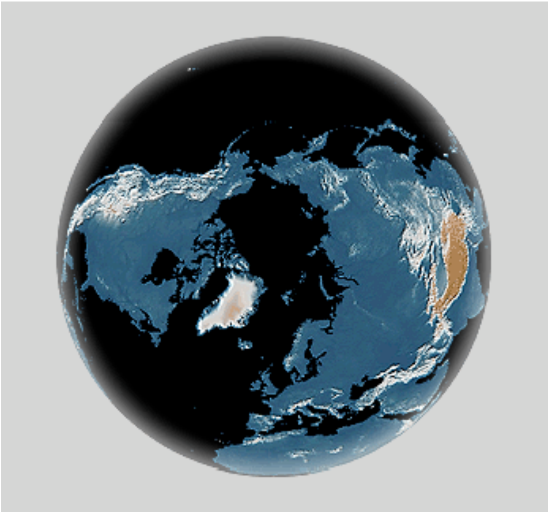


Web Content Delivery

Reading: Section 9.1.2 and 9.4.3

COS 461: Computer Networks
Spring 2009 (MW 1:30-2:50 in CS105)

Mike Freedman
Teaching Assistants: Wyatt Lloyd and Jeff Terrace
http://www.cs.princeton.edu/courses/archive/spring09/cos461/

Outline

- HTTP review
- Persistent HTTP
- HTTP caching
- Proxying and content distribution networks
  – Web proxies
  – Hierarchical networks and Internet Cache Protocol (ICP)
  – Modern distributed CDNs (Akamai)

HTTP Basics (Review)

- 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 info about past client requests

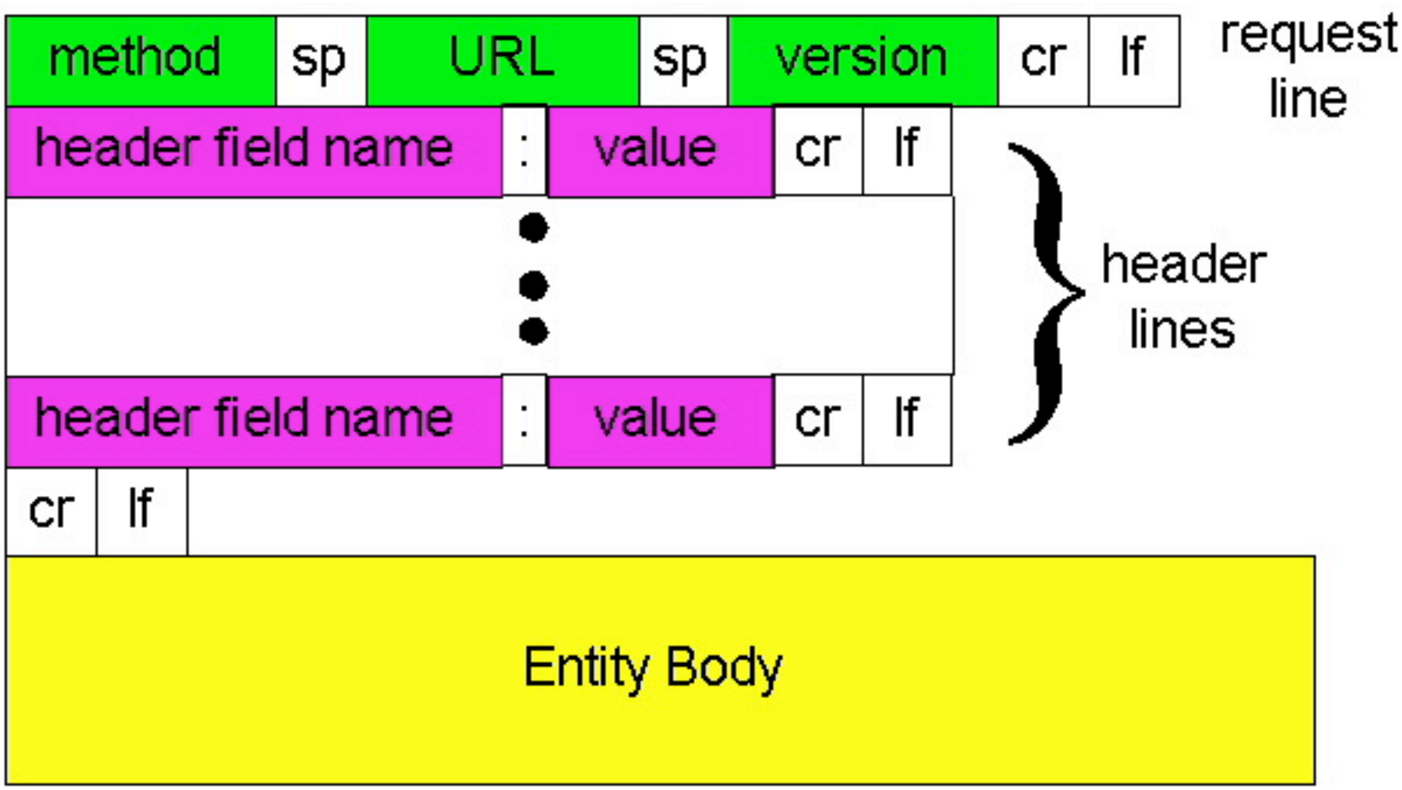
HTTP Request

- 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 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

- 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

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

…..

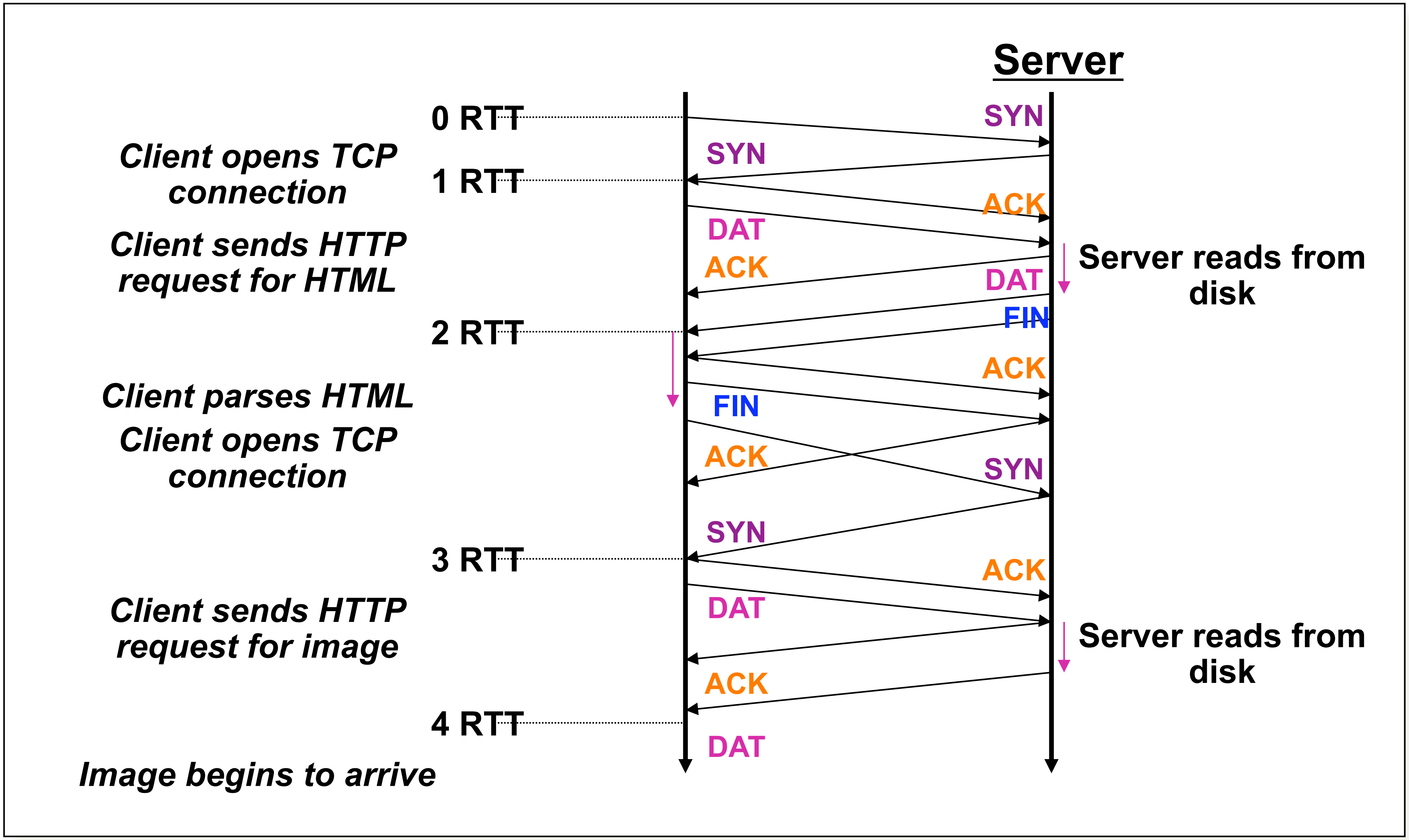
How to Mark End of Message?

- Content-Length
  - Must know size of transfer in advance
- Close connection
  - Only server can do this
- Implied length
  - E.g., 304 never have body content
- Transfer-Encoding: chunked (HTTP/1.1)
  - After headers, each chunk is content length in hex, CRLF, then body. Final chunk is length 0.

Outline

- HTTP review
- Persistent HTTP
- HTTP caching
- Proxying and content distribution networks
  - Web proxies
  - Hierarchical networks and Internet Cache Protocol (ICP)
  - Modern distributed CDNs (Akamai)

Single Transfer Example



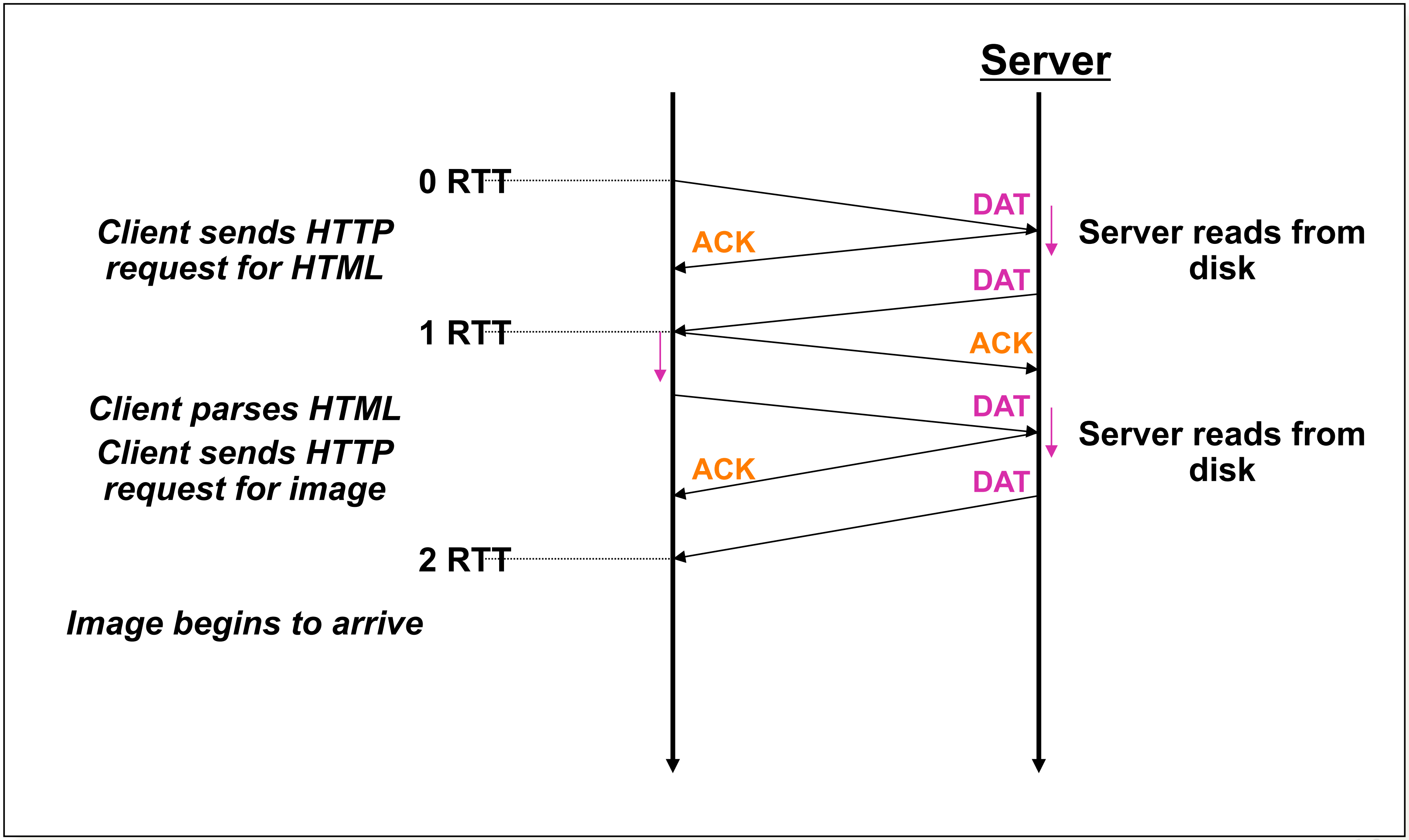
Problems with simple model

- Multiple connection setups
  - Three-way handshake each time
- 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 forced to keep TIME_WAIT connection state

TCP Interaction: Short Transfers

- Multiple connection setups
  - Three-way handshake each time
- Round-trip time estimation
  - Maybe large at the start of a connection (e.g., 3 seconds)
  - Leads to latency in detecting lost packets
- Congestion window
  - Small value at beginning of connection (e.g., 1 MSS)
  - May not reach a high value before transfer is done
- Detecting packet loss
  - Timeout: slow ☹
  - Duplicate ACK
    - Requires many packets in flight
    - Which doesn't happen for very short transfers ☹

Persistent Connection Example



Persistent HTTP

Non-persistent HTTP issues:

- Requires 2 RTTs per object
- OS must allocate 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 object

Persistent with pipelining:

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

Outline

- HTTP review
- Persistent HTTP
- HTTP caching
- Proxying and content distribution networks
  - Web proxies
  - Hierarchical networks and Internet Cache Protocol (ICP)
  - Modern distributed CDNs (Akamai)

HTTP Caching

- Clients often cache documents
  - When should origin be checked for changes?
  - Every time? Every session? Date?
- HTTP includes caching information in headers
  - HTTP 0.9/1.0 used: “Expires: <date>”; “Pragma: no-cache”
  - HTTP/1.1 has “Cache-Control”
    - “No-Cache”, “Private”, “Max-age: <seconds>”
    - “E-tag: <opaque value>”
- If not expired, use cached copy
- If expired, use condition GET request to origin
  - “If-Modified-Since: <date>”, “If-None-Match: <etag>”
  - 304 (“Not Modified”) or 200 (“OK”) response

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 (compat; 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

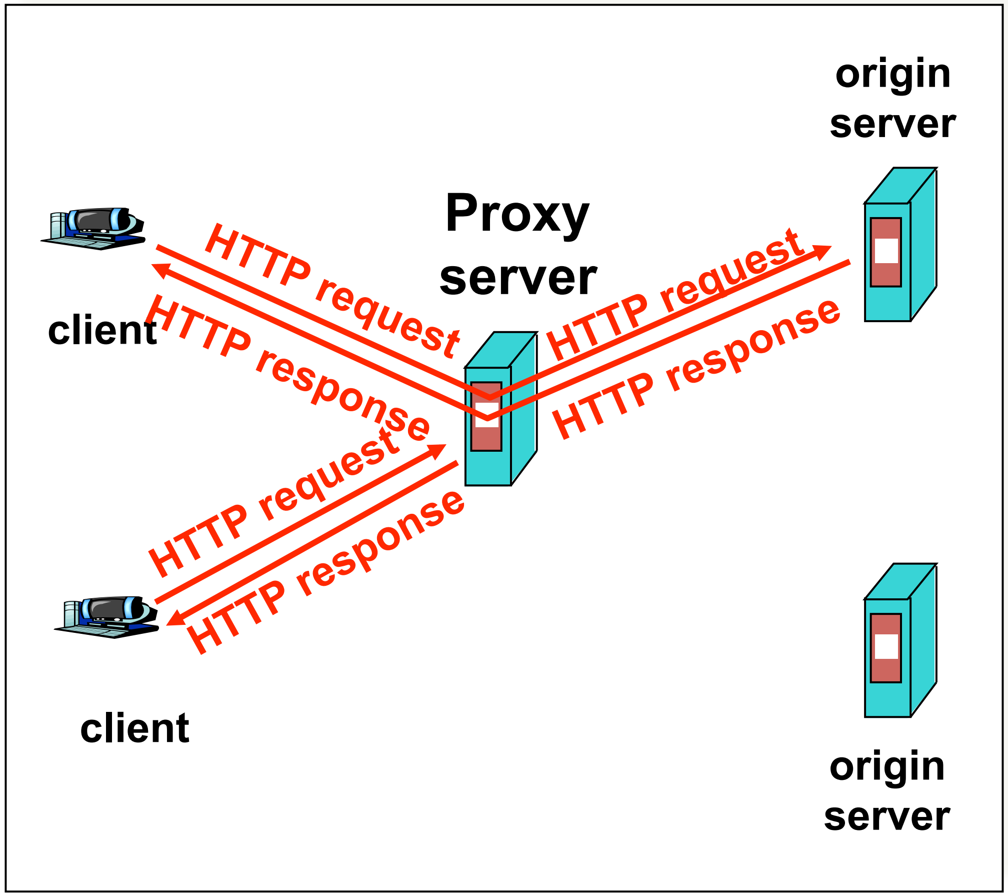
Connection: Keep-Alive

Keep-Alive: timeout=15, max=100

ETag: "7a11f-10ed-3a75ae4a”

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, then returns to client



Caching Example (1)

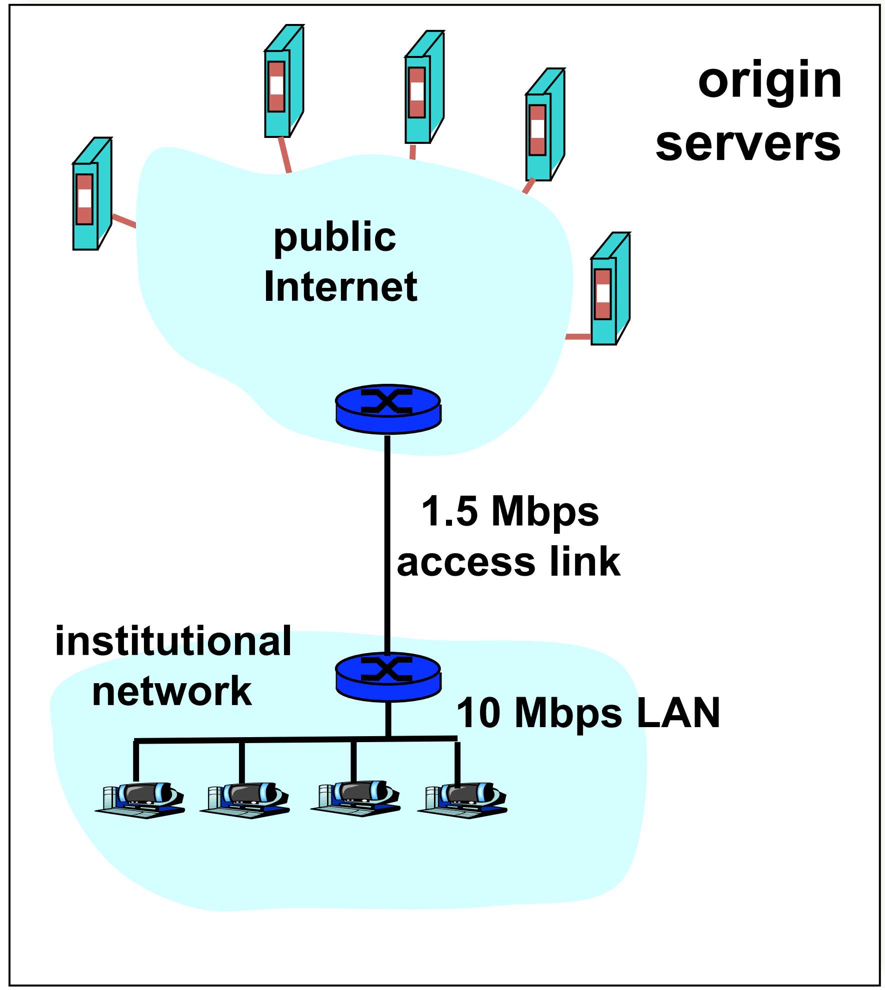
Assumptions

- Average object size = 100K bits
- Avg. request rate from browsers to origin servers = 15/sec
- Delay from institutional router to any origin server and back to router = 2 sec

Consequences

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

= 2 sec + minutes + milliseconds



Caching Example (2)

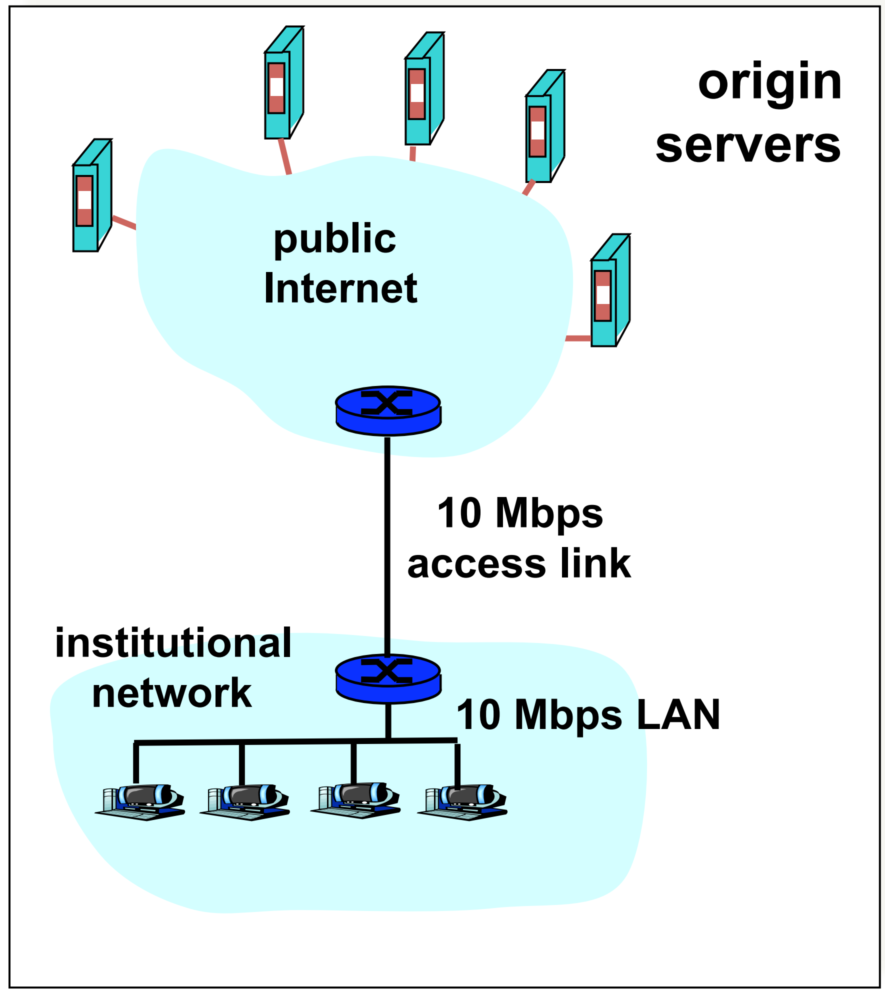
Possible Solution

- 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 + minutes + milliseconds



Caching Example (3)

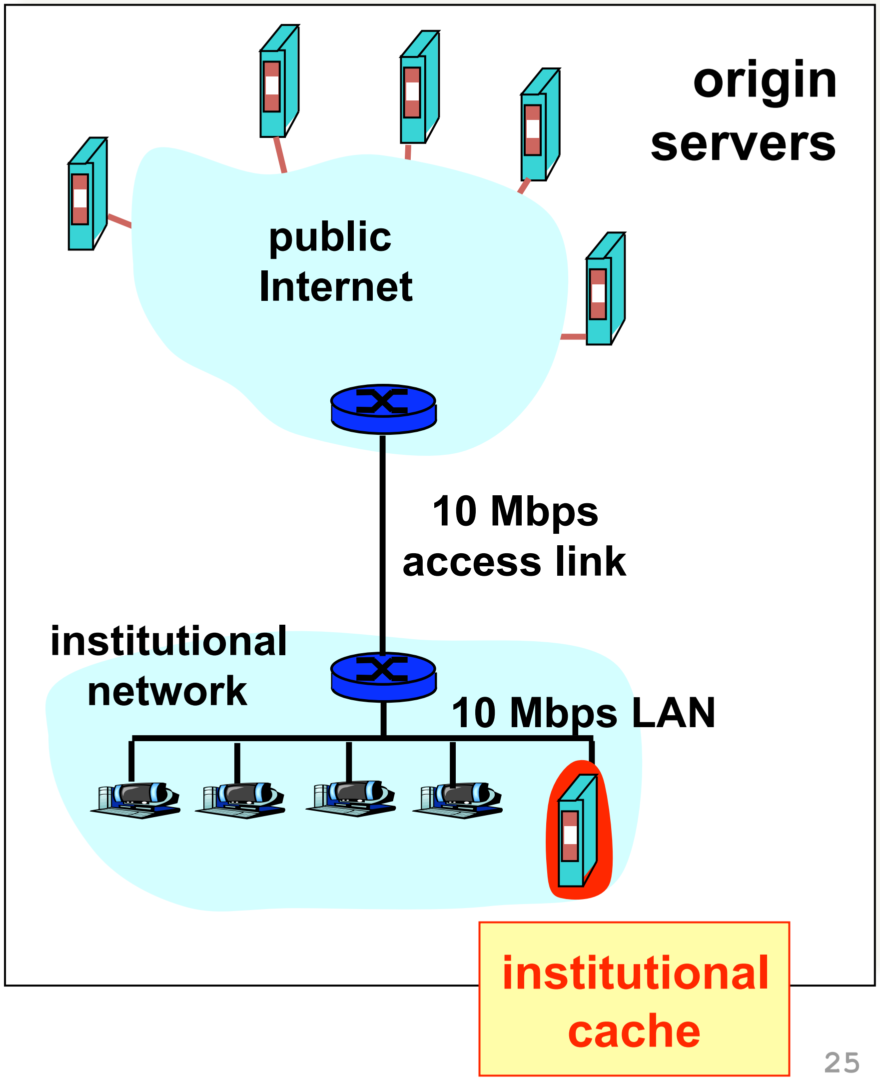
Install Cache

- Support hit rate is 40%

Consequences

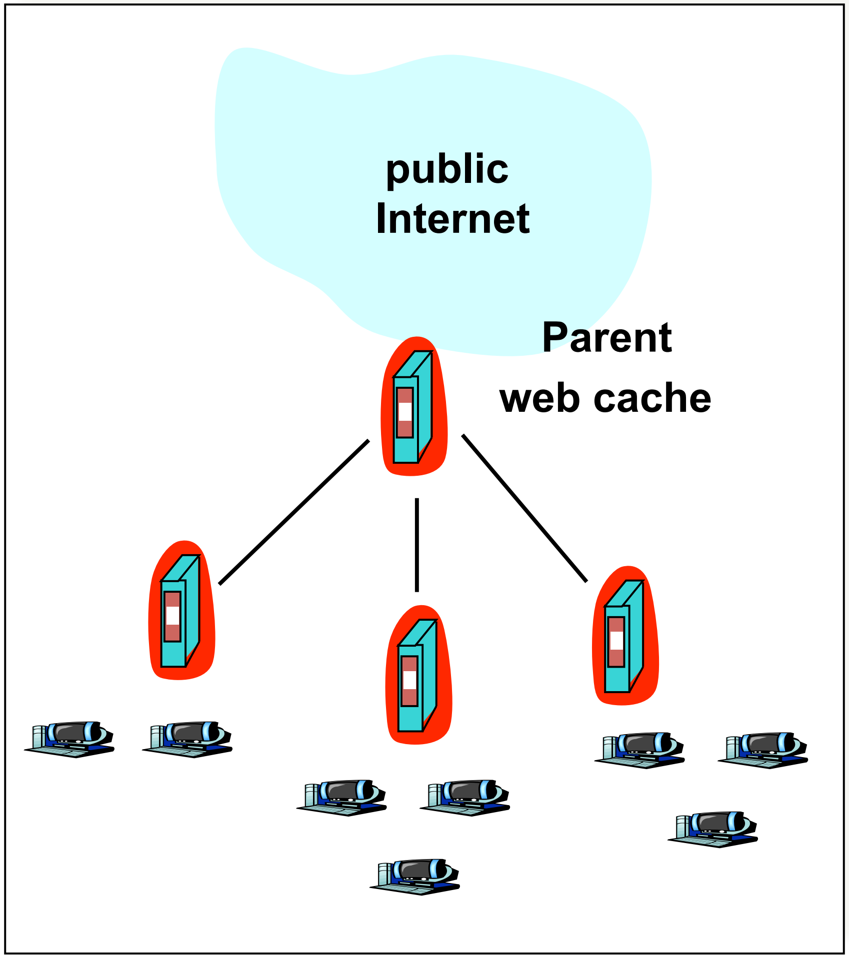
- 40% requests satisfied almost immediately (say 10 msec)
- 60% requests satisfied by origin
- Utilization of access link down to 60%, yielding negligible delays
- Weighted average of delays

= .6*2 s + .4*10 ms < 1.3 s



When a single cache isn't enough

- What if the working set is > proxy disk?
  - Cooperation!
- A static hierarchy
  - Check local
  - If miss, check siblings
  - If miss, fetch through parent
- Internet Cache Protocol (ICP)
  - ICPv2 in RFC 2186 (& 2187)
  - UDP-based, short timeout



Problems

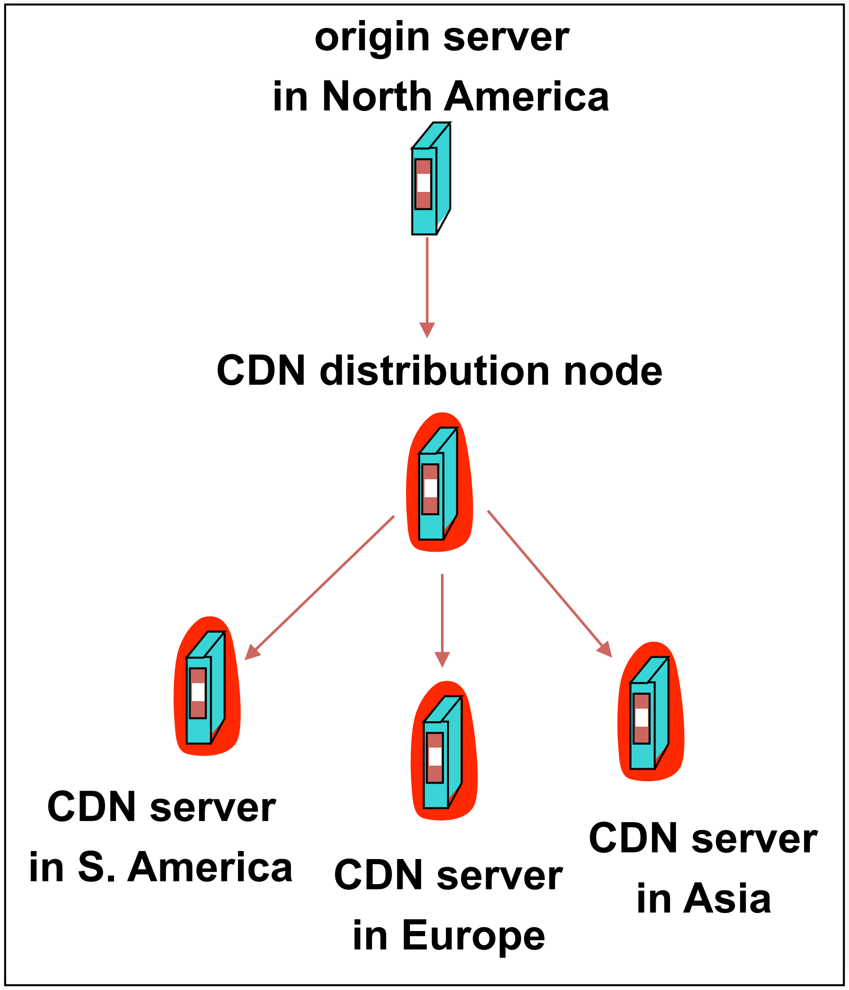
- Significant fraction (>50%?) of HTTP objects uncachable
- Sources of dynamism?
  - Dynamic data: Stock prices, scores, web cams
  - CGI scripts: results based on passed parameters
  - Cookies: results may be based on passed data
  - SSL: encrypted data is not cacheable
  - Advertising / analytics: owner wants to measure # hits
    - Random strings in content to ensure unique counting

Content Distribution Networks (CDNs)

- Content providers are CDN customers

<u>Content replication</u>

- CDN company installs thousands of servers throughout Internet
  - In large datacenters
  - Or, close to users
- CDN replicates customers' content
- When provider updates content, CDN updates servers



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
  - As part of application: HTTP redirect
  - As part of naming: DNS

Trade-offs between approaches

- Routing based (IP anycast)
  - Pros:
  - Cons:
- Application based (HTTP redirects)
  - Pros:
  - Cons:
- Naming based (DNS selection)
  - Pros:
  - Cons:

Trade-offs between approaches

- Routing based (IP anycast)
  - Pros: Transparent to clients, works when browsers cache failed addresses, circumvents many routing issues
  - Cons: Little control, complex, scalability, TCP can't recover, …
- Application based (HTTP redirects)
  - Pros:
  - Cons:
- Naming based (DNS selection)
  - Pros:
  - Cons:

Trade-offs between approaches

- Routing based (IP anycast)
  - Pros: Transparent to clients, works when browsers cache failed addresses, circumvents many routing issues
  - Cons: Little control, complex, scalability, TCP can't recover, …
- Application based (HTTP redirects)
  - Pros: Application-level, fine-grained control
  - Cons: Additional load and RTTs, hard to cache
- Naming based (DNS selection)
  - Pros:
  - Cons:

Trade-offs between approaches

- Routing based (IP anycast)
  - Pros: Transparent to clients, works when browsers cache failed addresses, circumvents many routing issues
  - Cons: Little control, complex, scalability, TCP can't recover, …
- Application based (HTTP redirects)
  - Pros: Application-level, fine-grained control
  - Cons: Additional load and RTTs, hard to cache
- Naming based (DNS selection)
  - Pros: Well-suitable for caching, reduce RTTs
  - Cons: Request by resolver not client, request for domain not URL, hidden load factor of resolver's population
    - Much of this data can be estimated "over time"

Outline

- HTTP review
- Persistent HTTP
- HTTP caching
- Proxying and content distribution networks
  - Web proxies
  - Hierarchical networks and Internet Cache Protocol (ICP)
  - Modern distributed CDNs (Akamai)

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.akamai.net/7/23/cnn.com/af/x.gif>
  - Or, cache.cnn.com, and CNN adds CNAME (alias) for cache.cnn.com → a73.g.akamai.net
- Client resolves aXYZ.g.akamaitech.net hostname

How Akamai Works

- Akamai only replicates static content
  - At least, simple version. Akamai also lets sites write code that run on their servers, but that's a pretty different beast

- Modified name contains original file name

- Akamai server is asked for content
  - First checks local cache
  - If not in cache, requests from primary server and caches file

How Akamai Works

- Root server gives NS record for akamai.net
- This nameserver returns NS record for g.akamai.net
  - Nameserver chosen to be in region of client's name server
  - TTL is large
- g.akamai.net nameserver chooses server in region
  - Should try to chose server that has file in cache (How?)
  - Uses aXYZ name and hash
  - TTL is small (Why?)
  - Small modification to before: (Why?)
    - CNAME cache.cnn.com → cache.cnn.com.akamaidns.net
    - CNAME cache.cnn.com.akamaidns.net → a73.g.akamai.net

Simple Hashing

- Given document group XYZ, 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?

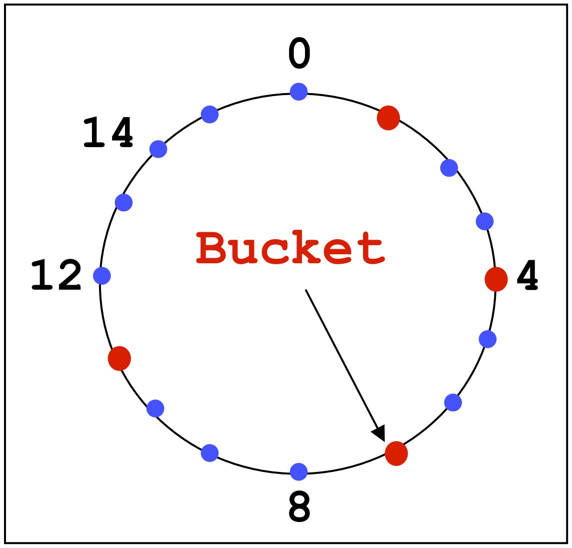
Consistent Hashing

- "view" = subset of all hash buckets that are visible
  - For this conversation, "view" is O(n) neighbors
  - But don't need strong consistency on views

- Desired features
  - Balanced: in any one view, load is equal across buckets
  - Smoothness: little impact on hash bucket contents when buckets are added/removed
  - Spread: small set of hash buckets that may hold an object regardless of views
  - Load: across views, # objects assigned to hash bucket is small

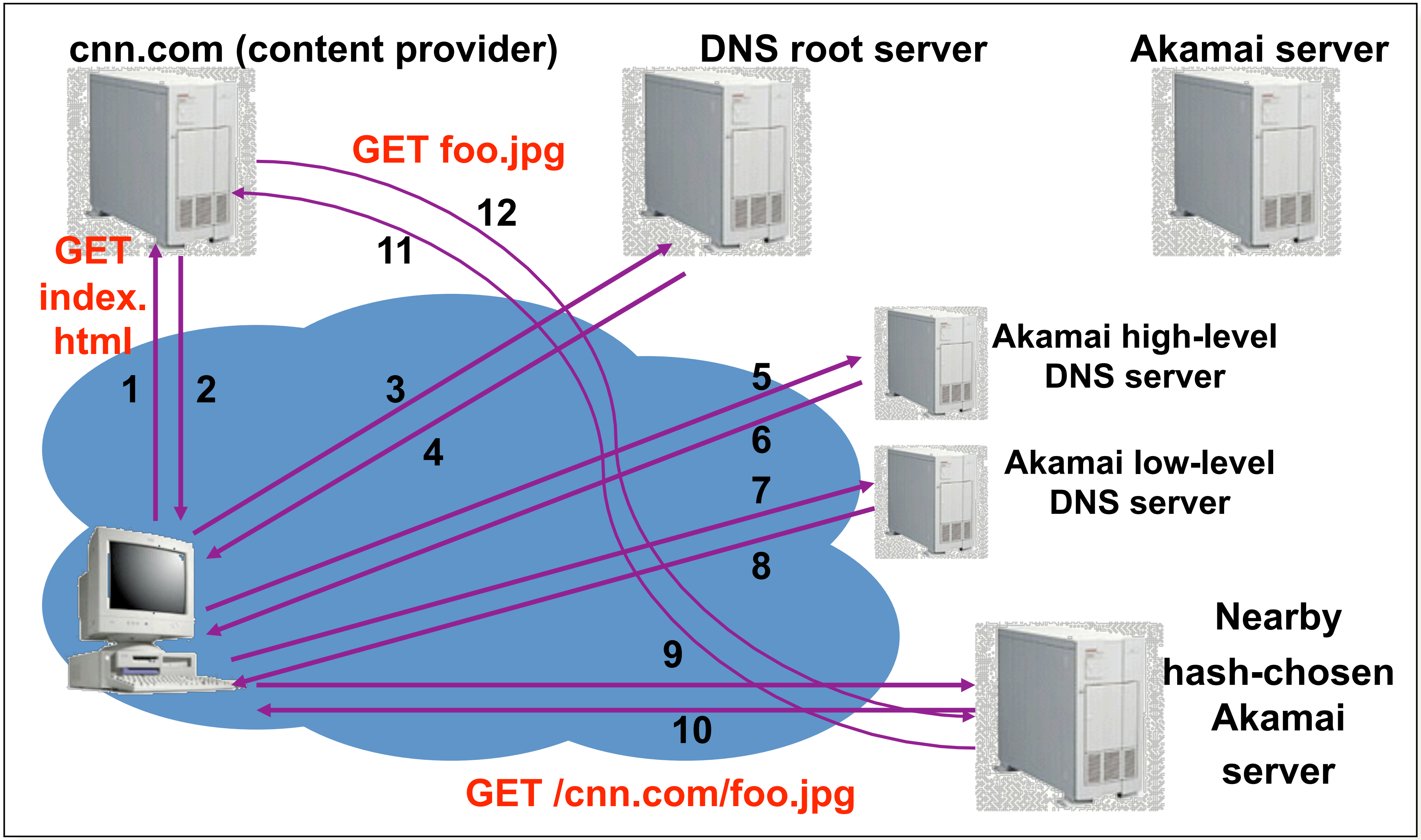
Consistent Hashing

- Construction
  - Assign each of C hash buckets to random points on mod 2^n circle; hash key size = n
  - Map object to random position on circle
  - Hash of object = closest clockwise bucket

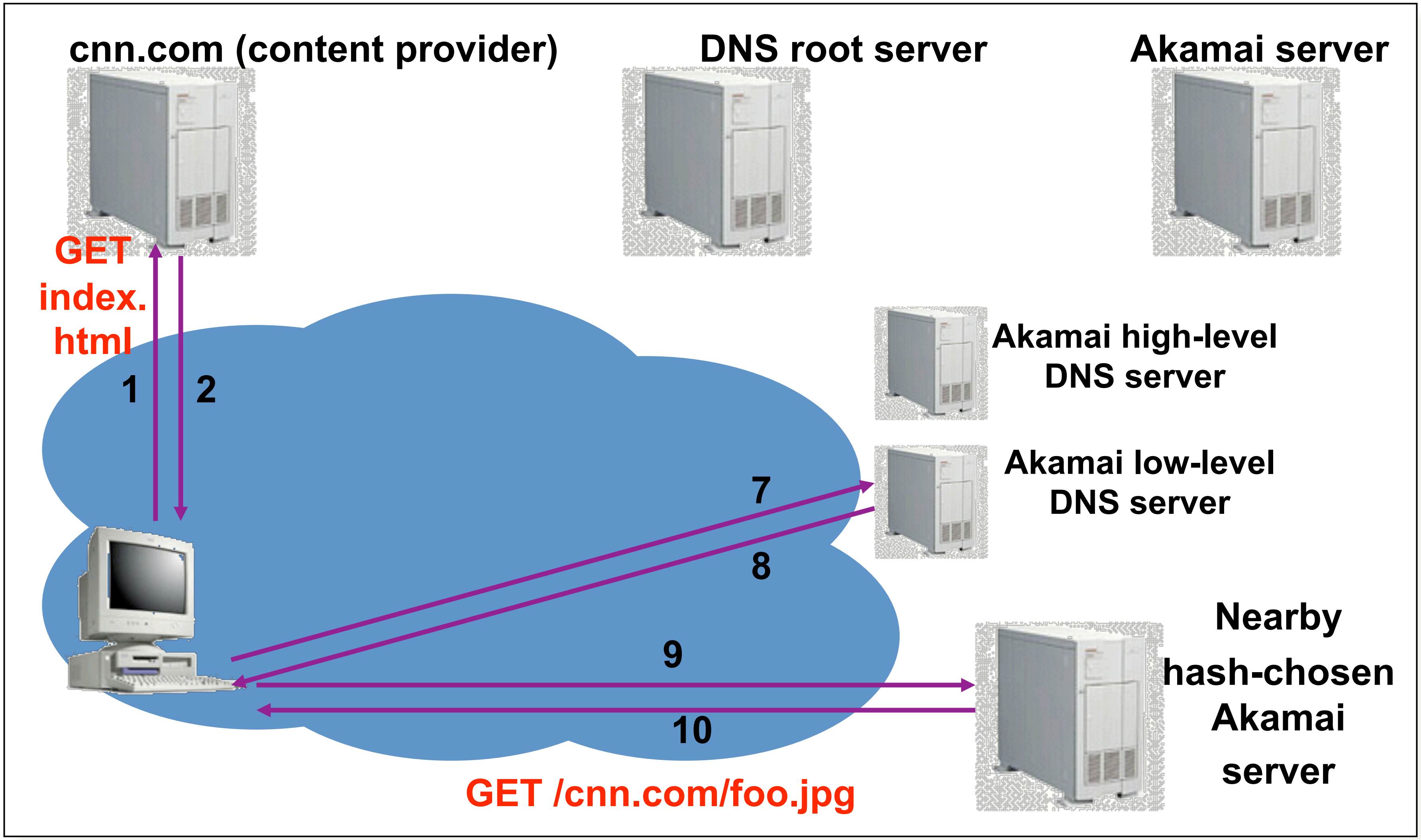


- Desired features
  - Balanced: No bucket responsible for large number of objects
  - Smoothness: Addition of bucket does not cause movement among existing buckets
  - Spread and load: Small set of buckets that lie near object
- Used layer in P2P Distributed Hash Tables (DHTs)

How Akamai Works



How Akamai Works – Already Cached



Summary

- HTTP: 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: Web proxies, cooperative proxies, and CDNs