Unit 1:
Content Distribution Networks (CDNs)

[Modified from slides by Lucile Sassatelli]

Outline

1. Content Distribution Networks (CDNs)
   a. Motivation
      a. Current internet content
      b. Several servers. Why?
      c. Web Caching vs CDNs
   b. How a CDN works
      a. CDN Basics
      b. Server selection
      c. Content storage (caching)
      d. Overlay Routing
   c. Akamai Example

Motivation

1. **Current and Future Internet Content**
2. One Server vs Several Servers. Why ?
3. Web Caching
4. Content Distribution Networks (CDNs)

Current and Future Internet Content

- Following info is extracted from [*Cisco Visual Networking Index: Forecast and Methodology, 2018–2023] report* *[updated March 9, 2020]*
  - Annual report [updated] on Internet evolution in terms of current Internet traffic and forecasts
- Cisco is one the main telco equipment vendors
- Idea is to get an understanding of the role (and future) of video delivery in the Internet

Current and Future Internet Content

- **Executive summary**
  - **Internet users: ca. 2/3 global population will have Internet access by 2023 (from 1/2 at 2018).**
    - > 70 % global population will have mobile connectivity by 2023 (from 66% at 2018)
  - **Fixed broadband speeds will more than double by 2023**.
    - By 2023, global fixed broadband speeds will reach 110.4 Mbps, up from 45.9 Mbps in 2018.
  - **Mobile (cellular) speeds will more than triple by 2023**.
    - The average mobile network connection speed was 13.2 Mbps in 2018 and will be 43.9 Mbps by 2023.

Current and Future Internet Content

- **Video highlights**
  - **IoT connections will be the fastest-growing device type**
    - growing from 33% to 50% of all devices types
  - **Conversely, video devices (tablets, smartphones, TVs) will slightly decrease in mix of devices**
    - decreasing from 43% to 37% of all devices types
  - **But, globally, IP video traffic is > 80% of all consumer Internet traffic now. Why?**
  - **Because video devices have a multiplier effect on traffic,** namely, Ultra-High-Definition (UHD), or 4K, TVs
    - 4K bit rate (18 Mbps) is > 2x HD bit rate , > 9x times SD bit rate.
    - By 2023, 66 % of the installed flat-panel TV sets will be UHD
    - **An Internet-enabled HD television (2-3 h of daily usage) would generate as much traffic as an entire household today, on an average.**
  - **Then, significant demand for bandwidth and video in the connected home of the future will continue.**

Role of video streaming in the Internet

- Overall, it is dominant and growing fast in terms of demanded bandwdith
- The importance is huge to ISPs (both fixed and mobile)
- Lots of money involved in the business
  - Video and CDN providers

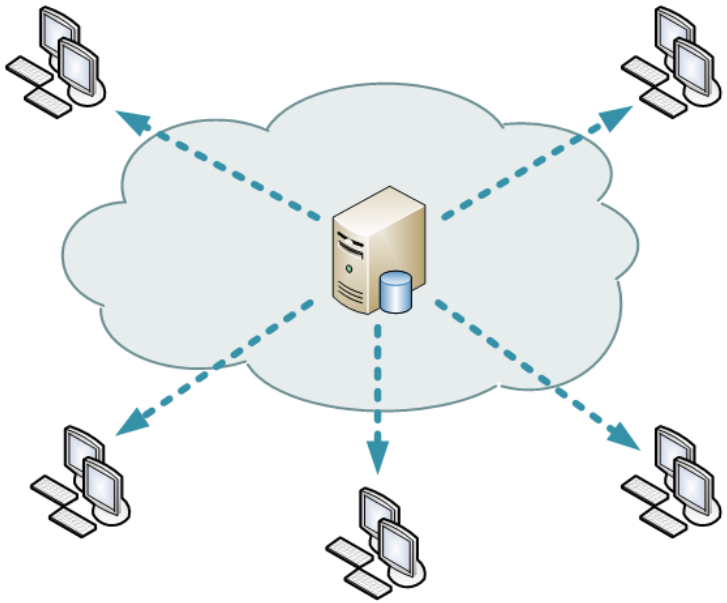
Motivation

1. Current Internet Content
2. **One Server vs Several Servers. Why ?**
3. Web Caching
4. Content Distribution Networks (CDNs)

Content Distribution Networks: Why?

- **In the beginning :SINGLE server**
  - probably located far from user
  - probably served blinking text
  - not necessarily devoted to serve content

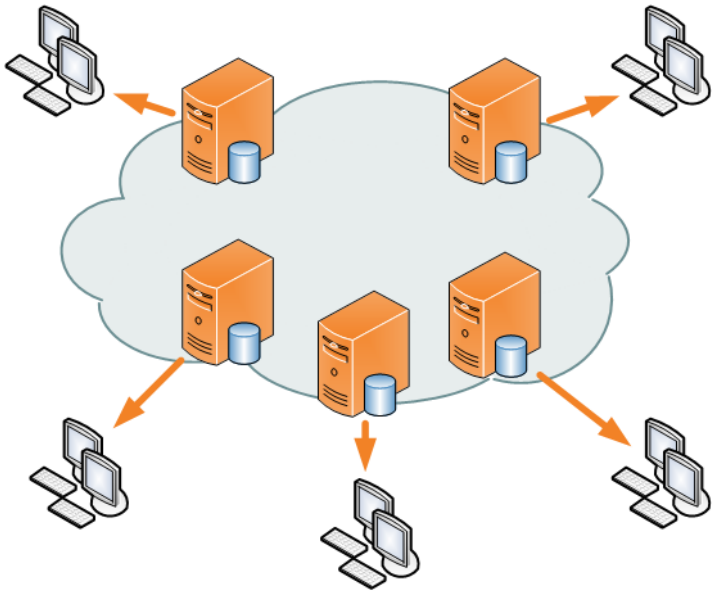
- Issues with this model
  - Bad users' experience (delay) :
    - Far from most clients
  - No reliability: single point of failure
    - Unplugging cable, hardware failure, natural disaster, hacking attacks (denial of service attack)
  - No scalability
    - Easily overloaded
    - Flash crowds



Wikipedia: https://commons.wikimedia.org/wiki/File:NCDN_-_CDN.png

Content Distribution Networks: Why?

- **Later, MULTIPLE servers :**
  - Content and applications are served from locations near to end users
  - devoted to serve content
- Better users' experience
  - Move content closer to end clients
  - Improve client perceived-latency
- Better scalability
  - Global capacity on demand
  - Offload traffic from content providers
- Cost effective
  - Reduce operational costs for service providers
  - No over-provisioning
  - No redundant datacenters
  - Simple to manage
- Better reliability
  - No single point of failure
  - Automatic failover
- Better Security
  - Traffic harder to steal
  - Defense in depth protects central infrastructure



Wikipedia: https://commons.wikimedia.org/wiki/File:NCDN_-_CDN.png

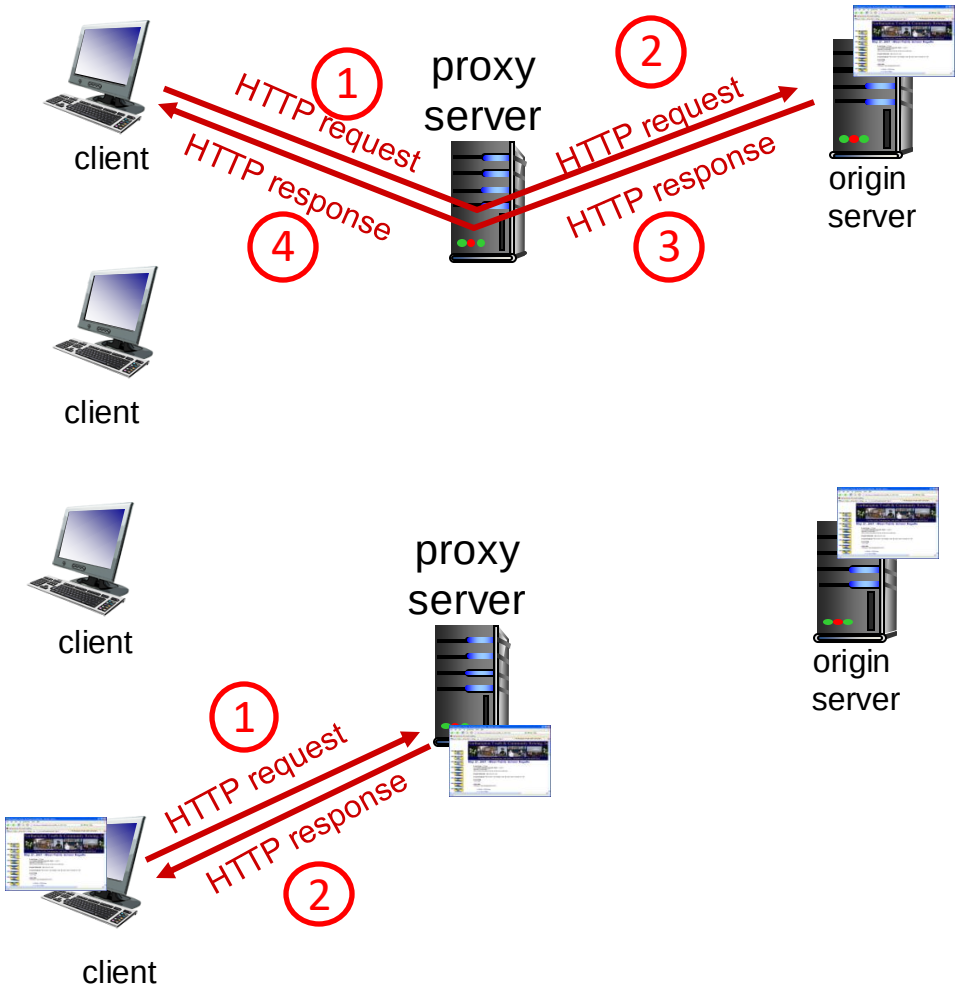
Motivation

1. Current Internet Content
2. One Server vs Several Servers. Why ?
3. **Web Caching**
4. Content Distribution Networks (CDNs)

A 1st step to CDN: Web proxy caching

Web Proxy Caches

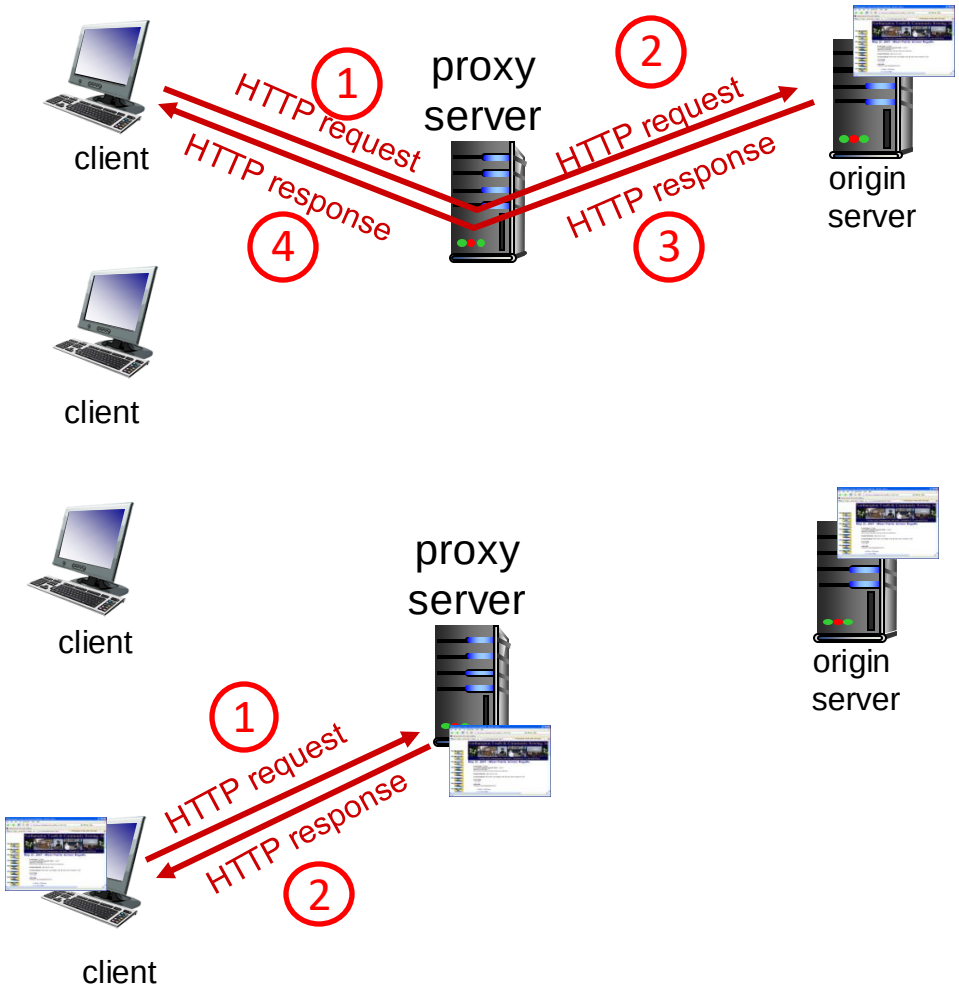
- ***They are not managed by CDNs***
- satisfy client request without involving origin server
- user sets browser: Web accesses via cache
- browser sends all HTTP requests to cache
  - object in cache: cache returns object
  - else cache requests object from origin server, then returns object to client
- 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)***



A 1st step to CDN: Web proxy caching

Web Proxy Caches

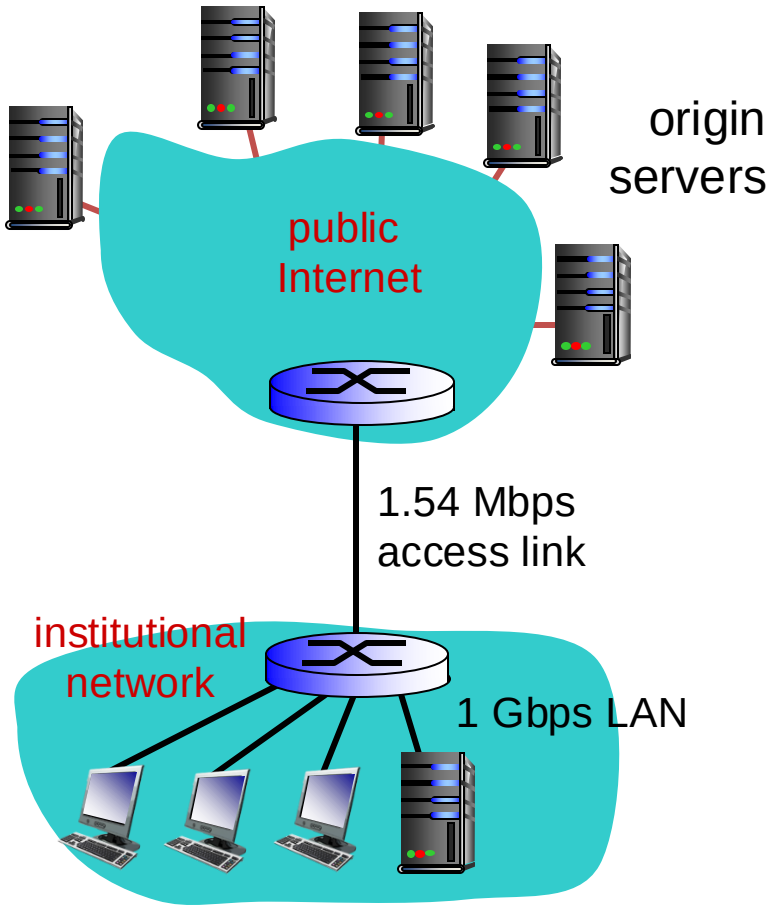
- ***They are not managed by CDNs***
- satisfy client request without involving origin server
- user sets browser: Web accesses via cache
- browser sends all HTTP requests to cache
  - object in cache: cache returns object
  - else cache requests object from origin server, then returns object to client
- 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)***



A 1st step to CDN: Web proxy caching

Example

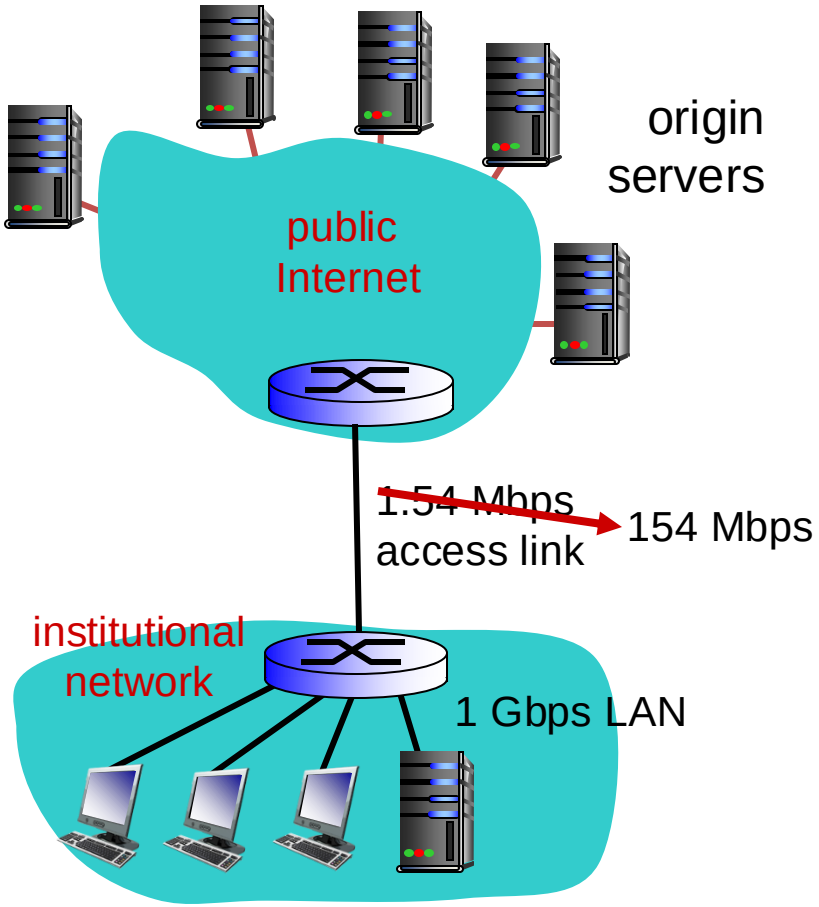
- *assumptions*:
  - avg object size: 100K bits
  - avg request rate from browsers to origin servers: 15/sec
  - avg data rate to browsers: 1.50 Mbps
  - RTT from institutional router to any origin server: 2 sec
  - access link rate: 1.54 Mbps
- *consequences*:
  - LAN utilization: 0.15%
  - access link utilization = 99%
  - total delay = Internet delay + access delay + LAN delay = 2 sec + minutes + µsecs



A 1st step to CDN: Web proxy caching

Example (increase access)

- *assumptions*:
  - avg object size: 100K bits
  - avg request rate from browsers to origin servers:15/sec
  - avg data rate to browsers: 1.50 Mbps
  - RTT from institutional router to any origin server: 2 sec
  - access link rate: ~~1.54 Mbps~~ → 154 Mbps
- *consequences*:
  - LAN utilization: 0.15%
  - access link utilization = ~~99%~~ → 0.99%
  - total delay = Internet delay + access delay + LAN delay = 2 sec + ~~minutes~~ + usecs → msecs



Cost: increased access link speed (not cheap!)

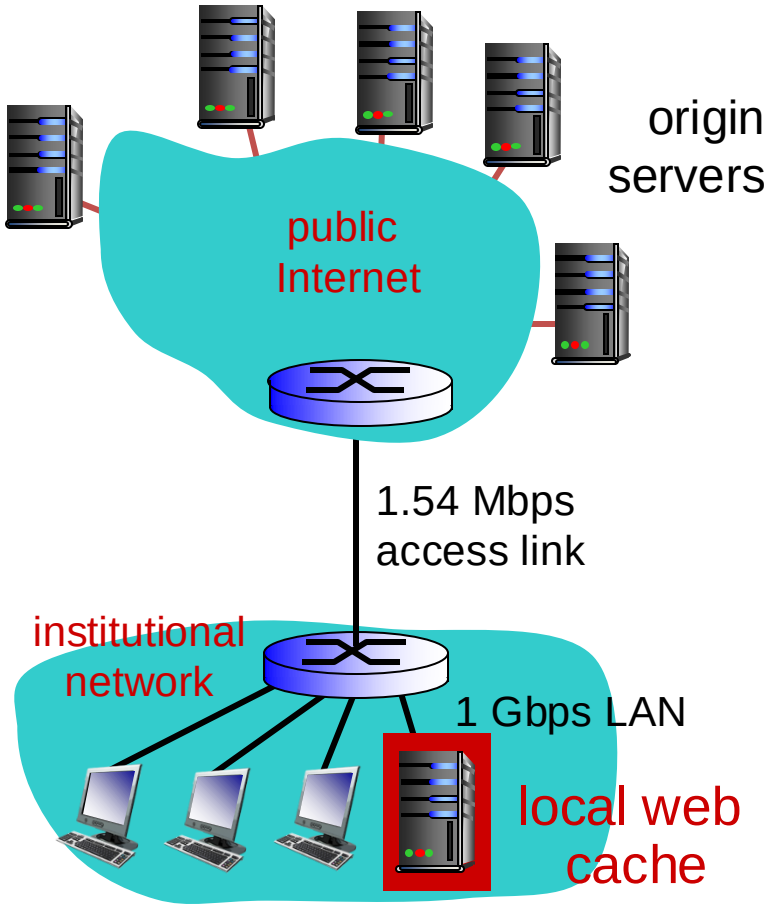
A 1st step to CDN: Web proxy caching

Example (install local cache)

- *assumptions*:
  - avg object size: 100K bits
  - avg request rate from browsers to origin servers:15/sec
  - avg data rate to browsers: 1.50 Mbps
  - RTT from institutional router to any origin server: 2 sec
  - access link rate: 1.54 Mbps
- *consequences*:
  - LAN utilization: 0.15%
  - access link utilization = ?
  - total delay = ?

How to compute link utilization, delay?

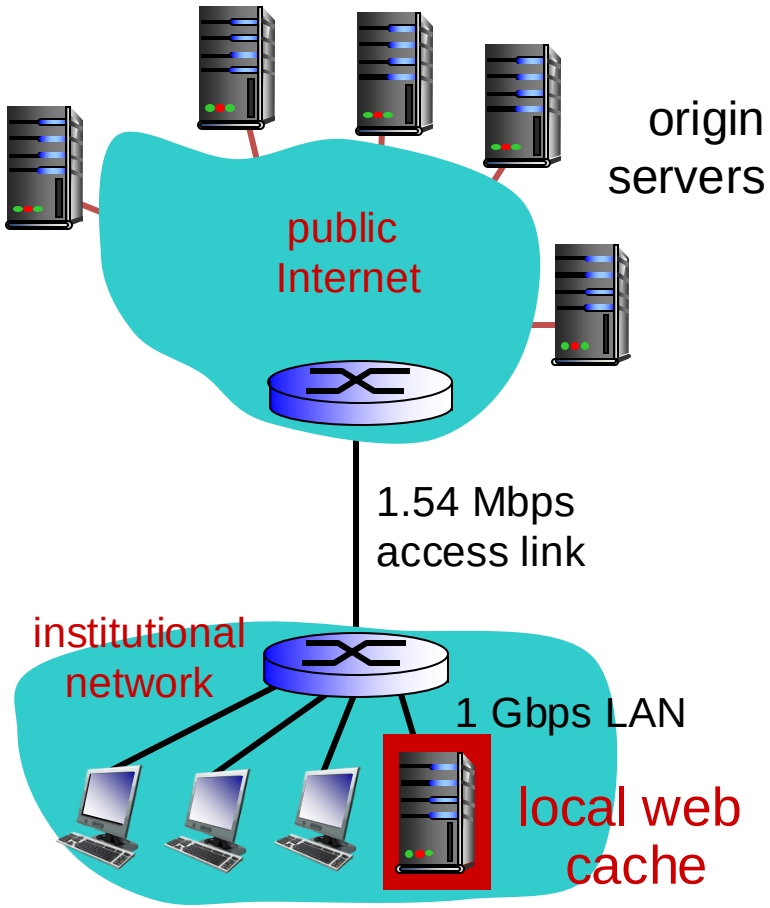
Cost: web cache (cheap!)



A 1st step to CDN: Web proxy caching

Calculating access link utilization, delay with cache:

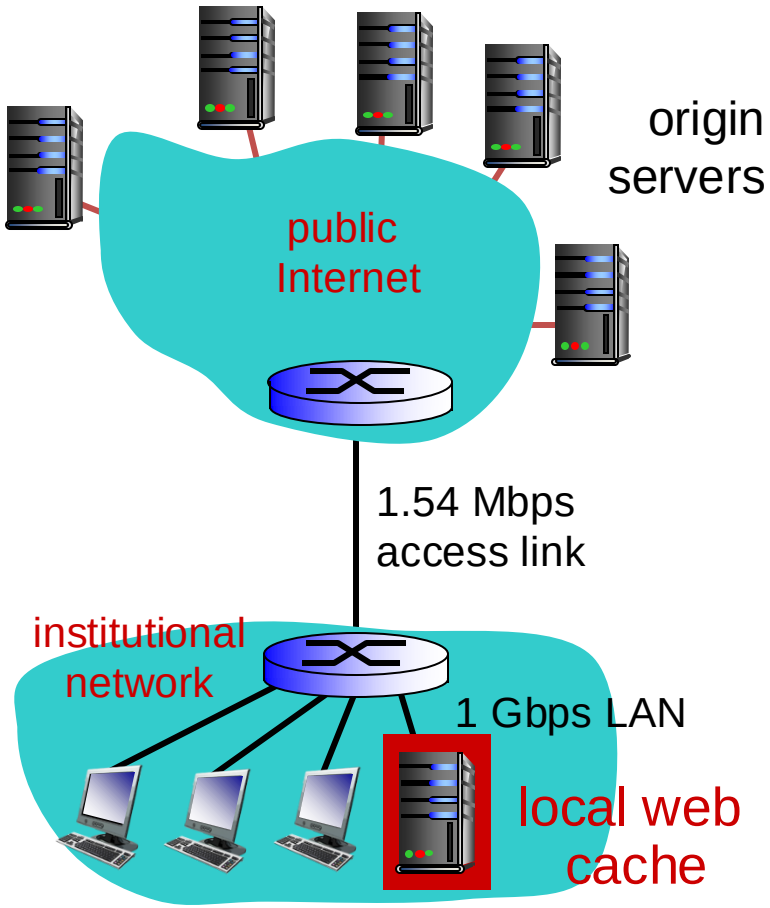
- suppose cache hit rate is 0.4
  - 40% requests satisfied at cache, 60% requests satisfied at origin
- access link utilization:
  - 60% of requests use access link
- data rate to browsers over access link = 0.6*1.50 Mbps = .9 Mbps
  - utilization = 0.9/1.54 = .58
- total delay
  - = 0.6 * (delay from origin servers) +0.4 * (delay when satisfied at cache)
  - = 0.6 (2.01) + 0.4 (~msecs)
  - = ~ 1.2 secs
  - less than with 154 Mbps link (and cheaper too!)



A 1st step to CDN: Web proxy caching

To Wrap about Web Caching

- ***Reactively*** replicates popular content
- Reduces origin server costs
- Reduces client ISP costs
- Smaller round-trip times to clients

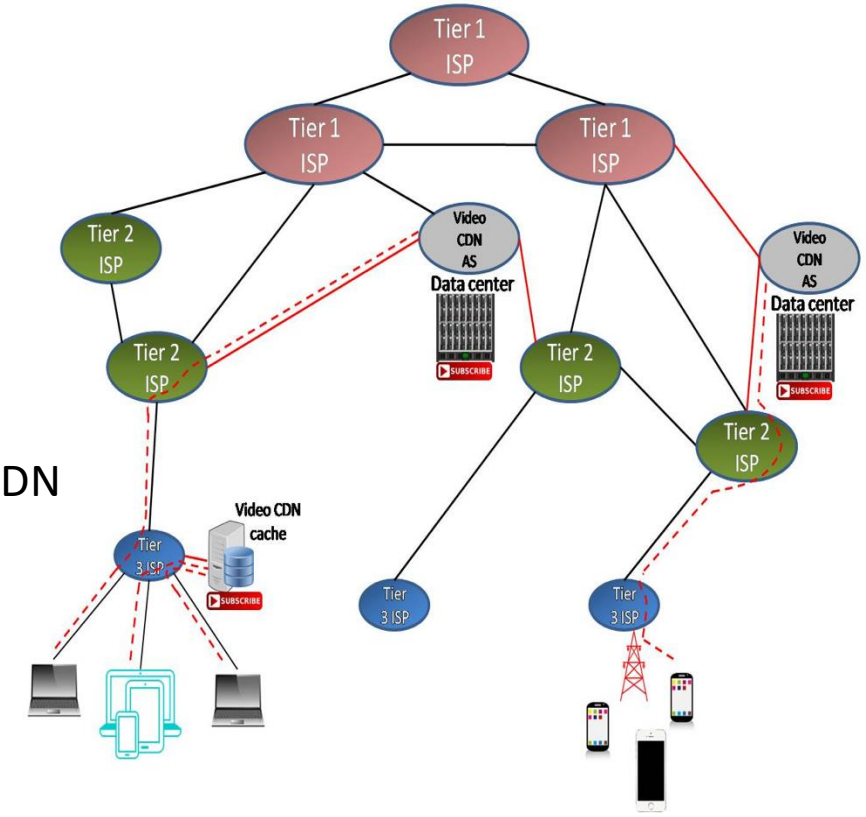


Motivation

1. Current Internet Content
2. One Server vs Several Servers. Why ?
3. Web Caching
4. **Content Distribution Networks (CDNs)**

Content Distribution Networks (CDNs)

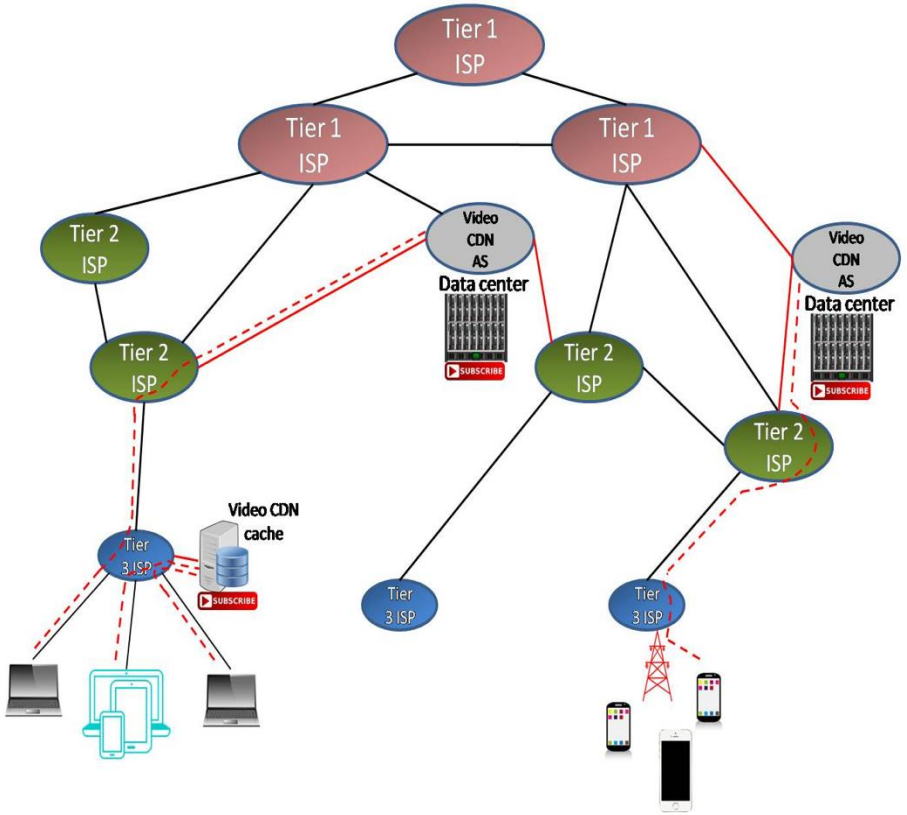
- Same principle as web caching
  - *content closer to end clients*
  - BUT ***proactive*** content replication
- They ”professionalize” the content replication
- Content replication
  - Content provider (CP) contracts with a CDN
  - CDN installs many servers throughout Internet (in large datacenter or close to users)
  - CDN replicates customers’ (CP) content
  - When provider updates content, CDN updates servers
- Updating the replicas
  - Updates pushed to replicas when the content changes



L. Sassatelli and M.-J. Montpetit. “Technologies and Architectures for Future IP Television Services”. In: A Comprehensive Guide to IPTV Delivery Networks. Ed. by A. K. Pathan, S. Fati, S. Azad. Wiley, Nov. 2016. HAL: https://hal.archives-ouvertes.fr/hal-01351930.

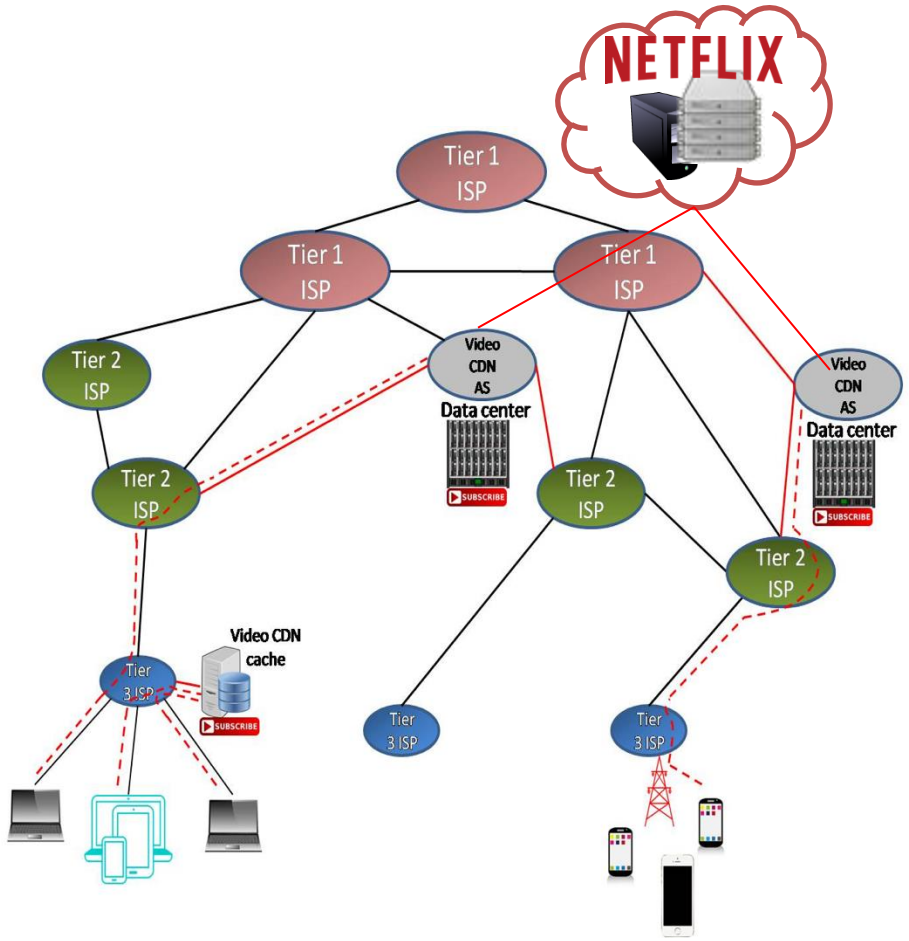
What is a CDN?

- Content Distribution Network
  - Also sometimes called Content Delivery Network
  - Most the world's bits are delivered by a CDN
- Primary Goals
  - Create replicas of content throughout the Internet
  - Ensure that replicas are always available
  - Directly clients to replicas that will give good performance



Stakeholders in content value chain

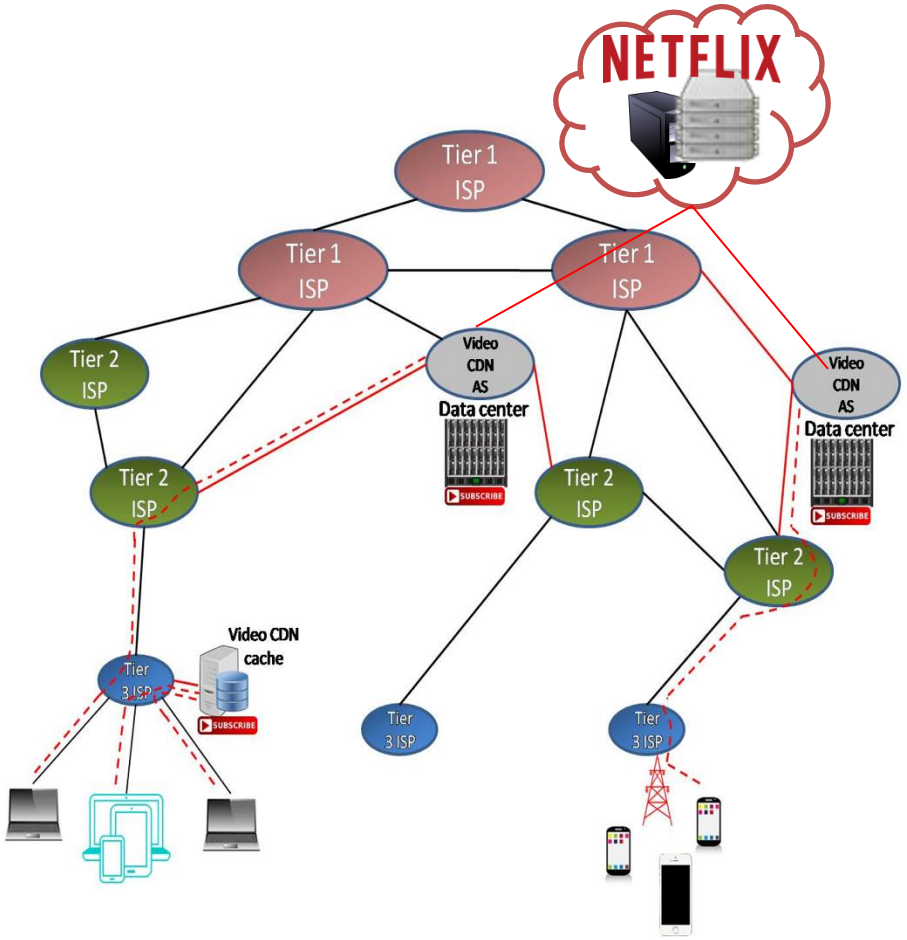
- Content Provider (CP) :
  - Create content
  - Origin servers
  - Ex. YouTube, Netflix, Apple TV
- Content Distribution Network (CDN) :
  - Replicate content
  - Distributed cache servers often at ISPs
  - Ex.: Akamai, Limelight, Level 3
- Internet Service Provider Network (ISP) :
  - Transport content
  - Several types (Tier 1, Tier 2, Tier 3)
  - High-speed network connecting CDN servers
  - Ex.: AT&T, Vodafone, Orange
- End users
  - Consume content
  - Can be located anywhere in the world



CP → CDN → ISP → Users

Stakeholders in content value chain

- Some times a stakeholder has several roles:
  - CP+CDN : Youtube (Google), Netflix, Comcast and Facebook
  - CDN+ISP: Orange, AT&T, Level3
- Tendency to Vertical Integration:
  - One stakeholder has all the value chain from content creation to content consumption :
    - In France, Orange



CP → CDN → ISP → Users

How a CDN works

1. **CDN Basics**
2. Server selection
3. Content storage (caching)
4. Overlay Routing

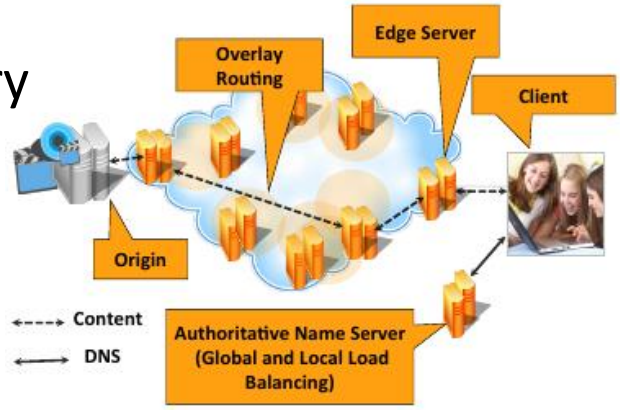
CDN basics

- Top-3 objectives:
  - high reliability,
  - fast and consistent performance,
  - low operating cost.

- We present some of the main steps from the instant that a browser or other application makes a request for content until that content is delivered.

1st step: DNS query

- Request for content => starts with a DNS query
- Query forwarded to the CDN's authoritative name server.
- It decides which of the CDN's clusters to serve the content from: decision with a variant of the ***stable marriage***



- Then a second set of name servers decide which particular web server or servers within the cluster will serve the content.
  - Within the cluster, load is managed using a **consistent hashing** algo
  - the client's application can issue the request to the web server
- The web servers that serve content to clients are called edge servers
- Akamai's CDN currently has over 170,000 edge servers located in over 1300 networks in 102 countries and serves 15-30% of all Web traffic.

2nd step: HTTP request

- When an edge server receives an HTTP request, it checks to see if the requested object is already present in the server's cache. If not, the server begins to query other servers in order:
  - other servers in the same cluster, who share a LAN.
  - If none of these servers have the object, it may ask a "parent" cluster.
  - If all else fails, the server may issue a request for the object to an origin server operated by the content provider, who is a customer of the CDN.
- As new objects are brought into the CDN's edge server, it may be necessary to evict older items from the cache: ***Bloom filters*** are used to decide which objects to cache and which not to
- Using the CDN servers as relays in an "***overlay routing network***". The idea is that exploring multiple overlay paths between the origin and the edge servers and picking the best ones can provide paths with lower latency and higher throughput.

CDN basics

- **Key Goal**: Send clients to server with best end-to-end performance
  - The best server can change over time
  - And this depends on client location, network conditions, server load, …
- Performance depends on
  - Server load
    - How to find replicated content
    - How to choose among known replicas
    - How to redirect client requests to closest/best replica?
  - Content at that server
    - How to replicate content
    - Where to replicate/cache content
    - Which content to cache?
  - Network conditions
    - How to route content to users

How a CDN works

1. CDN Basics
2. **Server selection**
3. Content storage (caching)
4. Overlay Routing

Optimizing Server Load: Server selection

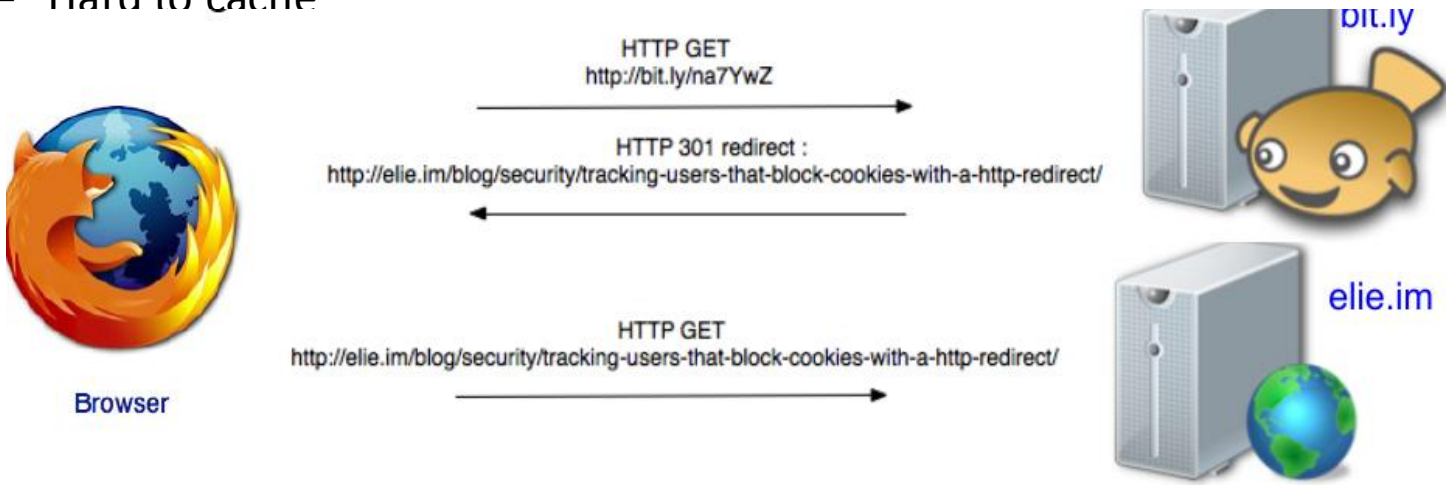
- Server Selection: main tool to optimize server load
  - Basically, mapping clients to servers

- Which server to assign to a client ?
  - Lowest load: to balance load on servers
  - Best performance: to improve client performance
    - Based on Geography (nearest one)? RTT? Throughput? Load?
  - Any alive node: to provide fault tolerance, availability
  - Cheapest bandwidth, electricity, …

- How to direct clients to a particular server (rather than a *group*)?
  - As part of *application*: HTTP redirects
  - As part of *routing*: IP anycast routing
  - As part of *naming*: DNS selection

Application based (HTTP redirects)

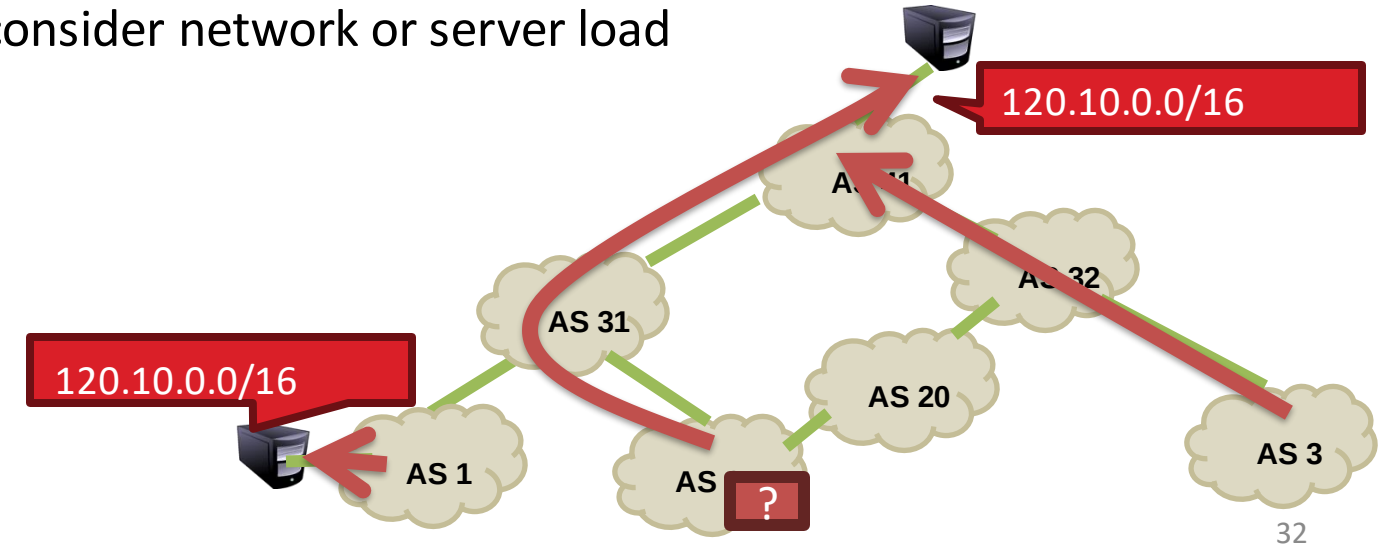
- Server replies with “HTTP redirect” to guide client to another server
  - Name given in the answer
- Advantages
  - Application-level, fine-grain control
  - Selection based on client IP address
- Disadvantages
  - Extra round-trips for TCP connection to server, additional load
  - Overhead on the server,
  - Hard to cache



Picture source : http://www.elie.net/

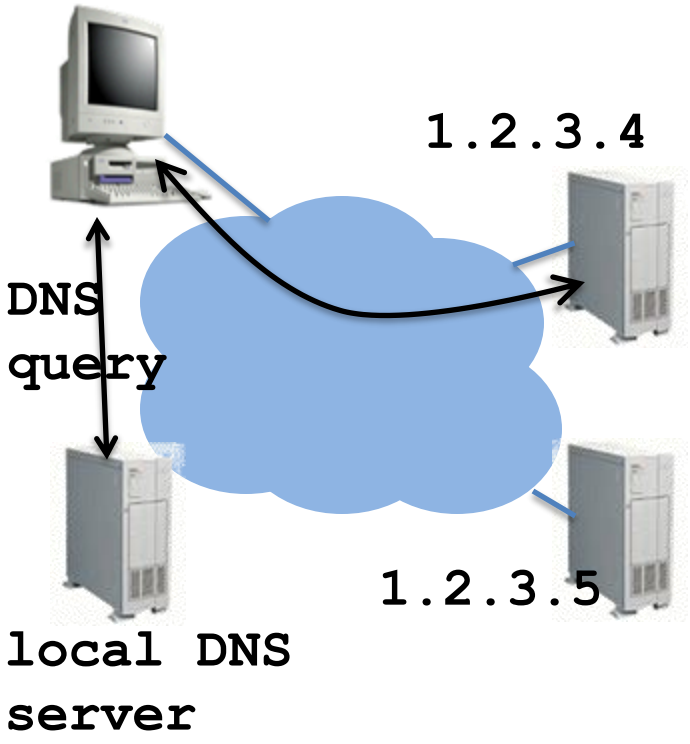
Routing based (IP anycast)

- An IP address in a prefix announced from multiple locations
- Advantages
  - No extra round trips, route to nearby server
  - Transparent to clients, works when browsers cache failed addresses, circumvents many routing issues
- Disadvantages
  - Little control: Different packets may go to different servers
  - Scalability issues: only simple request-response apps
  - Does not consider network or server load



Naming based (DNS selection)

- DNS name to address mapping under control of the CP
  - CP's DNS servers provide authoritative answers to DNS requests
- DNS mapping often anycast -> one name maps to one of several addresses
  - Suitable server can be chosen for a given client at a given time instance
- Advantages
  - Avoid TCP set-up delay (reduce TTS)
  - DNS caching reduces overhead
    - Uses existing, scalable DNS infrastructure
  - Relatively fine control
  - Well-suitable for caching
- Disadvantage
  - Based on IP address of local DNS server
    - Assumes that client and DNS server are close.
  - "Hidden load" effect of resolver's population
  - DNS TTL limits adaptation (small TTLs ignored)
  - Request by resolver not client
  - Request for domain not URL



Choosing the Cluster

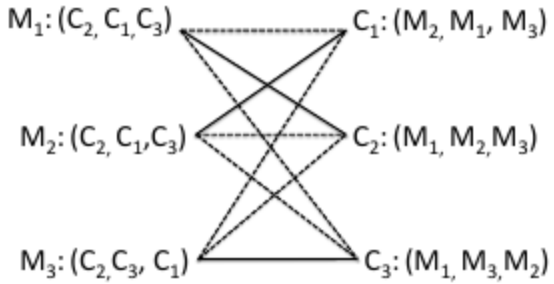
- Global load balancing is the process of mapping clients to the server clusters of the CDN.
- Rather than making load balancing decisions individually for the billions of clients, cluster assignments at the granularity of map units.
- A map unit: < IP address prefix, traffic class >
  - Tens of traffic classes such as video, web content, applications, software downloads, etc., each of which has distinctive properties.
  - Tens of millions of map units to capture all of the clients and traffic classes of the trillions of requests per day served by Akamai.

Choosing the Cluster: Global Load Balancing

- The goal of global load balancing is to assign each map unit M_i, $1 \leq i \leq M$, to a server cluster C_j , $1 \leq j \leq N$.
- For each map unit, the candidate clusters are ordered in descending order of preference
- Likewise, each server cluster C_j has preferences regarding which map units it would like to serve.
  - For example, a cluster deployed in an upstream provider ISP may prefer to serve clients in the ISP's downstream customer ISPs.
- The goal of global load balancing is to assign map units to clusters such that preferences are accounted for and capacity constraints are met.

Choosing the Cluster: Stable Allocations

- Classical algorithmic paradigm for assigning map units to clusters in global load balancing.
- Finding a stable marriage is achieved by a simple and distributed algorithm called the Gale-Shapley algorithm, that works in rounds
- The solution provides each map unit with the most preferred cluster possible in any stable marriage
  - fits the CDN's mission of maximizing performance for clients.



A stable marriage (marked in bold) is a matching of map units to clusters such that no unmatched pair prefer each other over their matched partners.

Choosing the Cluster: Stable Allocations

- Extension for CDN:
  - Unequal number of map units and clusters
    - Tens of millions of map units versus thousands of clusters
  - Partial preference lists: unnecessarily expensive to measure and rank every cluster for each map unit. It suffices to rank only those clusters that are close.
  - Modeling integral demands and capacities: canonical stable marriage problem considers unit value demands and capacity, it can be extended to the case in which capacity and demand are expressed as arbitrary integers. For each map unit, we estimate the content access traffic generated by its clients and represent that as a demand value. Likewise, for each cluster, we can estimate its capacity, which is the amount of demand it can serve.

Choosing the Cluster: Stable Allocations

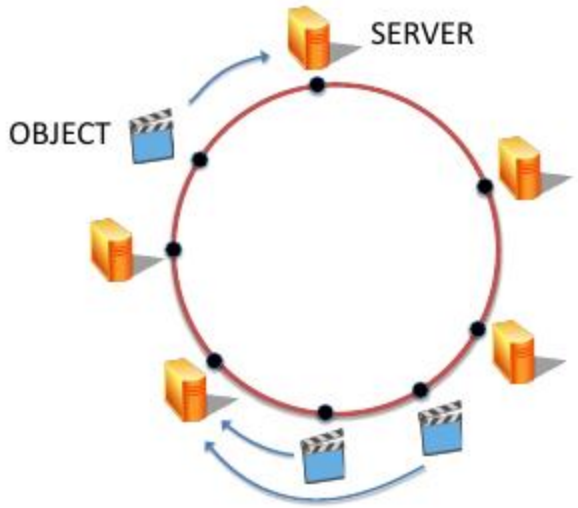
- Challenges:
  - Complexity and scale: load balancing across tens of millions of map units and thousands of clusters for over a dozen traffic classes
  - Time to solve: the map unit assignment must be recomputed every 10 to 30 seconds, as network performance and client demand change on short time scales.
  - Demand and capacity estimation: tight feedback loop to estimate demand and capacity based on past history
  - Incremental and persistent allocation.

Load balancing within a single cluster: from basic hashing…

- Hashing: remember the basics
- When a hash table is used in a sequential computer program, an arbitrary subset $S \subset U$ of $O(n)$ objects of interest are inserted into the buckets of B: each element $x \in S$ is inserted into bucket $h(x)$.
- An array of linked lists can be used to represent the buckets: insertion, removal and testing can be implemented in a straightforward manner so that the expected time per operation is constant.
- In the CDN setting, an object is a file such as a JPEG image or HTML page, and a bucket is the cache of a distinct web server.
- But: a server (bucket) may fail. The class of hash functions like $h(x)$ above does not deal well with that:
  - Simply remapping all of the objects in the lost bucket to another bucket is not ideal because then one bucket stores double the expected load.
  - Balancing the load by renumbering the existing buckets and rehashing the elements using a new hash function has the disadvantage that many objects will have to be transferred between buckets.

Load balancing within a single cluster: … to consistent hashing

- Consistent hashing solves this problem in a clever way. Consistent hashing first maps both objects and buckets (servers) to the unit circle. An object is then mapped to the next server that appears on the circle in clockwise order.
- If a bucket fails: the objects that were formerly mapped to the bucket are instead mapped to the next bucket that appears in clockwise order on the unit circle.
- If a new bucket is added: the only objects that have to be moved are those that are mapped to the new bucket.



Load balancing within a single cluster: Consistent hashing

- Popular Objects: a single object may be so popular that it is not possible for a single server within a cluster to satisfy all of the requests → must be served by more than one server.
- → A straightforward extension to consistent hashing would be to map a popular object to the next *k* servers that appear in clockwise order on the unit circle, where *k* is a function of the popularity of the object.
  - One disadvantage of this approach, however, is that if two very popular objects happen to hash to nearby positions , a lot of buckets they map to will overlap
- → the CDN uses a separate mapping of the buckets to the unit circle for each object

Load balancing within a single cluster: Consistent hashing

- Although an object is a single file to be served by a web server, the CDN does not hash each object independently: objects of the same content provider are hashed together.
- When a content provider signs up as a customer of the CDN, they are granted one or more serial numbers. The content provider's objects are grouped together by serial number, and all objects with the same serial number are hashed to the same bucket (or same set of buckets, if popular)
  - Beneficial when multiple objects must be fetched by to render a single web page.
  - The browser can make a persistent HTTP connection to just one randomly-chosen server in the set, and then fetch all of the objects from that server ->only 1 TCP cx

Load balancing within a single cluster: Consistent hashing

- ***Exemple:***

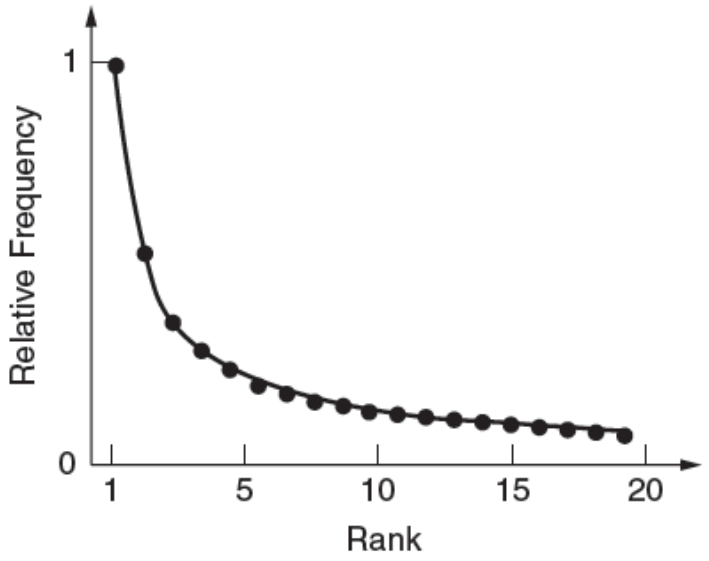
1. Serial numbers assigned to content providers range from 1 to 2048
2. CDN has assigned the serial number 212 to a customer owning www.example.com.
3. The DNS CNAME mechanism indicates that *www.example.com* is an alias for the canonical domain name *a212.g.akamai.net*.
4. When an authoritative name server operated by the CDN receives a request to resolve the name a212.g.akamai.net from a resolving name server, it first determines which cluster should serve the ensuing HTTP requests from browsers sharing this resolving name server.
    1. The cluster is chosen using ***stable allocations.***
5. Then ***consistent hashing*** is used to determine which server(s) within the cluster should serve objects under serial number 212.
6. Depending on the current popularity of the content being served under serial number 212, the authoritative name server for the CDN returns the first *k* addresses, skipping any servers that are not operational.
7. The *k* servers in the DNS response are listed in random order to help spread the load among them.

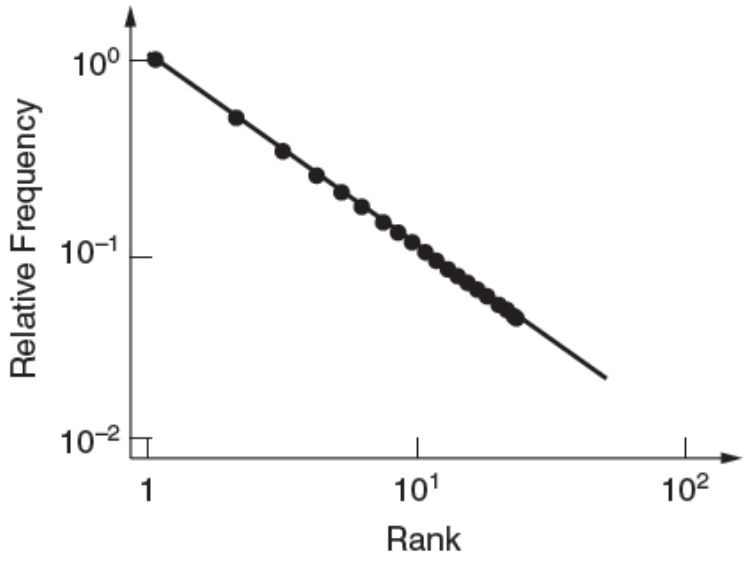
How a CDN works

1. CDN Basics
2. Server selection
3. **Content storage (caching)**
4. Overlay Routing

Optimizing storage: caching

- Where to cache content?
  - Popularity of Web objects is Zipf-like : $N_r \sim r^{-\alpha}$ where $\alpha < 1$
    - Also called heavy-tailed and power law
  - Small number of sites cover large fraction of requests (mice and elephants):
    - many small/unpopular and few large/popular flows – mice and elephants
- Given this observation, how should cache-replacement work?
  - Cache only popular contents





Optimizing storage: caching

- Cache-replacement policies :
  - which content to evict to leave room for the reactive (proactive) caching ?
- FIFO (first-in, first-out): The oldest content is always evicted.
  - Pros: very simple
  - Cons: ignores recency, popularity, size,..
- LRU (least recently used): The least recently referenced (with the *lowest requests count*) object is removed
  - Pros: temporal locality, simple to implement and fast :
    - New (or again referenced) objects are inserted at the head of the list.
    - Replacement takes place at the end of the list.
  - Cons: ignores popularity, size …

Optimizing storage: caching

- LFU (least frequently used): The least frequently referenced object is removed
  - Two implementations :
    - *Perfect LFU*. Perfect LFU counts all requests from the past to an object *i*. Request counts persist across replacements.
    - *In-Cache LFU*. Counts are defined for cache objects only.
  - Pros: Considers explicitly content popularity:
    - Different Web objects have different popularity values resulting in different access frequency values.
    - Replacement takes place at the end of the list.
  - Cons:
    - *Complexity*: We need to keep the hit count for the objects
    - *Cache pollution*: objects that were very popular during one time period can remain in the cache even when they are not requested for a long time period.
    - *Similar values*: Many objects can have the same frequency count.
    - Ignores content size

Optimizing storage: caching

- GSDF(Greedy-dual size frequency): The content *i* with the lowest priority p_i is removed, where p_i is computed like that :

$$p_i = f_i\,(c_i / s_i) + L$$

  - Where f_i is content frequency, c_i is content fetching cost, s_i is content size and L is a running aging factor, which is initialized to zero and updated when a replacement occurs with the priority value of the replaced content file.
  - Obviously this policy favors small content files to increase the cache hit rate (in terms of objects), in parallel to the access frequency and the recency (aging) of the cached content (through the monotonically increasing parameter *L*)
  - Pros: Considers explicitly content recency, popularity, sizes.
  - Cons: Complex implementation, sensitivity to the parameters values

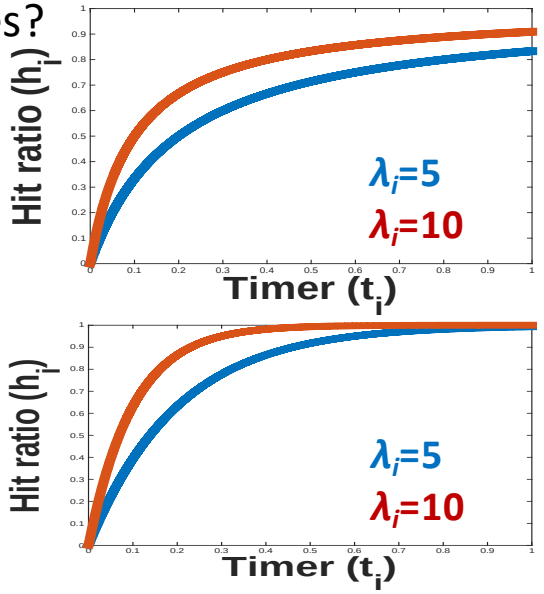
Optimizing storage: caching

- **Optimal cache design with time-to-live (TTL) caches**
  - **TTL cache:** eviction of *content i* occurs upon the expiration of a *timer* t_i
  - TTL used in DNS caches, why not in content caches?
  - Two TTL cache designs:
    - *Non-reset TTL Cache:* TTL is only set at cache misses, i.e. TTL is not reset upon cache hits.

$$h_i = 1 - \frac{1}{1 + \lambda_i t_i}$$

    - *Reset TTL Cache*: TTL is set each time the content is requested (cache misses + cache hits).

$$h_i = 1 - e^{-\lambda_i t_i}$$



  - where requests for file *i* arrive according to a Poisson process with rate λ_i and h_i is the *hit ratio* (*probability of hitting file i).*
  - **The *arrival rate* λ_i is related to *popularity of* file *i*, For example:**

popularities ~Zipf distribution with parameter $s = 0.8 < 1$, i.e. $\lambda_i = 1 / i^s$

Dehghan, Mostafa, et al. "A utility optimization approach to network cache design." Computer Communications, IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on. IEEE, 2016.

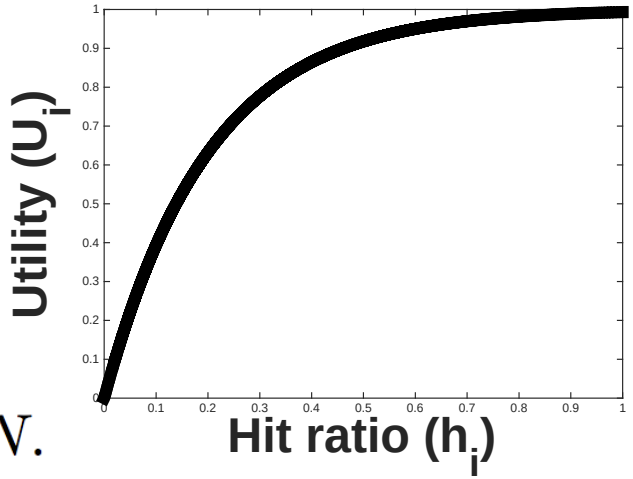
Optimizing storage: caching

- **Design cache problem:**
  - U_i : [0,1] → R is a utility function representing the "satisfaction" perceived by observing hit probability h_i. $U_i(\cdot)$ is assumed to be increasing, continuously differentiable, and strictly concave.
  - B is the cache size in terms of the expected no. of files (Assumption: all the files have the same size)

$$\text{maximize} \quad \sum_{i=1}^{N} U_i(h_i)$$

$$\text{such that} \quad \sum_{i=1}^{N} h_i = B$$

$$0 \leq h_i \leq 1, \quad i = 1, 2, \ldots, N.$$



Dehghan, Mostafa, et al. "A utility optimization approach to network cache design." Computer Communications, IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on. IEEE, 2016.

Optimizing storage: caching

- **Optimal solution:**
  - Using timer based caching techniques for controlling the hit probabilities: $0<t_i<\infty \rightarrow 0 < h_i < 1$, we can write the Lagrangian function as (α is the lagrangian multiplier):

$$\mathcal{L}(\mathbf{h},\alpha) = \sum_{i=1}^{N} U_i(h_i) - \alpha\left[\sum_{i=1}^{N} h_i - B\right]$$
$$= \sum_{i=1}^{N}\left[U_i(h_i) - \alpha h_i\right] + \alpha B,$$

  - where Applying optimality condition, we find the optimal solutions:

$$\frac{\partial \mathcal{L}}{\partial h_i} = \frac{\mathrm{d}U_i}{\mathrm{d}h_i} - \alpha = 0 \quad \Rightarrow \quad U_i'(h_i) = \alpha$$
$$h_i = U_i'^{-1}(\alpha)$$

  - Replacing in the cache storage constraint we obtain: $\sum_i h_i = \sum_i U_i'^{-1}(\alpha) = B$

Dehghan, Mostafa, et al. "A utility optimization approach to network cache design." Computer Communications, IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on. IEEE, 2016.

Optimizing storage: caching

- **Optimal solution:**
  - If the hit ratios $h_i(\alpha)$ are known, we can find the timers t_i for the
    - *Non-reset TTL Cache*

$$t_i = -\frac{1}{\lambda_i}\left(1 - \frac{1}{1 - U_i'^{-1}(\alpha)}\right)$$

    - *Reset TTL Cache*

$$t_i = -\frac{1}{\lambda_i}\log\left(1 - U_i'^{-1}(\alpha)\right)$$

Dehghan, Mostafa, et al. "A utility optimization approach to network cache design." Computer Communications, IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on. IEEE, 2016.

Optimizing storage: caching

- **Utility functions**

1. ***Identical Utility:*** All files have the same utility: $U_i(h_i) = U(h_i)$ for all i.

$$\sum_{i=1}^{N} U'^{-1}(\alpha) = NU'^{-1}(\alpha) = B$$

For the *hit probabilities* h_i we obtain:

$$U'^{-1}(\alpha) = B/N \qquad\qquad h_i = B/N, \quad \forall i.$$

And for the *timers* t_i:

– *Non-reset TTL Cache*

$$t_i = \frac{B}{\lambda_i(N-B)}$$

– *Reset TTL Cache*

$$t_i = -\frac{1}{\lambda_i}\log\left(1 - \frac{B}{N}\right).$$

Dehghan, Mostafa, et al. "A utility optimization approach to network cache design." Computer Communications, IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on. IEEE, 2016.

Optimizing storage: caching

- **Utility functions**

2. **β-*Fair Utility:*** Family of isoelastic functions *with β parameter* and coefficient $w_i \geq 0$ denoting the weight for file *i*.

$$U_i(h_i) = \begin{cases} w_i \frac{h_i^{1-\beta}}{1-\beta} & \beta \geq 0, \beta \neq 1; \\ w_i \log h_i & \beta = 1, \end{cases}$$

Special cases:

- **β=0 → Linear Utility:** $U_i(h_i) = w_i h_i \rightarrow \max_{h_i} \sum_i w_i h_i$
- For the *hit probabilities* h_i, we obtain (not from derivative of L, intuitively):

$$h_i = 1, i = 1, \ldots, B \qquad h_i = 0, i = B+1, \ldots, N \qquad w_1 \geq \ldots \geq w_N$$

- And for the *timers* t_i (regardless *Non-reset or reset TTL*)

$$t_i = \infty, i = 1, \ldots, B \qquad t_i = 0, i = B+1, \ldots, N$$

  - **If $w_i = \lambda_i$,** this policy becomes the **Least-Frequently Used (LFU) policy**

Dehghan, Mostafa, et al. "A utility optimization approach to network cache design." Computer Communications, IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on. IEEE, 2016.

Optimizing storage: caching

- **Utility functions**

2. **β-*Fair Utility:*** Family of isoelastic functions *with β parameter* and coefficient $w_i \geq 0$ denoting the weight for file *i*.

$$U_i(h_i) = \begin{cases} w_i \frac{h_i^{1-\beta}}{1-\beta} & \beta \geq 0, \beta \neq 1; \\ w_i \log h_i & \beta = 1, \end{cases}$$

Special cases:

- **β=1** → $U_i(h_i) = w_i \log h_i$ → $\max_{h_i} \sum_i w_i \log h_i$
- For the *hit probabilities* h_i we obtain:

$$\sum_i U_i'^{-1}(\alpha) = \sum_i w_i/\alpha = B \qquad \alpha = \sum_i w_i/B. \qquad h_i = U_i'^{-1}(\alpha) = \frac{w_i}{\sum_j w_j} B.$$

- And the *timers* t_i are computed as usual from h_i
- This utility function implements a **proportionally fair policy**: $\boldsymbol{\lambda_i}$,
- With $\boldsymbol{w_i = \lambda_i}$, the hit probability of file *i* is proportional to the request arrival rate $\boldsymbol{\lambda_i}$.

Dehghan, Mostafa, et al. "A utility optimization approach to network cache design." Computer Communications, IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on. IEEE, 2016.

Optimizing storage: caching

- **Utility functions**

2. **β-*Fair Utility:*** Family of isoelastic functions *with β parameter* and coefficient $w_i \geq 0$ denoting the weight for file *i*.

$$U_i(h_i) = \begin{cases} w_i \frac{h_i^{1-\beta}}{1-\beta} & \beta \geq 0, \beta \neq 1; \\ w_i \log h_i & \beta = 1, \end{cases}$$

Special cases:

- **β=2** → $U_i(h_i) = -w_i/h_i$ → $\max_{h_i} \sum_i \frac{-w_i}{h_i}$
- For the *hit probabilities* h_i we obtain:

$$\sum_i U_i'^{-1}(\alpha) = \sum_i \sqrt{w_i}/\sqrt{\alpha} = B \qquad \alpha = \left(\sum_i \sqrt{w_i}\right)^2 / B^2 \qquad h_i = U_i'^{-1}(\alpha) = \frac{\sqrt{w_i}}{\sqrt{\alpha}} = \frac{\sqrt{w_i}}{\sum_j \sqrt{w_j}} B$$

- And for the *timers* t_i are computed as usual from h_i
- This utility function is known to yield **minimum potential delay fairness**. It was shown that the TCP congestion control protocol implements such a utility function.

Dehghan, Mostafa, et al. "A utility optimization approach to network cache design." Computer Communications, IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on. IEEE, 2016.

Optimizing storage: caching

- **Utility functions**

2. **β-*Fair Utility:*** Family of isoelastic functions *with β parameter* and coefficient $w_i \geq 0$ denoting the weight for file *i*.

$$U_i(h_i) = \begin{cases} w_i \frac{h_i^{1-\beta}}{1-\beta} & \beta \geq 0, \beta \neq 1; \\ w_i \log h_i & \beta = 1, \end{cases}$$

Special cases:

- **β = ∞** → $\max_{h_i} \min_{i} h_i$
- For the *hit probabilities* h_i we obtain (not from derivative of L, intuitively):

$$h_i = B/N, \quad \forall i.$$

- And for the *timers* t_i are computed as usual from h_i
- The utility function defined here maximizes the *minimum hit probability*, and corresponds to the **max-min fairness.** Note that using identical utility functions for all files resulted in similar hit probabilities as this case.

Dehghan, Mostafa, et al. "A utility optimization approach to network cache design." Computer Communications, IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on. IEEE, 2016.

Optimizing storage: caching

- **Utility functions**

3. ***FIFO Utility:*** A non-reset TTL Cache can be used to implement a FIFO policy using the right value *T (characteristic time)* for the timer:

$$t_i = T, i = 1, \ldots, N \quad h_i = 1 - 1/(1+\lambda_i T) \quad \sum_i h_i = B.$$

- h_i(T)→ increasing function of *T,* h_i(T) → decreasing function of α
- Then, *T* is a decreasing function of *α* → In particular: T = 1/*α (from hit ratios formulae)*
- From the non-reset TTL hit ratio and with T = 1/*α,* we obtain:

$$h_i = 1 - \frac{1}{1+\lambda_i T}. \qquad U_i'^{-1}(\alpha) = 1 - \frac{1}{1+\lambda_i/\alpha}.$$

- Computing the inverse of the precedent function: $U_i'(h_i) = \frac{\lambda_i}{h_i} - \lambda_i,$

- And integrating the two sides of the above equation, we find the **FIFO utility**:

$$U_i(h_i) = \lambda_i(\log h_i - h_i).$$

Dehghan, Mostafa, et al. "A utility optimization approach to network cache design." Computer Communications, IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on. IEEE, 2016.

Optimizing storage: caching

- **Utility functions**

4. ***LRU Utility:*** A reset TTL Cache can be used to implement a LRU policy using the right value *T (characteristic time)* for the timer:

$$t_i = T, i = 1, \ldots, N \qquad h_i = 1 - e^{-\lambda_i T} \qquad \sum_i h_i = B.$$

- $h_i(T)$ → increasing function of *T*, $h_i(T)$ → decreasing function of α
- Then, *T* is a decreasing function of *α* → In particular: T = 1/*α (from hit ratios formulae)*
- From the non-reset TTL hit ratio and with T = 1/*α*, we obtain:

$$U_i'^{-1}(\alpha) = 1 - e^{-\lambda_i/\alpha}$$

- Computing the inverse of the precedent function: $U_i'(h_i) = \dfrac{-\lambda_i}{\log(1 - h_i)}$
- And integrating the two sides of the above equation, we find the **LRU utility**:

$$U_i(h_i) = \lambda_i \mathrm{li}(1 - h_i), \qquad \mathrm{li}(x) = \int_0^x \frac{\mathrm{d}t}{\ln t}.$$

Dehghan, Mostafa, et al. "A utility optimization approach to network cache design." Computer Communications, IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on. IEEE, 2016.

Getting an idea of whether the content is cached: Bloom Filters

- To approximately represent dynamically evolving sets in a space efficient manner.
- Useful in content delivery in: content summarization and content filtering.
- Basics:
- Suppose that we want to store a set $S = \{e_1, e_2, \cdots, e_n\}$ in a manner that allows us to efficiently insert new elements into the set and answer membership queries of the form "Is element *e* in set *S*?"
- A simple data structure is a hash function *h* that maps elements to a hash table $T[1 \cdots m]$, where each table entry stores 0 or 1, 0 first. When *e* is inserted into set S: $T[h(e)]$ is set to 1
- To check if $e \in S$, we simply need to check if $T[h(e)]$ is 1.
  - "false positives" are possible with this solution when e and $e' \in S$ are such that $h(e) = h(e')$
  - a "false negative" cannot occur
- Bloom filters are a generalization:
  - uses multiple hash functions $h_1, h_2, \cdots, h_k$ to reduce the probability of a false positive.
  - Still a table $T[1 \cdots m]$ with binary values, initialized to 0.
  - To insert *e*: the bits $T[h_i(e)], 1 \leq i \leq k$, are set to 1.
  - To check if $e \in S$: are $T[h_i(e)] = 1$, for all $1 \leq i \leq k$
  - False positives are still possible but less likely

Getting an idea of whether the content is cached: Bloom Filters

- Quantify the tradeoff between the false positive probability *p*, the number of elements *n*, the number of bits *m*, and the number of hash functions *k*

  -> exercise session

Bloom filters: for Content Summarization

- Cache summarization: a Bloom filter is used to succinctly summarize the list of objects stored in a CDN server.
- More space-efficient than storing a list of URLs associated with the objects in cache.
- Cache summarization can be used to locate which server cache has which objects.
  - For instance, if the CDN's servers periodically exchange cache summaries, a server that does not have a requested object in its cache can find it on other servers
- Example: implemented in popular web caching proxies like Squid, which create cache summaries called "digests"
  - Squid uses a Bloom filter with k = 4 where a single 128-bit MD5 hash of the object's identifier is partitioned into four 32-bit chunks and treated as four separate hashes.

Bloom filters: for Content Filtering

- Cache filtering: a Bloom filter to determine what objects to cache in the first place.
- Without cache filtering: a CDN's server caches each object that it serves.
  - When the disk cache fills up, the objects are evicted using a *cache replacement policy.*
  - But: no reason to cache objects likely to be accessed only once.
  - -> a significant amount of disk space would be saved (and number writes to disk)

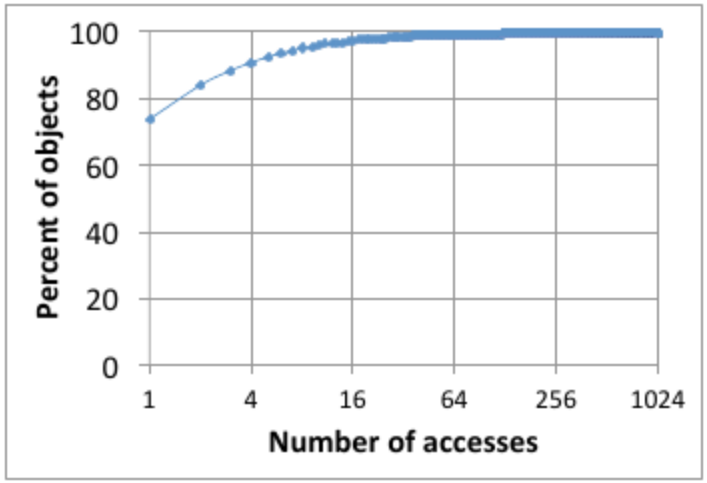


Figure 5: On a typical CDN server cluster serving web traffic over two days, 74% of the roughly 400 million objects in cache were accessed only once and 90% were accessed less than four times.

Bloom filters: for Content Filtering

- Cache-on-second-hit rule: a simple cache filtering rule
  - caches an object only when it is accessed for a second time within a specific time period.
  - can be implemented by storing the set of objects that have been accessed in a Bloom filter.
  - Upon request, the server first checks to see if the object has been accessed before by examining the Bloom filter.
    - If not, object fetched and served but not cached.
    - If so, the object has been accessed before, fetched, served, and stored

Bloom filters: for Content Filtering

- Most CDNs implement cache replacement algorithms such as LRU, which evict less popular objects such as one-hit-wonders when the cache is full.
- -> Filtering out the less popular objects and not placing them in cache at all provides additional disk space for more popular objects and increases the byte hit rate.

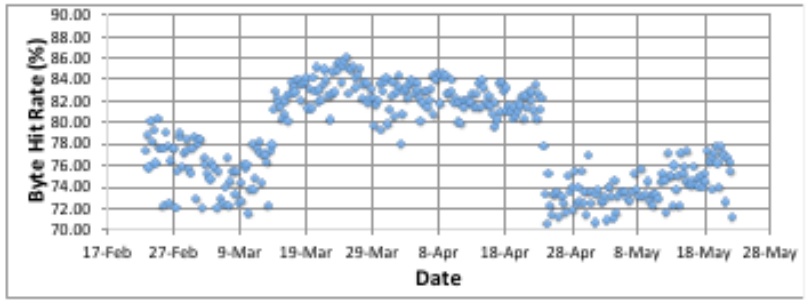


Figure 6: Byte hit rates increased when cache filtering was turned on between March 14th and April 24th because not caching objects that are accessed only once leaves more disk space to store more popular objects.

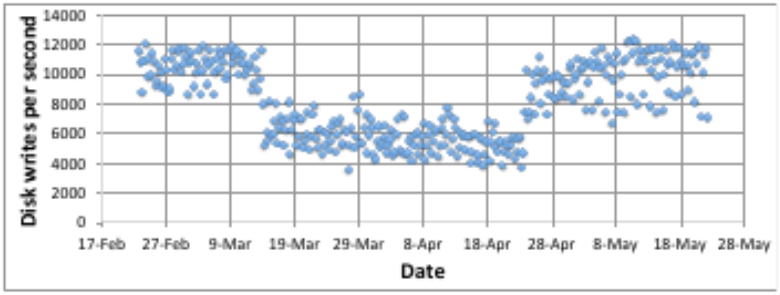


Figure 7: Turning on cache filtering decreases the rate of disk writes by nearly one half because objects accessed only once are not written to disk.

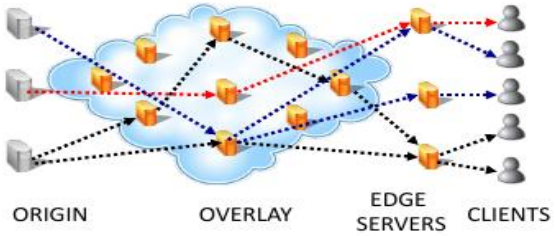
Bloom filters: Challenges

- Speeding up the Bloom filter: needs to be extremely fast because Bloom filter operations are on the critical path for client-perceived CDN performance
- Sizing the Bloom filter: space efficiency is the main reason why Bloom filters are used in place of simpler hash table based solutions
- Given the number *n* of objects and false positive proba *p*, the number of *k* of hash function and Bloom filter table size (*m*) can be calculated.
- Complex rules may need to be implemented in practice that take into consideration other object popularity metrics and other characteristics such as size.

How a CDN works

1. CDN Basics
2. Server selection
3. Content storage (caching)
4. **Overlay Routing**

Overlay routing



- Most web sites have some content that is either not cacheable or cacheable for only short periods of time. Ex: a banking portal - dynamic web content
  - A significant part is personalized for individual users and cannot be cached,
  - though some page elements such as CSS, JS, and images may be common across users and cached.
- CDNs also host applications that users can interact with in real-time: cannot be cached
- Live streaming and teleconferencing are delivered by CDNs in real-time and are uncacheable.
- Even for cachable content, often time-to-live (TTL) to periodically refresh it (ex: stock chart)
- A common framework that can capture all of the above situations is shown in the fig:
  - origins that create the content,
  - edge servers that clients access to consume the content
  - an overlay network that is responsible for transporting the content from the origins to the edges.
- Clients request the content from a proximal edge server which downloads the requested content from the origin via the overlay network.
  - case of a web site: the origin is a collection of application servers, databases, and web servers deployed by the content provider in one or more data centers on the Internet.
  - case of a live stream: the origin denotes the servers that receive the stream in real-time from the encoders capturing the event.
- The edge servers are operated by the CDN and are deployed in more than a thousand data centers around the world, so as to be proximal to clients

Overlay routing

- A path from origin to the edge server that does not pass through any intermediate overlay servers is called the direct path. The direct path is always used when no overlay path is superior to it.
- Key problem: how to construct an overlay to provide efficient communication between origins and edge servers.
- An overlay construction algorithm takes as input
  - client demands, which dictate which origins need to send their content to which edge servers,
  - real-time network measurements of latency, loss, and available bandwidth for the paths through the Internet between origins, overlay servers, and edge servers.
- Algorithmic Solutions

Overlay routing

- Dynamic web content:
  - latency sensitive -> customized algorithms for the well-studied all-pairs shortest-path (APSP) problem.
  - Note that overlay construction involves both cost and capacity constraints.
  - A first step to constructing an APSP instance is to perform Lagrangian relaxation (i.e., penalizing violations of constraints rather than forbidding them) to encode the capacity constraints as a cost that can be added to existing link-performance-dependent cost terms.
- Live videos:
  - throughput sensitive -> account for the capacity constraints in the overlay and the bandwidth bottlenecks in the Internet.
  - One approach is to formulate a mixed integer program (MIP) that captures all the performance, capacity, and bandwidth constraints.
  - The MIP often cannot be solved efficiently as it is NP-hard.
  - -> To overcome this problem, we can “relax” the integral variables in the MIP so that they can take on real values, resulting in a linear program (LP).

Overlay routing: benefits

- On a consistent basis, overlay routing provides better QoE (faster download or fewer video rebuffers).
  - the overlay paths may have lower latency and/or higher throughput depending.
  - the CDN can save the overhead of establishing TCP cx between different nodes by holding them persistently.
  - the CDN may also use optimized transport protocols between their nodes.
- Catastrophic events such as a cable cut are not rare on the Internet -> overlays provide alternate paths when the direct path from origin to edge is impacted

Overlay routing: benefits

- E.g.: April 2010, a large-scale Internet outage occurred when a submarine communications cable system (linking Europe, Middle East and South Asia) was cut.
  - The cable underwent repairs from 25 April to 29 April during which time several cable systems were affected, severely impacting Internet connectivity in many regions across the Middle East, Africa, and Asia.
  - Figs shows the download time by clients in Asia
    - agents distributed across India, Malaysia, and Singapore to download a dynamic (i.e., uncacheable) web page approximately 70KB in size hosted at an origin in Boston.

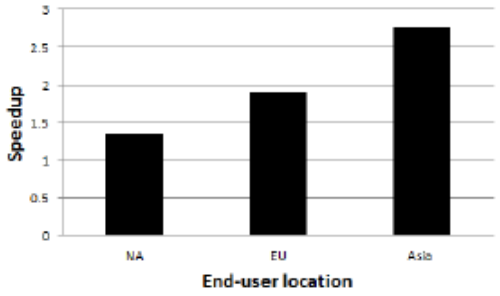


Figure 10: A routing overlay provides significant speedups by choosing better performing paths from the origin to the client. Key: North America (NA), Europe (EU), Asia.

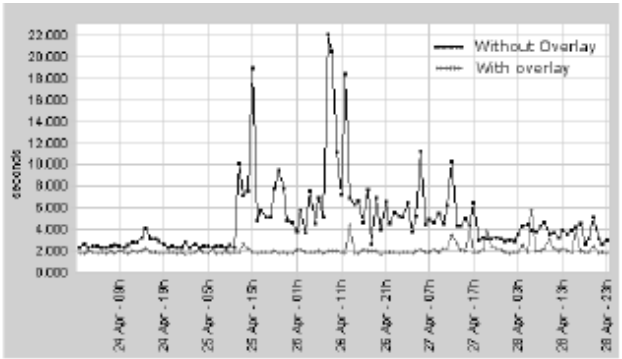


Figure 11: Performance of the routing overlay during a cable cut.

Outline

1. Content Distribution Networks (CDNs)
   a. Motivation
      a. Current Internet Content
      b. Several servers. Why?
      c. Web Caching vs CDNs
   b. How a CDN works
      a. CDN Basics
      b. Server selection
      c. Content storage (caching)
      d. Overlay Routing
   c. Akamai Example

Akamai Example

Akamai Statistics (2014)

- Deployment
  - 147K+ servers, 1200+ networks, 650+ cities, 92 countries
  - highly hierarchical, caching depends on popularity
  - 4 yr depreciation of servers
  - Many servers inside ISPs, who are thrilled to have them
  - Deployed inside 100 new networks in last few years
- Customers
  - 250K+ domains: all top 60 eCommerce sites, all top 30 M&E companies, 9 of 10 to banks, 13 of top 15 auto manufacturers
- Overall stats
  - 5+ terabits/second, 30+ million hits/second, 2+ trillion deliveries/day, 100+ PB/day, 10+ million concurrent streams
  - 15-30% of Web traffic

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 (*Akamizing Links*)
  - E.g. <img src="http://**www**.cnn.com/af/x.gif"> replaced with

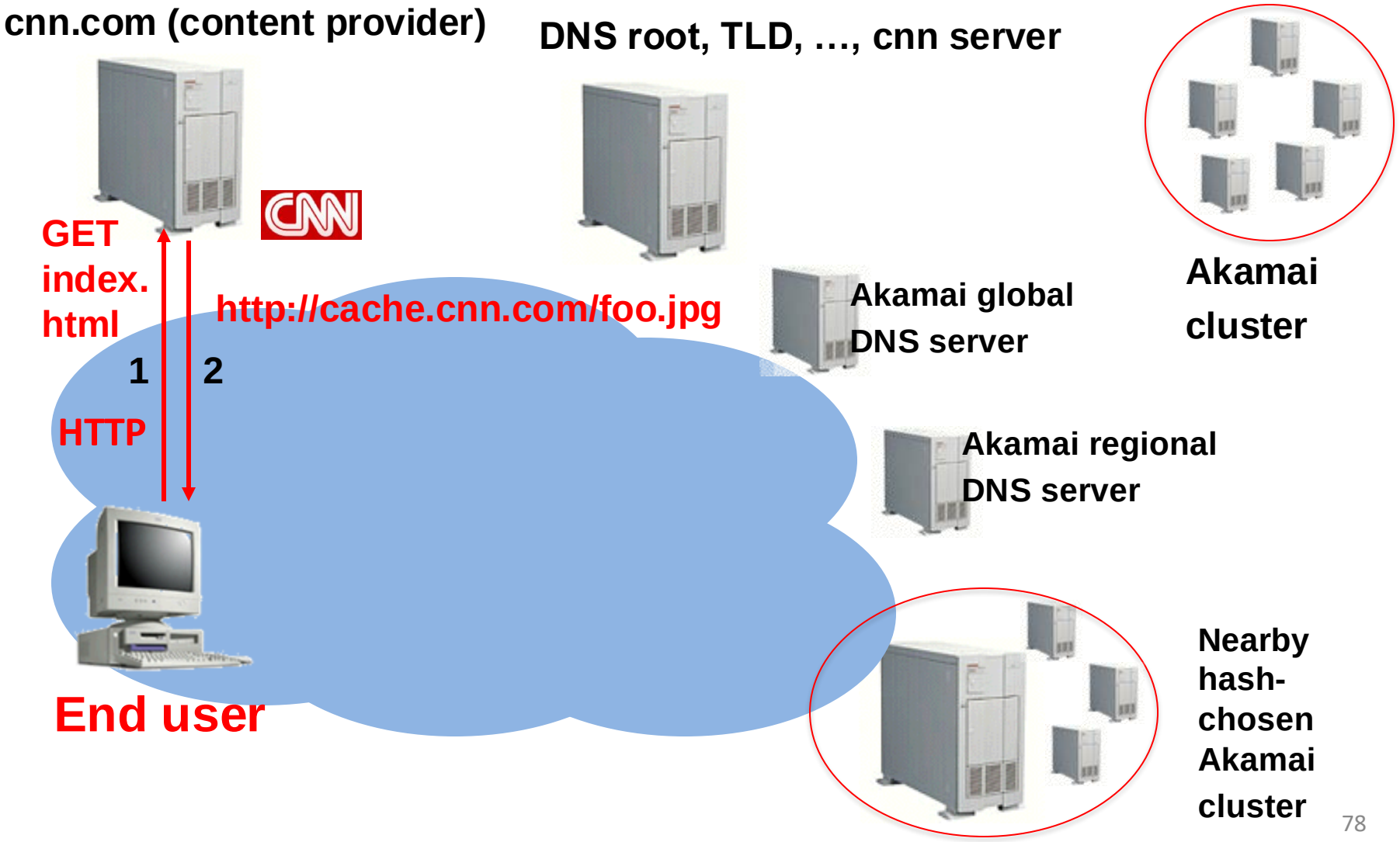
    <img src=http://**a73**.g.akamai.net/7/23/www.cnn.com/af/x.gif>
  - Modified name contains original file name
    - <img src=http://a73.g.akamai.net/7/23/*www.cnn.com/af/x.gif*>
- DNS query: CNN (CP) adds CNAME (alias) for cache.cnn.com

  cache.cnn.com → a73.g.akamai.net
- Client resolves aXYZ.g.akamaitech.net hostname
  - Maps to a server in one of Akamai's clusters

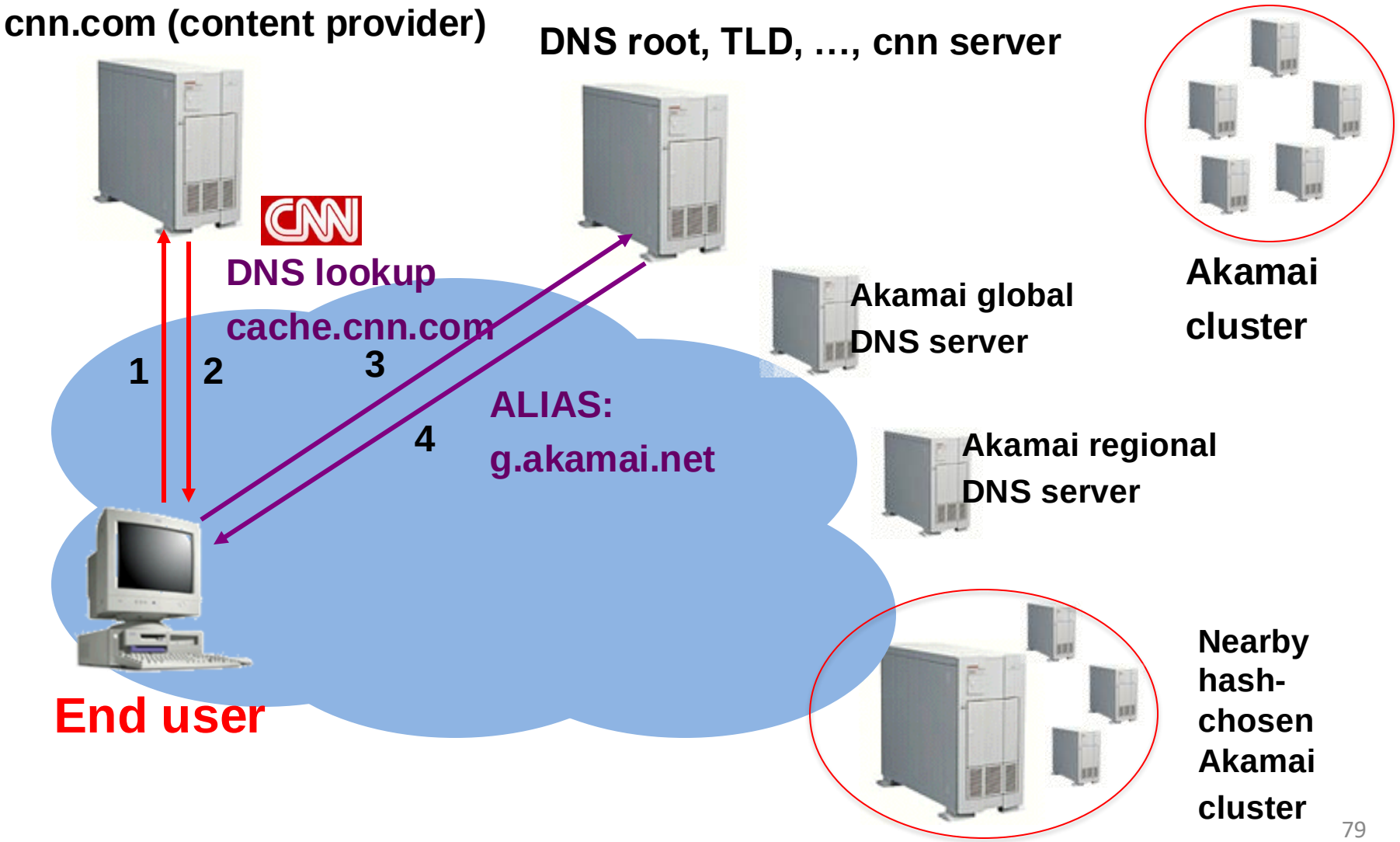
How Akamai Works

- Root server gives NS record for akamai.net
- This nameserver returns NS record for g.akamai.net
  - Nameserver chosen to be in region of client's name server
  - TTL is large
- g.akamai.net nameserver chooses server in region
  - Should try to chose server that has file in cache
  - Uses aXYZ name and hash
  - TTL is small
- When Akamai server is found, it is asked for content
  1. Checks local cache
  2. Check other servers in local cluster
  3. Otherwise, requests from primary server and cache file

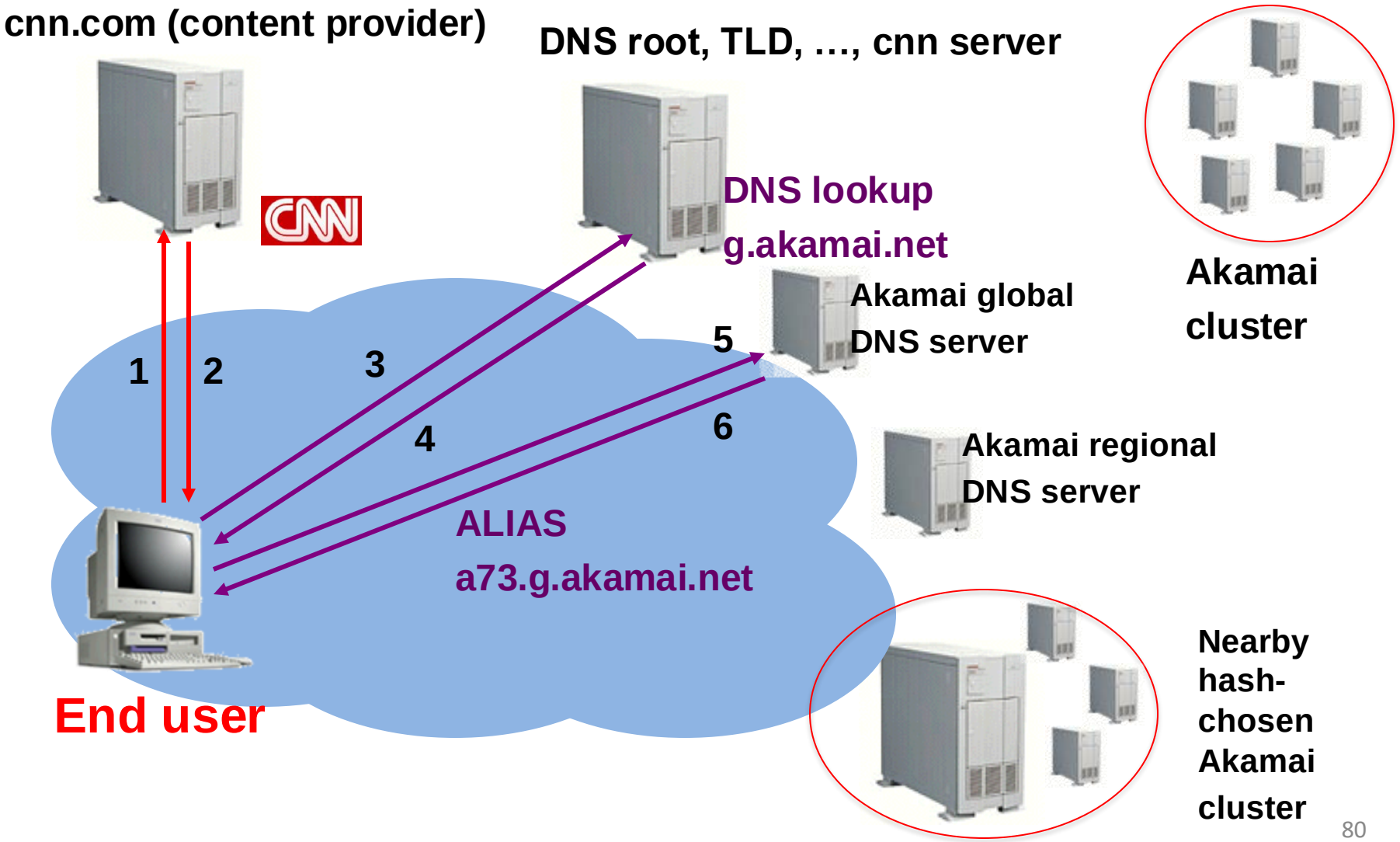
How Akamai Works



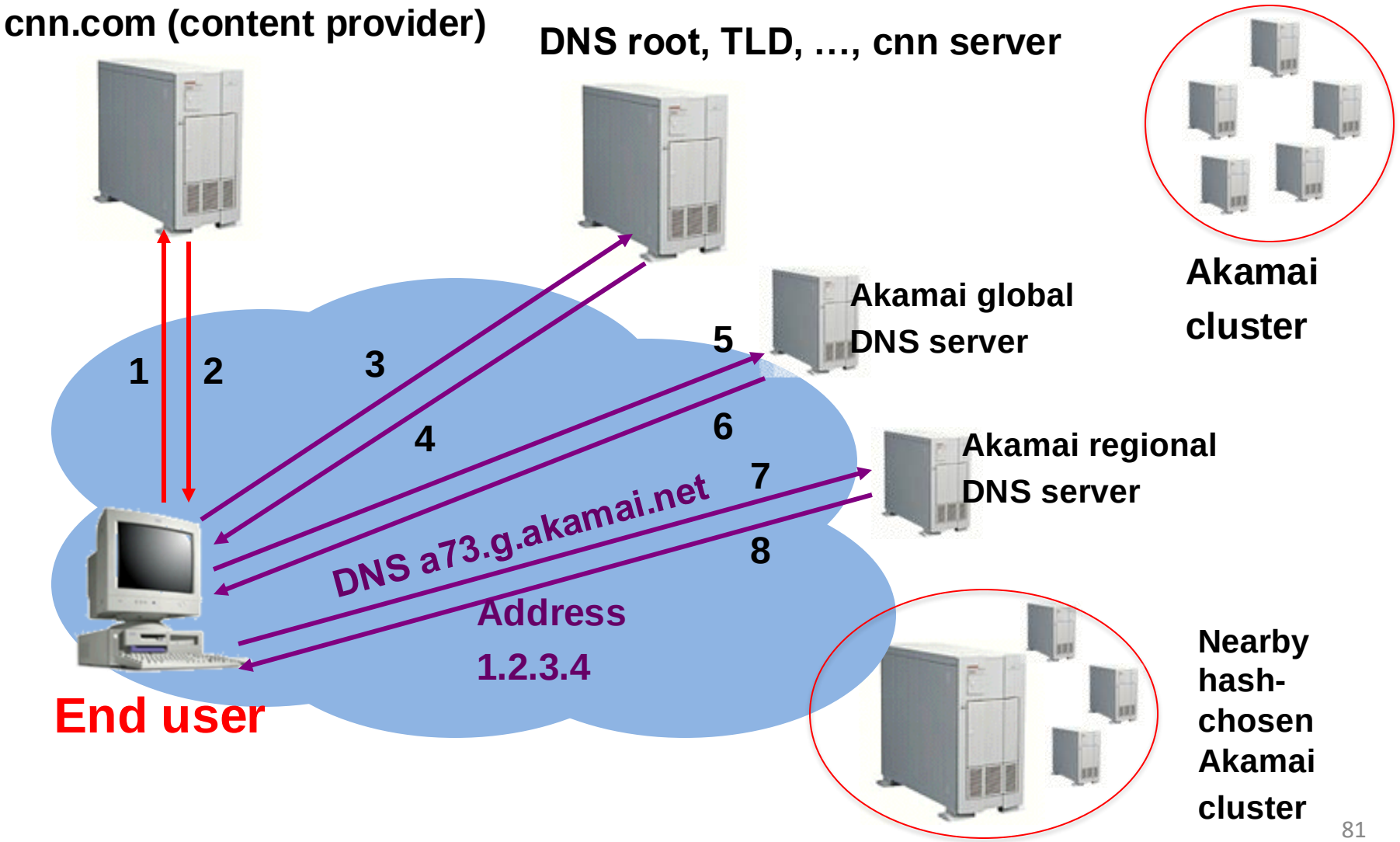
How Akamai Works



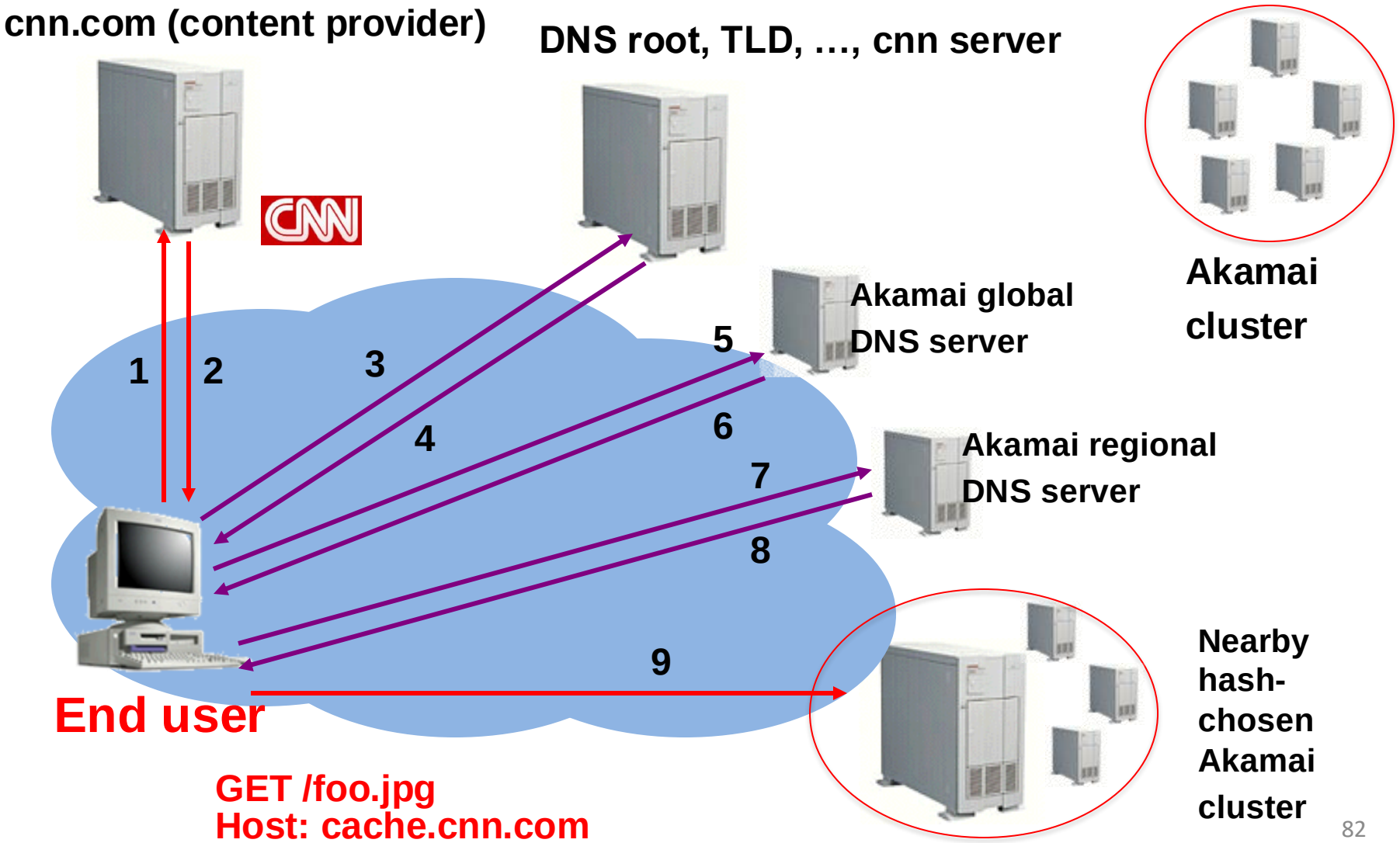
How Akamai Works



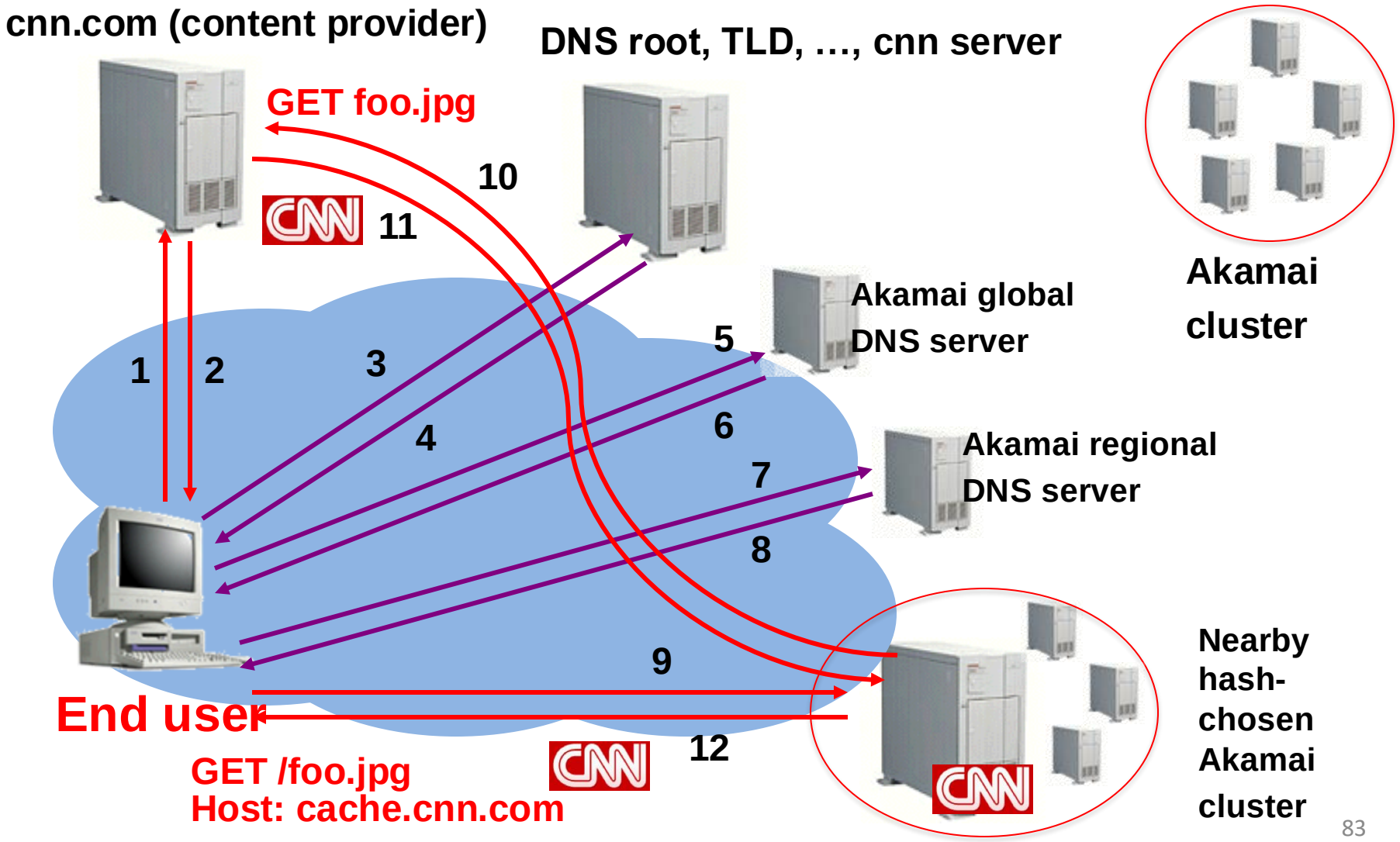
How Akamai Works



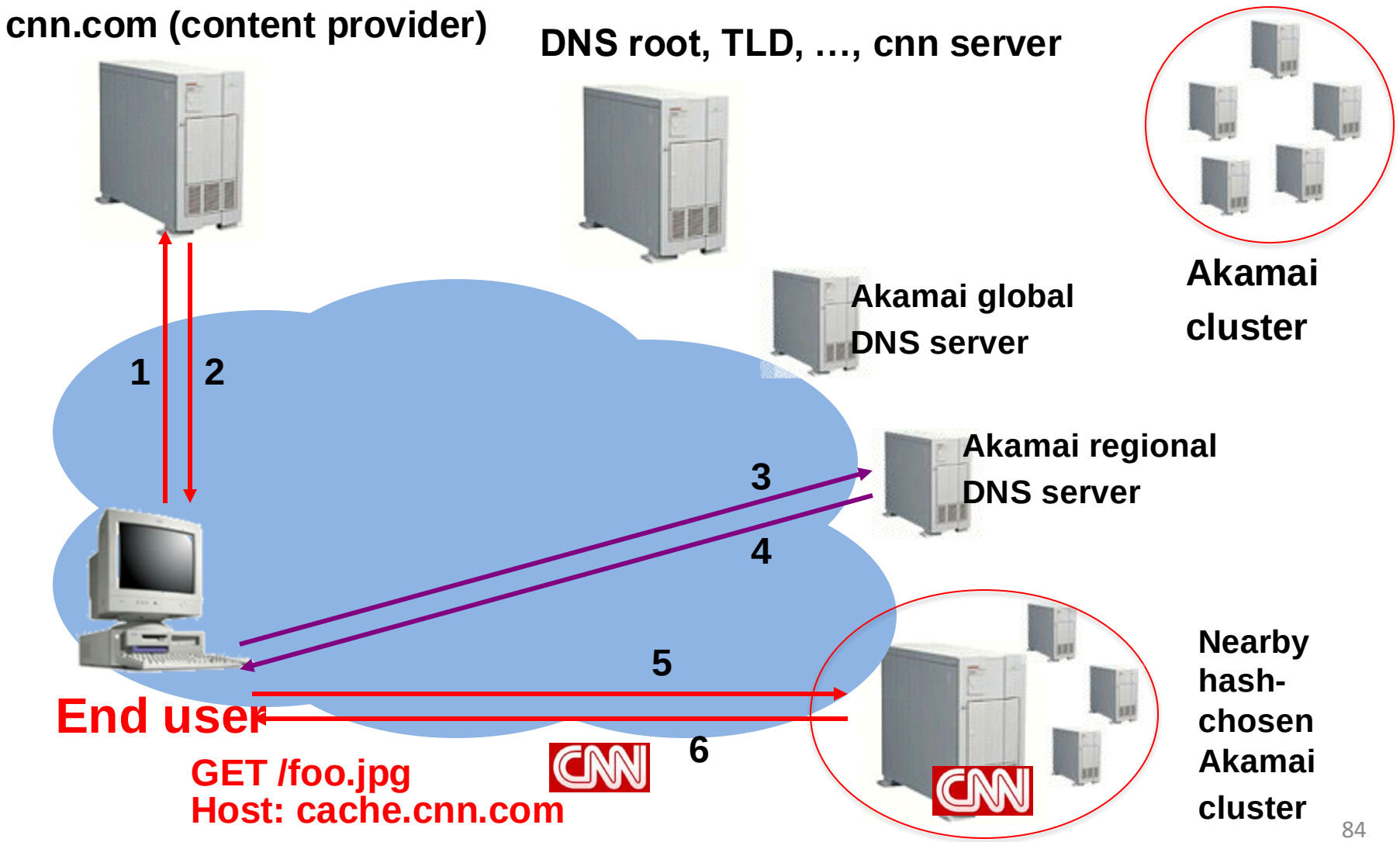
How Akamai Works



How Akamai Works

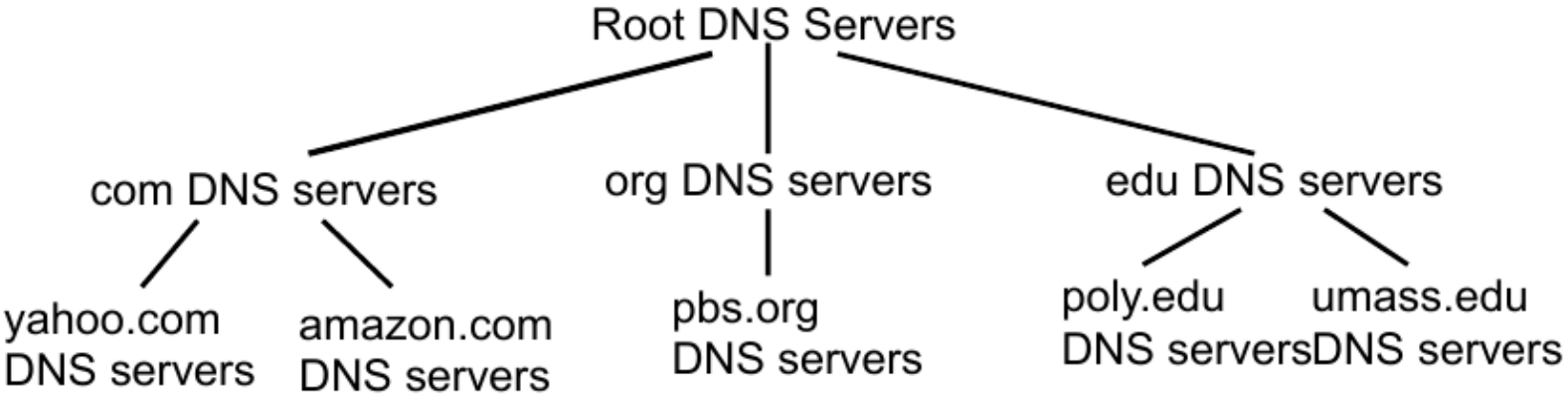


How Akamai Works (*Already cached*)



ANNEX SLIDES

Refresher: DNS – Distributed, Hierarchical Database



- Client wants IP for www.amazon.com:
  1. client queries a root server to find com DNS server
  2. client queries com DNS server to get amazon.com DNS server
  3. client queries amazon.com DNS server to get IP address for www.amazon.com
- Local DNS servers may do this on behalf of client (recursively)
  - They also cache results which get stale over time -> ask again from authoritative server

Role of DNS in content distribution

- DNS is typically used to perform initial assignments of clients to servers
  - Client resolves video server's DNS name to IP address
- DNS name to address mapping under control of the content provider
  - Content provider's DNS servers provide authoritative answers to DNS requests
- DNS mapping often anycast -> one name maps to one of several addresses
  - Suitable server can be chosen for a given client at a given time instance