16:332:543: Communication Networks I

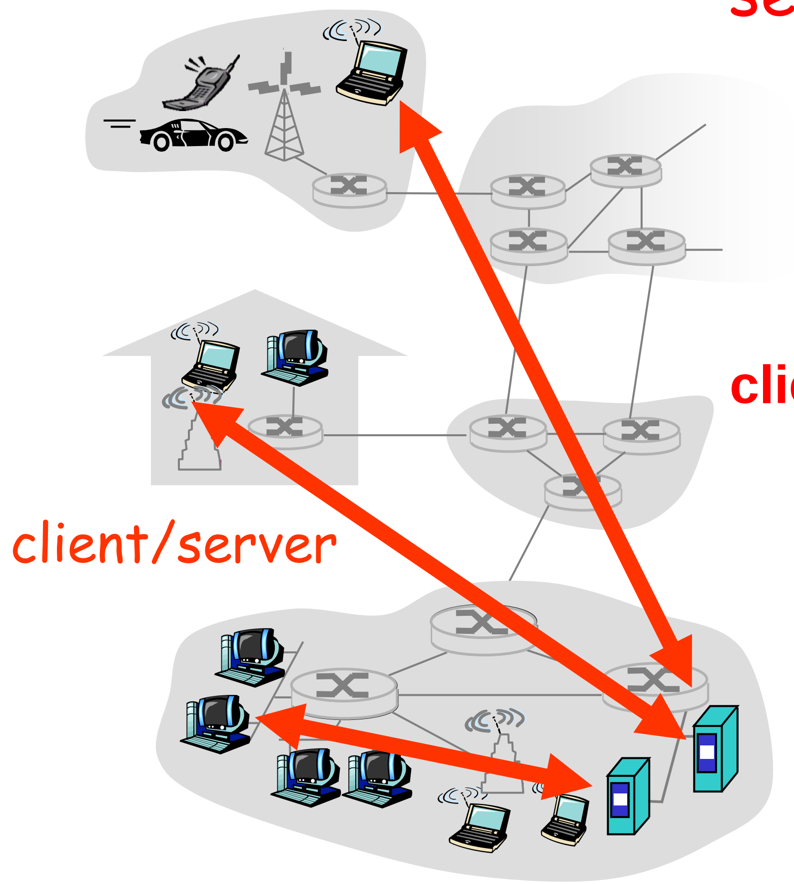
Application Layer

Based in part on slides from J.F. Kurose and K.W. Ross covering material from their book “Computer Networking – A Top-Down approach”

Application architectures

- Client-server
- Peer-to-peer (P2P)
- Hybrid of client-server and P2P

Client-server architecture



server:
- always-on host
- permanent IP address
- Scaling: server farms, data center

clients:
- communicate with server
- may be intermittently connected
- do not communicate directly with each other

Application layer protocol

- Types of messages exchanged,
  - e.g., request, response
- Message syntax:
  - what fields in messages & how fields are delineated
- Message semantics
  - meaning of information in fields
- Rules for when and how processes send & respond to messages

Public-domain protocols:

- defined in RFCs
- allows for interoperability
- e.g., HTTP, SIP

Proprietary protocols:

- e.g., Skype

Transport service requirements for applications

Data loss

- audio can tolerate some loss
- file download requires 100% reliable data transfer

Timing

- Internet telephony, interactive games require low delay

Throughput

- multimedia requires minimum throughput
- “elastic apps” more flexible

Security

- Encryption, data integrity, …

Transport service requirements of common apps

Application	Data loss	Throughput	Time Sensitive
file transfer	no loss	elastic	no
e-mail	no loss	elastic	no
Web documents	no loss	elastic	no
real-time audio/video	loss-tolerant	audio: 5kbps-1Mbps video:10kbps-5Mbps	yes, 100’s msec
stored audio/video	loss-tolerant	same as above	yes, few secs
interactive games	loss-tolerant	few kbps up	yes, 100’s msec
instant messaging	no loss	elastic	yes and no

Internet transport protocols services

TCP service:

- ***connection-oriented:*** setup required between client and server processes
- ***reliable transport*** between sending and receiving process
- ***flow control:*** sender won't overwhelm receiver
- ***congestion control:*** throttle sender when network overloaded
- ***does not provide:*** timing, minimum throughput guarantees, security

UDP service:

- **unreliable data transfer** between sending and receiving process
- **does not provide: connection setup, reliability, flow control, congestion control, timing, throughput guarantee, or security**

Q: **Why is there a UDP?**

Internet apps: application, transport protocols

Application	Application layer protocol	Underlying transport protocol
e-mail	SMTP [RFC 2821]	TCP
remote terminal access	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
file transfer	FTP [RFC 959]	TCP
streaming multimedia	HTTP (eg Youtube), RTP [RFC 1889]	TCP or UDP
Internet telephony	SIP, RTP, proprietary (e.g., Skype)	typically UDP

Web and HTTP

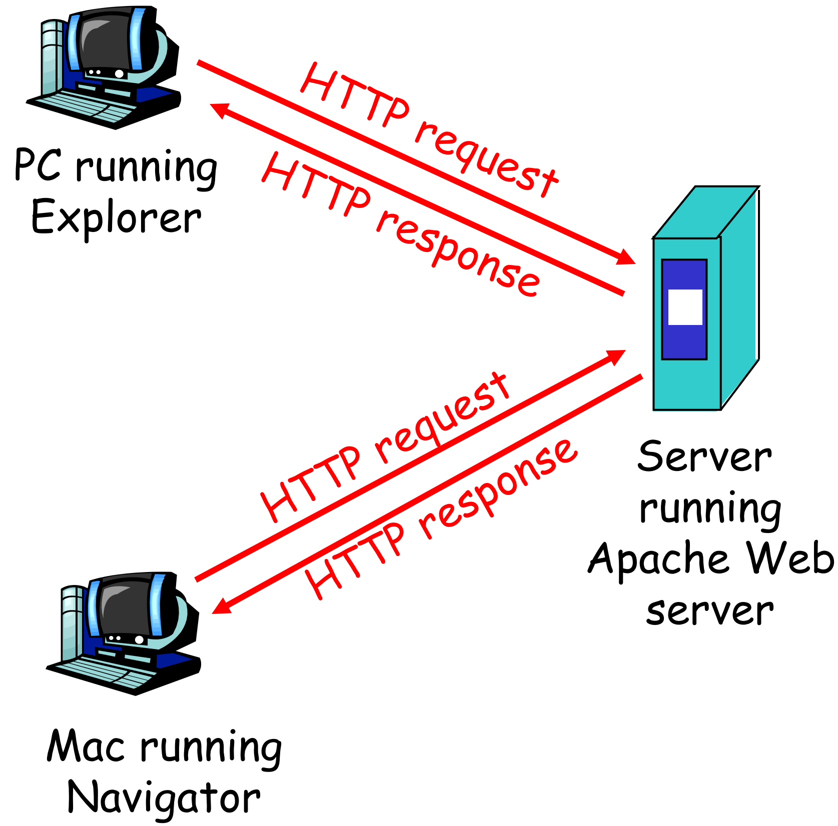
- Web page consists of objects
- Object can be HTML file, JPEG image, Java applet, audio file,…
- Web page consists of base HTML-file which includes several referenced objects
- Each object is addressable by a URL
- Example URL:

$$\underbrace{\texttt{www.someschool.edu}}_{\text{host name}}\texttt{/}\underbrace{\texttt{someDept/pic.gif}}_{\text{path name}}$$

HTTP overview

HTTP: hypertext transfer protocol

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



HTTP overview (continued)

- Uses TCP
- HTTP is "stateless"
- Nonpersisten HTTP
- Persistent HTTP

Example: HyperText Transfer Protocol

```
GET /courses/archive/fall13/EE6776/ HTTP/1.1
Host: www.ee.columbia.edu
User-Agent: Mozilla/4.03
```

Request

```
HTTP/1.1 200 OK
Date: Wed, 9 Sep 2015 13:09:03 GMT
Server: Netscape-Enterprise/3.5.1
Last-Modified: Sun, 6 Sep  2015 11:12:23 GMT
Content-Length: 6821
(data data data ...)
```

Response

Stateless Protocol

- Stateless protocol
  - Each request-response exchange treated independently
  - Clients and servers not required to retain state
- Statelessness to improve scalability
  - Avoid need for the server to retain info across requests
  - Enable the server to handle a higher rate of requests
- However, some applications need state
  - To uniquely identify the user or store temporary info
  - E.g., personalize a Web page, compute profiles or access statistics by user, keep a shopping cart, etc.
  - Lead to the introduction of "cookies" in the mid 1990s

HTTP Resource Meta-Data

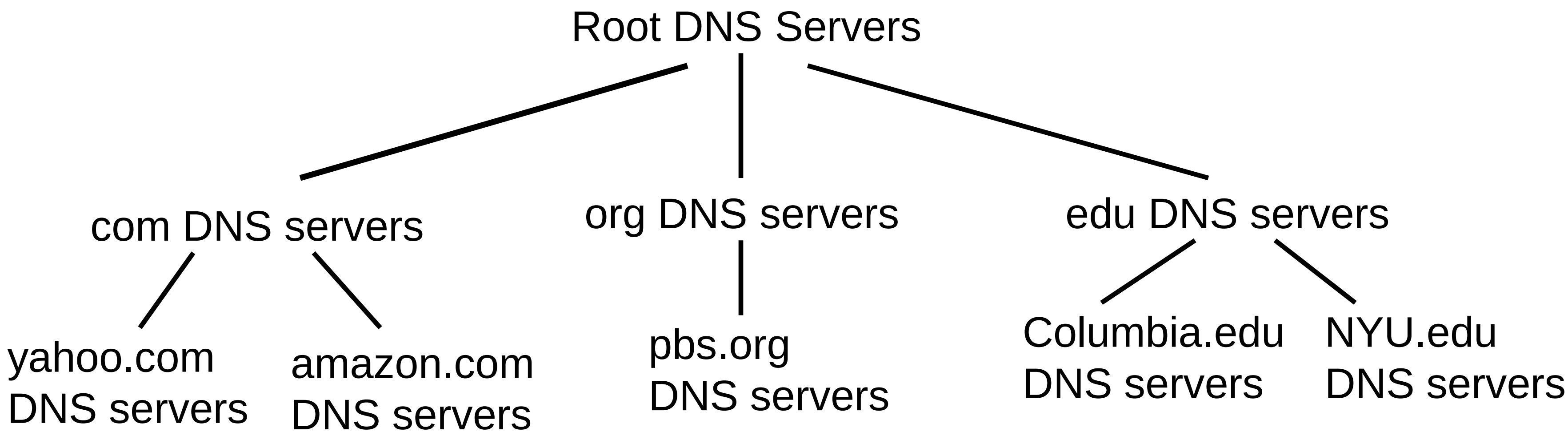
- Meta-data
  - Information relating to a resource
  - … but not part of the resource itself
- Example meta-data
  - Size of a resource
  - Type of the content
  - Last modification time

DNS: Domain Name System

Domain Name System:

- *distributed database* implemented in hierarchy of many *name servers*
- *application-layer protocol*
  - host and servers communicate to *resolve* names (address/name translation)

Distributed, Hierarchical Database



Client wants IP for www.amazon.com; 1st approx:

- client queries a root server to find com DNS server
- client queries com DNS server to get amazon.com DNS server
- client queries amazon.com DNS server to get IP address for www.amazon.com

More about Web caching

- cache acts as both client and server
- typically cache is installed by ISP (university, company, residential ISP)

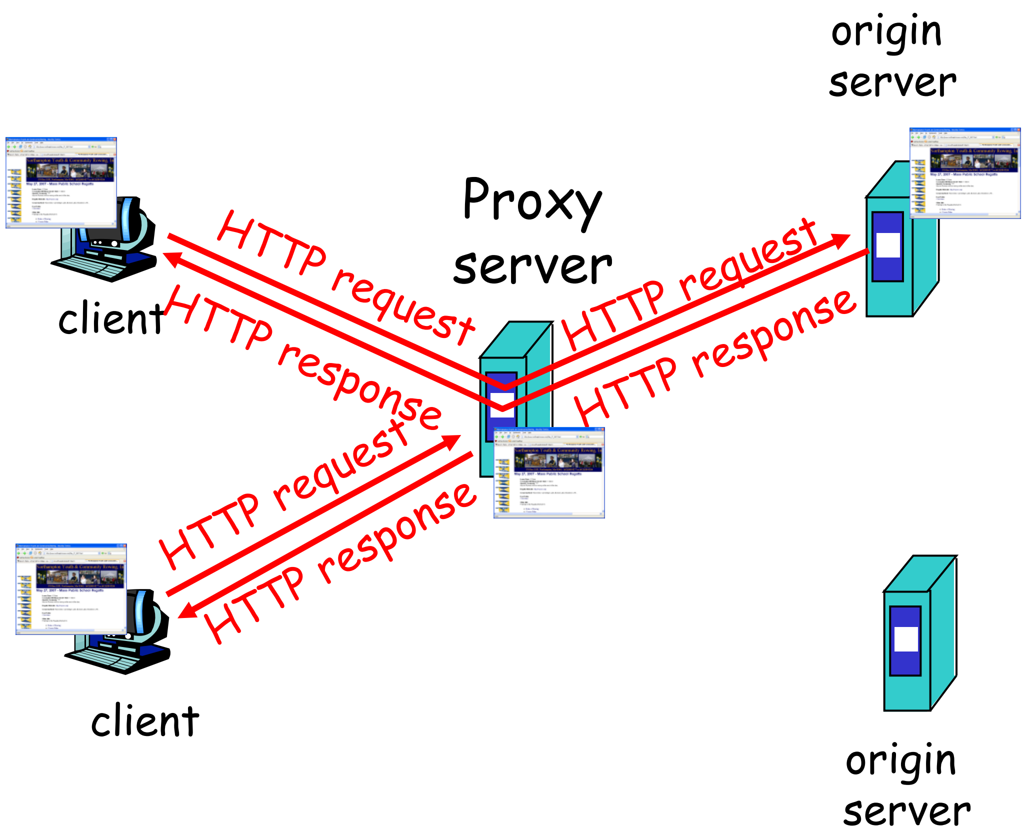
Why Web caching?

- reduce response time for client request
- reduce traffic on an institution's access link.
- Can be integrated with CDN (replication)

Web caches (proxy server)

Goal: satisfy client request without involving origin server

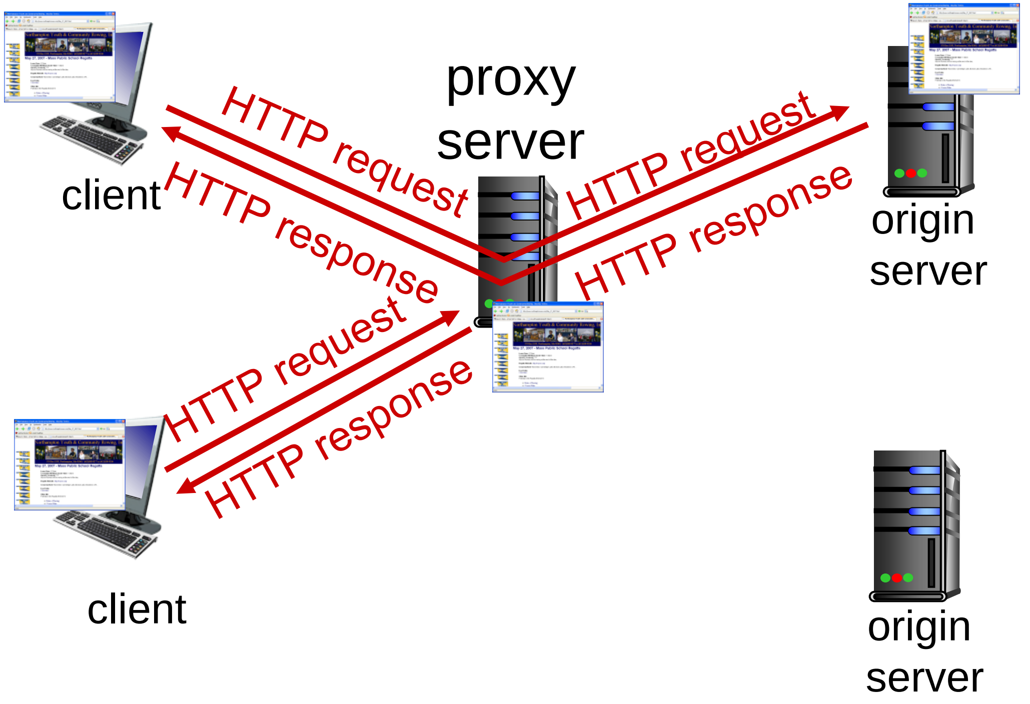
- user sets browser: Web accesses via cache
- browser sends all HTTP requests to cache
  - object in cache: cache returns object
  - else cache requests object from origin server, then returns object to client



Web caches (proxy server)

goal: satisfy client request without involving origin server

- user sets 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



More about Web caching

- cache acts as both client and server
  - server for original requesting client
  - client to origin server
- typically cache is installed by ISP (university, company, residential ISP)

why Web caching?

- reduce response time for client request
- reduce traffic on an institution's access link
- Internet dense with caches: enables "poor" content providers to effectively deliver content (so too does P2P file sharing)

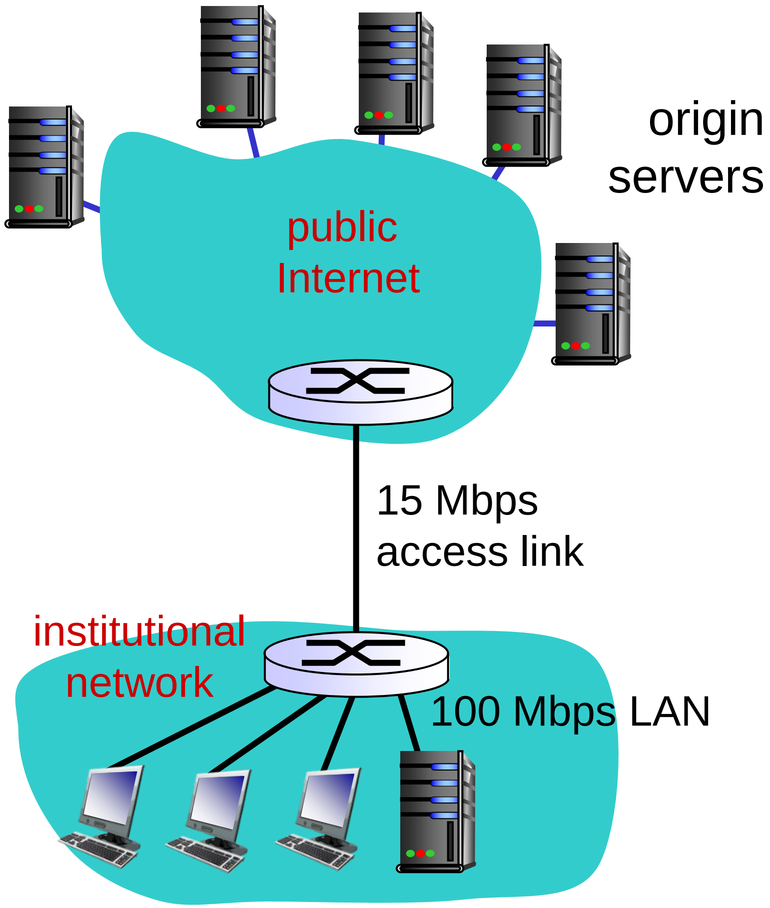
Caching example:

assumptions:

- avg object size: 1 Mbits
- avg request rate from browsers to origin servers:15/sec
- avg data rate to browsers: 15 Mbps
- RTT from institutional router to any origin server: 2 sec
- access link rate: 15 Mbps

consequences:

- LAN utilization: 15% *problem!*
- access link utilization = 100%
- total delay = Internet delay + access delay + LAN delay
  = 2 sec + minutes ? + usecs



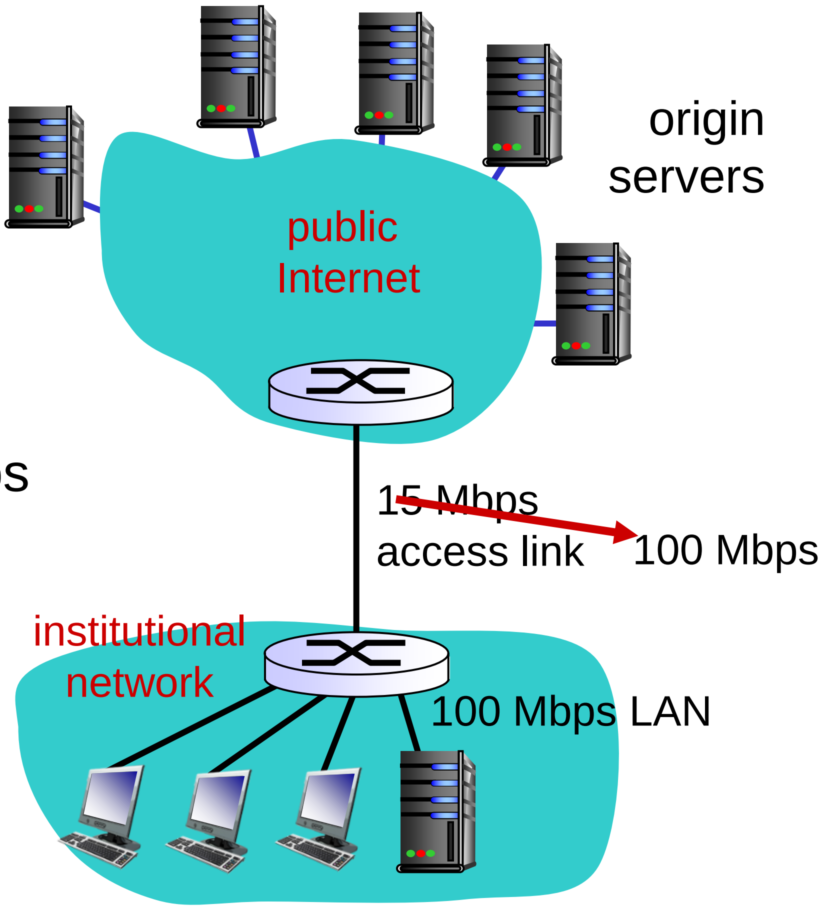
Caching example:

assumptions:

- avg object size: 1 Mbits
- avg request rate from browsers to origin servers:15/sec
- avg data rate to browsers: 15 Mbps
- RTT from institutional router to any origin server: 2 sec
- access link rate: ~~15 Mbps~~ 100 Mbps

consequences:

- LAN utilization: 15%
- access link utilization = ~~100%~~ 15%
- total delay = Internet delay + access delay + LAN delay
  = 2 sec + ~~minutes~~ ? + usecs
  msecs



Caching example: install local cache

assumptions:

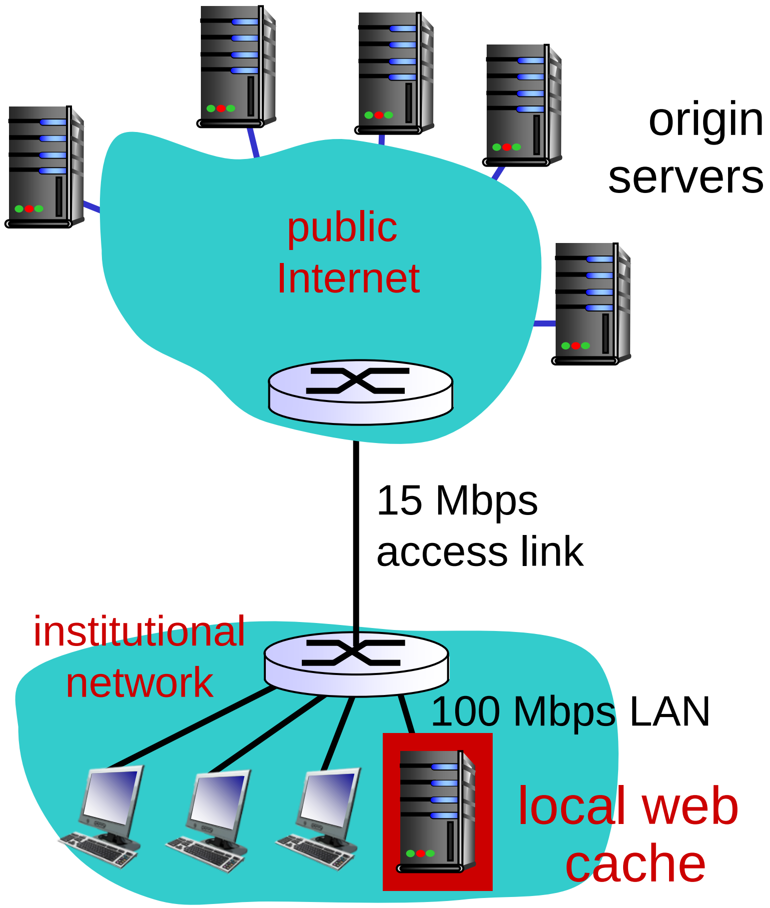
- avg object size: 1Mbits
- avg request rate from browsers to origin servers:15/sec
- avg data rate to browsers: 15 Mbps
- RTT from institutional router to any origin server: 2 sec
- access link rate: 15 Mbps

consequences:

- LAN utilization: 15%
- access link utilization ?
- total delay = ?

How to compute link utilization, delay?

Cost: web cache (cheap!)

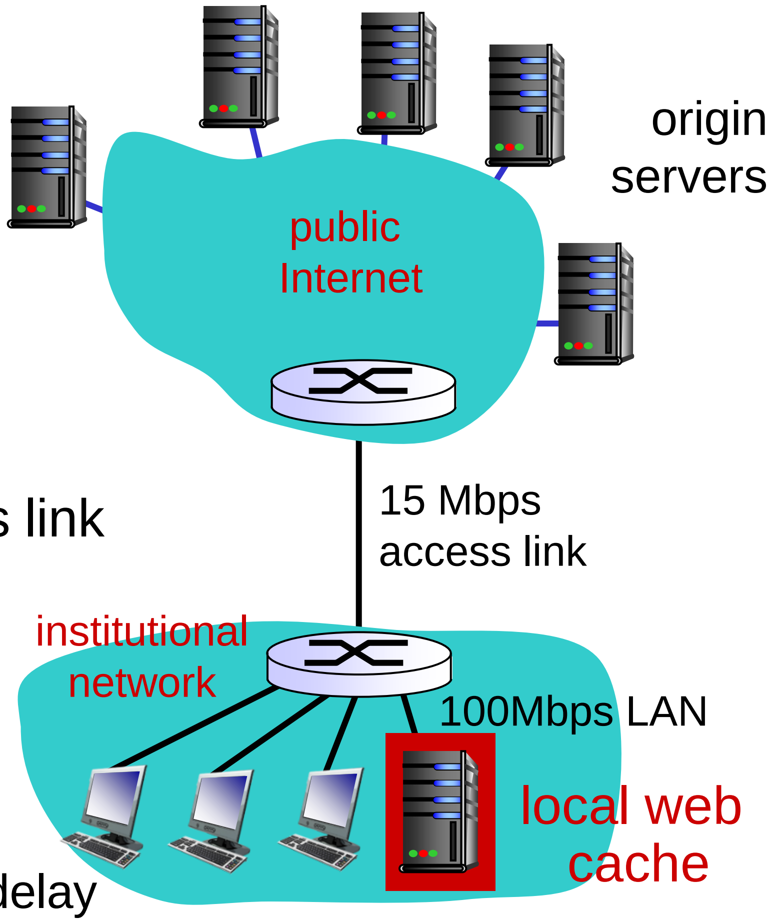


Caching example: install local cache

Calculating access link utilization, delay with cache:

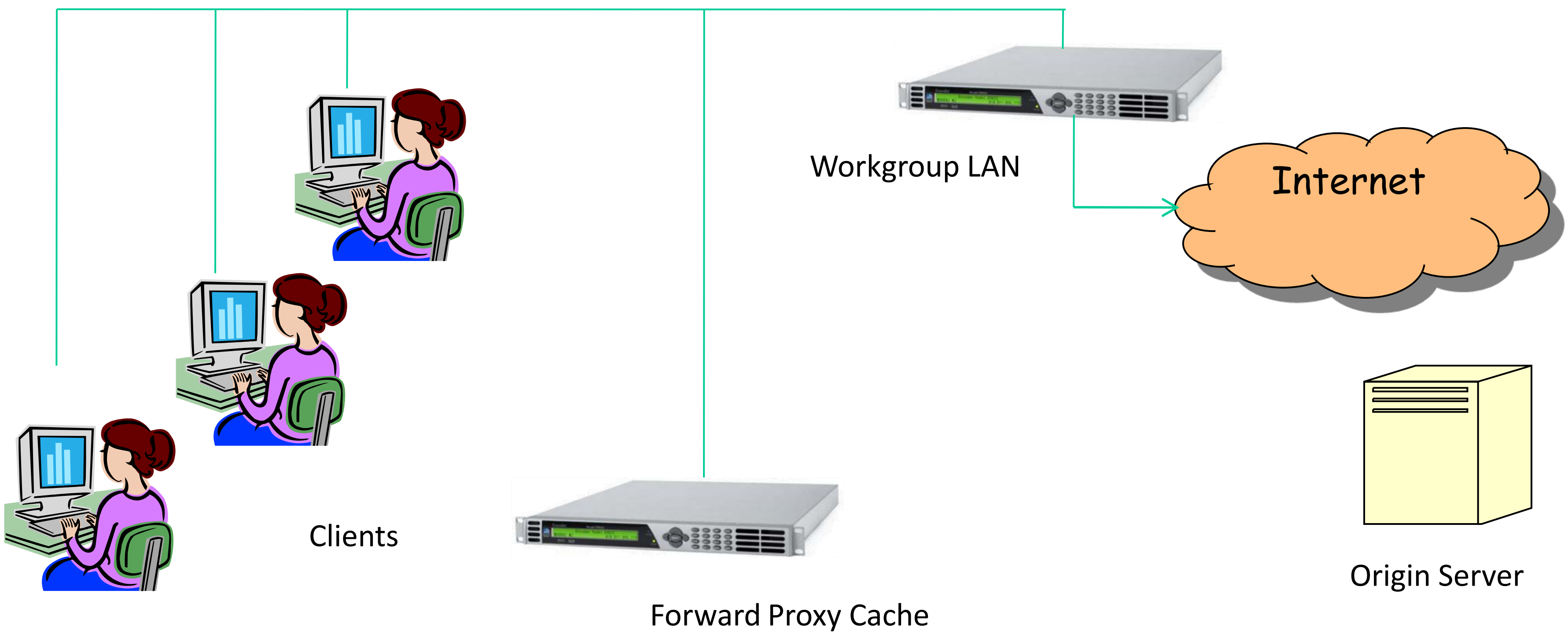
- suppose cache hit rate is 0.4
  - 40% requests satisfied at cache, 60% requests satisfied at origin

- access link utilization:
  - 60% of requests use access link
- data rate to browsers over access link = 0.6*15 Mbps = 9 Mbps
  - utilization = 9/15 = 0.6



- total delay
  - = 0.6 * (delay from origin servers) +0.4 * (delay when satisfied at cache)
  - = 0.6 (2.01) + 0.4 (~msecs) = ~ 1.2 secs
  - less than with 100 Mbps link (and cheaper too!)

Cache Types: Forward Proxy



Forward Proxy acts on behalf of content consumers

The Request is first sent over the LAN to the Forward Proxy

Network administrators set up Forward Proxy to help speed up web access for users

Try This!

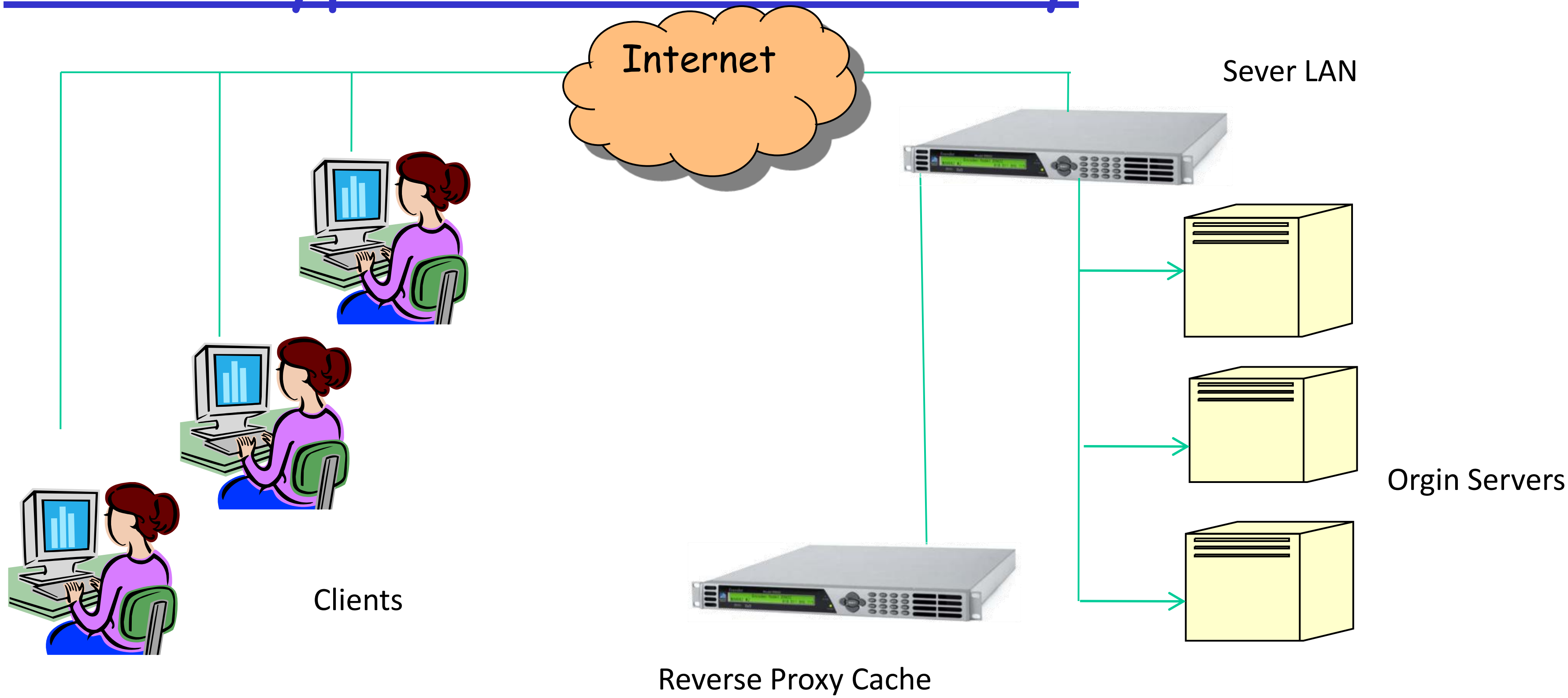
Find out the forward proxy address of your workgroup

Open your Browser

(For IE) Tools>Internet Options>Connections>LAN settings

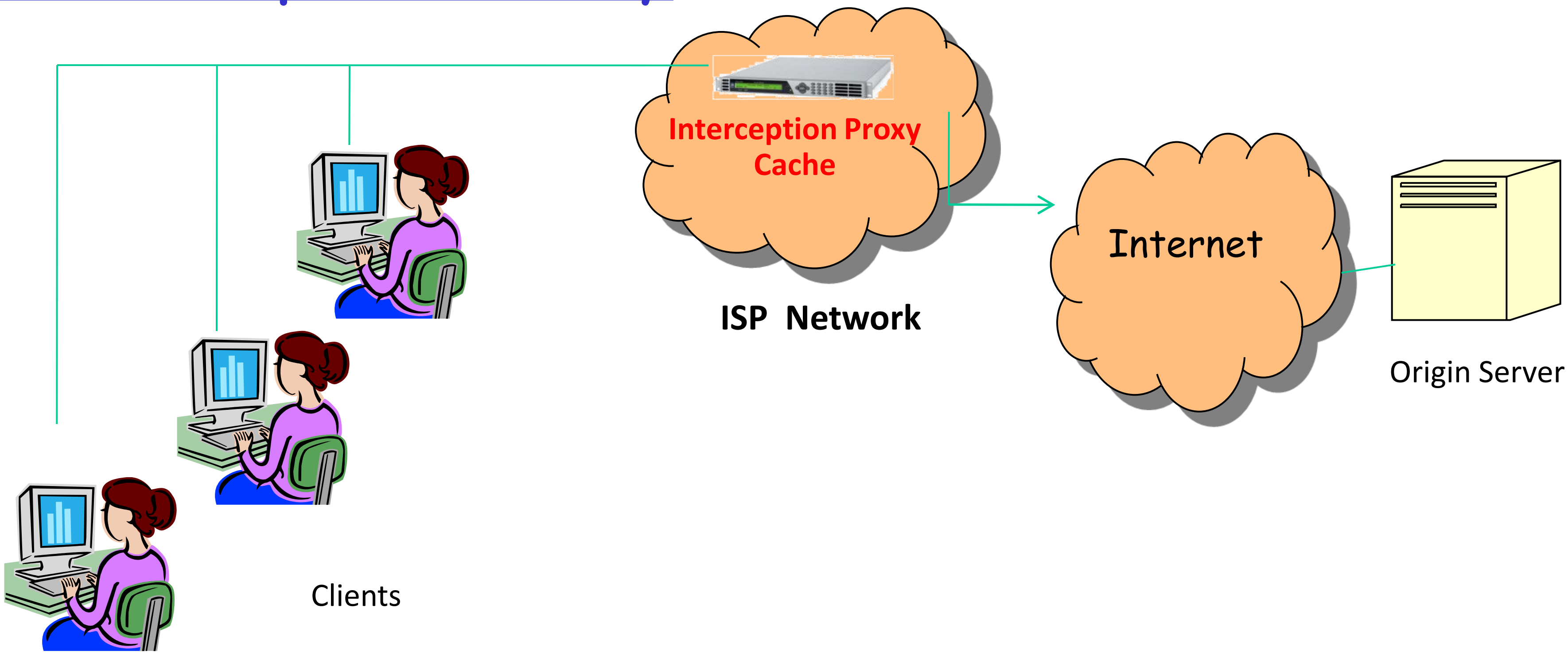
You can see your Proxy address

Cache Types : Reverse Proxy



- **Also called sever accelerator Reverse Proxy acts on behalf of origin server**
- ***The Request is sent first to the reverse proxy cache***
- **Web Farms set up reverse proxies to improve performance &scalability.**

Interception Proxy



- Interception Proxy acts on behalf of the ISP
- *The Request is first sent over the ISP Network to the Interception Proxy*
- ISPs set up Interception Proxy to help speed up web access for customers and reduce wide area bandwidth costs

Caching vs. Replication

- Motivations for moving content close to users
  - Reduce latency for the user
  - Reduce load on the network and the server
  - Reduce cost for transferring data on the network
- Caching
  - Replicating the content "on demand" after a request
  - Storing the response message locally for future use
  - May need to verify if the response has changed
  - … and some responses are not cacheable
- Replication (basic CDN Solution)
  - Planned replication of the content in multiple locations
  - Can replicate scripts that create dynamic responses
- Hybrid (reactive+proactive) solutions of caching and replication possible

Caching vs. Replication (Continued)

- Caching initially viewed as very important in HTTP
  - Many additions to HTTP to support caching
  - … and, in particular, cache validation
- Deployment of caching proxies in the 1990s
  - Service providers and enterprises deployed proxies
  - … to cache content across a community of users
  - Though, sometimes the gains weren't very dramatic
- Then, content distribution networks emerged
  - Companies (like Akamai) that replicate Web sites
  - Host all (or part) of a Web site for a content provider
  - Place replicas all over the world on many machines

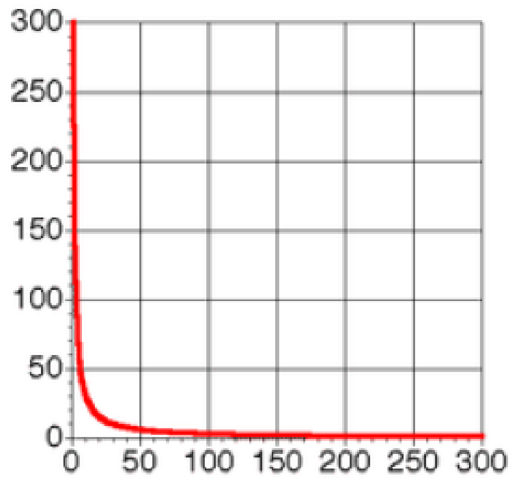
Content Characterization

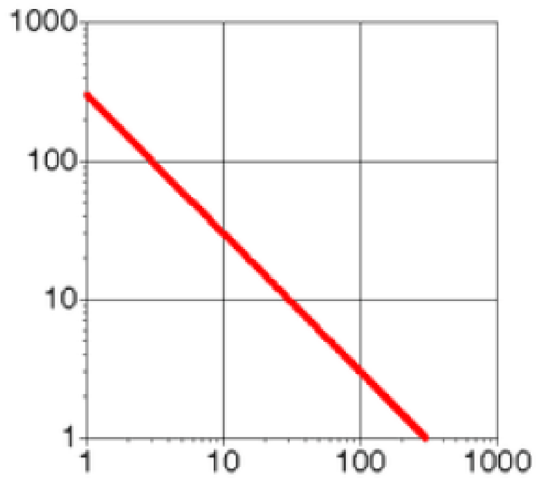
- **Size** (Bytes):
  - Short-tail distribution, e.g. exponential:
  - Long tail distribution, e.g. Parto:
- **Coding**:
  - Coding standard
  - Rate
- **QoS Requirement**
  - Loss/delay tolerance
- Cachability:
  - Static
  - Dynamic : role of TTL
- Popularity: e.g. Zipf distribution (we will return to this)
- Semantics: role of context

Content Request Characterization

- Request Arrival Process:
  - Examples
    - Poisson (exponential inter-arrivals): $P(N_T = k) = (\lambda T)^k e^{-\lambda T}/k!$
    - Modulated Poisson
- Temporal Locality
  - **Clustering in time:** requests for a certain item are correlated in time
- Spatial Locality
  - **Clustering in space:** requests for a group of items are correlated

Zipf's Law





- Zipf's law: The frequency of an event P as a function of rank i is a power law function:

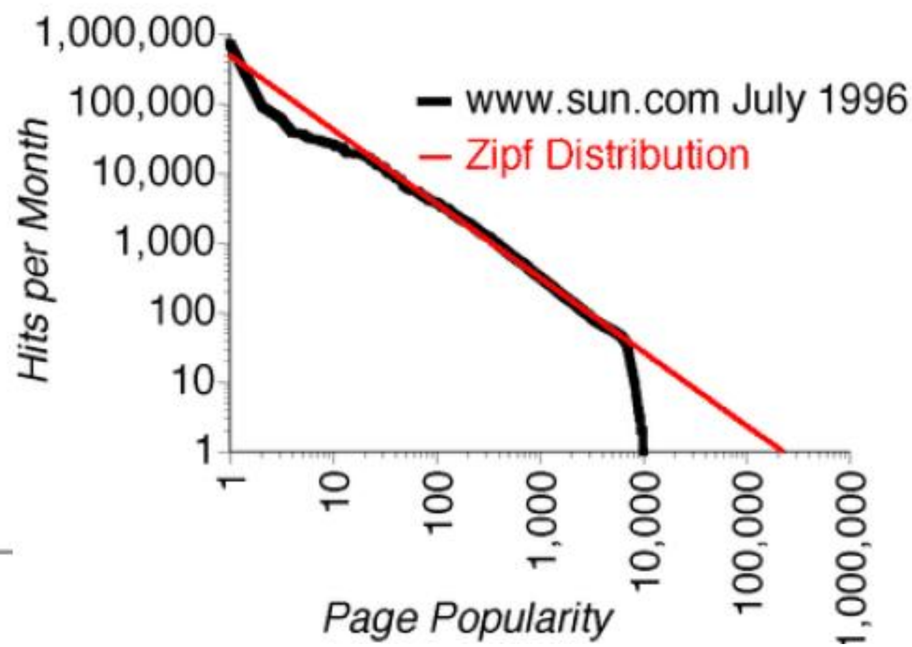
$$P_i = \Omega / i^{\alpha}, \qquad \text{where } \alpha > 0$$

Zipf's Law

- Observed to be true for
  - Frequency of written words in English texts
  - Population of cities
  - Income of a company as a function of rank

Zipf's Law vs. Web Access

- For a given server, page access by rank follows Zipf's law
- Web requests from a fixed population of users follows Zipf's law $0.64 < \alpha < 0.83$

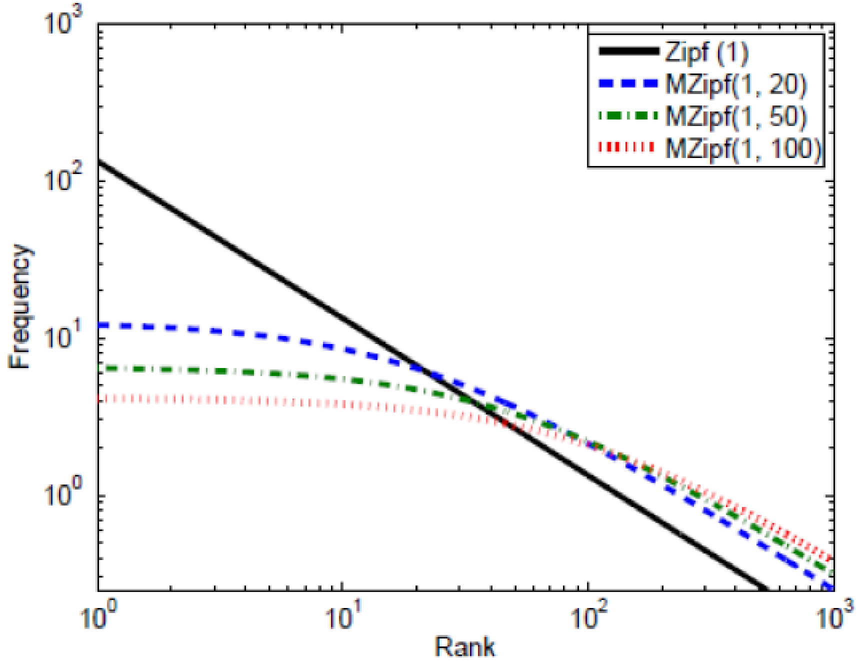


Observations

- Top %1 of all documents account for %20 - %35 of proxy requests
- Top %10 account for %45 - %55 of requests
- It takes %25 to %40 of all documents to account for %70 of requests
- It takes %70 to %80 of all documents to account for %90 of requests

Zipf-Mandelbrot Distribution

- Some content has popularity distribution with more flattened head



Source: O. Saleh and M. Hefeeda

Zipf versus Mandelbrot-Zipf for different q (plateau factor) values.

The Zipf-Mandelbrot Distribution

Empirical data suggests that rank statistics resemble Zipf-Mandelbrot distribution

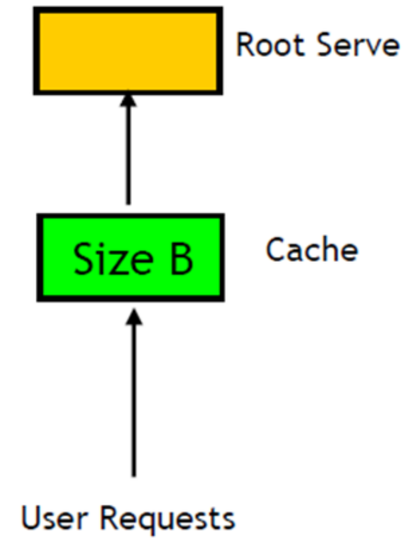
Relative frequency of i-th most popular item is

$$p_i = \frac{K}{(q+i)^\alpha}, \qquad i = 1, \ldots, N,$$

with

- $\alpha \geq 0$: shape parameter
- $q \geq 0$: shift parameter
- $K = \left[\sum_{i=1}^{N} \frac{1}{(q+i)^\alpha}\right]^{-1}$ normalization constant

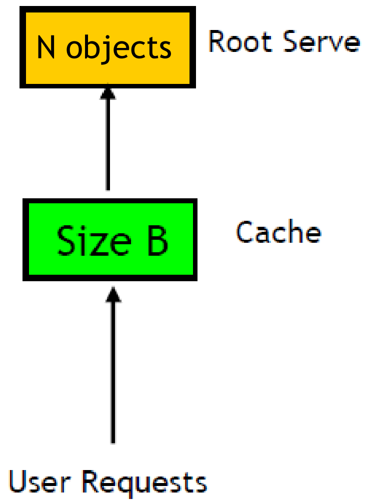
Cache Performance



- **cache hit:** requested object is found in the cache.
- **cache miss:** requested object is not found in the cache
- **miss cost:** cost of retrieving object from origin
- **hit rate:** fraction of requests served from cache
- **miss rate:** (1 - hit rate)

Hit Rate Calculation - Single-Level Caching

- Total number of files=N
- Object sizes are identical=1 unit
- Probability a request is for file i is p_i (Zipf)
- Cache size= B units
- Request arrival rate = λ
- Optimal caching policy: cache the most popular B files

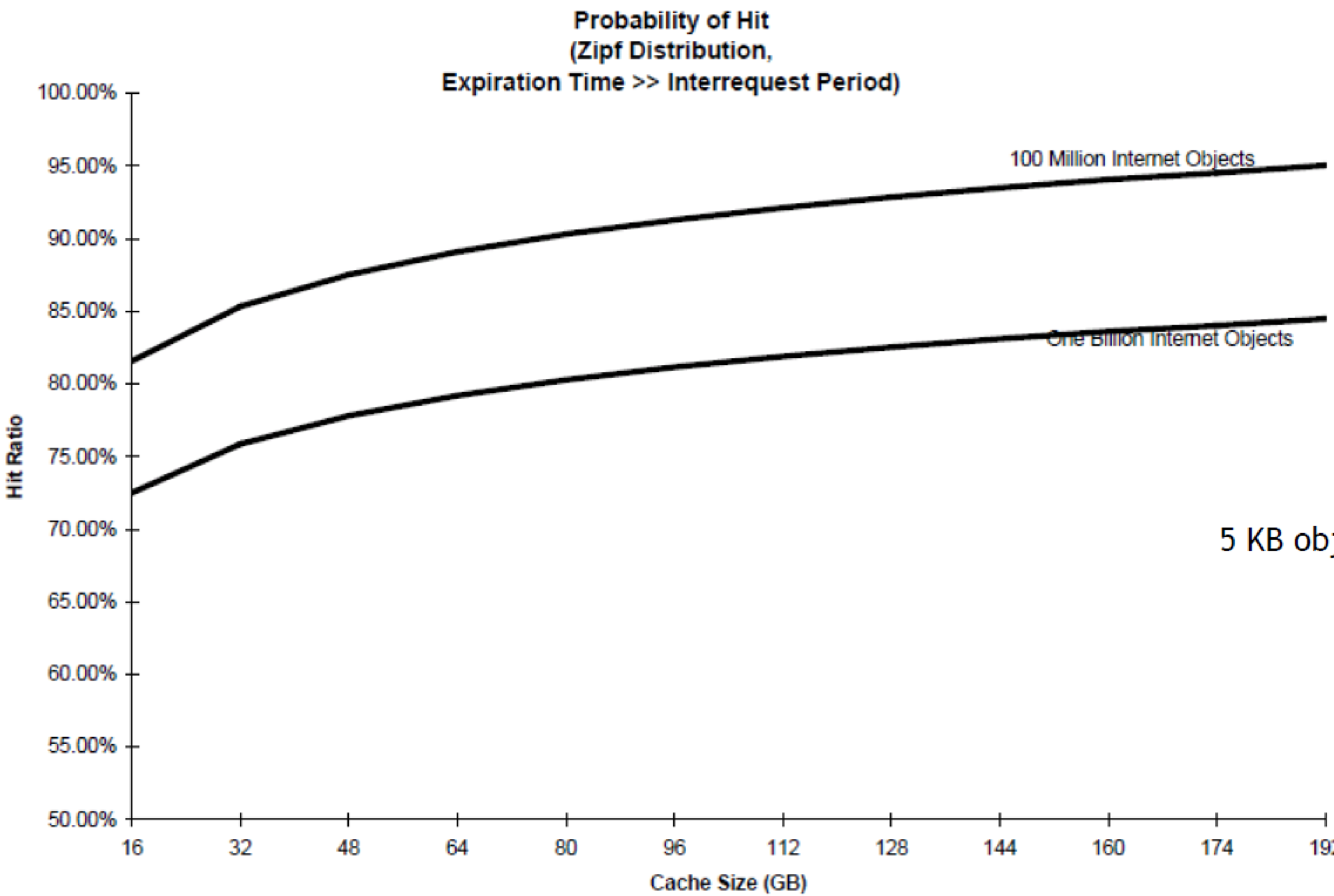


Hite rate = H = $\sum_{i=1}^{B} p_i = \sum_{i=1}^{B} \frac{\Omega}{i^\alpha}$ where $\Omega = \left[\sum_{i=1}^{N} \frac{1}{i^\alpha}\right]^{-1}$

Load on server = $\lambda(1 - H)$

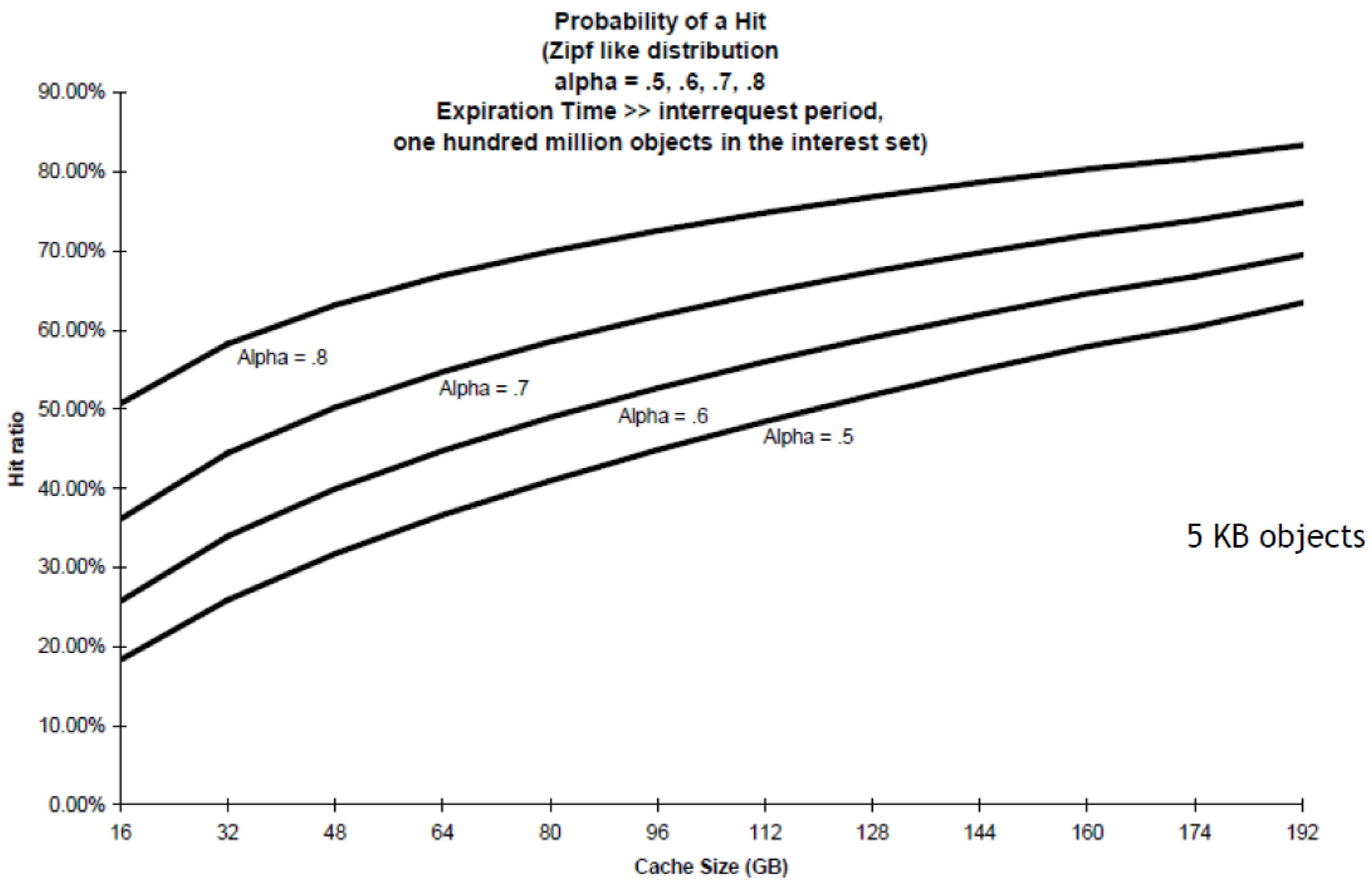
Asymptotics: for large B, N, and $\alpha = 1$, H grows like $log(B)$

Effect of Cache Size on Hit Rate



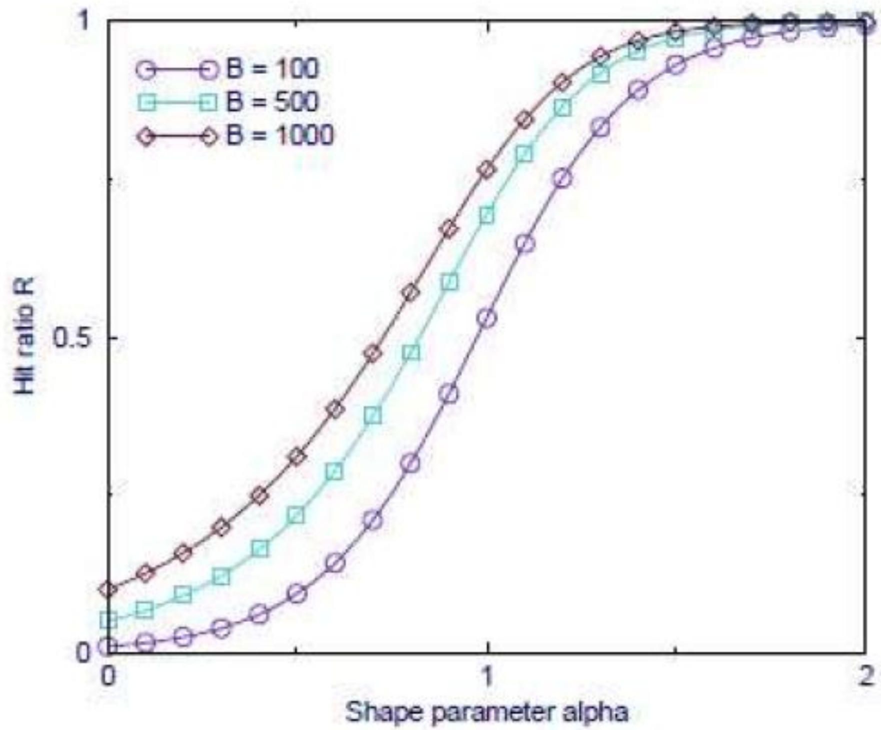
5 KB objects

Effect of alpha on Hit rate



Effect of α on the Hit rate

Shape parameter α varies with content type, and strongly impacts cache effectiveness

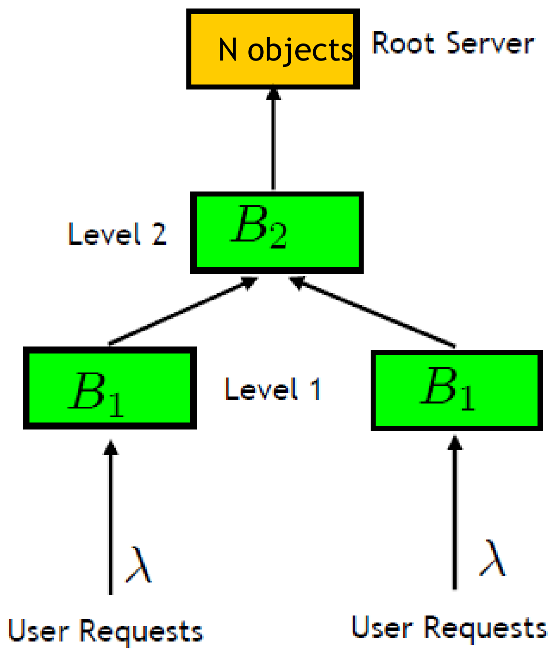


Hit ratio as function of shape parameter α for various cache sizes B and population of $N = 10,000$ content items

For large B, N, and $\alpha = 1$, H grows like $log(B)$

Hit Rate Calculation - Hierarchical Caching

- Caching Policy
  - Level-1: cache the most popular objects
  - Level-2: cache the less popular objects
- $B_1 + B_2 < N$
- Assume identical arrivals to each cache at level-1
- Compute hit rate at level-1, H_1
- Arrival rate at level-2 = $2\lambda(1 - H_1)$
- Request rate at root server= $2\lambda(1 - H_1)(1 - H_2)$



$$H_1 = \sum_{i=1}^{B_1} p_i = \sum_{i=1}^{B_1} \frac{\Omega}{i^\alpha}$$

$$H_2 = \sum_{i=B_1+1}^{B_1+B_2} \theta p_i \qquad \text{where} \quad \theta = \left[\sum_{i=B_1+1}^{N} p_i \right]^{-1}$$

Exercise:

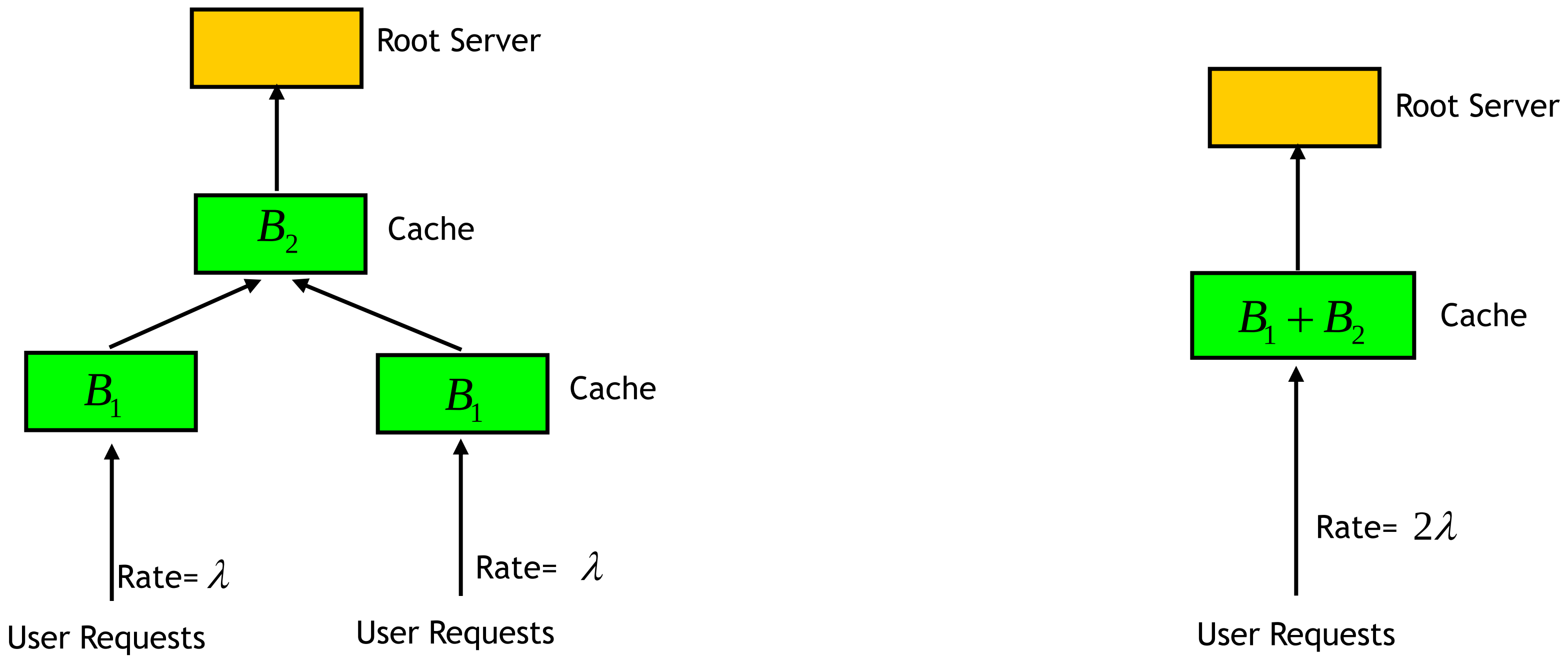
Consider the two networks shown. Show that the request arrival rate at the root server is the same for both networks (derive relevant expressions for both networks and compare). What are the advantages and disadvantages of each network . Assume for each network:

Total number of files (stored in the root server) = N

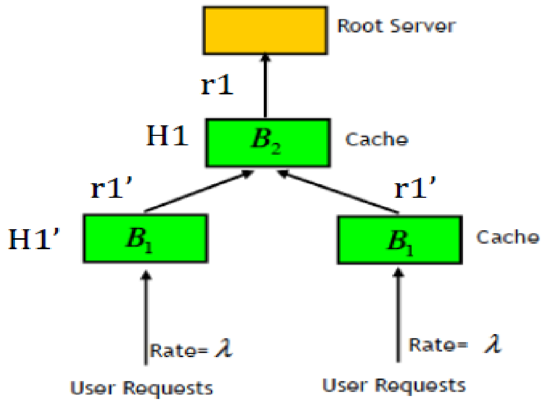
File sizes are identical = 1 unit

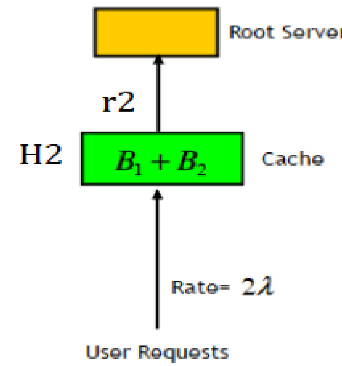
Probability a request is for file i is p_i (Zipf distributed)

Cache sizes are as shown (in the green boxes)



Solution:





$$r1 = 2\lambda(1 - H1')(1 - H1)$$

$$r2 = 2\lambda(1 - H2)$$

$$1 - H1' = 1 - \sum_{i=1}^{B1} pi$$

$$1 - H2 = 1 - \sum_{i=1}^{B1+B2} pi$$

$$1 - H1 = 1 - \sum_{i=B1+1}^{i=B1+B2} pi \Big/ \sum_{i=B1+1}^{N} pi$$

$$(1 - H1)(1 - H1') = (1 - \sum_{i=1}^{B1} pi)(1 - \sum_{i=B1+1}^{i=B1+B2} pi \Big/ \sum_{i=B1+1}^{N} pi)$$

$$= (1 - \sum_{i=1}^{B1} pi)(1 - \sum_{i=B1+1}^{i=B1+B2} pi \Big/ 1 - \sum_{i=1}^{B1} pi)$$

$$= 1 - \sum_{i=1}^{B1} pi - \sum_{i=B1+1}^{i=B1+B2} pi$$

$$= 1 - \sum_{i=1}^{B1+B2} pi$$

$$= 1 - H2$$

$$r1 = 2\lambda(1 - H1')(1 - H1) = 2\lambda(1 - H2) = r2$$

Variable size content

- Assume you know the popularity distribution p_i
- Assume object i has size s_i

- Problem: Find object placement that maximizes the hit rate:
  - NP-Complete
  - Known as “Knapsack problem”
- Greedy approximation:
  - Fill the cache with the objects having the highest ‘density” values p_i/s_i
  - Known to perform at most twice worse than the optimal solution

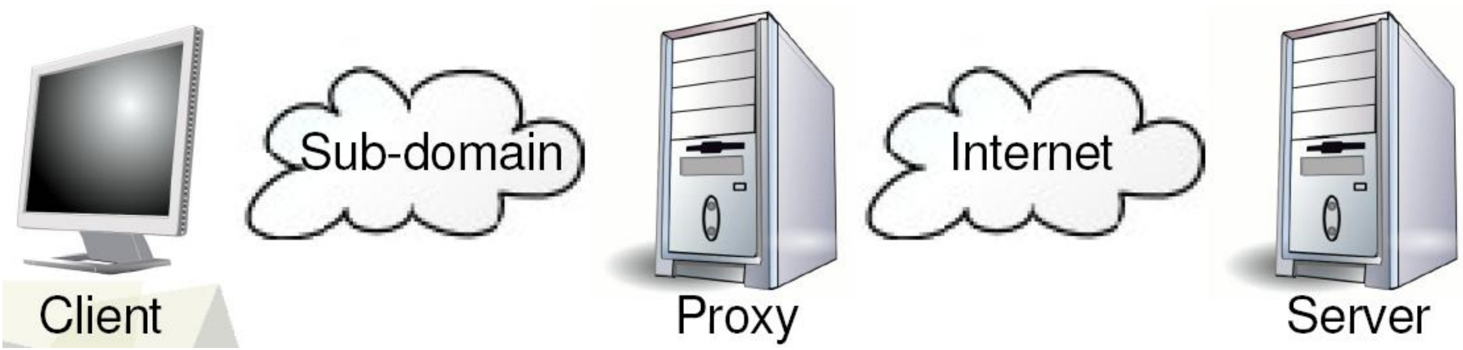
Extensions

- Different popularity distributions for different user communities
- Different arrival rates
- Caching with dynamic content (role of Time-To-Live (TTL))
- Cooperative Caching

Cache Consistency

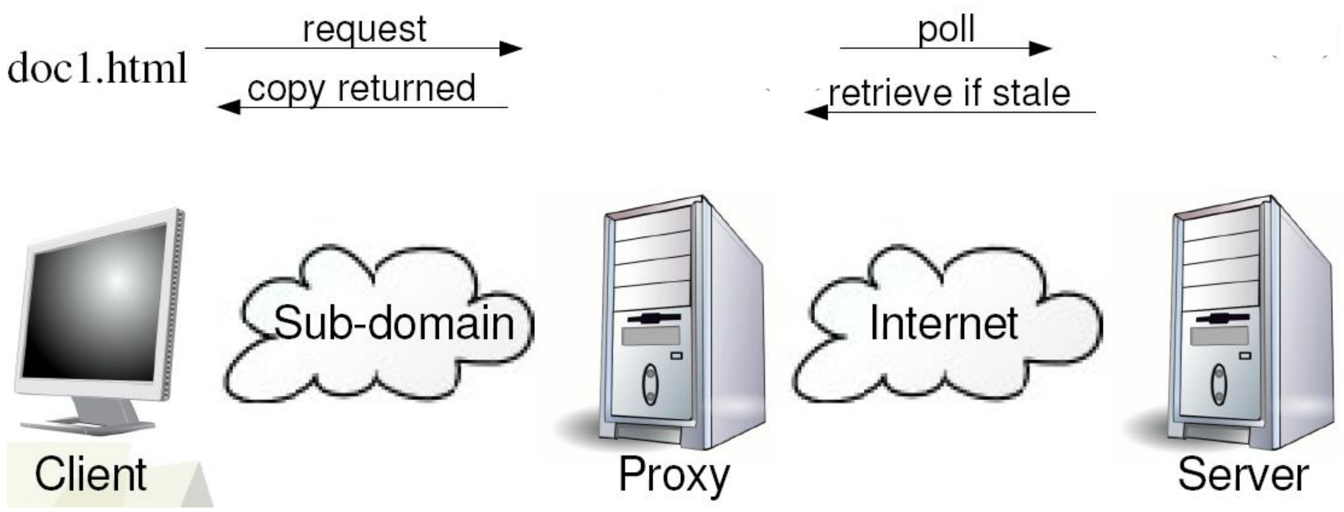
- Objects expire and need to be refreshed
  - How and when to refresh?

- Goals
  - Avoid stale objects
  - Keep non useful traffic as low as possible



Cache Consistency: Polling

Solution 1: Poll server for every request



Implemented in HTTP using the optional “if-modified-since" request header field

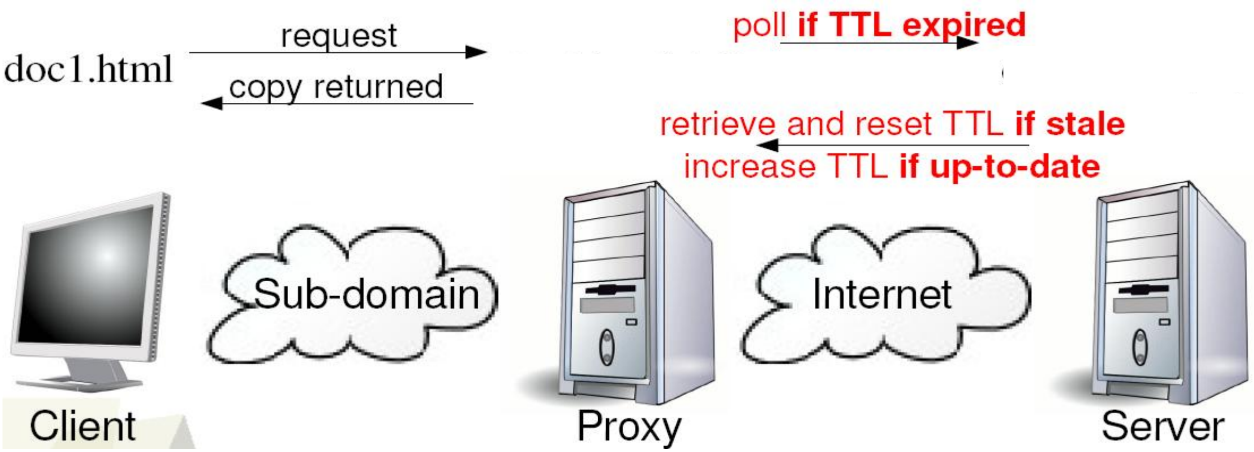
Benefit: strong consistency

Drawback: very slow cache hit

Cache Consistency: Polling

Solution 2: polling if TTL expires, widely used

- Associate a TTL (12 hours or 2 days) with each cached object



implemented in HTTP using the optional "expires" header field

Benefit: fast cache hit

Drawback: weak cache consistency (some staleness) due to TTL estimation error

- Cache analysis with Time to Live
- Parameters:
  - λ = request rate
  - μ = rate of object change
  - α = Zipf parameter (object popularity)

- The steady state hit rate is

$$\mathsf{H} = \sum_{1 \le i \le \mathsf{B}} p_i \frac{\lambda \quad p_i}{\lambda \quad p_i + \mu}$$

Taking p_i proportional to $1/i^{\alpha}$, a very close approximation to this sum is given by

$$\int_1^{\mathsf{B}} \frac{1}{Cx^{\alpha}} \left(\frac{1}{1 + \frac{\mu x^{\alpha} C}{\lambda}} \right) dx,$$

where

$$C = \int_{1 \le x \le n} \frac{1}{x^{\alpha}} dx.$$

The reality is …

- Popularity distribution:
  - May not be known accurately
  - Non-stationary
- Content:
  - Variable size
  - Variable cost
  - Dynamic, different expiration times
- Request rate:
  - May not be known accurately
  - Non-stationary
  - “use once” effect

Selecting an Object Replacement Algorithm

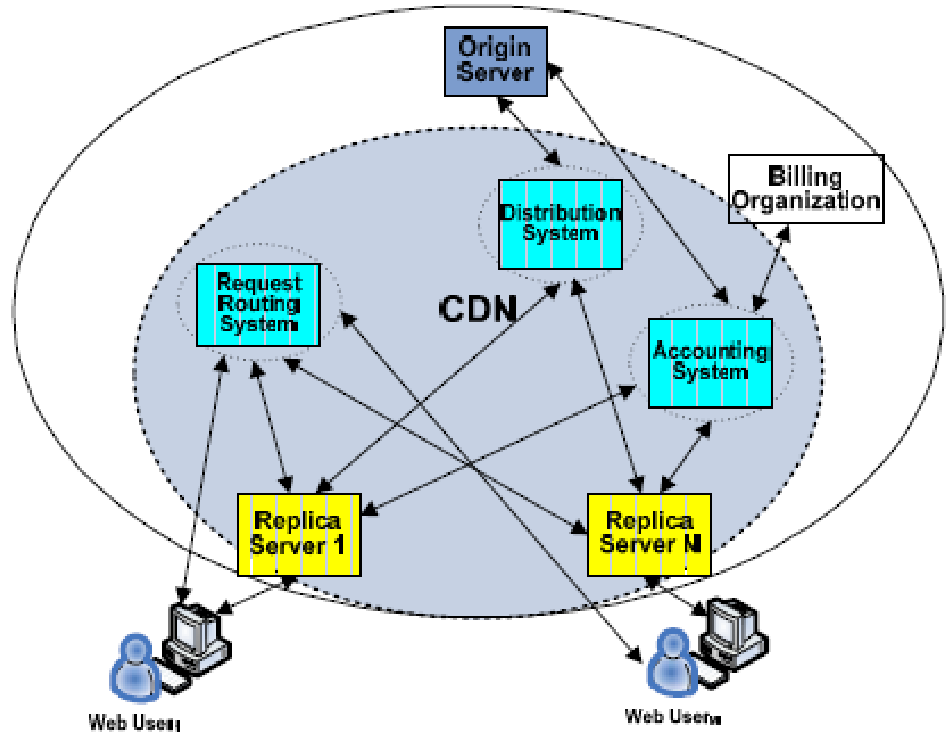
- The problem can be associated with a function that given:
  - The state of the cache, and
  - A newly retrieved document
- Decides the following:
  - Should the retrieved document be cached?
  - If yes and no space available, which existing entry to discard?
- State of the cache:
  - The set of documents stored
  - For each document, a set of state variables which typically include statistical information associated with the document

Replacement Rules for Items in the Cache

- Least Recently Used (LRU)
- First In First Out (FIFO)
- Least Frequently Used (LFU)
- Next to Expire (NTE)
- Largest File First (LFF)

CDN Architecture

- Content delivery component
  - Origin server and a set of edge servers (surrogates) to replicate content
- Request-routing component
  - Direct user requests to edge servers
  - Interact with the distribution component to keep an up-to-date view of content
- Content distribution component
  - Moves content from the origin to edge servers and ensures consistency
- Accounting component
  - Maintains logs of client accesses and records usage of the servers
  - Assists in traffic reporting and usage-based billing



Akamai

- **Research began at MIT in 1995**
- **Company founded in 1998**
- **The largest CDN provider to date**
  - 170,000 servers in located across nearly 1,300 networks in 102 countries
  - Handles 15-30% of today’s Internet traffic!
  - 85% of world's Internet users within a single network hop of an Akamai server
- **The market share leader (approx. 60%)**
- **Examples of service provided**
  - **managed edge services to content providers (**e.g. Apple, Best Buy, ESPN**)**
  - **Boosting cloud computing performance with edge computing**
- **Uses proprietary *mathematical algorithms* and patented technologies**

Request Routing – The Akamai Way

Typical Page

Total page 87,550 bytes
Total Akamai Served 68,756 bytes

Logos
3,395 bytes

Navigation Bar
9,674 bytes

Banner Ads
16,174 bytes

Gif links
22,395 bytes

Fresh Content
17,118 bytes

78% Page Served by Akamai

Source: Bruce Maggs, Akamai Technologies

HTML Title Page for www.xyz.com with Embedded objects

```
<html>
<head>
<title>Welcome to xyz.com!</title>
</head>
<body>
<img src="http://www.xyz.com/jpgs/navbar1.jpg">
<img src="http://www.xyz.com/jpgs/navbar1.jpg">
<h1>Welcome to our Web site!</h1>
<a href="page2.html">Click here to enter</a>
</body>
</html>
```

Content Delivery using Akamai

Embedded URLs are Converted to ARLs

<html>

<head>

<title>Welcome to xyz.com!</title>

</head>

<body>

ak

<img src=" http://~~www~~.xyz.com/logos/logo.gif">

<img src=" http://~~www~~.xyz.com/jpgs/navbar1.jpg">

<h1>Welcome to our Web site!</h1>

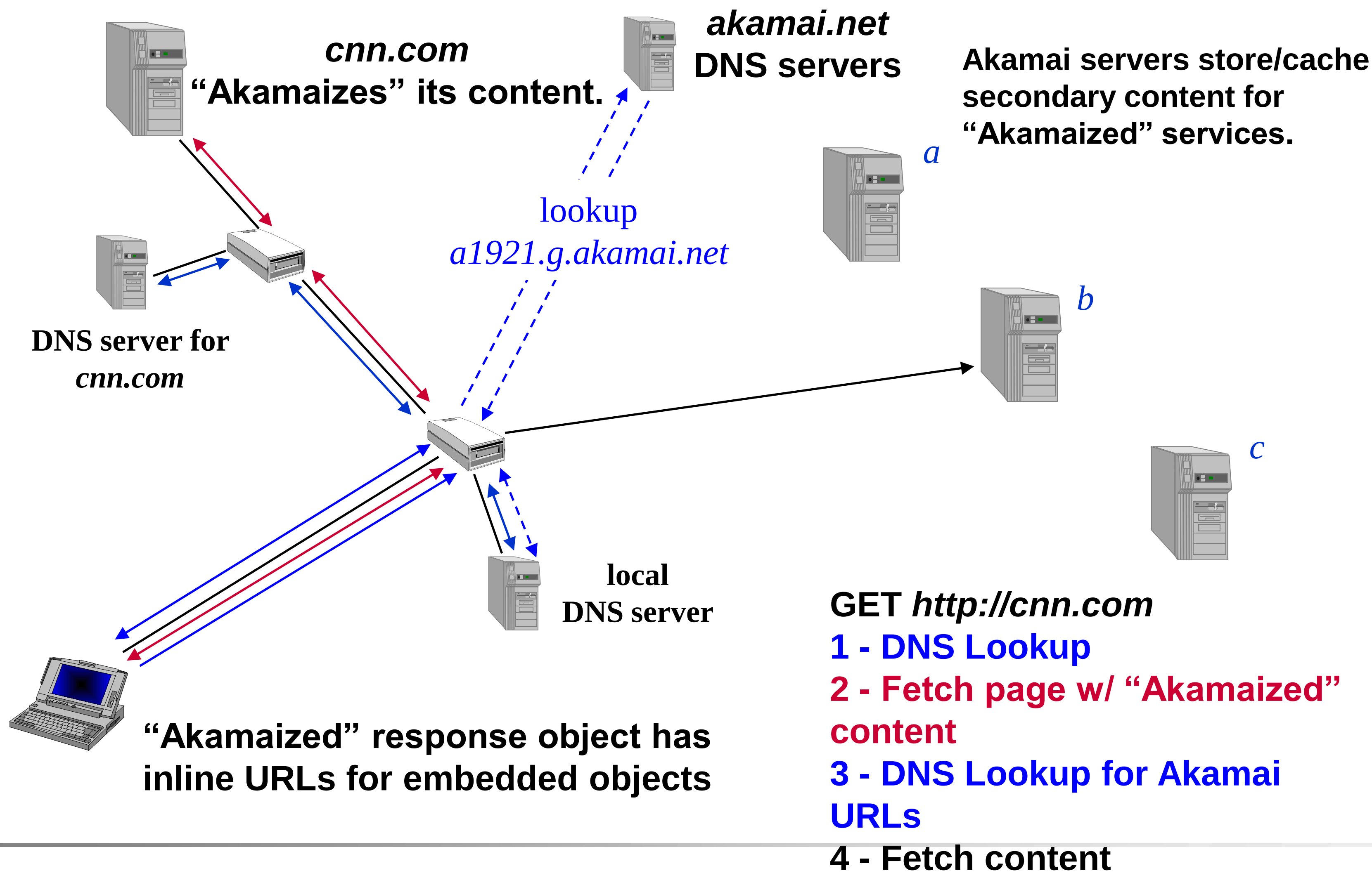
Click here to enter

</body>

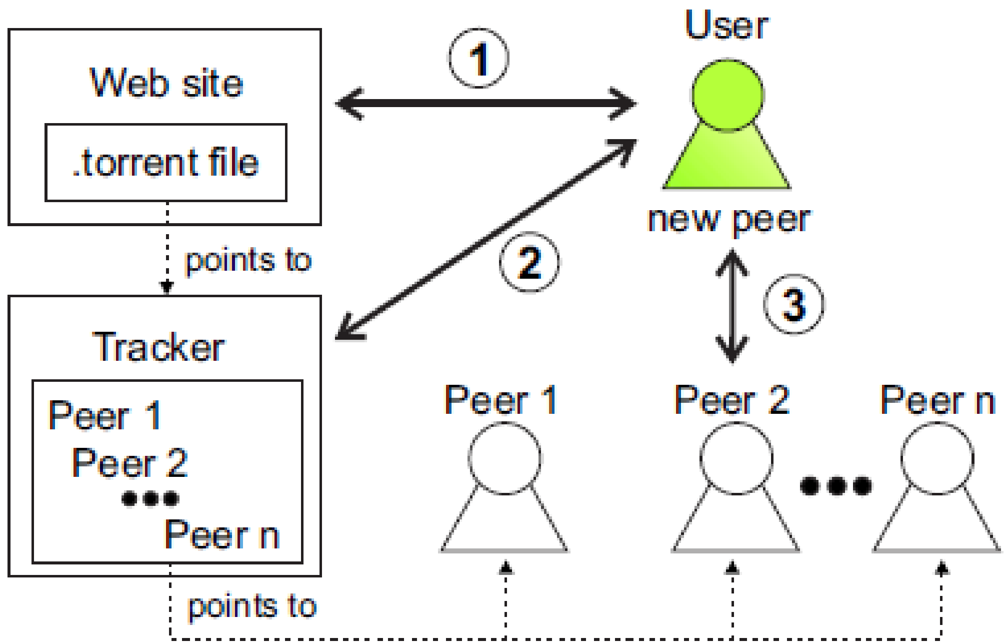
</html>

Source: Bruce Maggs, Akamai Technologies

Detailed example: DNS-based Request Routing



P2P - BitTorrent



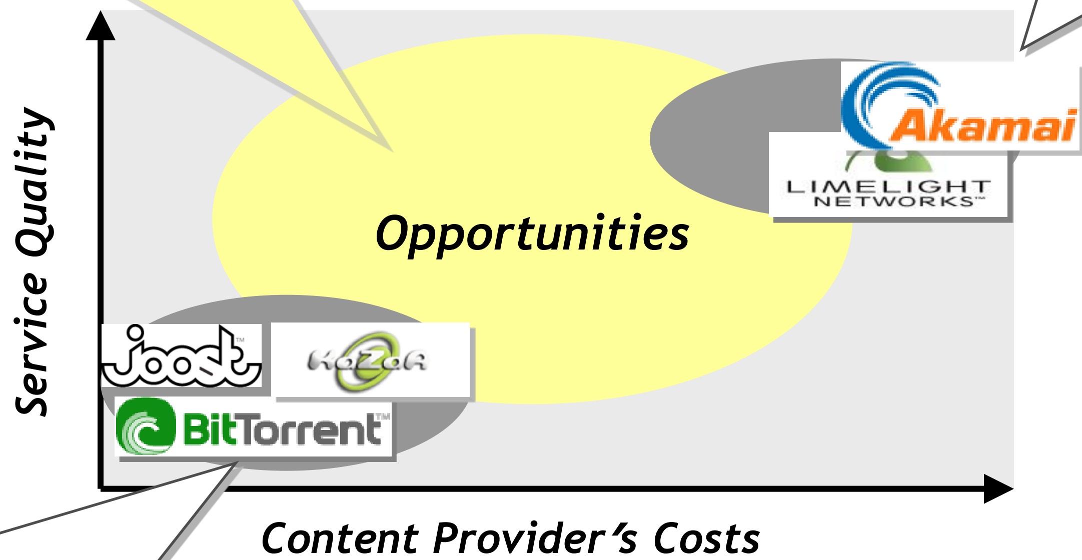
- Distributed file sharing technology, optimized for large files
- A peer acts as a client and a server at the same time
- Content pieces are efficiently and fairly distributed/shared
- Performance: can be efficient, but unpredictable
- Analysis methods: queuing models, fluid models

Hybrid Content Distribution Architectures

Hybrid Networks

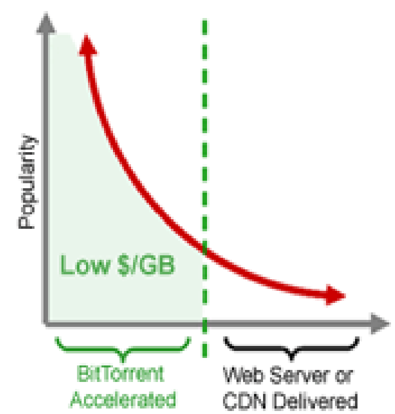
- Combine the best of peer-to-peer and traditional content distribution

Traditional Content Distribution Networks



Peer-to-Peer Networks

BitTorrent DNA



Hybrid CDN/P2P Services