15-441 Computer Networking
The Web

Web history

- 1945: Vannevar Bush, “As we may think”, Atlantic Monthly, July, 1945.
  - describes the idea of a distributed hypertext system.
  - a “memex” that mimics the “web of trails” in our minds.
- 1989: Tim Berners-Lee (CERN) writes internal proposal to develop a distributed hypertext system
  - connects “a web of notes with links”.
  - intended to help CERN physicists in large projects share and manage information
- 1990: Tim BL writes graphical browser for Next machines.

Web history (cont)

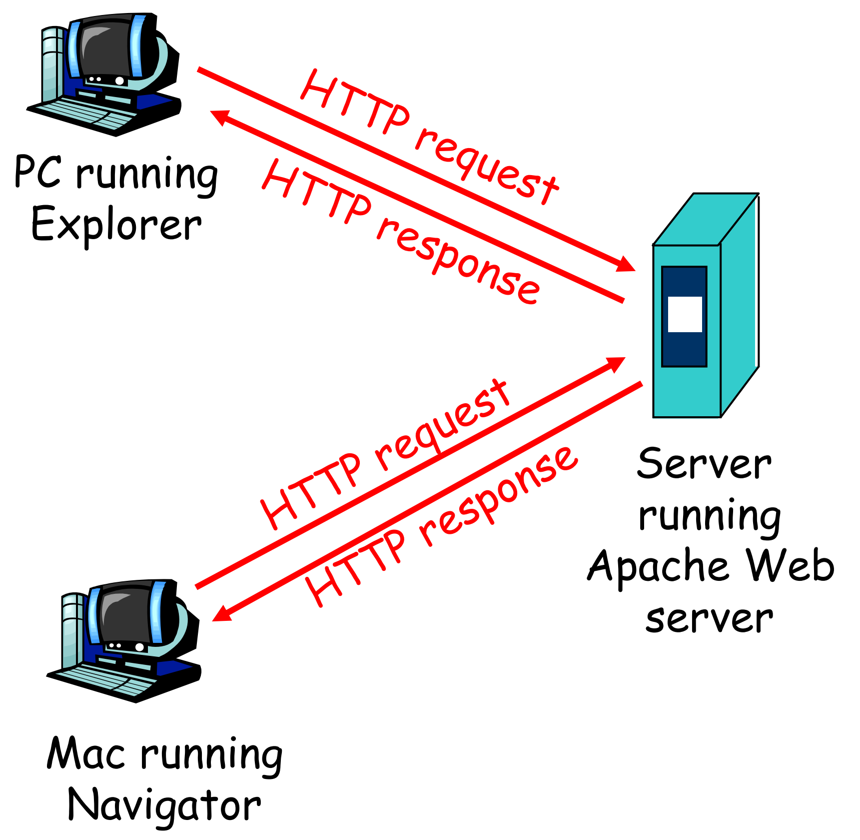
- 1992
  - NCSA server released
  - 26 WWW servers worldwide
- 1993
  - Marc Andreessen releases first version of NCSA Mosaic Mosaic version released for (Windows, Mac, Unix).
  - Web (port 80) traffic at 1% of NSFNET backbone traffic.
  - Over 200 WWW servers worldwide.
- 1994
  - Andreessen and colleagues leave NCSA to form "Mosaic Communications Corp" (Netscape).

Design the Web

- How would a computer scientist do it?
- What are the important considerations?
  - What are NOT important?
- What should be the basic architecture?
  - What are the components?
  - What are the interfaces of components?

Basic Concepts

- client/server model
  - *client:* browser that requests, receives, "displays" Web objects
  - *server:* Web server sends objects in response to requests
- HTTP: Web's application layer protocol
  - HTTP 1.0: RFC 1945
  - HTTP 1.1: RFC 2068



Basic Concepts

- Web page consists of objects
- Web page consists of base HTML-file which includes several referenced objects
- Object can be HTML file, JPEG image, Java applet, audio file,…
- Each page or object is addressable by a URL

Overview of Concepts in This Lecture

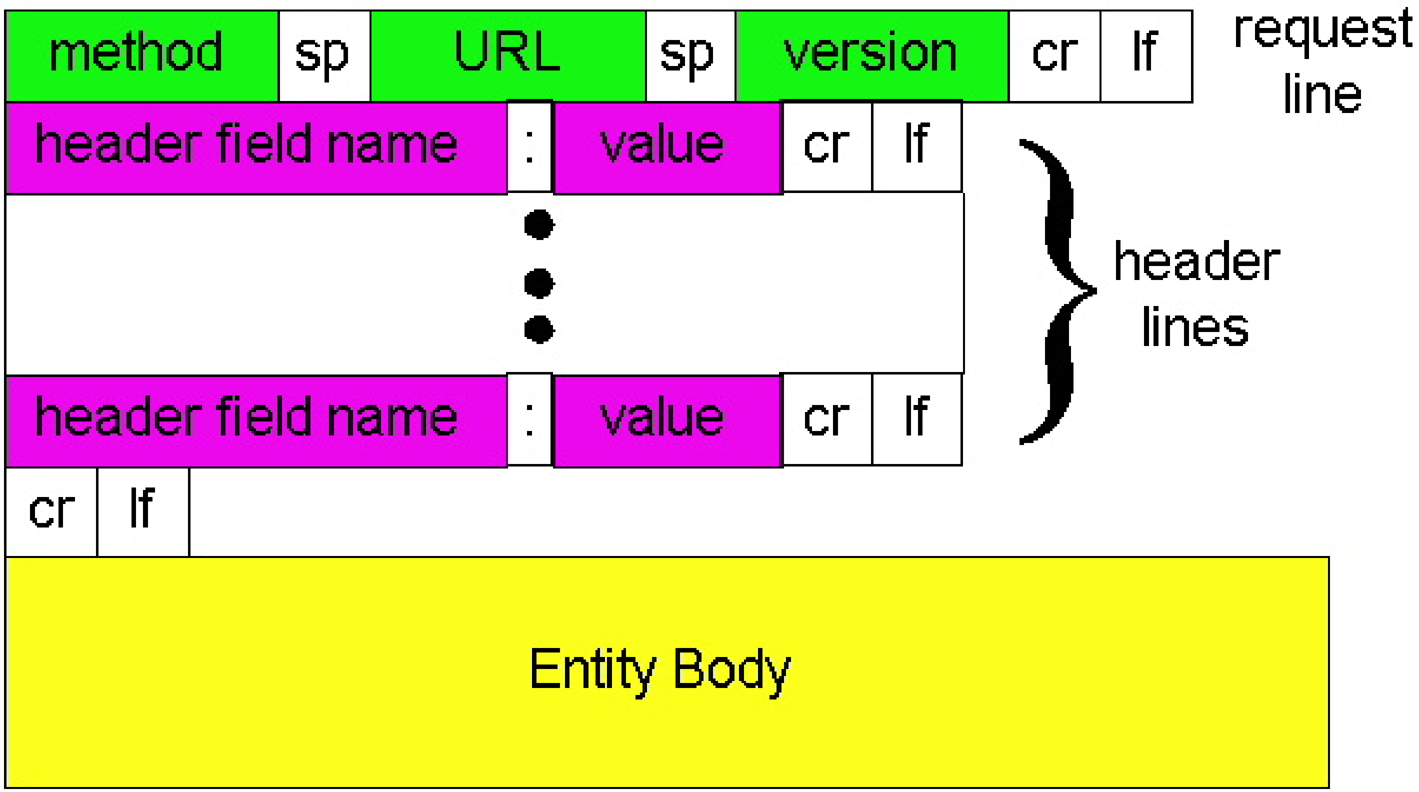
- HTTP
- Interaction between HTTP and TCP
- Persistent HTTP
- Caching
- Content Distribution Network (CDN)

- State
  - What is stateless protocol? Advantages and disadvantages?
  - What type of states are used in the Web?
  - Issues of maintaining state

HTTP Basics

- HTTP layered over bidirectional byte stream
  - Almost always TCP
- Interaction
  - Client sends request to server, followed by response from server to client
  - Requests/responses are encoded in text
- Stateless
  - Server maintains no information about past client requests

HTTP Request



HTTP Request Example

GET / HTTP/1.1

Accept: */*

Accept-Language: en-us

Accept-Encoding: gzip, deflate

User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 5.0)

Host: www.intel-iris.net

Connection: Keep-Alive

HTTP Response Example

HTTP/1.1 200 OK

Date: Tue, 27 Mar 2001 03:49:38 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

Last-Modified: Mon, 29 Jan 2001 17:54:18 GMT

ETag: "7a11f-10ed-3a75ae4a"

Accept-Ranges: bytes

Content-Length: 4333

Keep-Alive: timeout=15, max=100

Connection: Keep-Alive

Content-Type: text/html

…..

HTTP Request

- Request line
  - Method
    - GET – return URI
    - HEAD – return headers only of GET response
    - POST – send data to the server (forms, etc.)
  - URL (relative)
    - E.g., /index.html
  - HTTP version

HTTP Request (cont.)

- Request headers
  - Authorization – authentication info
  - Acceptable document types/encodings
  - From – user email
  - If-Modified-Since
  - Referrer – what caused this page to be requested
  - User-Agent – client software
- Blank-line
- Body

HTTP Response

- Status-line
  - HTTP version
  - 3 digit response code
    - 1XX – informational
    - 2XX – success
      - 200 OK
    - 3XX – redirection
      - 301 Moved Permanently
      - 303 Moved Temporarily
      - 304 Not Modified
    - 4XX – client error
      - 404 Not Found
    - 5XX – server error
      - 505 HTTP Version Not Supported
  - Reason phrase

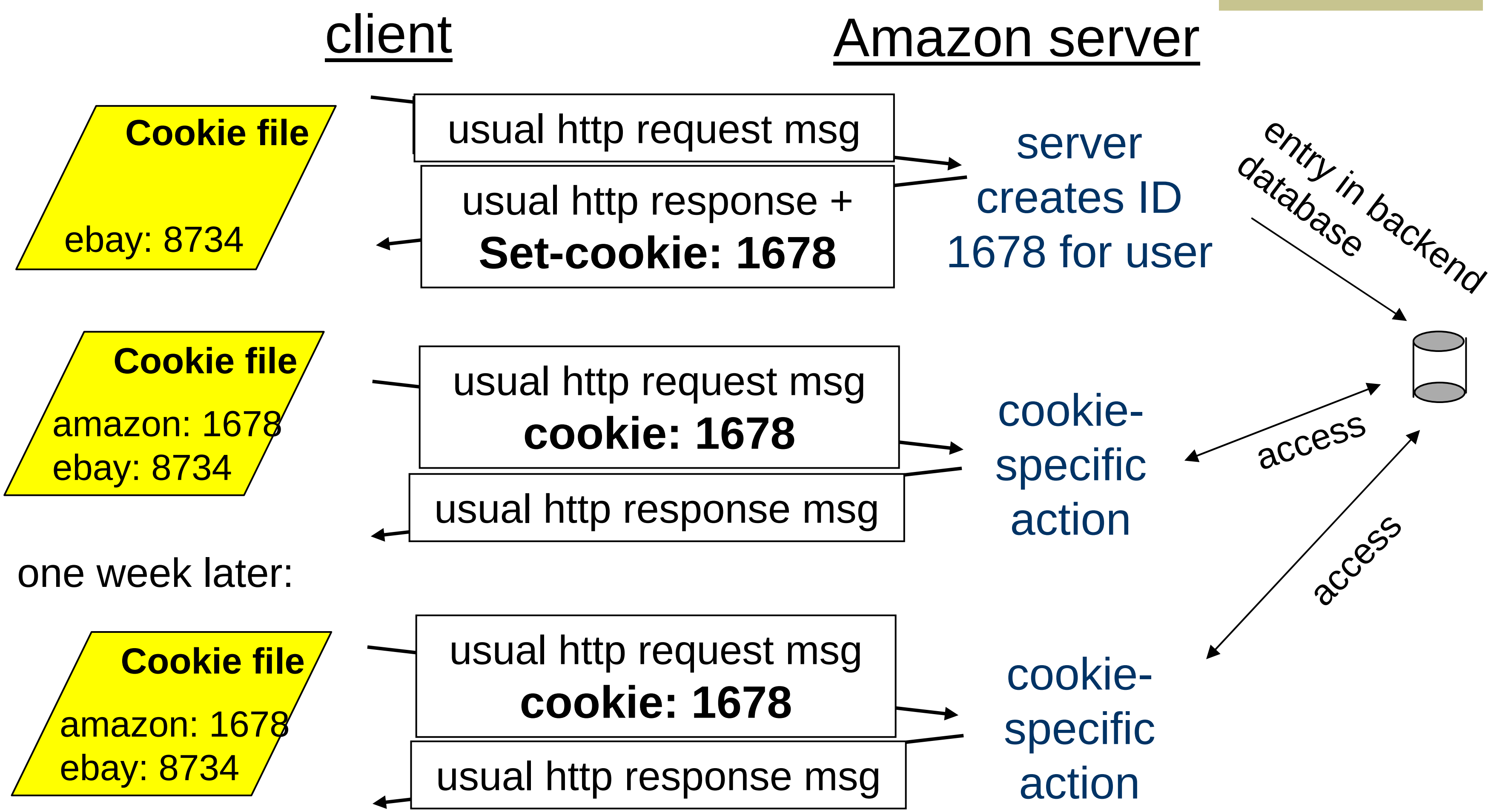
HTTP Response (cont.)

- Headers
  - Location – for redirection
  - Server – server software
  - WWW-Authenticate – request for authentication
  - Allow – list of methods supported (get, head, etc)
  - Content-Encoding – E.g x-gzip
  - Content-Length
  - Content-Type
  - Expires
  - Last-Modified
- Blank-line
- Body

How to Mark End of Message?

- Size of message → Content-Length
  - Implications:
    - must know size of transfer in advance
    - What applications are not appropriate?
- Close connection
  - Only server can do this

Cookies: Keeping “State” (Cont.)



Cookies: Keeping “state”

Many major Web sites use cookies

Four components:

1) Cookie header line in the HTTP response message
2) Cookie header line in HTTP request message
3) Cookie file kept on user’s host and managed by user’s browser
4) Back-end database at Web site

Example:

- Susan access Internet always from same PC
- She visits a specific e-commerce site for first time
- When initial HTTP requests arrives at site, site creates a unique ID and creates an entry in backend database for ID

Outline

- Web intro, HTTP
- Persistent HTTP
- HTTP caching
- Content distribution networks

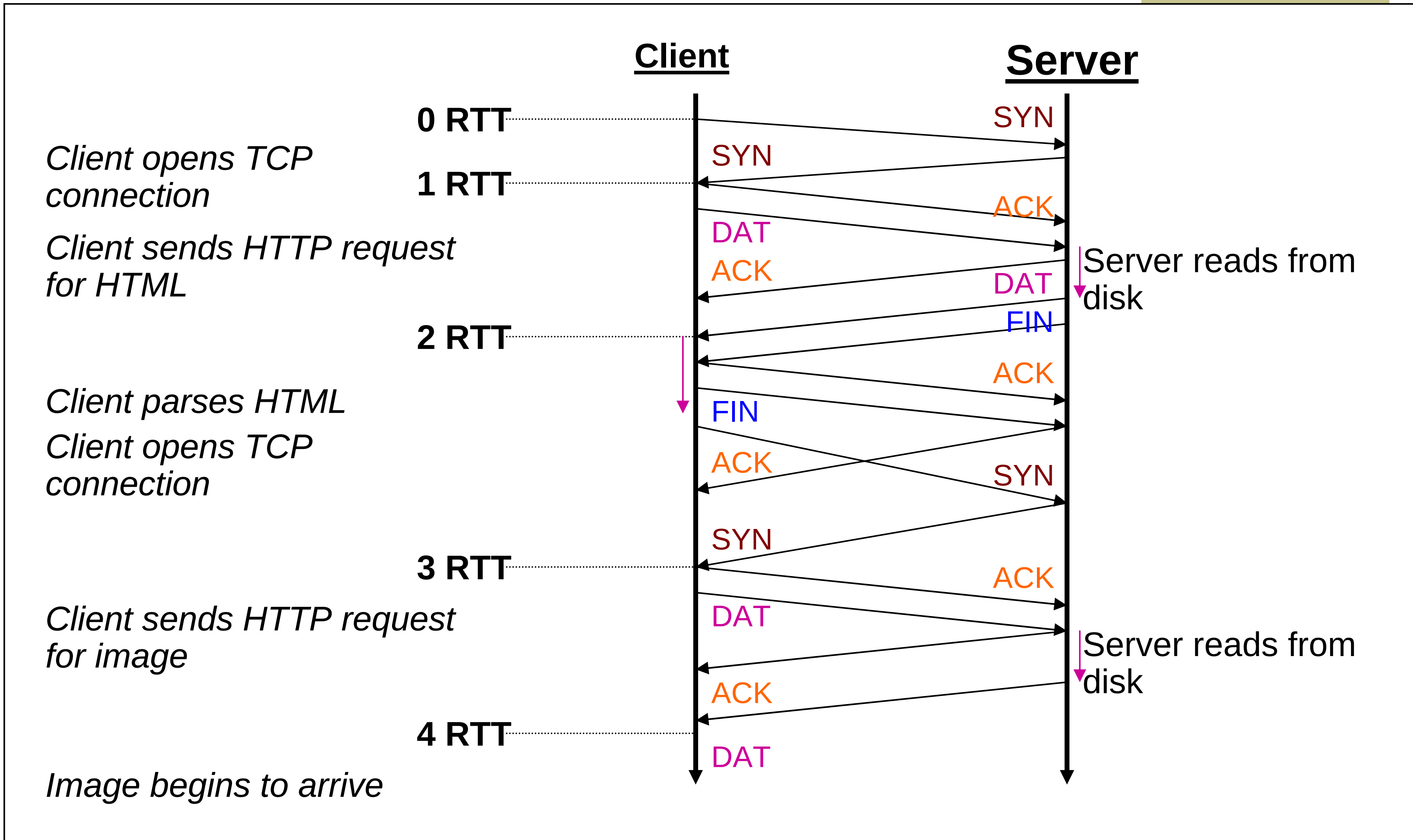
Typical Workload (Web Pages)

- Multiple (typically small) objects per page
- File sizes
  - Heavy-tailed
    - Pareto distribution for tail
    - Lognormal for body of distribution
- Embedded references
  - Number of embedded objects = pareto – $p(x) = ak^a x^{-(a+1)}$

HTTP 0.9/1.0

- One request/response per TCP connection
  - Simple to implement
- Disadvantages
  - Multiple connection setups → three-way handshake each time
    - Several extra round trips added to transfer
  - Multiple slow starts

Single Transfer Example



More Problems

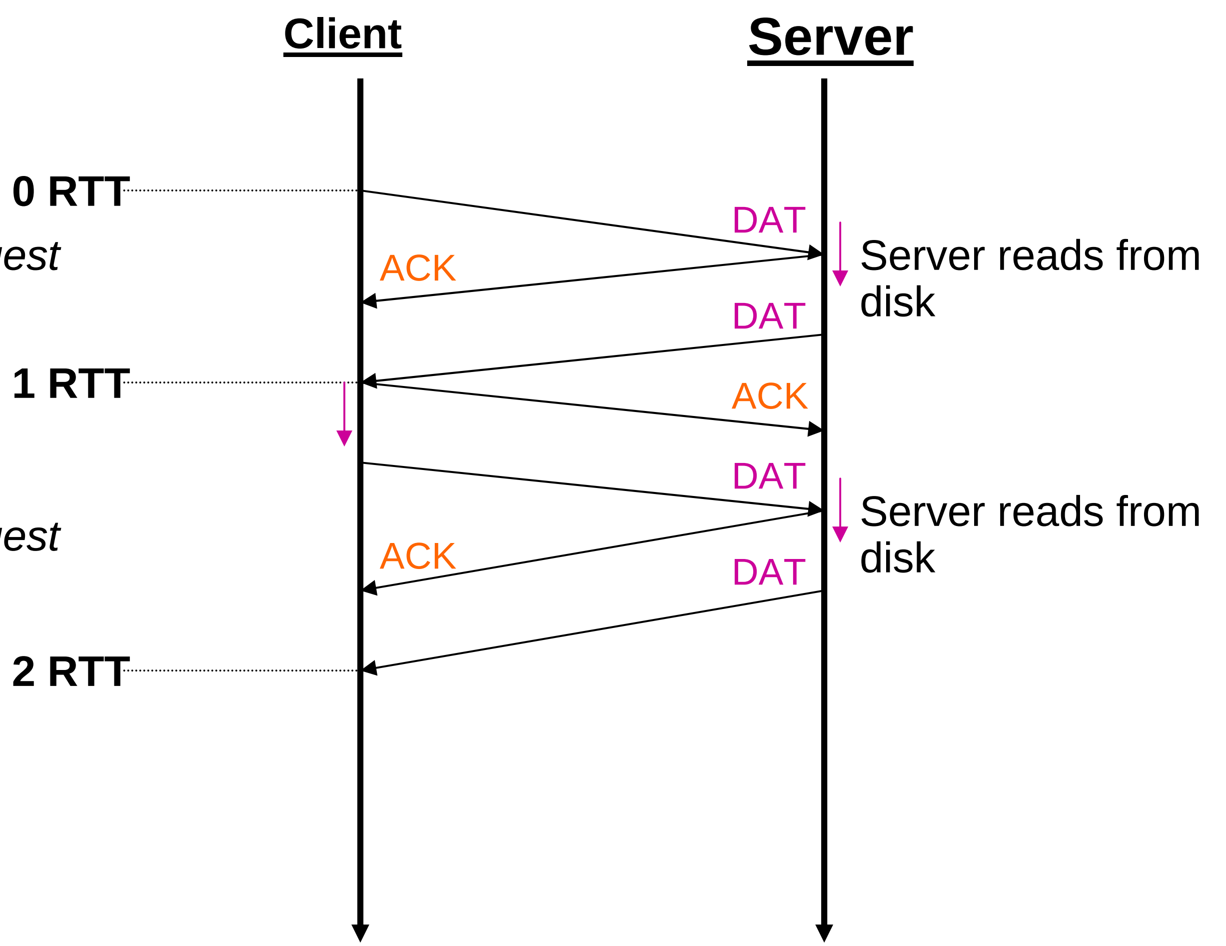
- Short transfers are hard on TCP
  - Stuck in slow start
  - Loss recovery is poor when windows are small
- Lots of extra connections
  - Increases server state/processing
- Server also forced to keep TIME_WAIT connection state
  - Why must server keep these?
  - Tends to be an order of magnitude greater than # of active connections, why?

Persistent Connection Solution

- Multiplex multiple transfers onto one TCP connection

- How to identify requests/responses
  - Delimiter → Server must examine response for delimiter string
  - Content-length and delimiter → Must know size of transfer in advance
  - Block-based transmission → send in multiple length delimited blocks
  - Store-and-forward → wait for entire response and then use content-length
  - Solution → use existing methods and close connection otherwise

Persistent Connection Example



Client sends HTTP request for HTML

Client parses HTML

Client sends HTTP request for image

Image begins to arrive

Persistent HTTP

Nonpersistent HTTP issues:

- Requires 2 RTTs per object
- OS must work and allocate host resources for each TCP connection
- But browsers often open parallel TCP connections to fetch referenced objects

Persistent HTTP

- Server leaves connection open after sending response
- Subsequent HTTP messages between same client/server are sent over connection

Persistent without pipelining:

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

Persistent with pipelining:

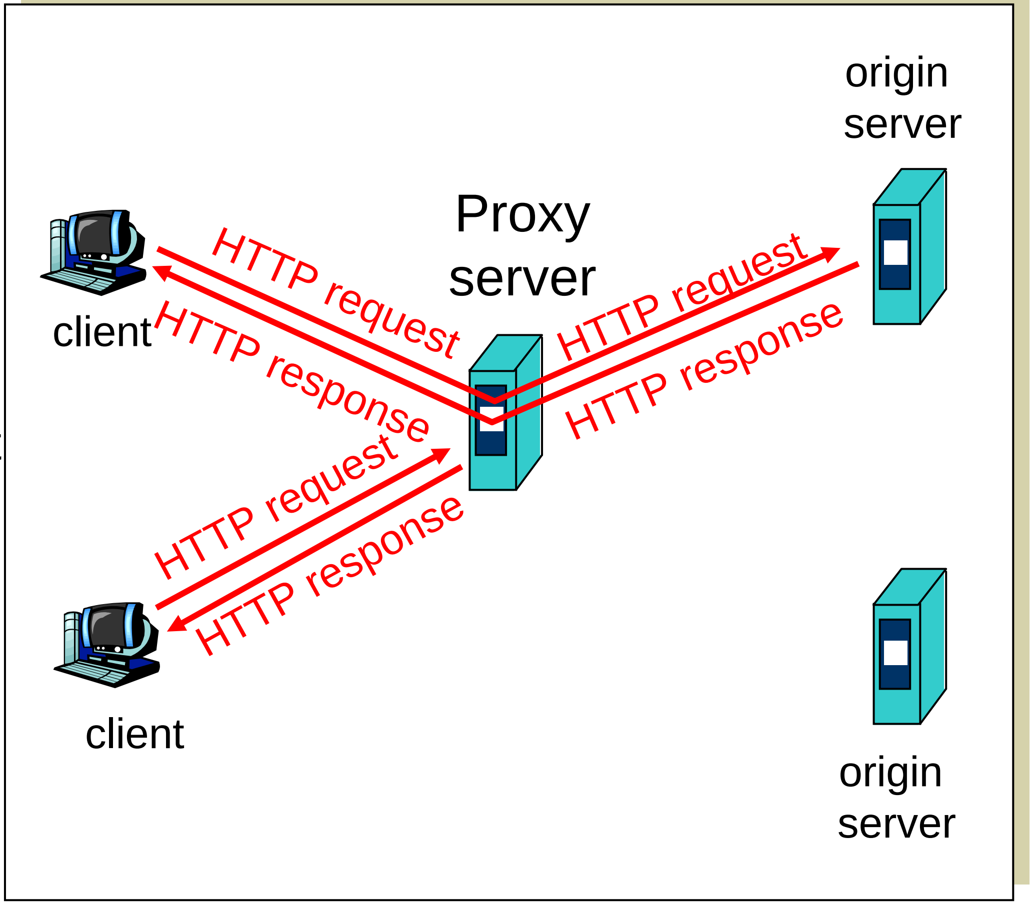
- Default in HTTP/1.1
- Client sends requests as soon as it encounters a referenced object
- As little as one RTT for all the referenced objects

Outline

- Web Intro, HTTP
- Persistent HTTP
- Caching
- Content distribution networks

Web Proxy Caches

- User configures 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



Caching Example (1)

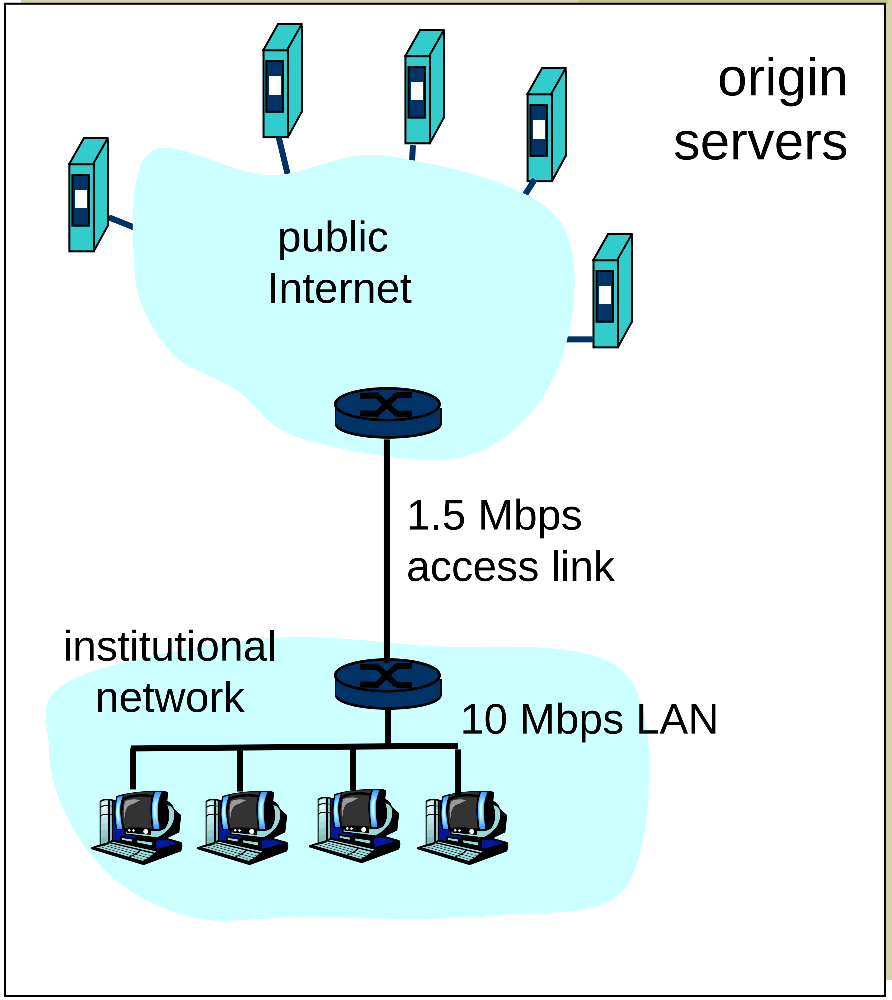
<u>Assumptions</u>

- Average object size = 100,000 bits
- Avg. request rate from institution's browser to origin servers = 15/sec
- Delay from institutional router to any origin server and back to router = 2 sec

<u>Consequences</u>

- Utilization on LAN = 15%
- Utilization on access link = 100%
- Total delay = Internet delay + access delay + LAN delay

= 2 sec + minutes + milliseconds



Caching Example (2)

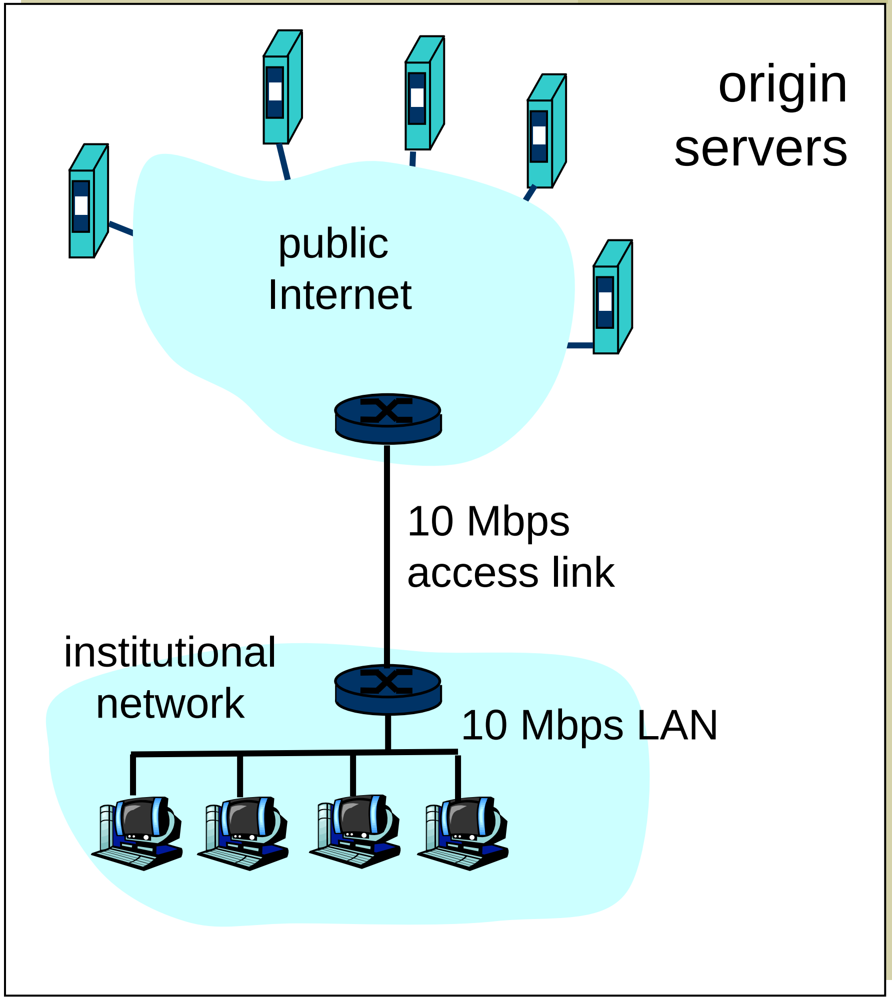
Possible solution

- Increase bandwidth of access link to, say, 10 Mbps
- Often a costly upgrade

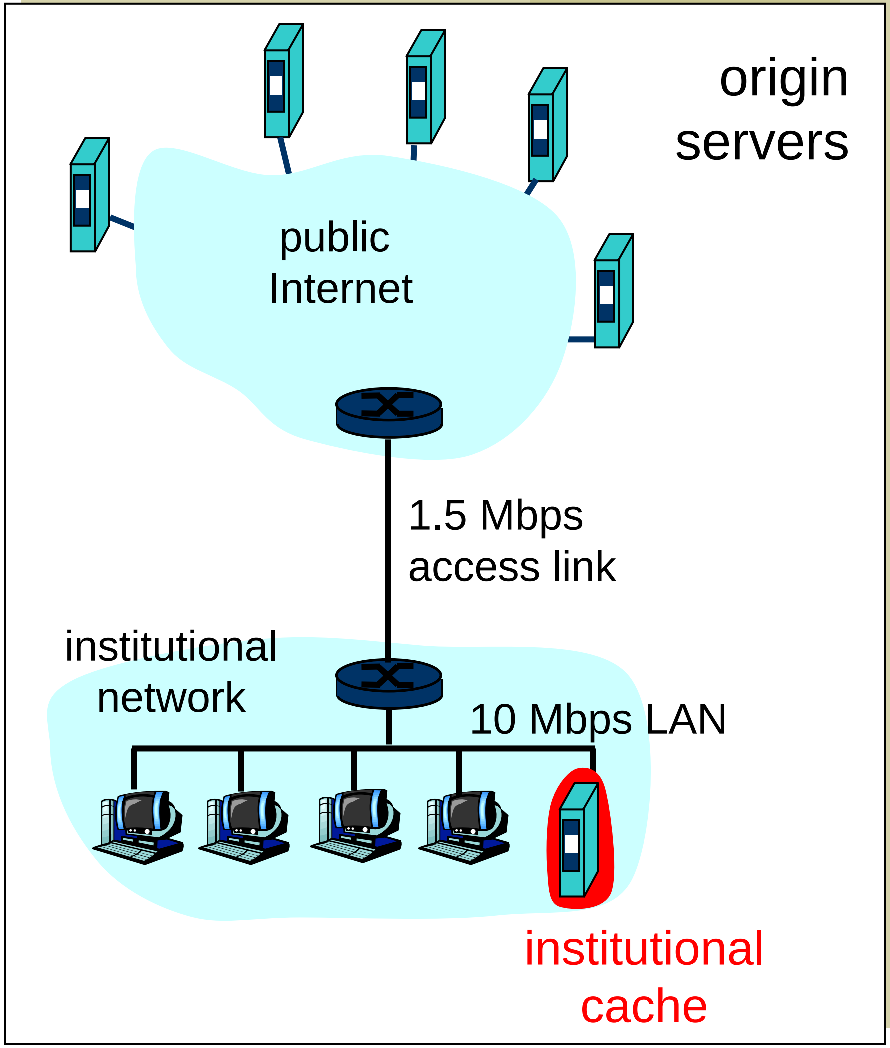
Consequences

- Utilization on LAN = 15%
- Utilization on access link = 15%
- Total delay = Internet delay + access delay + LAN delay

= 2 sec + msecs + msecs



Caching Example (3)



Install cache

- Suppose hit rate is .4

Consequence

- 40% requests will be satisfied almost immediately (say 10 msec)
- 60% requests satisfied by origin server
- Utilization of access link reduced to 60%, resulting in negligible delays
- Weighted average of delays

= .6*2 sec + .4*10msecs < 1.3 secs

HTTP Caching

- Clients often cache documents
  - Challenge: update of documents
  - If-Modified-Since requests to check
    - HTTP 0.9/1.0 used just date
    - HTTP 1.1 has an opaque “entity tag” (could be a file signature, etc.) as well
- When/how often should the original be checked for changes?
  - Check every time?
  - Check each session? Day? Etc?
  - Use Expires header
    - If no Expires, often use Last-Modified as estimate

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"

Problems

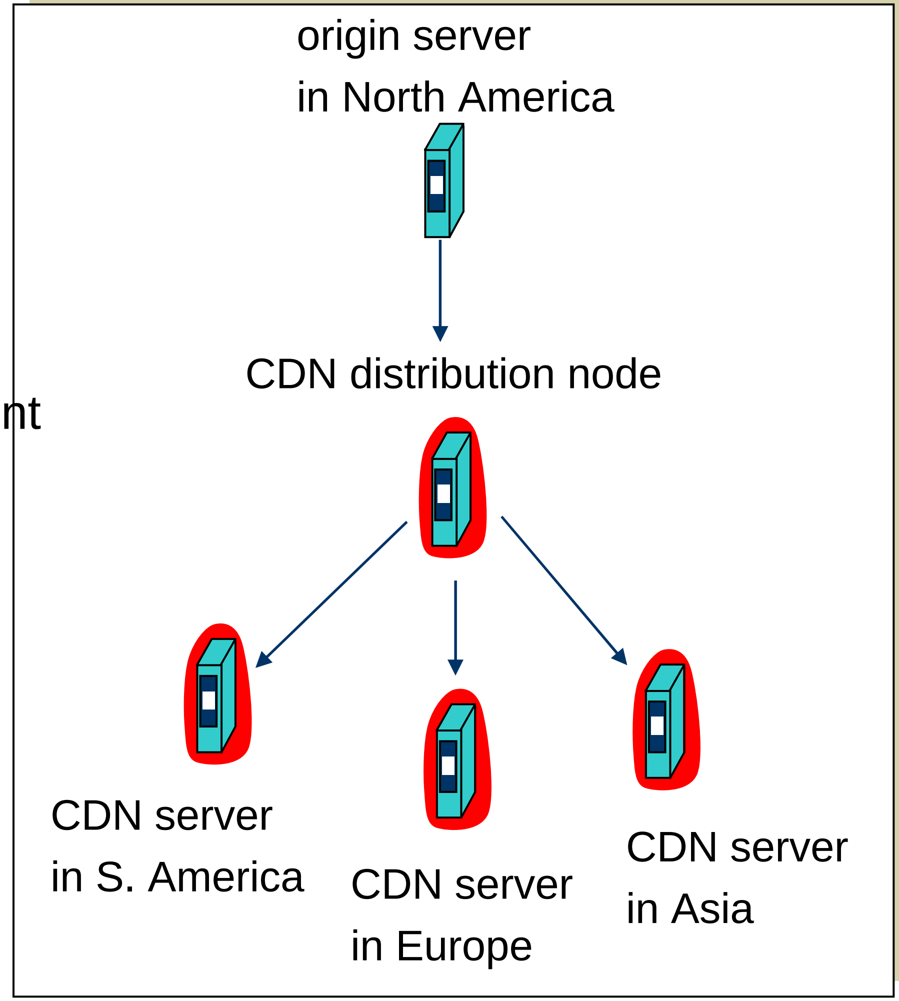
- Over 50% of all HTTP objects are uncacheable – why?
- Not easily solvable
  - Dynamic data → stock prices, scores, web cams
  - CGI scripts → results based on passed parameters
- Obvious fixes
  - SSL → encrypted data is not cacheable
    - Most web clients don’t handle mixed pages well →many generic objects transferred with SSL
  - Cookies → results may be based on passed data
  - Hit metering → owner wants to measure # of hits for revenue, etc.
- What will be the end result?

Content Distribution Networks (CDNs)

- The content providers are the CDN customers.

Content replication

- CDN company installs hundreds of CDN servers throughout Internet
  - Close to users
- CDN replicates its customers' content in CDN servers. When provider updates content, CDN updates servers



Outline

- HTTP intro and details
- Persistent HTTP
- HTTP caching
- Content distribution networks

Content Distribution Networks & Server Selection

- Replicate content on many servers
- Challenges
  - How to replicate content
  - Where to replicate content
  - How to find replicated content
  - How to choose among know replicas
  - How to direct clients towards replica

Server Selection

- Which server?
  - Lowest load → to balance load on servers
  - Best performance → to improve client performance
    - Based on Geography? RTT? Throughput? Load?
  - Any alive node → to provide fault tolerance
- How to direct clients to a particular server?
  - As part of routing → anycast, cluster load balancing
    - Not covered ☹
  - As part of application → HTTP redirect
  - As part of naming → DNS

Application Based

- HTTP supports simple way to indicate that Web page has moved (30X responses)
- Server receives Get request from client
  - Decides which server is best suited for particular client and object
  - Returns HTTP redirect to that server
- Can make informed application specific decision
- May introduce additional overhead → multiple connection setup, name lookups, etc.
- While good solution in general, but…
  - HTTP Redirect has some design flaws – especially with current browsers

Naming Based

- Client does name lookup for service
- Name server chooses appropriate server address
  - A-record returned is "best" one for the client
- What information can name server base decision on?
  - Server load/location → must be collected
  - Information in the name lookup request
    - Name service client → typically the local name server for client

How Akamai Works

- Clients fetch html document from primary server
  - E.g. fetch index.html from cnn.com
- URLs for replicated content are replaced in html
  - E.g. <img src=“http://cnn.com/af/x.gif”> replaced with <img src=“http://a73.g.akamaitech.net/7/23/cnn.com/af/x.gif”>
- Client is forced to resolve aXYZ.g.akamaitech.net hostname

How Akamai Works

- How is content replicated?
- Akamai only replicates static content (*)
- Modified name contains original file name
- Akamai server is asked for content
  - First checks local cache
  - If not in cache, requests file from primary server and caches file

* (At least, the version we're talking about today. Akamai actually lets sites write code that can run on Akamai's servers, but that's a pretty different beast)

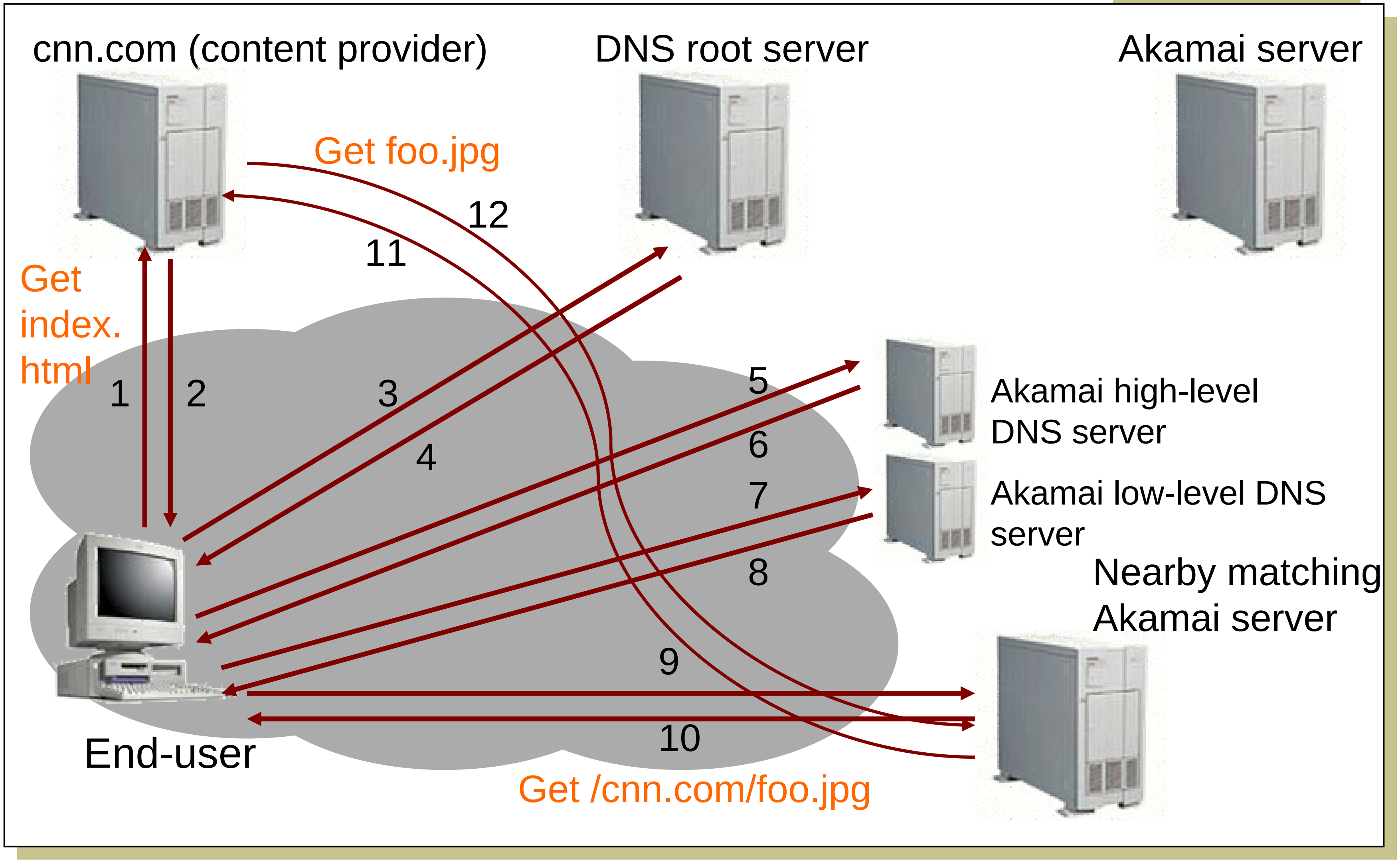
How Akamai Works

- Root server gives NS record for akamai.net
- Akamai.net name server returns NS record for g.akamaitech.net
  - Name server chosen to be in region of client's name server
  - TTL is large
- G.akamaitech.net nameserver chooses server in region
  - Should try to chose server that has file in cache - How to choose?
  - Uses aXYZ name and hash
  - TTL is small → why?

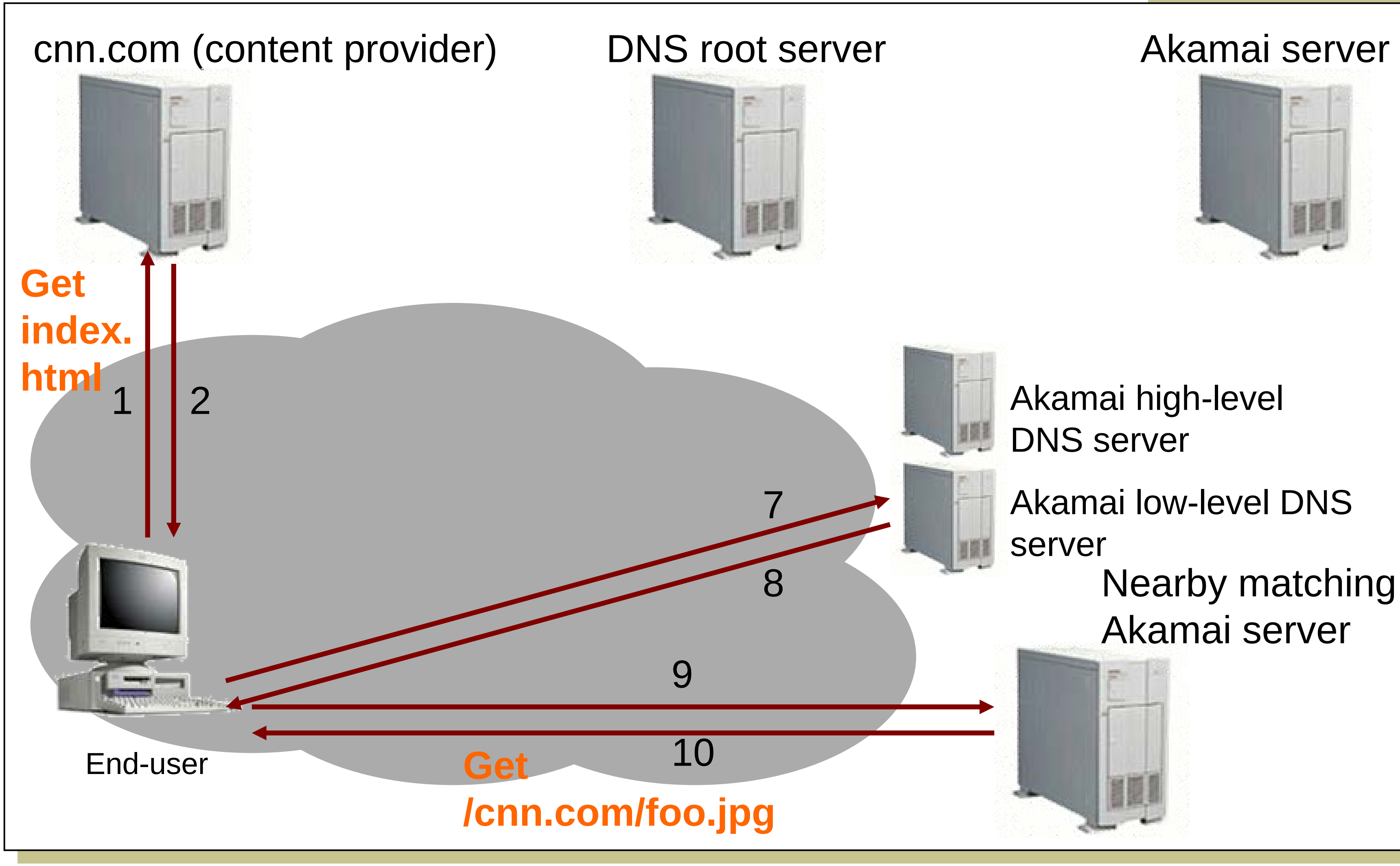
Simple Hashing

- Given document XYZ, we need to choose a server to use
- Suppose we use modulo
- Number servers from 1…n
  - Place document XYZ on server (XYZ mod n)
  - What happens when a servers fails? n → n-1
    - Same if different people have different measures of n
  - Why might this be bad?

How Akamai Works



Akamai – Subsequent Requests



Summary

- Simple text-based file exchange protocol
  - Support for status/error responses, authentication, client-side state maintenance, cache maintenance
- Interactions with TCP
  - Connection setup, reliability, state maintenance
  - Persistent connections
- How to improve performance
  - Persistent connections
  - Caching
  - Replication
- State
  - Deal with maintenance & consistency

Caching Proxies – Sources for Misses

- Capacity
  - How large a cache is necessary or equivalent to infinite
  - On disk vs. in memory → typically on disk
- Compulsory
  - First time access to document
  - Non-cacheable documents
    - CGI-scripts
    - Personalized documents (cookies, etc)
    - Encrypted data (SSL)
- Consistency
  - Document has been updated/expired before reuse
- Conflict
  - No such misses

Naming Based

- Round-robin
  - Randomly choose replica
  - Avoid hot-spots
- [Semi-]static metrics
  - Geography
  - Route metrics
  - How well would these work?
- Predicted application performance
  - How to predict?
  - Only have limited info at name resolution