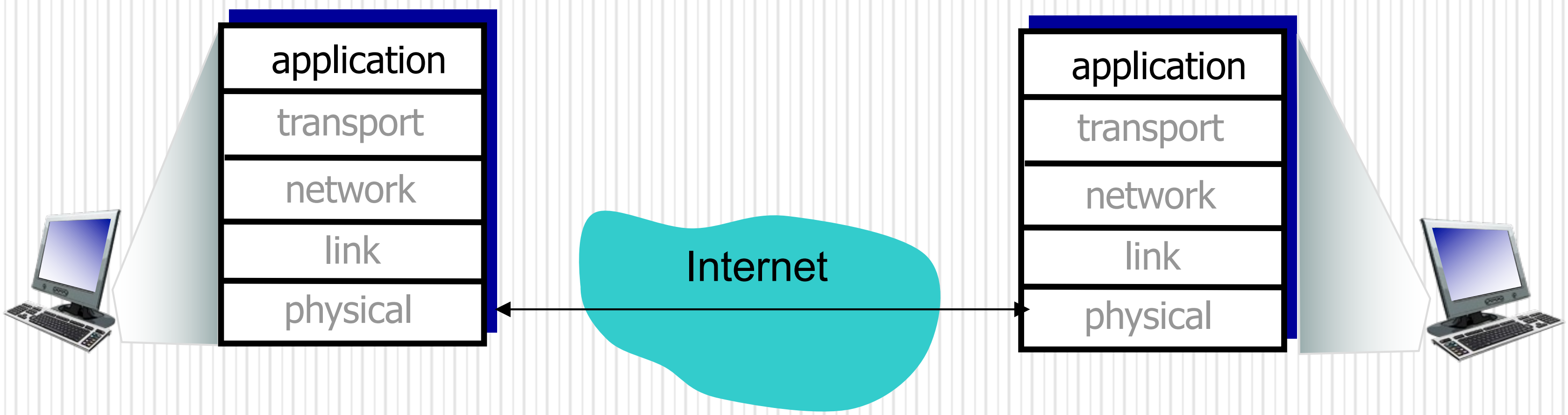
Application Layer

K & R Chapter 2

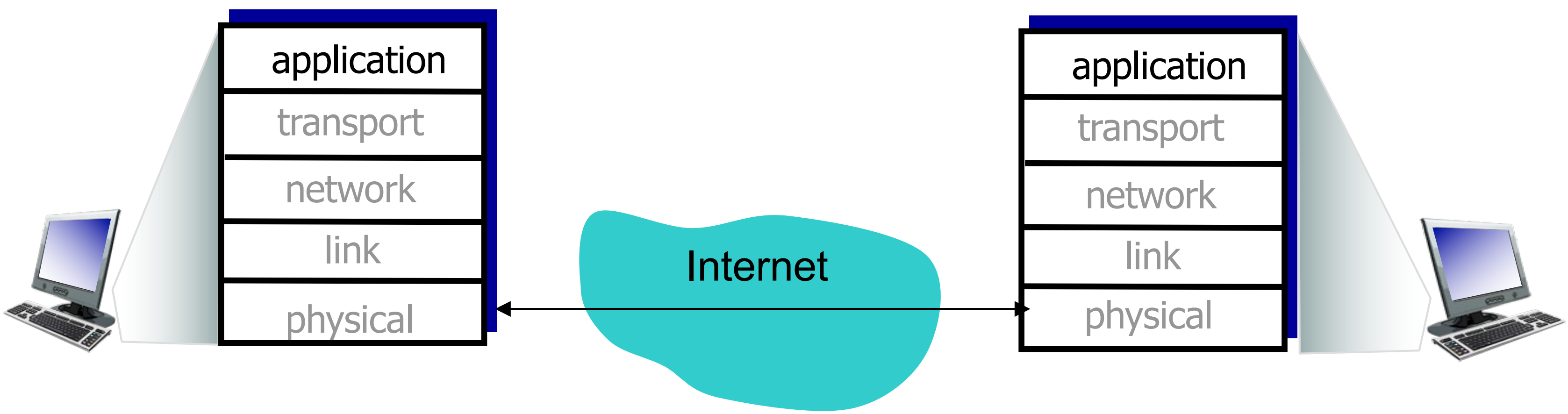


Outline

- Principles of network applications
- Web and HTTP
- DNS
- Socket programming

Application layer

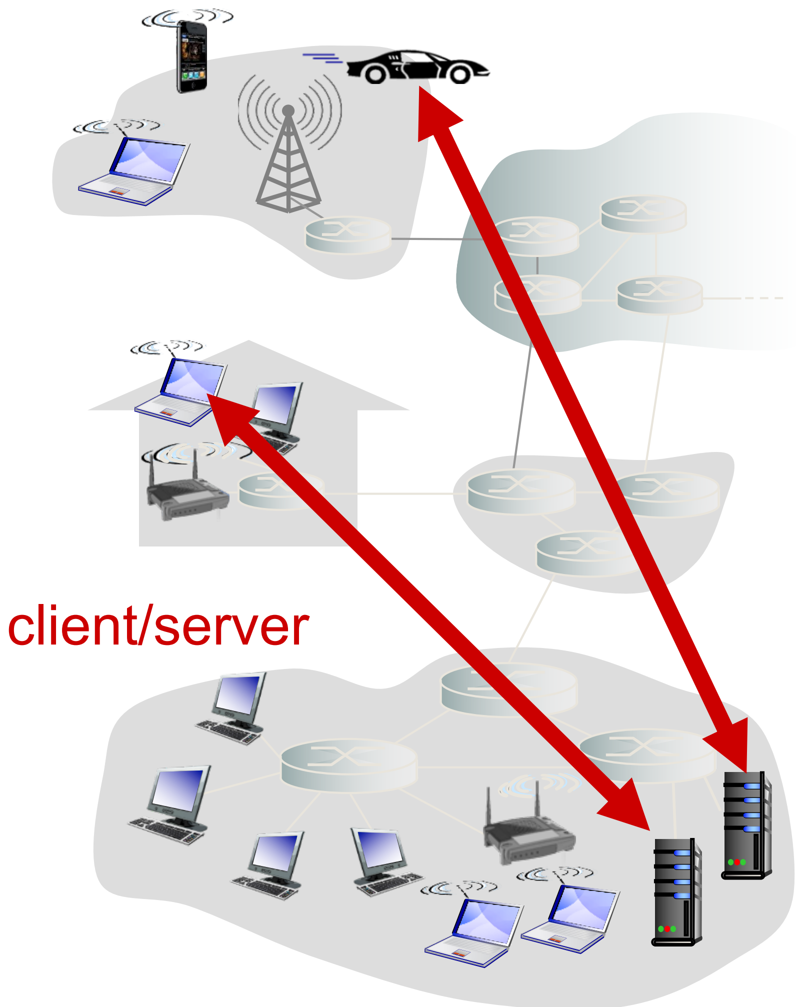
- Services: supporting network applications
  - Protocols: FTP, SMTP, HTTP, SIP, RTP, RTSP
  - Applications: file transfer, email, web browser, skype, multimedia streaming …
- *Use* socket interfaces from the transport layer (TCP & UDP)



Application layer architectures

- possible structure of applications:
  - client-server
  - peer-to-peer (P2P)
  - hybrid

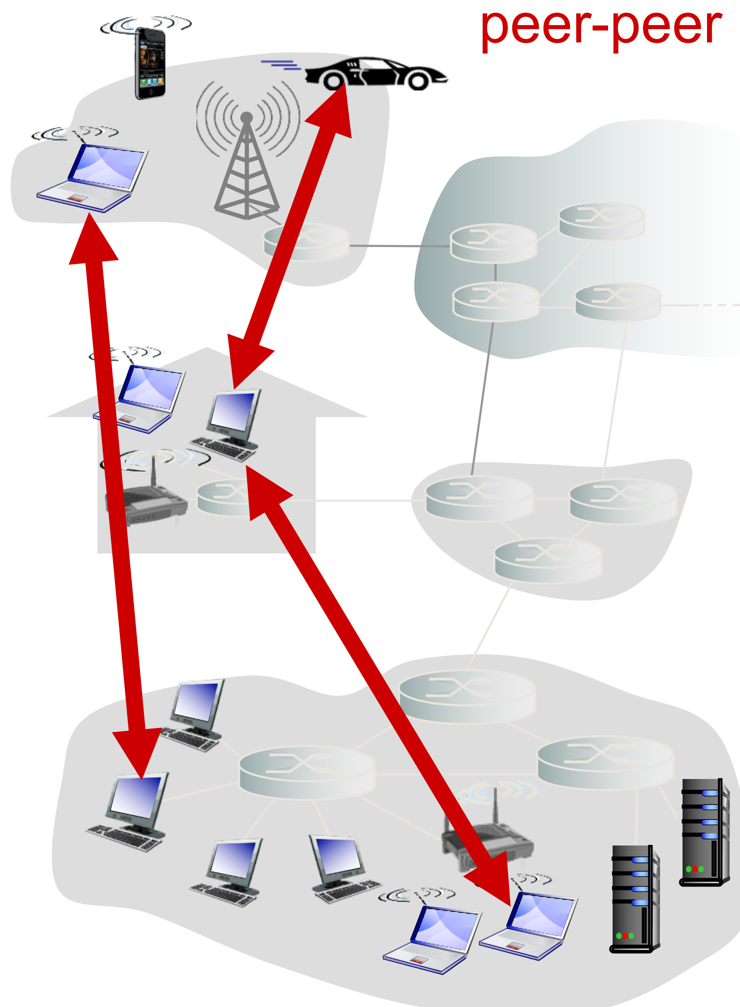
Client-server architecture



- 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

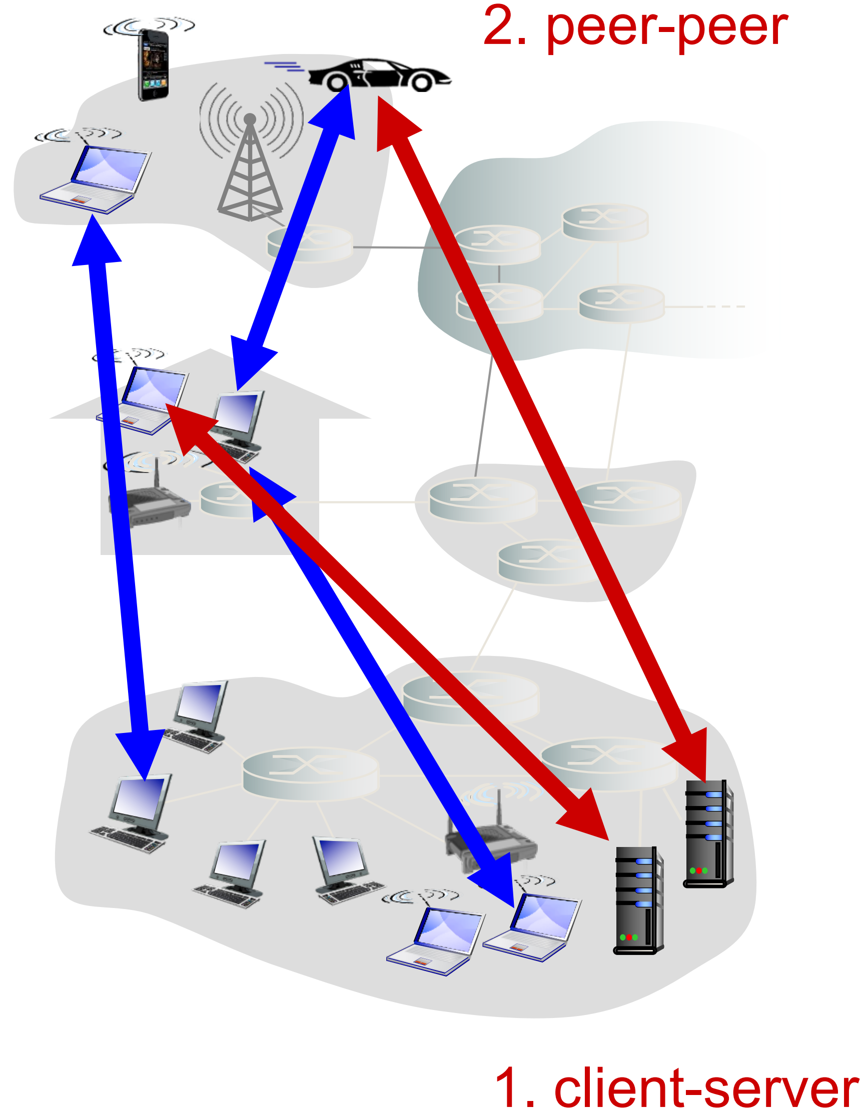
P2P architecture

- no always-on server
- arbitrary end systems directly communicate
- peers request service from other peers, provide service in return to other peers
  - self scalability – new peers bring new service capacity, as well as new service demands
- peers are intermittently connected and change IP addresses
  - complex management



Hybrid

- Servers provides account authentication and maintains information for clients
- Data communication is done directly between clients



Communicating Processes

process: program running within a host

- within same host, two processes communicate using inter-process communication (supported by OS)
- processes in different hosts communicate by exchanging messages

clients, servers

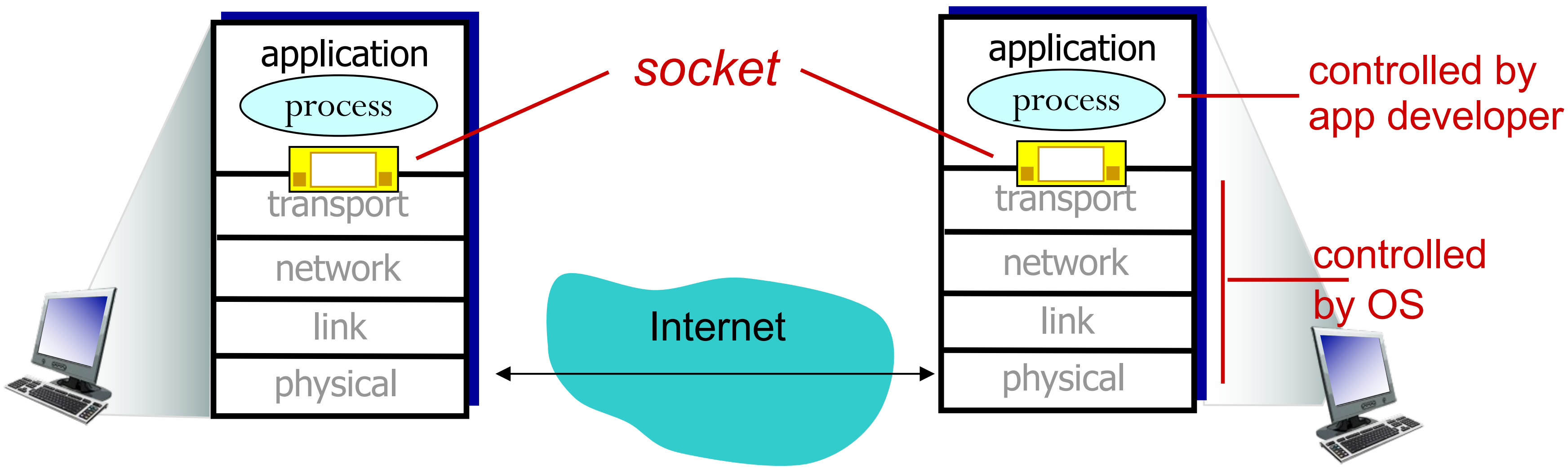
- client process: process that initiates communication
- server process: process that waits to be contacted

aside

applications with P2P architectures have both client processes & server processes

Sockets

- process sends/receives messages to/from its socket
- socket analogous to door
  - sending process shoves message out of a door
  - sending process relies on transport infrastructure on other side of door to deliver message to the socket at the receiving process



Addressing processes

- to receive messages, process must have identifier
- host device has unique IP address
- identifier includes both IP address and port numbers associated with process on host.
- example port numbers:
  - HTTP server: 80
  - SMTP mail server: 25
- to send HTTP request to www.cas.mcmaster.ca web server:
  - IP address: 130.113.68.10
  - port number: 80

Requirements: common apps

application	data loss	throughput	time sensitive
file transfer	no loss	elastic	no
e-mail	no loss	elastic	no
Web documents	no loss	elastic	no
real-time audio/video	loss-tolerant	audio: 5kbps-1Mbps video:10kbps-5Mbps	yes, 100’s msec
stored audio/video	loss-tolerant	same as above	Sensitive to jitter
interactive games	loss-tolerant	few kbps up	yes, 100’s msec
text messaging	no loss	elastic	yes and no

Internet transport protocols services

TCP service:

- reliable transport between sending and receiving process
- flow control: sender won't overwhelm receiver
- congestion control: throttle sender when network overloaded
- does not provide: timing, minimum throughput guarantee, security
- connection-oriented: setup required between client and server processes

UDP service:

- unreliable data transfer between sending and receiving process
- does not provide: reliability, flow control, congestion control, timing, throughput guarantee, security, connection setup

Internet apps: application, transport protocols

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 (& SCTP)
Internet telephony	SIP, RTP, proprietary (e.g., Skype), WebRTC	TCP or UDP, (& SCTP)

A conceptual design of a video conferencing system

Outline

- Principles of network applications
- Web and HTTP
- DNS
- Socket programming

Web and HTTP

First, a review…

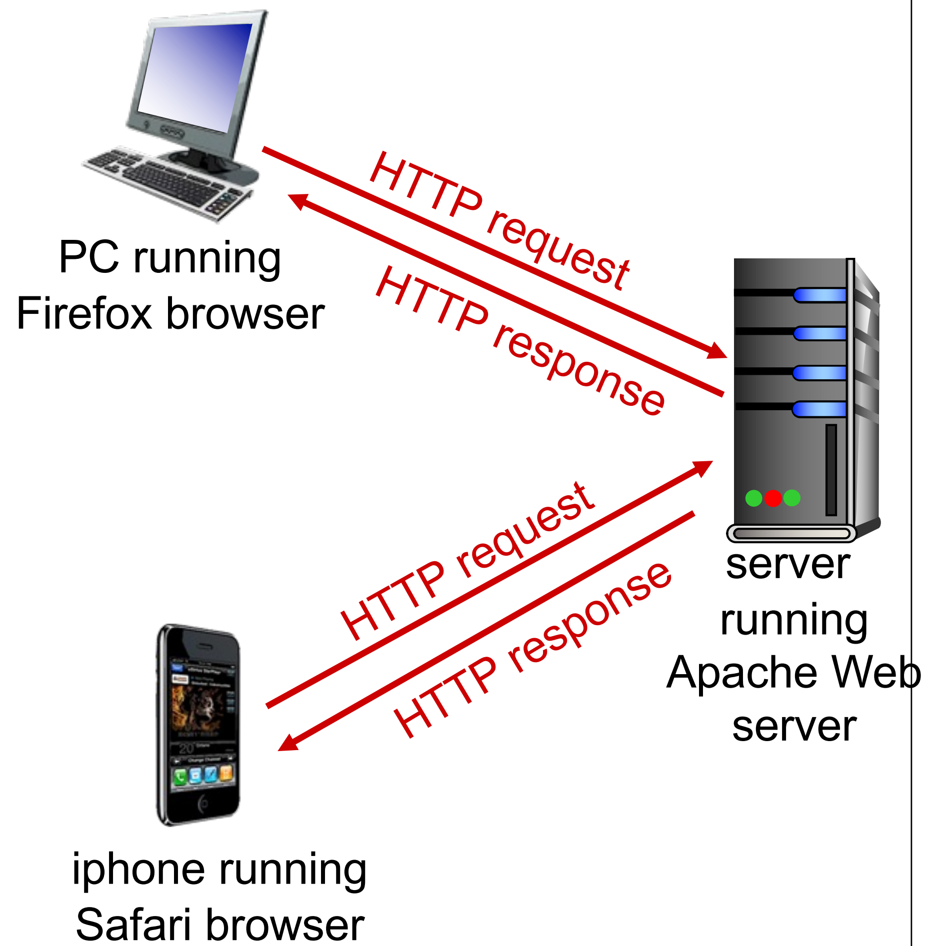
- web page consists of objects
- object can be HTML file, JPEG image, Java applet, audio file,…
- web page consists of base HTML-file which includes several referenced objects
- each object is addressable by a uniform resource identifier (URI), e.g.,

$$\underbrace{\texttt{www.someschool.edu}}_{\text{host name}}\texttt{/}\underbrace{\texttt{someDept/pic.gif}}_{\text{path name}}$$

HTTP overview

HTTP: hypertext transfer protocol

- Web's application layer protocol
- Web application follows 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 Evolution

HTTP/1.0	HTTP/1.1	HTTP/2	HTTP/3
1996	1997, 1999, 2014	2009 SPDY by Google 2015 RFC	2020 draft 2022 published in RFC 9114
Non-persistent connections	Persistent connections HTTP pipelining	Binary framing Data stream multiplexing	QUIC + UDP

Brief history of HTTP: https://hpbn.co/brief-history-of-http/

HTTP overview (continued)

uses TCP (prior to HTTP/3):

- 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"

- Responses do not require knowledge of past client requests

aside

protocols that maintain "state" are complex!

- past history (state) must be maintained
- if server/client crashes, their views of "state" may be inconsistent, must be reconciled

HTTP connections

Non-persistent HTTP

- at most one object sent over TCP connection
  - connection then closed
- HTTP/1.0 uses non-persistent HTTP
- Typically multiple TCP connections

Persistent HTTP

- multiple objects can be sent over a single TCP connection between client, server
- HTTP/1.1 uses persistent connections in default mode

Non-persistent HTTP Example

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

1RTT

time

Non-persistent HTTP Example (cont.)

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

time

6. Steps 1-5 repeated for each of 10 jpeg objects (in parallel)

Persistent HTTP Example

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

Persistent HTTP Example (cont.)

time

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

6. Steps 2-5 repeated for each of 10 jpeg objects (sequentially)

7. HTTP server closes TCP connection after some time or when the client closes the connection

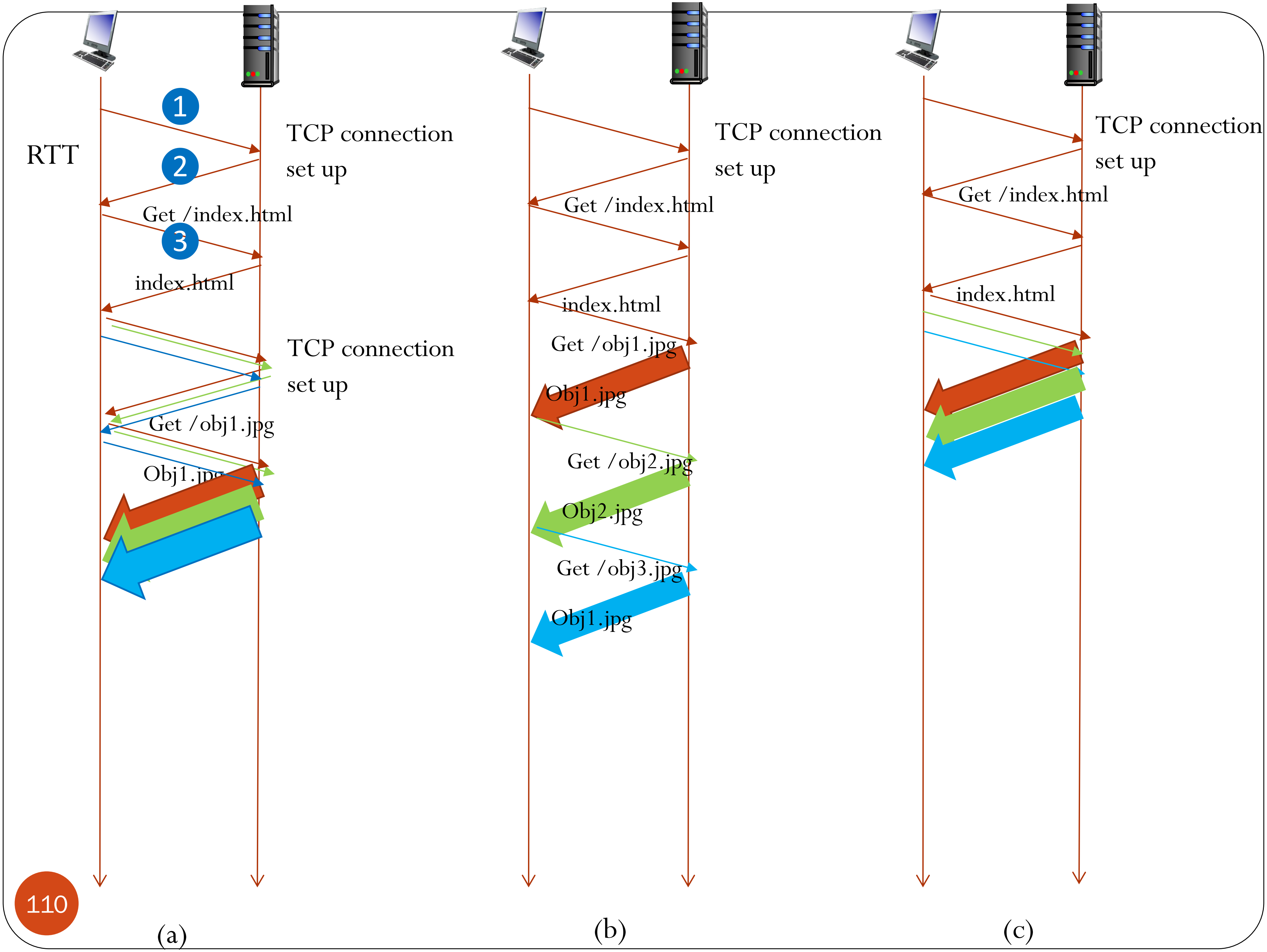
Persistent HTTP

Persistent **without** pipelining:

- client issues new request only when previous response has been received
- one round trip time (RTT) for each referenced object

Persistent **with** pipelining:

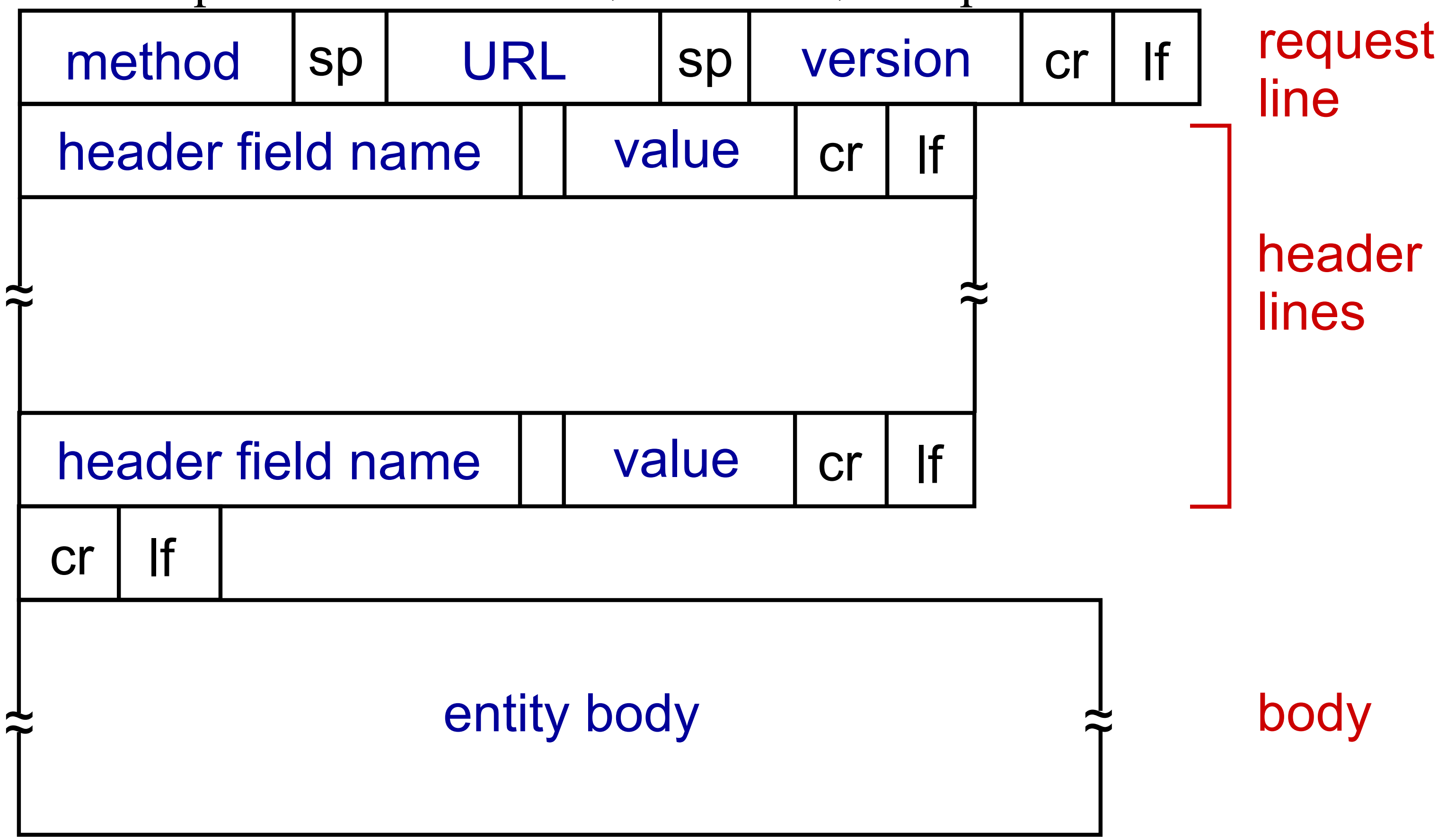
- default in HTTP/1.1
- client sends requests as soon as it encounters a referenced object
- as little as one RTT for all the referenced objects



1
2
3
RTT
TCP connection
set up
Get /index.html
index.html
TCP connection
set up
Get /obj1.jpg
Obj1.jpg
TCP connection
set up
Get /index.html
index.html
Get /obj1.jpg
Obj1.jpg
Get /obj2.jpg
Obj2.jpg
Get /obj3.jpg
Obj1.jpg
TCP connection
set up
Get /index.html
index.html
(a)
(b)
(c)

HTTP request message

- HTTP request message:
  - ASCII (human-readable format)
  - Request line: method, resource, and protocol version



HTTP request message

- HTTP request message:
  - ASCII (human-readable format)
  - Request line: method, resource, and protocol version

request line (GET, POST, HEAD commands)

carriage return character

line-feed character

```
GET /index.php HTTP/1.1\r\n
Host: www.cas.mcmaster.ca\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
```

header lines

carriage return, line feed at start of line indicates end of header lines

Time in seconds to keep the connection open when idle

Method types

HTTP/1.0:

- GET
  - Request an object specified by the URL
- POST
  - Request that the server accept the entity enclosed in the request
- HEAD
  - asks server to leave requested object out of response (only meta info)

HTTP/1.1:

- GET, POST, HEAD
- PUT
  - uploads file/resource in entity body to path specified in URL field
- DELETE
  - deletes resource specified in the URL field

Security Vulnerabilities

- URI in clear-text (GET)

http://www.things.com/orders.asp?custID=101&name=Trudeau&part=555A&qy=20&price=10

- Attackers can exploit buffer overflow in buggy server software by getting/posting a very long URI

http://www.things.com/orders.asp?phone_no=314159265358979323846264338327950288

HTTP response message

- HTTP response message:
  - Status line: protocol version, status code, status phrase

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 ...
```

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

Trying out HTTP (client side) for yourself

1. Telnet to your favorite Web server:

telnet www.google.com 80

opens TCP connection to port 80 (default HTTP server port) at cis.poly.edu. anything typed in sent to port 80 at cis.poly.edu

2. type in a GET HTTP request:

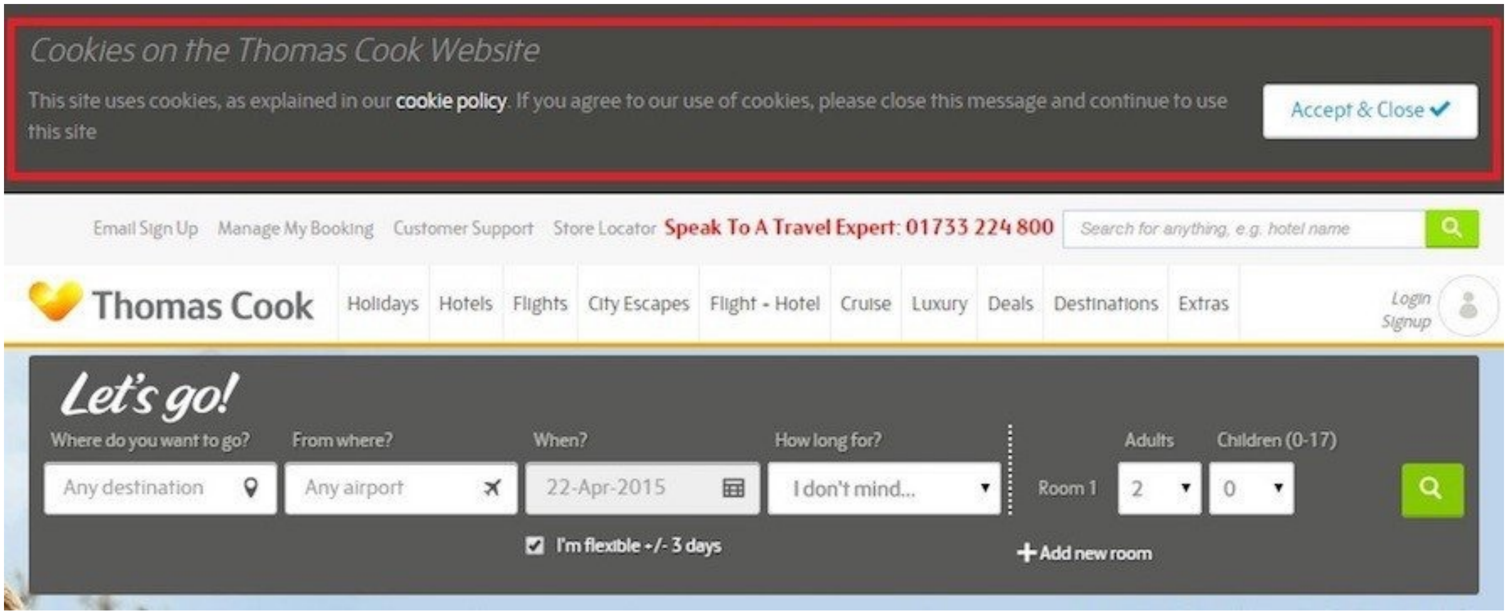
GET /index.html HTTP/1.1
Host: www.google.com

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!

(or use Wireshark to look at captured HTTP request/response)

User-server state: cookies



EU legislation on cookies

EUROPA websites must follow the Commission's guidelines on privacy and data protection and inform users that cookies are not being used to gather information unnecessarily.

The ePrivacy directive – more specifically Article 5(3) – requires prior informed consent for storage or for access to information stored on a user's terminal equipment. In other words, you must ask users if they agree to most cookies and similar technologies (e.g. web beacons, Flash cookies, etc.) before the site starts to use them.

For consent to be valid, it must be informed, specific, freely given and must constitute a real indication of the individual's wishes.

http://ec.europa.eu/ipg/basics/legal/cookies/index_en.htm

User-server state: cookies

HTTP itself is stateless

But often desirable to identify users

- Restriction
- Customizing content

Cookies: http messages carry state

There are four components:

1) cookie header line of HTTP response message

2) cookie header line in next HTTP request message

3) cookie file kept on user's host, managed by user's browser

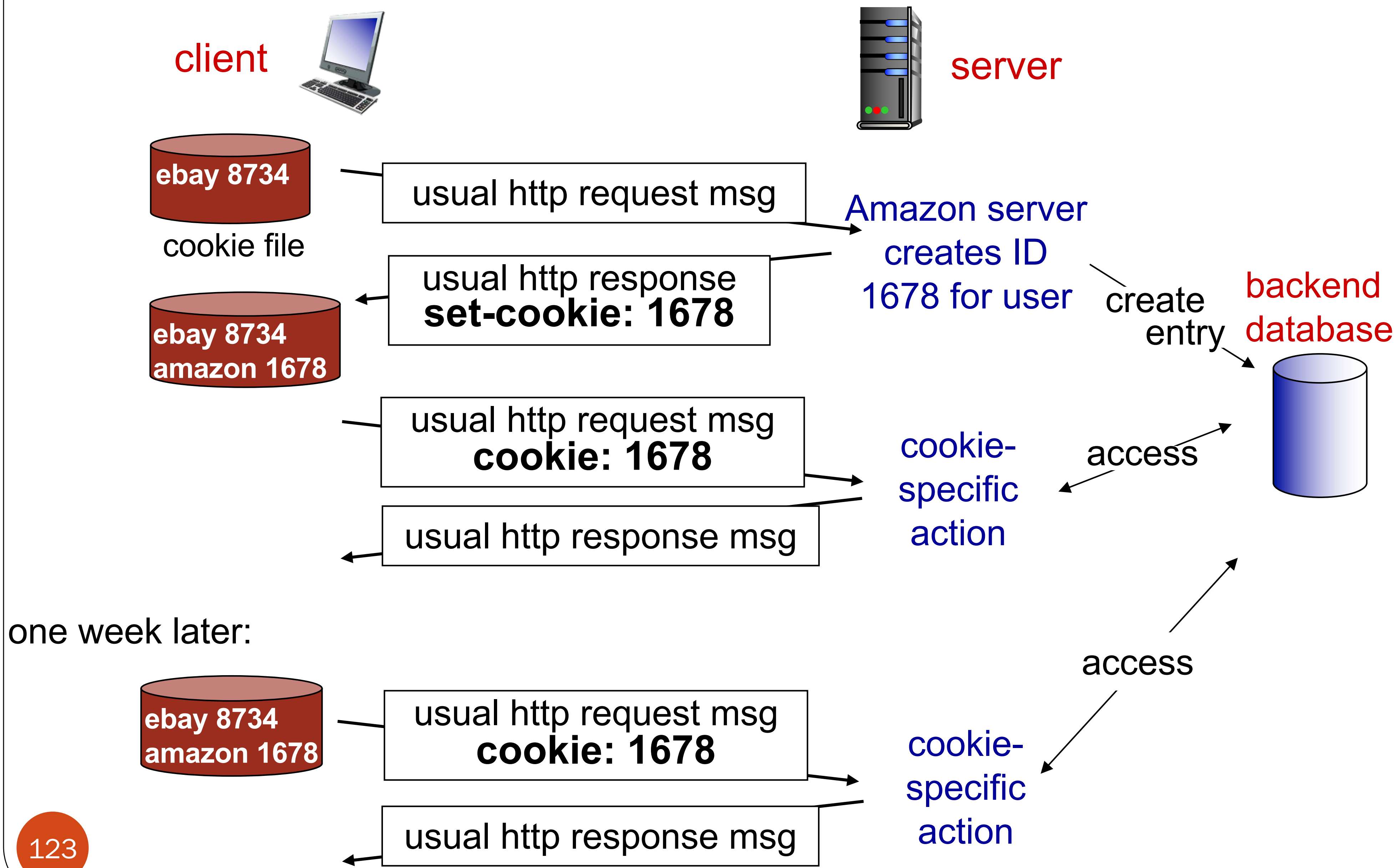
4) back-end database at Web site

User-server state: cookies

Example:

- Susan always access Internet from her PC
- Visited eBay before and visits Amazon for first time
- when initial HTTP requests arrives at site, site creates:
  - unique ID
  - entry in backend database for ID

Cookies: keeping “state” (cont.)



Cookies (continued)

what cookies can be used for:

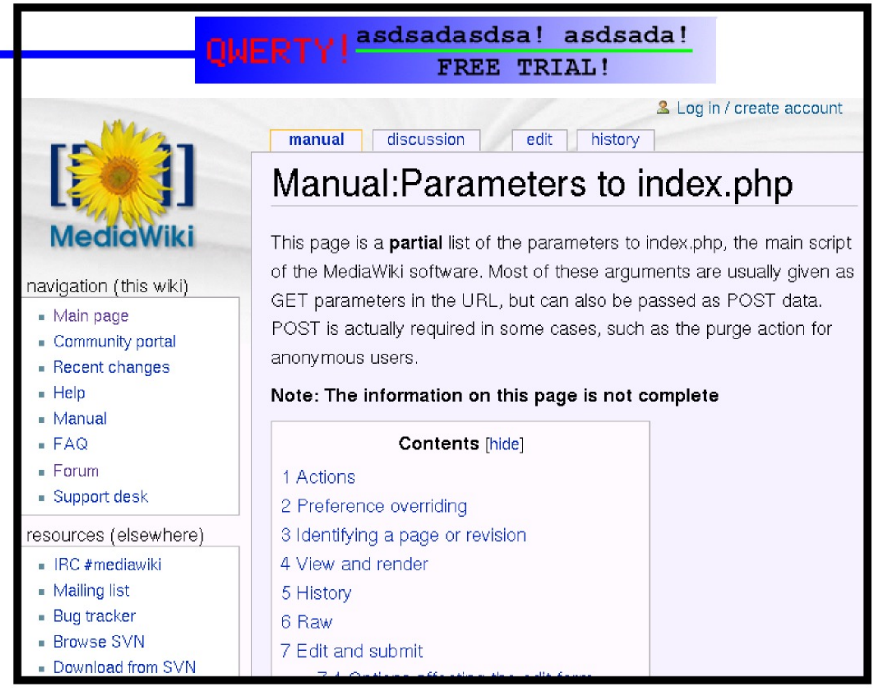
- authorization
- shopping carts
- recommendations
- user session state

aside

cookies and privacy:

- cookies permit sites to learn a lot about you
- Third-party cookies can track users across multiple sites

cia.gov



mediawiki.org

lotofbanners.com

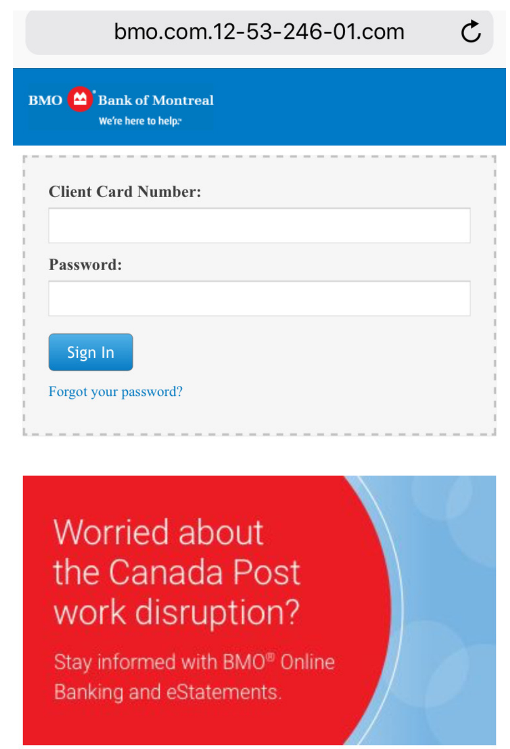
Outline

- Principles of network applications
- Web and HTTP
- DNS -- **domain name system**
- Socket programming

DNS: Domain Name System

>> traceroute www.mcmaster.ca
traceroute to pinwps02.uts.mcmaster.ca (130.113.64.30), 64 hops max, 52 byte packets

- People use hostname
- Hosts, routers use IP address for addressing datagrams
- How to map between IP address and hostname?



DNS Service

- Hostname to IP address translation
- Host/server aliasing
  - Canonical vs alias names

>> traceroute www.mcmaster.ca
traceroute to pinwps02.uts.mcmaster.ca (130.113.64.30), 64 hops max, 52 byte packets

- Load distribution
  - Replicated Web servers: set of IP addresses for one canonical name

traceroute google.com
traceroute: Warning: google.com has multiple addresses; using 74.125.226.105

traceroute google.com
traceroute: Warning: google.com has multiple addresses; using 74.125.226.104

Note: the results subject to changes over time.

DNS: Domain Name System

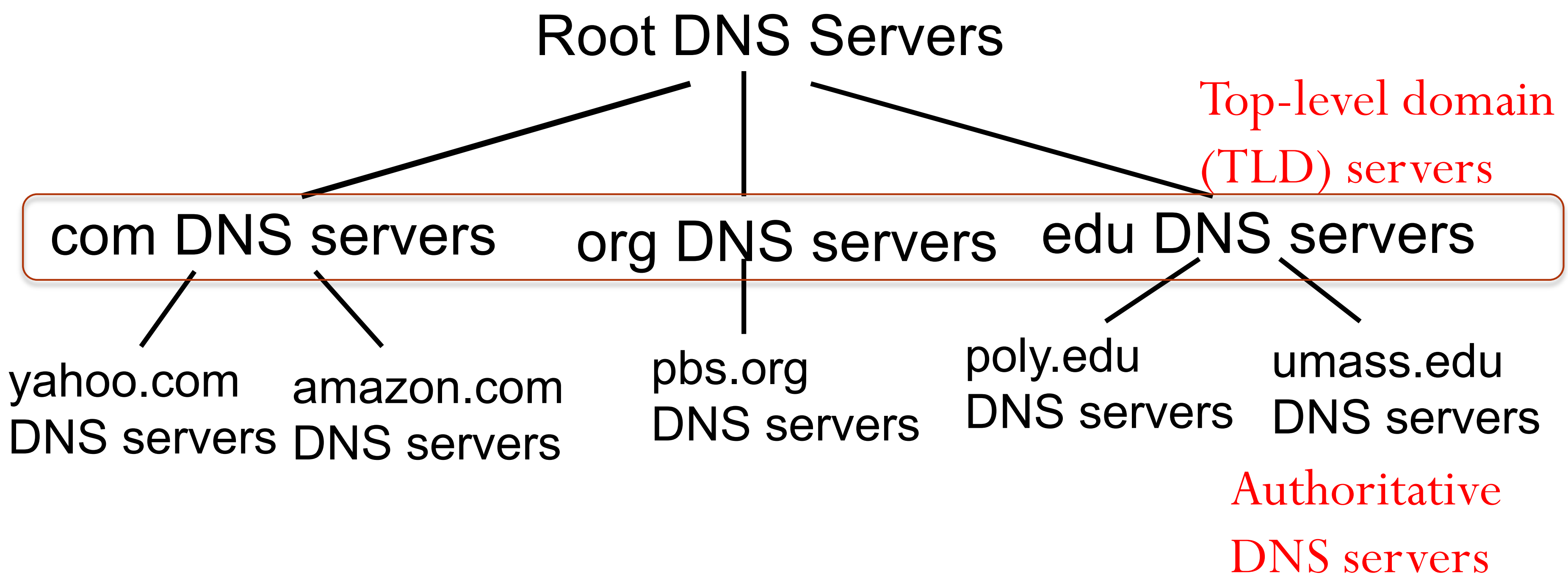
- distributed database implemented in hierarchy of many name servers
- application-layer protocol
  - address/name translation

Why not centralize DNS?

- single point of failure
- traffic volume
- Latency to access distant centralized database
- Maintenance

doesn’t *scale!*

Distributed, Hierarchical Database



Local DNS servers

Local Name Server

- Does not strictly belong to hierarchy
- Each ISP (residential ISP, company, university) has one.
  - Also called “default name server”
- When a host makes a DNS query, query is sent to its local DNS server
  - Acts as a proxy, forwards query into hierarchy.

DNS: Root name servers

- 13 logical servers in total (over 1700 physical servers)
- contacted by local name server that can not resolve name
- root name server:
  - contacts authoritative name server for the appropriate TLD domains if name mapping not known
  - gets mapping
  - returns mapping to local name server

TLD and Authoritative Servers

- Top-level domain (TLD) servers:
  - responsible for com, org, net, edu, etc, and all top-level country domains uk, fr, ca, jp.
    - Eg. .edu for edu TLD
  - Store records for authoritative DNS servers of the next level
- Authoritative DNS servers:
  - organization's DNS servers, providing authoritative hostname to IP mappings for organization's servers (e.g., Web and mail).
  - Can be maintained by organization or service provider

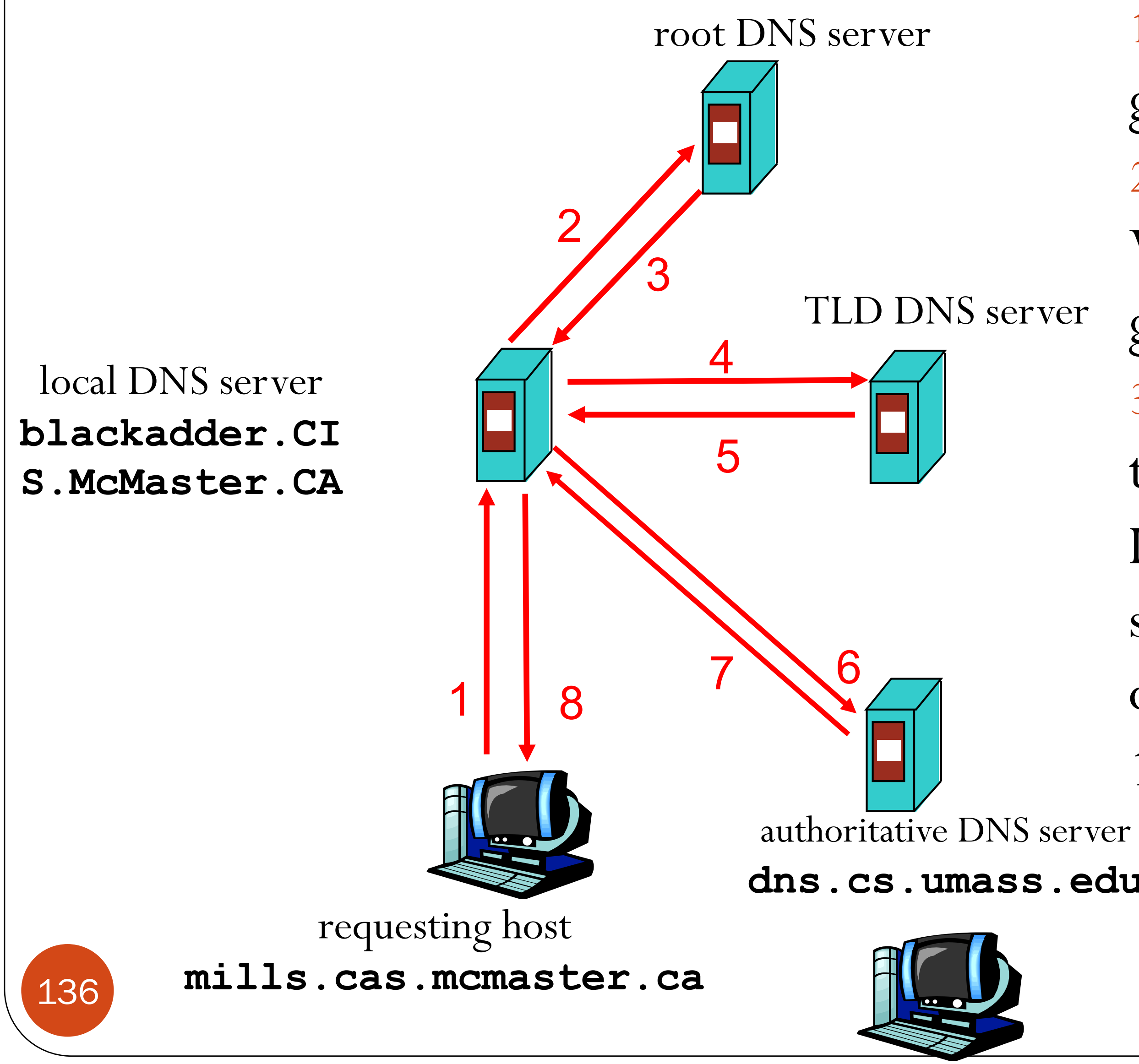
DNS: caching and updating records

- once (any) name server learns mapping, it *caches* mapping
  - cache entries timeout (disappear) after some time
  - TLD servers typically cached in local name servers
    - Thus root name servers not often visited
- update/notify mechanisms under design by IETF
  - RFC 2136
  - http://www.ietf.org/html.charters/dnsind-charter.html

How is DNS query resolved

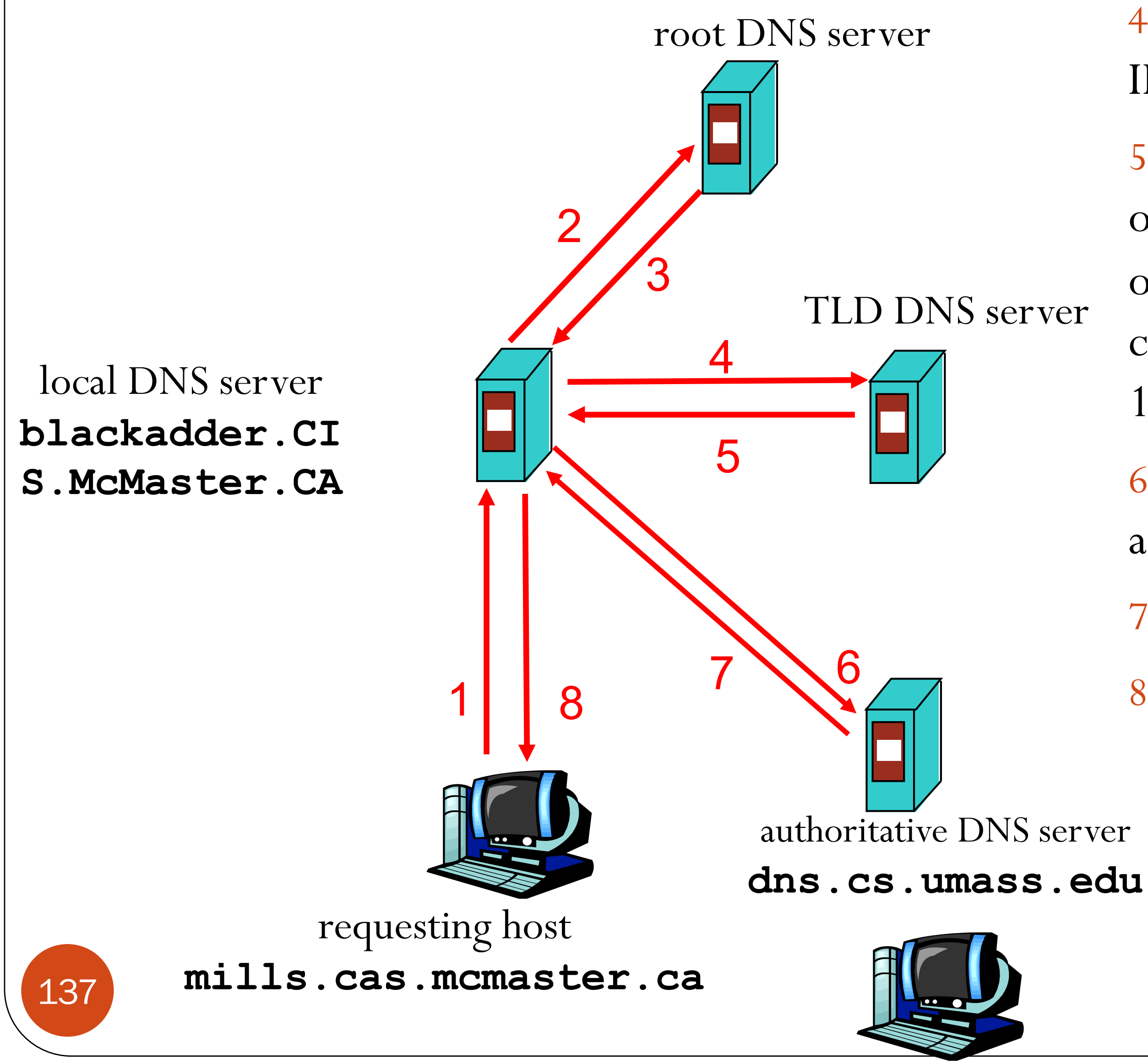
- Host at mills.cas.mcmaster.ca wants IP address for gaia.cs.umass.edu
- Assume results are not cached anywhere but at the authoritative DNS servers

How is DNS query resolved



1. What is the IP address of gaia.cs.umass.edu?
2. (recursive)Hi, root DNS, What is the IP address of gaia.cs.umass.edu?
3. (I don't know), but here is the list of domain names and IP addresses of .edu TLD server you can contact (e.g., d.edu-servers.net or 192.31.80.30 (iterative)

How is DNS query resolved



4. Hi, d.edu-servers.net, what is the IP address of gaia.cs.umass.edu?
5. (I don't know), but here is the list of domain names and IP addresses of authoritative DNS server you can contact (e.g., ns1.umass.edu or 128.119.10.27)
6. Hi, ns1.umass.edu, what is the IP address of gaia.cs.umass.edu?
7. The answer is 128.119.245.12
8. The answer is 128.119.245.12

DNS Messages Example

dig www.mcmaster.ca

```
; <<>> DiG 9.10.6 <<>> www.mcmaster.ca
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 58553
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 3, ADDITIONAL: 7

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

;; ANSWER SECTION:
www.mcmaster.ca. 3600 IN CNAME pinwps02.uts.mcmaster.ca.
pinwps02.uts.mcmaster.ca. 3600 IN A 130.113.64.30

;; AUTHORITY SECTION:
uts.mcmaster.ca. 86400 IN NS pindns01.mcmaster.ca.
uts.mcmaster.ca. 86400 IN NS pindns03.mcmaster.ca.
uts.mcmaster.ca. 86400 IN NS pindns05.mcmaster.ca.

….
```

```
94 8.705925( 192.168.0.12     24.226.10.193    DNS      75 Standard query 0xcfcd  A www.mcmaster.ca
95 8.722821( 24.226.10.193    192.168.0.12     DNS     195 Standard query response 0xcfcd  CNAME wwwma
```

```
▷ User Datagram Protocol, Src Port: 54242 (54242), Dst Port: domain (53)
▽ Domain Name System (query)
    [Response In: 95]
    Transaction ID: 0xcfcd
  ▷ Flags: 0x0100 Standard query
    Questions: 1
    Answer RRs: 0
    Authority RRs: 0
    Additional RRs: 0
  ▽ Queries
    ▽ www.mcmaster.ca: type A, class IN
        Name: www.mcmaster.ca
        Type: A (Host address)
        Class: IN (0x0001)
```

What transport protocol do DNS query & response use?

```
▶ User Datagram Protocol, Src Port: domain (53), Dst Port: 50580 (50580)
▼ Domain Name System (response)
    [Request In: 1538]
    [Time: 0.004561000 seconds]
    Transaction ID: 0xbb75
  ▶ Flags: 0x8580 Standard query response, No error
    Questions: 1
    Answer RRs: 2
    Authority RRs: 3
    Additional RRs: 7
  ▶ Queries
  ▼ Answers
    ▶ www.mcmaster.ca: type CNAME, class IN, cname pinwps02.uts.mcmaster.ca
    ▶ pinwps02.uts.mcmaster.ca: type A, class IN, addr 130.113.64.30
  ▼ Authoritative nameservers
    ▶ uts.mcmaster.ca: type NS, class IN, ns pindns03.mcmaster.ca
    ▶ uts.mcmaster.ca: type NS, class IN, ns pindns05.mcmaster.ca
    ▶ uts.mcmaster.ca: type NS, class IN, ns pindns01.mcmaster.ca
  ▶ Additional records
```

DNS records

- DNS info stored as resource records (RRs)
  - RR format: (name, TTL, class, type, value)

    TTL (Time To Live) used by authoritative DNS for indicating the validity of RR to caching DNS

- Type=A (Address)
  - name = hostname
  - value = IP address
  - e.g., pinwps02.uts.mcmaster.ca. 3600 IN A 130.113.64.30
- Type=NS (Name Server)
  - name = domain
  - value = name of dns server for domain
  - e.g., uts.mcmaster.ca. 86400 IN NS pindns03.mcmaster.ca.

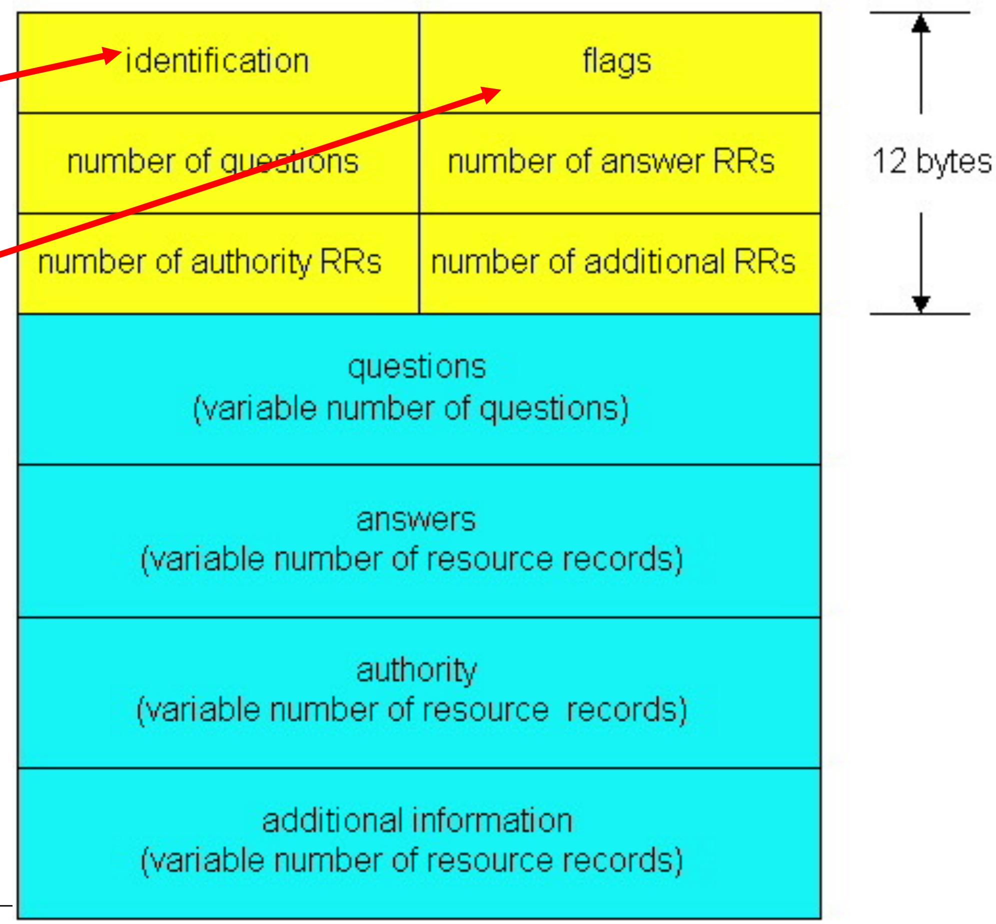
DNS records

- Type=CNAME(Canonical NAME)
  - name = hostname
  - value = canonical name
  - Example: www.mcmaster.ca. 3600 IN CNAME pinwps02.uts.mcmaster.ca
- Type=MX (Mail eXchanger)
  - name = domain in email address
  - value = canonical name of mail server

DNS protocol, messages

DNS protocol : *query* and *reply* messages, both with same *message format*

- Application layer protocol uses **UDP Port 53**
  - Also uses TCP too, but not always implemented
- **msg header**
  - identification: 16 bit # for query, reply to query uses same #
- flags:
  - query or reply
  - recursion desired
  - recursion available
  - reply is authoritative



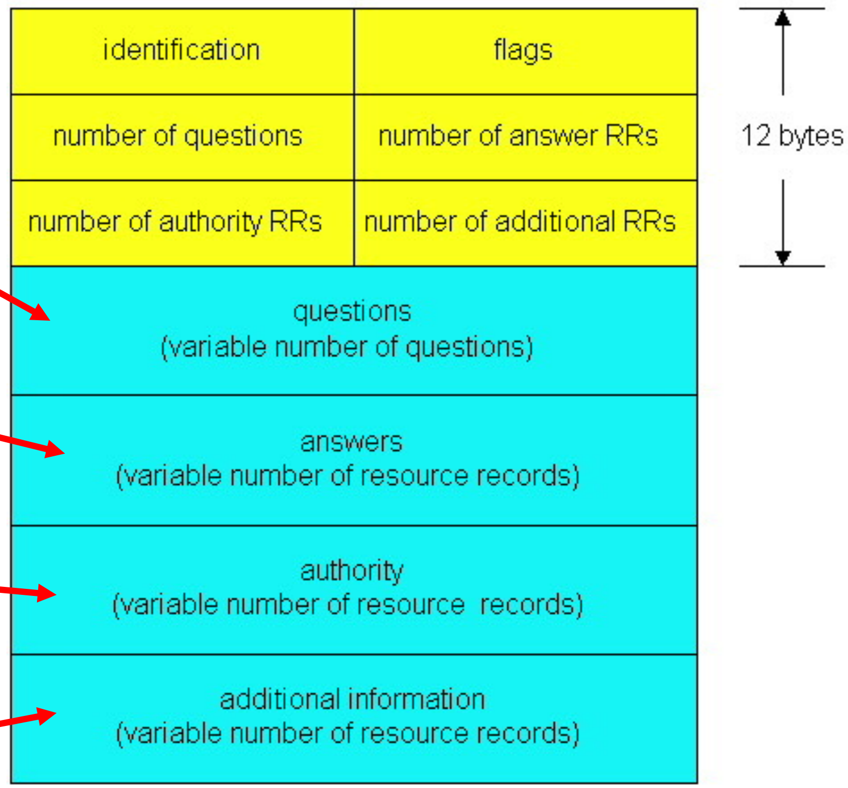
DNS protocol, messages

Name, type fields for a query

RRs in response to query

records for other authoritative servers

additional "helpful" info that may be used



Attacking DNS

DDoS attacks

- Bombard root servers with traffic (October 21, 2002)
  - one hr, Oct 21, 2002, ICMP Ping attacks on 13 Root servers
  - 24 hrs, Feb. 6th, 2007
  - Nov. 30th, 2015/Dec. 1st, 2015, 5 millions queries
- Remedies
  - Traffic Filtering
  - Distributing requests to other root servers
  - Local DNS servers cache IPs of TLD servers, allowing root server bypass

Attacking DNS

DDoS attacks

- Bombard TLD servers
  - Potentially more dangerous

Aug. 27, 2013 Servers running China's ".cn" top level domain (TLD) came under attack Sunday starting at about 2 a.m. Eastern time. The China Internet Network Information Center, which runs the TLD servers, confirmed the attack and apologized to affected users.

Attacking DNS

Redirect attacks

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

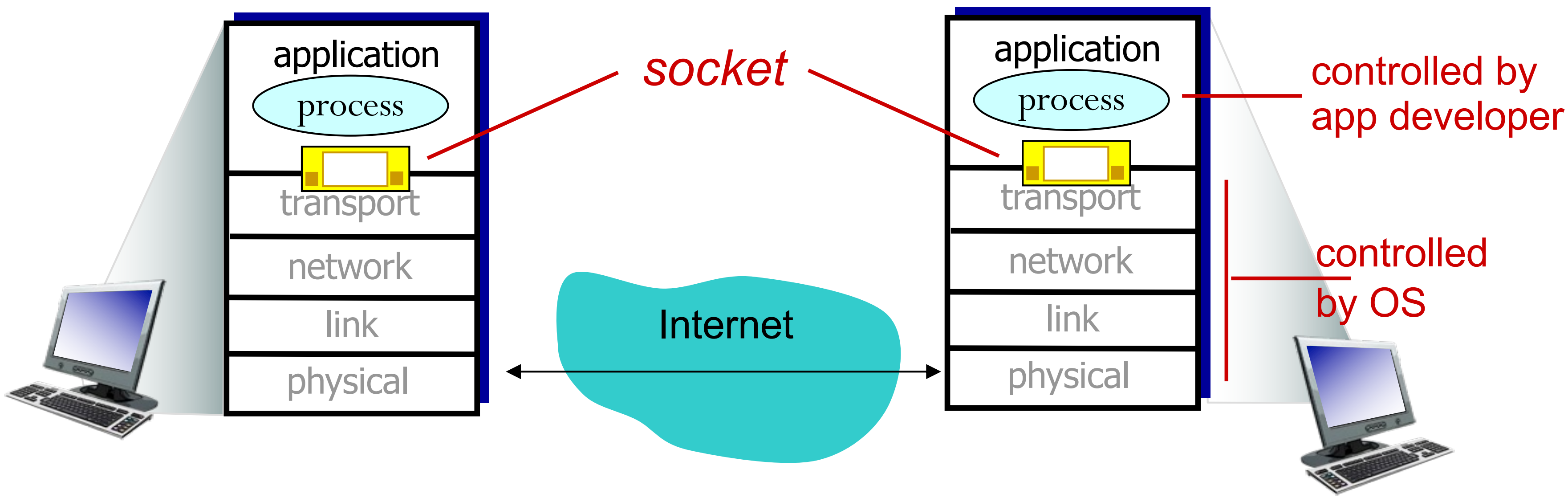
Solution: use cryptographically signed response, e.g., DNSSEC and DNSCurve

Outline

- Principles of network applications
- Web and HTTP
- DNS -- domain name system
- **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, connection-less
  - TCP: reliable, byte stream-oriented, connection-oriented

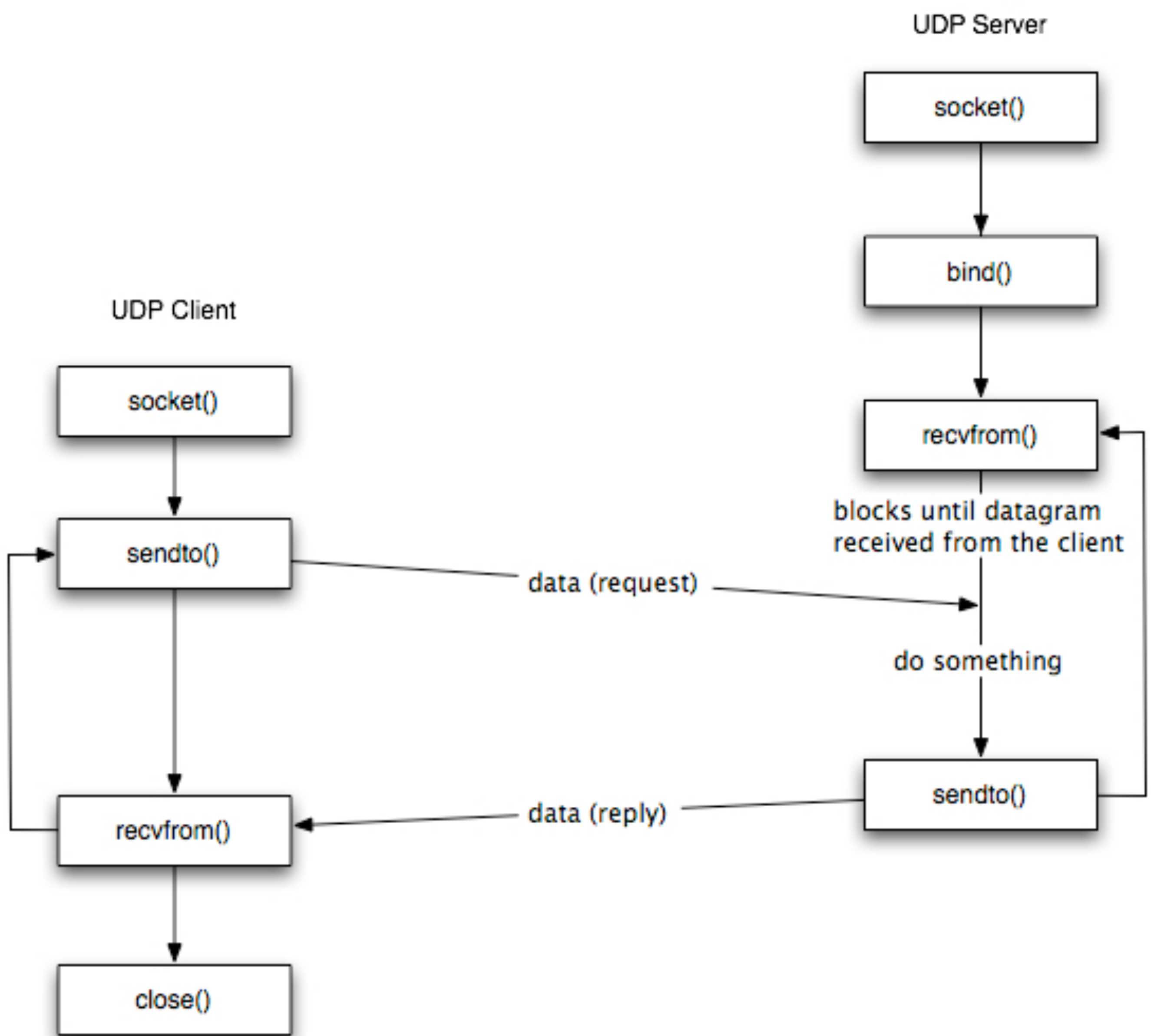
Application Example:

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 upper case.
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

- UDP: no "connection" between client & server
  - no handshaking before sending data
  - Sender program explicitly attaches IP destination address and port # to each packet
  - rcvr program 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



Example app: UDP client

Python UDPClient

include Python's socket library

create UDP socket

get user keyboard input

Attach server name, port to message; send into socket

read reply (max buffersize) characters from socket into string

print out received string and close socket

```
from socket import *
host = '127.0.0.1'
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_DGRAM)
message = input('Input lowercase sentence:')
clientSocket.sendto(bytes(message, 'utf-8'),(host,
serverPort))
modifiedMessage, serverAddress =
clientSocket.recvfrom(2048)
print(modifiedMessage.decode("utf-8"))
clientSocket.close()
```

Example app: UDP server

Python UDPServer

create UDP socket

bind socket to local port number 12000

loop forever

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

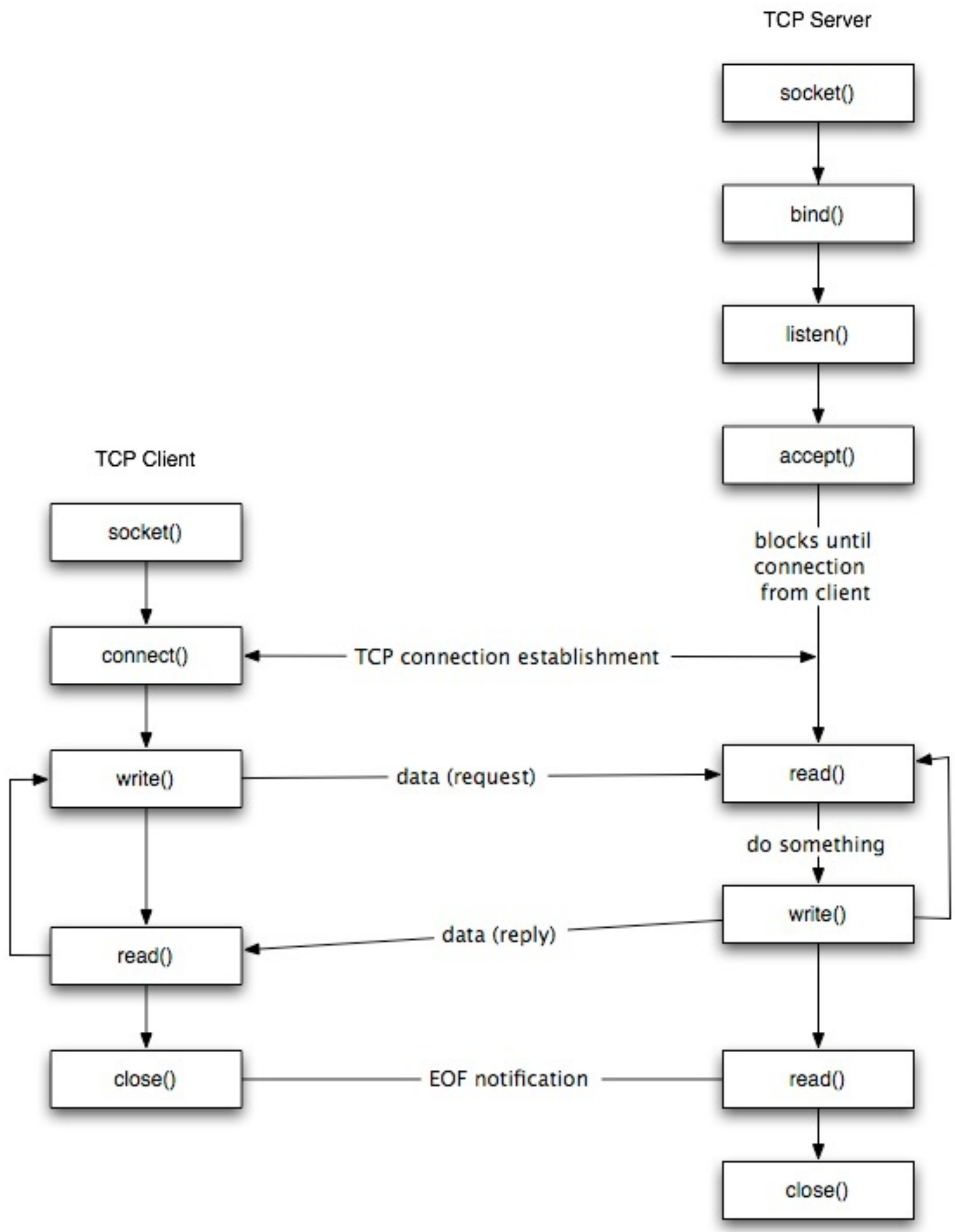
send upper case string back to this client

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET, SOCK_DGRAM)
serverSocket.bind(("", serverPort))
print ('The server is ready to receive')
while 1:
    message, clientAddress = serverSocket.recvfrom(2048)
    modifiedMessage = (message.decode("utf-8")).upper()
    serverSocket.sendto(bytes(modifiedMessage,'utf-8'),
        clientAddress)
```

Socket programming with TCP

- client must connect to server
  - server process must first be running
  - server must have created socket (door) that welcomes client's contact
- client connects to server by:
  - Creating TCP socket, specifying IP address, port number of server process
  - Client TCP establishes connection to server TCP
- when contacted by client, server TCP creates a 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

Client/server socket interaction: TCP



Example app: TCP client

Python TCPClient

create TCP socket for server, remote port 12000

No need to attach server name, port

```
from socket import *
serverName = '127.0.0.1'
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName,serverPort))
message = input('Input lowercase sentence:')
clientSocket.send(bytes(message, 'utf-8'))
modifiedMessage = clientSocket.recv(2048)
print('From Server: ' + modifiedMessage.decode("utf-8"))
clientSocket.close()
```

Example app: TCP server

Python TCPServer

create TCP welcoming socket

server begins listening for incoming TCP requests

loop forever

server waits on accept() for incoming requests, new socket created on return

read bytes from socket (but not address as in UDP)

close connection to this client (but *not* welcoming socket)

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET, SOCK_STREAM)
serverSocket.bind(("", serverPort))
serverSocket.listen(1)
print ('The server is ready to receive')
while 1:
    connectionSocket, addr = serverSocket.accept()
    sentence = connectionSocket.recv(1024).decode("utf-8")
    capitalizedSentence = sentence.upper()
    connectionSocket.send(bytes(capitalizedSentence,'utf-8'))
    connectionSocket.close()
```

Comparison of UDP & TCP Sockets

	TCP		UDP	
	Client	Server	Client	Server
Socket	SOCKET_STREAM	SOCKET_STREAM	SOCKET_DGRAM	SOCKET_DGRAM
Bind to a fixed port		x		x
Connection setup	x	x		
Send	send	send	sendto(data, (IP, port))	sendto(data, (IP, port))
Recv	recv	recv	recvfrom()	recvfrom()

Port Number

- 16 bit: 0 – 65535
- Port numbers are application layer addresses
- The port numbers in the range from 0 to 1023 are the *well-known ports* or *system ports*
  - SSH port 22
  - SMTP port 25
  - HTTP port 80
- Port numbers from1024 to 49151 can be used w/o superuser privileges (sometimes registered by a service, e.g. bittorrent 6888 – 6900)
- > 49151 private ports (linux uses the port range 32768 to 60999)

Chapter 2: summary

- application architectures
  - client-server
  - P2P
  - Hybrid
- application service requirements:
  - reliability, bandwidth, delay
- TCP, UDP, SSL
- HTTP
  - Non-persistent vs persistent
  - Messages
  - Method types
  - Cookies
- Web cashing
- DNS
  - Services
  - 4 records
  - Hierarchical
  - Iterative, recursive query
  - Dig
  - Attacking DNS
    - DDoS attacks
    - Redirect attacks
- Socket programming