Computer Networks and Applications

COMP 3331/COMP 9331

Week 1

Application Layer (Principles, Web)

Chapter 2, Sections 2.1 – 2.2

2. Application Layer: outline

2.1 principles of network applications

2.2 Web and HTTP

2.3 electronic mail

- SMTP, POP3, IMAP

2.4 DNS

2.5 P2P applications

2.6 video streaming and content distribution networks (CDNs)

2.7 socket programming with UDP and TCP

2. Application layer

our goals:

- conceptual, implementation aspects of network application protocols
  - transport-layer service models
  - client-server paradigm
  - peer-to-peer paradigm
- learn about protocols by examining popular application-level protocols
  - HTTP
  - SMTP / POP3 / IMAP
  - DNS
- creating network applications
  - socket API

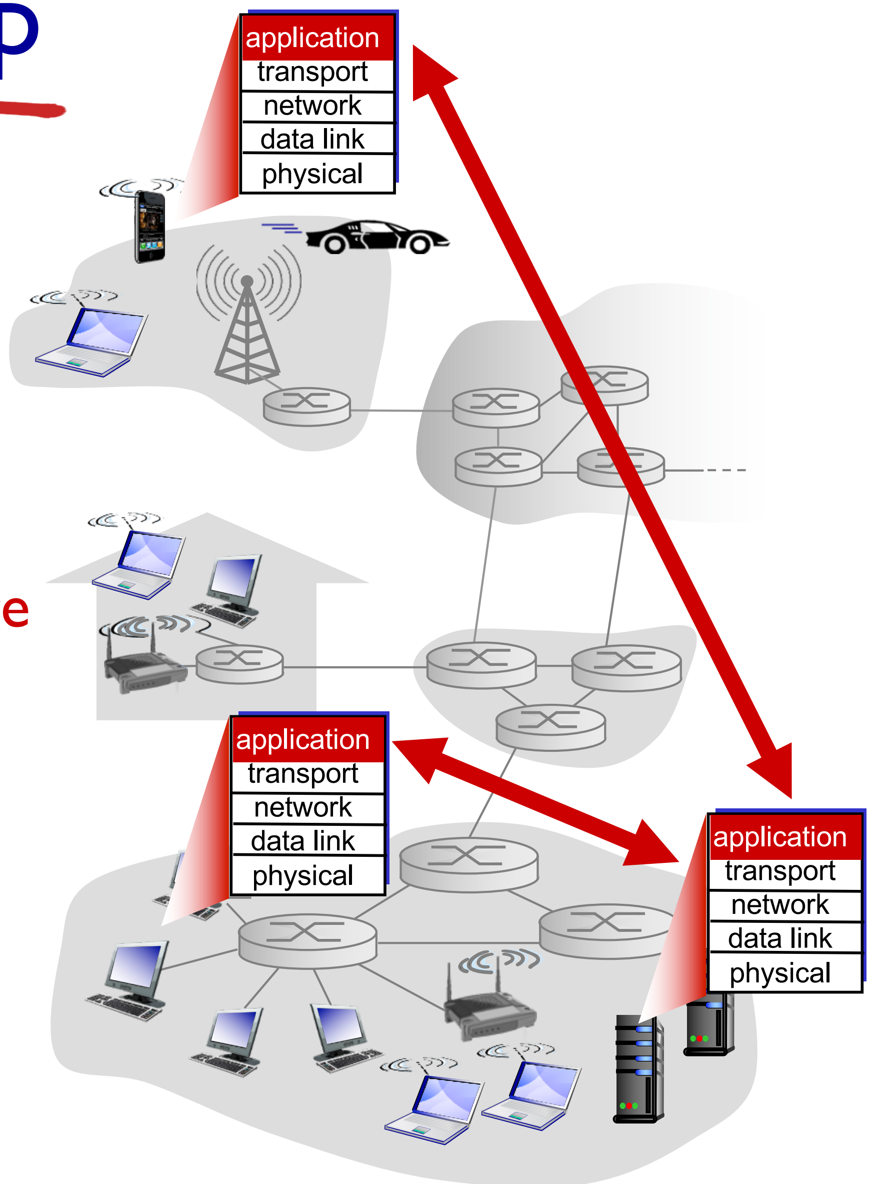
Creating a network app

Write programs that:

- run on (different) *end systems*
- communicate over network
- e.g., web server software communicates with browser software

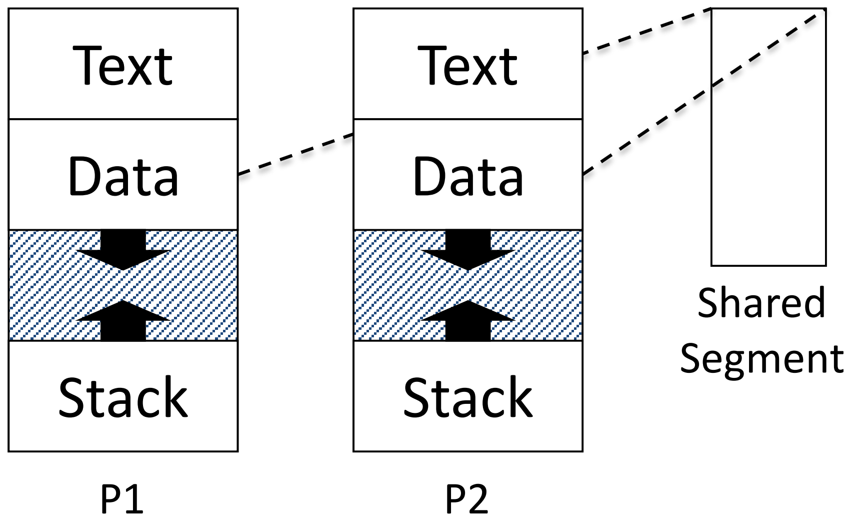
No need to write software for network-core devices

- network-core devices do not run user applications
- applications on end systems allows for rapid app development



Interprocess Communication (IPC)

- Processes talk to each other through Inter-process communication (IPC)

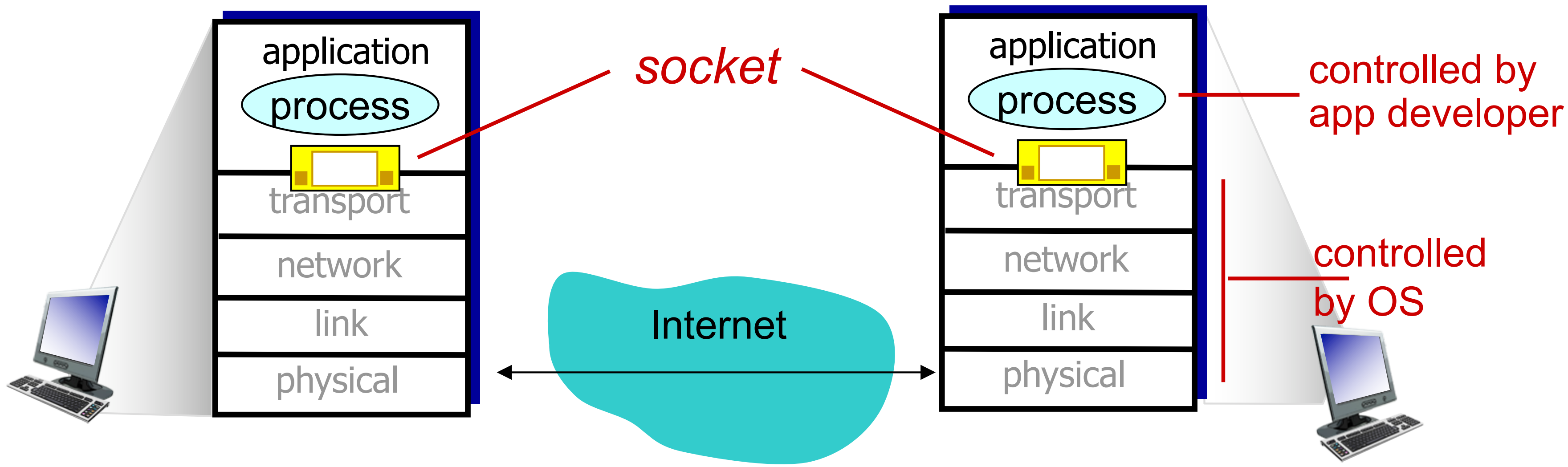


- On a single machine:
  - Shared memory

- Across machines:
  - We need other abstractions (message passing)

Sockets

- process sends/receives messages to/from its socket
- socket analogous to door
  - sending process shoves message out door
  - sending process relies on transport infrastructure on other side of door to deliver message to socket at receiving process
- Application has a few options, OS handles the details

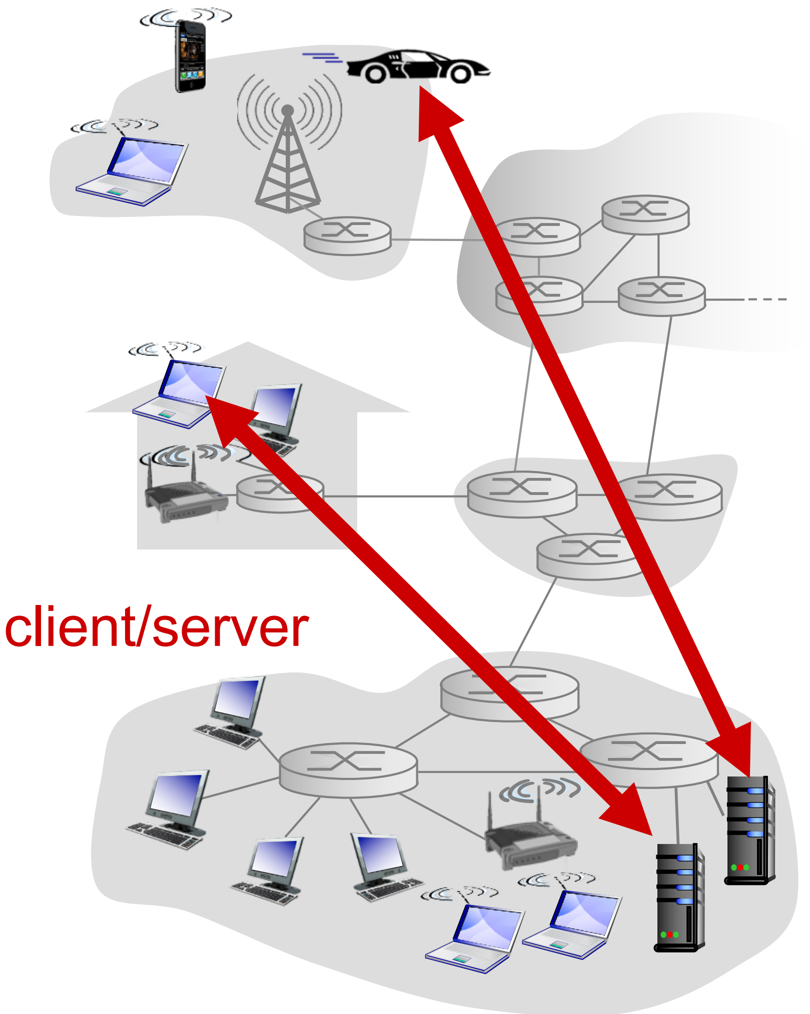


Addressing processes

- to receive messages, process must have *identifier*
- host device has unique 32-bit IP address
- Q: does IP address of host on which process runs suffice for identifying the process?
  - A: no, *many* processes can be running on same host

- *identifier* includes both IP address and port numbers associated with process on host.
- example port numbers:
  - HTTP server: 80
  - mail server: 25
- to send HTTP message to cse.unsw.edu.au web server:
  - IP address: 129.94.242.51
  - port number: 80

Client-server architecture



server:

- Exports well-defined request/response interface
- long-lived process that waits for requests
- Upon receiving request, carries it out

clients:

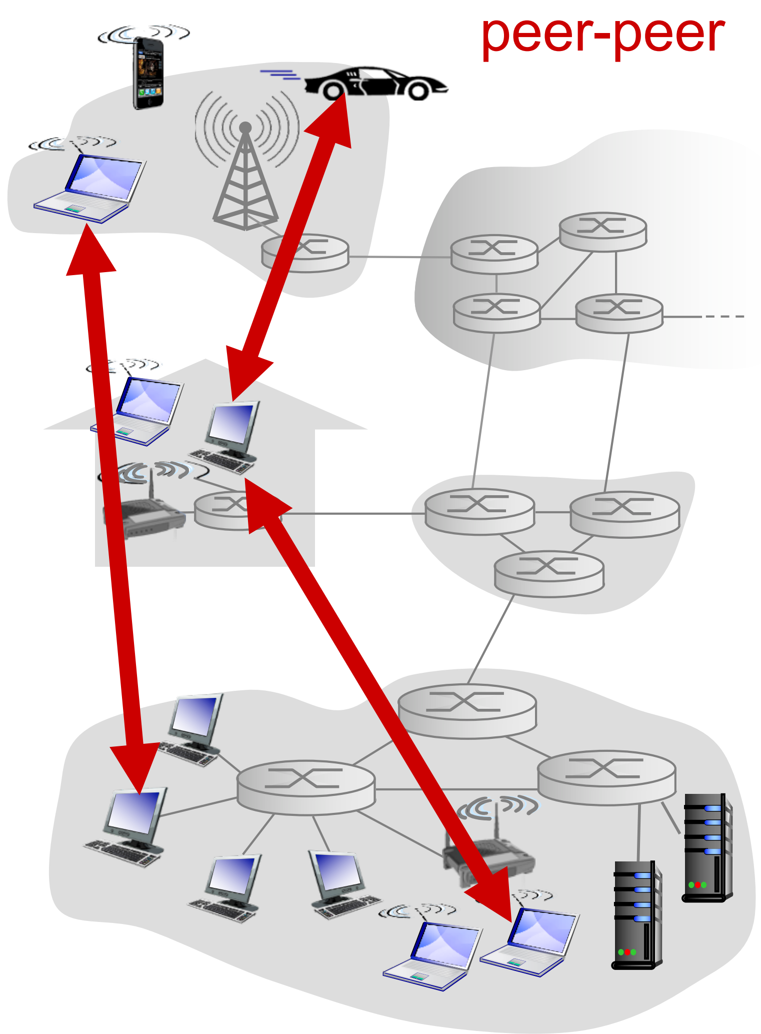
- Short-lived process that makes requests
- "User-side" of application
- Initiates the communication

Client versus Server

- Server
  - Always-on host
  - Permanent IP address (rendezvous location)
  - Static port conventions (http: 80, email: 25, ssh:22)
  - Data centres for scaling
  - May communicate with other servers to respond
- Client
  - May be intermittently connected
  - May have dynamic IP addresses
  - Do not communicate directly with each other

P2P architecture

- *no* always-on server
  - No permanent rendezvous involved
- arbitrary end systems (peers) directly communicate
- Symmetric responsibility (unlike client/server)
- Often used for:
  - File sharing (BitTorrent)
  - Games
  - Video distribution, video chat
  - In general: "distributed systems"

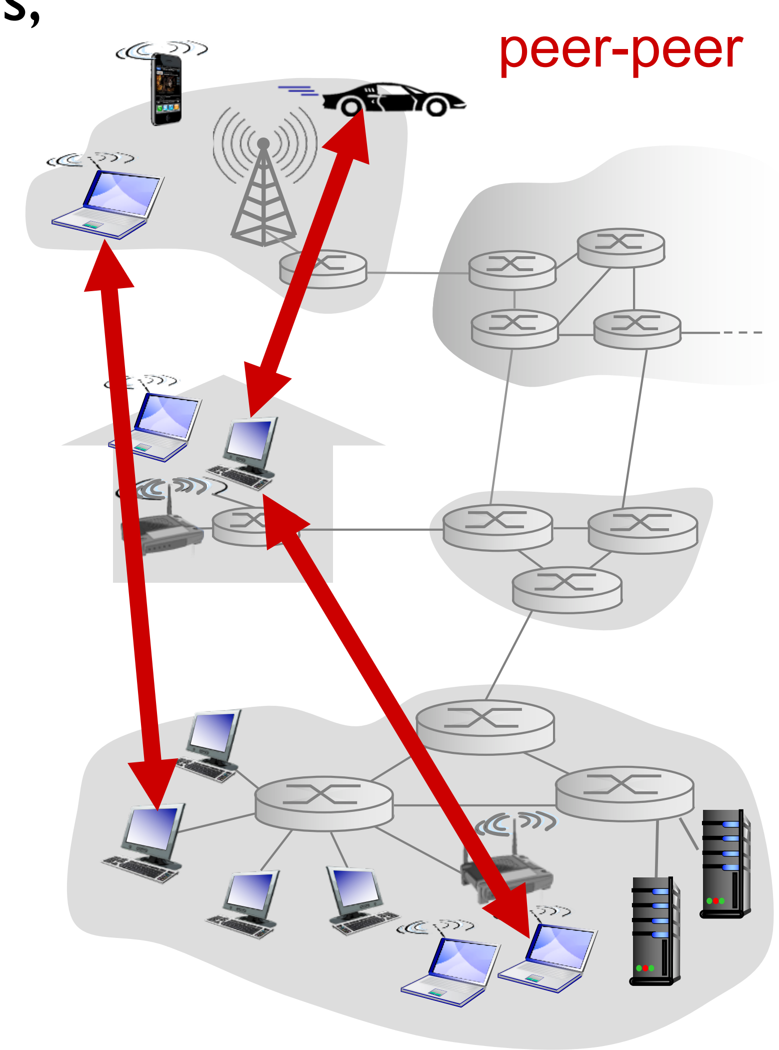


P2P architecture: Pros and Cons

+ 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
+ Speed: parallelism, less contention
+ Reliability: redundancy, fault tolerance
+ Geographic distribution

-Fundamental problems of decentralized control
    - State uncertainty: no shared memory or clock
    - Action uncertainty: mutually conflicting decisions

-Distributed algorithms are complex



App-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
- allows for interoperability
- e.g., HTTP, SMTP

proprietary protocols:

- e.g., Skype

What transport service does an app need?

data integrity

- some apps (e.g., file transfer, web transactions) require 100% reliable data transfer
- other apps (e.g., audio) can tolerate some loss

timing

- some apps (e.g., Internet telephony, interactive games) require low delay to be "effective"

throughput

- some apps (e.g., multimedia) require minimum amount of throughput to be "effective"
- other apps ("elastic apps") make use of whatever throughput they get

security

- encryption, data integrity, …

Transport service 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: 50kbps-1Mbps video:100kbps-5Mbps	yes, 100's msec
stored audio/video	loss-tolerant	same as above	yes, few secs
interactive games	loss-tolerant	few kbps up	yes, 100's msec
Chat/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, orconnection setup,

Q: why bother? Why is there a UDP?

NOTE: More on transport later on

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

2. Application Layer: outline

2.1 principles of network applications
- app architectures
- app requirements

2.2 Web and HTTP

2.3 electronic mail
- SMTP, POP3, IMAP

2.4 DNS

2.5 P2P applications

2.6 video streaming and content distribution networks (CDNs)

2.7 socket programming with UDP and TCP

The Web – Precursor

Self study

Ted Nelson

- **1967**, Ted Nelson, Xanadu:
  - A world-wide publishing network that would allow information to be stored not as separate files but as connected literature
  - Owners of documents would be automatically paid via electronic means for the virtual copying of their documents
- Coined the term "Hypertext"

The Web – History

Tim Berners-Lee

- World Wide Web (WWW): a distributed database of "pages" linked through Hypertext Transport Protocol (HTTP)
  - First HTTP implementation - 1990
    - Tim Berners-Lee at CERN
  - HTTP/0.9 – 1991
    - Simple GET command for the Web
  - HTTP/1.0 –1992
    - Client/Server information, simple caching
  - HTTP/1.1 – 1996
  - HTTP2.0 - 2015

http://info.cern.ch/hypertext/WWW/TheProject.html

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 *URL,* e.g.,

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

Uniform Resource Locator (URL)

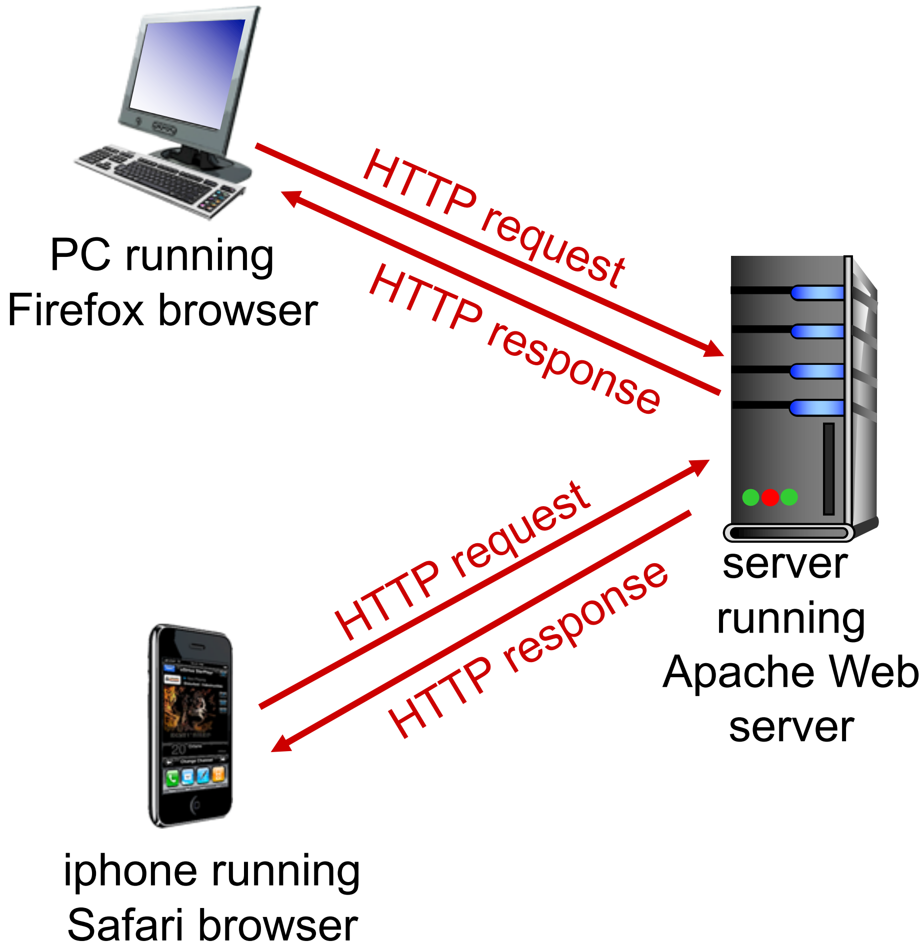
`protocol://host-name[:port]/directory-path/resource`

- *protocol*: http, ftp, https, smtp *etc.*
- *hostname*: DNS name, IP address
- *port:* defaults to protocol's standard port; *e.g.* http: 80 https: 443
- *directory path*: hierarchical, reflecting file system
- *resource*: Identifies the desired resource

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



HTTP overview (continued)

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

HTTP request message

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

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
```

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 ...
```

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`

`451 Unavailable for Legal Reasons`

`429 Too Many Requests`

`418 I'm a Teapot`

HTTP is all text

- Makes the protocol simple
  - Easy to delineate messages (\r\n)
  - (relatively) human-readable
  - No issues about encoding or formatting data
  - Variable length data
- Not the most efficient
  - Many protocols use binary fields
    - *Sending "12345678" as a string is 8 bytes*
    - *As an integer, 12345678 needs only 4 bytes*
  - Headers may come in any order
  - Requires string parsing/processing

Request Method types ("verbs")

HTTP/1.0:

- GET
  - Request page
- POST
  - Uploads user response to a form
- HEAD
  - asks server to leave requested object out of response

HTTP/1.1:

- GET, POST, HEAD
- PUT
  - uploads file in entity body to path specified in URL field
- DELETE
  - deletes file specified in the URL field
- TRACE, OPTIONS, CONNECT, PATCH
  - For persistent connections

Uploading form input

POST method:

- web page often includes form input
- input is uploaded to server in entity body

Get (in-URL) method:

- uses GET method
- input is uploaded in URL field of request line:

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

User-server state: cookies

many Web sites use cookies

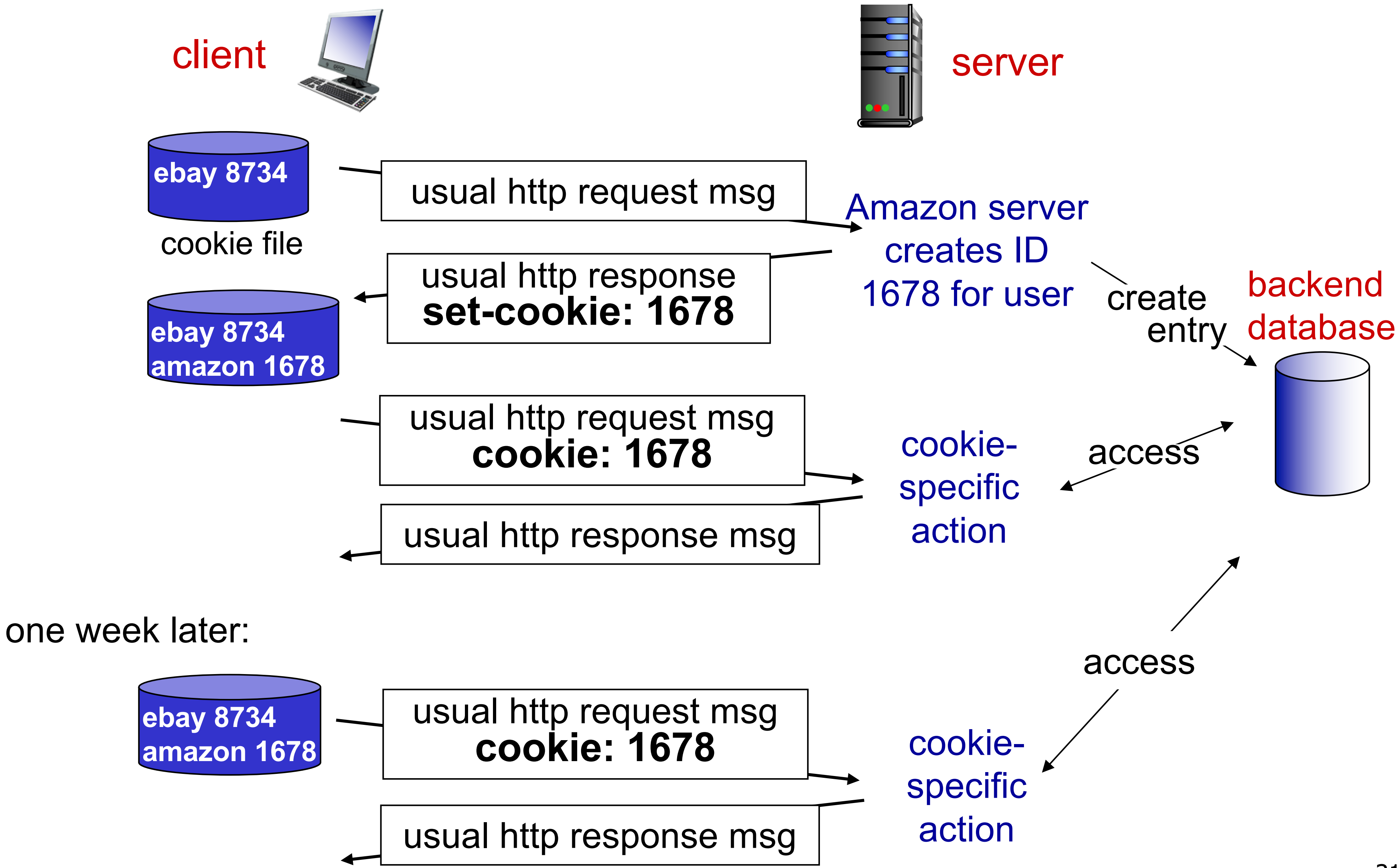
four components:

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 always access Internet from PC
- visits specific e-commerce site for first time
- when initial HTTP requests arrives at site, site creates:
  - unique ID
  - entry in backend database for ID

Cookies: keeping "state" (cont.)



The Dark Side of Cookies

- Cookies permit sites to learn a lot about you

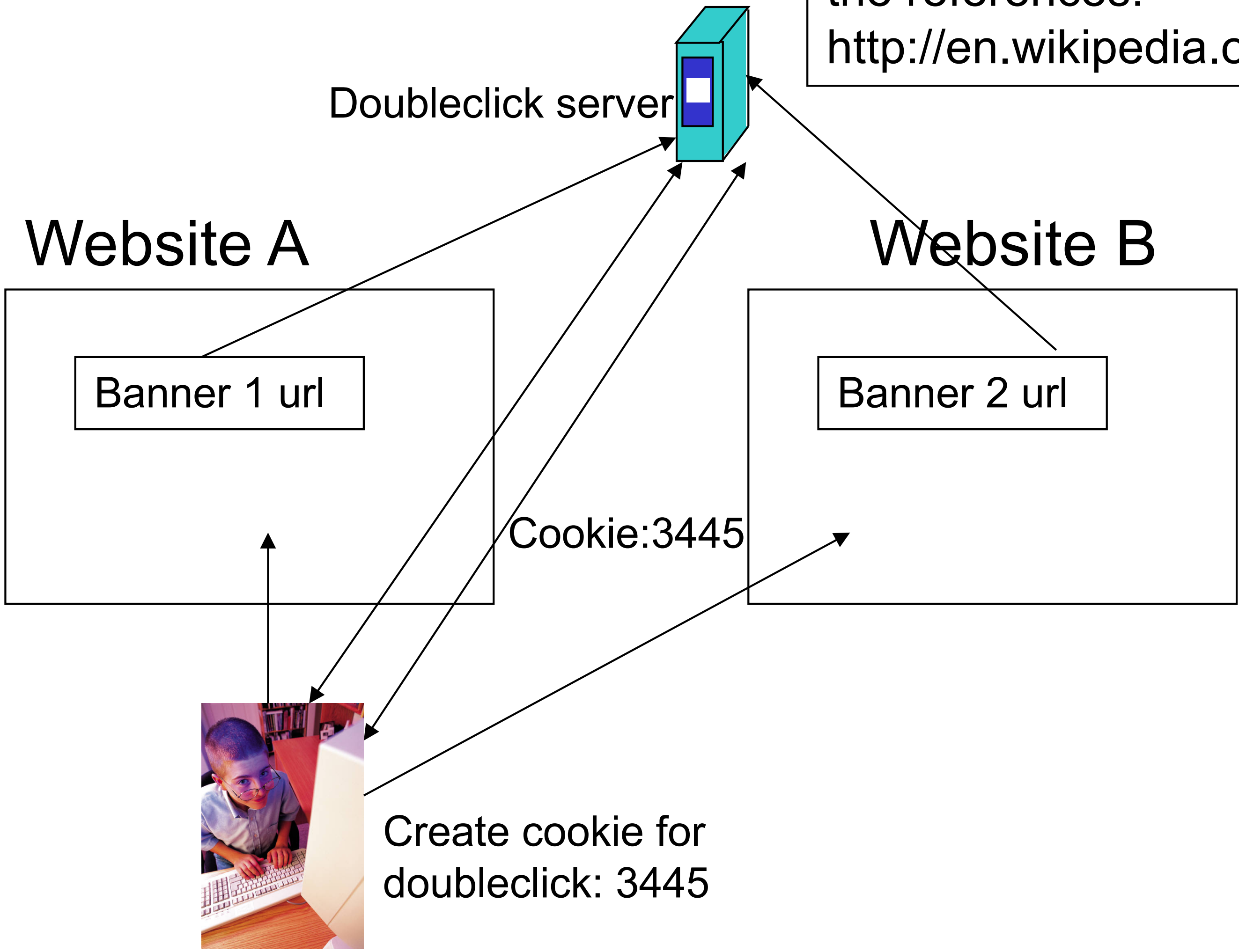
- You may supply name and e-mail to sites (and more)

- 3rd party cookies (from ad networks, etc.) can follow you across multiple sites
  - Ever visit a website, and the next day ALL your ads are from them ?
    - Check your browser's cookie file (cookies.txt, cookies.plist)
    - Do you see a website that you have never visited

- You COULD turn them off
  - But good luck doing anything on the Internet !!

Third party cookies

For more, check the following link and follow the references:
http://en.wikipedia.org/wiki/HTTP_cookie



Performance of HTTP

- Page Load Time (PLT) as the metric
  - From click until user sees page
  - Key measure of web performance
- Depends on many factors such as
  - page content/structure,
  - protocols involved and
  - Network bandwidth and RTT

Performance Goals

- User
  - fast downloads
  - high availability

- Content provider
  - happy users (hence, above)
  - cost-effective infrastructure

- Network (secondary)
  - avoid overload

Solutions?

- User
  - fast downloads
  - high availability

- Content provider
  - happy users (hence, above)
  - cost-effective infrastructure

- Network (secondary)
  - avoid overload

Improve HTTP to achieve faster downloads

Solutions?

- User
  - fast downloads
  - high availability
- Content provider
  - happy users (hence, above)
  - cost-effective delivery infrastructure
- Network (secondary)
  - avoid overload

Improve HTTP to achieve faster downloads

Caching and Replication

Solutions?

- User
  - fast downloads
  - high availability

Improve HTTP to achieve faster downloads

- Content provider
  - happy users (hence, above)
  - cost-effective delivery infrastructure

Caching and Replication

- Network (secondary)
  - avoid overload

Exploit economies of scale (Webhosting, CDNs, datacenters)

How to improve PLT

- Reduce content size for transfer
  - Smaller images, compression
- Change HTTP to make better use of available bandwidth
  - Persistent connections and pipelining
- Change HTTP to avoid repeated transfers of the same content
  - Caching and web-proxies
- Move content closer to the client
  - CDNs

HTTP Performance

- Most Web pages have multiple objects
  - *e.g.*, HTML file and a bunch of embedded images
- How do you retrieve those objects (naively)?
  - *One item at a time*
- New TCP connection per (small) object!

non-persistent HTTP

- at most one object sent over TCP connection
  - connection then closed
- downloading multiple objects required multiple connections

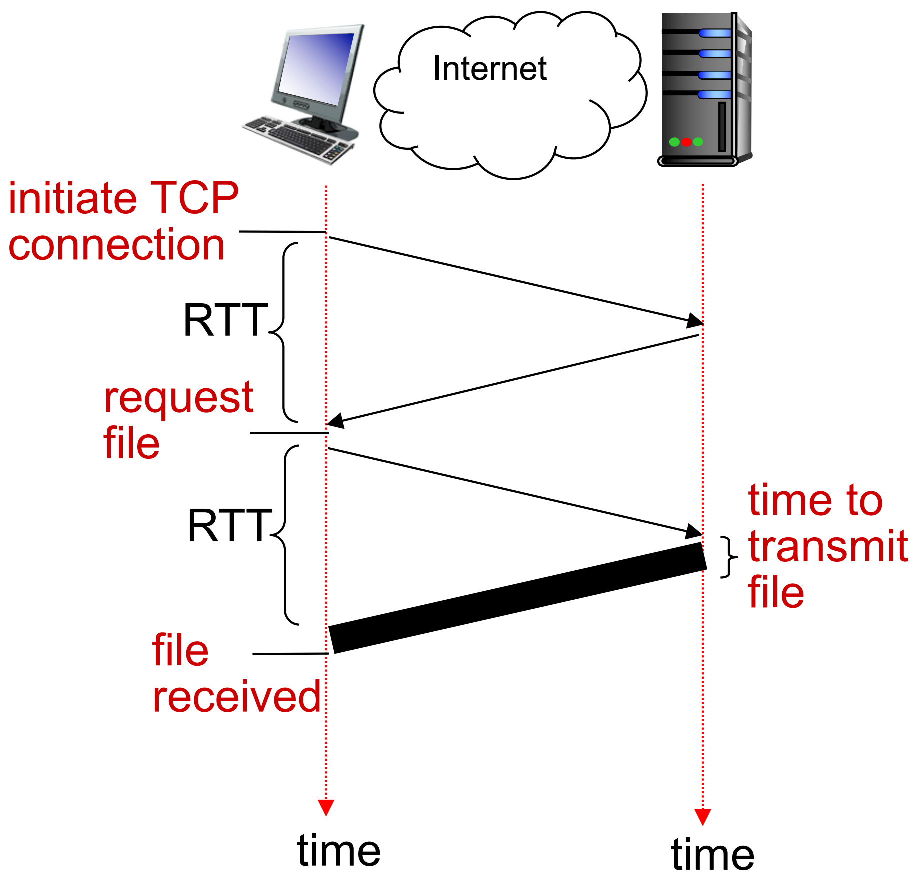
Non-persistent HTTP: response time

RTT (definition): time for a small packet to travel from client to server and back

HTTP response time:

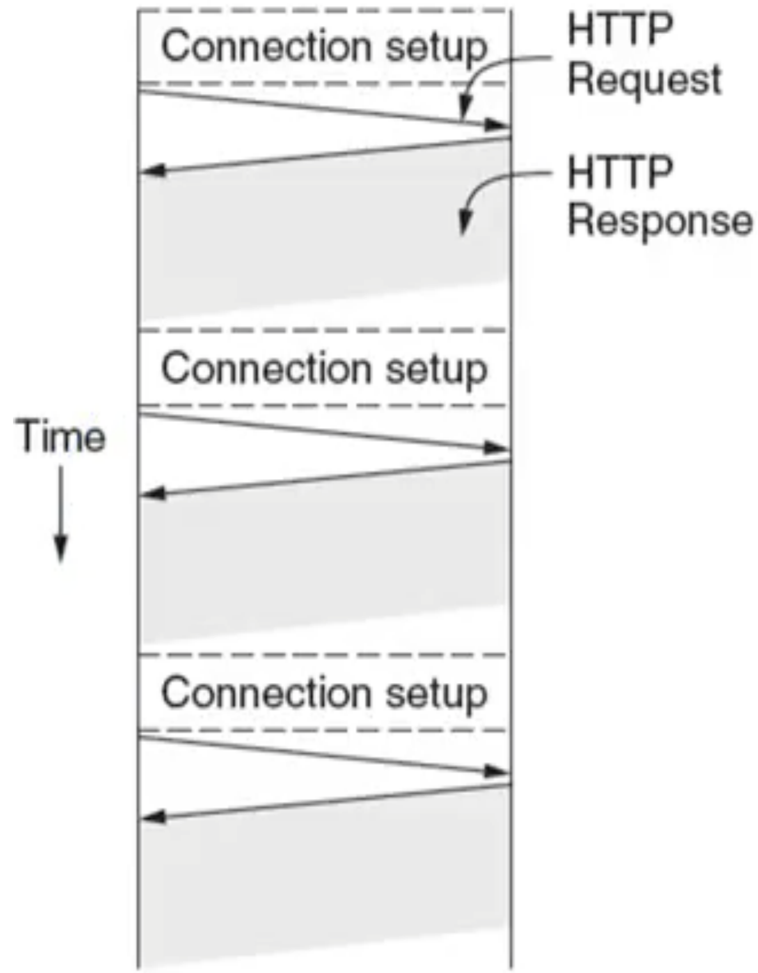
- one RTT to initiate TCP connection
- one RTT for HTTP request and first few bytes of HTTP response to return
- file transmission time
- non-persistent HTTP response time =

  2RTT+ file transmission time



HTTP/1.0

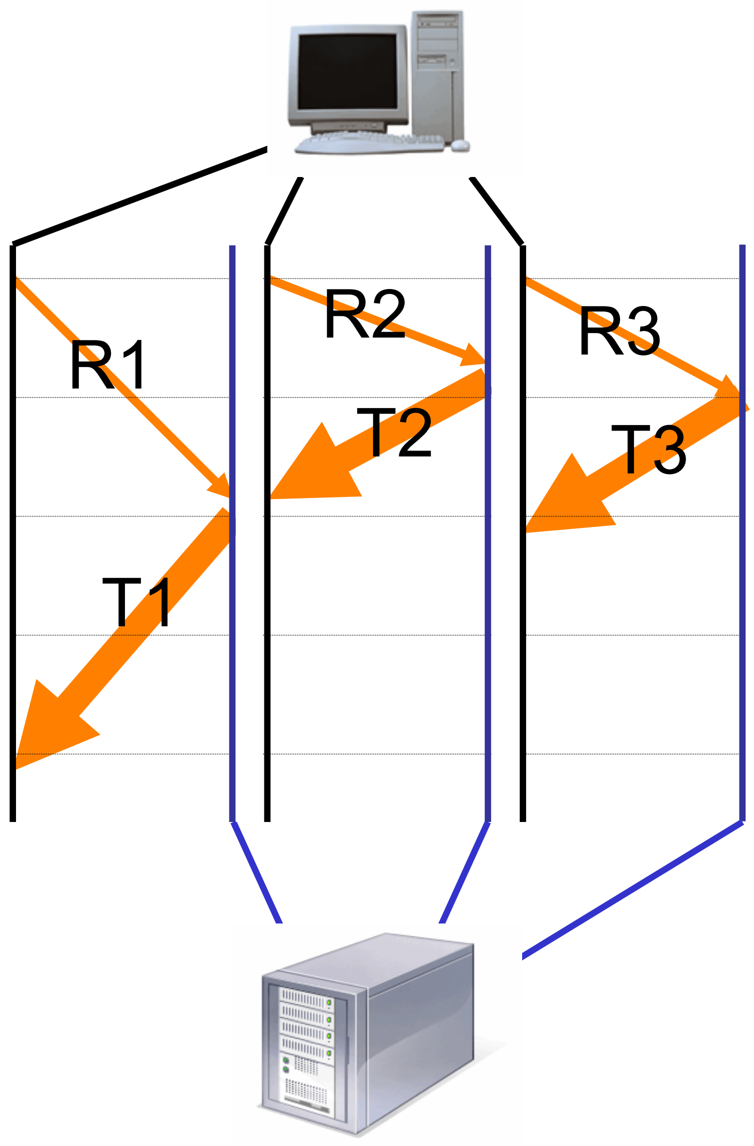
- Non-Persistent: One TCP connection to fetch one web resource
- Fairly poor PLT
- 2 Scenarios
  - Multiple TCP connections setups to the same server
  - Sequential request/responses even when resources are located on different servers
- Multiple TCP slow-start phases (more in lecture on TCP)



Improving HTTP Performance:

Concurrent Requests & Responses

- Use multiple connections *in parallel*
- Does not necessarily maintain order of responses



Quiz: Parallel HTTP Connections

- What are potential downsides of parallel HTTP connections, i.e. can opening too many parallel connections be harmful and if so in what way?

Persistent HTTP

Persistent HTTP

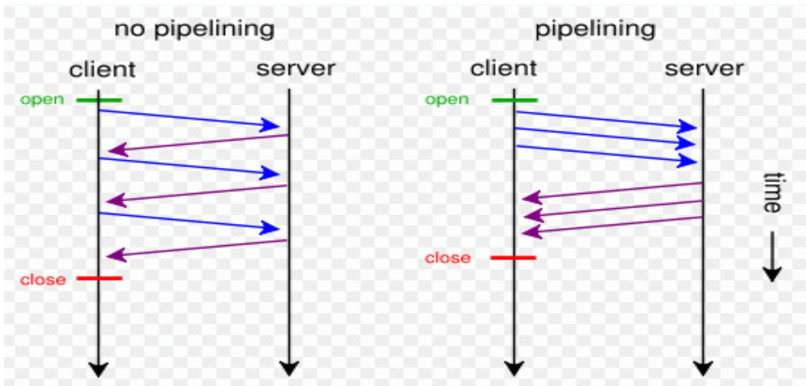
- server leaves TCP connection open after sending response
- subsequent HTTP messages between same client/server are sent over the same TCP connection
- Allow TCP to learn more accurate RTT estimate (APPARENT LATER IN THE COURSE)
- Allow TCP congestion window to increase (APPARENT LATER)
- i.e., leverage previously discovered bandwidth (APPARENT LATER)

Persistent without pipelining:

- client issues new request only when previous response has been received
- one RTT for each referenced object

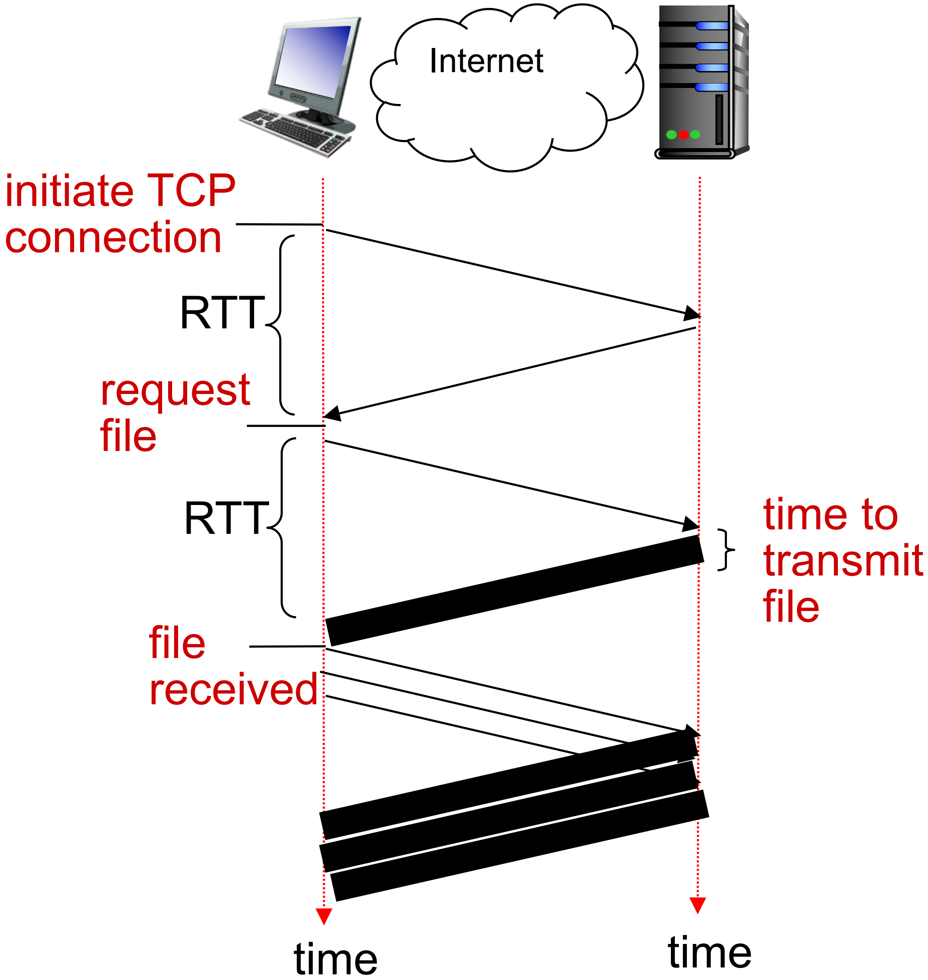
Persistent with pipelining:

- introduced 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



HTTP 1.1: response time with pipelining

Website with one index page and three embedded objects



How to improve PLT

- Reduce content size for transfer
  - Smaller images, compression
- Change HTTP to make better use of available bandwidth
  - Persistent connections and pipelining
- Change HTTP to avoid repeated transfers of the same content
  - Caching and web-proxies
- Move content closer to the client
  - CDNs

Improving HTTP Performance: Caching

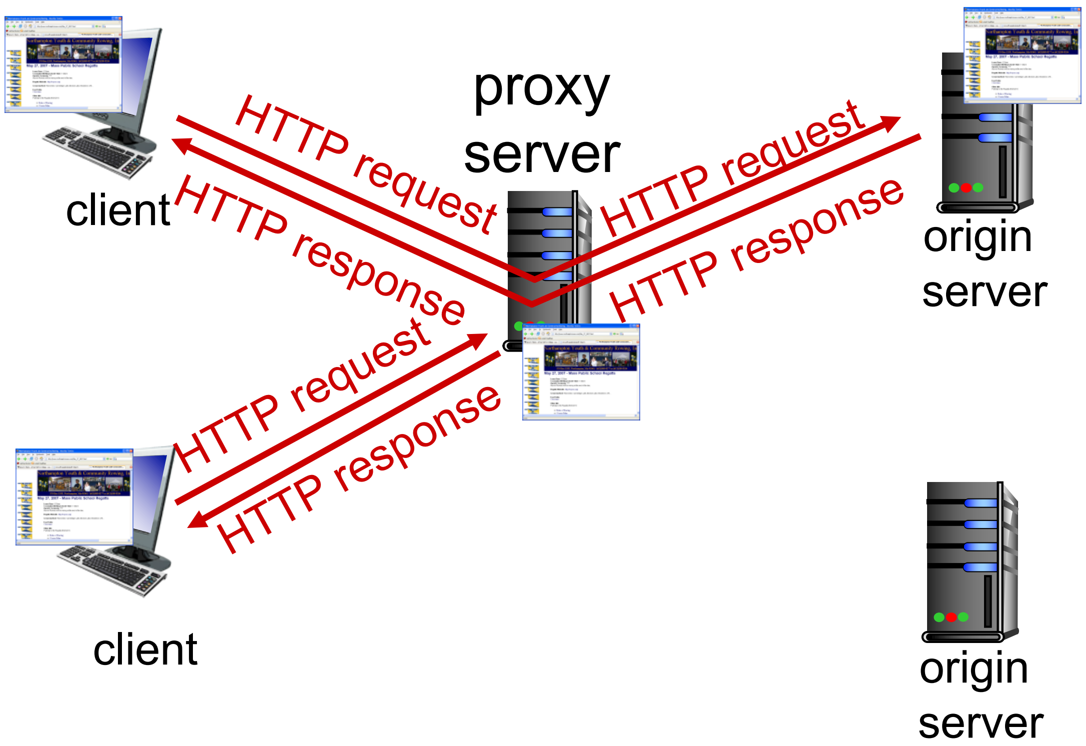
- Why does caching work?
  - Exploits *locality of reference*

- How well does caching work?
  - Very well, up to a limit
  - Large overlap in content
  - But many unique requests

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



More about Web caching

- 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)

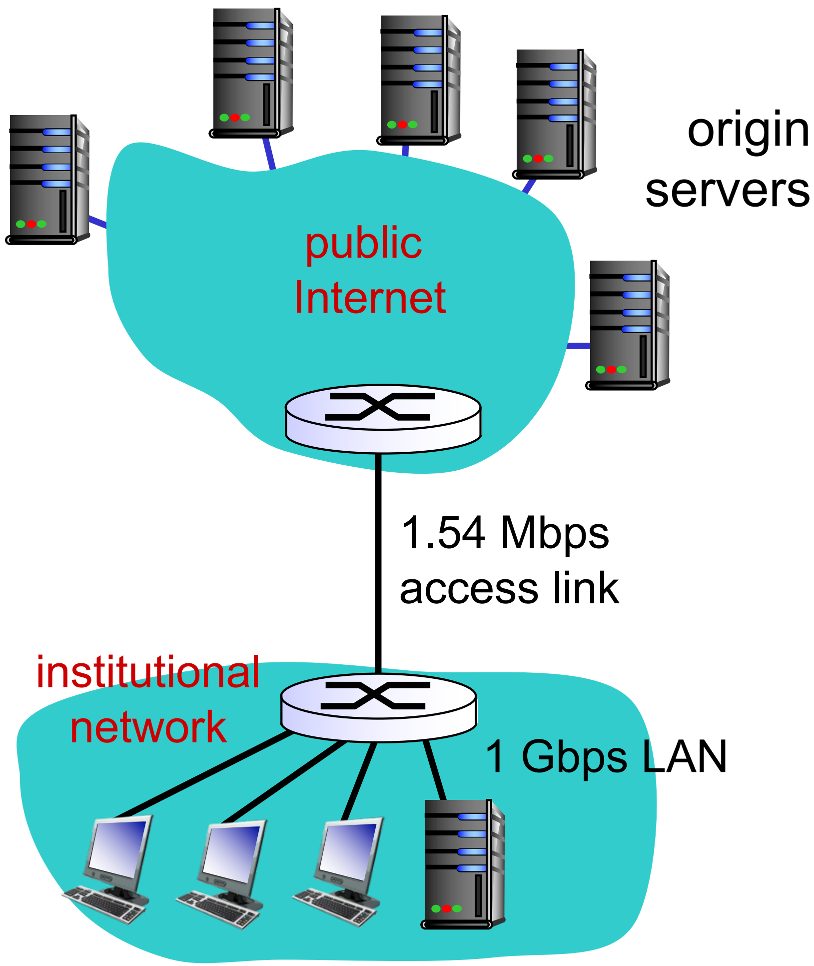
why Web caching?

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

Caching example:

assumptions:

- avg object size: 100K bits
- avg request rate from browsers to origin servers:15/sec
- avg data rate to browsers: 1.50 Mbps
- RTT from access router to any origin server: 2 sec
- access link rate: 1.54 Mbps



consequences:

- LAN utilization: 0.15%
- access link utilization = 99% *problem!*
- total delay = Internet delay + access delay + LAN delay
  = 2 sec + minutes + usecs

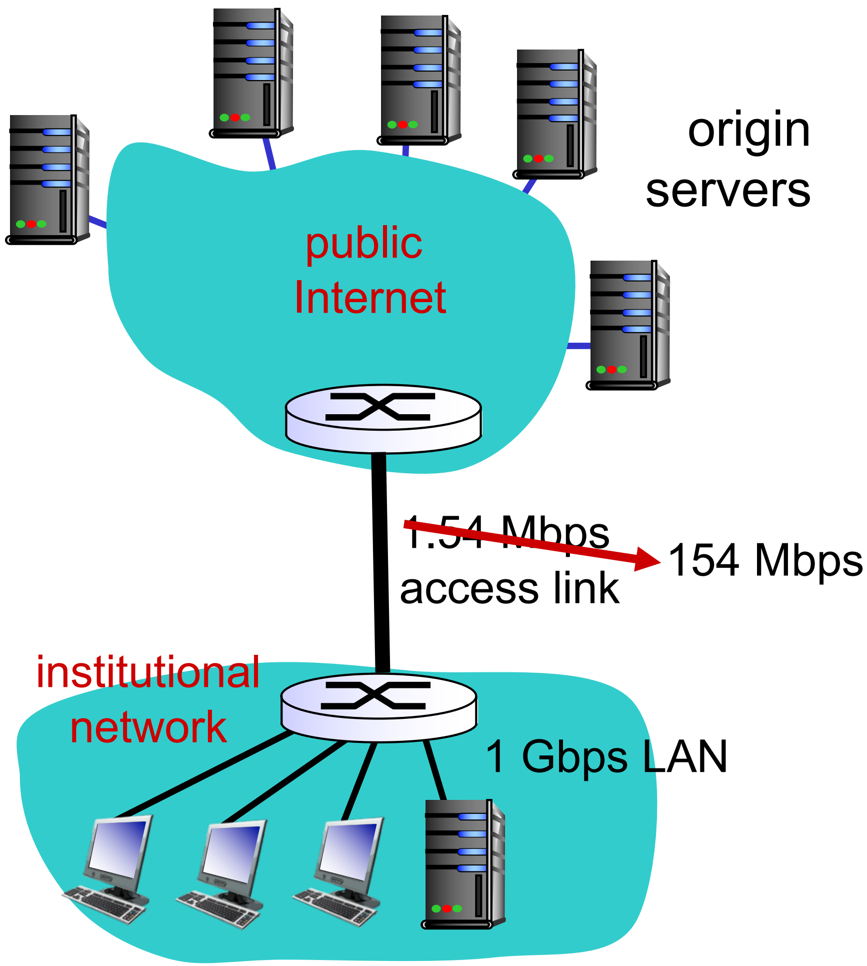
Caching example: fatter access link

assumptions:

- avg object size: 100K bits
- avg request rate from browsers to origin servers: 15/sec
- avg data rate to browsers: 1.50 Mbps
- RTT from access router to any origin server: 2 sec
- access link rate: ~~1.54 Mbps~~ 154 Mbps

consequences:

- LAN utilization: 0.15%
- access link utilization = ~~99%~~ 0.99%
- total delay = Internet delay + access delay + LAN delay
  = 2 sec + ~~minutes~~ msecs + usecs



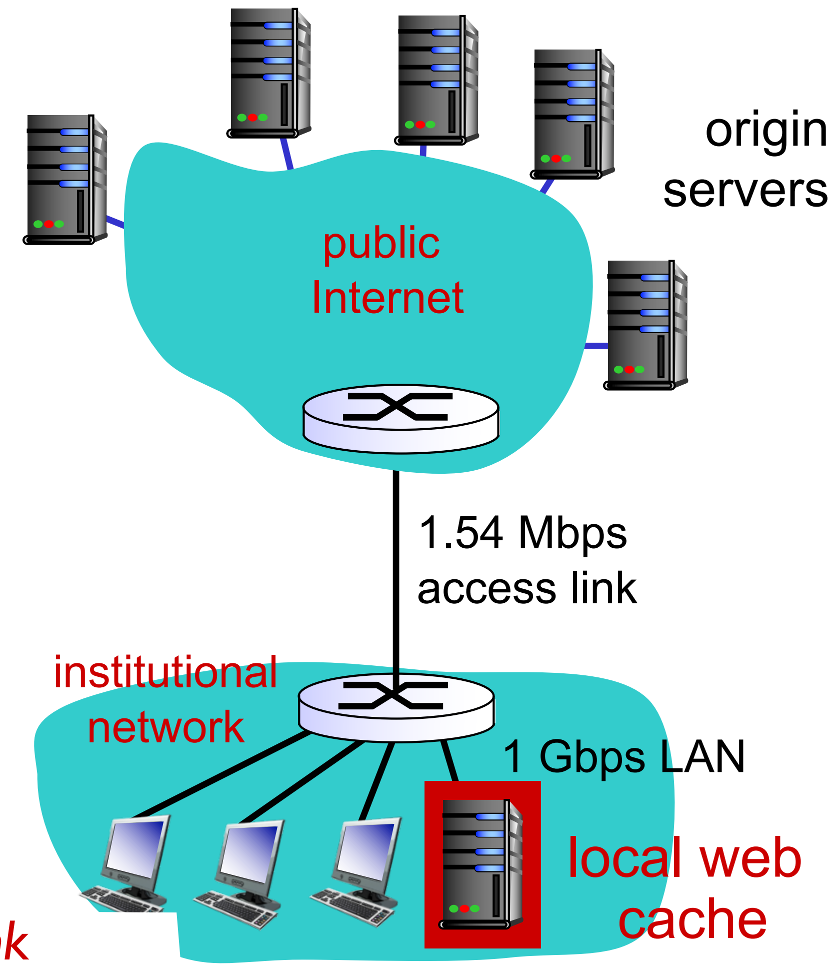
Cost: increased access link speed (not cheap!)

Caching example: install local cache

assumptions:
- avg object size: 100K bits
- avg request rate from browsers to origin servers:15/sec
- avg data rate to browsers: 1.50 Mbps
- RTT from access router to any origin server: 2 sec
- access link rate: 1.54 Mbps

consequences:
- LAN utilization: ?
- access link utilization = ?
- total delay = ?

How to compute link utilization, delay?



Cost: web cache (cheap!)

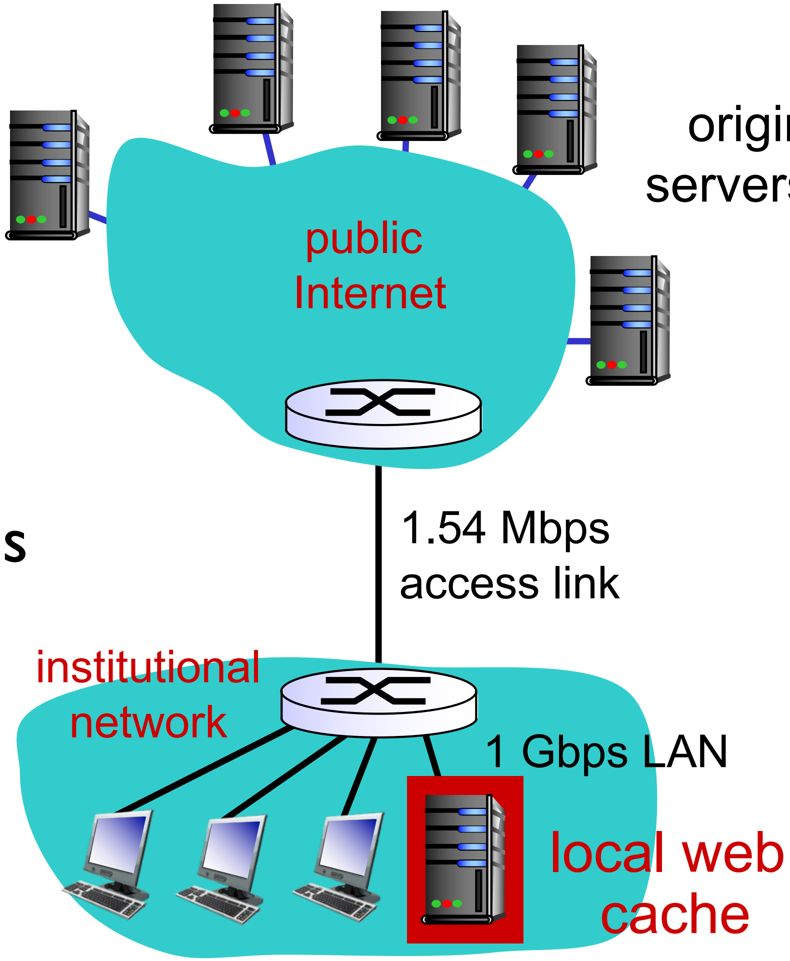
Caching example: install local cache

Calculating access link utilization, delay with cache:

- suppose cache hit rate is 0.4
  - 40% requests satisfied at cache, 60% requests satisfied at origin

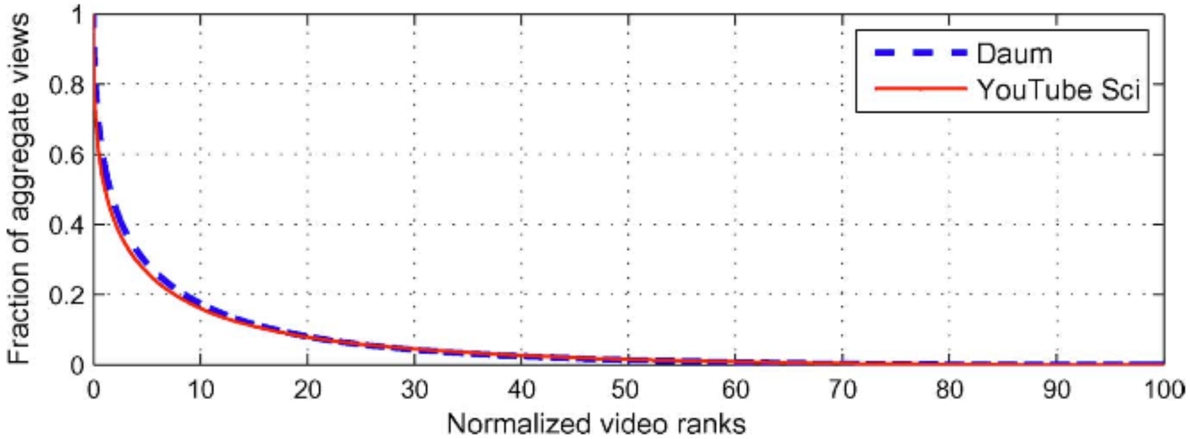
- access link utilization:
  - 60% of requests use access link
- data rate to browsers over access link = 0.6*1.50 Mbps = .9 Mbps
  - utilization = 0.9/1.54 = .58

- total delay
  - = 0.6 * (delay from origin servers) +0.4 * (delay when satisfied at cache)
  - = 0.6 (2.01) + 0.4 (~msecs)
  - = ~ 1.2 secs
  - less than with 154 Mbps link (and cheaper too!)



But what is the likelihood of cache hits?

- Distribution of web object requests generally follows a Zipf-like distribution
- *The probability that a document will be referenced k requests after it was last referenced is roughly proportional to 1/k* . That is, web traces exhibit excellent **temporal locality**.



Video content exhibits similar properties: 10% of the top popular videos account for nearly 80% of views, while the remaining 90% of videos account for total 20% of requests.

Paper – http://yongyeol.com/papers/cha-video-2009.pdf

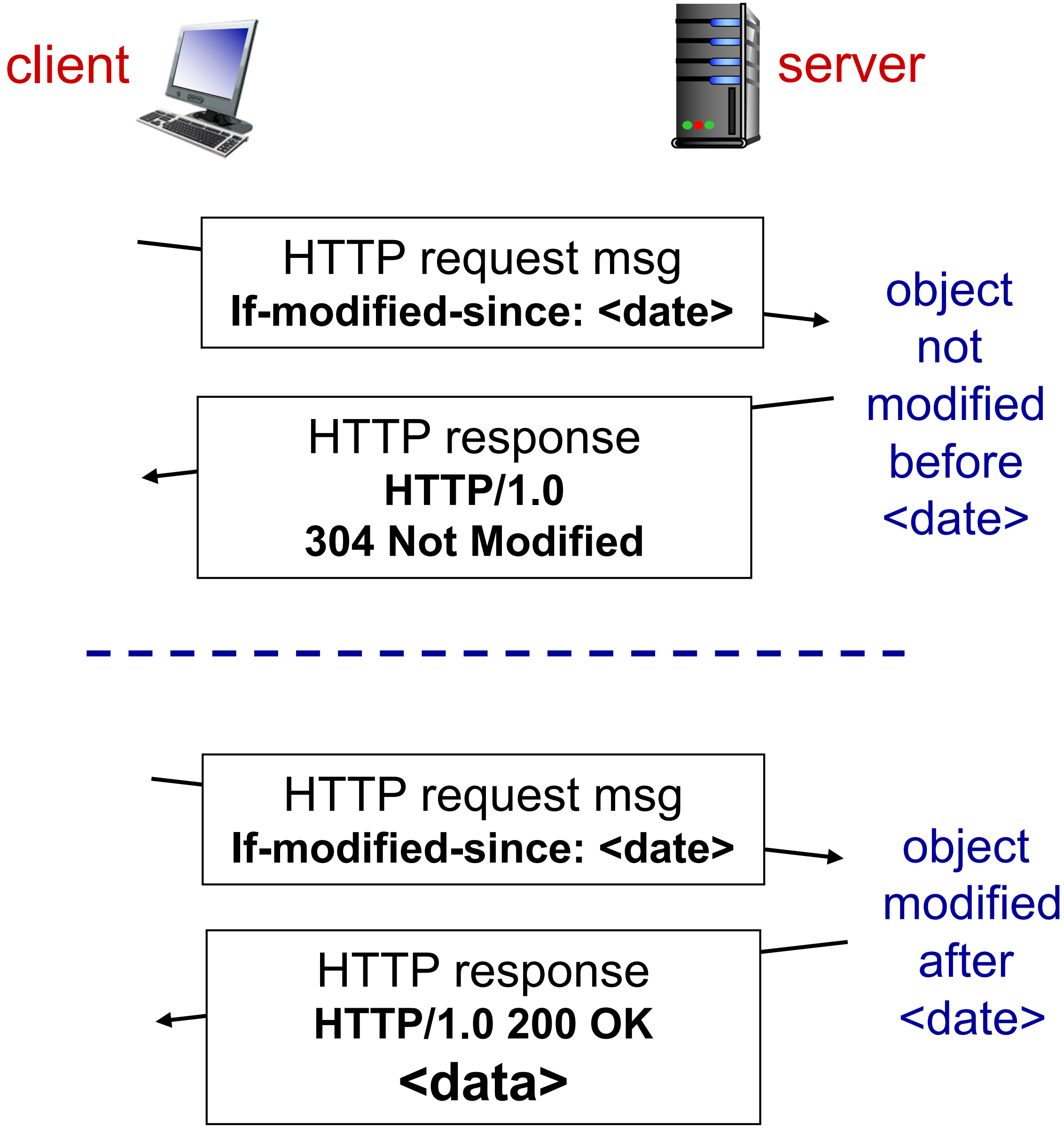
Paper – "Web Caching and Zipf-like Distributions: Evidence and Implications"
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.34.8742&rep=rep1&type=pdf

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`



Example Cache Check Request

```
GET / HTTP/1.1
Accept: */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
If-Modified-Since: Mon, 29 Jan 2001 17:54:18 GMT
If-None-Match: "7a11f-10ed-3a75ae4a"
User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows NT
   5.0)
Host: www.intel-iris.net
Connection: Keep-Alive
```

Example Cache Check Response

```
HTTP/1.1 304 Not Modified
Date: Tue, 27 Mar 2001 03:50:51 GMT
Server: Apache/1.3.14 (Unix)  (Red-Hat/Linux) mod_ssl/2.7.1
   OpenSSL/0.9.5a DAV/1.0.2 PHP/4.0.1pl2 mod_perl/1.24
Connection: Keep-Alive
Keep-Alive: timeout=15, max=100
ETag: "7a11f-10ed-3a75ae4a"
```

Improving HTTP Performance: Replication

- Replicate popular Web site across many machines
  - Spreads load on servers
  - Places content closer to clients
  - Helps when content isn't cacheable

- Problem:
  - Want to direct client to particular replica
    - Balance load across server replicas
    - Pair clients with nearby servers
  - Expensive

- Common solution:
  - DNS returns different addresses based on client's geo location, server load, *etc.*

More on this later

Improving HTTP Performance: CDN

- Caching and replication as a service
- Integrate forward and reverse caching functionality
- Large-scale distributed storage infrastructure (usually) administered by one entity
  - *e.g.,* Akamai has servers in 20,000+ locations
- Combination of (pull) caching and (push) replication
  - **Pull:** Direct result of clients' requests
  - **Push:** Expectation of high access rate
- Also do some processing
  - Handle *dynamic* web pages
  - *Transcoding*
  - *Maybe do some security function – watermark IP*

What about HTTPS?

- HTTP is insecure
- HTTP basic authentication: password sent using base64 encoding (can be readily converted to plaintext)
- HTTPS: HTTP over a connection encrypted by Transport Layer Security (TLS)
- Provides:
  - Authentication
  - Bidirectional encryption
- Widely used in place of plain vanilla HTTP

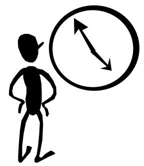
What’s on the horizon: HTTP/2

- Google SPDY (speedy) -> HTTP/2: (RFC 7540 May 2015)
- Better content structure
- Improvements
  - Severs can **push** content and thus reduce overhead of an additional request cycle
  - Fully multiplexed
    - Requests and responses are sliced in smaller chunks called frames, frames are tagged with an ID that connects data to the request/response
    - overcomes Head-of-line blocking in HTTP 1.1
  - Prioritisation of the order in which objects should be sent (e.g. CSS files may be given higher priority)
  - Data compression of HTTP headers
    - Some headers such as cookies can be very long
    - Repetitive information

More details: https://http2.github.io/faq/
Demo: https://http2.akamai.com/demo

Summary

- Application Layer (Chapter 2)
  - Principles of Network Applications
  - HTTP
- Next Week: Application Layer (contd.)
  - E-mail
  - DNS



Reading Exercise
Chapter 2: 2.3 – 2.7