CS 3700
Networks and Distributed Systems

Lecture 17: Client-Server Systems

(Partially off slides by David Anderson at CMU, Ken Thompson at Cornell)

Client-Server Computing

- 99% of all distributed systems use client-server architectures!
- Today: look at the client-server architectures
- Detailed example: The Web

Outline

- Client-Server systems
- HTTP
- Scaling up: CDNs

Client-Server concept

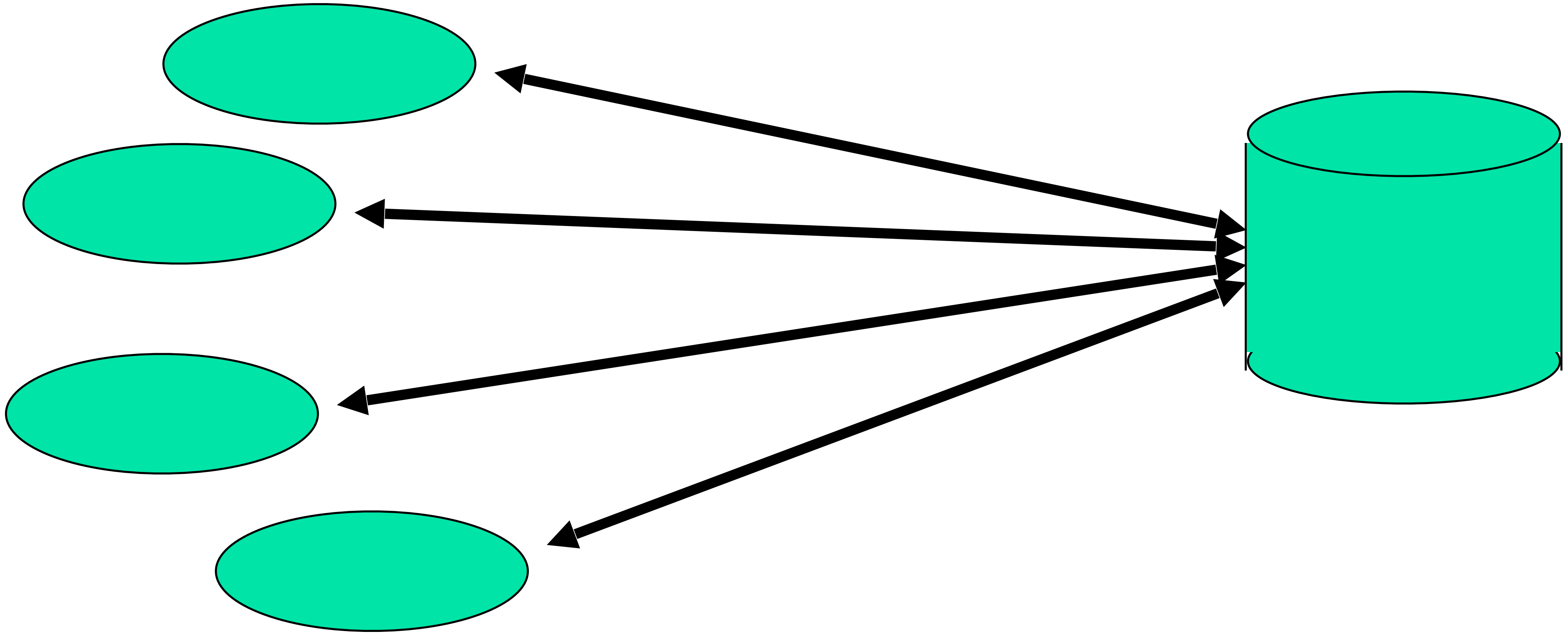
- Server program is shared by many clients

- RPC protocol typically used to issue requests
  - RPC = Remote Procedure Call

- Server may manage special data, run on an especially fast platform, or have an especially large disk

- Client systems handle "front-end" processing and interaction with the human user

Server and its clients



Examples of servers

- Network file server
- Database server
- Network information server
- Web server
- Domain name service
- Microsoft Exchange (email) server
- Kerberos authentication server

Business examples

- Risk manager for a bank: tracks exposures in various currencies or risk in investments
- Theoretical price for securities or bonds: traders use this to decide what to buy and what to sell
- Server for an ATM: decides if your withdrawal will be authorized

Bond pricing example

- Server receives market trading information, currency data, interest rates data
- Has a database of all the bonds on the market
- Client expresses interest in a given bond, or in finding a bond with certain properties
- Server calculates what that bond (or what each bond) should cost given current conditions

Why use a client-server approach?

- Pricing parameters are “expensive” (in terms of computing resources) to obtain: must monitor many data sources and precompute many time-value of money projections for each bond
- Computing demands may be extreme: demands a very high performance machine
- Database of bonds is huge: large storage, more precomputation

On client side

- Need a lot of CPU and graphics power to display the data and interact with the user
- Dedicated computation provides snappy response time and powerful decision making aids
- Can “cache” or “save” results of old computations so that if user revisits them, won’t need to reissue identical request to server

Summary of typical split

- Server deals with bulk data storage, high perf. computation, collecting huge amounts of background data that may be useful to any of several clients
- Client deals with the "attractive" display, quick interaction times
- Use of caching to speed response time

Typical issues in design

- Client is generally simpler than server: may be single-threaded, can wait for reply to RPC's
- Server is generally multithreaded, designed to achieve extremely high concurrency and throughput.
  - Much harder to develop
- Reliability issue: if server goes down, all its clients may be "stuck". Usually addressed with some form of backup or replication.

Outline

- Client-Server systems
- HTTP
- Scaling up: CDNs

HTTP Basics

- HTTP layered over bidirectional byte stream
- 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

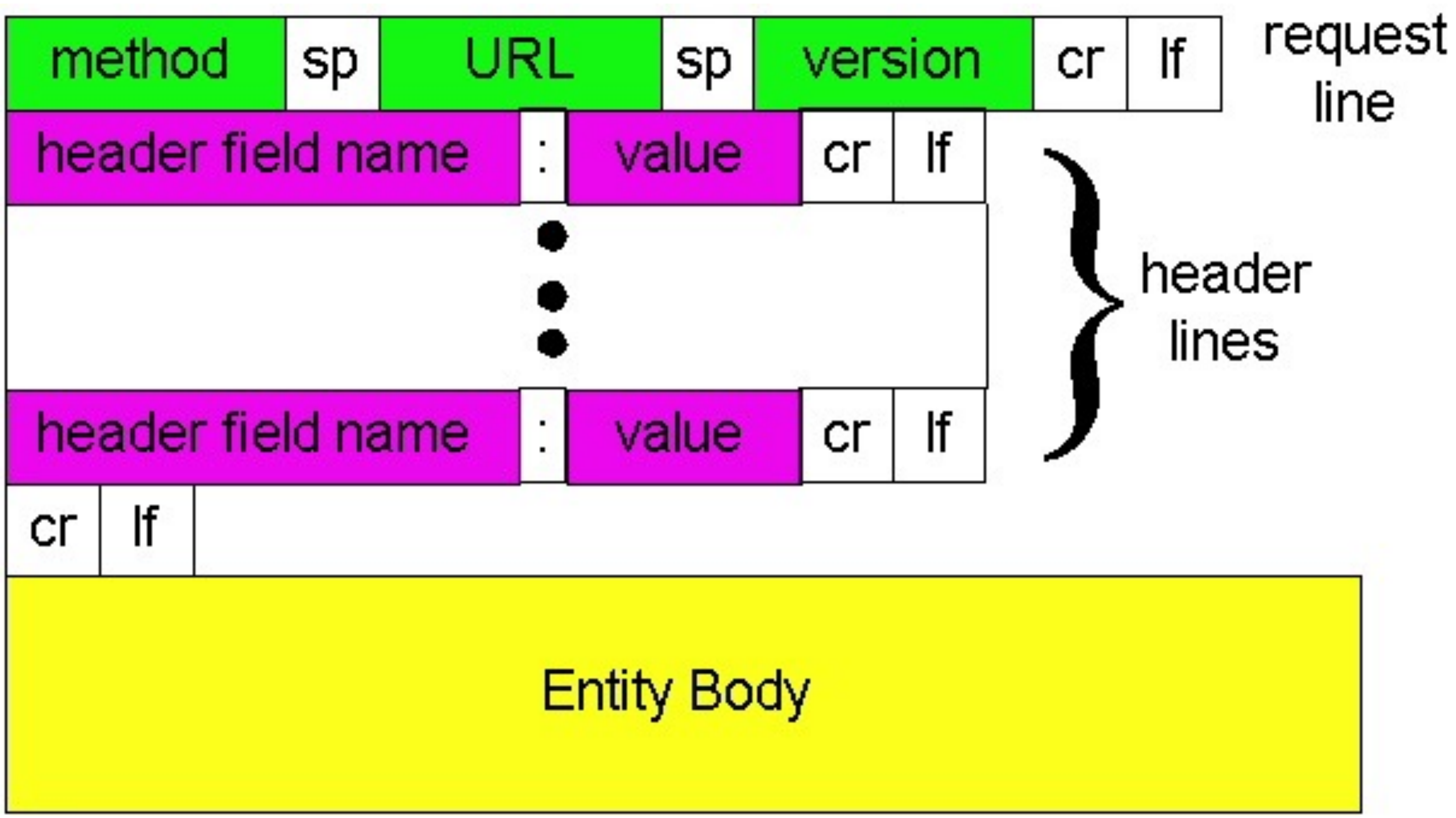

```
GET /foo/bar.html HTTP/1.1
```

- 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

- Request headers (each ended with CRLF)
  - Acceptable document types/encodings
  - From – user email
  - If-Modified-Since
  - Referrer – what caused this page to be requested
  - User-Agent – client software
  - Cookie - previously stored information
  - Content-Length - Size of data (only on POST)
- Blank-line (CRLF)
- Body

HTTP Request (visual)



HTTP Request Example


```
GET /blah.html?foo=bar 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)
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
Cache-Control: private

...DATA...
```

Web pages

- Multiple (typically small) objects per page
  - E.g., each image, JS, CSS, etc downloaded separately

- Single page can have 100s of HTTP transactions!

- File sizes
  - Heavy-tailed
  - Most transfers/objects very small

- Problem: Browser can't render complete page until all downloaded

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, One Image

Client opens TCP connection

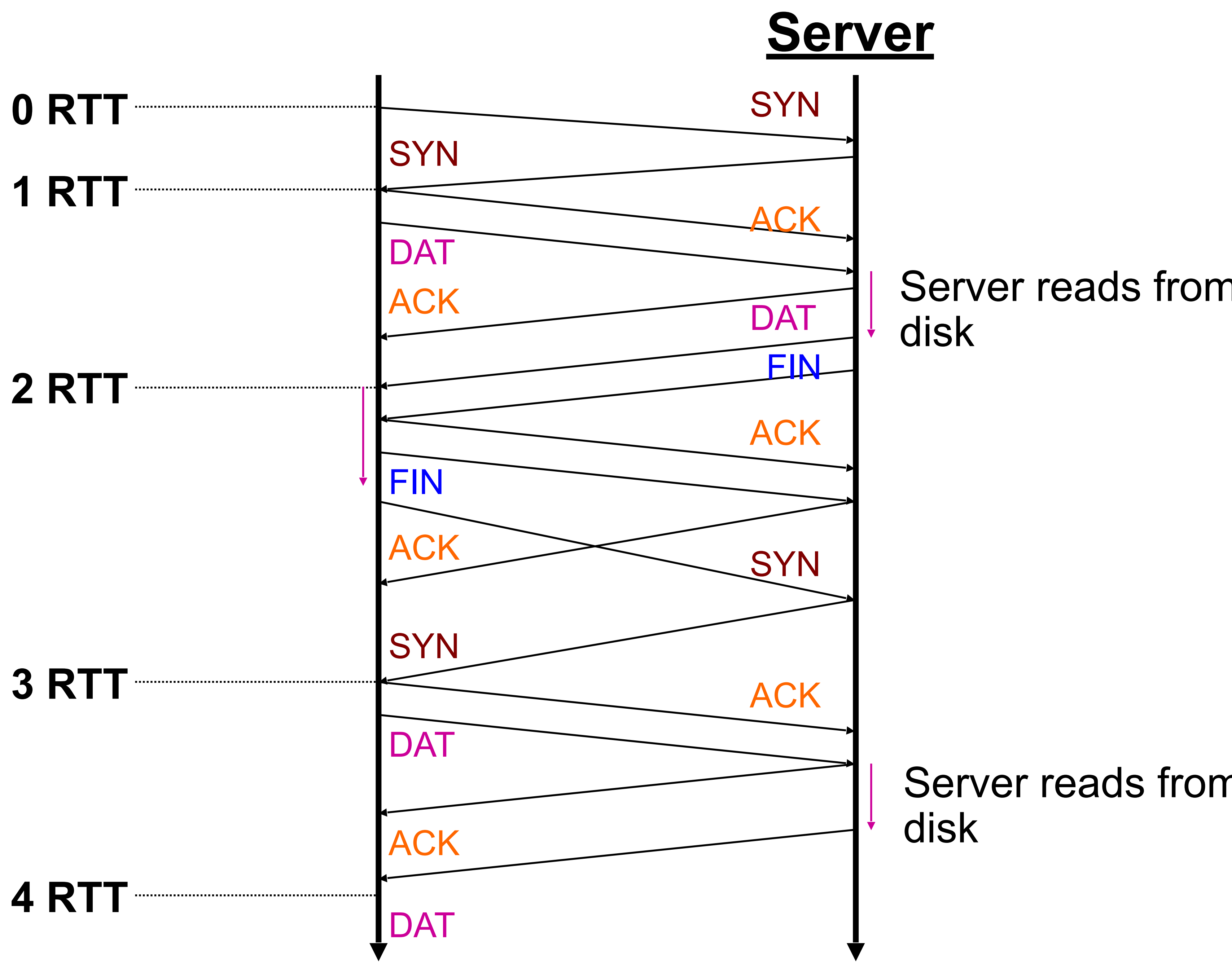
Client sends HTTP request for HTML

Client parses HTML

Client opens TCP connection

Client sends HTTP request for image

Image begins to arrive



More Problems

- Short transfers are hard on TCP
  - Stuck in slow start
  - Loss recovery is poor when windows are small
  - SYN/ACK overhead is highest

- Lots of extra connections
  - Increases server state/processing

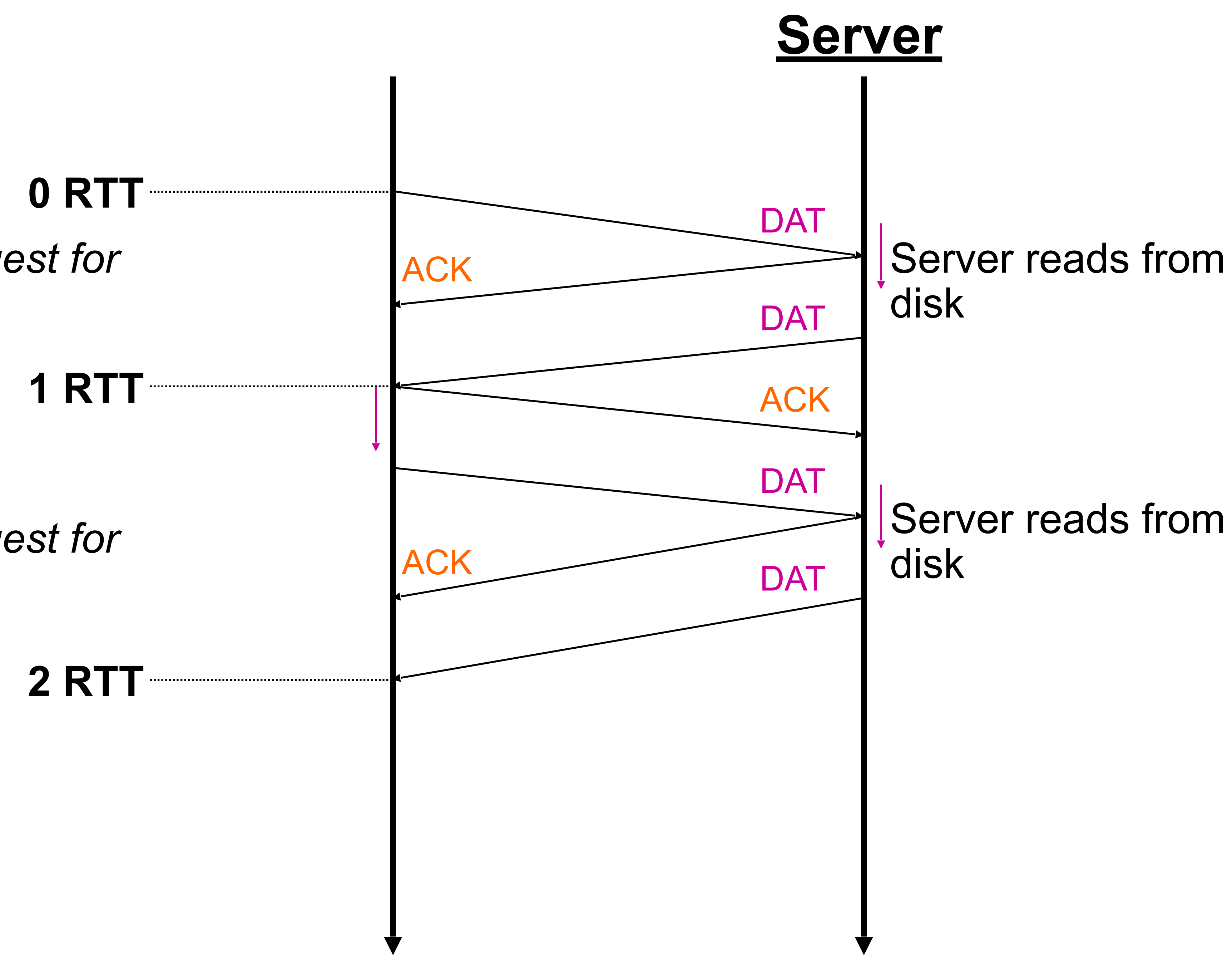
- Server also forced to keep TIME_WAIT connection state
  - Why must server keep these?
  - Tends to be an order of magnitude greater than # of active connections, why?

Persistent Connections

- Multiplex multiple transfers onto one TCP connection

- Client keeps connection open
  - Can send another request after the first completes
  - Must announce intention via a header
    - Connection: keepalive
  - Server must say how long response is, so client knows when done
    - Content-Length: XXX

Persistent Connection Example



Client sends HTTP request for HTML

Client parses HTML

Client sends HTTP request for image

Image begins to arrive

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 “etag” (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)
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)
Connection: Keep-Alive
Keep-Alive: timeout=15, max=100
ETag: "7a11f-10ed-3a75ae4a”
```

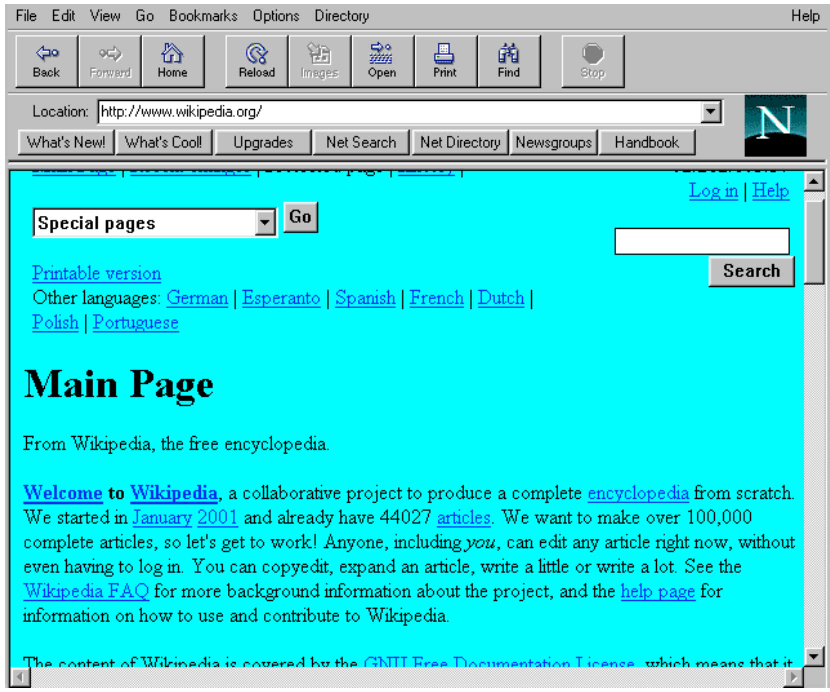
Content in today's Internet

- Most flows are HTTP
  - Web is at least 52% of traffic
  - Median object size is 2.7K, average is 85K (as of 2007)
- HTTP uses TCP, so it will
  - Be ACK clocked
  - For Web, likely never leave slow start
- Is the Internet designed for this common case?
  - Why?

Outline

- Client-Server systems
- HTTP
- Scaling up: CDNs

Evolution of Serving Web Content



- In the beginning…
  - …there was a single server
  - Probably located in a closet
  - And it probably served blinking text
- Issues with this model
  - Site reliability
    - Unplugging cable, hardware failure, natural disaster
  - Scalability
    - Flash crowds (aka Slashdotting)

Replicated Web service

- Use multiple servers

- Advantages
  - Better scalability
  - Better reliability

- Disadvantages
  - How do you decide which server to use?
  - How to do synchronize state among servers?

Load Balancers

- Device that multiplexes requests across a collection of servers
  - All servers share one public IP
  - Balancer transparently directs requests to different servers
- How should the balancer assign clients to se
  - Random / round-robin
    - When is this a good idea?
  - Load-based
    - When might this fail?
- Challenges
  - Scalability (must support traffic for n hosts)
  - State (must keep track of previous decisions)
    - RESTful APIs reduce this limitation

Load balancing: Are we done?

- Advantages
  - Allows scaling of hardware independent of IPs
  - Relatively easy to maintain
- Disadvantages
  - Expensive
  - Still a single point of failure
  - Location!

Where do we place the load balancer for Wikipedia?

Popping up: HTTP performance

- For Web pages
  - RTT matters most
  - Where should the server go?

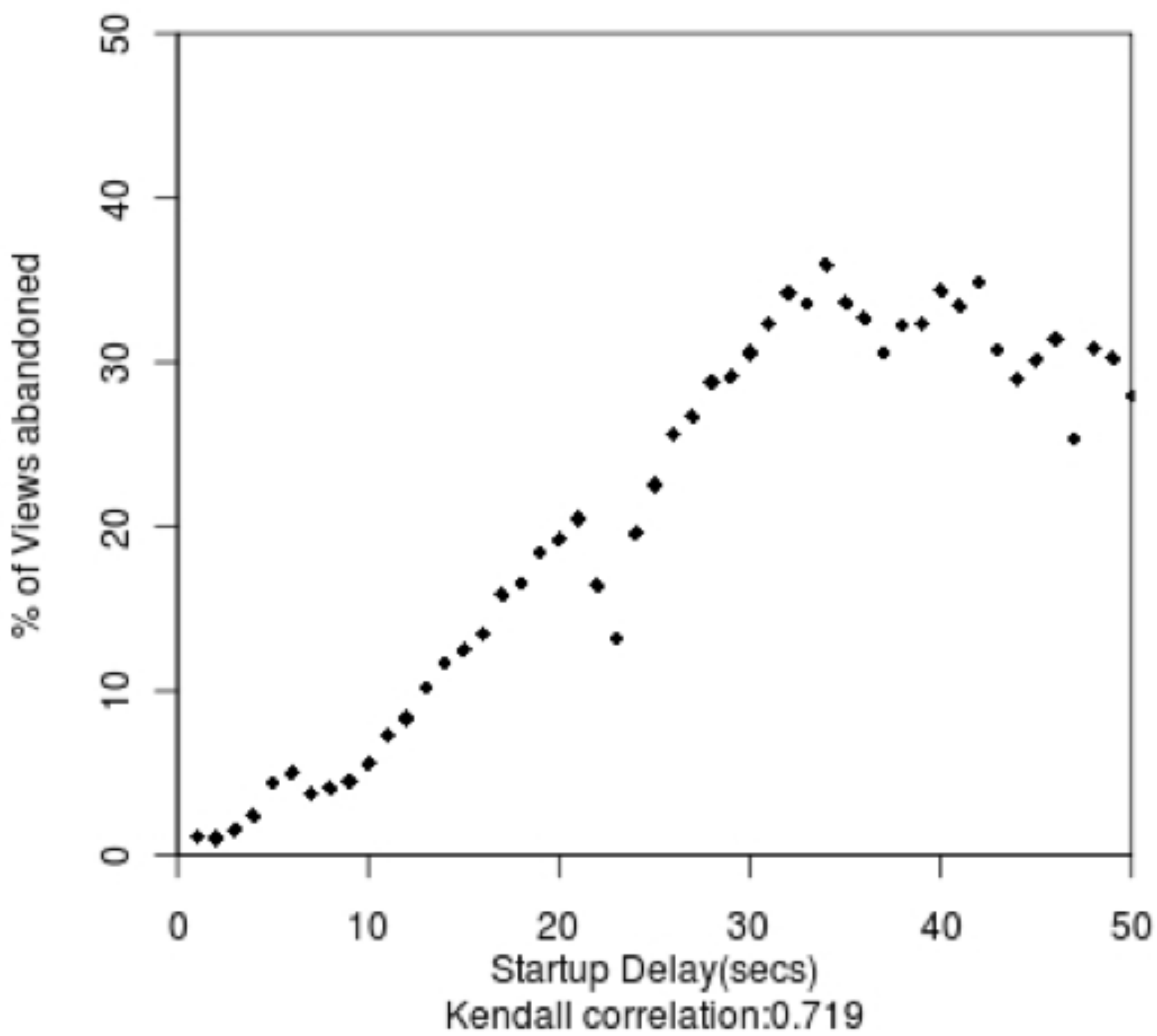
- For video
  - Available bandwidth matters most
  - Where should the server go?

- Is there one location that is best for everyone?

Server placement

Why speed matters

- Impact on user experience
  - Users navigating away from pages
  - Video startup delay



Why speed matters

- Impact on user experience
  - Users navigating away from pages
  - Video startup delay
- Impact on revenue
  - Amazon: increased revenue 1% for every 100ms reduction in PLT
  - Shopzilla:12% increase in revenue by reducing PLT from 6 seconds to 1.2 seconds
- Ping from BOS to LAX: ~100ms

Strawman solution: Web caches

- ISP uses a middlebox that caches Web content
  - Better performance – content is closer to users
  - Lower cost – content traverses network boundary once
  - Does this solve the problem?

- No!
  - Size of all Web content is too large
    - Zipf distribution limits cache hit rate
  - Web content is **dynamic** and **customized**
    - Can't cache banking content
    - What does it mean to cache search results?

What is a CDN?

- Content Delivery Network
  - Also sometimes called Content Distribution Network
  - At least half of the world's bits are delivered by a CDN
    - Probably closer to 80/90%

- Primary Goals
  - Create replicas of content throughout the Internet
  - Ensure that replicas are always available
  - Directly clients to replicas that will give good performance

Key Components of a CDN

- Distributed servers
  - Usually located inside of other ISPs
  - Often located in IXPs (coming up next)
- High-speed network connecting them
- Clients (eyeballs)
  - Can be located anywhere in the world
  - They want fast Web performance
- Glue
  - Something that binds clients to “nearby” replica servers

Examples of CDNs

- Akamai
  - 147K+ servers, 1200+ networks, 650+ cities, 92 countries
- Limelight
  - Well provisioned delivery centers, interconnected via a private fiber-optic connected to 700+ access networks
- Edgecast
  - 30+ PoPs, 5 continents, 2000+ direct connections
- Others
  - Google, Facebook, AWS, AT&T, Level3, Brokers

Inside a CDN

- Servers are deployed in clusters for reliability
  - Some may be offline
    - Could be due to failure
    - Also could be “suspended” (e.g., to save power or for upgrade)
- Could be multiple clusters per location (e.g., in multiple racks)
- Server locations
  - Well-connected points of presence (PoPs)
  - Inside of ISPs

Mapping clients to servers

- CDNs need a way to send clients to the "best" server
  - The best server can change over time
  - And this depends on client location, network conditions, server load, …
  - What existing technology can we use for this?

- DNS-based redirection
  - Clients request www.foo.com
  - DNS server directs client to one or more IPs based on request IP
  - Use short TTL to limit the effect of caching

CDN redirection example


```
choffnes$ dig www.fox.com


;; ANSWER SECTION:
www.fox.com.             510     IN      CNAME   www.fox-rma.com.edgesuite.net.
www.fox-rma.com.edgesuite.net. 5139 IN           CNAME   a2047.w7.akamai.net.
a2047.w7.akamai.net.   4        IN      A       23.62.96.128
a2047.w7.akamai.net.   4        IN      A       23.62.96.144
a2047.w7.akamai.net.   4        IN      A       23.62.96.193
a2047.w7.akamai.net.   4        IN      A       23.62.96.162
a2047.w7.akamai.net.   4        IN      A       23.62.96.185
a2047.w7.akamai.net.   4        IN      A       23.62.96.154
a2047.w7.akamai.net.   4        IN      A       23.62.96.169
a2047.w7.akamai.net.   4        IN      A       23.62.96.152
a2047.w7.akamai.net.   4        IN      A       23.62.96.186
```

DNS Redirection Considerations

- Advantages
  - Uses existing, scalable DNS infrastructure
  - URLs can stay essentially the same
  - TTLs can control “freshness”

- Limitations
  - DNS servers see only the DNS server IP
    - Assumes that client and DNS server are close. Is this accurate?
  - Small TTLs are often ignored
  - Content owner must give up control
  - Unicast addresses can limit reliability