15-441: Computer Networking

The “Web”

Thomas Harris

(slides from Srini Seshan’s Fall ’01 course)

Overview

- HTTP Basics
- HTTP Fixes
- Web Caches
- Content Distribution Networks

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
- How to mark end of message?
  - Size of message Content-Length
    - Must know size of transfer in advance
  - Delimiter MIME style Content-Type
    - Server must “byte-stuff”
  - Close connection
    - Only server can do this

HTTP Request

- Request line
  - Method
    - GET – return URI
    - HEAD – return headers only of GET response
    - POST – send data to the server (forms, etc.)
  - URI
    - E.g. http://www.seshan.org/index.html with a proxy
    - E.g. /index.html if no proxy
  - HTTP version

HTTP Request

- 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 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.seshan.org
Connection: Keep-Alive
```

HTTP Response

- Status-line
  - HTTP version
  - 3 digit response code
    - 1XX – informational
    - 2XX – success
    - 3XX – redirection
    - 4XX – client error
    - 5XX – server error
  - Reason phrase

HTTP Response

- 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
…..
```

Typical Workload

- Multiple (typically small) objects per page
- Request sizes
  - In one measurement paper median 1946 bytes, mean 13767 bytes
  - Why such a difference? Heavy-tailed distribution
    - Pareto – $p(x) = ak^{a}x^{-(a+1)}$
- File sizes
  - Why different than request sizes?
  - Also heavy-tailed
    - Pareto distribution for tail
    - Lognormal for body of distribution

Typical Workload

- Popularity
  - Zipf distribution ($P = kr^{-1}$)
  - Surprisingly common
- Embedded references
  - Number of embedded objects = pareto
- Temporal locality
  - Modeled as distance into push-down stack
  - Lognormal distribution of stack distances
- Request interarrival
  - Bursty request patterns

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 file signature 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.seshan.org
Connection: Keep-Alive
```

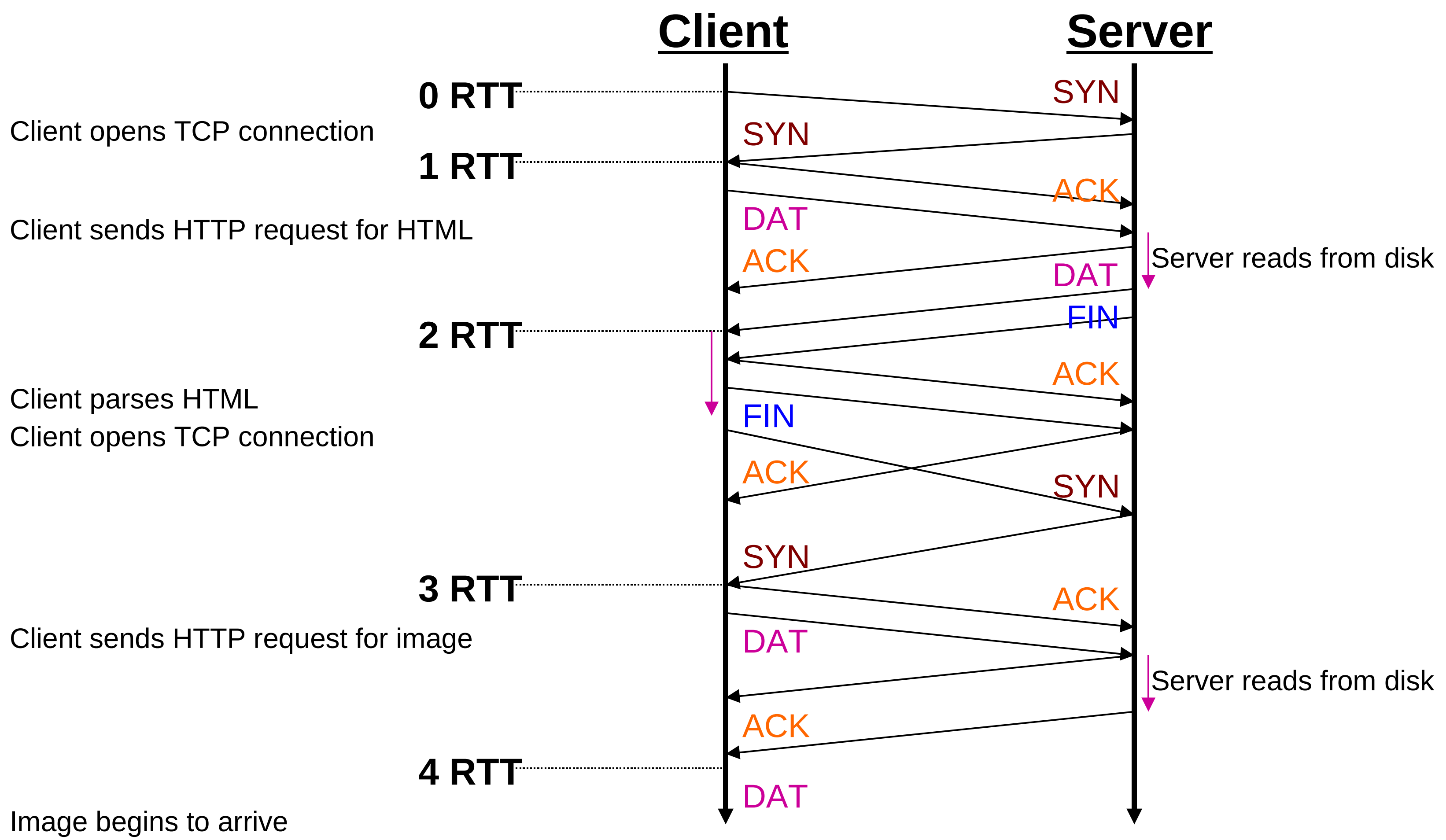
Example Cache Check Response

```
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"
```

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?

Overview

- HTTP Basics
- HTTP Fixes
- Web Caches
- Content Distribution Networks

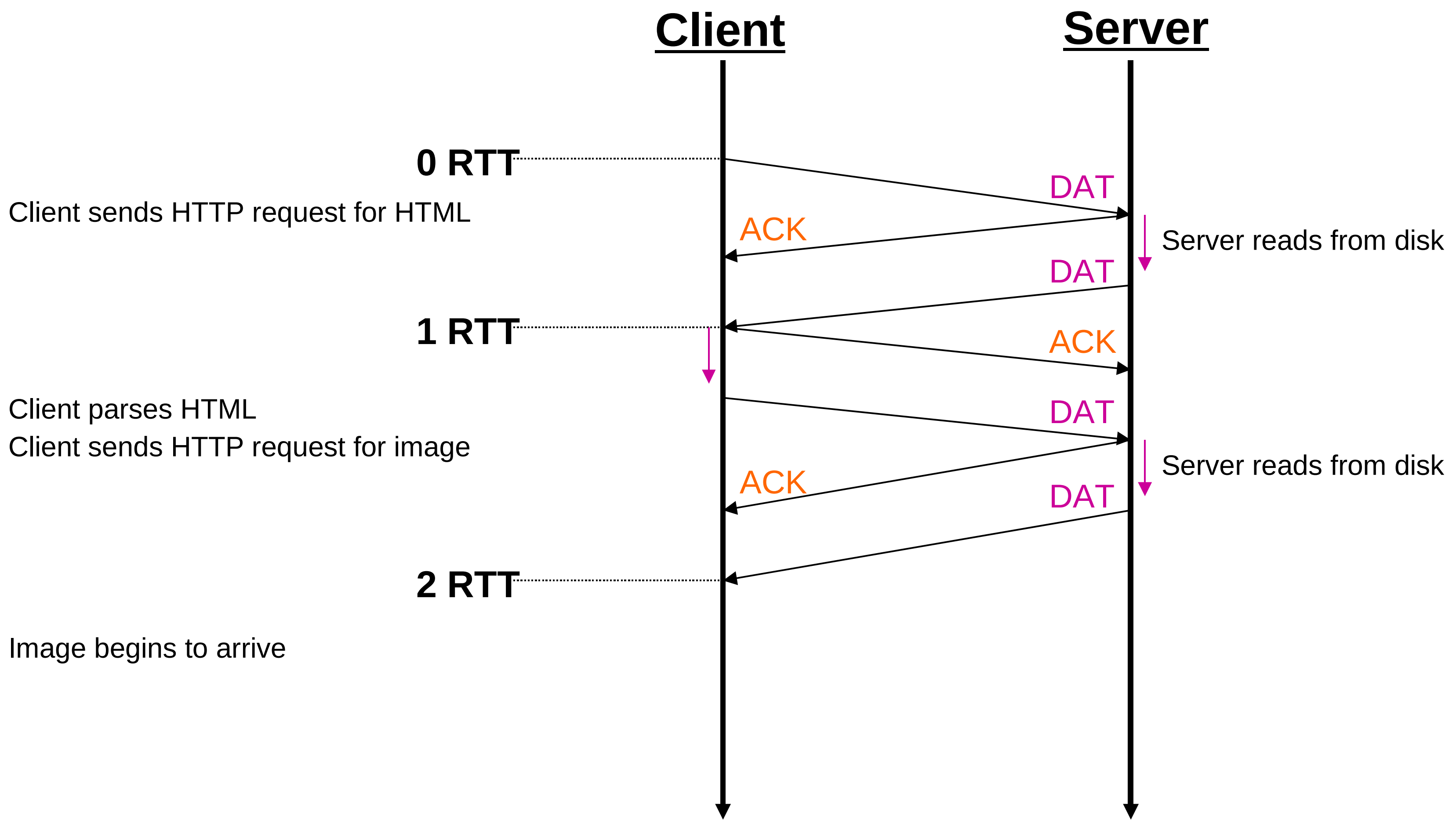
Netscape Solution

- Use multiple concurrent connections to improve response time
  - Different parts of Web page arrive independently
  - Can grab more of the network bandwidth than other users
- Doesn’t necessarily improve response time
  - TCP loss recovery ends up being timeout dominated because windows are small

Persistent Connection Solution

- Multiplex multiple transfers onto one TCP connection
  - Serialize transfers client makes next request only after previous response
- How to demultiplex requests/responses
  - Content-length and delimiter same problems as before
  - Block-based transmission – send in multiple length delimited blocks
  - Store-and-forward – wait for entire response and then use content-length
  - PM95 solution – use existing methods and close connection otherwise

Persistent Connection Example



Persistent Connection Solution

- Serialized requests do not improve interactive response
- Pipelining requests
  - Getall – request HTML document and all embeds
    - Requires server to parse HTML files
    - Doesn’t consider client cached documents
  - Getlist – request a set of documents
    - Implemented as a simple set of GETs
- Prefetching
  - Must carefully balance impact of unused data transfers
  - Not widely used due to poor hit rates

Persistent Connection Performance

- Benefits greatest for small objects
  - Up to 2x improvement in response time
- Server resource utilization reduce due to fewer connection establishments and fewer active connections
- TCP behavior improved
  - Longer connections help adaptation to available bandwidth
  - Larger congestion window improves loss recovery

Remaining Problems

- Application specific solution to transport protocol problems
- Stall in transfer of one object prevents delivery of others
- Serialized transmission
  - Much of the useful information in first few bytes
  - Can “packetize” transfer over TCP
    - HTTP 1.1 recommends using range requests
    - MUX protocol provides similar generic solution
- Solve the problem at the transport layer
  - Fix TCP so it works well with multiple simultaneous connections

Overview

- HTTP Basics
- HTTP Fixes
- Web Caches
- Content Distribution Networks

Web Caching

- Why cache HTTP objects?
  - Reduce client response time
  - Reduce network bandwidth usage
    - Wide area vs. local area use
  - These two objectives are often in conflict
    - May do exhaustive local search to avoid using wide area bandwidth
    - Prefetching uses extra bandwidth to reduce client response time

Web Proxies

- Also used for security
  - Proxy is only host that can access Internet
  - Administrators makes sure that it is secure
- Performance
  - How many clients can a single proxy handle?
- Caching
  - Provides a centralized coordination point to share information across clients
- How to index
  - Early caches used file system to find file
  - Metadata now kept in memory on most caches

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 issue

Cache Hierarchies

- Use hierarchy to scale a proxy to more than limited population
  - Why?
    - Larger population = higher hit rate
    - Larger effective cache size
  - Why is population for single proxy limited?
    - Performance, administration, policy, etc.
- NLANR cache hierarchy
  - Most popular
  - 9 top level caches
  - Internet Cache Protocol based (ICP)
  - Squid/Harvest proxy

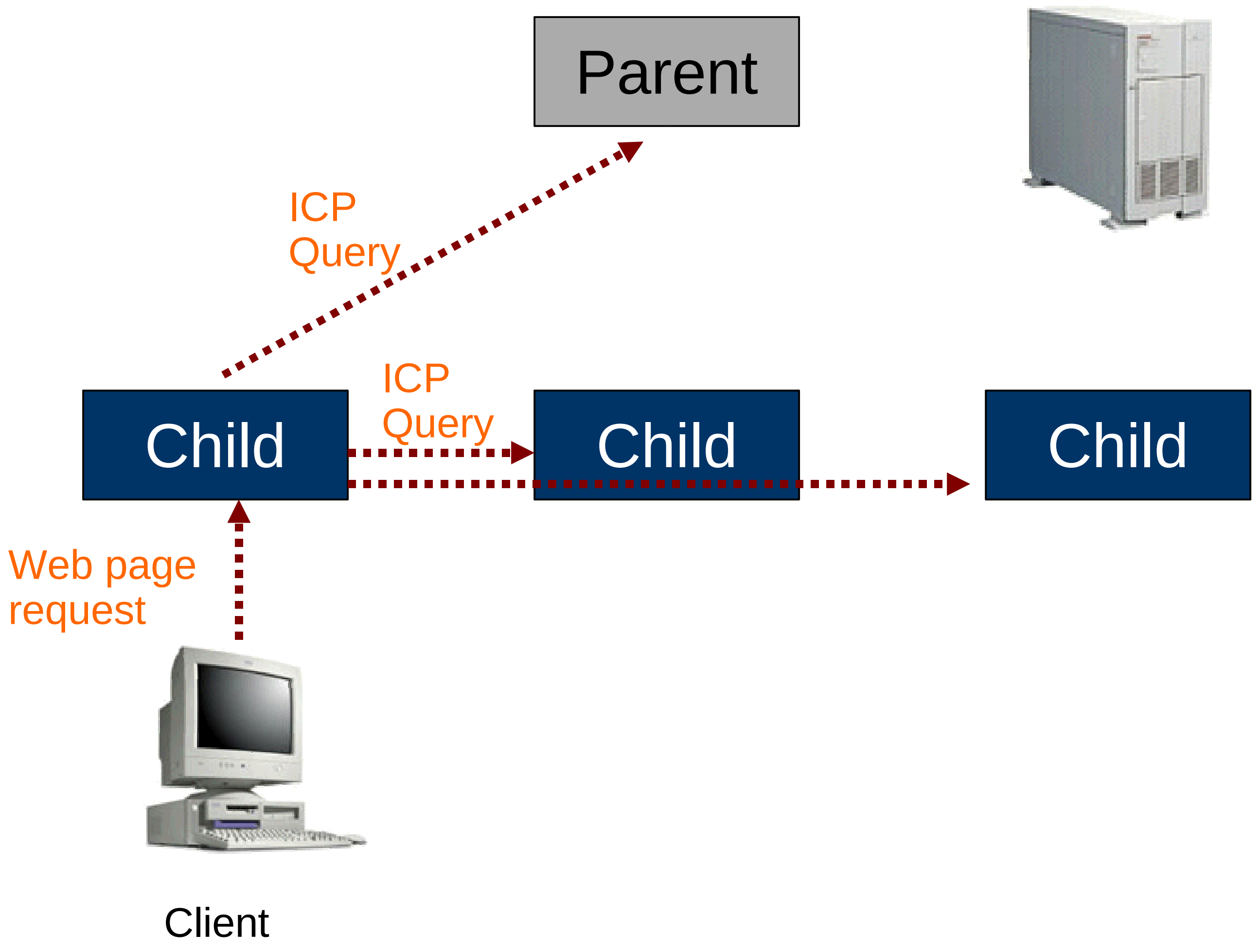
ICP

- Simple protocol to query another cache for content
- Uses UDP – why?
- ICP message contents
  - Type – query, hit, hit_obj, miss
  - Other – identifier, URL, version, sender address (is this needed?)
  - Special message types used with UDP echo port
    - Used to probe server or “dumb cache”
- Transfers between caches still done using HTTP

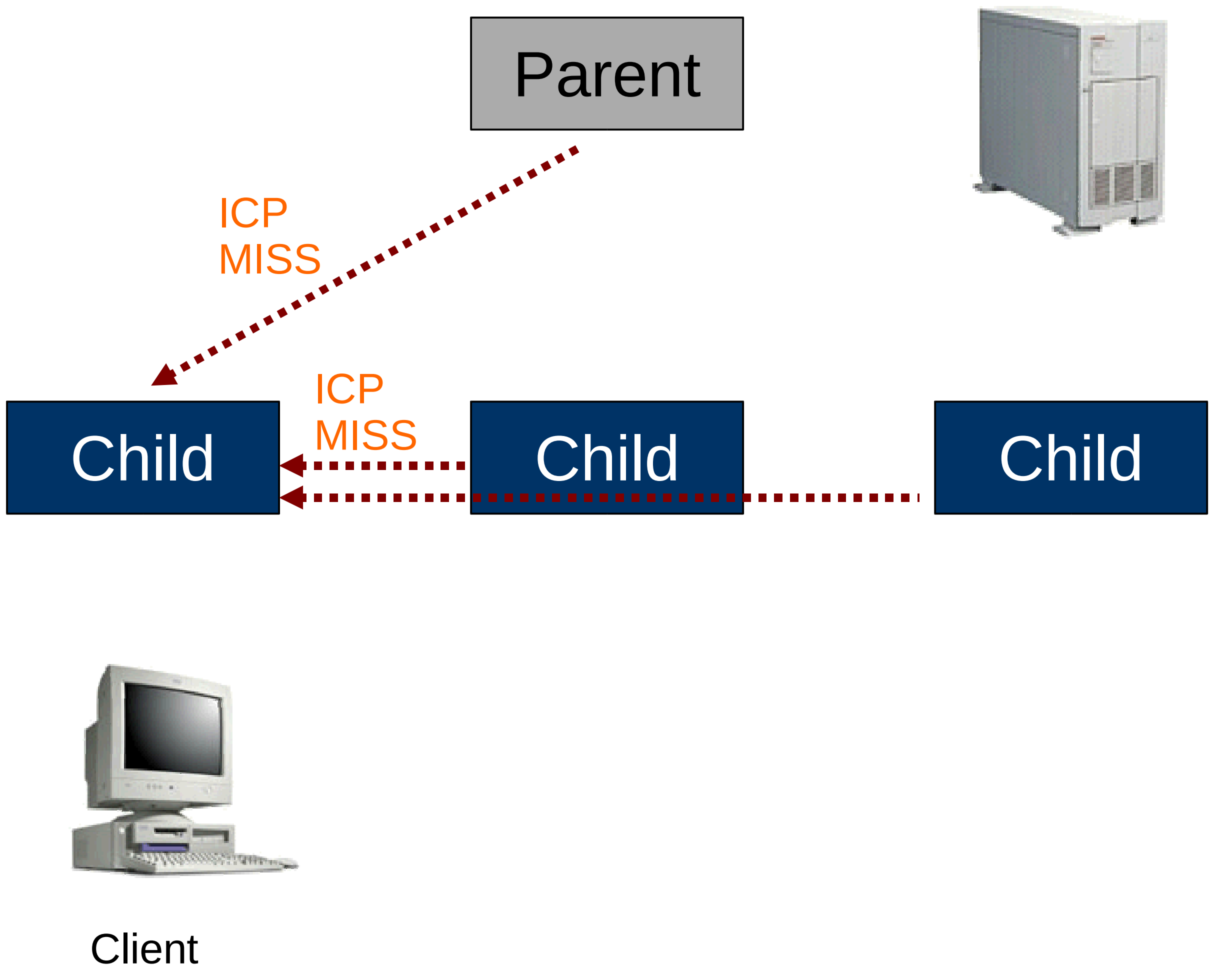
Squid Cache ICP Use

- Upon query that is not in cache
  - Sends ICP_Query to each peer (or ICP_Decho to echo port of peer caches that do not speak ICP)
  - May also send ICP_Secho to origin server's echo port
  - Sets time to short period (default 2 sec)
- Peer caches process queries and return either ICP_Hit or ICP_Miss
- Proxy begins transfer upon reception of ICP_Hit, ICP_Decho or ICP_Secho
- Upon timer expiration, proxy request object from closest (RTT) parent proxy
  - Would be better to direct to parent that is towards origin server

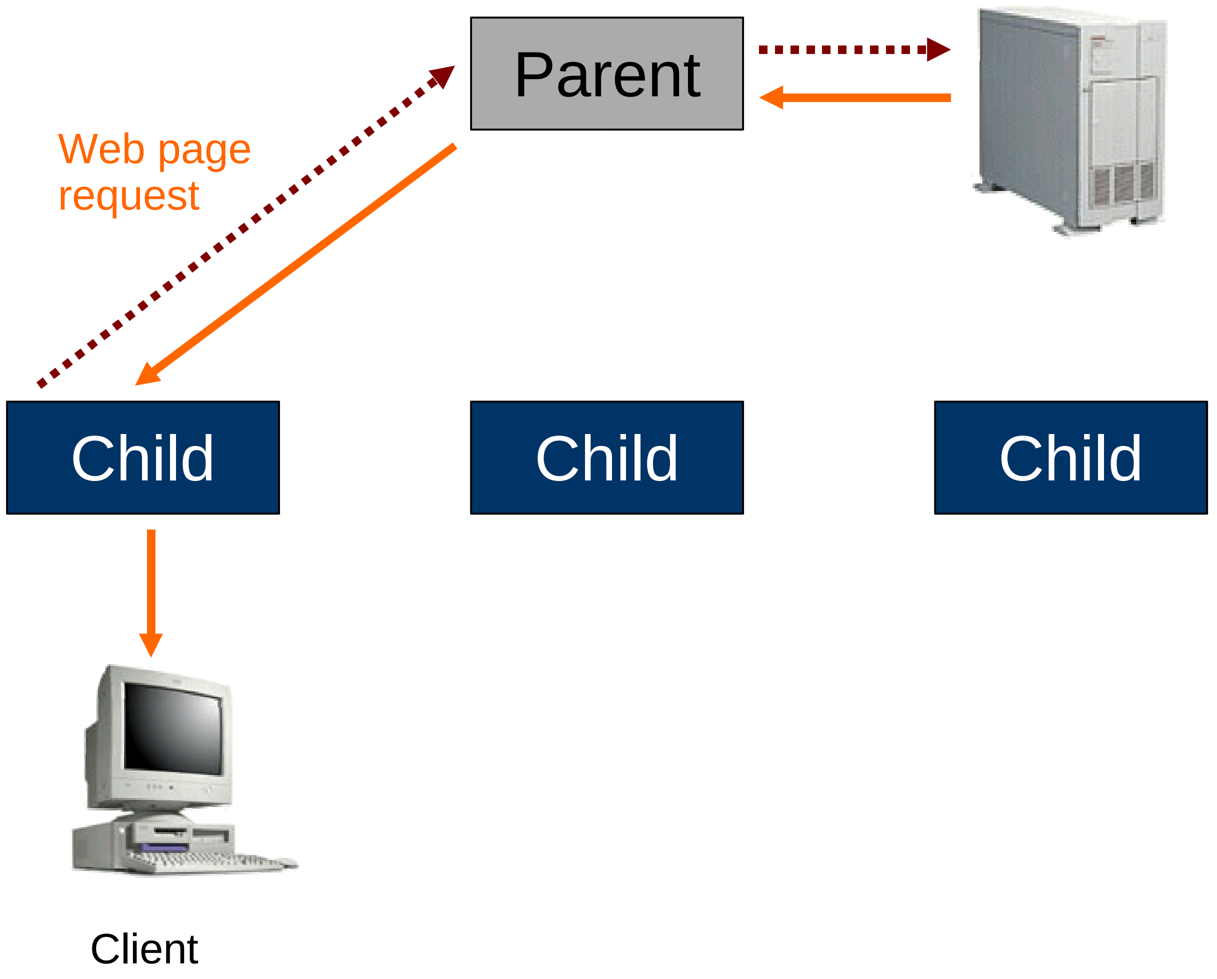
Squid


Parent
ICP Query
ICP Query
Child
Child
Child
Web page request
Client

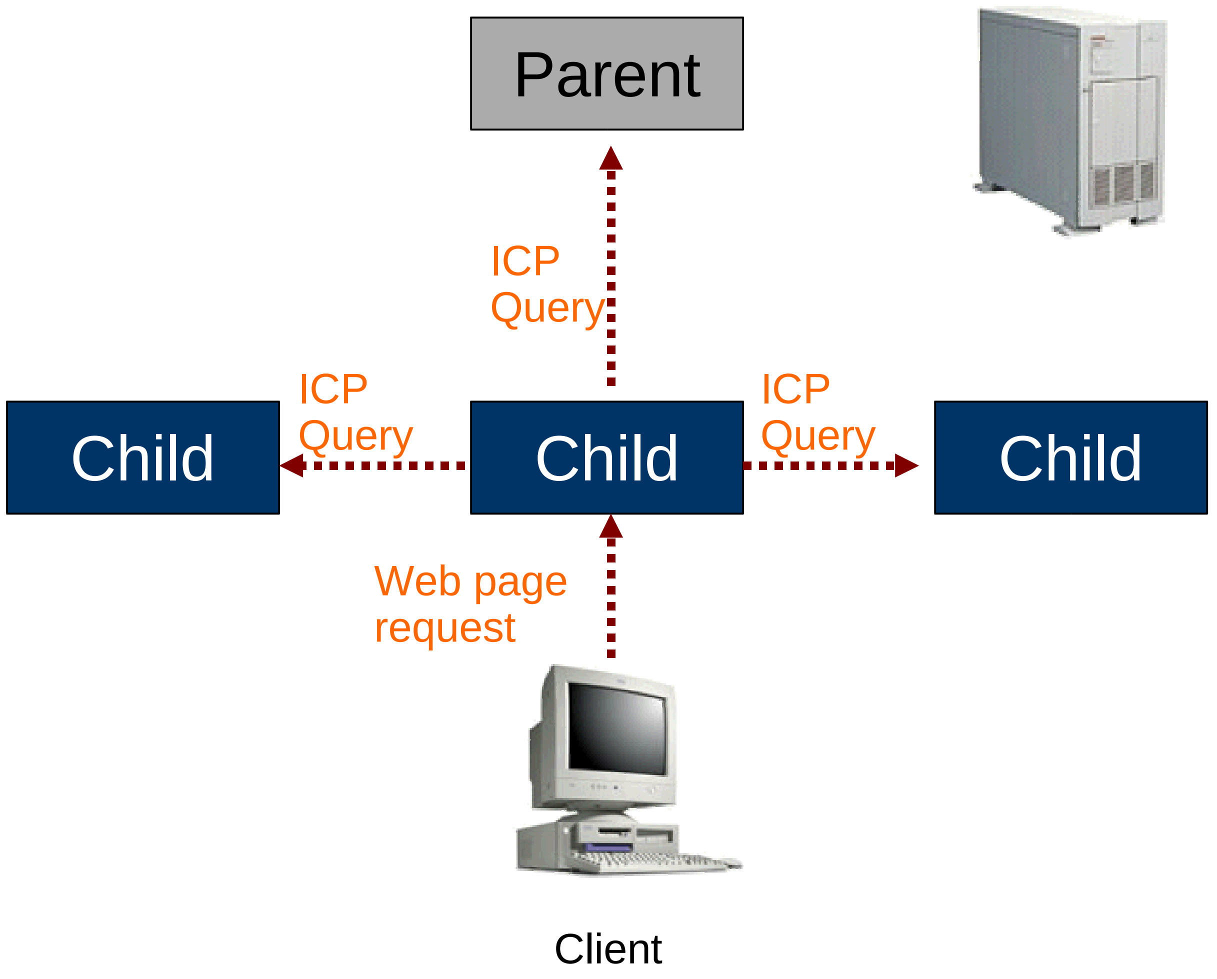
Squid



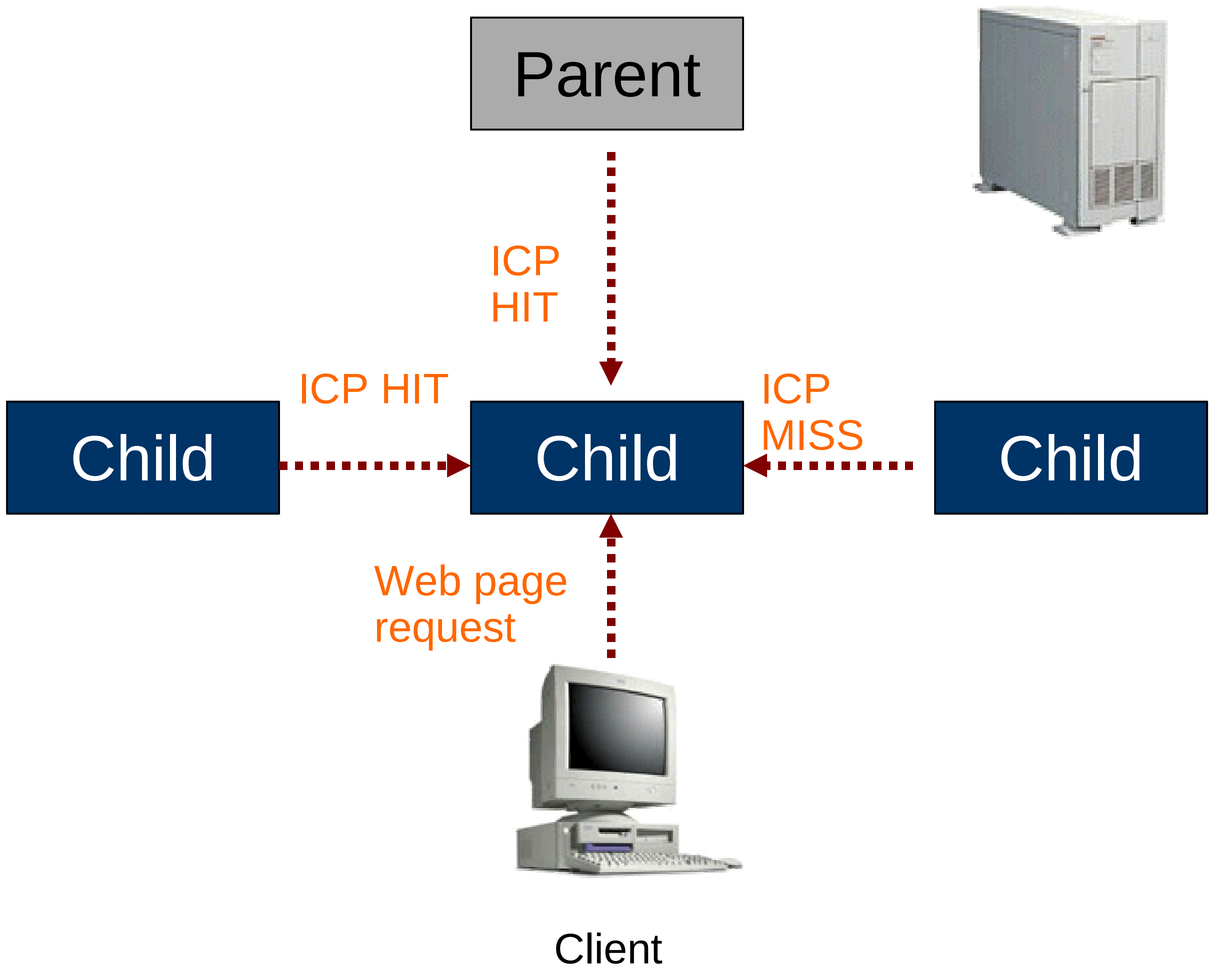
Squid



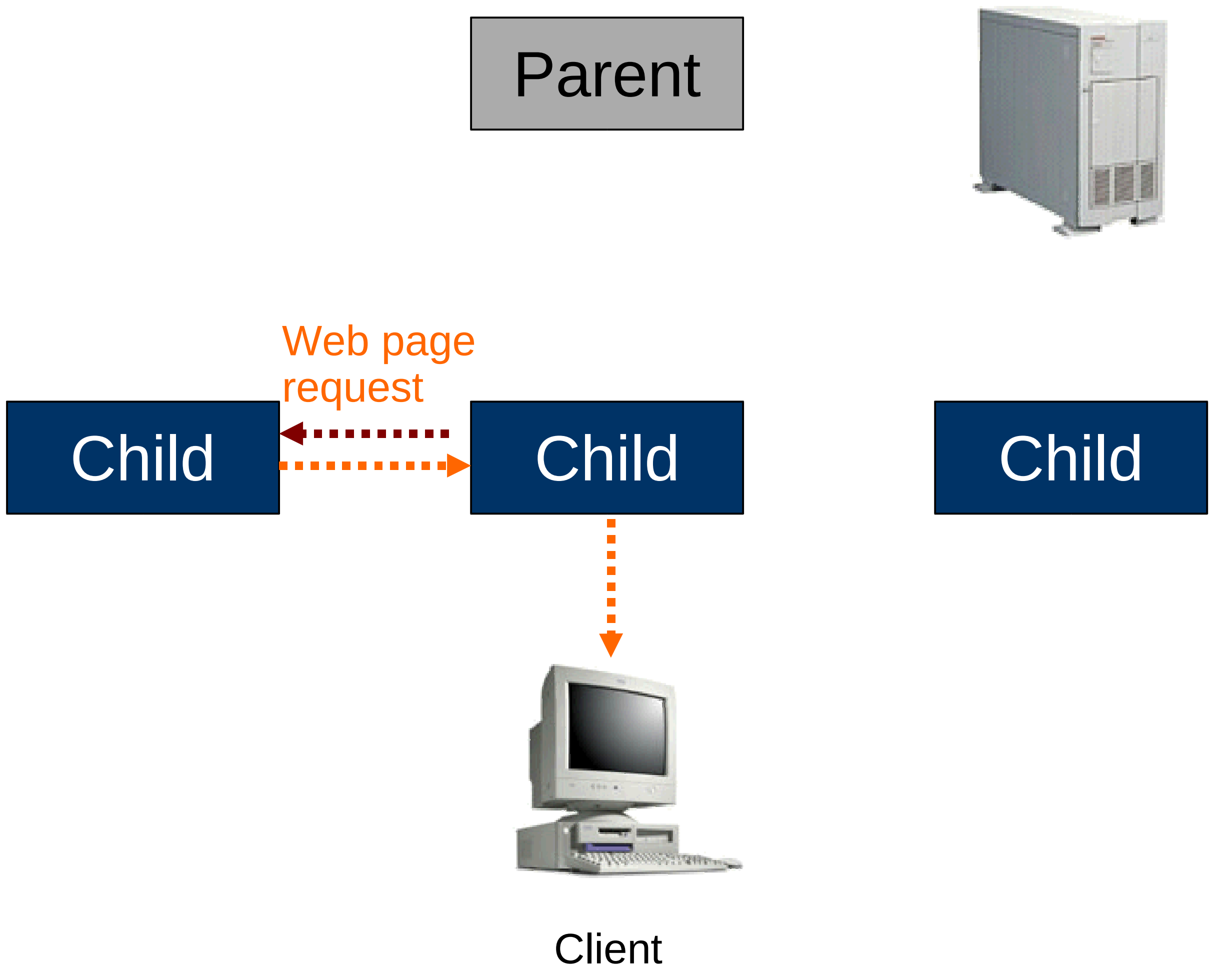
Squid


Parent
ICP Query
ICP Query
ICP Query
Child
Child
Child
Web page request
Client

Squid


Parent
ICP HIT
ICP HIT
ICP MISS
Child
Child
Child
Web page request
Client

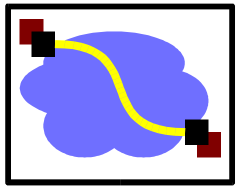
Squid



ICP vs HTTP

- Why not just use HTTP to query other caches?
- ICP is lightweight – positive and negative
  - Makes it easy to process quickly
  - Caches may process many more ICP requests than HTTP requests
  - HTTP has many functions that are not supported by ICP
  - ICP does not evolve with HTTP changes
  - Adds extra RTT to any proxy-proxy transfer

Optimal Cache Mesh Behavior



- Minimize number of hops through mesh
  - Each hop add significant latency
    - ICP hops can cost a 2 sec timeout each!
    - Strict hierarchies cost disk lookup, etc.
  - Especially painful for misses
- Share across many users and scale to many caches
  - ICP does not scale to a large number of peers
- Cache and fetch data close to clients

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?

Proxy Implementation Problems

- Aborted transfers
  - Many proxies transfer entire document even though client has stopped eliminates saving of bandwidth
- Making objects cacheable
  - Proxy’s apply heuristics cookies don’t apply to some objects, guesswork on expiration
  - May not match client behavior/desires
- Client misconfiguration
  - Many clients have either absurdly small caches or no cache
- How much would hit rate drop if clients did the same things as proxies

Questions – Population Size

- How does population size affect hit rate?
- Critical to understand usefulness of hierarchy or placement of caches
- Issues: frequency of access vs. frequency of change (ignore working set size $\rightarrow$ infinite cache)
- UW/Msoft measurement $\rightarrow$ hit rate rises quickly to about 5000 people and very slowly beyond that
- Proxies/Hierarchies don't make much sense for populations > 5000
  - Single proxies can easily handle such populations
  - Hierarchies only make sense for policy/administrative reasons

Questions – Common Interests

- Do different communities have different interests?
  - I.e. do CS and English majors access same pages? IBM and Pepsi workers?
- Has some impact UW departments have about 5% higher hit rate than randomly chosen UW groups
  - Many common interests remain
- Is this true in general? UW students have more in common than IBM & Pepsi workers
- Some related observations
  - Geographic caching – server traces have shown that there is geographic locality to interest
  - UW & MS hierarchy performance is bad – could be due to size or interests?

Overview

- HTTP Basics
- HTTP Fixes
- Web Caches
- Content Distribution Networks

CDN

- 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
    - Discussed in DNS/server selection lecture
    - DNS, HTTP 304 response, anycast, etc.
- 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.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
- Akamai server is asked for content
  - First checks local cache
  - If not in cache, requests file from primary server and caches file

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 choses server in region
  - Should try to chose server that has file in cache - How to choose?
  - Uses aXYZ name and consistent hash
  - TTL is small

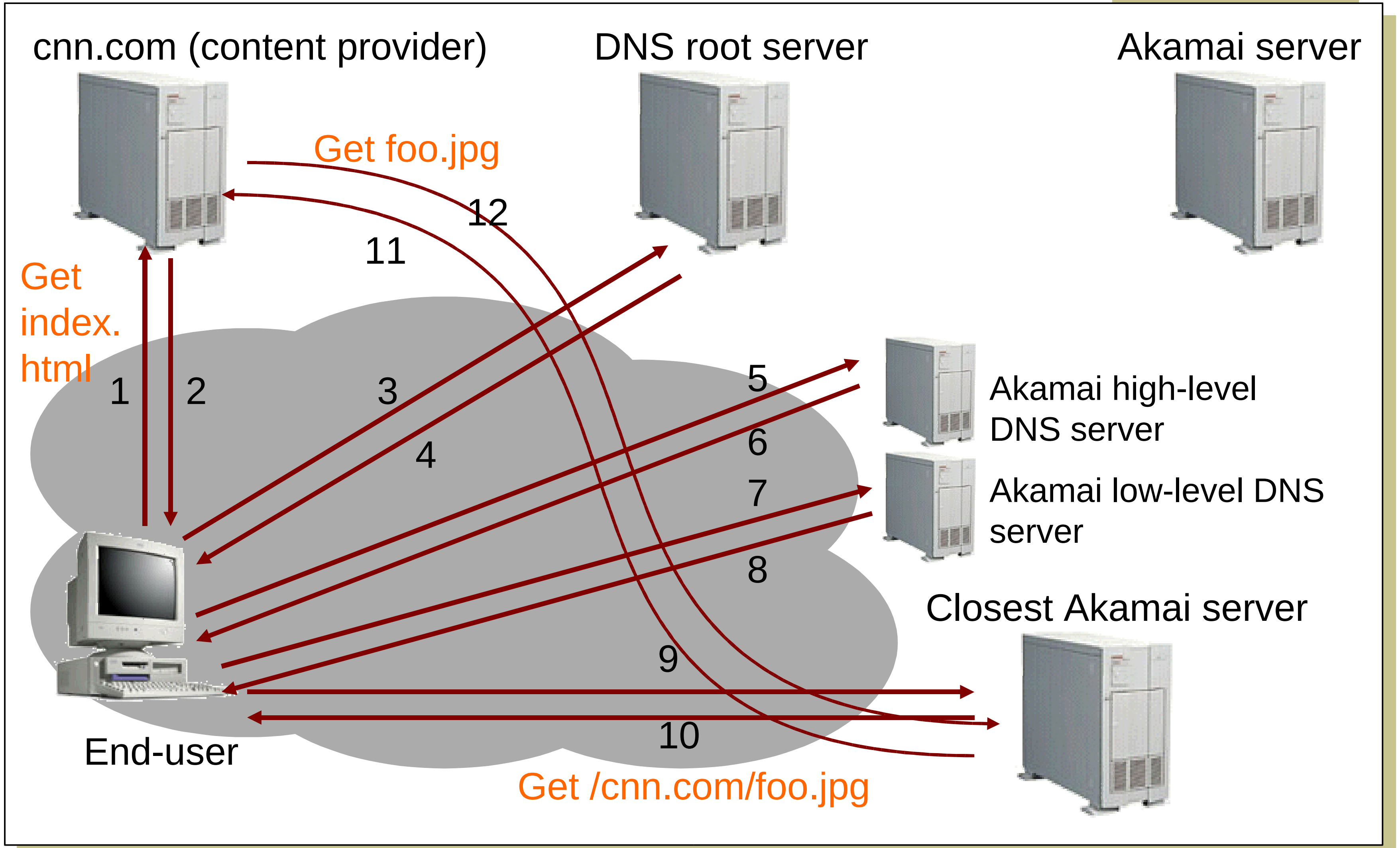
Consistent Hash

- “view” = subset of all hash buckets that are visible
- Desired features
  - 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 all views # of objects assigned to hash bucket is small

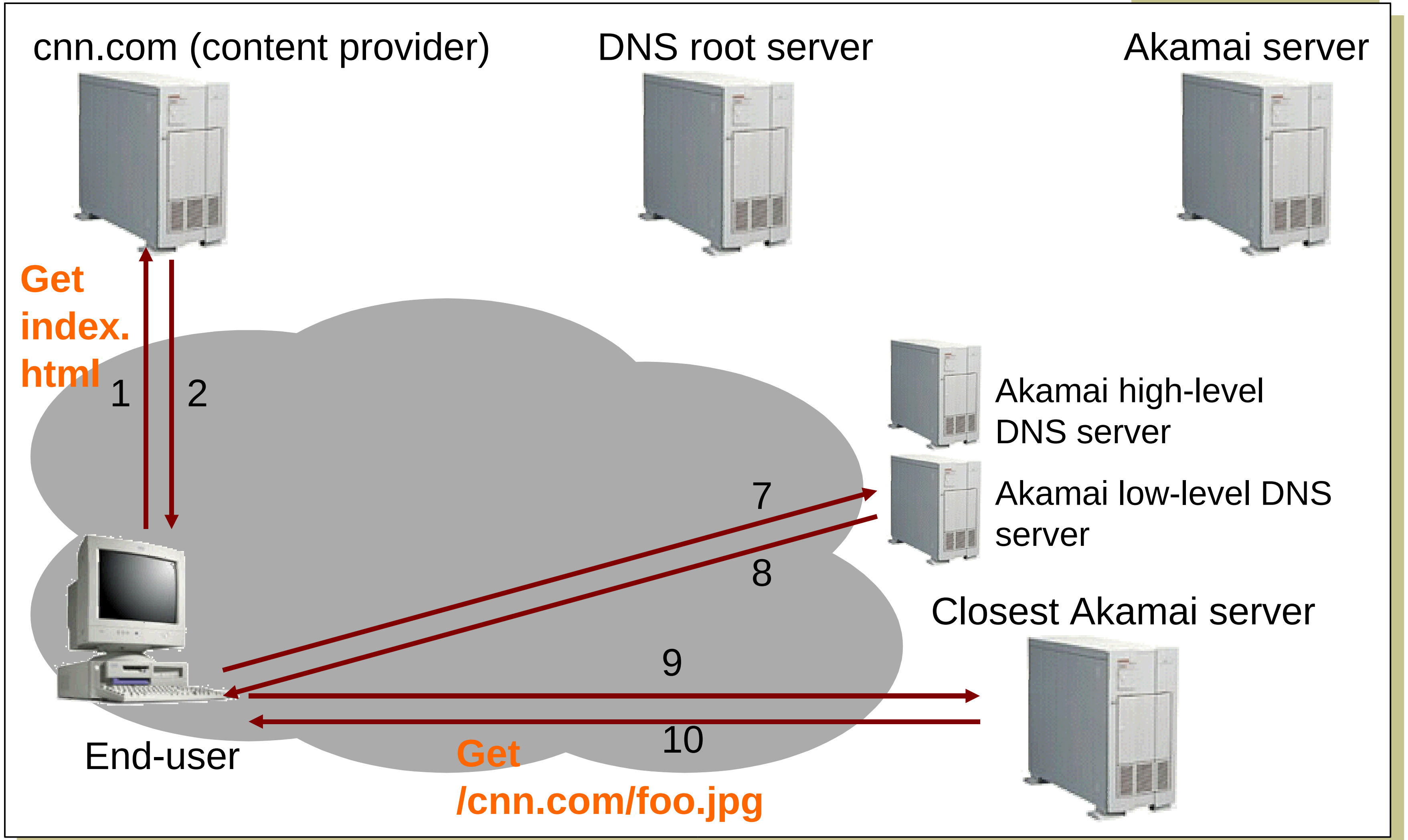
Consistent Hash – Example

- Construction
  - Assign each of C hash buckets to Klog(C) random points on unit interval
  - Map object to random position on unit interval
  - Hash of object = closest bucket
- Monotone addition of bucket does not cause movement between existing buckets
- Spread & Load small set of buckets that lie near object
- Balance no bucket is responsible for large portion of unit interval

How Akamai Works



Akamai – Subsequent Requests


cnn.com (content provider)
DNS root server
Akamai server
Get index. html
1
2
Akamai high-level DNS server
7
Akamai low-level DNS server
8
Closest Akamai server
9
10
End-user
Get /cnn.com/foo.jpg