\n
Accept: text/html,application/xhtml+xml\r\n
Accept-Language: en-us,en;q=0.5\r\n
Accept-Encoding: gzip,deflate\r\n
Accept-Charset: ISO-8859-1,utf-8;q=0.7\r\n
Keep-Alive: 115\r\n
Connection: keep-alive\r\n
\r\n
  
```

CS3700 - Instructor: Dr. Zhu

22

HTTP request message: general format



CS3700 - Instructor: Dr. Zhu

23

HTTP Request message: Method types

- HTTP/1.0:
 - **GET**: Just request a webpage by leaving the “Entity Body” empty
 - **POST**: Upload form input to server in “entity body” when user fills out a form
 - URL Method: Use **GET** method and upload input in **URL field** of **request line**
`www.somesite.com/animalsearch?monkeys&banana`
 - **HEAD**: = GET but server leaves requested object out of response(for debugging)
- HTTP/1.1:
 - GET, POST, HEAD
 - **PUT**: Upload file (object) in entity body to path specified in URL field on a server
 - **DELETE**: Delete file specified in the URL field on a server

CS3700 - Instructor: Dr. Zhu

24

HTTP response message: from Server to Client

status line
(protocol
status code
status phrase)

header
lines

data, e.g.,
requested
HTML file

```
HTTP/1.1 200 OK\r\n
Date: Sun, 26 Sep 2010 20:09:20 GMT\r\n
Server: Apache/2.0.52 (CentOS)\r\n
Last-Modified: Tue, 30 Oct 2007 17:00:02
GMT\r\n
ETag: "17dc6-a5c-bf716880"\r\n
Accept-Ranges: bytes\r\n
Content-Length: 2652\r\n
Keep-Alive: timeout=10, max=100\r\n
Connection: Keep-Alive\r\n
Content-Type: text/html; charset=ISO-8859-1\r\n
\r\n
data data data data data ...
```

CS3700 - Instructor: Dr. Zhu

25

HTTP response status codes

- status code appears in 1st line in server-to-client response message.
- some sample codes:
 - 200 OK**
 - request succeeded, requested object later in this msg
 - 301 Moved Permanently**
 - requested object moved, new location specified later in this msg (Location:)
 - 400 Bad Request**
 - request msg not understood by server
 - 404 Not Found**
 - requested document not found on this server
 - 505 HTTP Version Not Supported**

CS3700 - Instructor: Dr. Zhu

26

Trying out HTTP (client side) using Telnet

1. Telnet to a Web server of your choice

```
telnet www.msudenver.edu 80 | tee -a <file name>
```

opens TCP connection to port 80 (default HTTP server port) at www.msudenver. anything typed in will be sent to port 80 at www.msudenver

2. type in a GET HTTP request:

```
GET /about/ HTTP/1.1
Host: www.msudenver.edu
```

by typing this in (hit carriage return twice), you send this minimal (but complete) GET request to HTTP server

3. look at response message sent by HTTP server!

CS3700 - Instructor: Dr. Zhu

27

HTTP/2

- Reference

- “Introduction to HTTP/2,” by Ilya Grigorik and Surma
https://developers.google.com/web/fundamentals/performance/http2/#top_of_page
- Chapter 12 HTTP/2, “High Performance Browser Networking,” by Ilya Grigorik
<https://hpbn.co/http2/>
- RFC 7540: Hypertext Transfer Protocol Version 2 (HTTP/2)
- RFC 7541

CS3700 - Instructor: Dr. Zhu

28

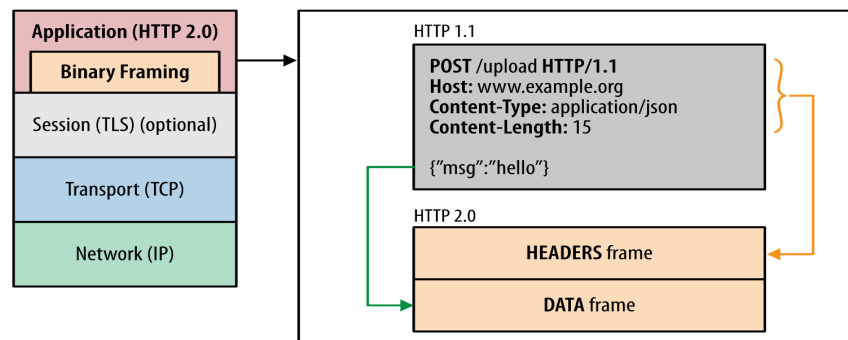
SPDY and HTTP/2

- **SPDY**: originally proposed by Google in 2011
 - An extension to HTTP 1.x
 - Implements all the major features of HTTP/2
 - Serves as a testbed for improvements to HTTP
 - Became the basis of HTTP/2 in 2012
 - Was ceased to exist in 2016, making way to HTTP/2
- **HTTP/2**
 - Uses a single, multiplexed connection to solve the HOL blocking in HTTP/1
 - Max connection limit per domain can be ignored
 - Compresses header data and sends it in a concise, binary format
 - Better than the plain text format used previously
 - Less need for popular HTTP 1.1 optimizations

CS3700 - Instructor: Dr. Zhu

29

Binary Framing Layer in HTTP/2



- Core of all performance enhancements of HTTP/2: the new binary framing layer
 - It dictates how client and server encapsulate and transfer HTTP messages
 - a new optimized encoding mechanism between the *socket interface* (TLS or TCP) and the higher HTTP API exposed to our applications
 - the HTTP semantics, such as *verbs*, *methods*, and *headers*, are unaffected
- Encoding is different: all HTTP/2 communication is split into smaller messages and frames, each of which is encoded in binary format.

CS3700 - Instructor: Dr. Zhu

30

Binary Framing

- Once an HTTP/2 connection is established, the client and server communicate by exchanging *frames*
 - Frames serve as the smallest unit of communication within the protocol
 - All frames share a **common 9-byte header**.

Bit	+0..7		+8..15	+16..23	+24..31		
0	Length				Type		
32	Flags						
40	R	Stream Identifier					
...	Frame Payload						

- The 24-bit **length** (of frame payload) field allows a frame to carry up to 2^{24} bytes of data
- The 8-bit **type** field determines the meaning of the flags and payload of the frame, such as **DATA**, **HEADERS**, **PRIORITY**, **RST_STREAM**, ...
- The 8-bit **flags** field communicates frame-type specific Boolean flags (the un-used flags for a given type in a frame must be cleared as 0's).
- The 1-bit **reserved** field is always set to 0.
- The 31-bit **stream identifier** field uniquely identifies the HTTP/2 stream.

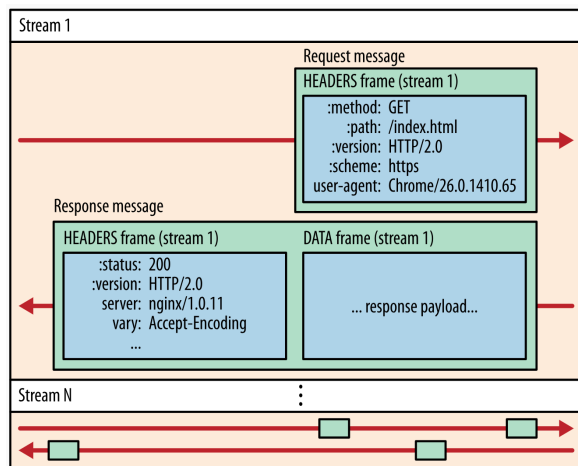
CS3700 - Instructor: Dr. Zhu

31

HTTP/2 Streams, Messages, and Frames

- This binary framing mechanism changes the data exchange between client/server.
- All communication is performed over a **single TCP connection** that carries any number of bidirectional streams.

- Stream**: a bidirectional flow of bytes in an HTTP connection carrying one or more messages
- Message**: a complete sequence of frames that map to a logical request or response message.
- Frame**: The smallest unit of communication, each containing a frame header, which contains an ID to identify the stream to which the frame belongs.



CS3700 - Instructor: Dr. Zhu

32

HTTP/2 Specification and Support

- HTTP/2 **Specification:**
 - Started with SPDY – draft 3
 - Two specifications: RFC7540 (HTTP/2) and RFC7541 (HPACK: header compression)
 - Implementations: HTTP/2 over TLS (**h2**) and HTTP/2 over TCP (**h2c**)
- HTTP/2 over **TLS (h2)**:
 - HTTP/2 was defined with TLS as optional
 - Firefox and Chrome: implement HTTP/2 only over TLS
 - Today, only HTTPS:// is allowed for HTTP/2
 - TLS must be at least v1.2, w/ cipher suite restrictions
<https://http2.github.io/http2-spec/#BadCipherSuites>
- HTTP/2 over **TCP (h2c)**:
 - Uses the Upgrade header
 - Already supported in CURL
 - Plans to support on IE (already?)

CS3700 - Instructor: Dr. Zhu

33

Upgrading to HTTP/2

- HTTP/1.x will be around for at least another decade
 - Most servers and clients will have to support both HTTP/1.x and HTTP/2.
- An HTTP/2 client and server must discover and negotiate which protocol will be used prior to exchanging application data.
 - Negotiating HTTP/2 via a secure connection with TLS and ALPN
 - the recommended mechanism to deploy and negotiate HTTP/2
 - the client and server negotiate the desired protocol as part of the TLS handshake without adding any extra latency or roundtrips
 - Or upgrading a plaintext connection to HTTP/2 w/o prior knowledge
 - Establishing an HTTP/2 connection over a regular, non-encrypted channel is still possible
 - both HTTP/1.x and HTTP/2 run on the same port (80)
 - the client has to use the HTTP Upgrade mechanism to negotiate the appropriate protocol
 - Or initiating a plaintext HTTP/2 connection with prior knowledge

CS3700 - Instructor: Dr. Zhu

34

“HTTP Upgrade” for negotiation (port 80)

```
GET /page HTTP/1.1
Host: server.example.com
Connection: Upgrade, HTTP2-Settings
Upgrade: h2c ❶
HTTP2-Settings: (SETTINGS payload) ❷
```

```
HTTP/1.1 200 OK ❸
Content-length: 243
Content-type: text/html
```

(... HTTP/1.1 response ...)

(or)

```
HTTP/1.1 101 Switching Protocols ❹
Connection: Upgrade
Upgrade: h2c
```

(... HTTP/2 response ...)

- ❶ Initial HTTP/1.1 request with HTTP/2 upgrade header
- ❷ Base64 URL encoding of HTTP/2 SETTINGS payload
- ❸ Server declines upgrade, returns response via HTTP/1.1
- ❹ Server accepts HTTP/2 upgrade, switches to new framing

35

Try HTTP/2 over tcp to a HTTP/1.1 server

telnet cs3700a.msudenver.edu 80

– Apache 2.4.18 that does not support HTTP/2 runs on cs3700a.

----- client's request via telnet (tcp) connection -----

```
GET / HTTP/1.1
Host: cs3700a.msudenver.edu
Connection: Upgrade, HTTP2-Settings
Upgrade: h2c
HTTP2-Settings:
User-Agent: whatever
```

----- apache server's response -----

```
HTTP/1.1 200 OK
Date: Wed, 16 May 2018 16:14:02 GMT
Server: Apache/2.4.18 (Ubuntu)
Last-Modified: Wed, 16 May 2018 16:03:43 GMT
ETag: "2d-56c54dced9e02"
Accept-Ranges: bytes
Content-Length: 45
Content-Type: text/html
```

```
<html><body><h1>It works!</h1></body></html>
```

CS3700 - Instructor: Dr. Zhu

36

Try HTTP/2 over tcp to a HTTP/2 Virtual Server

telnet cs3700a.msudenver.edu 8080

- Apache 2.4.39 w/ http2 runs on cs3700a at port 8080 (adding “LoadModulehttp2_module modules/mod_http2.so” and “Protocols h2 h2c http/1.1” in the file titled “httpd.conf”)

----- client's request via telnet (tcp) connection -----

GET / HTTP/1.1

Host: cs3700a.msudenver.edu

Connection: Upgrade, HTTP2-Settings

Upgrade: h2c

HTTP2-Settings:

User-Agent: whatever

----- apache server's response -----

HTTP/1.1 101 Switching Protocols

Upgrade: h2c

Connection: Upgrade

j?a??m_J

45_?I|???M-<html><body><h1>It works!</h1></body></html>

CS3700 - Instructor: Dr. Zhu

37

Try HTTP/1 to a HTTP2 Virtual Server

telnet cs3700a.msudenver.edu 80

- Apache 2.4.33 w/ http2 runs on cs3700a at port 80 (adding “LoadModulehttp2_module modules/mod_http2.so” and “Protocols h2 h2c http/1.1” in the file titled “httpd.conf”)

----- client's request via telnet (tcp) connection -----

GET / HTTP/1.1

Host: cs3700a.msudenver.edu

User-Agent: whatever

----- apache server's response -----

HTTP/1.1 200 OK

Date: Mon, 20 Jan 2020 22:26:48 GMT

Server: Apache/2.4.39 (Unix)

Upgrade: h2,h2c

Connection: Upgrade

Last-Modified: Mon, 11 Jun 2007 18:53:14 GMT

ETag: "2d-432a5e4a73a80"

Accept-Ranges: bytes

Content-Length: 45

Content-Type: text/html

<html><body><h1>It works!</h1></body></html>

CS3700 - Instructor: Dr. Zhu

38

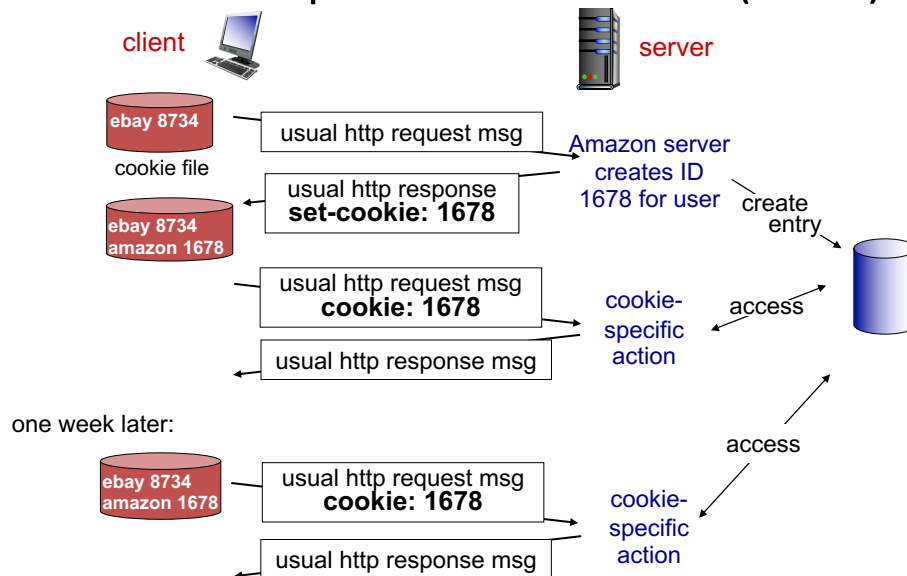
Cookies: keep User-Server State

- **cookies can be used for:**
 - authorization
 - shopping carts
 - recommendations
 - user session state (Web e-mail)
- **how to keep “state”:**
 - protocol endpoints: maintain state at sender/receiver over multiple transactions
 - cookies: http messages carry state
- **Cookies: an invasion of privacy**
 - cookies permit sites to learn a lot about you
 - you may supply name and e-mail to sites that may sell your information to a third party

CS3700 - Instructor: Dr. Zhu

39

Cookies: keep User-Server State (Cont.)

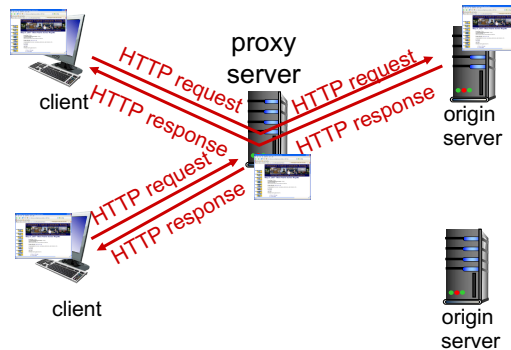


CS3700 - Instructor: Dr. Zhu

40

Web caches (proxy server)

- **goal:** satisfy client request without involving origin server
 - user sets browser: Web accesses via cache
 - browser sends all HTTP requests to cache
 - object in cache: cache returns object
 - else cache requests object from origin server, then returns object to client
- cache acts as both client and server
- typically cache is installed by ISP (university, company, residential ISP)
- Web caching:
 - reduce response time for client request
 - reduce traffic on an institution's access link
 - enables “poor” content providers to effectively deliver content

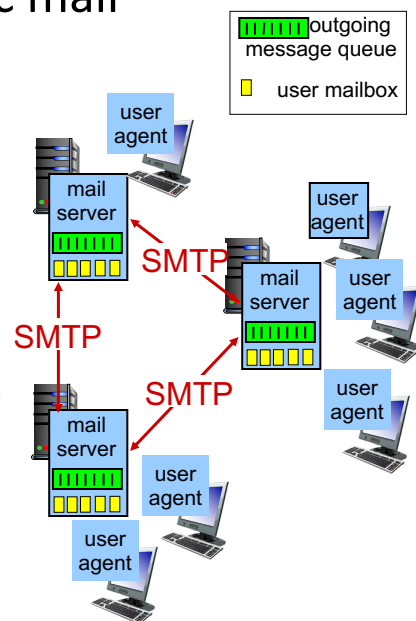


CS3700 - Instructor: Dr. Zhu

41

Electronic mail

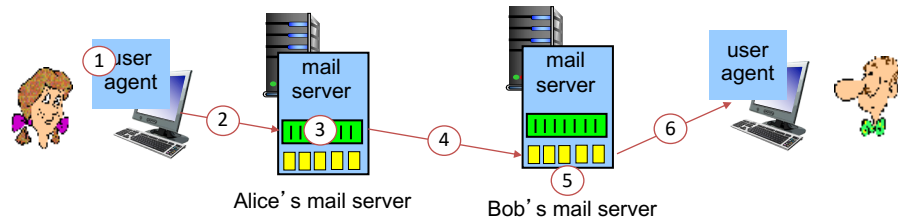
- Three major components:
 - user agents
 - mail servers
 - simple mail transfer protocol: SMTP
- **User Agent** (a.k.a. “mail reader”)
 - composing, editing, reading mail messages
 - Outlook, Thunderbird, iPhone mail client, ...
 - PUSH outgoing mail onto server via **SMTP**
 - PULL incoming mail from server via mail access protocols
- **mail servers:**
 - **mailbox** contains incoming messages for user
 - **message queue** of outgoing (to be sent) mail messages
 - **SMTP protocol** is used between mail servers to send email messages
 - **client:** mail server sending a mail
 - **server:** mail server receiving a mail



CS3700 - Instructor: Dr. Zhu

42

Scenario: Alice sends message to Bob



- 1) Alice uses UA to compose message "to" bob@someschool.edu
- 2) Alice's UA PUSHes message to her mail server via SMTP; 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

*Direct TCP connection between two mail servers w/o any intermediate mail sever

CS3700 - Instructor: Dr. Zhu

43

Electronic Mail: SMTP [RFC 2821]

- uses TCP to reliably transfer email message from client to server
- Server, the receiving mail server, listens at port 25
- direct transfer: from sending server to receiving server
- three phases of transfer
 - handshaking (greeting)
 - transfer of messages
 - closure
- command/response interaction (like HTTP, FTP)
 - **commands:** ASCII text
 - HELO, MAIL FROM, RCPT TO, DATA, QUIT**
 - **response:** status code and phrase, ASCII text
 - status code: **220, 250, 354, 221**, etc.
- **Mail messages (DATA)** must be in 7-bit ASCII
 - Each character is encoded using 8 bits, a Byte

CS3700 - Instructor: Dr. Zhu

44

Mail message format

- SMTP: protocol for exchanging email messages
- RFC 822: standard for text message format:

- header lines, e.g.,

To: ... \r\n

From: ... \r\n

Subject: ... \r\n

NOT SMTP MAIL FROM, RCPT TO commands!

- Blank line between header and body, e.g.,

\r\n

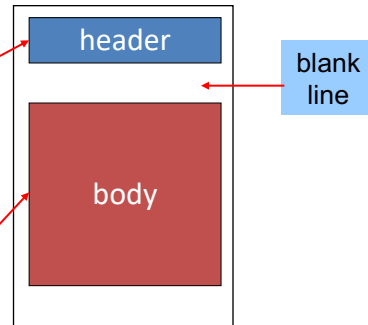
- Body: the “message”, e.g.,

how are you doing?\r\n

etc.\r\n

- CRLF.CRLF marks end of message, e.g.,

.\r\n



CS3700 - Instructor: Dr. Zhu

45

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: To: bob@hamburger.edu
C: From: alice@crepes.fr
C: Subject: ketchup and pickles
C:
C: Do you like ketchup?
C: How about pickles?
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 hamburger.edu closing connection
```

CS3700 - Instructor: Dr. Zhu

46

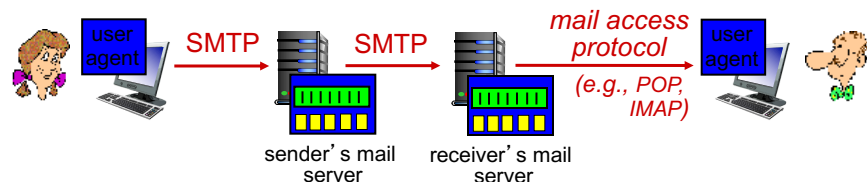
Try SMTP interaction for yourself:

- **telnet hamburger.edu 25**
 - Metro email server: smtp.msudenver.edu
vmwhub01.services.metro
vmwhub02.services.metro
 - Tried the following (<http://www.emailaddressmanager.com/tips/mail-settings.html>)
 - Google Gmail Outgoing Mail Server (SMTP): smtp.gmail.com (SSL) on port 465
 - Yahoo Outgoing Mail Server (SMTP) - smtp.mail.yahoo.com (SSL enabled, port 995)
 - MSN Outgoing Mail Server - smtp.email.msn.com
 - Lycos Outgoing Mail Server - smtp.mail.lycos.com
 - see 220 reply from server
 - enter HELO, MAIL FROM, RCPT TO, DATA, QUIT commands
- an email may be sent to your email account at msudenver.edu sometime, but it may be classified as a junk mail.
 → Most likely, at certain point, the command will be reject due to blocked telnet port, invalid address, unauthorized sender issues, etc.

CS3700 - Instructor: Dr. Zhu

47

Mail access protocols



- SMTP: PUSH mail from UA to sender's server or from sender's server to receiver's server
- mail access protocol: retrieval from server (PULL)
 - **POP**: Post Office Protocol [RFC 1939]: authorization, download
 - **IMAP**: Internet Mail Access Protocol [RFC 1730]: more features, including manipulation of stored msgs on server
 - **HTTP**: gmail, Hotmail, Yahoo! Mail, etc.

CS3700 - Instructor: Dr. Zhu

48

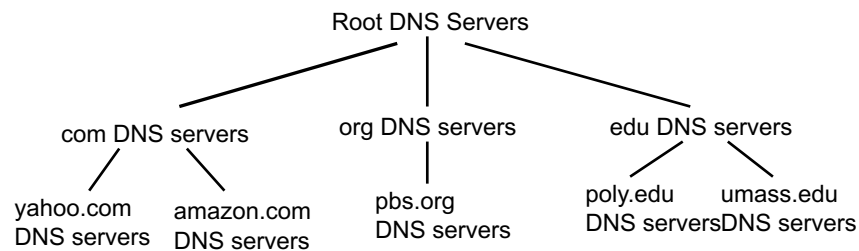
DNS: domain name system

- *distributed database*
 - implemented in hierarchy of many *name servers*
- *application-layer protocol*: hosts, name servers communicate to *resolve* names (address/name translation)
 - core Internet function
 - complexity at network's "edge"
- *DNS services*
 - hostname to IP address translation
 - host aliasing: canonical (real) names and aliases
 - mail server aliasing
 - load distribution
 - replicated Web servers: many IP addresses correspond to one name
- *why not centralize DNS?*
 - single point of failure
 - traffic volume
 - distant centralized database
 - maintenance

CS3700 - Instructor: Dr. Zhu

49

DNS: a distributed, hierarchical database

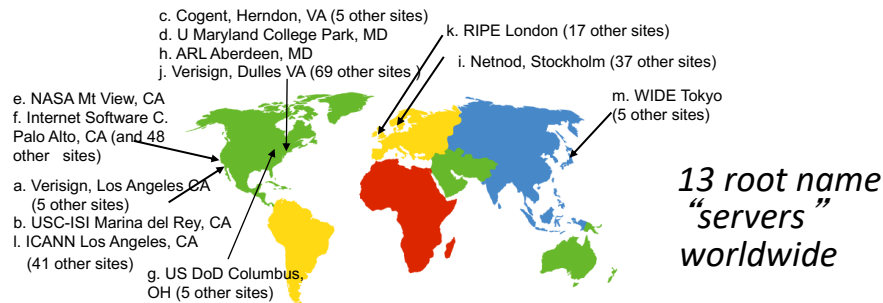


- *client wants IP for www.amazon.com; 1st approx:*
 - client queries root server to find com DNS server
 - client queries .com DNS server to get amazon.com DNS server
 - client queries amazon.com DNS server to get IP address for www.amazon.com

CS3700 - Instructor: Dr. Zhu

50

DNS: root name servers



- contacted by local name server that can not resolve name
- root name server:
 - contacts authoritative name server if name mapping not known
 - gets mapping
 - returns mapping to local name server

CS3700 - Instructor: Dr. Zhu

51

TLD servers 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.: uk, fr, ca, jp
 - Network Solutions maintains servers for .com TLD
 - Educause for .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

CS3700 - Instructor: Dr. Zhu

52

Local DNS name server

- 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

CS3700 - Instructor: Dr. Zhu

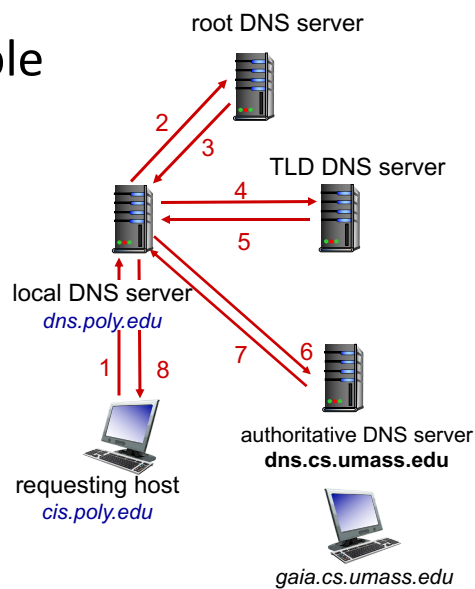
53

DNS name resolution example

- host at cis.poly.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”



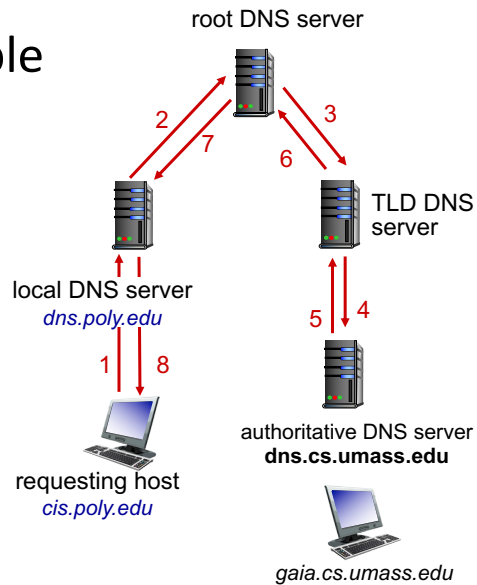
CS3700 - Instructor: Dr. Zhu

54

DNS name resolution example

recursive query:

- puts burden of name resolution on contacted name server
- heavy load at upper levels of hierarchy?



CS3700 - Instructor: Dr. Zhu

55

DNS: caching, updating records

- 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
- 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

CS3700 - Instructor: Dr. Zhu

56

DNS records

- **DNS**: distributed db storing resource records (**RR**)

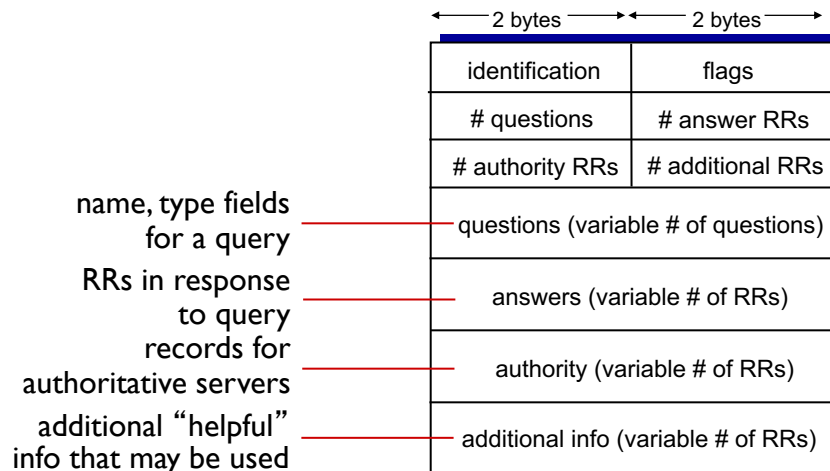
RR format: (**name**, **value**, **type**, **ttl**)

- **type=A**:
 - **name** is hostname
 - **value** is IP address
- **type=NS**:
 - **name** is domain (e.g., amazon.com, or .com)
 - **value** is hostname of a name server (e.g., the hostname of an authoritative dns server for the .amazon.com domain, or the hostname of a TLD dns server for the .com 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**:
 - **name** is domain (e.g., amazon.com)
 - **value** is name of mailserver associated with **name**

CS3700 - Instructor: Dr. Zhu

57

DNS protocol, messages



CS3700 - Instructor: Dr. Zhu

58

Inserting records into 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 two RRs into .com TLD server:
 - (networkutopia.com, dns1.networkutopia.com, NS)
 - (dns1.networkutopia.com, 212.212.212.1, A)
- create authoritative server type A record for www.networkutopia.com; type MX record for networkutopia.com

CS3700 - Instructor: Dr. Zhu

59

Attacking DNS

- 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
- Redirect attacks
 - Man-in-middle
 - Intercept queries
 - DNS poisoning
 - Send bogus replies to DNS server, which caches
- Exploit DNS for DDoS
 - Send queries with spoofed source address: target IP
 - Requires amplification

CS3700 - Instructor: Dr. Zhu

60