Networks

Principles of Network Applications

Adopted from material in “Computer Networking: A Top Down Approach” by Kurose and Ross and slides developed by William Conner

Principles of Network Applications

Section 2.1

Review: APRANET Reference Model

- Used by TCP/IP protocol suite (Internet)
- Presentation and session layer rolled into application
- Used in this course

Application
Transport
Network
Link
Physical

Application Layer

- Top layer of protocol stack
- Example network apps
  - E-mail
  - Web
  - Instant Messaging
  - Remote login
  - P2P file sharing
  - Gaming
  - Video streaming (YouTube, Hulu, Netflix)
  - Voice-over-IP
  - Video Conferencing (Zoom)
  - Social networking (Twitter, Facebook)

Networked Application Development

- Only program end systems
- Communicate over network (often via socket API or RFC framework)
- No need to reimplement services provided by lower layers
  - Transport layer provides process-to-process communication (and sometimes reliability)
  - Network layer provides host-to-host communication

Client-Server Architecture

- Highly available server with (relatively) stable IP address and port
- Clients send requests and receive responses
- Servers send responses and receive requests
- For example, Browser and Web server

Peer-to-Peer Architecture

- Intermittent peers that simultaneously act as both clients and servers
- Often rely on stable servers for bootstrapping
- For example, BitTorrent, Skype (before 2016)

Application Protocols

- Request and response types
- Message format
- Message ordering
- Message semantics
- Rules and actions

Application Protocols

- Open standards
  - Usually defined in RFCs
  - For example, HTTP, SMTP, SIP
- Proprietary
  - Often defined by a company
  - For example, Skype

Addressing Processes

- Uniquely identified by IP address and port number
  - IP address identifies host machine
  - Hosts usually run multiple processes
  - Port identifies process on host

- Some well known ports:

HTTP	80
DNS	53
SMTP	25

Transport Layer Services

- Applications depend on transport layer to (de)multiplex signals to ports

- Process-to-process communication
  - Unreliable (UDP) vs. Reliable (TCP)
  - Flow control (TCP)
  - Congestion control (TCP)

- More next week

Application / Transport

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 (YouTube) RTP [RFC 1889]	TCP or UDP
Internet Telephony	SIP, RTP, proprietary (Skype)	TCP or UDP

Thank You!

Networks

The Web and HTTP

Adopted from material in “Computer Networking: A Top Down Approach” by Kurose and Ross and slides developed by William Conner

The Web and HTTP

Section 2.2

World Wide Web (WWW)

- Invented by Berners-Lee in 1989
- Mosaic provides GUI to WWW in 1993
- Initial “killer app” for the Internet in the 1990s
- Three major components:
  - Uniform Resource Locator (URL)
  - Hypertext Markup Language (HTML)
  - Hypertext Transfer Protocol (HTTP)

Uniform Resource Locator

- String that uniquely defines a web resource
- What are the URL components?
- Typical usage:

```
scheme:[//host][/path][?param=value]
```

- Less common:

```
scheme:[//[user[:password]@]host[:port]][/path][?query][#fragment]
```

Hypertext Markup Language

- Base web document file
- Can reference other web objects
  - Images, audio, video, bytecode for plug-ins, and other HTML objects
- Other common document languages
  - CSS
  - JavaScript

Hypertext Transfer Protocol

- Application layer protocol for the Web
- Client-server paradigm
  - Web browser (client)
  - Web server (server)
- Typically uses TCP for transport
  - QUIC is a notable exception

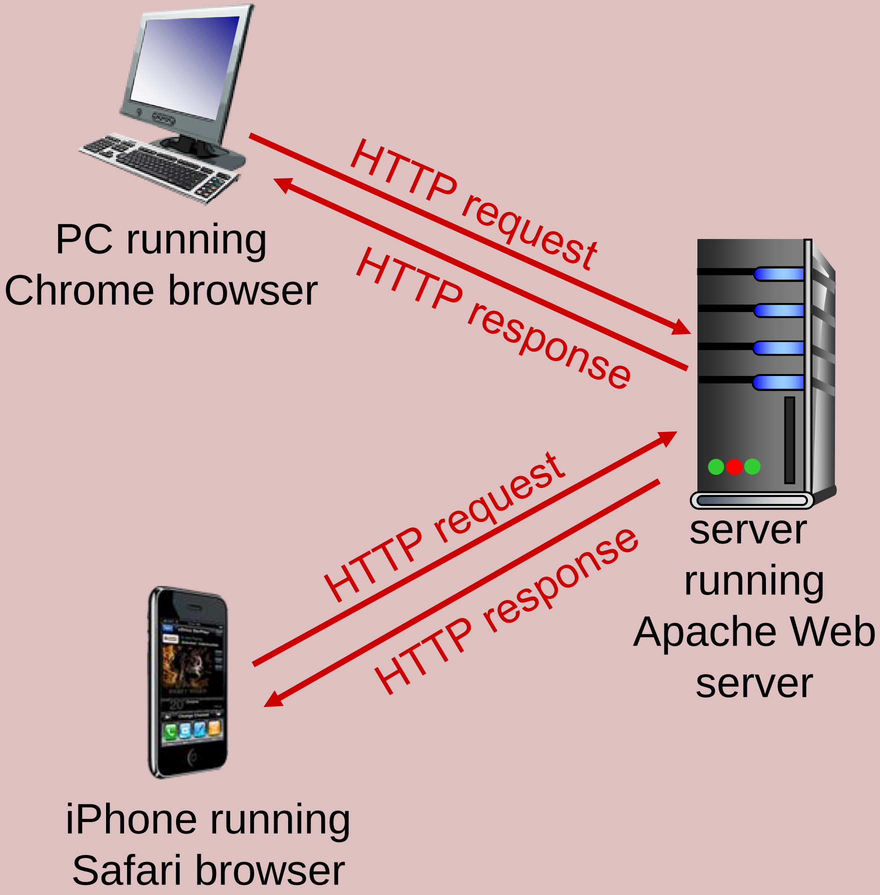
Quick UDP Internet Connection

- Avoids TCP connection establishment delays
- Implements reliability in the application layer on top of UDP (unreliable transport)
- Initially Google Chrome browser talking to Google Web servers
- Being standardized by the IETF (still in development)

HTTP Statelessness

- Protocol does not support maintaining information about past client requests
- Maintaining state adds complexity to protocol implementation
- Where is state typically maintained?

HTTP Request-Response Protocol



HTTP Generic Message Format

- Human-readable text-based messages

- All lines end with carriage return-line feed (CRLF)
  - Also known as `\r\n` in your favorite programming language

- Applicable to both requests and responses

HTTP Generic Message Format

```
<start-line>
```
request method or response status code

```
<message-headers>
<empty-line>
[<message-body>]
[<message-trailers>]
```

HTTP Message Headers

- General format
  - `<header-name>: <header-value>`
- Mostly optional
  - `Host` required for HTTP/1.1 requests
  - Why would the `Host` header be required?

HTTP Requests

```
<request-line>
<general-headers>
<request-headers>
<entity-headers>
<empty-line>
[<message-body>]
[<message-trailers>]
```

HTTP Requests

- Request line

  `<METHOD> <request-url> <VERSION>`

- Headers
  - General: typically about message itself, can appear in request or response
  - Request: client request details for server
  - Entity: describe entity in message body (if any)

HTTP Requests

```
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
```

HTTP Requests

```
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
```

Request headers

General headers

HTTP Responses

```
<status-line>
<general-headers>
<response-headers>
<entity-headers>
<empty-line>
[<message-body>]
[<message-trailers>]
```

HTTP Responses

- Status line

```
<VERSION> <status-code> <reason-phrase>
```

- Headers
  - General: refer to message itself, can appear in request or response
  - Response: additional response data
  - Entity: describe entity in message body (if any)

HTTP Responses

```
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 Responses

```
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 ...
```

Entity headers

Response headers

General headers

HTTP Methods

`GET`	Retrieve resource from server
`HEAD`	Retrieve headers, but not resource
`POST`	Submit data to server (for example, a form)
`OPTIONS`	Retrieve available methods
`PUT`	Store resource at server
`DELETE`	Delete resource
`TRACE`	Retrieve copy of request for diagnostics
`CONNECT`	Open tunnel via proxy

HTTP Status Codes

`1xx`	Informational message (`100 Continue`)
`2xx`	Success (`200 OK`)
`3xx`	Redirection (`301 Moved Permanently`)
`4xx`	Client error (`404 Not Found`)
`5xx`	Server error (`501 Not Implemented`)

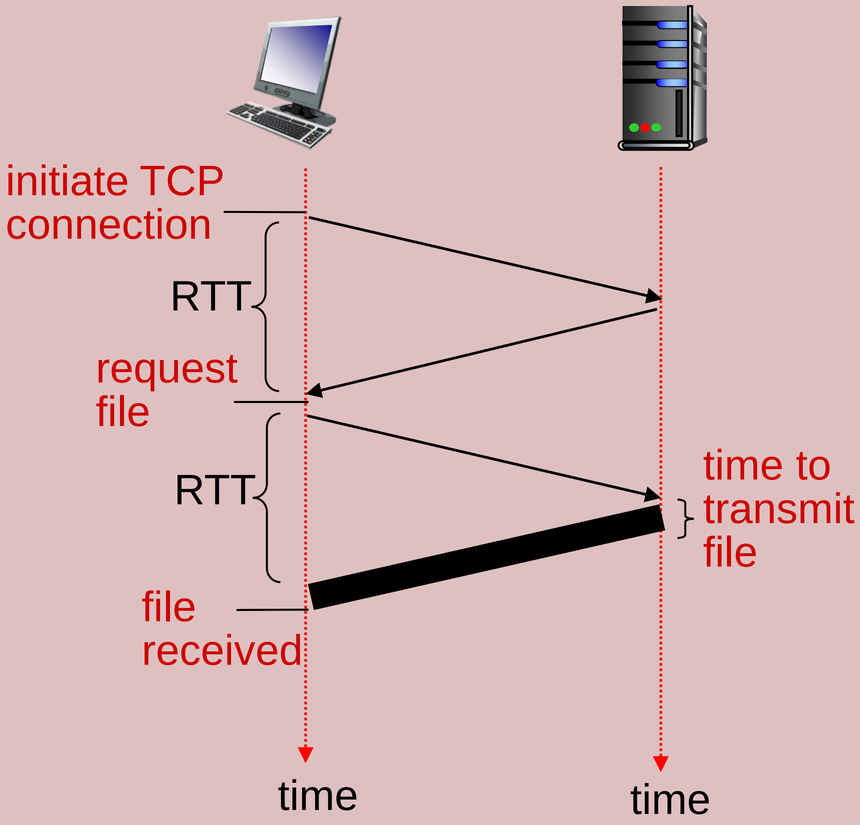
HTTP Connections

- Originally, a single transitory TCP connection for request-response (HTTP/0.9 and HTTP/1.0)
  - TCP connection established (3 segments)
  - Request / response exchange
  - TCP connection terminated (4 segments)

- HTML file often references other objects

- What if request/response exchange averages approximately 10 segments?

HTTP Connections



HTTP Connections

- Non-persistent HTTP connections
  - Downloading multiple objects requires multiple TCP connections
  - Open/close connection overhead per object

- Persistent HTTP connections
  - Default introduced in HTTP/1.1
  - Download multiple objects in a single TCP connection
  - Open/close connection overhead amortized across multiple objects

HTTP Pipelining

- Works in conjunction with persistent connections
- Send requests for multiple objects without waiting for response

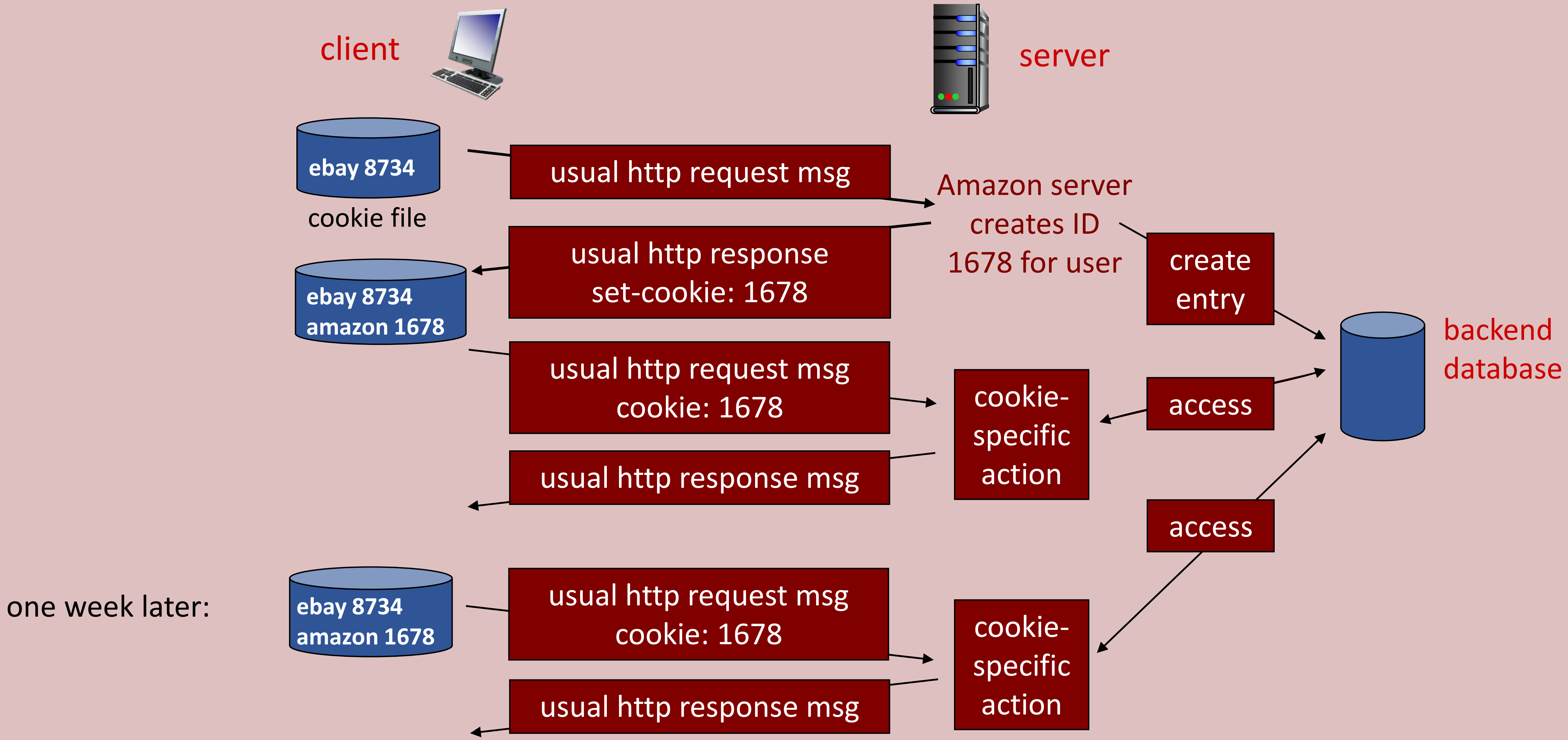
Parallel HTTP Connections

- Should be unnecessary with pipelining over persistent connections
- Frowned upon (adds load to server and network due to TCP overhead)
- HTTP/1.1 allows up to 2 persistent connections per server
- Some browser implementations do not conform and allow more than 2

HTTP Cookies

- Workaround for statelessness of HTTP
- `Set-Cookie` header in HTTP response
- Cookie file managed locally by browser and remotely by Web server database
- `Cookie` header sent in subsequent HTTP request

HTTP Cookies



HTTP Cookies

- Typical applications
  - Authorization
  - Shopping carts
  - Recommendations
  - User sessions (for example, e-mail)

- Privacy implications
  - Track user interactions across multiple requests
  - First-party
  - Third-party (for example, web bugs)

Web Caching

- Reduce user-perceived latency while browsing
- Reduce redundant requests
  - Less network transmission load
  - Less origin server processing load
- Browser caches versus proxy servers

Cache-Related HTTP Headers

- `Cache-Control`: directives that manage caching (more details in RFC 2616)
- `Expires`: specifies date/time when object should be considered stale; ignored if `Cache-Control: max age` is present
- `If-Modified-Since`: specifies time bound on when objects should be returned

Cache Replacement Strategies

- Least Recently Used (LRU)
  - Assign age to objects based on last access
  - Remove oldest object
- Least Frequently Used (LFU)
  - Rank objects by access frequency
  - Remove object that is least frequently used
- Size of objects (SIZE)
  - Delete largest object
  - Hopefully make space for multiple smaller objects

Browser Cache

- Dedicated local resources
  - Portion of browser process memory
  - Portion of file system disk space

- Cache revalidation: check cached version to origin server version

Browser Cache

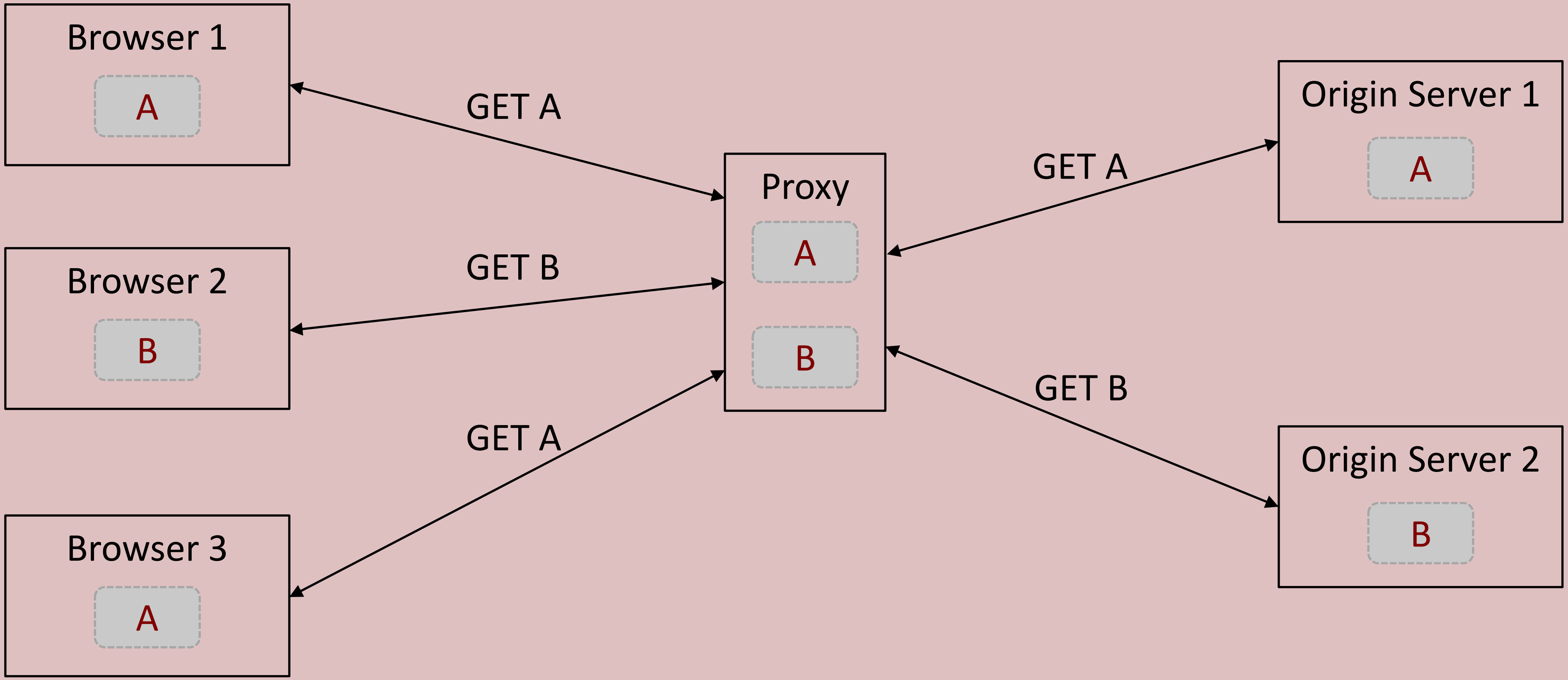
- Cache consistency
  - Strong: revalidate for each request
  - Weak: use heuristic to determine whether or not revalidation is necessary

- Most browsers offer option to force request to origin server

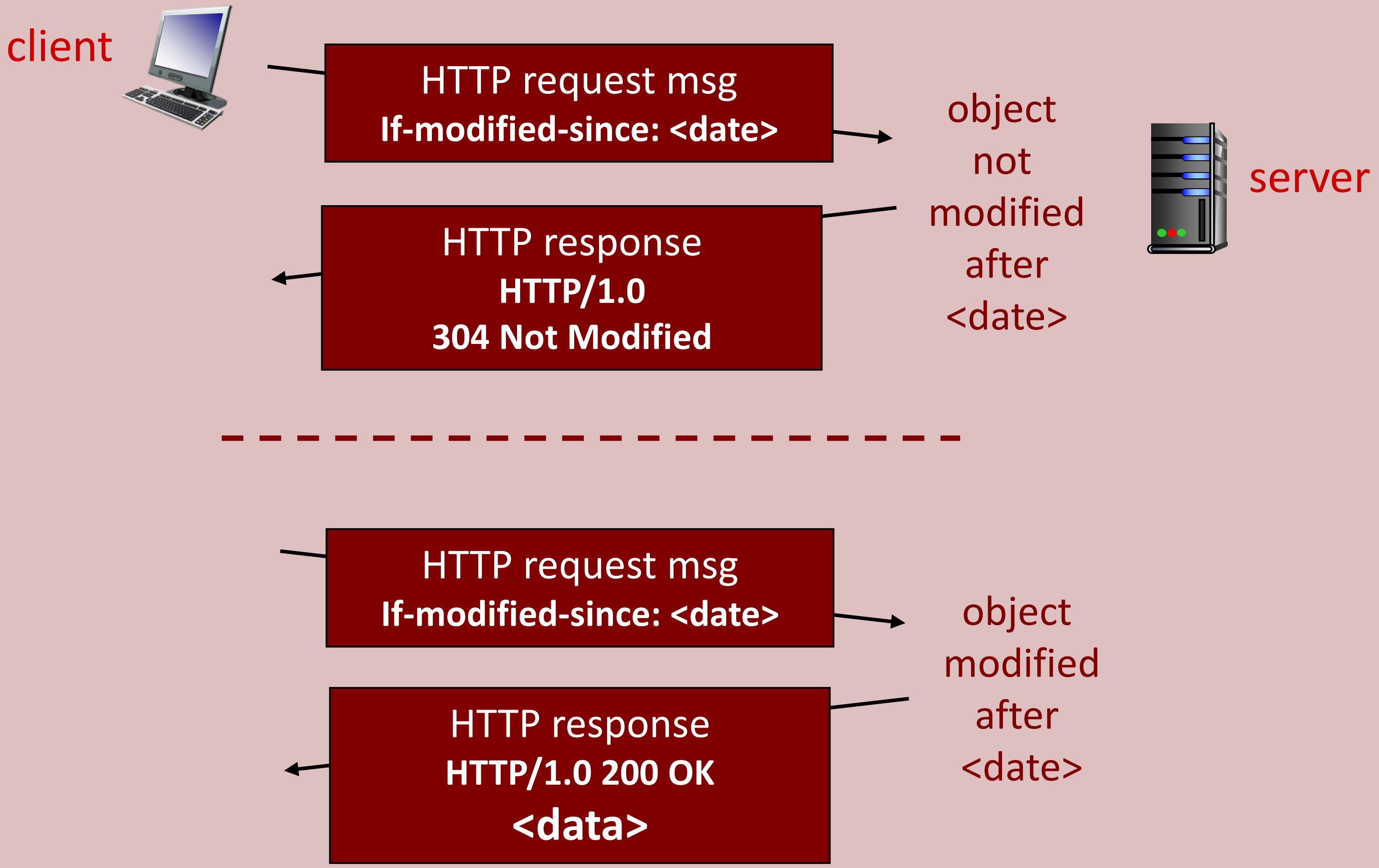
Web Proxy Server

- Acts as server to browser
- Acts as client to origin server
- Organization or ISP can place proxy server between browser and origin servers
  - Caching
  - Modify requests and/or responses

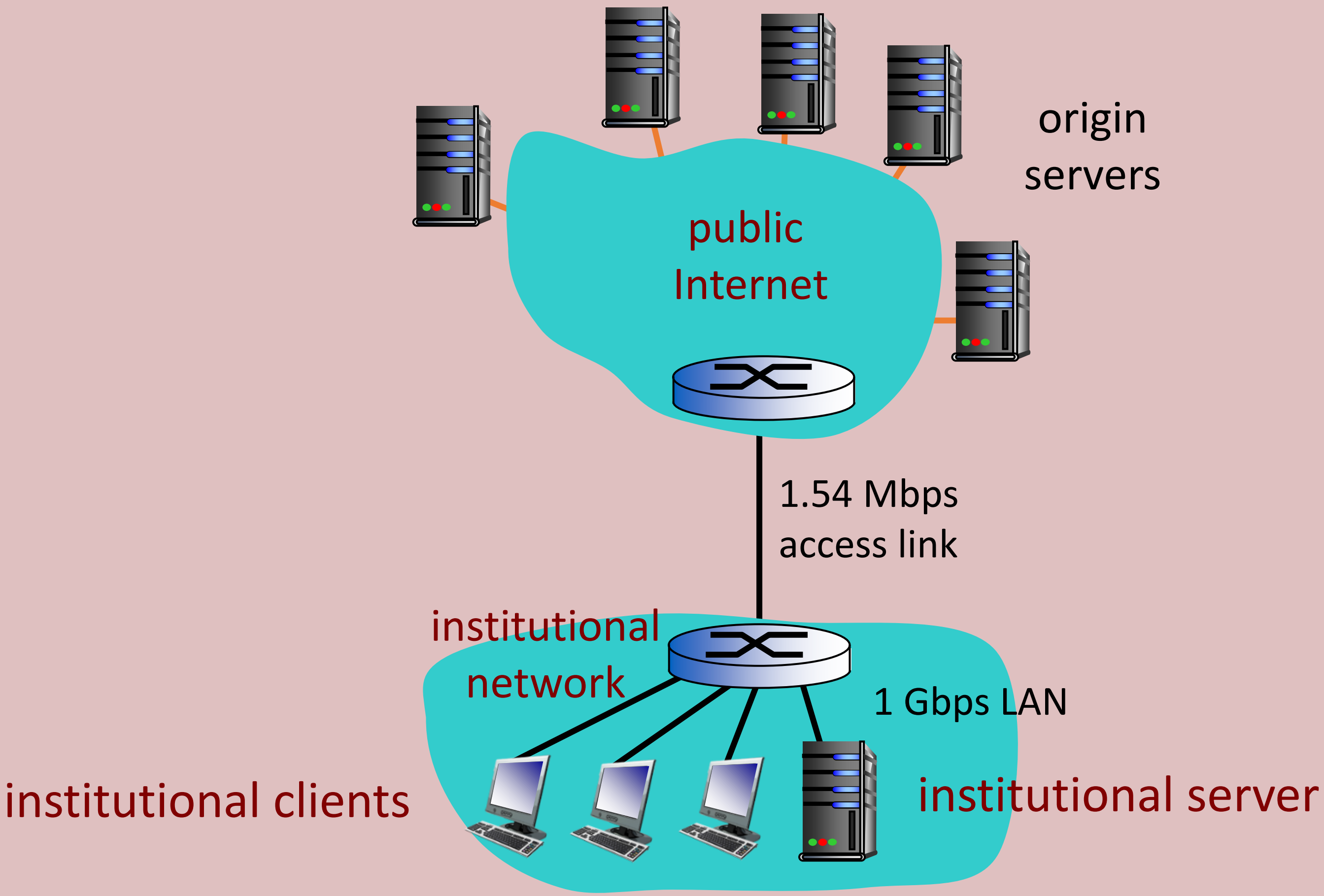
Web Proxy Server



Conditional GET



Caching Example



Thank You!

Networks

DNS

Adopted from material in “Computer Networking: A Top Down Approach” by Kurose and Ross and slides developed by William Conner

DNS

Section 2.4

Domain Name System (DNS)

- Name resolution
  - Mapping hostnames to IP addresses
  - Mapping IP addresses to hostnames
  - Host aliasing
  - Mail server aliasing
- Load balancing
- Implemented as hierarchical, distributed database

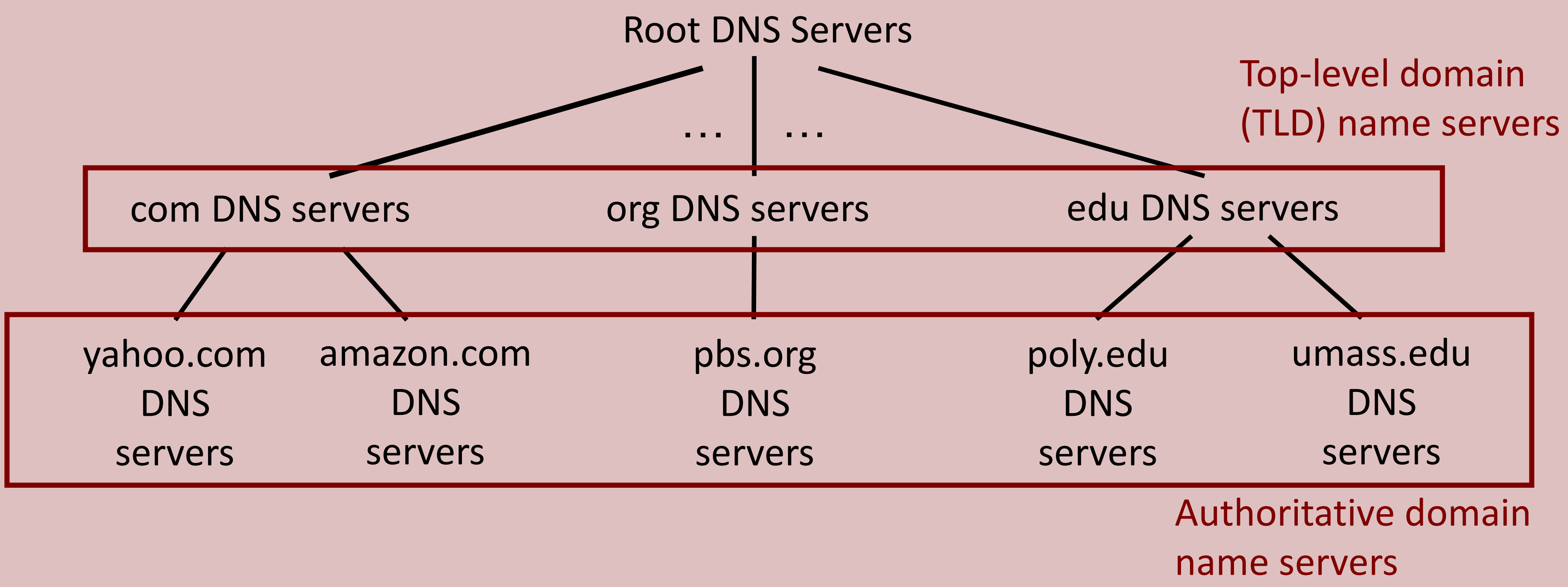
DNS Hierarchical Servers

- Root: top of hierarchy
- Top-level domain (TLD): com, edu, org, net, etc.
- Authoritative: maintained by an organization
- Local: “default name server”, typically provided by ISP, not part of hierarchy

How Can You Bypass Local DNS Server?

- 8.8.8.8
- 8.8.4.4
- Others

DNS Hierarchy



DNS Root Name Servers

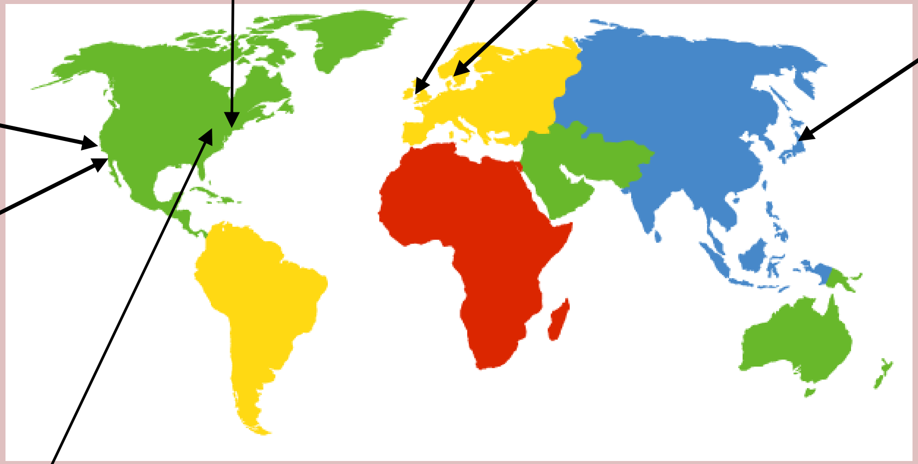
c. Cogent, Herndon, VA (5 other sites)
d. U Maryland College Park, MD
h. ARL Aberdeen, MD
j. Verisign, Dulles VA (69 other sites)

k. RIPE London (17 other sites)

i. Netnod, Stockholm (37 other sites)

e. NASA Mt View, CA
f. Internet Software C.
Palo Alto, CA (and 48 other sites)

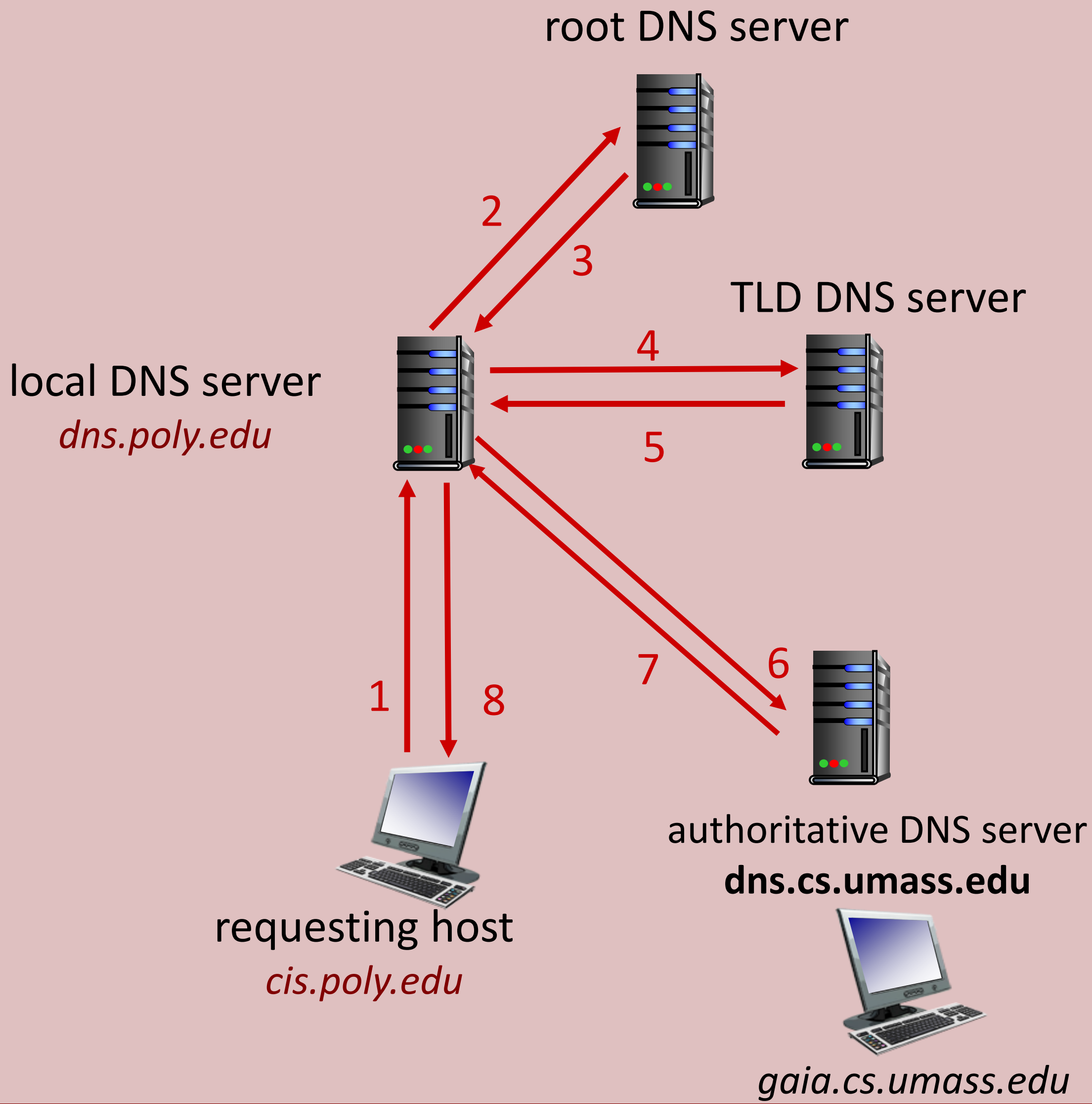
m. WIDE Tokyo (5 other sites)



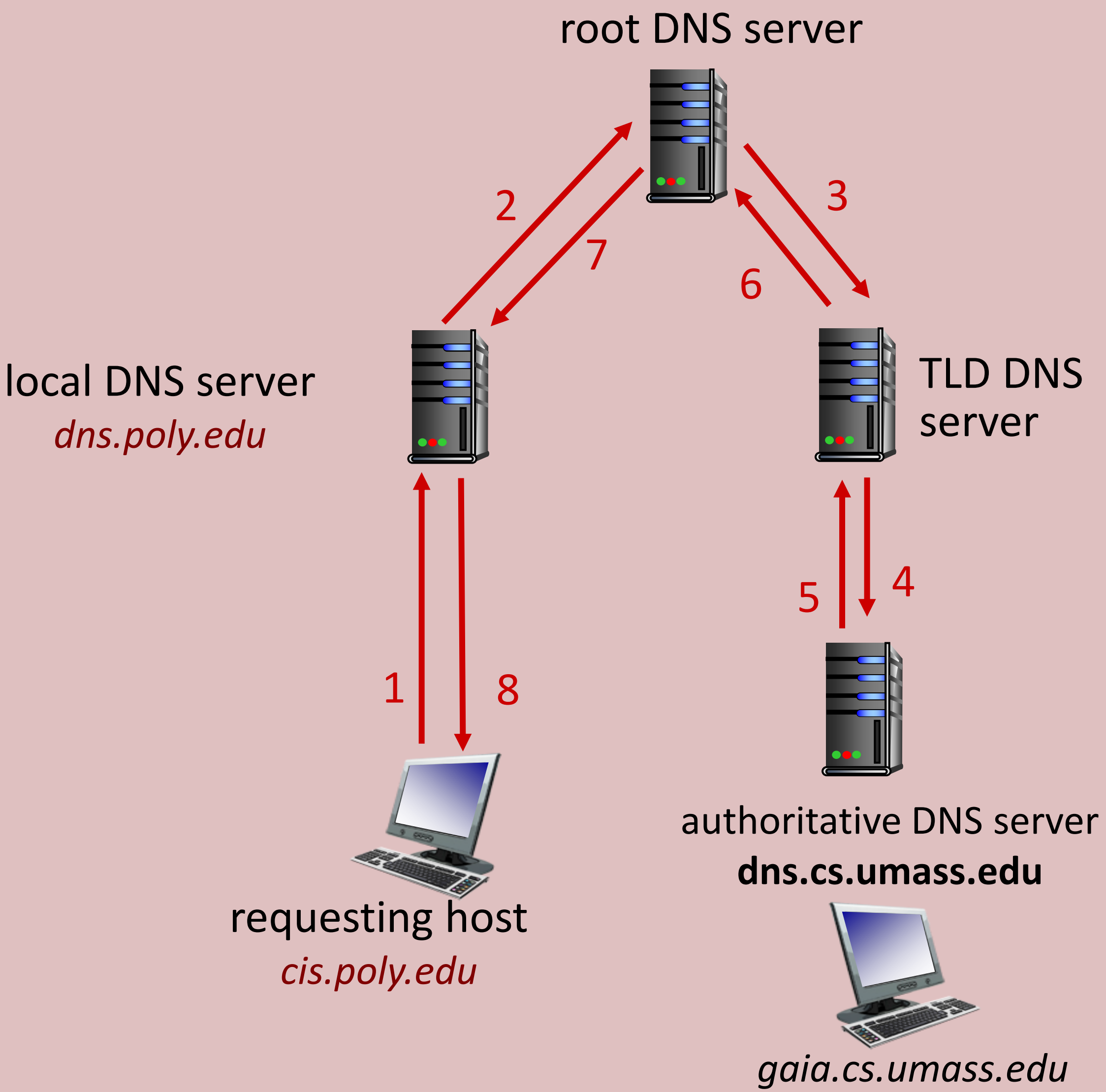
a. Verisign, Los Angeles, CA
(5 other sites)
b. USC-ISI Marina del Rey, CA
l. ICANN Los Angeles, CA
(41 other sites)

g. US DoD
Columbus, OH (5
other sites)

Iterative DNS Query



Recursive DNS Query



Iterative vs. Recursive

- Which type of query reduces the load on TLD and root name servers?

DNS Cache

- Name servers cache learned mappings
  - Entries removed after some TTL
  - Local name servers typically cache TLDs

- Cached entries might be stale
  - New hostname-to-IP mapping
  - Cached entries expire after TTL

- Notify/update mechanism specified in RFC 2136 to avoid stale entries

DNS Resource Records

- DNS database entries are resource records (RR) with format (name, value, type, ttl)

RR Code	RR Type	Name	Value
A	IPv4 address	Hostname	32-bit IP address
AAAA	IPv6 address	Hostname	128-bit IP address
NS	Name server	Domain name	Authoritative domain server for domain
CNAME	Canonical name	Alias	Canonical name
MX	Mail exchange	Domain name	Mail server responsible for domain

DNS Transports

- Uses both UDP and TCP for underlying transport

- UDP
  - Most name resolutions
  - No need to establish connection
  - Short and fast message exchange

- TCP
  - Messages that exceed 512 bytes
  - For example, Zone transfer

host

```
$ host linux.cs.uchicago.edu
linux.cs.uchicago.edu has address 128.135.164.112
linux.cs.uchicago.edu has address 128.135.164.173
linux.cs.uchicago.edu has address 128.135.164.171
linux.cs.uchicago.edu has address 128.135.164.172
linux.cs.uchicago.edu has address 128.135.164.115
```

nslookup

```
$ nslookup linux.cs.uchicago.edu
Server:         128.135.164.141
Address:        128.135.164.141#53

Non-authoritative answer:
Name:   linux.cs.uchicago.edu
Address: 128.135.164.171
Name:   linux.cs.uchicago.edu
Address: 128.135.164.112
Name:   linux.cs.uchicago.edu
Address: 128.135.164.173
Name:   linux.cs.uchicago.edu
Address: 128.135.164.172
Name:   linux.cs.uchicago.edu
Address: 128.135.164.115
```

dig

```
$ dig linux.cs.uchicago.edu
<other stuff>
;; ANSWER SECTION:
linux.cs.uchicago.edu.  120     IN      A       128.135.164.115
linux.cs.uchicago.edu.  120     IN      A       128.135.164.171
linux.cs.uchicago.edu.  120     IN      A       128.135.164.173
linux.cs.uchicago.edu.  120     IN      A       128.135.164.172
linux.cs.uchicago.edu.  120     IN      A       128.135.164.112
<other stuff>
```

Hostnames

- What are the hostnames for those five IP addresses?
- How can you find out?

DNS Attacks

- DDoS attacks
  - Overwhelm root servers with queries
  - Overwhelm TLD servers with queries
- DNS cache poisoning
  - Put bogus mappings into name servers
  - Can be prevented with DNSSEC
- Amplification
  - Send queries with spoofed IP source target address
  - Name server send replies DDoS to target

Thank You!

Networks

Peer-to-Peer

Adopted from material in “Computer Networking: A Top Down Approach” by Kurose and Ross and slides developed by William Conner

Peer-to-Peer

Section 2.5

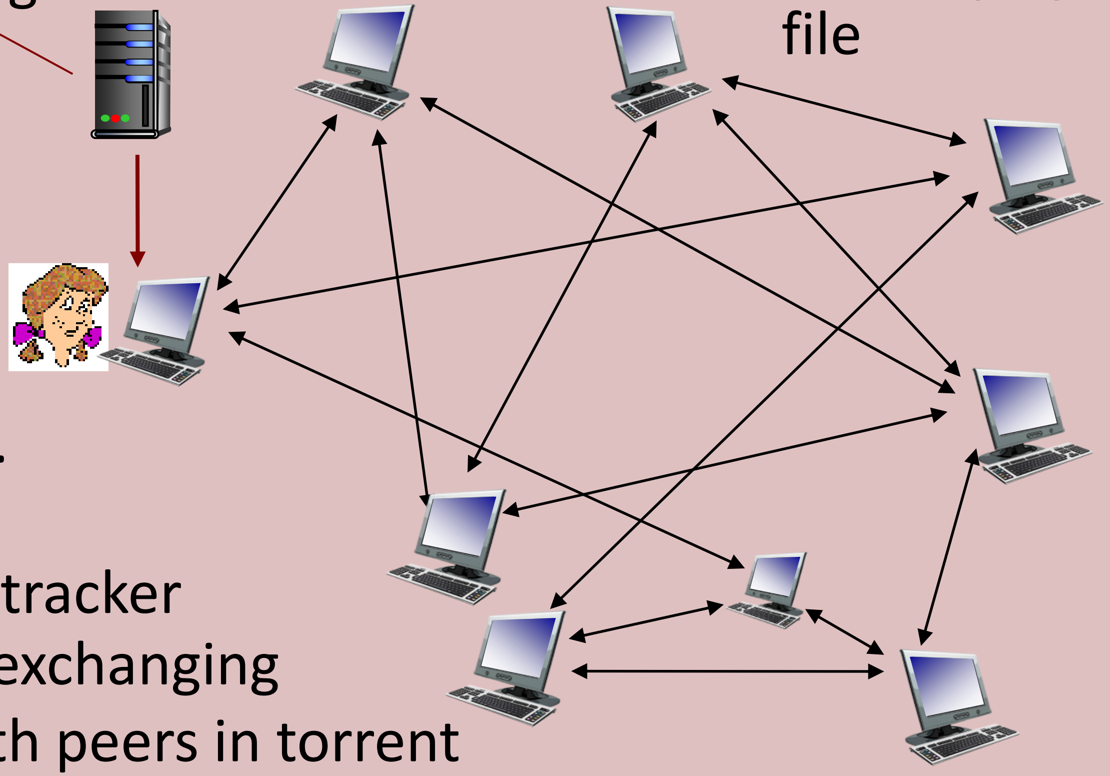
P2P File Distribution

- No always-on server
- Peers intermittently login and change IP addresses
- End hosts directly communicate with each other
- BitTorrent is a major p2p file distribution
  - Users exchange file chunks
  - File chunk is 256KB

BitTorrent

tracker: tracks peers participating in torrent

torrent: group of peers exchanging chunks of a file



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

BitTorrent

- Bootstrapping for new peer
  - Registers with tracker to get list of peers
  - Joins torrent by connecting with peers
- After bootstrapping, peer downloads and uploads file chunks to/from other peers
- Can change peers over time
- After download complete
  - Selfishly leave, or
  - Altruistically remain to upload to peers

BitTorrent

- Downloading chunks
  - Request available chunk list from peers
  - Request rarest missing chunk first

- Uploading chunks (tit-for-tat)
  - Send chunks to 4 peers sending to you at highest rate
  - “Choke” other peers
  - Re-evaluate top-4 uploaders every 10 seconds
  - Optimistically “unchoke” random peer every 30 seconds to see if they make your top-4

Thank You!

Networks

Video Streaming and CDNs

Adopted from material in “Computer Networking: A Top Down Approach” by Kurose and Ross and slides developed by William Conner

Video Streaming and CDNs

Section 2.6

Video Streaming

- Video is 58% of global downstream traffic*

Provider	Global Downstream*	Number of Users
Netflix	15%	139M (2019)
YouTube	11%	2B (2019)
Amazon Prime Video	4%	26M (2017)

- Challenges
  - Scaling to over 1B users (CDNs)
  - Supporting device and network heterogeneity (DASH)

*Source: Sandvine Global Internet Phenomena – US (October 2018)

Video Coding

- Digital image: array of picture elements (aka pixels)
- Digital video: sequence of digital images displayed at constant rate (like 24 fps)
- Video coding: take advantage of redundancies to decrease encoding bits
  - Spatial: within single image
  - Temporal: across consecutive images

DASH Video Streaming

- Dynamic Adaptive Streaming over HTTP (DASH)

- Video file format
  - Divided into multiple chunks
  - Each chunk encoded at different bit rate and stored
  - Manifest file: provides URLs for chunks at the different bit rates

DASH Video Streaming

- Regular server
  - Serves video chunks from requested URLs
  - No difference from Web server

- Intelligent client
  - Periodically measures downstream bandwidth
  - Requests chunk URLs based on current bandwidth conditions according to manifest
  - Manages request timing
  - Sends request to close servers with available bandwidth

Content Distribution Network (CDN)

- Scaling video streaming to millions of simultaneous viewers

- Why multiple, geographically distributed video servers are necessary?
  - Avoid single point of failure
  - Reduce server-side network congestion
  - Not possible to be close to all viewers

Content Distribution Network (CDN)

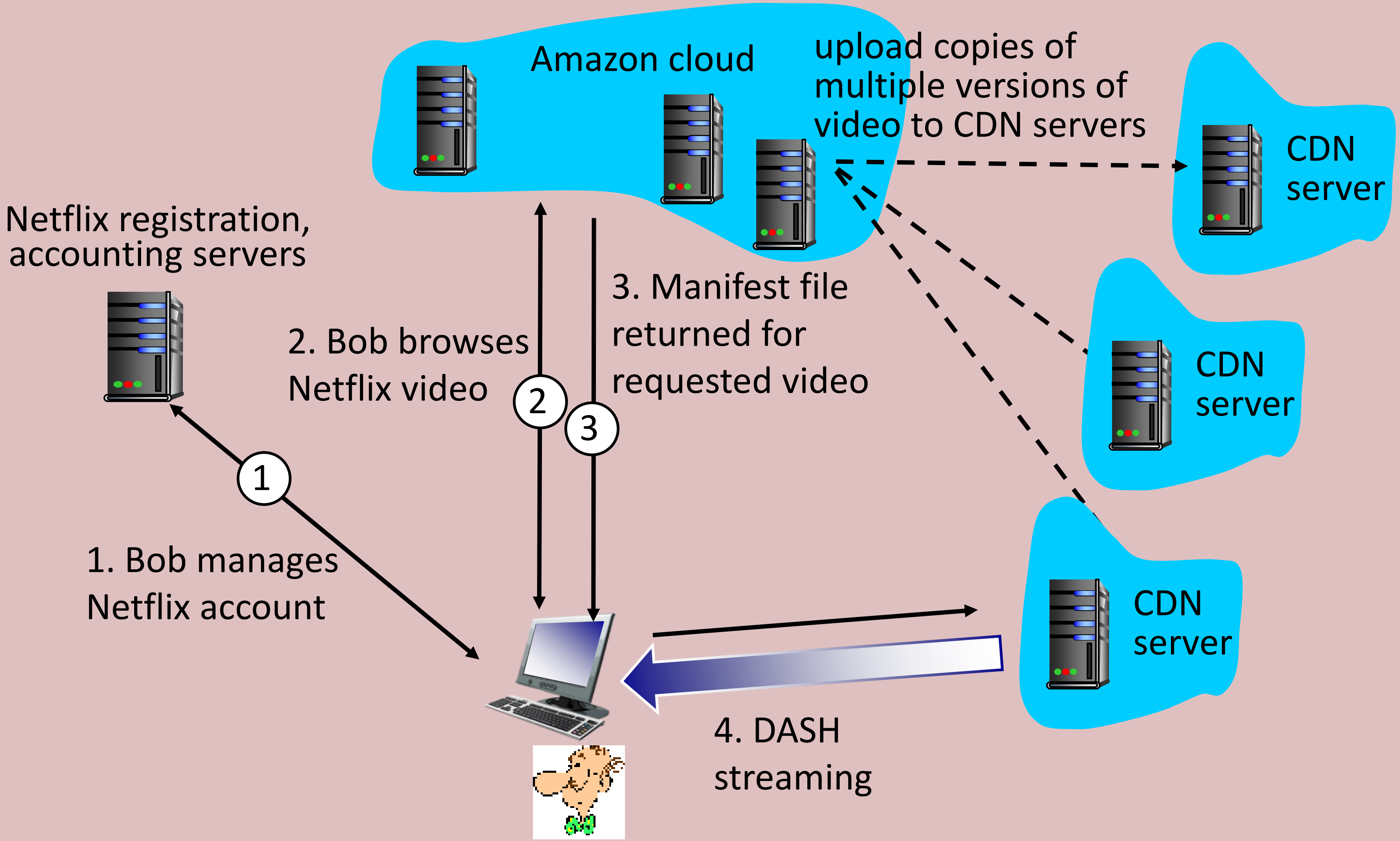
- Serve video (and other Web content) from multiple geographically distributed servers
- Place CDN servers close to viewers' access networks rather than origin servers
- Third-party versus private CDNs
  - Akamai (1700+ locations)
  - Fastly
  - Netflix Open Connect
  - Google Global Cache

CDN Server Placement

- Enter Deep Philosophy (Akamai)
  - Deploy servers in access ISPs
  - Closer to users
  - Highly distributed

- Bring Home Philosophy (Limelight)
  - Deploy servers at Internet Exchange Points (IXPs)
  - Lower maintenance and management overhead

Netflix


Amazon cloud
upload copies of multiple versions of video to CDN servers
CDN server
CDN server
CDN server
Netflix registration, accounting servers
2. Bob browses Netflix video
3. Manifest file returned for requested video
2
3
1
1. Bob manages Netflix account
4. DASH streaming

Thank You!

Networks

File Transfer

Adopted from material in “Computer Networking: A Top Down Approach” by Kurose and Ross and slides developed by William Conner

File Transfer

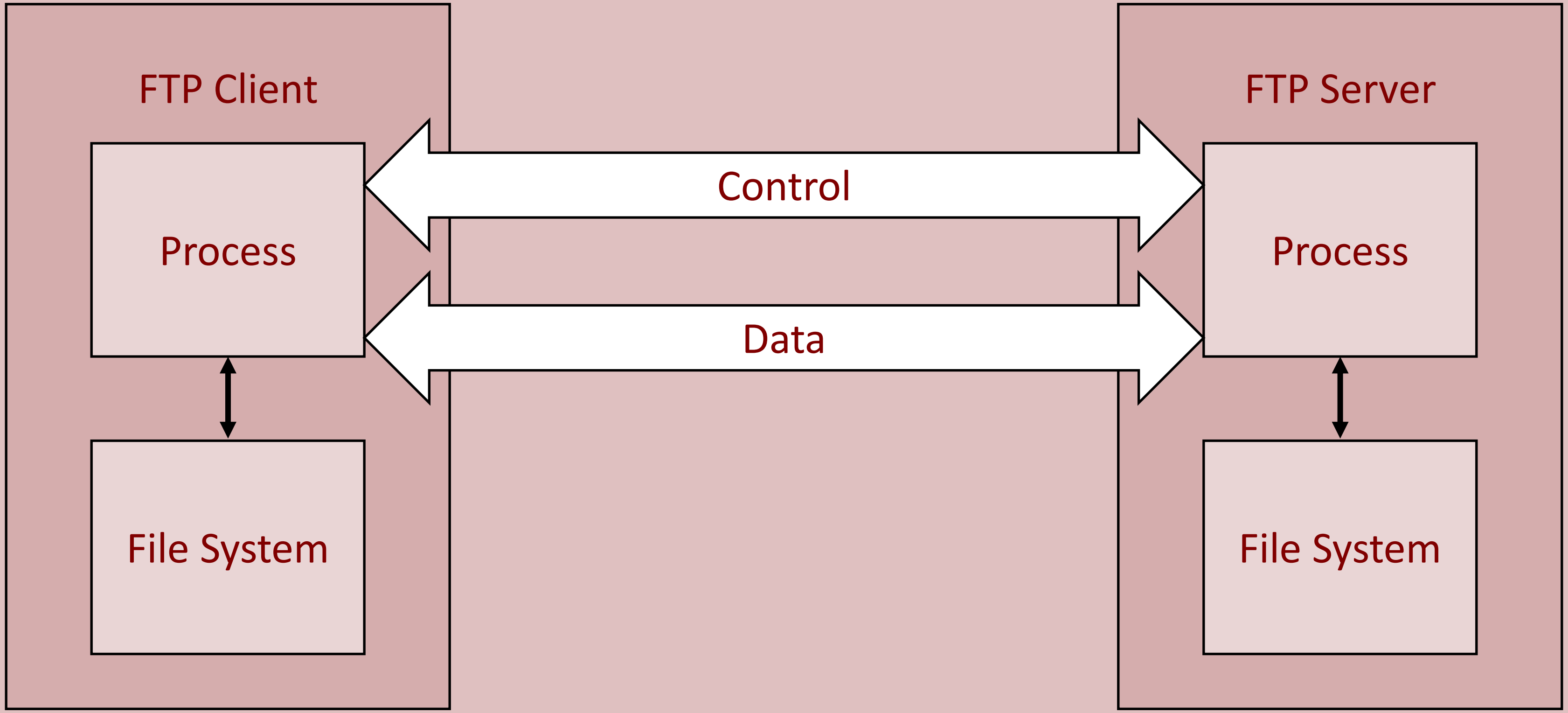
File Transfer Protocol (FTP)

- One of the earliest application protocols in the TCP/IP suite
- Text-based (similar to HTTP)
- Efficient and reliable transfer of files between any two connected devices
- Adheres to client/server paradigm
- Supports many user commands

FTP Communication Channels

- Two logical communication channels are maintained between the client and server
- Control: single TCP connection for exchange of FTP commands and replies; no file data
- Data: one or more TCP connections for sending/receiving file data

FTP Operational Model



Example FTP User Commands

Command	Description
`user <username>`	login (prompts for password)
`get <src> [<dst>]`	read file
`put <src> [<dst>]`	write file
`rename <old> <new>`	rename file
`cd <dir>`	change directory to
`delete <file>`	delete file
`rmdir <dir>`	remove directory
`quit`	terminate session and exit

Example FTP Internal Commands

Command	Description
USER	username for FTP session
PASS	password for authentication
PORT	client data port to server
RETR	retrieve file from server
STOR	store file on server
DELE	delete file from server
CWD	change working directory
TYPE	specify file data type (ASCII, image, etc)
QUIT	logout

Trivial File Transfer Protocol (TFTP)

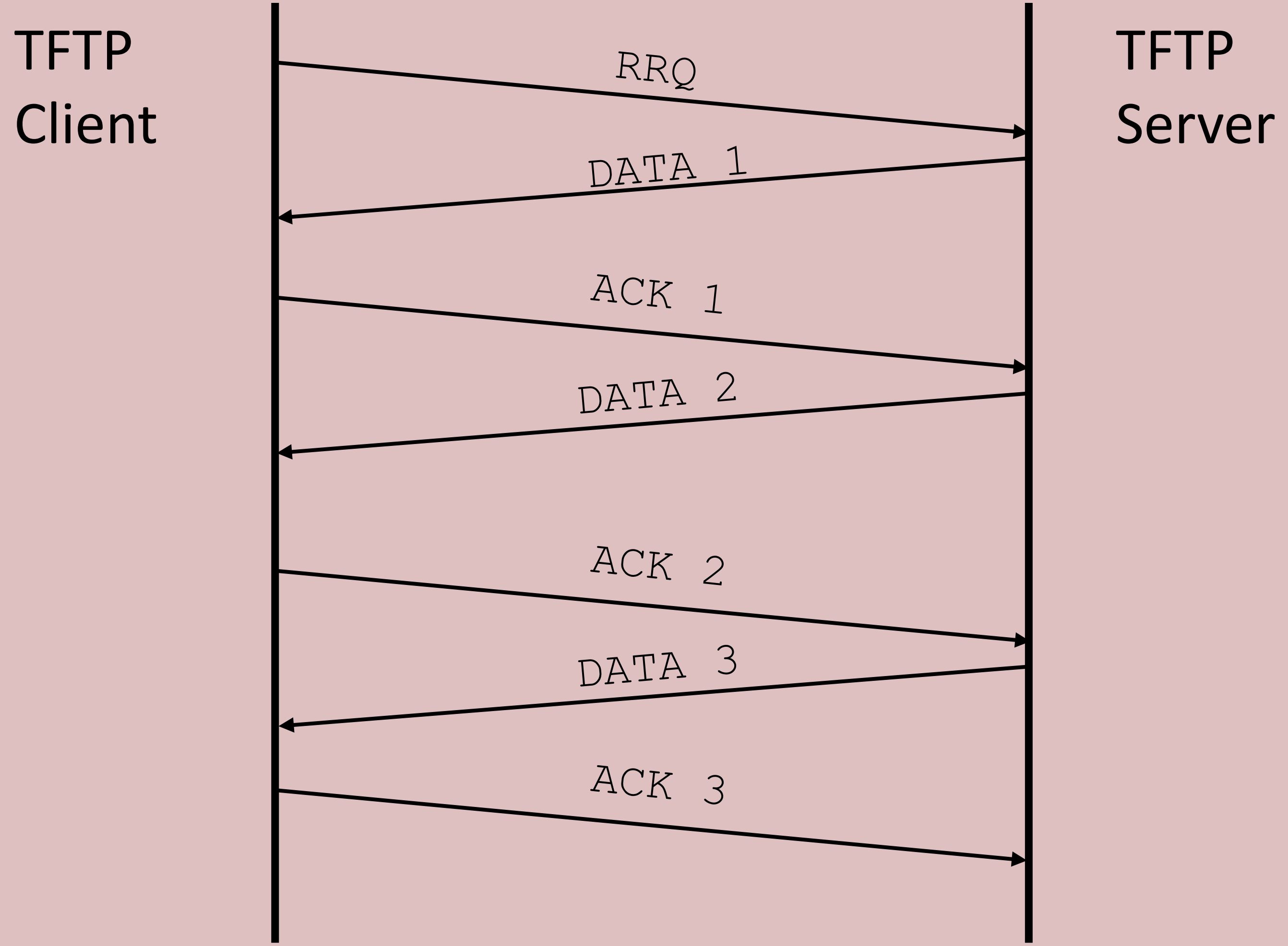
- FTP alternative with smaller footprint and less functionality
- Binary protocol, rather than text-based
- Runs on top of UDP, rather than TCP
    - Reliable data transmission is implemented in the application layer

TFTP Message Types

Message Type	Description
`RRQ`	read request
`WRQ`	write request
`DATA`	data block
`ACK`	acknowledgements
`ERROR`	error notification

NOTE: More details available in RFC 1350

TFTP Read (File Download)



TFTP Write (File Upload)

Thank You!