Web, HTTP, Caching, CDNs

Outline

- Web
- HyperText Transfer Protocol (HTTP)
- Inefficiencies in HTTP
- HTTP Persistent Connections
- Caching
- CDNs
- Consistent Hashing

Web

- Original goal of the web: mechanism to organize and retrieve information
- Inspired by hypertext -- one document links to another
  - HyperText Markup Language (HTML) -- used to define basic content and layout of a web page
  - Supplemented by Cascading Style Sheets (CSS), JavaScript, images, documents, Flash/Silverlight, and other files
- Uniform resource locator (URL) specifies location of an object
  - Perform DNS lookup to obtain IP address of web server to contact
- Client and web server communicate using HTTP

HyperText Transfer Protocol (HTTP)

- Runs atop TCP
- Plain text messages in a request/response sequence
  - lines terminated by \r\n

HTTP Request

- Start line
  - Method to execute
    - GET -- retrieve document
    - HEAD -- retrieve metadata about document
    - POST -- send data to server
  - URL -- may exclude DN and put this in an option
  - HTTP/1.0 or HTTP/1.1
- Options/parameters
  - User-Agent -- browser name/version, OS name/version
  - Host -- DN portion of URL
  - Cookie
- Blank line
- Data -- only for methods like POST

HTTP Reply

- Start line
  - HTTP/1.0 or HTTP/1.1
  - Status
    - 200 OK
    - 404 Not found
    - 403 Forbidden
    - 301 Moved permanently
- Options/parameters
  - Content-Length
  - Content-Type
  - Server – server name/version
  - Cache-Control – how long object can be cached
  - Last-Modified
- Blank line
- Data

Example : Fetching a Web Page

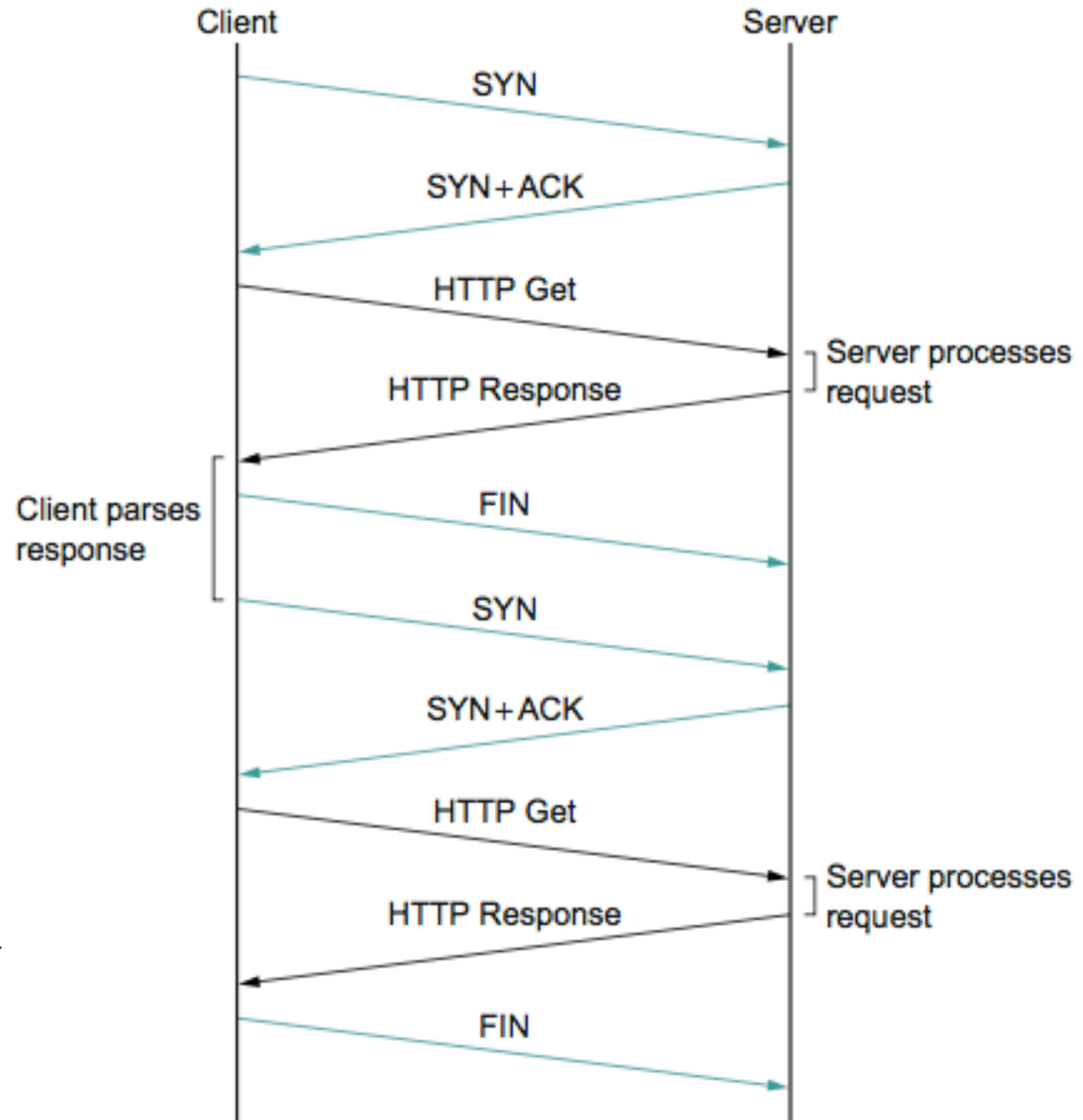
- DNS lookup
- Establish TCP connection
- Send HTTP request
- Receive HTTP reply
- Close TCP connection
- Parse HTML
- Establish TCP connection
- Send HTTP request for image
- Receive HTTP reply for image
- Close TCP connection

Example : Fetching a Web Page - Contd

- ...request other objects in page
- ...perform more DNS lookups if objects (e.g. ads) are in different domains (e.g., CDN)
- ...render page while other objects are being fetched

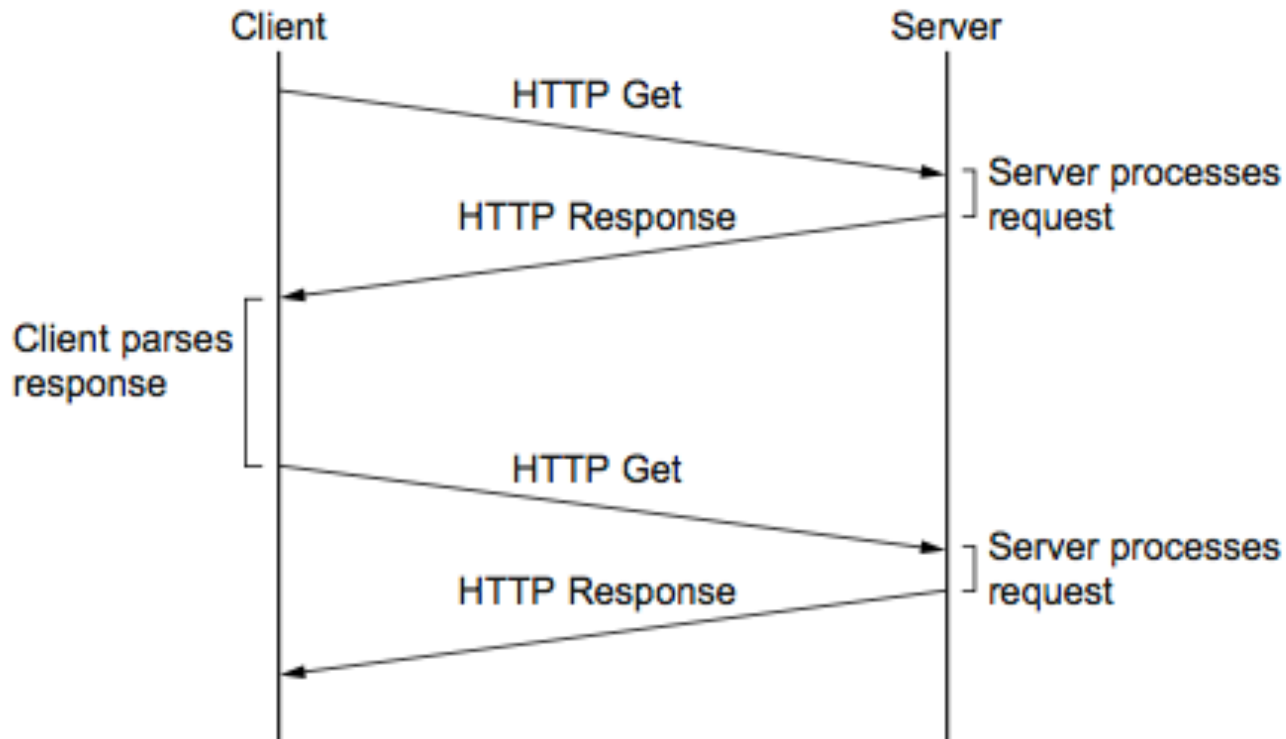
Inefficiencies in HTTP

- Problem: Using a separate TCP connection for each object in a web page has a lot of overhead for connection setup and teardown
  - For each object: 2 RTTs for connection setup plus at least 1 RTT for fetching data



Inefficiencies in HTTP - Contd

- Solution: HTTP 1.1 introduced persistent connections



HTTP Persistent Connections

- Exchange multiple request/response messages over the same TCP connection
- Only need to establish one connection to each server providing content for a page
  - If content is coming from multiple servers (e.g., main page and ads come from 2 different domains), you still need > 1 connection

HTTP Persistent Connections - Contd

- Also benefits throughput
  - For each connection, congestion window starts at 1 and is increasingly exponentially using slow start
  - Takes lots of RTTs to reach maximum throughput -- 8 RTTs for 1 Mbps link; 15 RTTs for 100Mbps link
  - Using one connection means initial slow start only occurs once
    - still invoke slow start later if timeout occurs due to loss, but ideally losses are handled through fast retransmit/fast recovery where slow start is not invoked
- Challenge: how long should a connection stay open?
  - Overhead at server to maintain connections for 1000s of clients
  - Throughput benefits far outweigh this overhead

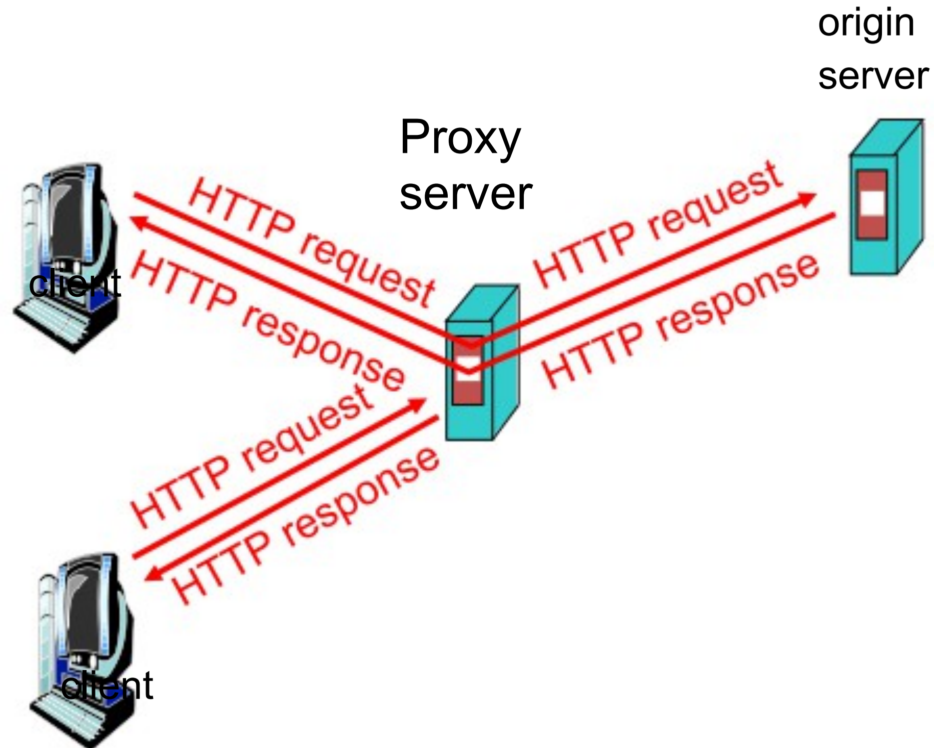
Web Caching and CDNs

Where can bottlenecks occur?

- First mile: client to its ISPs
- Last mile: server to its ISP
- Server: compute/memory limitations
- ISP interconnections/peerings: congestion inside the network

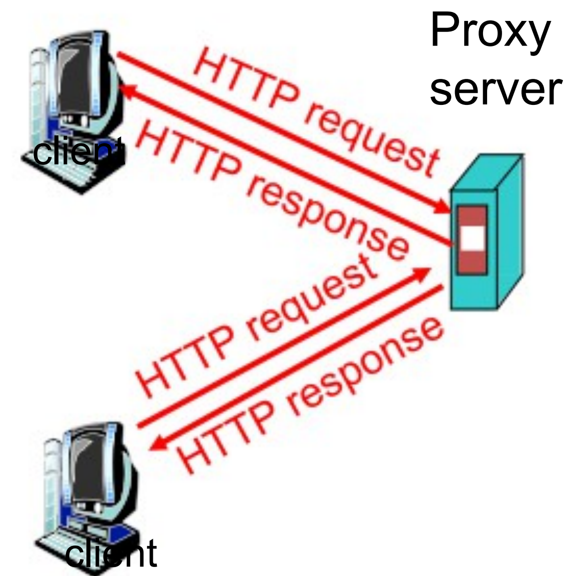
- Caching at various locations to overcome these bottlenecks

Proxy Caches



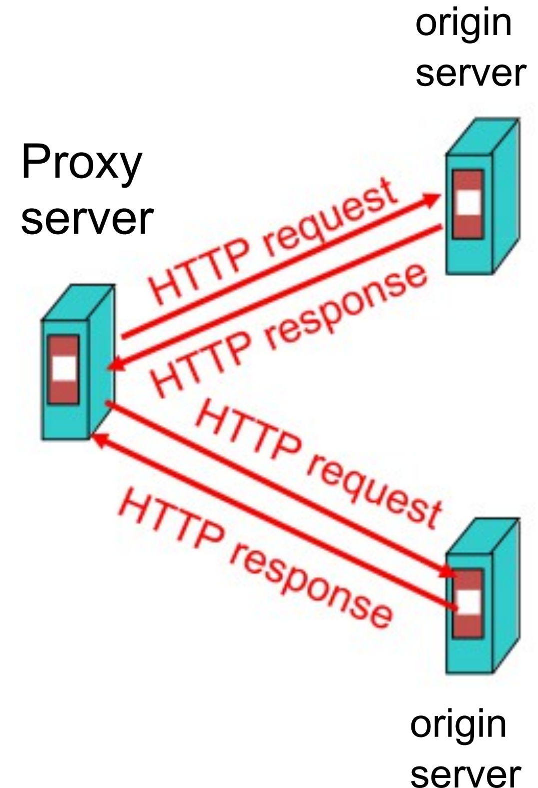
Forward Proxy

- Cache “close” to the client
    - Under administrative control of client-side AS
- Explicit proxy
    - Requires configuring browser
- Implicit proxy
    - Service provider deploys an “on path” proxy
    - … that intercepts and handles Web requests



Reverse Proxy

- Cache "close" to server
  - Either by proxy run by server or in third-party content distribution network (CDN)
- Directing clients to the proxy
  - Map the site name to the IP address of the proxy



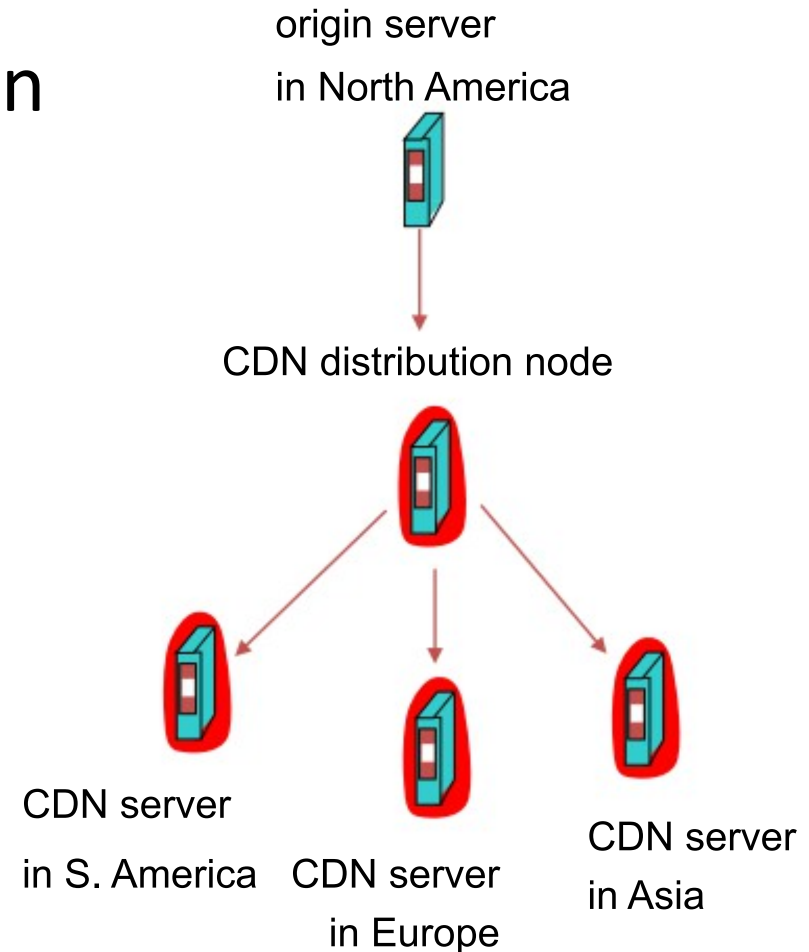
Limitations of Web Caching

- Much content is not cacheable
  - Dynamic data: stock prices, scores, web cams
  - CGI scripts: results depend on parameters
  - Cookies: results may depend on passed data
  - SSL: encrypted data is not cacheable
  - Analytics: owner wants to measure hits
- Stale data
  - Or, overhead of refreshing the cached data

Content Distribution Networks

Content Distribution Network

- Proactive content replication
  - Content provider (e.g., CNN) contracts with a CDN
- CDN replicates the content
  - On many servers spread throughout the Internet
- Updating the replicas
  - Updates pushed to replicas when the content changes



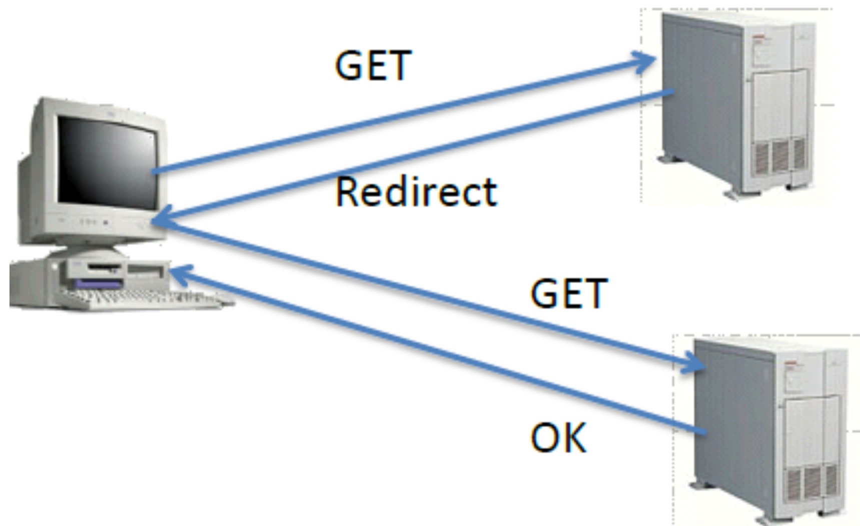
Server Selection Policy

Requires continuous monitoring of liveness, load, and performance

- Live server
  - For availability
- Lowest load
  - To balance load across the servers
- Closest
  - Nearest geographically, or in round-trip time
- Best performance
  - Throughput, latency, …
- Cheapest bandwidth, electricity, …

Server Selection Mechanism

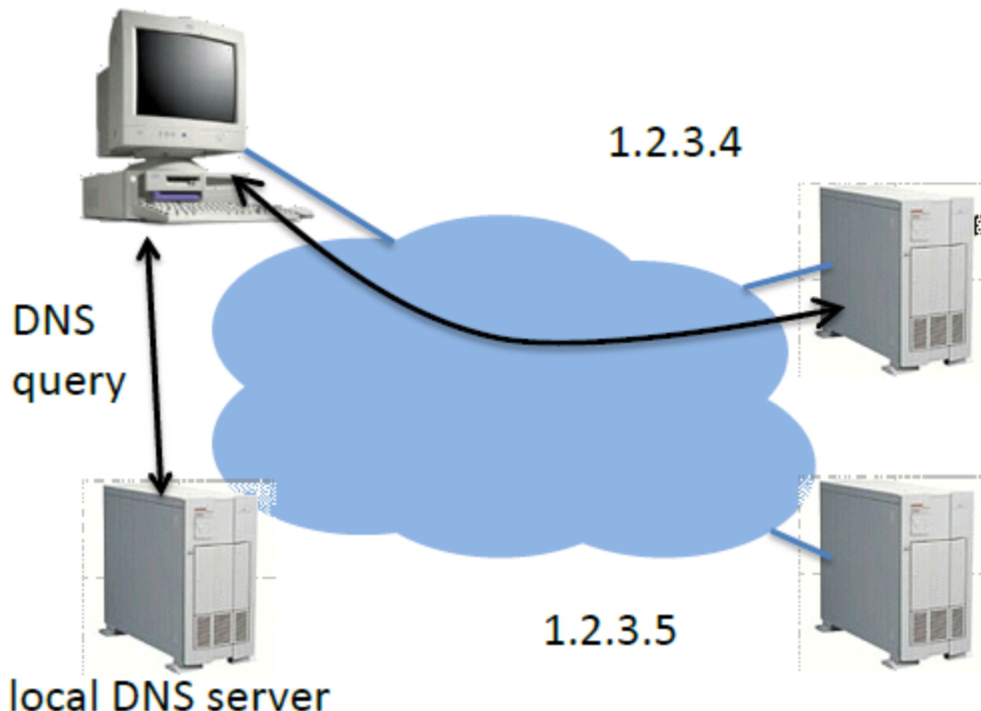
- Application
  - HTTP redirection



- Advantages
  - Fine-grain control
  - Selection based on client IP address
- Disadvantages
  - Extra round-trips for TCP connection to server
  - Overhead on the server

Server Selection Mechanism

- Naming
  - DNS-based server selection



- Advantages
  - Avoid TCP set-up delay
  - DNS caching reduces overhead
  - Relatively fine control

Disadvantage

- Based on IP address of local DNS server
- "Hidden load" effect
- DNS TTL limits adaptation

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 on CDN servers in an *on demand* fashion.
- Example: Akamai networks

How Akamai Works

- Clients fetch html document from primary server
  - E.g. fetch index.html from cnn.com
- “Akamaized” 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

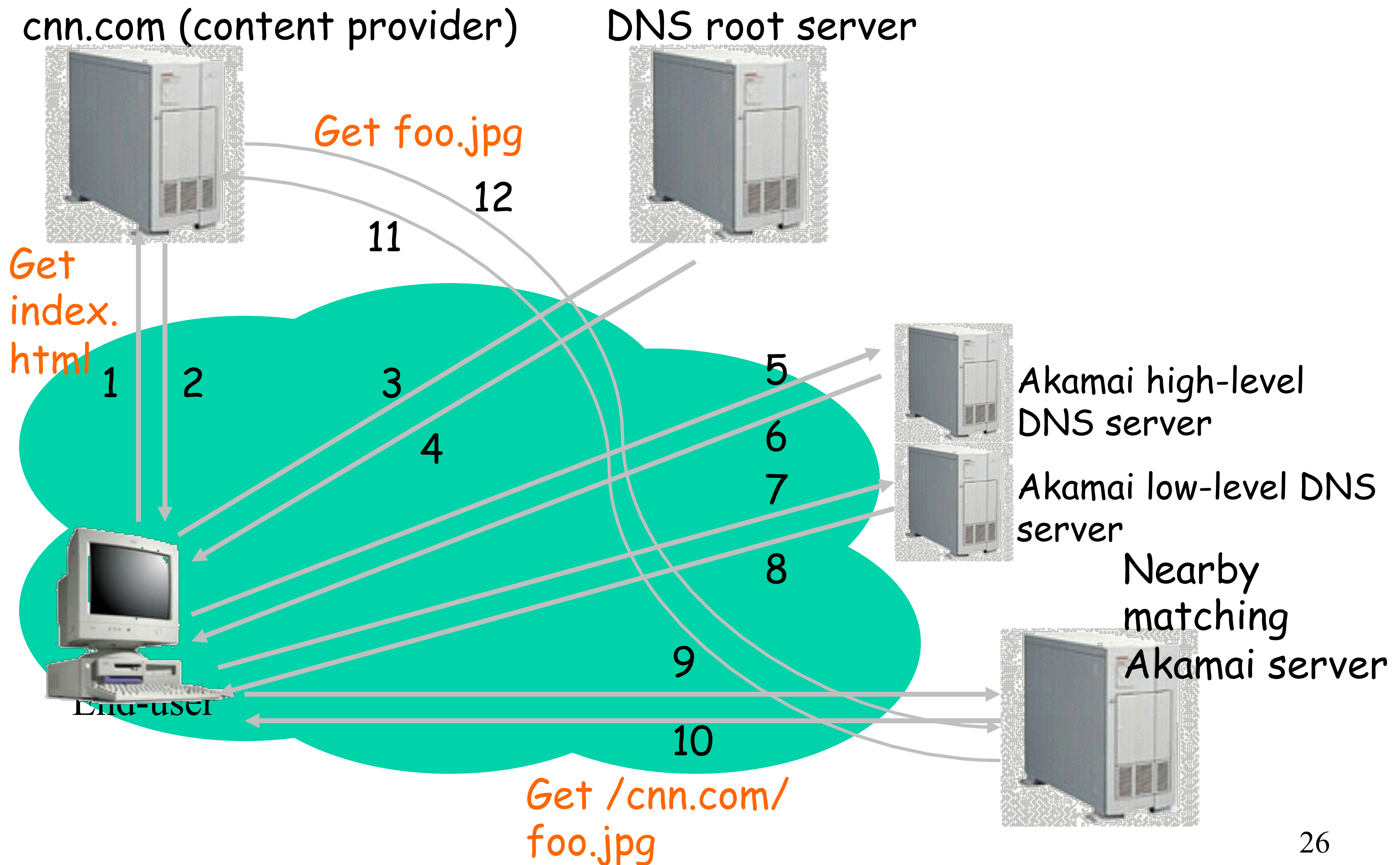
How Akamai Works

- Only static content is “Akamaized”

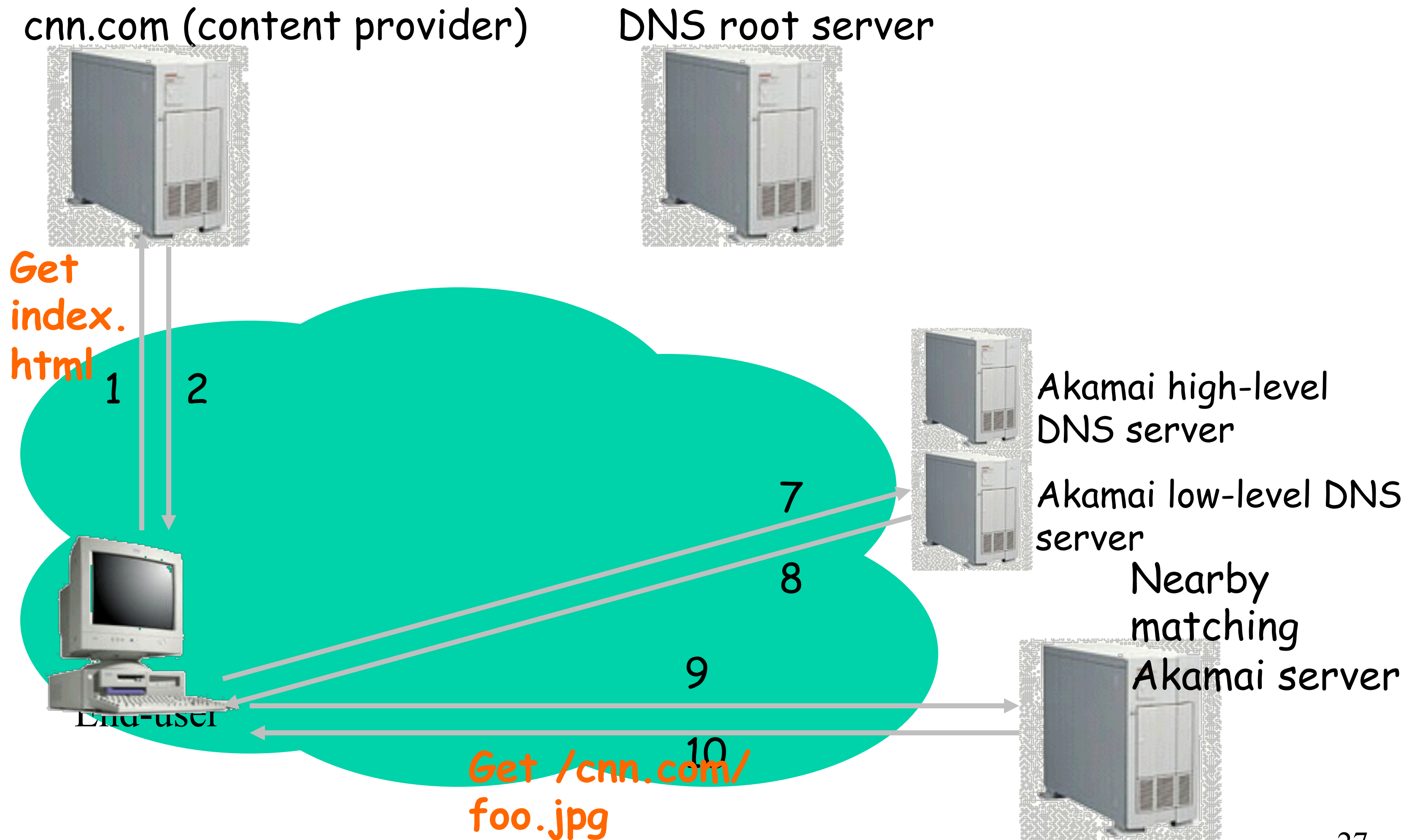
- Modified name contains original file name and content provider ID

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

Overview of How Akamai Works



Akamai – Subsequent Requests



Recap: 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
    - Out-of-band measurements to obtain this
- G.akamaitech.net nameserver chooses server in region
  - A collection of serves in each region
  - Which server to choose?
  - Uses aXYZ name

Simple Hashing

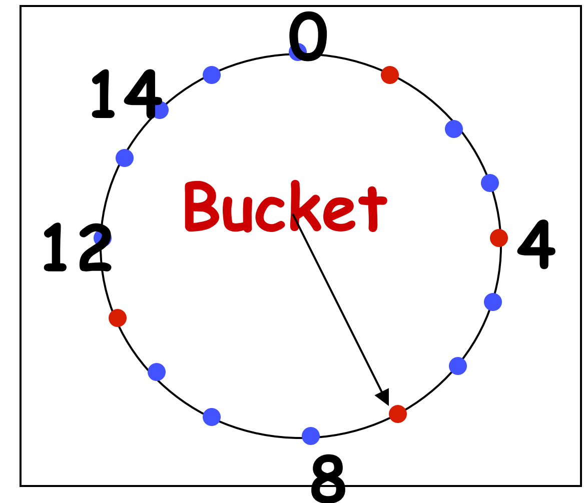
- Given document XYZ, we need to choose a server to use
- Suppose we use the "mod" function
- Number servers from 1…n
  - Place document XYZ on server (XYZ mod n)
  - What happens when a servers fails? n → n-1
  - Why might this be bad?

Consistent Hash

- Desired features
  - Balanced – 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 a set of object
  - Load – # of objects assigned to hash bucket is small

Consistent Hash – Example

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



- Smoothness → 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 number of objects

Load Aware

```
SelectServer(name, S)
  for each server s_i in S
    weight_i = hash(name, s_i)
  sort weight
  for each server s_j in decreasing order of weight_j
    if (load(s_j) < threshold) then
      return s_j
  return server with highest weight
```