WEB CACHES

(v 0.1)

PXC

Manel Guerrero <guerrero@ac.upc.edu>

Contents

- Web Caches
- Kinds of Web Caches
- How do they Work?
- Internet Traffic
- Content Distribution Networks

Sources

(That is, places from which we've done merciless cut 'n' pastes)

- www.mnot.net/cache_docs/
- http://www.wikipedia.org/
- Kurose slides
- Previous FIB-AAD Slides.

What's a Web Cache?

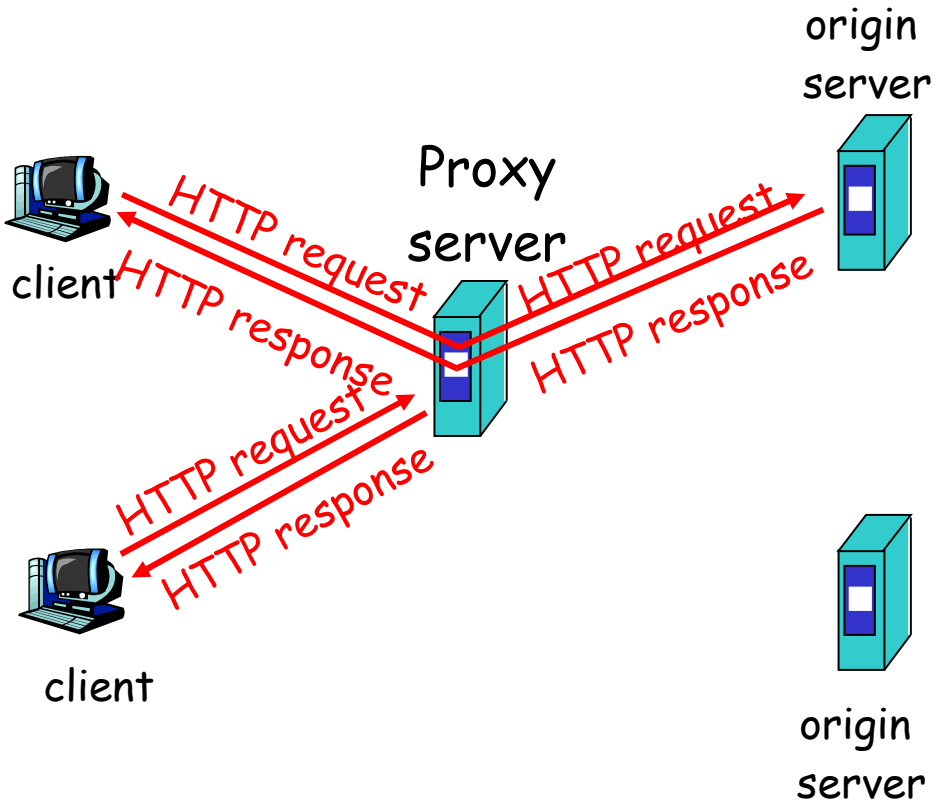
- A **Web cache** sits between one or more Web servers (also known as **origin servers**) and a **client** or many clients, and watches requests come by, **saving copies of the responses**, like HTML pages, images and files (collectively known as **representations**), for itself.

- Then, if there is another request for the same URL, it can use the response that it has, instead of asking the origin server for it again.

Example: Proxy Caches

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



Why a Web Cache?

- **To reduce latency:** Because the request is satisfied from the cache (which is closer to the client) instead of the origin server, it takes less time for it to get the representation and display it. This makes the Web seem more responsive.

- **To reduce network traffic:** Because representations are reused, it reduces the amount of bandwidth used by a client. This saves money if the client is paying for traffic, and keeps their bandwidth requirements lower and more manageable.

Kinds of Web Caches (1/2)

- **Browser Caches:** Used when you visit the same page again and when the same images are used throughout the same site.
- **Proxy Caches:** Large corporations and ISPs often set them up on their firewalls.
  - Either you point your browser's proxy setting to the proxy or you use interception. Interception proxies have requests Web requests redirected to them by the underlying network itself.
  - Proxy caches are very good at reducing latency and network traffic.

Kinds of Web Caches (2/2)

- **Gateway Caches:** Also known as 'reverse proxy caches' or 'surrogate caches', gateway caches are also intermediaries, but instead of being deployed by network administrators to save bandwidth, they're typically **deployed by Webmasters** themselves, to make their **sites more scalable, reliable and better performing**.
  - Requests can be routed to gateway caches by a number of methods, but typically some form of **load balancer** is used to make them look like the origin server to clients.
  - **Content Delivery Networks** (CDNs) distribute gateway caches throughout the Internet and sell caching to interested Web sites. Examples: Speedera and Akamai.

How Web Caches Work? (1/2)

1. If the response's headers tell the cache not to keep it, it won't.
2. If no validator (an **ETag** or **Last-Modified** header) is present on a response, it will be considered uncacheable.
3. If the request is **authenticated** or **secure**, it won't be cached.
4. A **cached representation is considered fresh** (able to be sent to a client without checking with the origin server) if:
    - It has an expiry time or other age-controlling header set, and is still within the fresh period.
    - If a browser cache has already seen the representation.
    - If a proxy cache has seen the representation recently.

How Web Caches Work? (2/2)

5. Fresh representations are served directly from the cache, without checking with the origin server.
6. If an **representation** is **stale**, the origin server will be asked to **validate** it, or tell the cache whether the copy that it has is still good.
7. Together, **freshness** and **validation** are the most important ways that a cache works with content. A fresh representation will be available instantly from the cache, while a validated representation will avoid sending the entire representation over again if it hasn't changed.

Caching and HTTP Headers

- HTTP headers tell how browser and proxy caches handle your representations. They are usually automatically generated by the Web server. You can configure your server to do it according to your needs.
- HTTP headers are sent before the HTML (separated by a blank line). Typical HTTP 1.1 response headers:

```
HTTP/1.1 200 OK
Date: Fri, 30 Oct 1998 13:19:41 GMT
Server: Apache/1.3.3 (Unix)
Cache-Control: max-age=3600, must-revalidate
Expires: Fri, 30 Oct 1998 14:19:41 GMT
Last-Modified: Mon, 29 Jun 1998 02:28:12 GMT
ETag: "3e86-410-3596fbbc"
Content-Length: 1040
Content-Type: text/html
```

Validators

- **Last-Modified**: The most common validator is the time that the document last changed. When a cache has an representation stored that includes a Last-Modified header, it can use it to ask the server if the representation has changed since the last time it was seen, with an **If-Modified-Since request**.
- HTTP 1.1 introduced a new kind of validator called the **ETag**. ETags are unique identifiers that are generated by the server and changed every time the representation does. Because the server controls how the ETag is generated, caches can be surer that if the ETag matches when they make a **If-None-Match request**, the representation really is the same.
- Most modern Web servers will generate both ETag and Last-Modified validators for static content automatically. But not for dynamic content.

Request to validate a representation

```
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.intel-iris.net
Connection: Keep-Alive
```

Reply to the request

```
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"
```

- 304: Not modified Therefore, the cached representation has been validated.
- Note that the ETag is the same.

Cache-Control Response Headers (1)

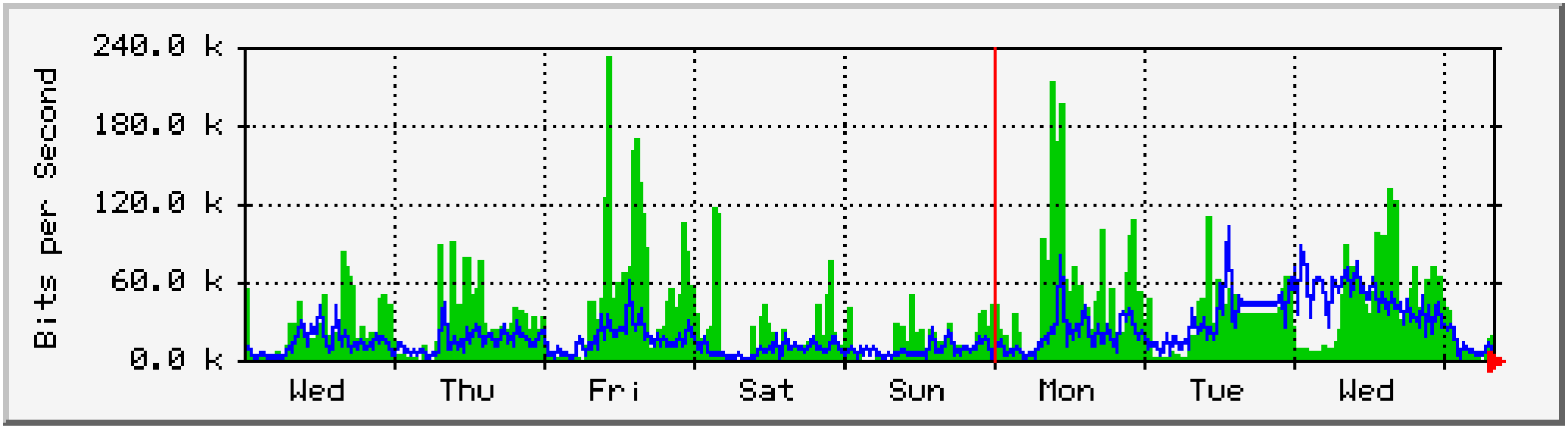
- **max-age=**[seconds] : Specifies the maximum amount of time that an representation will be considered fresh. Similar to Expires, this directive is relative to the time of the request, rather than absolute.
- **s-maxage=**[seconds] : Similar to max-age, except that it only applies to shared (e.g., proxy) caches.
- **public** : Marks authenticated responses as cacheable; normally, if HTTP authentication is required, responses are automatically uncacheable.
- **no-cache** : Forces caches to submit the request to the origin server for validation before releasing a cached copy, every time. This is useful to assure that authentication is respected (in combination with public), or to maintain rigid freshness.

Cache-Control Response Headers (2)

- **no-store** : Instructs caches not to keep a copy of the representation under any conditions.
- **must-revalidate** : Tells caches that they must obey any freshness information you give them about a representation. HTTP allows caches to serve stale representations under special conditions; by specifying this header, you're telling the cache that you want it to strictly follow your rules.
- **proxy-revalidate** : similar to must-revalidate, except that it only applies to proxy caches.

Tráfico en los servidores

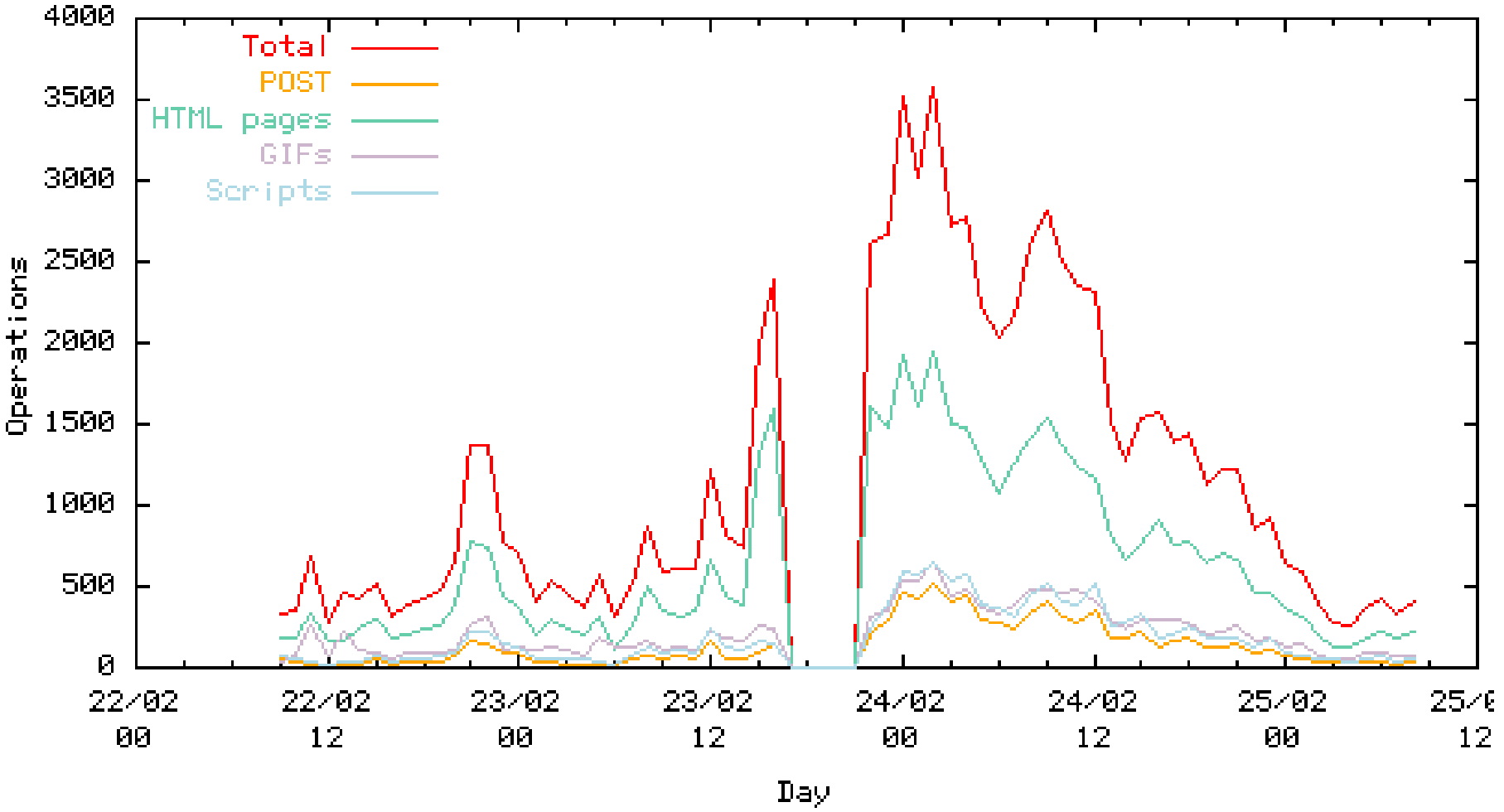
- La demanda que experimenta un servidor varía extremadamente (comportamiento fractal, “heavy tailed”, auto-similar, …)
  - Ocurre en sistemas complejos, gran población y con memoria
  - El valor medio puede ser poco probable …



Evolución del tráfico entrante y saliente en un sitio web típico durante una semana. Puede verse la gran variación horaria y la reducción de tráfico durante el fin de semana.

“Efecto Slashdot”

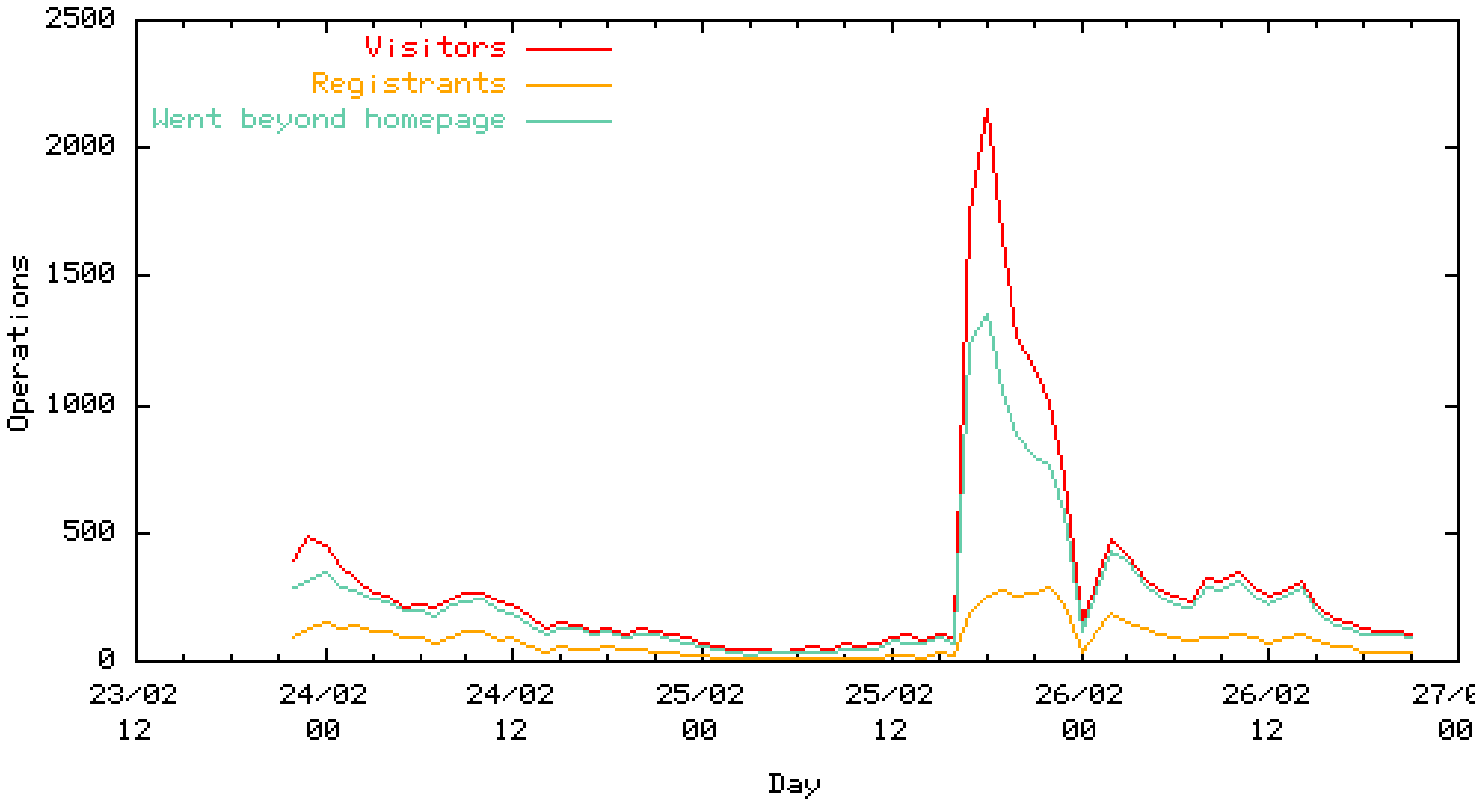
On February 23, 1999, around 15:43 European Time, the Linux Counter was listed on Slashdot, causing a breakdown of services.



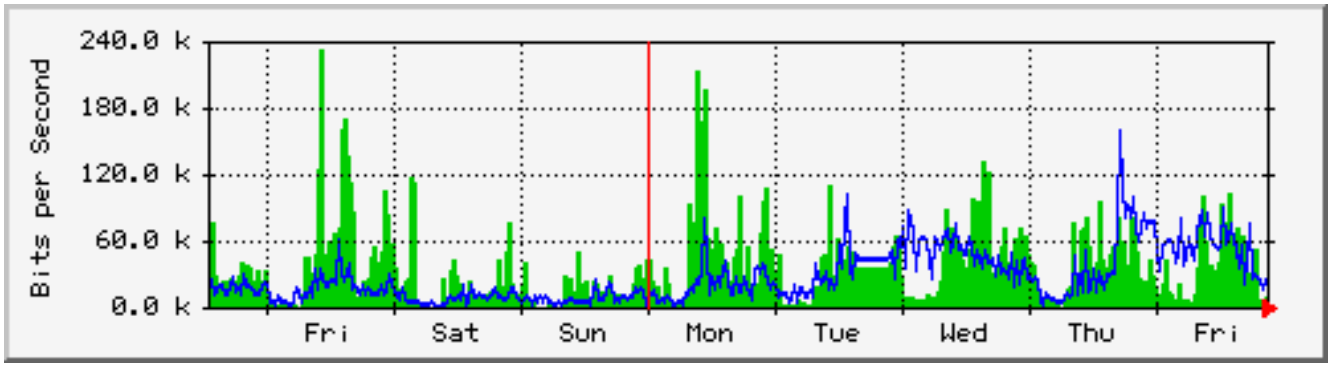
Efecto “slashdot” en http://counter.li.org.
Más información en: http://counter.li.org/slashdot/

“Efecto Slashdot” (II)

On Thursday, February 25, 1999, at 11:07 their time, they did it again.

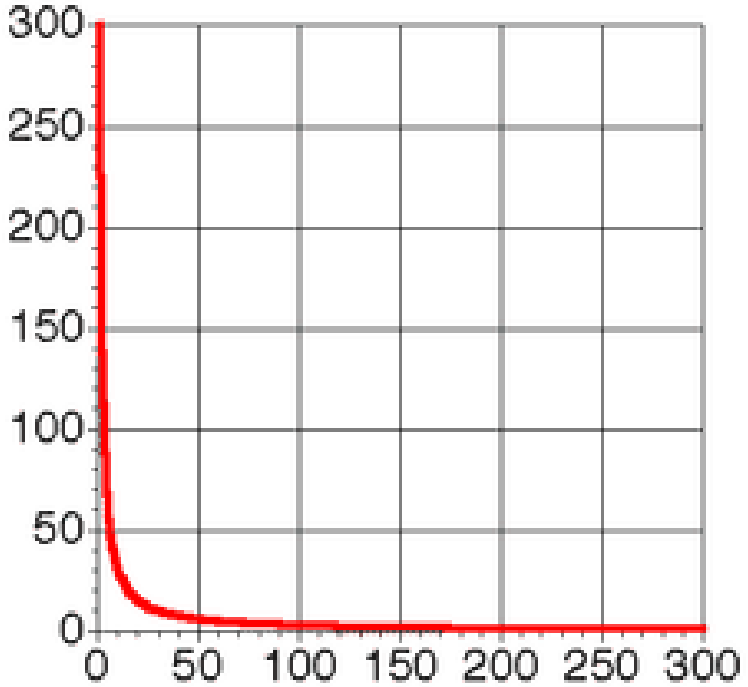


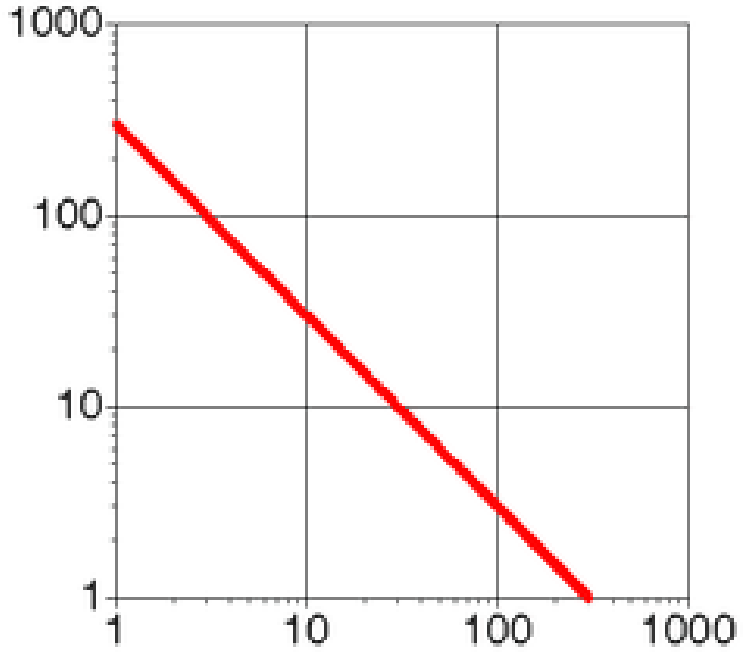
Una semana:
Slashdot I & II



La demanda sigue Ley de Zipf

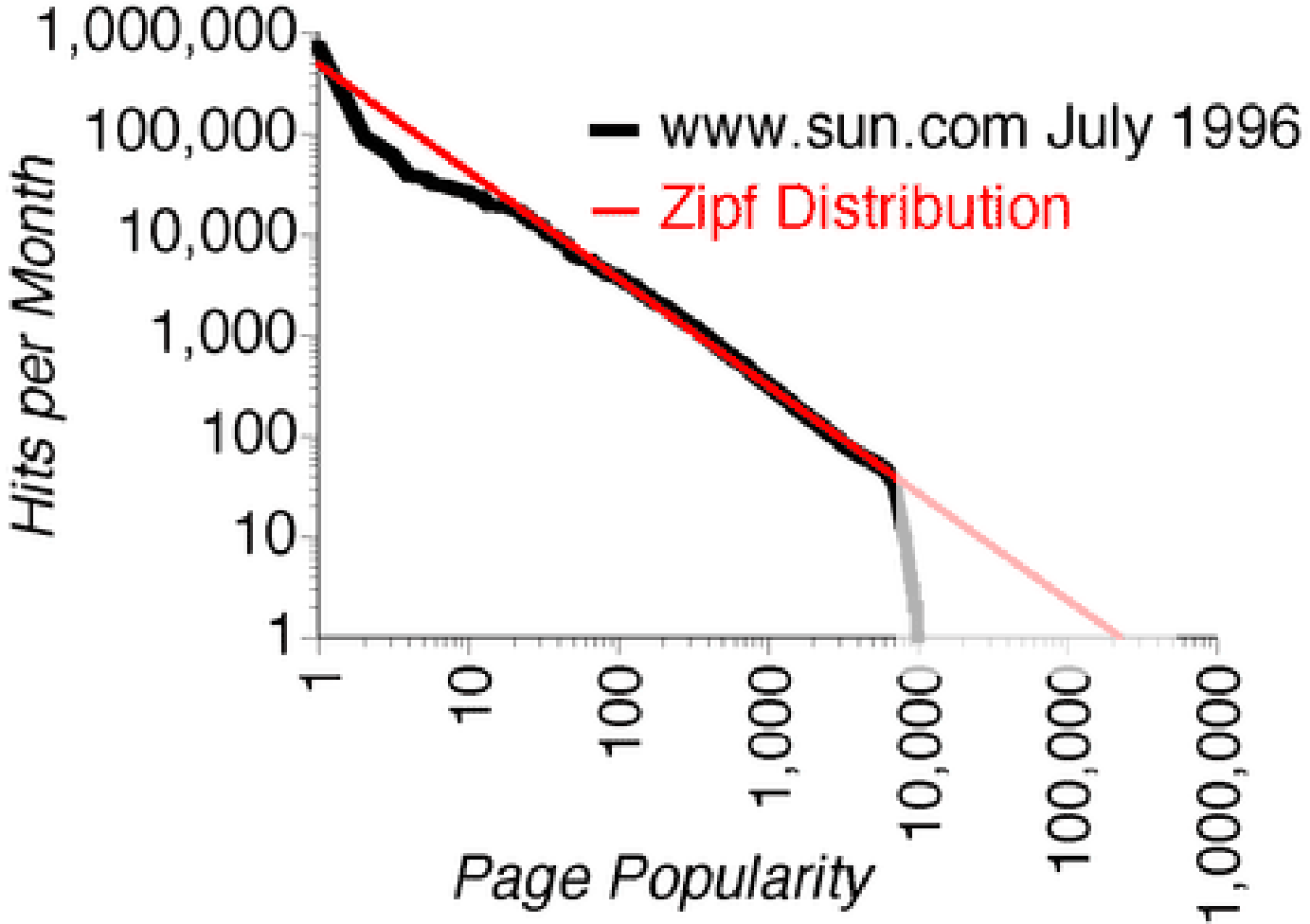
- George Kingsley Zipf (1902-1950)
  - La frecuencia de ocurrencia de cierto evento (P) como función del rango (i) cuando el rango viene determinado por la frecuencia de ocurrencia, es una función potencial Pi ~$1/i^a$, con el exponente a cercano a la unidad.
    - Frecuencia de palabras en Inglés. En 423 artículos de la revista TIME (245.412 palabras), "the" es la que más aparece: 15.861, "of" en segundo lugar: 7239 veces, "to" en tercer lugar: 6331 veces





Un caso …

- Número de visitas de las páginas de www.sun.com ordenadas por popularidad. Se ajusta bastante a una distribución de Zipf.



Perfil típico de demanda Web

- El tamaño medio de objeto=10-15 Kbytes, la mediana=2-4 Kbytes. Abundan **objetos pequeños** aunque se encuentra una cantidad no despreciable de objetos grandes (Mbytes).
- La mayoría de accesos al Web es para **objetos gráficos**, seguido de documentos html. El 1-10% son objetos dinámicos.
- Una página html tiene media **10 imágenes** y varios enlaces a otras.
- Un 40% de accesos para objetos considerados no cacheables.
- Popularidad de objetos web muy dispar: una pequeña fracción de objetos responsable de la mayoría de accesos, sigue la ley de Zipf
- El ritmo de acceso para objetos estáticos es mucho mayor que el ritmo de modificación.
- En escala de tiempo inferior al minuto, el tráfico web es a ráfagas: valores medios durante decenas de segundo muy poco fiables.
- Un 5-10% de accesos al Web se cancelan antes de finalizar.
- Casi todos los servidores usan el puerto 80

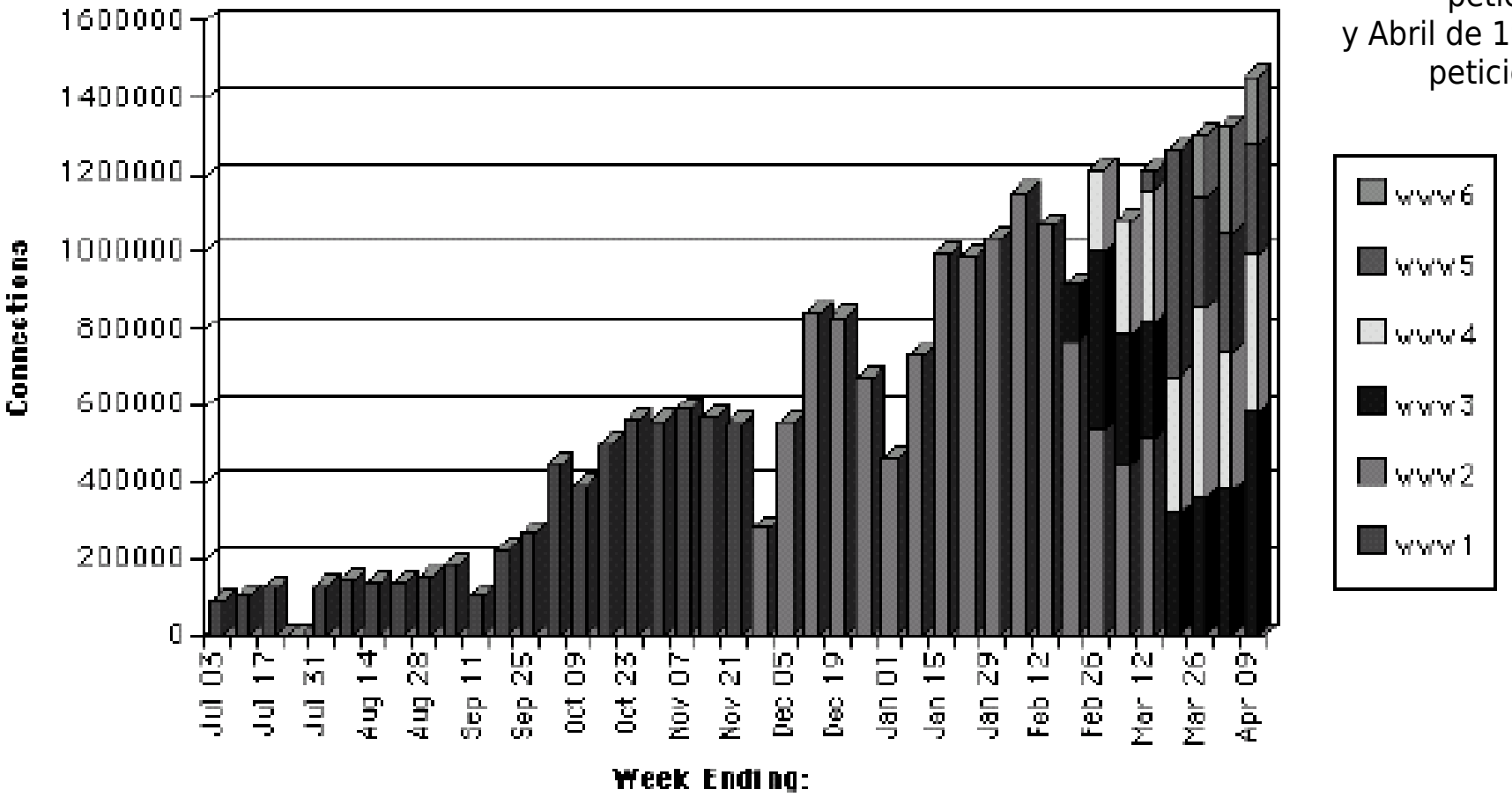
Generadores de carga

- Puede ser necesario probar la “capacidad” de nuestro servidor con “demanda sintética”.
  - **Apache JMeter**
    - Rendimiento servidor en documentos y recursos estáticos y dinámicos (archivos, Servlets, scripts Perl, objetos Java, consultas a bases de datos, servidores FTP Servers, etc).
      Simula diferentes tipos de carga extrema de la red, el servidor o un cierto objeto
      - http://jakarta.apache.org/jmeter
  - **Surge**
    - genera peticiones Web con características estadístiscas que simulan con mucha precisión la demanda típica de un servidor web
      - http://www.cs.bu.edu/faculty/crovella/links.html
  - **Microsoft Web Application Stress o WAS**
    - Prueba un sitio con IIS + ASP
      - http://webtool.rte.microsoft.com

Servidores replicados

- Cuando la carga aumenta puede aumentarse el número de servidores (misma ubicación: consistencia y reparto carga)
  - El primer web con una demanda importante fue http://www.ncsa.uiuc.edu. Tuvo que usar 4 servidores replicados para satisfacer la demanda.

Julio de 1993 (91.000 peticiones/semana) y Abril de 1994 (1.500.000 peticiones/semana).



Reparto de carga entre varios servidores

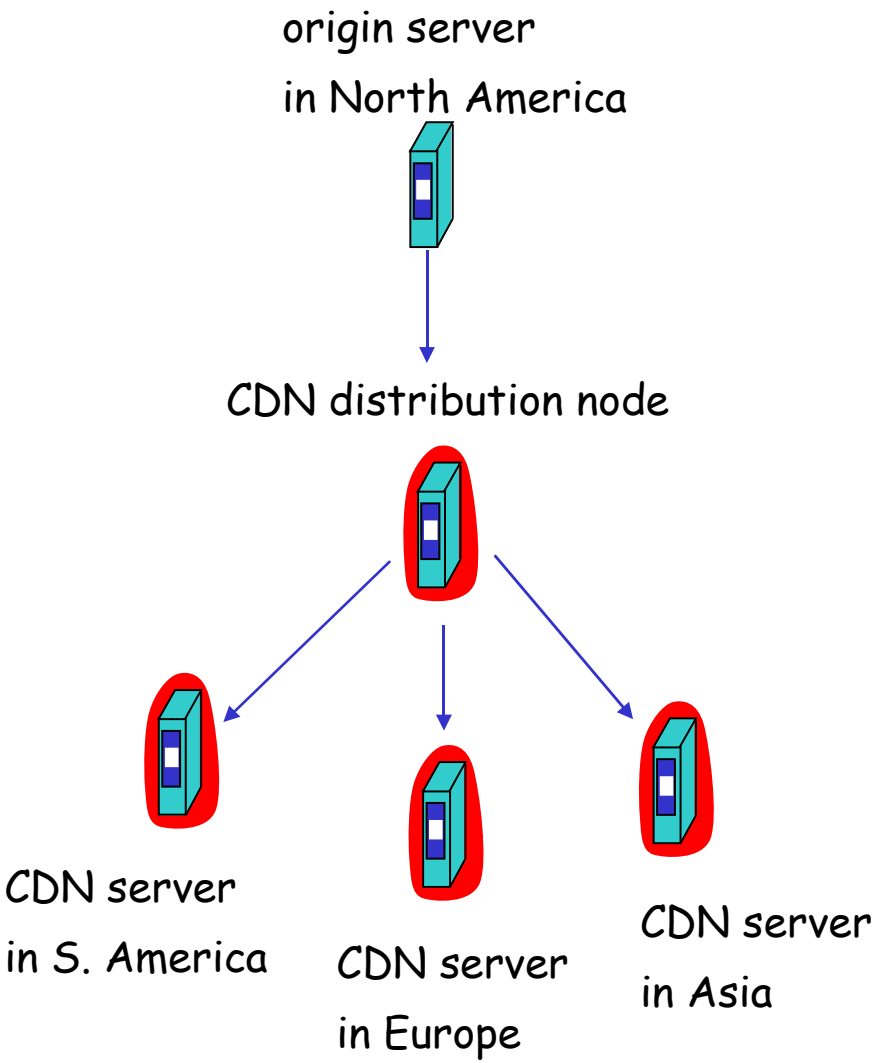
- Varios “trucos” para repartir peticiones entre varias máquinas:
  - “Mirrors”: un programa que redirige la petición http a la réplica mejor
  - DNS devuelva varias direcciones IP (el modo “round robin”). Los clientes pueden hacer peticiones http cada vez a una dirección IP distinta.
  - Redirección de transporte (“L4 Switch”): un router mira los paquetes IP de conexiones **TCP** hacia un servidor Web (puerto 80) y las redirige a la máquina interna menos cargada
  - Redirección a nivel de aplicación (“L7 Switch”): un router que mira las conexiones **http** y puede decidir a qué réplica contactar en función del URL solicitado. Muy complejo.
  - Mandar todas las peticiones a un **proxy inverso** que responda o con contenido guardado en la caché o pase la petición a uno o varios servidores internos.
- Si además las réplicas se sitúan cerca de los clientes, mejor rendimiento y más predecible. Inconveniente: difícil y caro montar servicio Web distribuido.

Content distribution networks (CDNs)

- The content providers are the CDN customers.

Content replication

- CDN company installs hundreds of CDN servers throughout Internet
  - in lower-tier ISPs, close to users
- CDN replicates its customers' content in CDN servers. When provider updates content, CDN updates servers



Solución: Redirección de la petición

- 2 técnicas para redirigir peticiones a servidores CDN:
  - **Redirección por DNS**

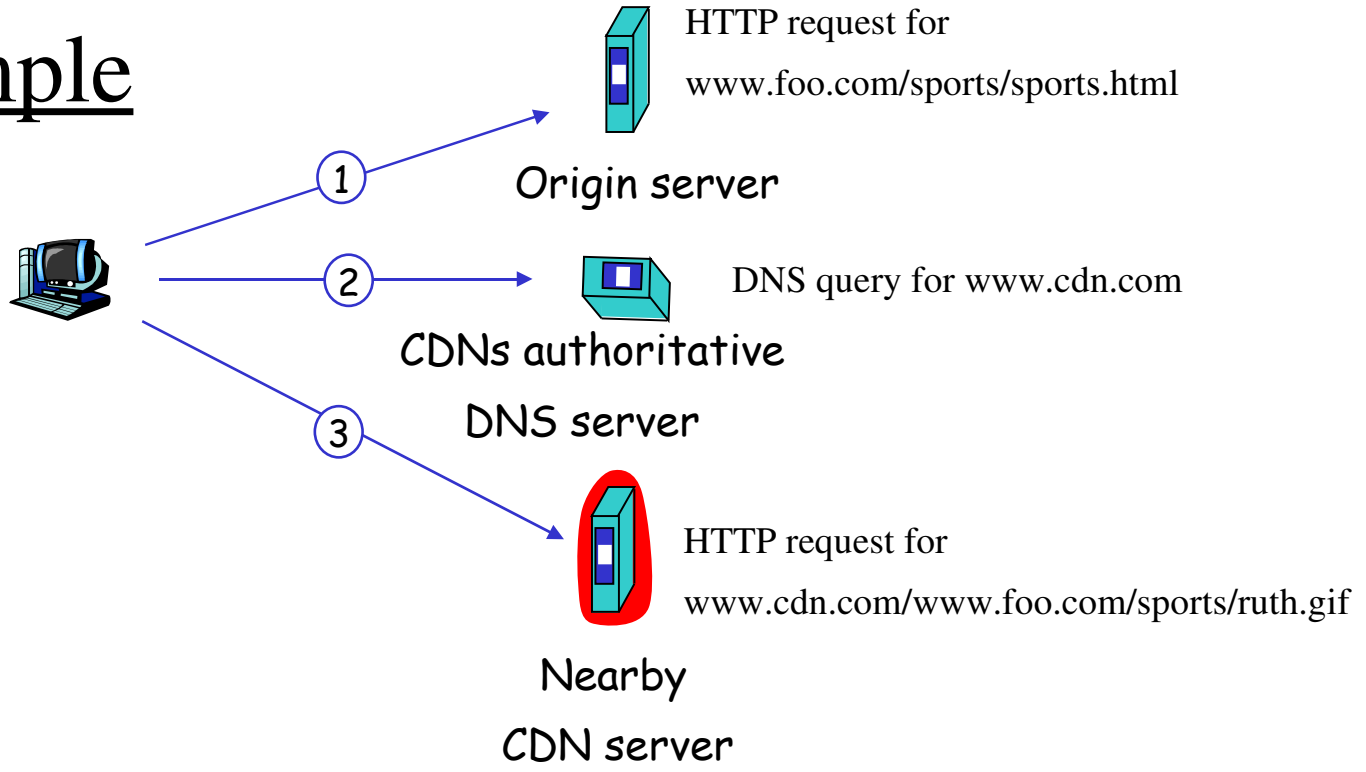
    Servidor DNS controlado por la infraestructura CDN. Distribuye las peticiones a servidores CDN según diferentes políticas (al menos cargado, al mas “próximo” al cliente (topología o geograficamente)

  - **Reescribir URL**

    Pagina principal viene de servidor origen, pero URL de objetos como gráficos está reescrita y apunta al servidor CDN.

(También se usan esquemas híbidos)

CDN example



origin server

- www.foo.com
- distributes HTML
- Replaces:

  http://www.foo.com/sports.ruth.gif

  with

  http://www.cdn.com/www.foo.com/sports/ruth.gif

CDN company

- cdn.com
- distributes gif files
- uses its authoritative DNS server to route redirect requests

Content distribution networks are *coordinated* caching systems ?

host

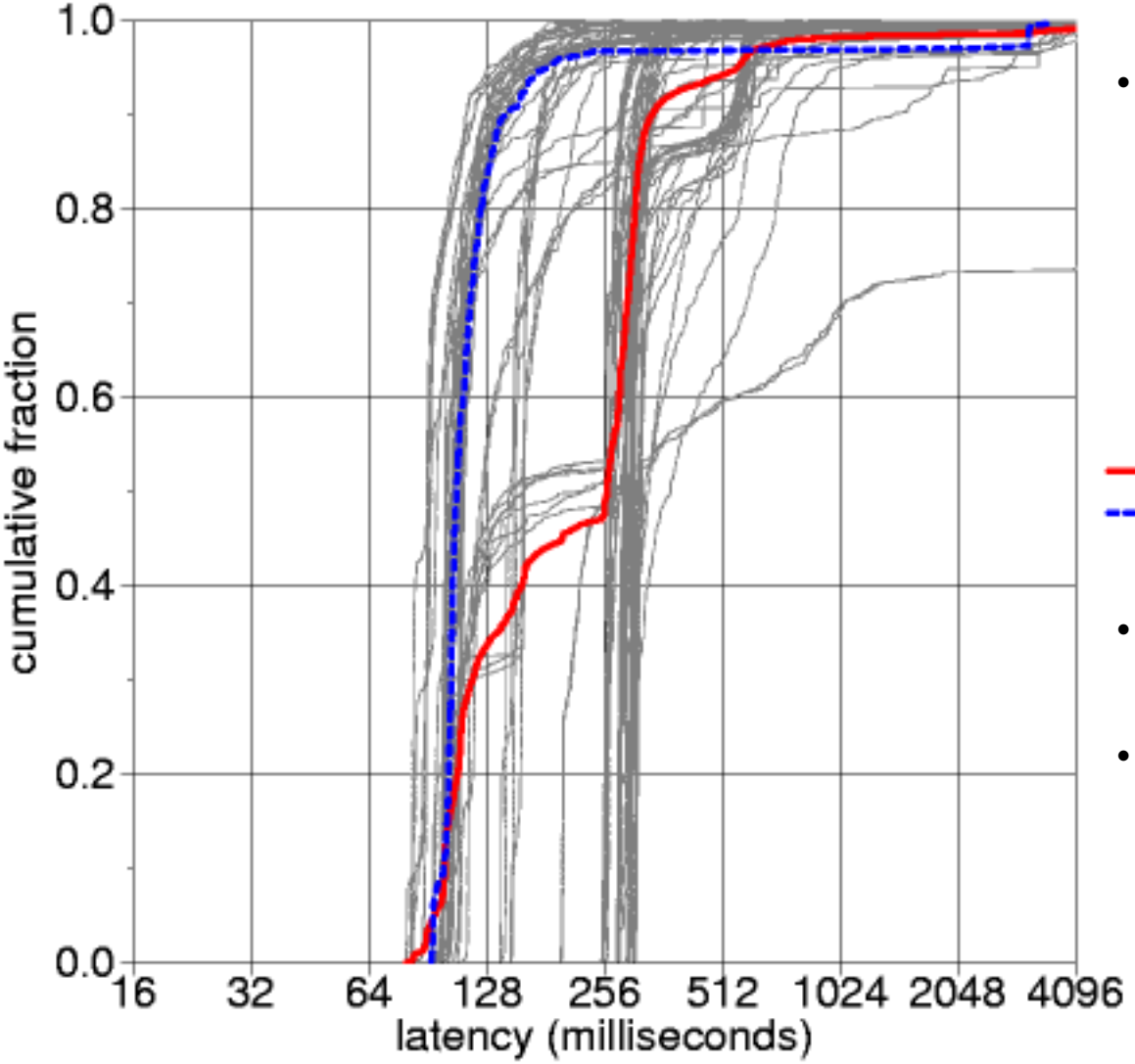
```
$ host www.microsoft.com
www.microsoft.com is an alias for toggle.www.ms.akadns.net.
toggle.www.ms.akadns.net is an alias for g.www.ms.akadns.net.
g.www.ms.akadns.net is an alias for lb1.www.ms.akadns.net.
lb1.www.ms.akadns.net has address 207.46.225.60
lb1.www.ms.akadns.net has address 207.46.18.30
lb1.www.ms.akadns.net has address 207.46.19.60
lb1.www.ms.akadns.net has address 207.46.19.190
lb1.www.ms.akadns.net has address 207.46.198.30
lb1.www.ms.akadns.net has address 207.46.198.60
lb1.www.ms.akadns.net has address 207.46.199.30
lb1.www.ms.akadns.net has address 207.46.199.60
$
```

Ejemplo: Akamai

- Más de **13.000** puntos de servicio en todo el mundo
- Contenido "Akamaizado":
  - el servidor de la empresa sirve html y devuelve los enlaces a contenidos incluídos apuntando al servidor más próximo
    - Algoritmo 1: [**direccionamiento geográfico**] El ARL se calcula según la región del demandante, se le envía al servidor de nombres de la "zona"
    - Algoritmo 2: [**reparto de carga en una ubicación**] El ARL contiene un índice hash que permite repartir la carga (via DNS/switch nivel 4) entre varios servidores ubicados en un mismo lugar
  - El usuario ve un URL normal (el de la página html)
    - En la ventana de estado sí se ven los ARL
  - El servidor de la empresa tiene "Logs" con visitas a html, pero la mayoría del tráfico lo sirve Akamai desde la proximidad del cliente.
  - Akamai **mantiene info de estado de la red**, de los clusters, de los "surrogate".
  - No siempre elige el "mejor posible" pero de los mejores, sí evita los peores.

Resultados

Akamai, location A



- No elige el *mejor* servidor CDN

- En >90% de los casos elección buena del servidor respecto a ubicación del cliente.
- En 10% elección aleatorio del servidor hubiera sido mejor. .

http://www.terena.nl/conf/wcw/Proceedings/S4/S4-1.pdf

Tipo de contenido servido por CDNs

- Peticiones HTTP servidas por CDNs*
  - Imágenes representan 96-98% del contenido o 40-60% de los bytes servidos por CDNs
  - De los CDNs estudiados Akamai sirve 85-98% de los objetos (bytes)
  - Tasas de hit en caches de imagenes servidas por CDNs son 20-30% mas altas que imagenes no servidas por CDNs

* Fuente: Y. Zhang et al. “On the Use and Performance of Content Distribution Networks, 2001.