ILLINOIS

CS 438: Computer Networks (Spring 2026)

Application Layer

Some slides are adapted from *Computer Networking: A Top-Down Approach.*
All material copyright 1996-2020
J.F Kurose and K.W. Ross, All Rights Reserved

Learning Objectives

- **Application-layer basics**
- DNS
- HTTP
- More on DNS and HTTP

What happens after you enter the following URL in your browser:

`https://illinois.edu/about/image.png`

host name (`https://illinois.edu`) / path name (`/about/image.png`)

An application-layer protocol defines:

- types of messages exchanged,
  - e.g., request, response
- message syntax:
  - what fields in messages & how fields are delineated
- message semantics
  - meaning of information in fields
- rules for when and how processes send & respond to messages

open protocols:

- defined in RFCs, everyone has access to protocol definition
- allows for interoperability
- e.g., HTTP, DNS, SMTP

proprietary protocols:

- e.g., Skype

© CS 438 Staff, University of Illinois

Example application-layer protocols

application	application-layer protocol	transport protocol
file transfer/download	FTP [RFC 959]	TCP
e-mail	SMTP [RFC 5321]	TCP
Web documents	HTTP 1.1 [RFC 7320]	TCP
Internet telephony	SIP [RFC 3261], RTP [RFC 3550], or proprietary	TCP or UDP
streaming audio/video	HTTP [RFC 7320], DASH	TCP
interactive games	WoW, FPS (proprietary)	UDP or TCP

© CS 438 Staff, University of Illinois

Learning Objectives

- Application-layer basics
- **DNS**
- HTTP
- More on DNS and HTTP

© CS 438 Staff, University of Illinois

Host Names vs. IP addresses

- Host names
  - Mnemonic name appreciated by humans
  - Variable length, full alphabet of characters
  - Provide little (if any) information about physical location
  - Examples: `www.cnn.com` and `bbc.co.uk`
- IP addresses
  - Numerical address appreciated by routers
  - Fixed length, binary number
  - Hierarchical, related to host location
  - Examples: `64.236.16.20` and `212.58.224.131`

Separating Naming and Addressing

- Names are easier to remember
  - `cnn.com` vs. `64.236.16.20` (but not shortened urls)
- Addresses can change underneath
  - Move `www.cnn.com` to `4.125.91.21`
  - e.g., renumbering when changing providers

© CS 438 Staff, University of Illinois

Separating Naming and Addressing

- Name could map to multiple IP addresses
  - `www.cnn.com` may refer to multiple (8) replicas of the Web site
  - Enables
    - Load-balancing
    - Reducing latency by picking nearby servers
    - Tailoring content based on requester’s location/identity
- Multiple names for the same address
  - e.g., aliases like `www.cnn.com` and `cnn.com`

© CS 438 Staff, University of Illinois

Scalable (Name ↔ Address) Mappings

- Originally: per-host file
  - Flat namespace
  - `/etc/hosts`
  - SRI (Menlo Park) kept master copy
  - Downloaded regularly

- Why not centralize DNS?
  - Single point of failure
  - Traffic volume
  - Distant centralized database
  - Maintenance

- Doesn’t scale!

© CS 438 Staff, University of Illinois

Domain Name Service (DNS)

- Large scale dynamic, distributed application
  - Replaced Network Information Center (NIC)
- RFC 1034 and 1035
- Name space
  - Set of possible names
- Bindings
  - Maps internet domain names into IP addresses
- Name server
  - Resolution mechanism

© CS 438 Staff, University of Illinois

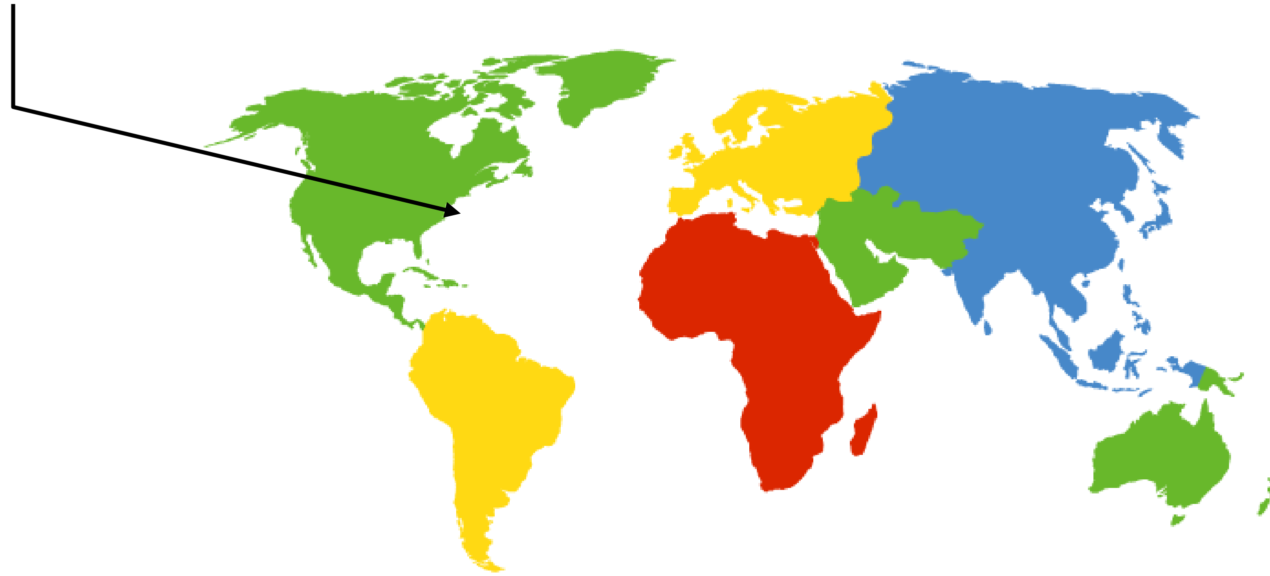
Applications' use of DNS

- Local DNS server ("default name server")
  - Usually near the endhosts that use it
  - Local hosts configured with local server (e.g., `/etc/resolv.conf`) or learn server via DHCP
- Client application
  - Extract server name (e.g., from the URL)
  - Do `getaddrinfo()` to trigger resolver code, sending message to server
- Server application
  - Extract client IP address from socket
  - Optional `getnameinfo()` to translate into name

© CS 438 Staff, University of Illinois

DNS Root

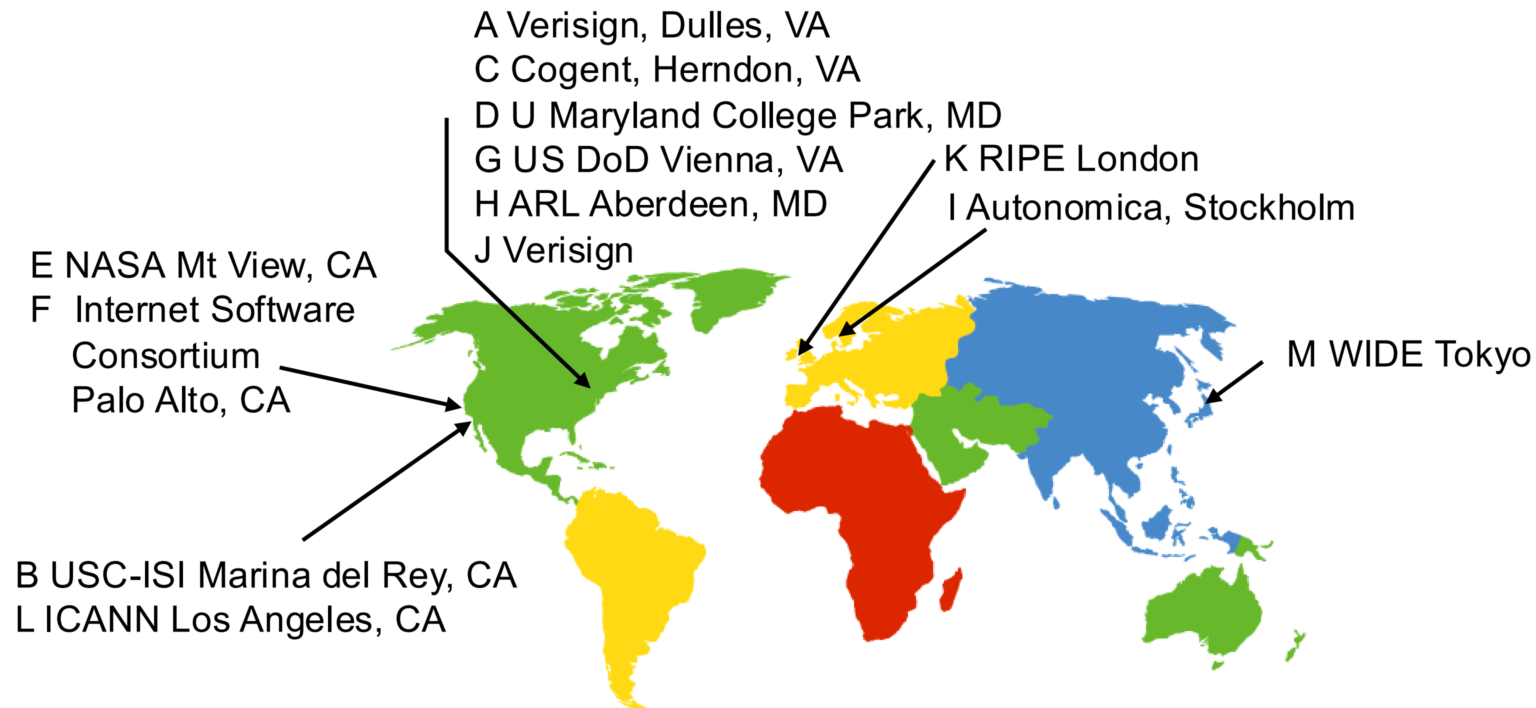
- Returns mapping to local DNS server
- How do we make the root scale?



© CS 438 Staff, University of Illinois

DNS Root Servers

- 13 root servers (see `http://www.root-servers.org/`)
  - Labeled A through M
- Does this scale?



© CS 438 Staff, University of Illinois

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`**.
    - Network Solutions maintains servers for **`com`** TLD
    - Educause for **`edu`** TLD
- Authoritative DNS servers
  - Organization's DNS servers
  - Provide authoritative hostname to IP mappings for organization's servers (e.g., Web, mail).
  - Can be maintained by organization or service provider

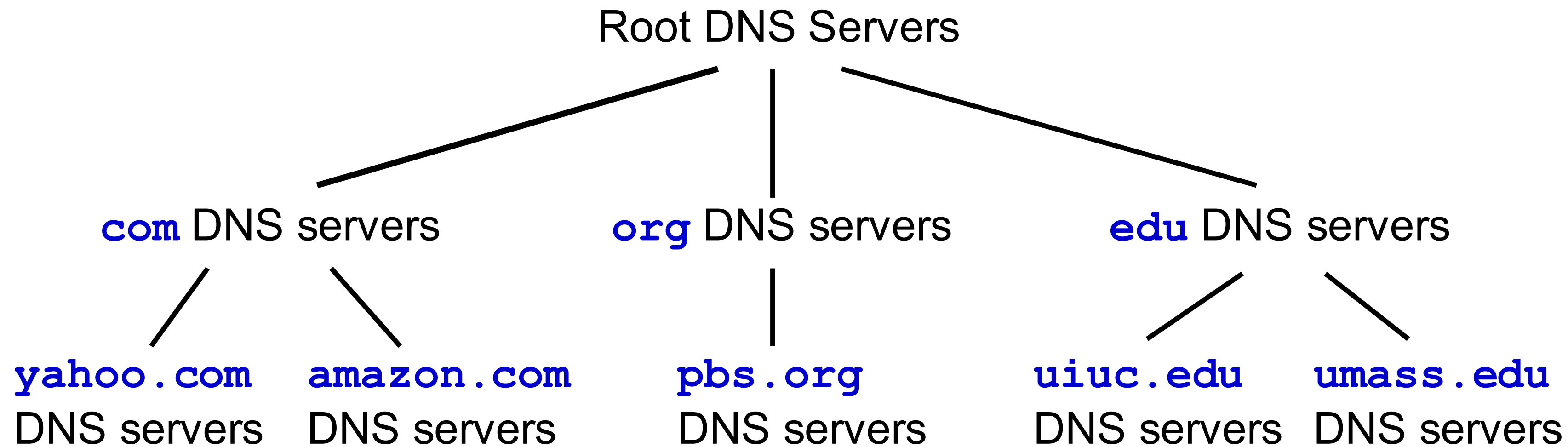
© CS 438 Staff, University of Illinois

Local Name Server

- One per ISP (residential ISP, company, university)
  - Also called "default name server"
- When host makes DNS query, query is sent to its local DNS server
  - Acts as proxy, forwards query into hierarchy
  - Reduces lookup latency for commonly searched hostnames

© CS 438 Staff, University of Illinois

Distributed, Hierarchical Database

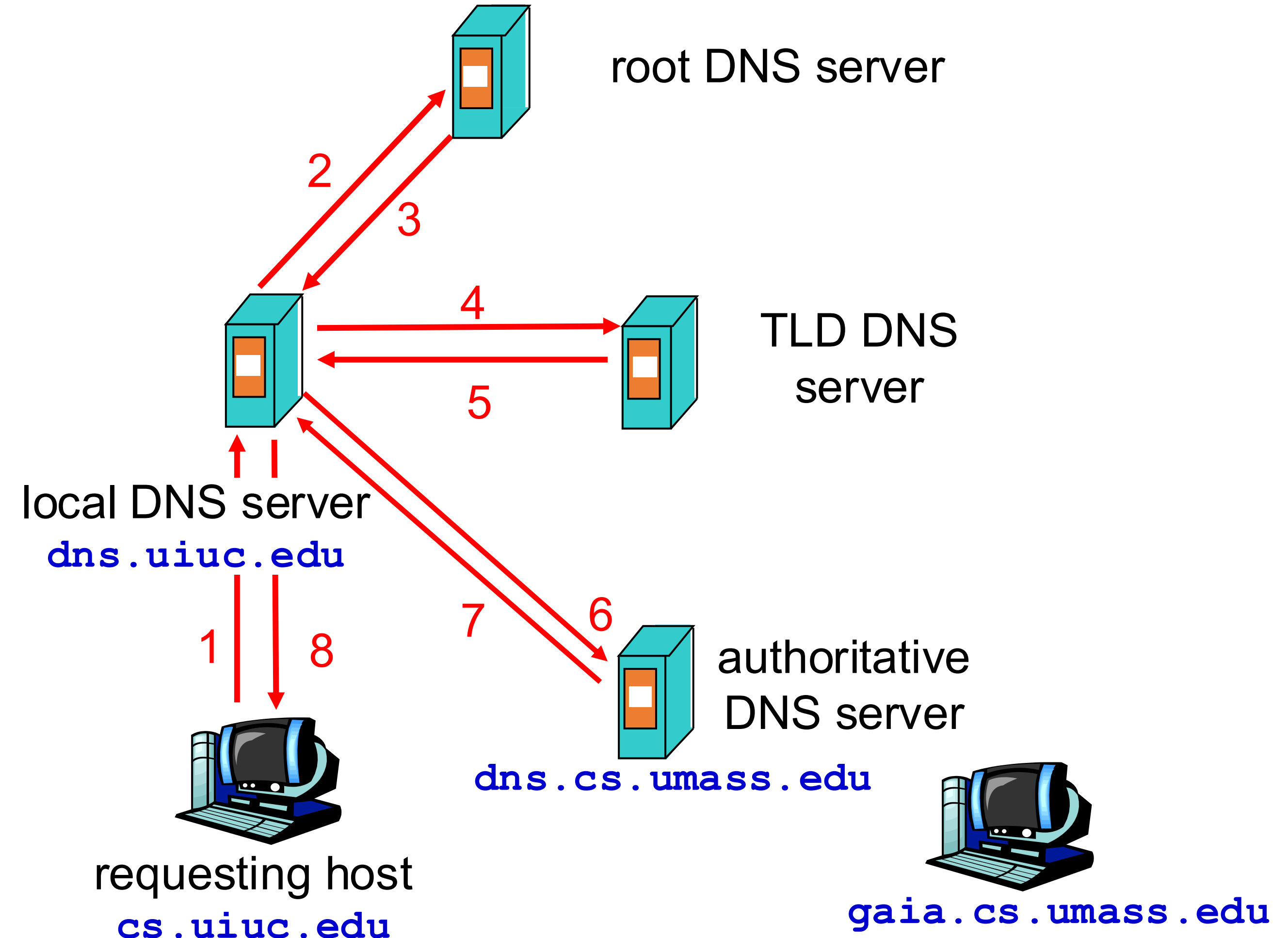


- Client wants IP for **`www.amazon.com`**
  - Client queries a root server to find **`com`** DNS server
  - Client queries **`com`** DNS server to get **`amazon.com`** DNS server
  - Client queries **`amazon.com`** DNS server to get IP address for **`www.amazon.com`**

© CS 438 Staff, University of Illinois

DNS – Name Server

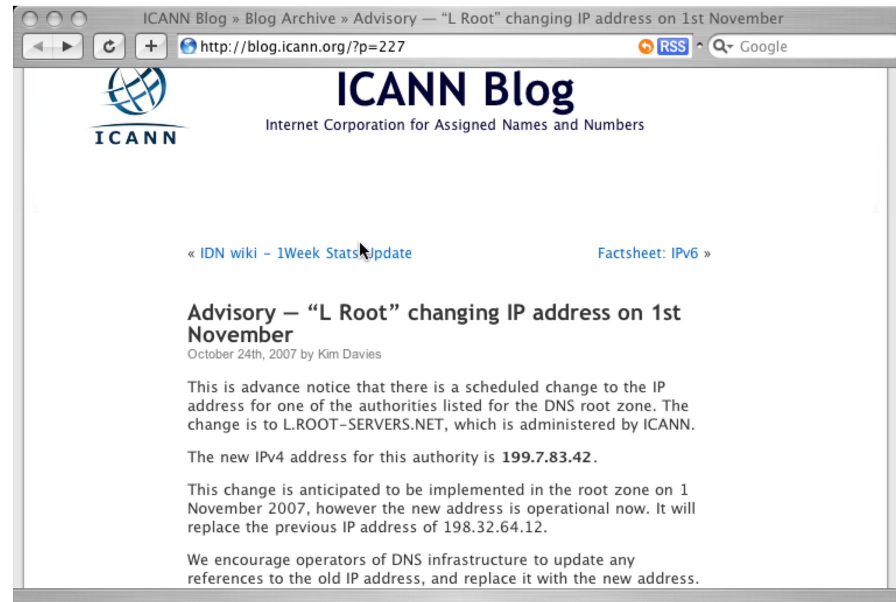
- Host at **`cs.uiuc.edu`**
  - Wants IP address for **`gaia.cs.umass.edu`**
- Recursive query
  - Ask server to get answer for you
  - e.g., request 1 and response 8
- Iterated query
  - Contacted server replies with name of server to contact
  - "I don't know this name, but ask this server"



© CS 438 Staff, University of Illinois

But how to know the root server IP?

- Hard-coded
- What if it changes?



© CS 438 Staff, University of Illinois

DNS: Caching

- Performing all these queries takes time
  - And all this before actual communication takes place
  - e.g., 1-second latency before starting Web download

- Caching can greatly reduce overhead
  - The top-level servers very rarely change
  - Popular sites (e.g., `www.cnn.com`) visited often
  - Local DNS server often has the information cached

© CS 438 Staff, University of Illinois

DNS: Caching

- How DNS caching works
  - DNS servers cache responses to queries
  - Responses include a “time to live” (TTL) field

- Once (any) name server learns mapping, it caches mapping
  - Cache entries timeout (disappear) after some time
  - TLD servers typically cached in local name servers longer
    - Thus root name servers not often visited

© CS 438 Staff, University of Illinois

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=PTR
  - `name` is reversed IP octets – why?
    - e.g., `78.56.34.12.in-addr.arpa`
  - `value` is corresponding hostname
- Type=CNAME
  - `name` is alias name for some "canonical" name
  - `value` is canonical name
  - e.g., `cs.mit.edu` or `ee.mit.edu` is really `eecs.mit.edu`
- Type=MX
  - `value` is name of mailserver associated with name
  - Also includes a weight/preference

© CS 438 Staff, University of Illinois

DNS Protocol

DNS protocol: *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
- Plus fields indicating size (0 or more) of optional header elements

16 bits	*16 bits*
Identification	**Flags**
# Questions	**# Answer RRs**
# Authority RRs	**# Additional RRs**
Questions (variable # of resource records)	
Answers (variable # of resource records)	
Authority (variable # of resource records)	
Additional information (variable # of resource records)	

© CS 438 Staff, University of Illinois

Reliability

- DNS servers are replicated
  - Name service available if at least one replica is up
  - Queries can be load-balanced between replicas
- Usually, UDP used for queries. Why?
  - Need reliability: must add and implement this on top of UDP
  - Spec supports TCP too, but not always implemented
- Try alternate servers on timeout
  - Exponential backoff when retrying same server

© CS 438 Staff, University of Illinois

Learning Objectives

- Application-layer basics
- DNS
- **HTTP**
- More on DNS and HTTP

© CS 438 Staff, University of Illinois

Web and HTTP

First, a quick review…

- web page consists of *objects,* each of which can be stored on different Web servers
- object can be HTML file, JPEG image, Java applet, audio file,…
- web page consists of *base HTML-file* which includes *several referenced objects, each* addressable by a *URL,* e.g.,

`www.someschool.edu/someDept/pic.gif`

host name: `www.someschool.edu`

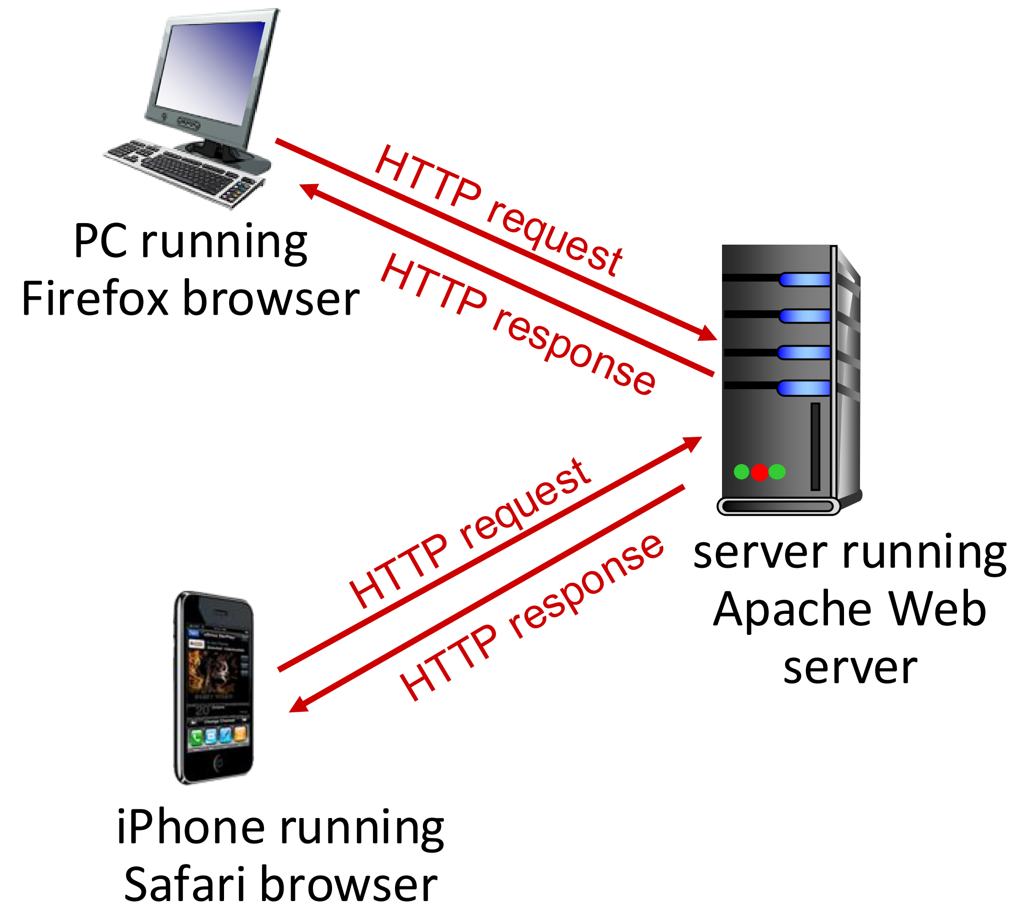
path name: `/someDept/pic.gif`

© CS 438 Staff, University of Illinois

HTTP overview

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
- *Note: we introduce "classic" HTTP first (until HTTP/2 & HTTP/3)*



© CS 438 Staff, University of Illinois

HTTP overview (continued)

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

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

© CS 438 Staff, University of Illinois

HTTP request message

- two types of HTTP messages: *request, response*
- HTTP request message:
  - human-readable text encoded with ASCII

carriage return character

line-feed character

request line (GET, POST, HEAD commands)

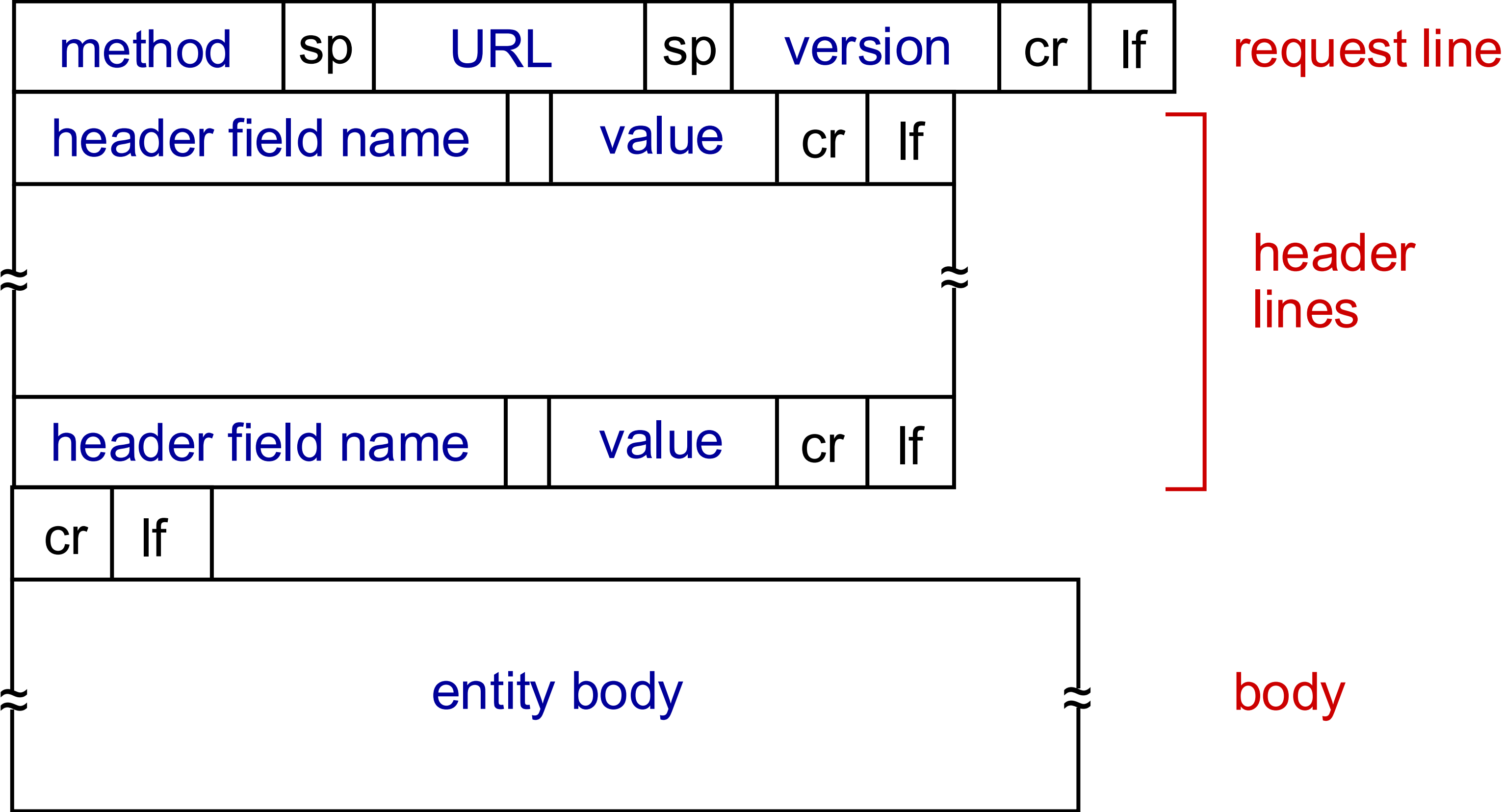
header lines

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

```
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
```


© CS 438 Staff, University of Illinois

HTTP request message: general format



© CS 438 Staff, University of Illinois

Other HTTP request messages

POST method:

- web page often includes form input
- user input sent from client to server in entity body (not idempotent)

GET method (for sending data to server):

- include user data in URL field of HTTP GET request message (following a '?'):

```
www.somesite.com/animalsearch?monkeys&banana
```

HEAD method:

- requests headers (only) that would be returned *if* specified URL were requested with an HTTP GET method.

PUT method:

- uploads new file (object) to server
- completely replaces file that exists at specified URL with content in entity body (idempotent)

© CS 438 Staff, University of Illinois

HTTP response message

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 ...
```


© CS 438 Staff, University of Illinois

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 message

301 Moved Permanently

- requested object moved, new location specified later in this message (in Location: field)

400 Bad Request

- request msg not understood by server

404 Not Found

- requested document not found on this server

505 HTTP Version Not Supported

© CS 438 Staff, University of Illinois

HTTP connections: two types

Non-persistent HTTP

1. TCP connection opened
2. at most one object sent over TCP connection
3. TCP connection closed

downloading multiple objects required multiple connections

Persistent HTTP

- TCP connection opened to a server
- multiple objects can be sent over *single* TCP connection between client, and that server
- TCP connection closed

© CS 438 Staff, University of Illinois

Non-persistent HTTP (HTTP 1.0): example

User enters URL: **`www.someSchool.edu/someDepartment/index.html`**

(containing 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/index.html

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

time

© CS 438 Staff, University of Illinois

Non-persistent HTTP (HTTP 1.0): example

User enters URL: **`www.someSchool.edu/someDepartment/index.html`**
(containing text, references to 10 jpeg images)

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

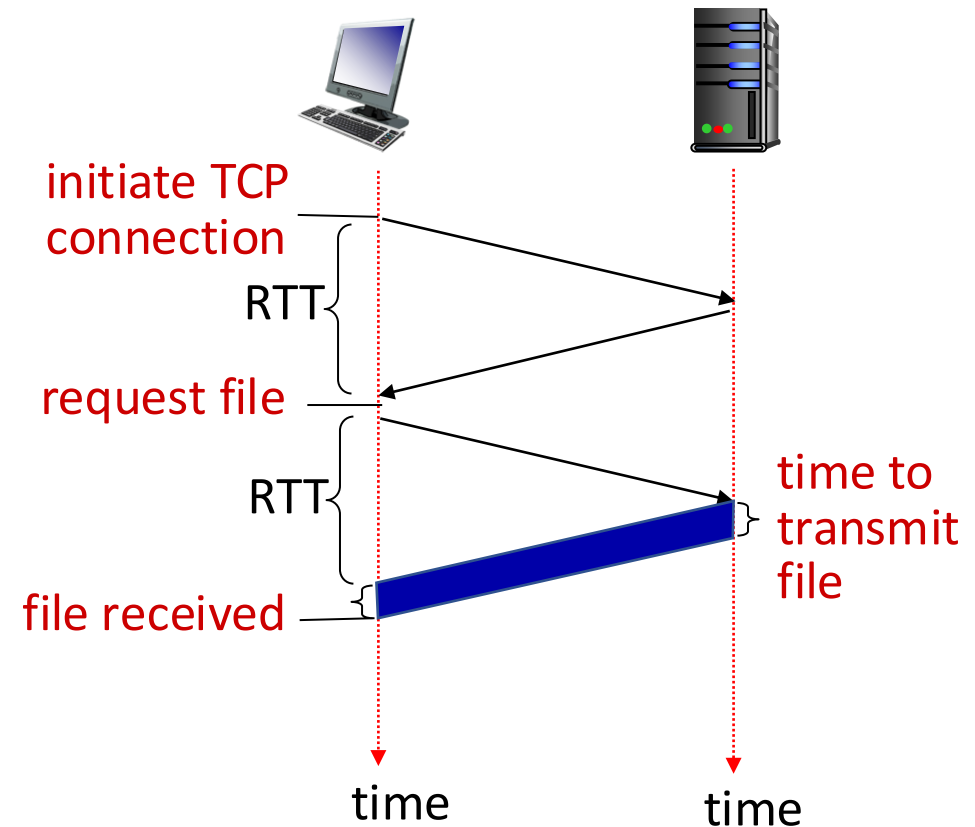
time

© CS 438 Staff, University of Illinois

Non-persistent HTTP (HTTP 1.0): response time

HTTP response time (per object):

- one RTT to initiate TCP connection
- one RTT for HTTP request and first few bytes of HTTP response to return
- object/file transmission time



Non-persistent HTTP response time = 2RTT+ file transmission time

© CS 438 Staff, University of Illinois

Persistent HTTP (HTTP 1.1)

Non-persistent HTTP:

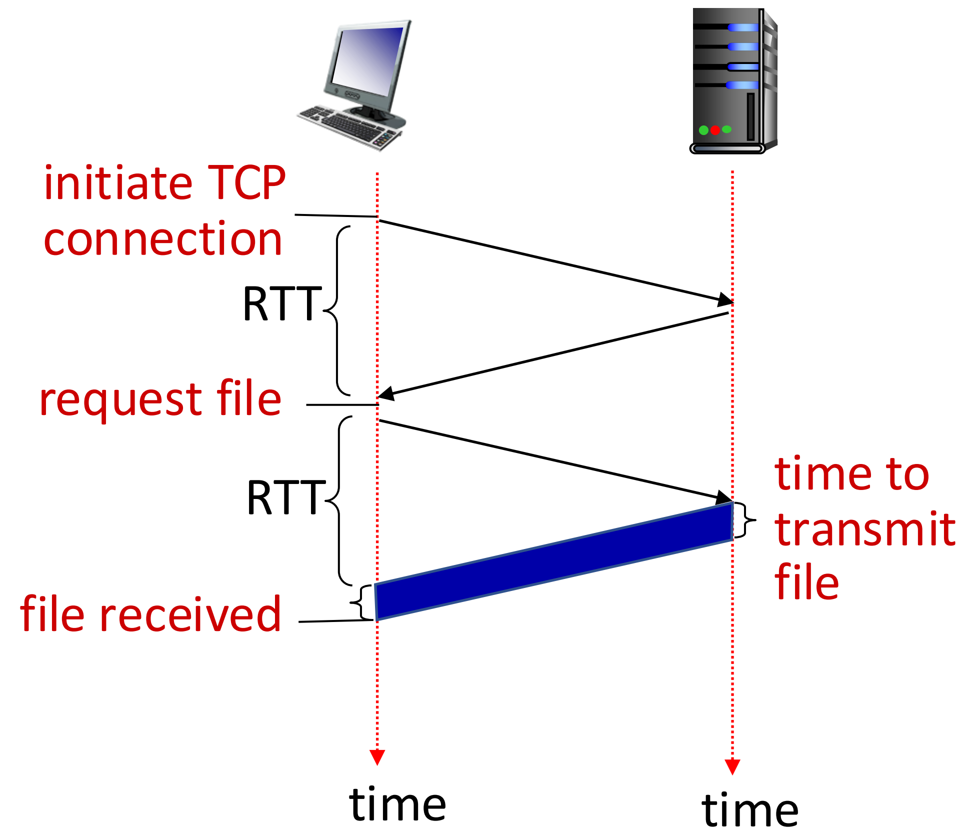
- requires 2 RTTs per object
- OS overhead for *each* TCP connection
- browsers often open multiple parallel TCP connections to fetch referenced objects in parallel

Persistent HTTP:

- server leaves connection open after sending response
- subsequent HTTP messages between same client/server sent over open connection
- requires only 1 RTT per object
- further supports *pipelining* (never took off)

© CS 438 Staff, University of Illinois

HTTP 1.0 vs. HTTP 1.1 (w/o and w/ pipelining)



Non-persistent HTTP

© CS 438 Staff, University of Illinois

04/28 Lecture

- Important update for grading of MP2: Campuswire #391
- Action required:
  - Did you simulate distributed routing protocols using a centralized approach?

MP3 Leaderboard

Last updated: 2026-04-28 12:00:05

Rules: To be eligible, teams must pass the required tests for each category. Each submission overwrites your team's previous score. Teams that don't meet eligibility are listed with a score of 0.

- **TL:** Tests 1–5, 7, and 8 must pass with full credit. Test 6 uses a loosened threshold (partial pass is sufficient). Test 9 results do not affect competition eligibility.
- **OE:** Tests 1–5 must pass with full credit. Test 6 uses a loosened threshold (partial pass is sufficient).

Deadline: The competition closes on **April 29th**. One final leaderboard update will run at midnight as April 29th rolls over into April 30th.

TCP-like (TL)

Rank	Team	Throughput (Mbps)
1 🏆	absk	10.3
2 🥈	Espressos	8.47
3 🥉	MPkiller	3.54

Open-ended (OE)

Rank	Team	Throughput (Mbps)
1 🏆	absk	9.59
2 🥈	networkrocks	5.65
3 🥉	NotVeryCreative	3.67

© CS 438 Staff, University of Illinois

HTTP 1.1: HOL blocking

HTTP 1.1: introduced multiple, pipelined GETs over single TCP connection

- server responds *in-order* (FCFS: first-come-first-served scheduling) to GET requests
- with FCFS, small object may have to wait for transmission (head-of-line (HOL) blocking) behind large object(s)

© CS 438 Staff, University of Illinois

HTTP 1.1: HOL blocking

HTTP 1.1: client requests 1 large object (e.g., video file) and 3 smaller ones

How to mitigate this?

server

client

GET O_4 GET O_3 GET O_2 GET O_1

object data requested

O_1

O_2

O_3

O_4

O_1 O_2 O_3 O_4

objects delivered in order requested: O_2, O_3, O_4 wait behind O_1

© CS 438 Staff, University of Illinois

HTTP/2

Key goal: decreased delay in multi-object HTTP requests

HTTP/2: [RFC 7540, 2015] increased flexibility at *server* in sending objects to client:

- methods, status codes, most header fields unchanged from HTTP 1.1
- divide objects into frames, schedule frames to mitigate HOL blocking
  - “stream ID” in frame header: uniquely identifies a request/response pair
- transmission order of requested objects based on object dependency and priority (not necessarily FCFS)
- *push* unrequested objects to client (rarely used today)

© CS 438 Staff, University of Illinois

Multiplexing in HTTP2

Client

TCP Connection

Server

Request 1
GET /styles.css

Request 2
GET /script.js

Request 3
GET /image.jpg

Binary Framing Layer

REQUESTS

Stream 5
GET /image.jpg

Stream 3
GET /script.js

Stream 1
GET /styles.css

RESPONSES

Stream 1 header	Stream 3 header	Stream 1 body	Stream 3 body	Stream 5 header	Stream 5 body

Binary Framing Layer

Request 1
GET /styles.css

Request 2
GET /script.js

Request 3
GET /image.jpg

Image source: blog post

© CS 438 Staff, University of Illinois

HTTP/2: stream multiplexing

HTTP/2: objects divided into frames, frame transmission interleaved

No HOL blocking now, right?

server

client

GET O_4 GET O_3 GET O_2 GET O_1

object data requested

O_1

O_2

O_3

O_4

O_2

O_4

O_3

O_1

O_2, O_3, O_4 delivered quickly, O_1 slightly delayed

© CS 438 Staff, University of Illinois

HTTP/2 to HTTP/3

Key goal: decreased delay in multi-object HTTP requests

HTTP/2 over single TCP connection means:

- recovery from packet loss still stalls all object transmissions
  - as in HTTP 1.1, browsers have incentive to open multiple parallel TCP connections to reduce stalling, increase overall throughput
- HTTP/3: QUIC over UDP

© CS 438 Staff, University of Illinois

Context: Evolving transport-layer functionality

- Different "flavors" of TCP developed, for specific scenarios:

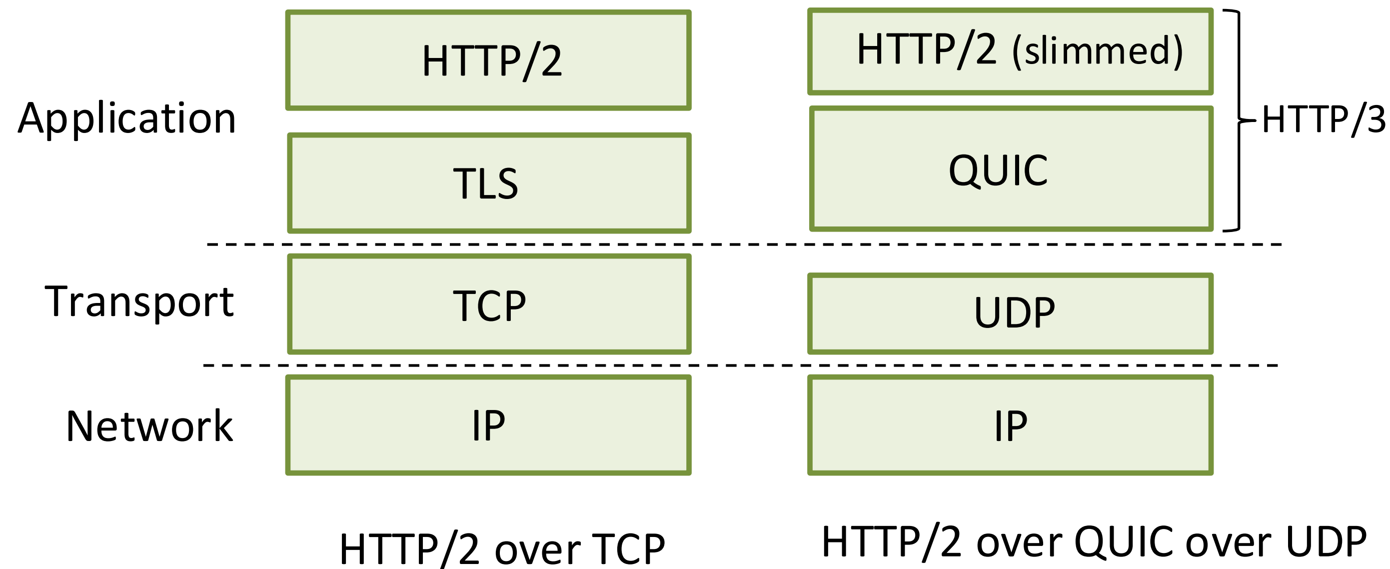
Scenario	Challenges
Long, fat pipes (large data transfers)	Many packets "in flight"; loss shuts down pipeline
Wireless networks	Loss due to noisy wireless links, mobility; TCP treat this as congestion loss
Long-delay links	Extremely long RTTs
Data center networks	Latency sensitive
Background traffic flows	Low priority, "background" TCP flows

- moving transport-layer functions to application layer, on top of UDP
- implication: TCP (kernel space) → UDP (user space)
  - \+ faster evolution without kernel constraints
  - − context switch overhead

© CS 438 Staff, University of Illinois

QUIC: Quick UDP Internet Connections

- application-layer protocol, on top of UDP
  - increase performance of HTTP
  - deployed on many Google servers, apps (Chrome, mobile YouTube app)



© CS 438 Staff, University of Illinois

QUIC: Quick UDP Internet Connections

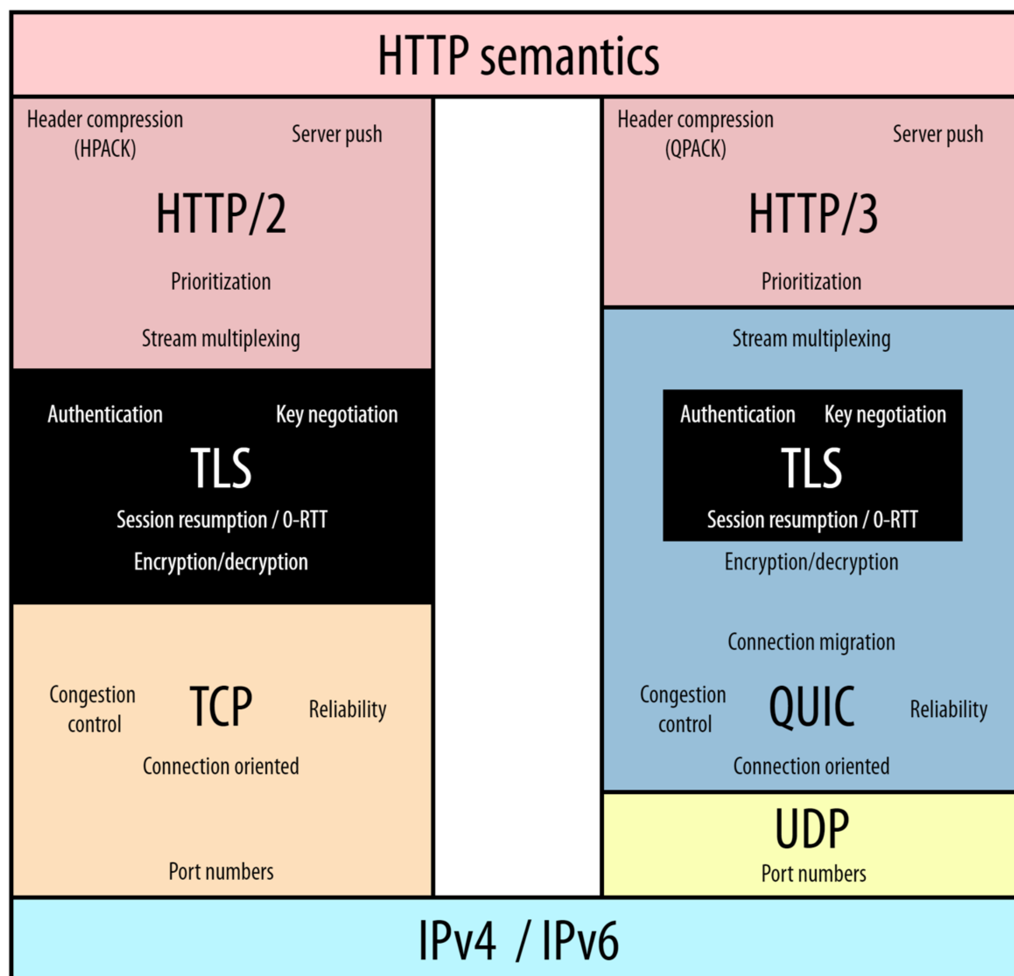


Image source: blog post

- Same HTTP methods, status codes, ...
- Connection-oriented
  - explicit connection identifier (CID)
  - unique packet numbers (even for retransmissions)
- Deeply integrates with TLS
  - connection establishment in one RTT
- Independent streams multiplexed over single QUIC connection
- Similar reliability and congestion control implemented over UDP

© CS 438 Staff, University of Illinois

QUIC: Deep Integration with TLS

Encrypted

HTTP data

TLS record header

TLS mac

TLS payload

TCP packet header

Source port	Destination port	Sequence number	ACK number	Receive window	length + flags	chksum + urgent	Options	TCP payload

Encrypted

Encrypted

HTTP data

QUIC packet header

ACK frame	Flow control frame	Stream frame header	Stream frame payload

QUIC mac

Flags	Connection ID	Packet number	QUIC payload

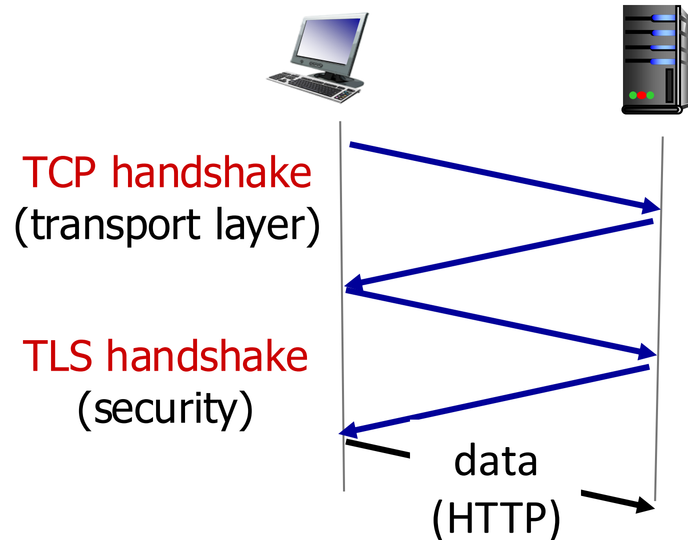
UDP packet header

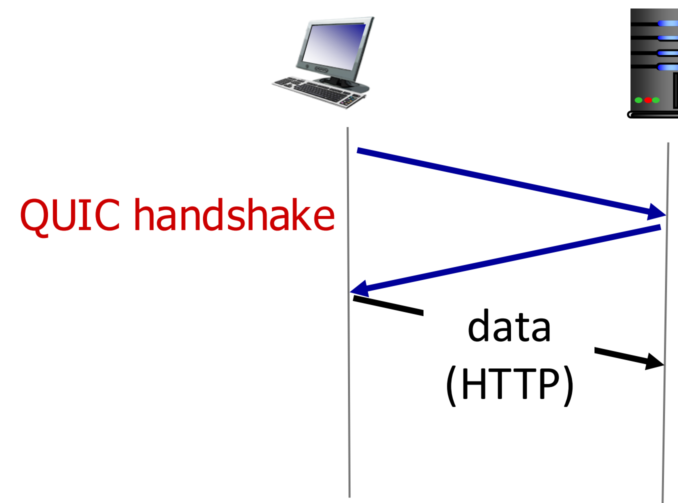
Source port	Destination port	chksum + length	UDP payload

Image source: blog post

© CS 438 Staff, University of Illinois

QUIC: Connection establishment





TCP (reliability, congestion control state)
+ TLS (authentication, crypto state)

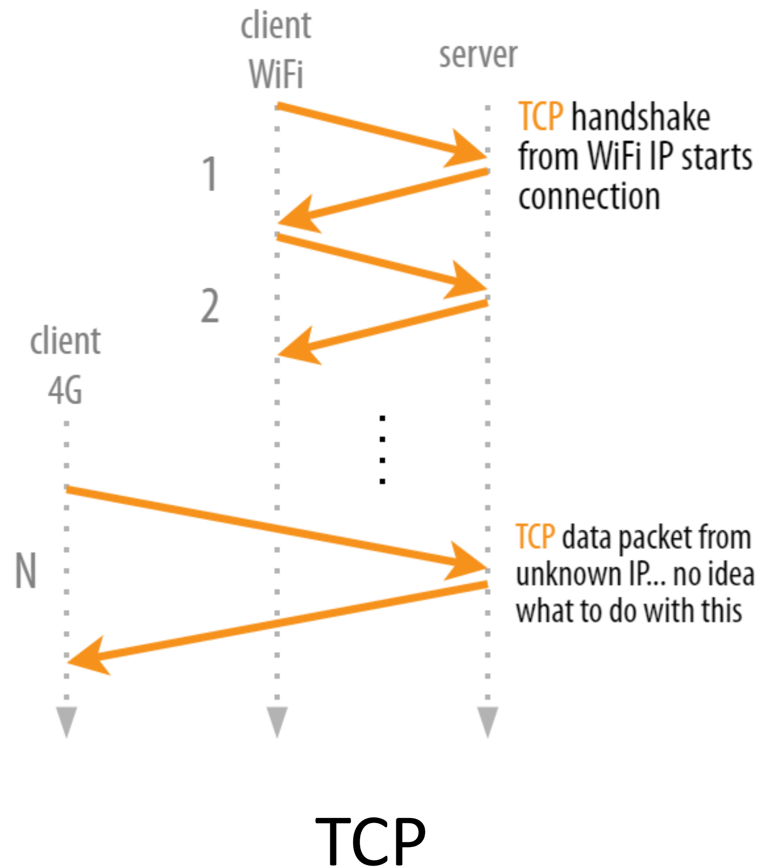
- 2 serial handshakes

QUIC: reliability, congestion control, authentication, crypto state

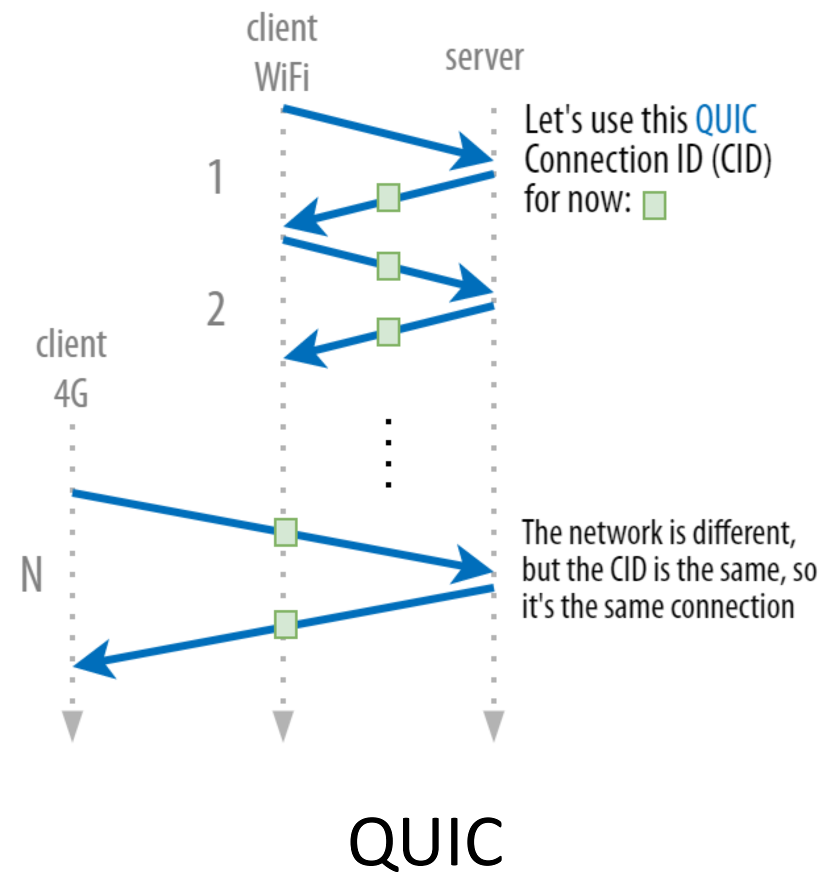
- 1 handshake

© CS 438 Staff, University of Illinois

QUIC: Connection migration



TCP



QUIC

Image source: blog post

© CS 438 Staff, University of Illinois

QUIC streams: parallelism, no HOL blocking

application

transport

HTTP GET

HTTP GET

HTTP GET

TLS encryption

TLS encryption

TCP RDT

error RDT

TCP Cong. Contr.

TCP Cong. Contr.

HTTP GET

HTTP GET

HTTP GET

QUIC encrypt

QUIC encrypt

QUIC encrypt

QUIC RDT

QUIC RDT

QUIC RDT

QUIC Cong. Cont.

UDP

QUIC encrypt

QUIC encrypt

QUIC encrypt

QUIC RDT

error!

QUIC RDT

QUIC Cong. Cont.

UDP

(a) HTTP/2: transport-layer HOL blocking

(b) HTTP/3: no HOL blocking

© CS 438 Staff, University of Illinois

Learning Objectives

- Application-layer basics
- DNS
- HTTP
- **More on DNS and HTTP**

© CS 438 Staff, University of Illinois

Inserting Resource Records into DNS

- Example: just created startup "FooBar"
- Get a block of address space from ISP
  - E.g., `212.44.9.128/25`
- Register foobar.com at a domain name registrar (e.g., Cloudflare)
  - Provide registrar with names and IP addresses of your authoritative name server (primary and secondary)
  - Registrar inserts RR pairs into the `com` TLD server:
    - `(foobar.com, dns1.foobar.com, NS)`
    - `(dns1.foobar.com, 212.44.9.129, A)`
- Put in your (authoritative) server `dns1.foobar.com`:
  - Type A record for `www.foobar.com`
  - Type MX record for `foobar.com`

© CS 438 Staff, University of Illinois

Setting up foobar.com

- In addition, need to provide reverse PTR bindings
  - e.g., `212.44.9.129` → `dns1.foobar.com`
- Normally, these go in `9.44.212.in-addr.arpa`
- Problem
  - You can't run the name server for that domain. Why not?
  - Because your block is `212.44.9.128/25`, not `212.44.9.0/24`
  - Whoever has `212.44.9.0/25` won't be happy with you owning their PTR records
- Solution: ISP runs it for you
  - Now it's more of a headache to keep it up-to-date :-(

© CS 438 Staff, University of Illinois

DNS Summary

- Domain Name System (DNS)
  - Distributed, hierarchical database
  - Distributed collection of servers
  - Caching to improve performance

- DNS currently lacks authentication (to be covered later)
  - Can’t tell if reply comes from the correct source
  - Can’t tell if correct source tells the truth
  - Malicious source can insert extra (mis)information
  - Malicious bystander can spoof (mis)information
  - Playing with caching lifetimes adds extra power to attacks

© CS 438 Staff, University of Illinois

Do you want cookies?

We value your privacy

We use cookies to enhance your browsing experience, serve personalized ads or content, and analyze our traffic. By clicking "Accept All", you consent to our use of cookies. Cookie Policy

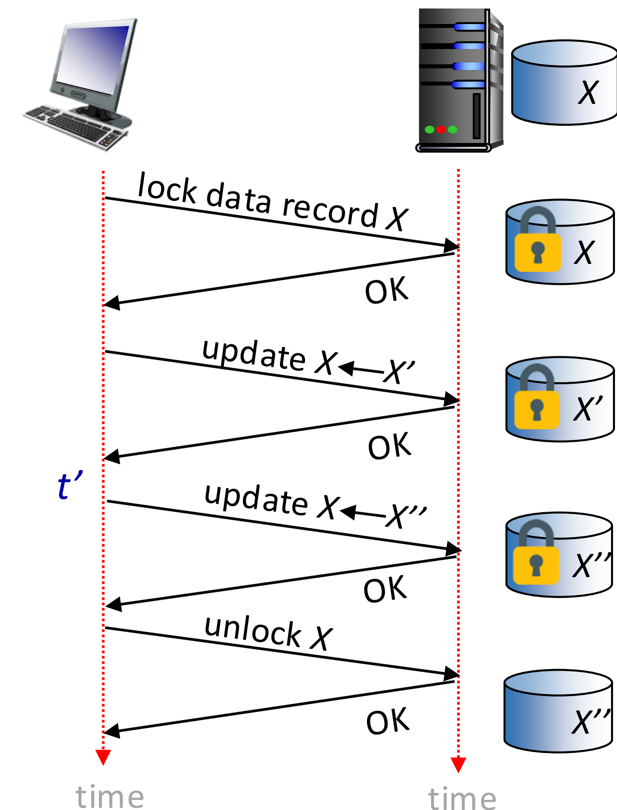
Customize | Reject All | Accept All

HTTP: Maintaining user/server state with cookies

Recall: HTTP GET/response interaction is *stateless*

- no notion of multi-step exchanges of HTTP messages to complete a Web "transaction"
  - no need for client/server to track "state" of multi-step exchange
  - all HTTP requests are independent of each other
  - no need for client/server to "recover" from a partially-completed-but-never-completely-completed transaction

a stateful protocol: client makes two changes to X, or none at all



Q: what happens if network connection or client crashes at *t'* ?

© CS 438 Staff, University of Illinois

Maintaining user/server state: cookies

Web sites and client browser use *cookies* to maintain some state between transactions

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

Example:

- Susan uses browser on laptop, visits specific e-commerce site for first time
- when initial HTTP requests arrives at site, site creates:
  - unique ID (aka "cookie")
  - entry in backend database for ID
- subsequent HTTP requests from Susan to this site will contain cookie ID value, allowing site to "identify" Susan

© CS 438 Staff, University of Illinois

Maintaining user/server state: cookies

client

server

ebay 8734

cookie file

usual HTTP request msg

Amazon server creates ID 1678 for user

usual HTTP response
set-cookie: 1678

ebay 8734
amazon 1678

create entry

backend database

usual HTTP request msg
cookie: 1678

cookie-specific action

access

usual HTTP response msg

one week later:

ebay 8734
amazon 1678

access

usual HTTP request msg
cookie: 1678

cookie-specific action

usual HTTP response msg

time

time

© CS 438 Staff, University of Illinois

HTTP cookies: comments

What 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
- vs. cookies: HTTP messages carry state

cookies and privacy:

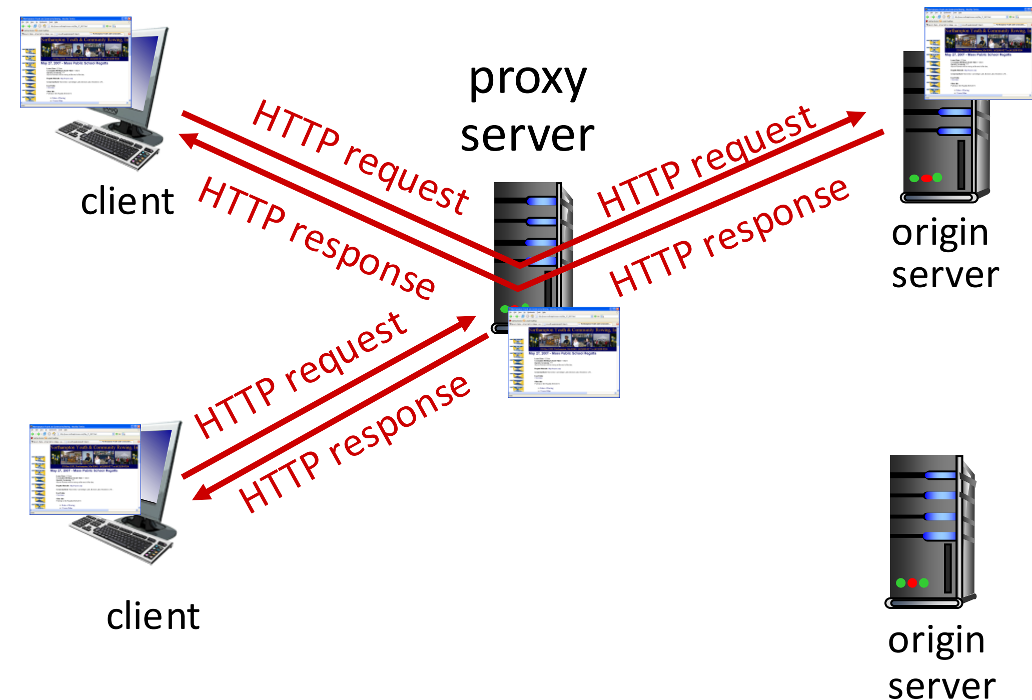
- cookies permit sites to *learn* a lot about you on their site.
- third party persistent cookies (tracking cookies) allow common identity (cookie value) to be tracked across multiple web sites

© CS 438 Staff, University of Illinois

Web caches (proxy servers)

Goal: satisfy client request without involving origin server

- user configures browser to point to a *Web cache*
- browser sends all HTTP requests to cache
  - *if* object in cache: cache returns object to client
  - *else* cache requests object from origin server, caches received object, then returns object to client



© CS 438 Staff, University of Illinois

Web caches (proxy servers)

- Web cache acts as both client and server
  - server for original requesting client
  - client to origin server
- typically cache is installed by ISP (university, company, residential ISP)

Why Web caching?

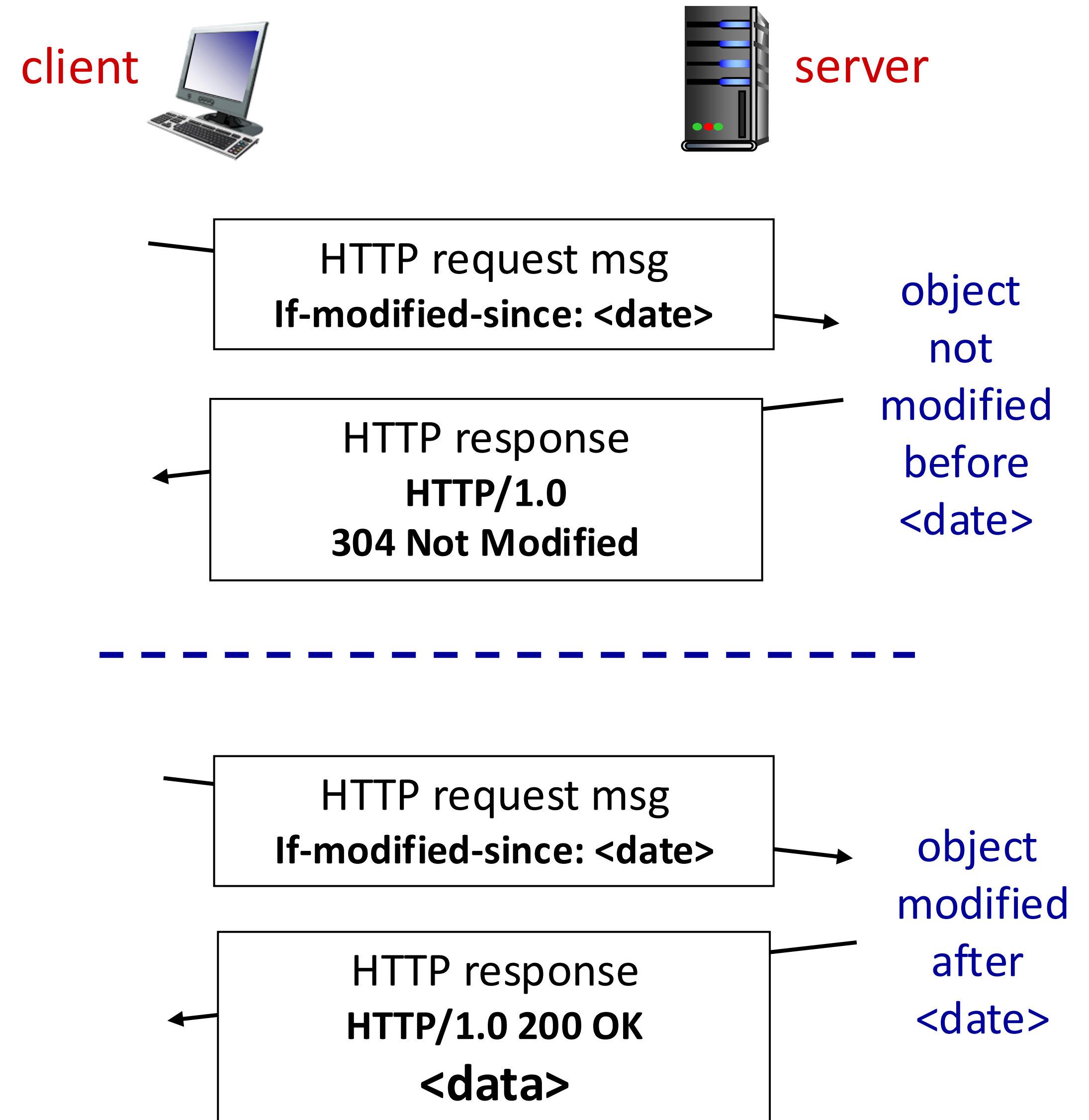
- reduce response time for client request
  - cache is closer to client
- reduce traffic on an institution's access link
- Internet is dense with caches
  - enables "poor" content providers to more effectively deliver content

© CS 438 Staff, University of Illinois

Conditional GET

Goal: don't send object if cache has up-to-date cached version
- no object transmission delay
- lower link utilization

- *cache:* specify date of cached copy in HTTP request
  If-modified-since: <date>
- *server:* response contains no object if cached copy is up-to-date:
  HTTP/1.0 304 Not Modified



© CS 438 Staff, University of Illinois