Network Applications: Operational Analysis; Load Balancing among Multiple Servers

Y. Richard Yang

http://zoo.cs.yale.edu/classes/cs433/

2/29/2016

Admin

- Assignment three status and questions.

Recap: Operational Analysis

- Objective: derive relationships among (measured) operational performance metrics

- T: observation interval
- Bi: busy time of device i
- i = 0 denotes system
- Ai: # arrivals to device i
- Ci: # completions at device i

$$\text{Arrival rate } \lambda_i = \frac{A_i}{T}$$

$$\text{Throughput } \mathrm{X}_i = \frac{C_i}{T}$$

$$\text{Utilization } \mathrm{U}_i = \frac{B_i}{T}$$

$$\text{Mean service time } \mathrm{S}_i = \frac{B_i}{C_i}$$

$$\text{Utilization } \mathrm{U}_i = X_i S_i$$

Forced Flow Law

- Assume each request visits device i Vi times

$$\text{Throughput } X_i = \frac{C_i}{T}$$

$$= \frac{C_i}{C_0} \frac{C_0}{T}$$

$$= V_i X$$

Bottleneck Device

$$\text{Utilization } U_i = X_i S_i$$

$$= V_i X S_i$$

$$= X V_i S_i$$

- Define Di = Vi Si as the total demand of a request on device i
- The device with the highest Di has the highest utilization, and thus is called the bottleneck

Bottleneck vs System Throughput

$$\text{Utilization } \mathrm{U}_i = XV_iS_i \leq 1$$

$$\rightarrow \quad X \leq \frac{1}{D_{\max}}$$

Example 1

- A request may need
  - 10 ms CPU execution time
  - 1 Mbytes network bw
  - 1 Mbytes file access where
    - 50% hit in memory cache
- Suppose network bw is 100 Mbps, disk I/O rate is 1 ms per 8 Kbytes (assuming the program reads 8 KB each time)
- Where is the bottleneck?

Example 1 (cont.)

- CPU:
  - D_{CPU}= 10 ms (e.g. 100 requests/s)
- Network:
  - D_{Net} = 1 Mbytes / 100 Mbps = 80 ms (e.g., 12.5 requests/s)
- Disk I/O:
  - Ddisk = 0.5 * 1 ms * 1M/8K = 62.5 ms (e.g. = 16 requests/s)

Example 2

- A request may need
  - 150 ms CPU execution time (e.g., dynamic content)
  - 1 Mbytes network bw
  - 1 Mbytes file access where
    - 50% hit in memory cache
- Suppose network bw is 100 Mbps, disk I/O rate is 1 ms per 8 Kbytes (assuming the program reads 8 KB each time)
- Bottleneck: CPU -> use multiple threads to use more CPUs, if available, to avoid CPU as bottleneck

Interactive Response Time Law

- System setup
  - Closed system with N users
  - Each user sends in a request, after response, think time, and then sends next request
  - *Notation*
    - *Z = user think-time, R = Response time*
  - The total cycle time of a user request is *R+Z*

In duration T, #requests generated by each user: *T/(R+Z) requests*

Interactive Response Time Law

- *If N users and flow balanced:*

System Throughput X = Toal# req./T

$$= \frac{N\frac{T}{R+Z}}{T}$$

$$= \frac{N}{R+Z}$$

$$R = \frac{N}{X} - Z$$

Bottleneck Analysis

$$X(N) \le \min\left\{\frac{1}{D_{max}}, \frac{N}{D+Z}\right\}$$

$$R(N) \ge \max\{D, ND_{max} - Z\}$$

- Here D is the sum of Di

Proof

$$X(N) \le \min\{\tfrac{1}{D_{\max}}, \tfrac{N}{D+Z}\}$$

$$R(N) \ge \max\{D, ND_{\max} - Z\}$$

- We know

$$X \le \frac{1}{D_{\max}} \qquad R(N) \ge D$$

Using interactive response time law:

$$R = \frac{N}{X} - Z \implies R \ge ND_{\max} - Z$$

$$X = \frac{N}{R+Z} \implies X \le \frac{N}{D+Z}$$

Summary: Operational Laws

- Utilization law: U = XS
- Forced flow law: Xi = Vi X
- Bottleneck device: largest Di = Vi Si
- Little's Law: Qi = Xi Ri
- Bottleneck bound of interactive response (for the given closed model):

$$X(N) \le \min\{\frac{1}{D_{\max}}, \frac{N}{D+Z}\}$$

$$R(N) \ge \max\{D, ND_{\max} - Z\}$$

In Practice: Common Bottlenecks

- No more file descriptors
- Sockets stuck in TIME_WAIT
- High memory use (swapping)
- CPU overload
- Interrupt (IRQ) overload

[Aaron Bannert]

http://video.google.com/videoplay?docid=-6304964351441328559#

YouTube Design Alg.

```
while (true)
{
  identify_and_fix_bottlenecks();
  drink();
  sleep();
  notice_new_bottleneck();
}
```

Summary: High-Perf. Network Server

- Avoid blocking (so that we can reach bottleneck throughput)
  - Introduce threads
- Limit unlimited thread overhead
  - Thread pool, async io
- Shared variables
  - Synchronization (lock, synchronized)
- Avoid busy-wait
  - Wait/notify; FSM
- Extensibility/robustness
  - Language support/Design for interfaces
- System modeling and measurements
  - Queueing analysis, operational analysis

Outline

- Recap
- Single network server
- Multiple network servers
  - Why multiple network servers

Why Multiple Servers?

- Scalability
  - Scaling beyond single server capability
    - There is a fundamental limit on what a single server can
      - process (CPU/bw/disk throughput)
      - store (disk/memory)
  - Scaling beyond geographical location capability
    - There is a limit on the speed of light
    - Network detour and delay further increase the delay

Why Multiple Servers?

- Redundancy and fault tolerance
  - Administration/Maintenance (e.g., incremental upgrade)
  - Redundancy (e.g., to handle failures)

Why Multiple Servers?

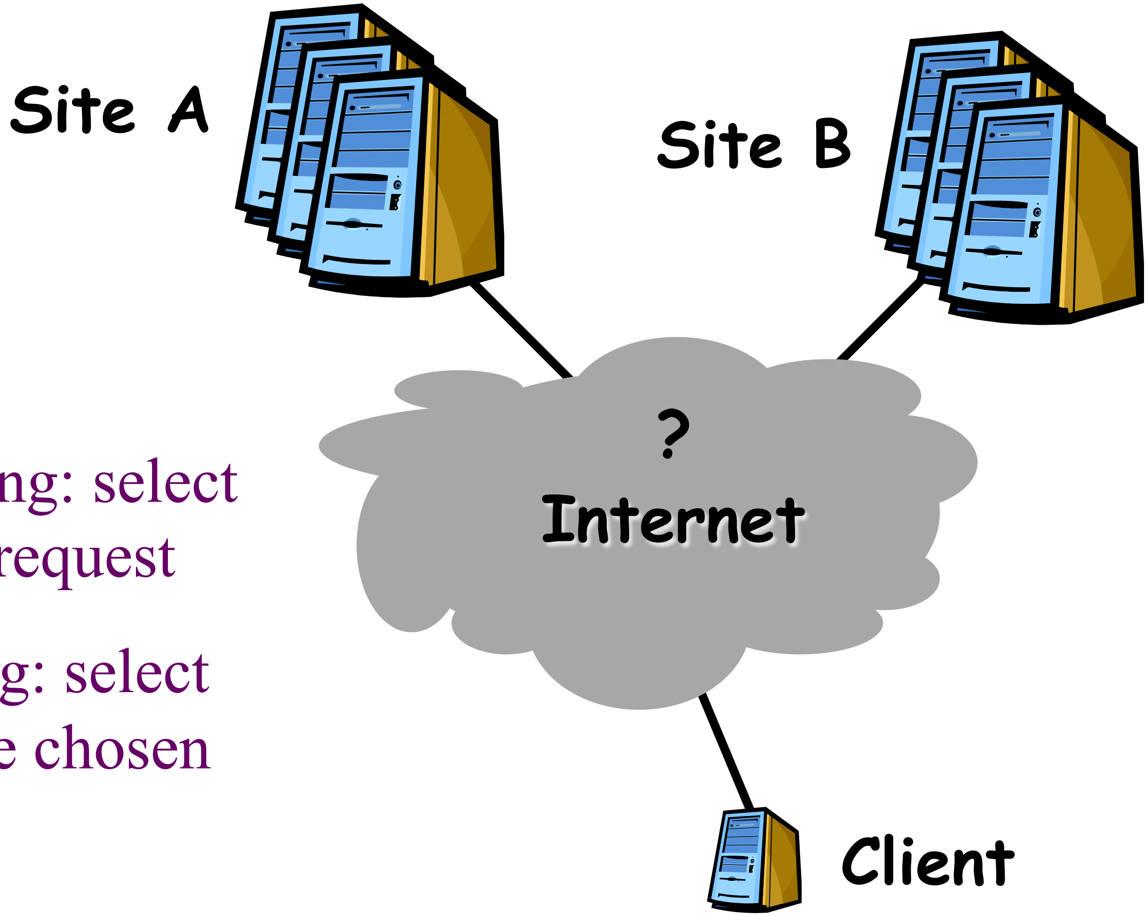
- System/software architecture
  - Resources may be naturally distributed at different machines (e.g., run a single copy of a database server due to single license; access to resource from third party)
  - Security (e.g., front end, business logic, and database)

Discussion: Key Technical Challenges in Using Multiple Servers

Outline

- Recap
- Single network server
- Multiple network servers
  - Why multiple servers
  - Request routing mechanisms

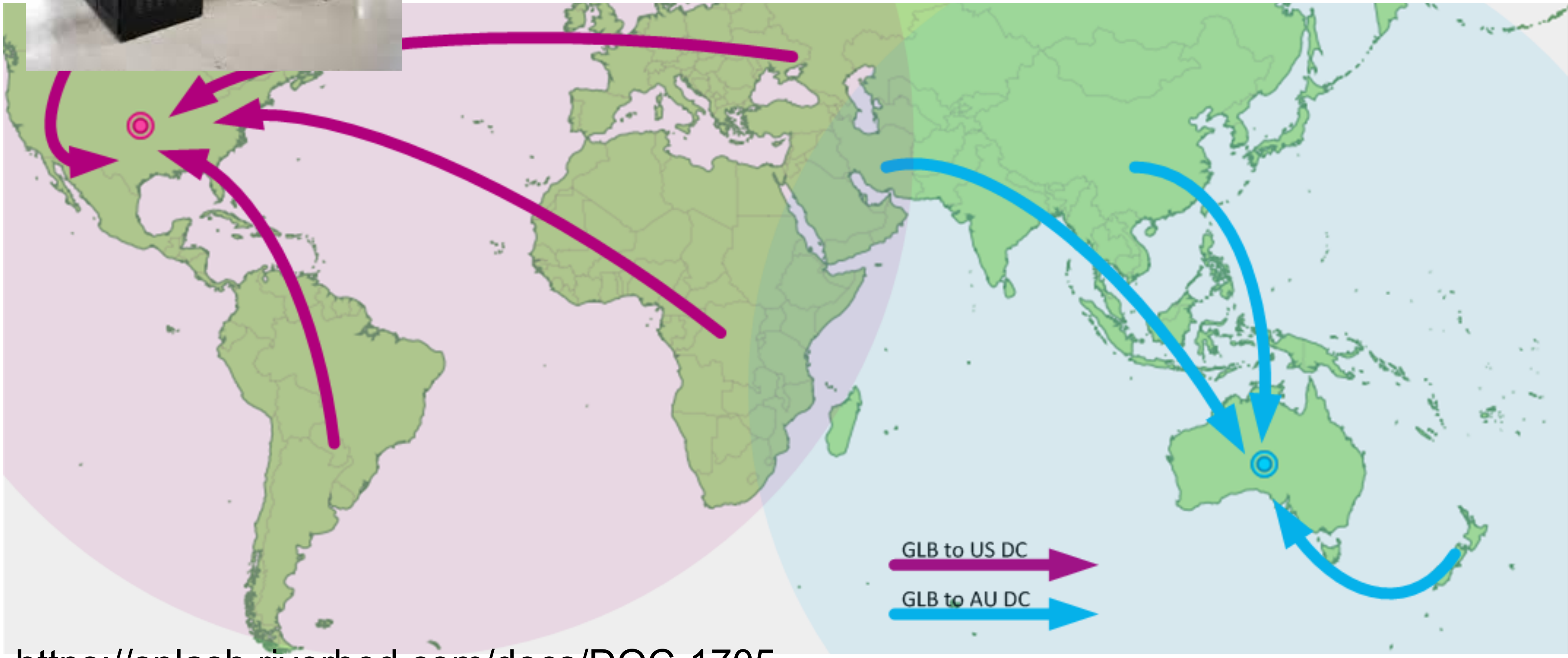
Request Routing: Overview



- Global request routing: select a server site for each request

- Local request routing: select a specific server at the chosen site

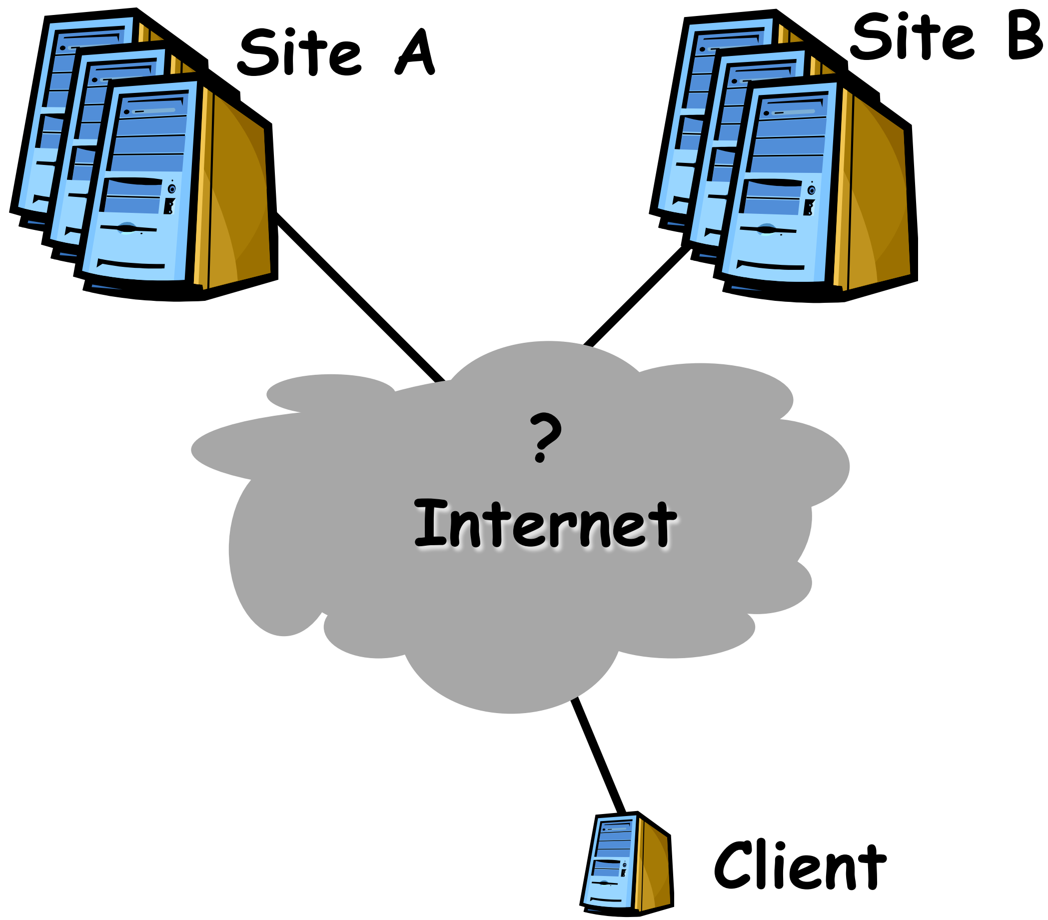
Request Routing: Overview



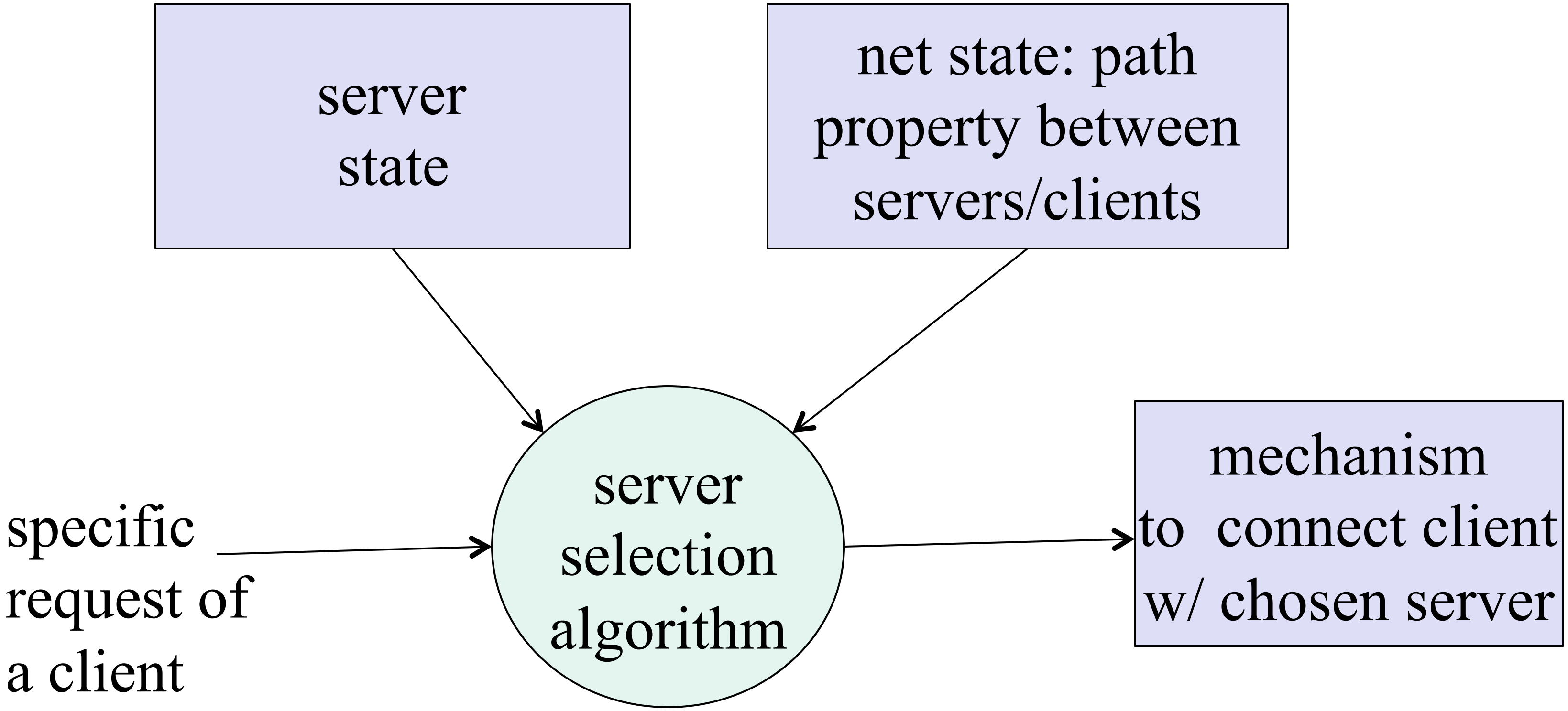
https://splash.riverbed.com/docs/DOC-1705

Request Routing: Basic Architecture

- Major components
  - Server state monitoring
    - Load (incl. failed or not); what requests it can serve
  - Network path properties between clients and servers
    - E.g., bw, delay, loss, network cost
  - Server selection alg.
  - Request direction mechanism

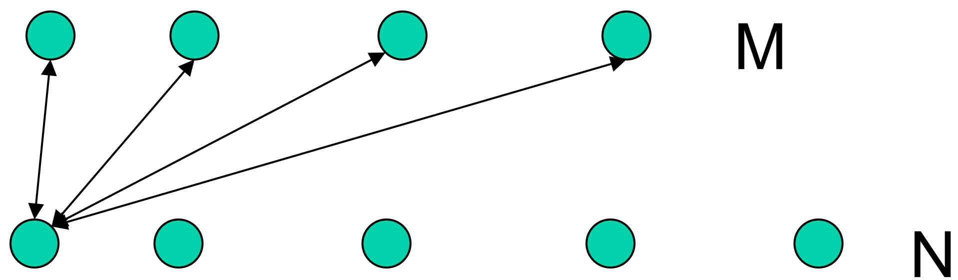


Request Routing: Basic Arch



Network Path Properties

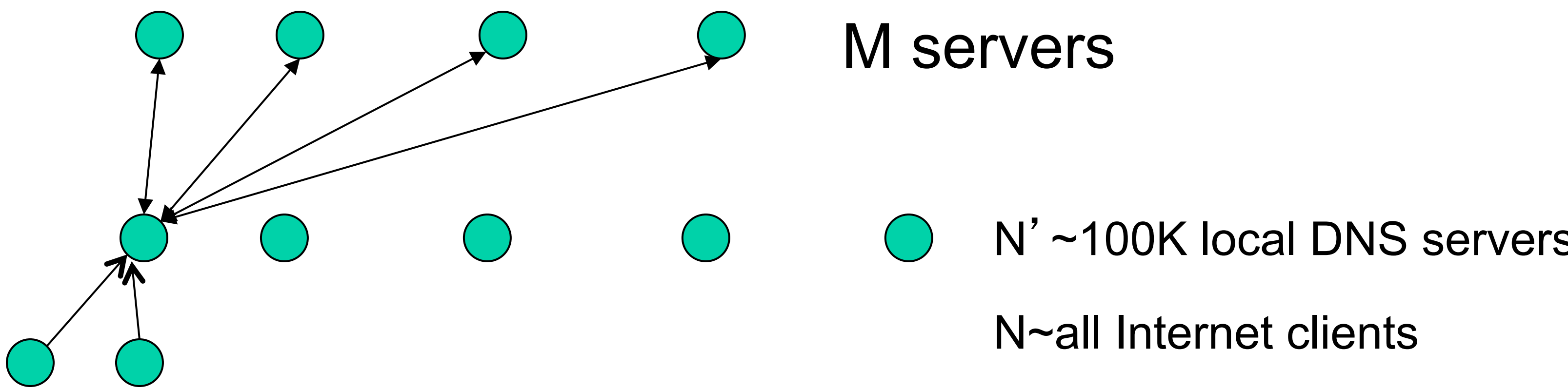
- Why is the problem difficult?
  - Scalability: if do measurements, complete measurements grow with N * M, where
    - N is # of clients
    - M is # of servers



  - Complexity/feasibility in computing path metrics

Network Path Properties: Improve Scalability

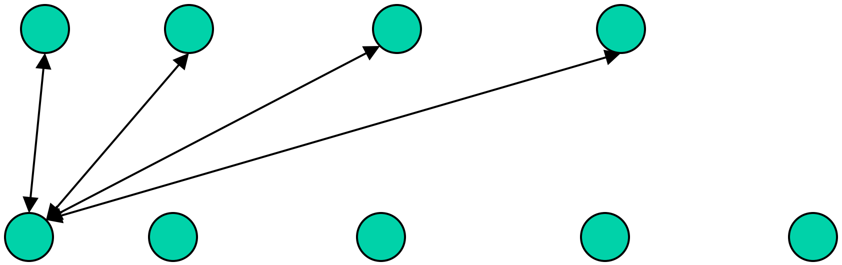
- Aggregation:
  - merge a set of IP addresses (reduce N and M)
    - E.g., when computing path properties, Akamai aggregates all clients sharing the same local DNS server



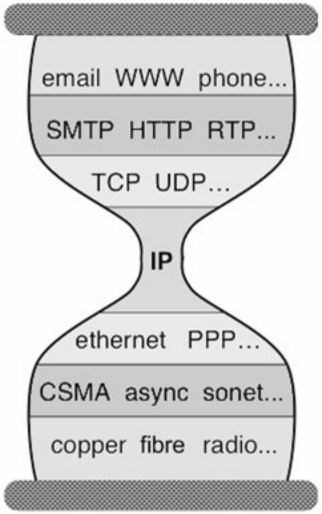
- Sampling and prediction
  - Instead of measuring N*M entries, we measure a subset and predict the unmeasured paths
  - We will cover it later in the course

Server Selection

- Why is the problem difficult?
  - What are potential problems of just sending each new client to the lightest load server?

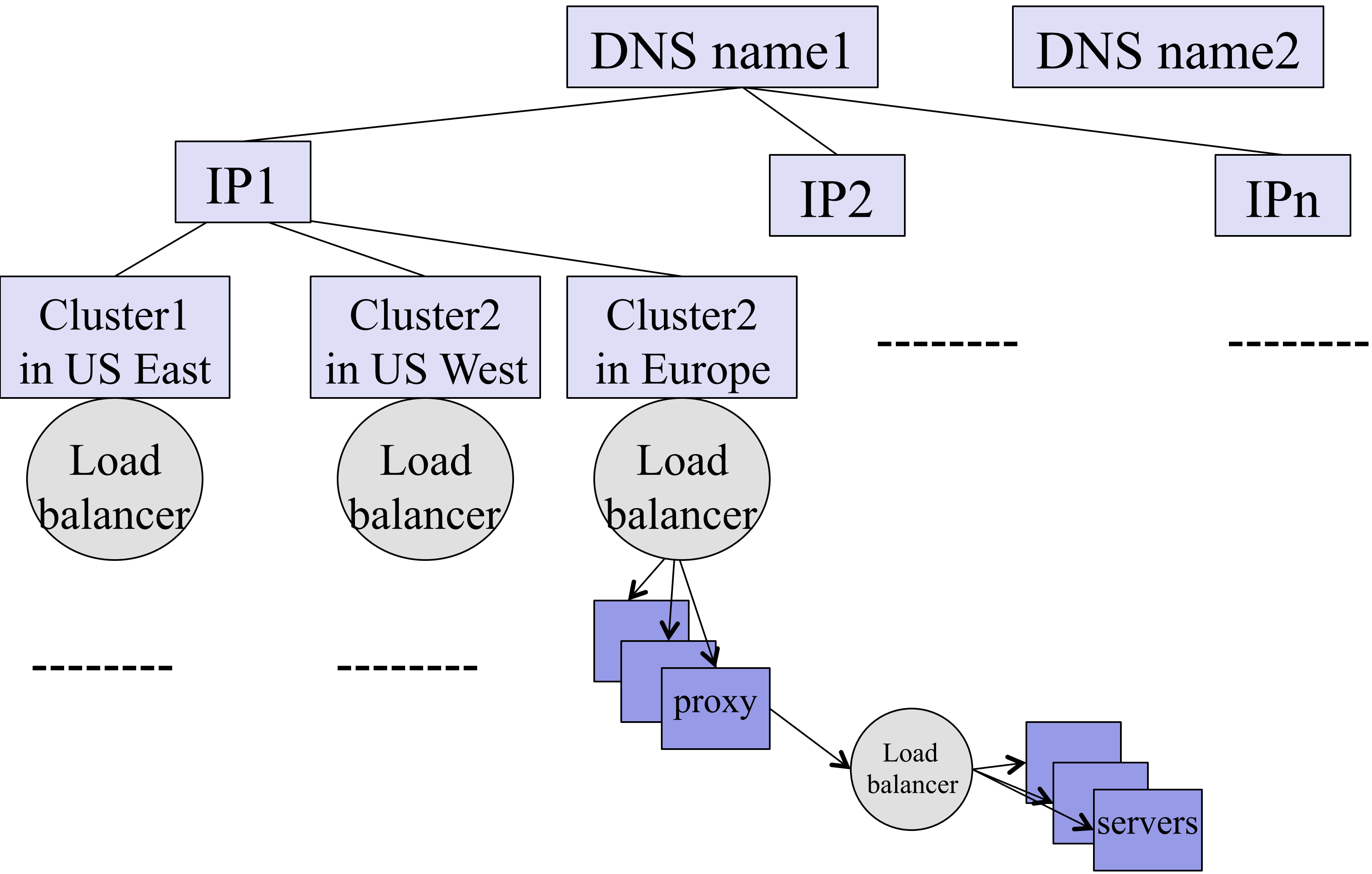


Client Direction Mechanisms

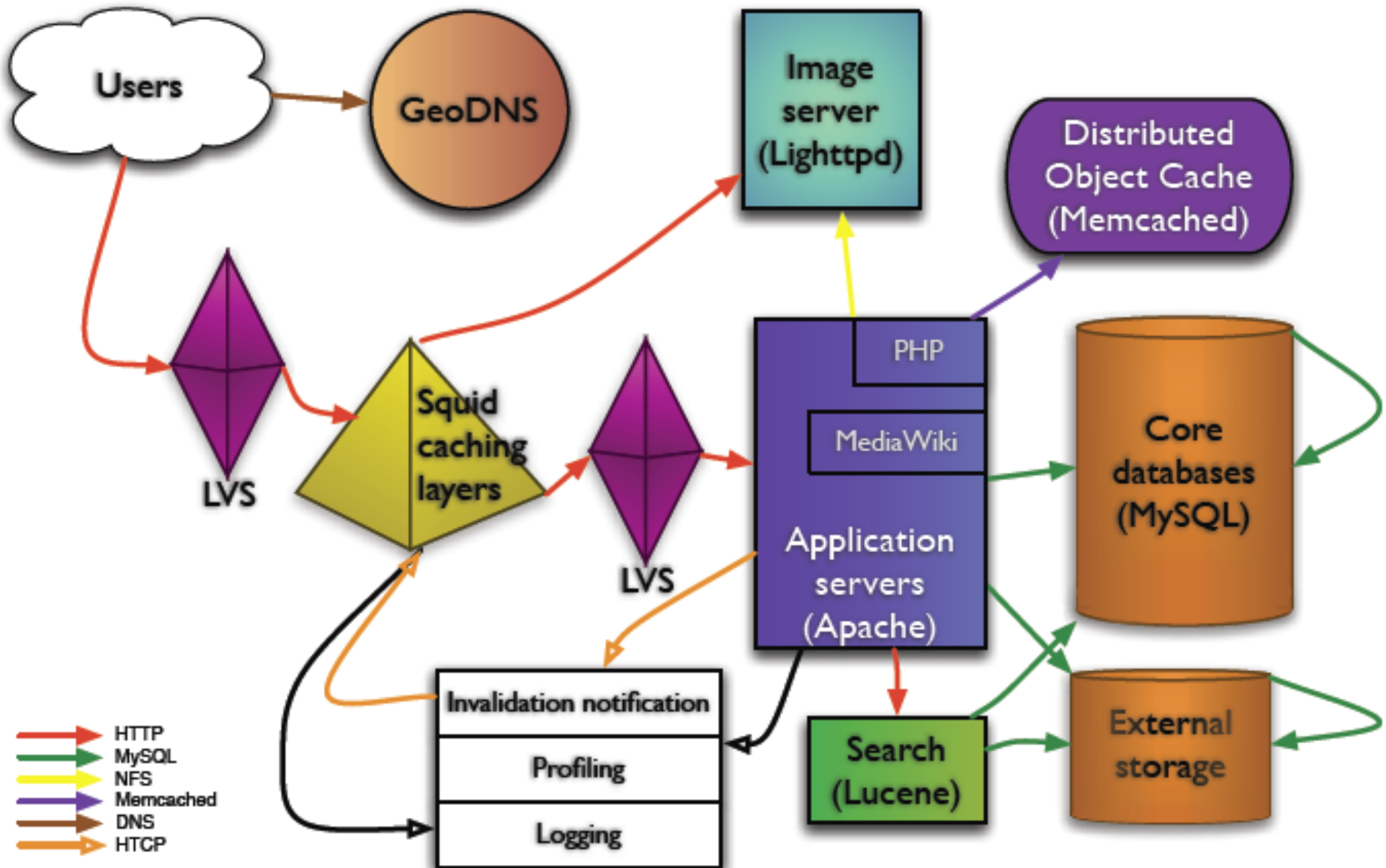


- Application layer
  - App/user is given a list of candidate server names
  - HTTP redirector
- DNS: name resolution gives a list of server addresses
- IP layer: Same IP address represents multiple physical servers
  - IP anycast: Same IP address shared by multiple servers and announced at different parts of the Internet. Network directs different clients to different servers
  - Smart-switch indirection: a server IP address may be a virtual IP address for a cluster of physical servers

Direction Mechanisms are Often Combined



Example: Wikipedia Architecture



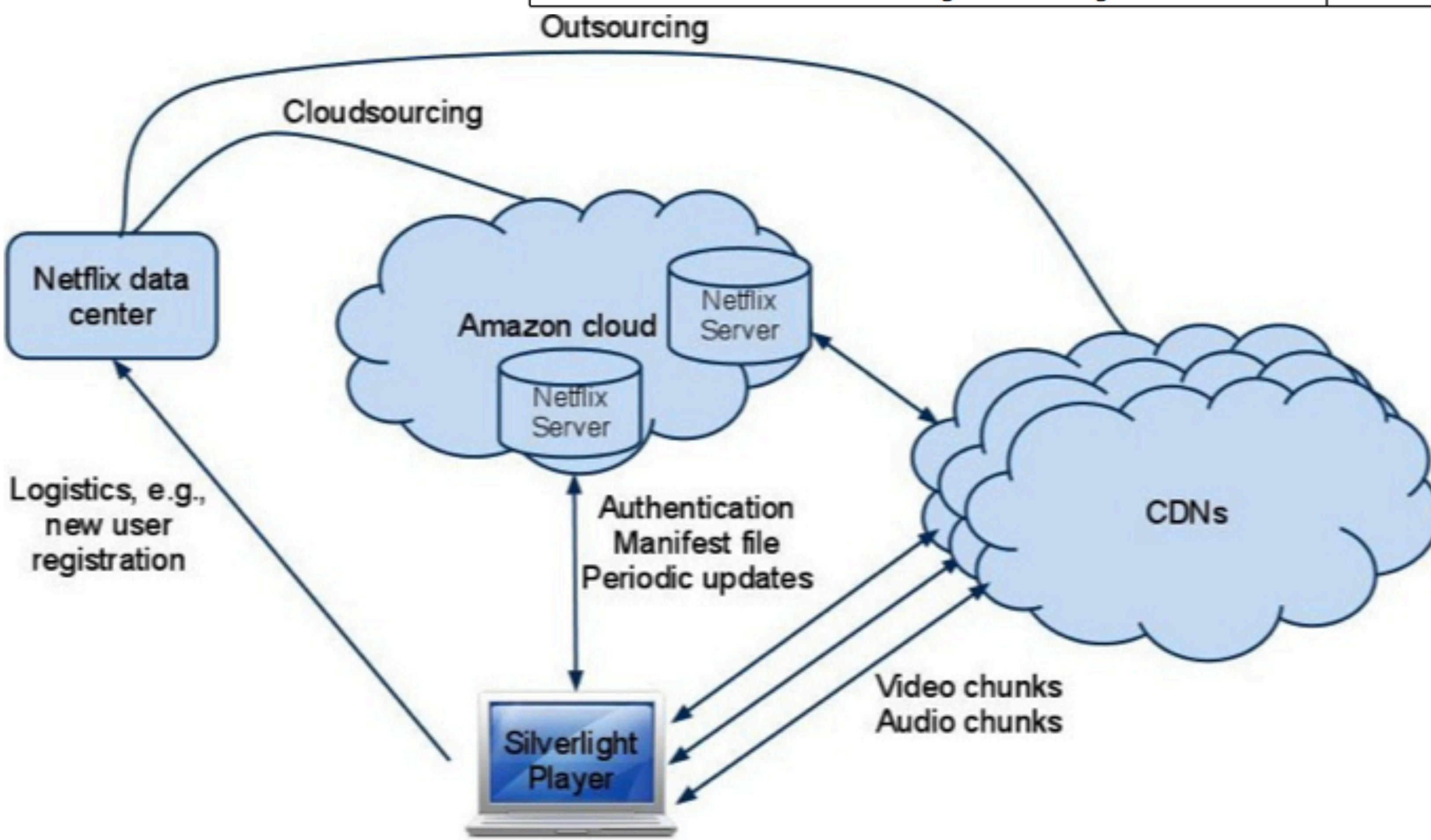
http://wikitech.wikimedia.org/images/8/81/Bergsma_-_Wikimedia_architecture_-_2007.pdf

Outline

- Recap
- Single network server
- Multiple network servers
  - Why multiple servers
  - Request routing mechanisms
    - Overview
    - Application-layer

Example: Netflix

Hostname	Organization
www.netflix.com	Netflix
signup.netflix.com	Amazon
movies.netflix.com	Amazon
agmoviecontrol.netflix.com	Amazon
nflx.i.87f50a04.x.lcdn.nflximg.com	Level 3
netflix-753.vo.llnwd.net	Limelight
netflix753.as.nflximg.com.edgesuite.net	Akamai



Example: Netflix Manifest File

- Client player authenticates and then downloads manifest file from servers at Amazon Cloud

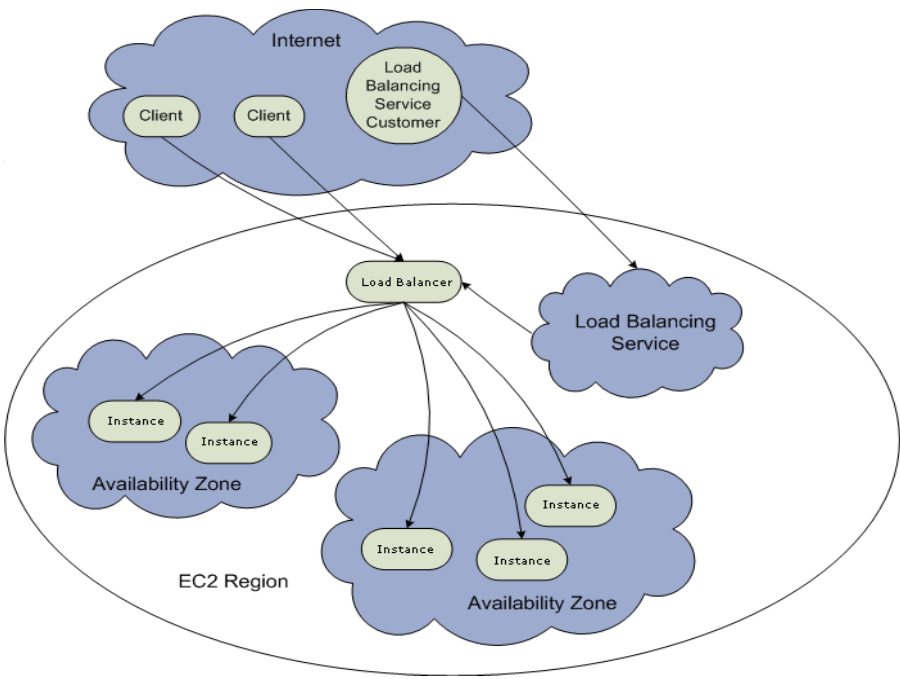

```xml
<nccp:cdns>
    <nccp:cdn>
        <nccp:name>level3</nccp:name>
        <nccp:cdnid>6</nccp:cdnid>
        <nccp:rank>1</nccp:rank>
        <nccp:weight>140</nccp:weight>
    </nccp:cdn>
    <nccp:cdn>
        <nccp:name>limelight</nccp:name>
        <nccp:cdnid>4</nccp:cdnid>
        <nccp:rank>2</nccp:rank>
        <nccp:weight>120</nccp:weight>
    </nccp:cdn>
    <nccp:cdn>
        <nccp:name>akamai</nccp:name>
        <nccp:cdnid>9</nccp:cdnid>
        <nccp:rank>3</nccp:rank>
        <nccp:weight>100</nccp:weight>
    </nccp:cdn>
</nccp:cdns>
```

Example: Netflix Manifest File


```
<nccp:bitrate>560</nccp:bitrate>
<nccp:videoprofile>
   playready-h264mpl30-dash
</nccp:videoprofile>
<nccp:resolution>
   <nccp:width>512</nccp:width>
   <nccp:height>384</nccp:height>
</nccp:resolution>
<nccp:pixelaspect>
   <nccp:width>4</nccp:width>
   <nccp:height>3</nccp:height>
</nccp:pixelaspect>v
<nccp:downloadurls>
   <nccp:downloadurl>
      <nccp:expiration>131xxx</nccp:expiration>
      <nccp:cdnid>6</nccp:cdnid>
      <nccp:url>http://nflx.i.../...</nccp:url>
   </nccp:downloadurl>
   <nccp:downloadurl>
      <nccp:expiration>131xxx</nccp:expiration>
      <nccp:cdnid>4</nccp:cdnid>
      <nccp:url>http://netflix.../...</nccp:url>
   </nccp:downloadurl>
   <nccp:downloadurl>
      <nccp:expiration>131xxx</nccp:expiration>
      <nccp:cdnid>9</nccp:cdnid>
      <nccp:url>http://netflix.../...</nccp:url>
   </nccp:downloadurl>
</nccp:downloadurls>
```

Example: Amazon Elastic Cloud 2 (EC2) Elastic Load Balancing

- Use the *create-load-balancer* command to create an Elastic Load Balancer.
- Use the *register-instances -with-load-balancer* command to regi Amazon EC2 instances that you want to load balance with the Elastic Load Balancer.
- Elastic Load Balancing automatically checks the health of your load balancing Amazon EC2 instances. You can optionally customize the health checks by using the *configure-healthcheck* command.
- Traffic to the DNS name provided by the Elastic Load Balancer is automatically distributed across healthy Amazon EC2 instances.



http://docs.aws.amazon.com/ElasticLoadBalancing/latest/DeveloperGuide/elastic-load-balancing.html

Details: Create Load Balancer

The operation returns the DNS name of your LoadBalancer. You can then map that to any other domain name (such as www.mywebsite.com)
(how?)

```
%aws elb create-load-balancer --load-balancer-name my-load-balancer --listeners "Protocol=HTTP,LoadBalancerPort=80,InstanceProtocol=HTTP,InstancePort=80" --availability-zones us-west-2a us-west-2b
```

```
Result:
```

{ "DNSName": "my-load-balancer-123456789.us-west-2.elb.amazonaws.com”}

http://docs.aws.amazon.com/cli/latest/reference/elb/create-load-balancer.html

Details: Configure Health Check

The operation configures how instances are monitored, e.g.,

```
%aws elb configure-health-check --load-balancer-name my-load-balancer --health-check Target=HTTP:80/png,Interval=30,UnhealthyThreshold=2,HealthyThreshold=2,Timeout=3
```

Result:


```
{
   "HealthCheck": {
        "HealthyThreshold": 2,
        "Interval": 30,
        "Target": "HTTP:80/png",
        "Timeout": 3,
        "UnhealthyThreshold": 2
 }
```


http://docs.aws.amazon.com/cli/latest/reference/elb/configure-health-check.html

Details: Register Instances

The operation registers instances that can receive traffic,

```
%aws elb register-instances-with-load-
balancer --load-balancer-name my-load-
balancer --instances i-d6f6fae3
```

Result:

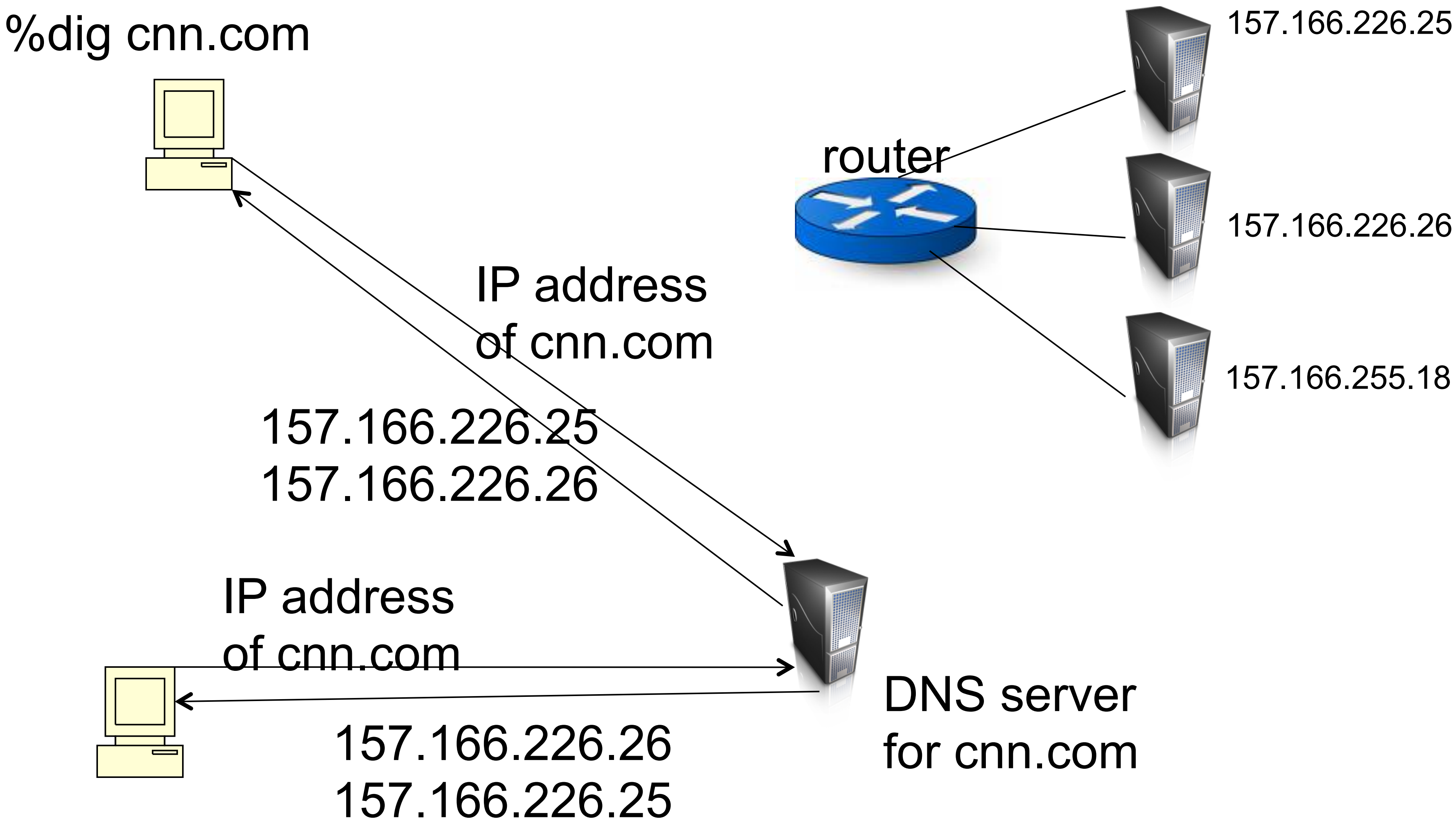

```
{  "Instances": [
     {"InstanceId": "i-d6f6fae3“},
     {"InstanceId": "i-207d9717“},
     {"InstanceId": "i-afefb49b“}
   ]
}
```


http://docs.aws.amazon.com/cli/latest/reference/elb/register-instances-with-load-balancer.html

Outline

- Recap
- Single network server
- Multiple network servers
  - Why multiple servers
  - Request routing mechanisms
    - Overview
    - Application-layer
    - DNS

Basic DNS Indirection and Rotation



CDN Using DNS (Akamai Architecture as an Example)

- Content publisher (e.g., NYTimes)
  - provides base HTML documents
  - runs origin server(s)

- Akamai runs
  - (~200,000) edge servers for hosting content
    - Deployment into 110 countries and 1400 networks
  - customized DNS redirection servers to select edge servers based on
    - closeness to client browser
    - server load

Source: https://www.akamai.com/us/en/about/facts-figures.jsp

Linking to Akamai

- Originally, URL Akamaization of embedded content: e.g.,

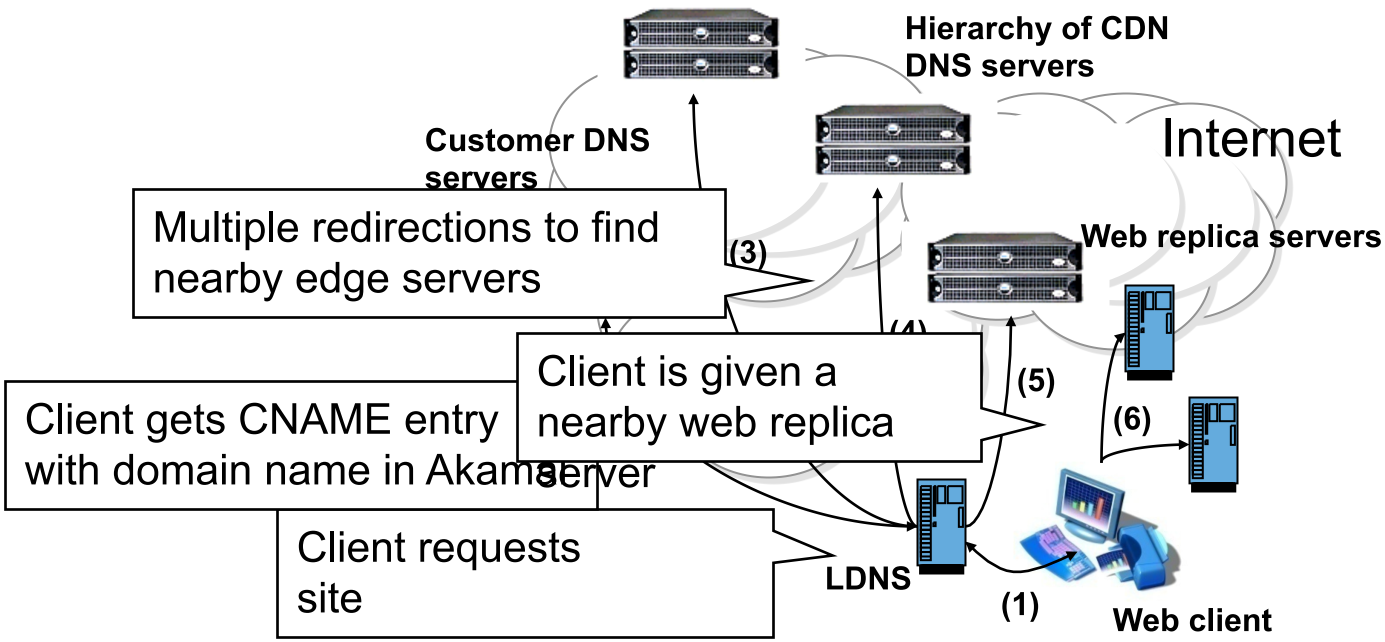
  <IMG SRC= http://www.provider.com/image.gif >
  changed to

  <IMGSRC = http://a661. g.akamai.net/hash/image.gif>

  Note that this DNS redirection unit is per customer, not individual files.

- URL Akamaization is becoming obsolete and supported mostly for legacy reasons
  - Currently most content publishers prefer to use DNS CNAME to link to Akamai servers

Akamai Load Direction Flow



More details see “Global hosting system”: FT Leighton, DM Lewin – US Patent 6,108,703, 2000.

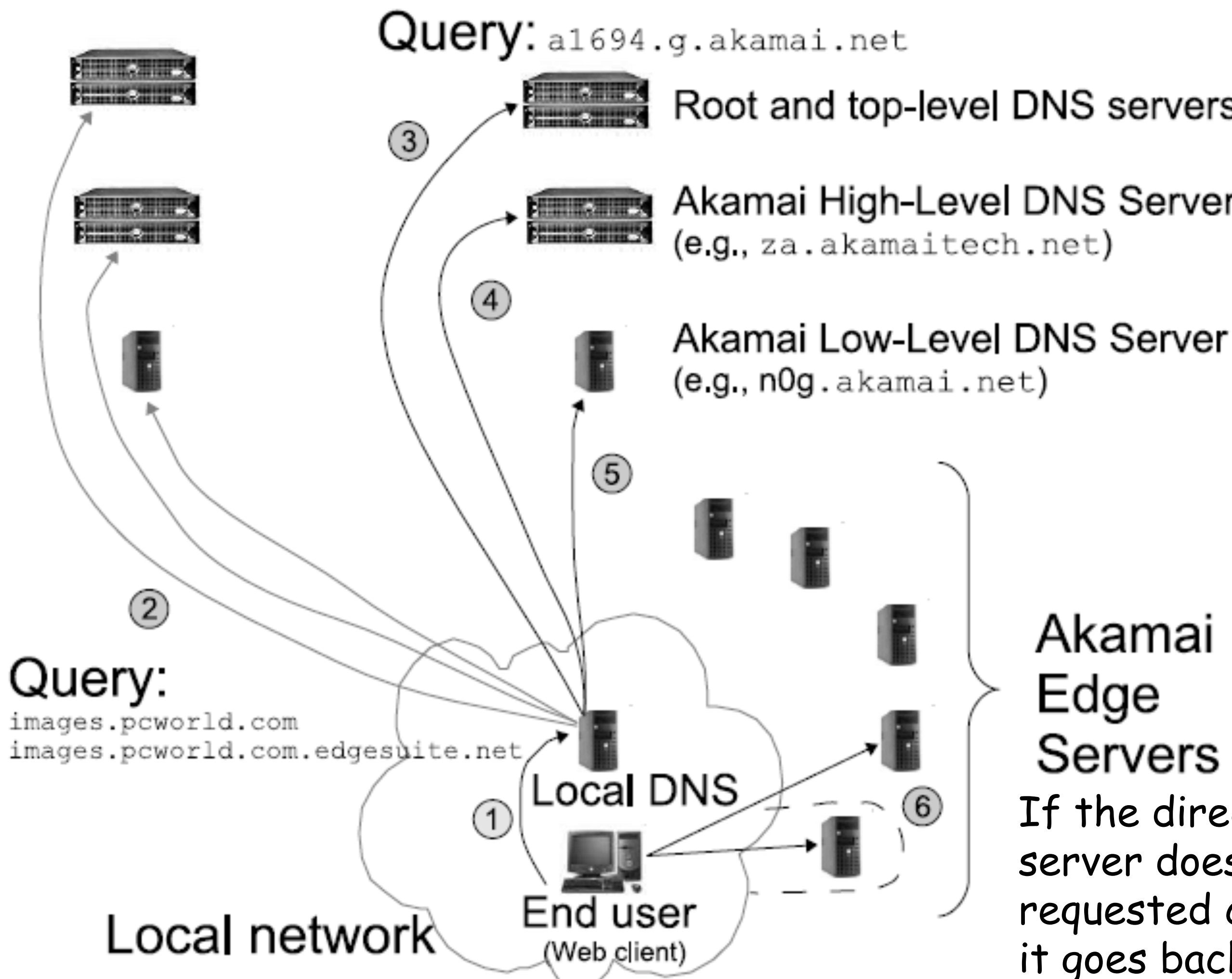
Exercise: Zoo machine

- Check any web page of New York Times and find a page with an image
- Find the URL
- Use

  ```
  %dig +trace +recurse
  ```

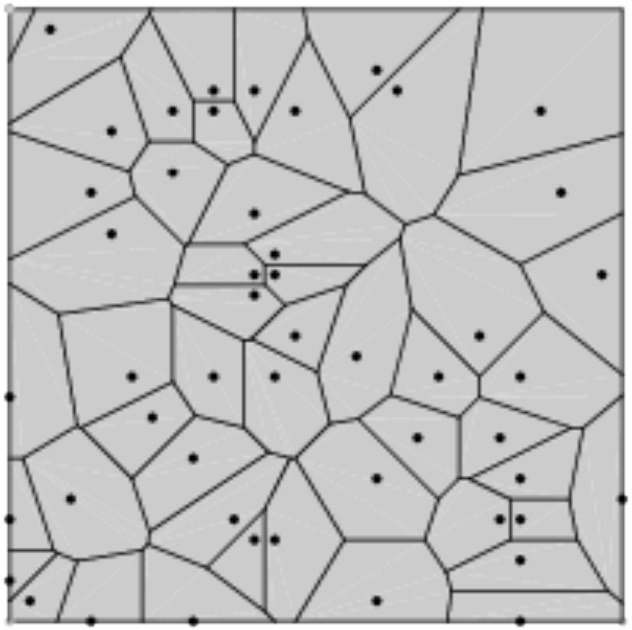
  to see Akamai load direction

Akamai Load Direction



If the directed edge server does not have requested content, it goes back to the original server (source).

Two-Level Direction



proximity: high-level DNS determines client location; directs to low-level DNS, who manages a close-by cluster

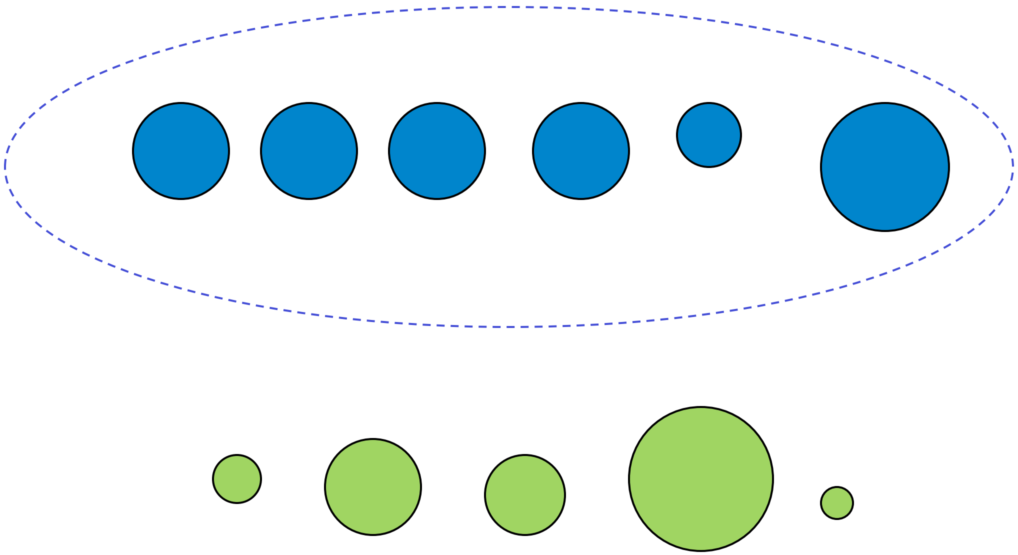
Local DNS Alg: Potential Input

- p(m, e): path properties (from a client site m to an edge sever e)
  - Akamai might use a one-hop detour routing (see akamai-detour.pdf)
- a^k_m: request load from client site m to publisher k
- x_e: load on edge server e
- caching state of a server e

Local DNS Alg

- Details of Akamai algorithms are proprietary
- A Bin-Packing algorithm (column 12 of Akamai Patent) every T second
  - Compute the load to each publisher k (called serial number)
  - Sort the publishers from increasing load
  - For each publisher, associate a list of random servers generated by a hash function
  - Assign the publisher to the first server that does not overload

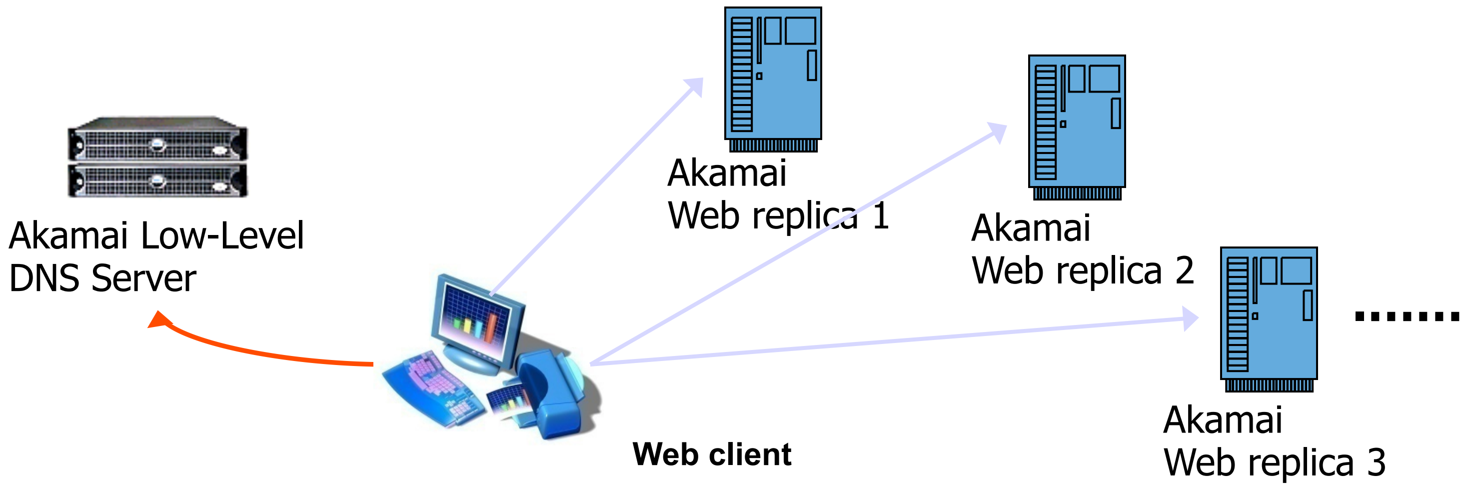
Hash Bin-Packing



LB: maps request to individual machines inside cluster

Experimental Study of Akamai Load Balancing

- Methodology
  - 2-months long measurement
  - 140 PlanetLab nodes (clients)
    - 50 US and Canada, 35 Europe, 18 Asia, 8 South America, the rest randomly scattered
  - Every 20 sec, each client queries an appropriate CNAME for Yahoo, CNN, Fox News, NY Times, etc.

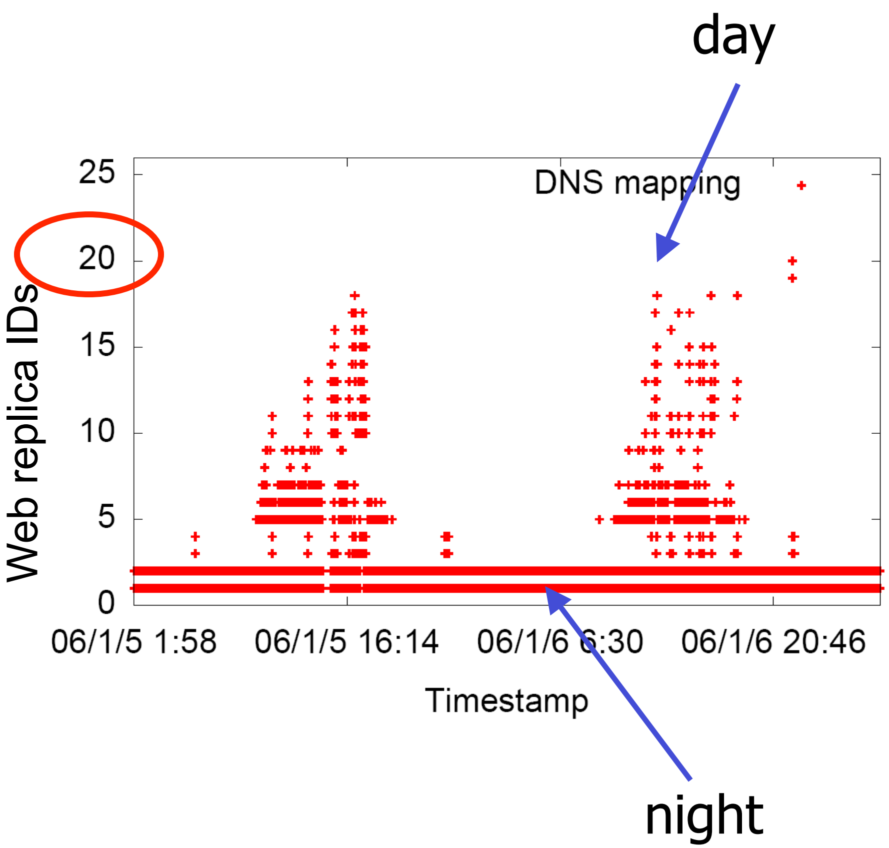


See http://www.aqualab.cs.northwestern.edu/publications/Ajsu06DBA.pdf

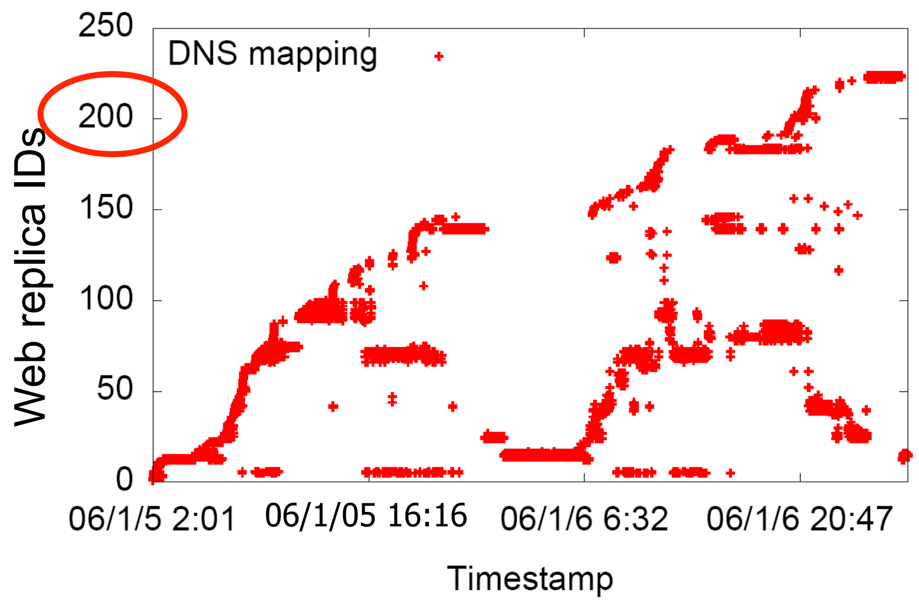
Server Pool: to Yahoo

Target: a943.x.a.yimg.com (Yahoo)

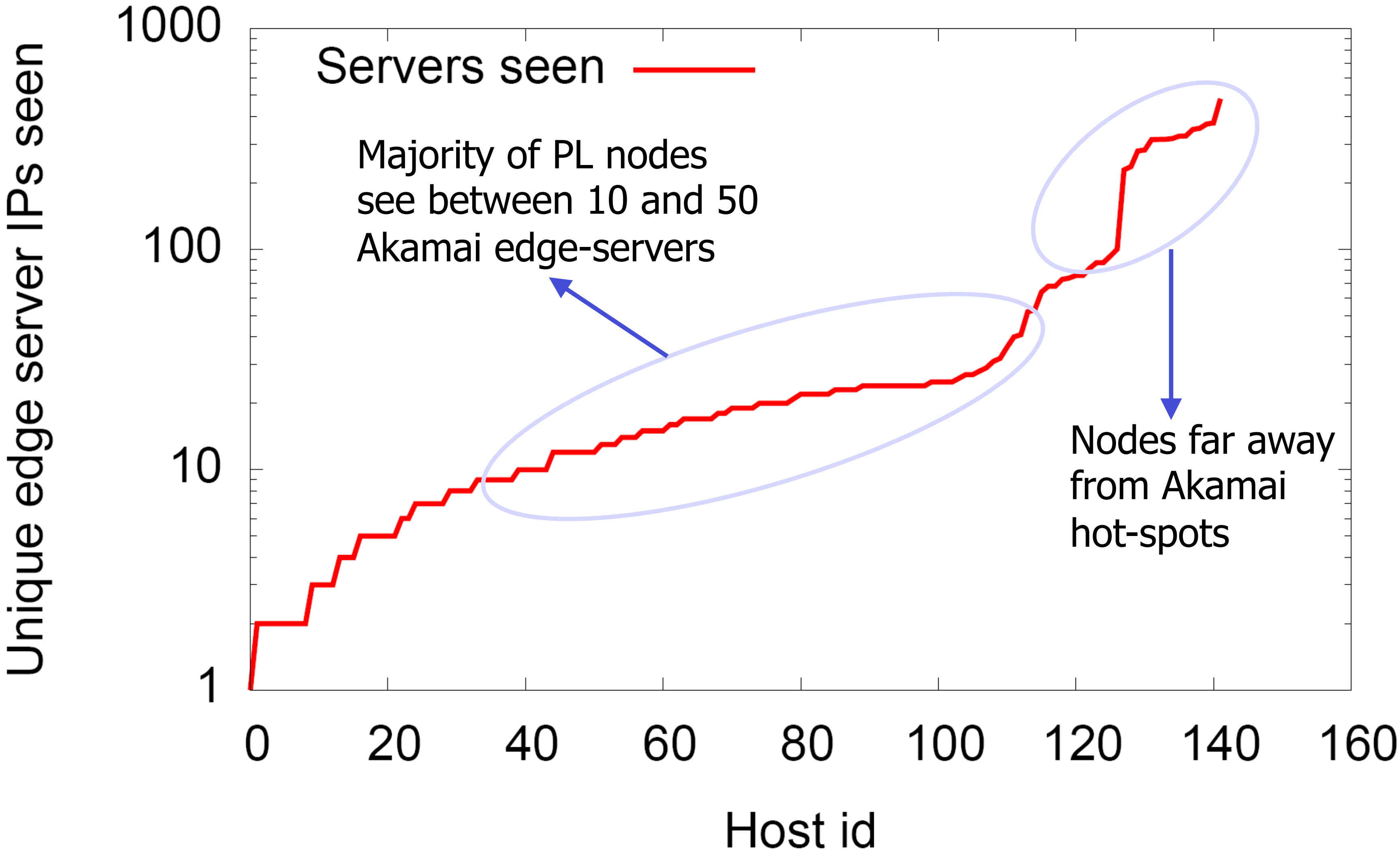
Client 1: Berkeley



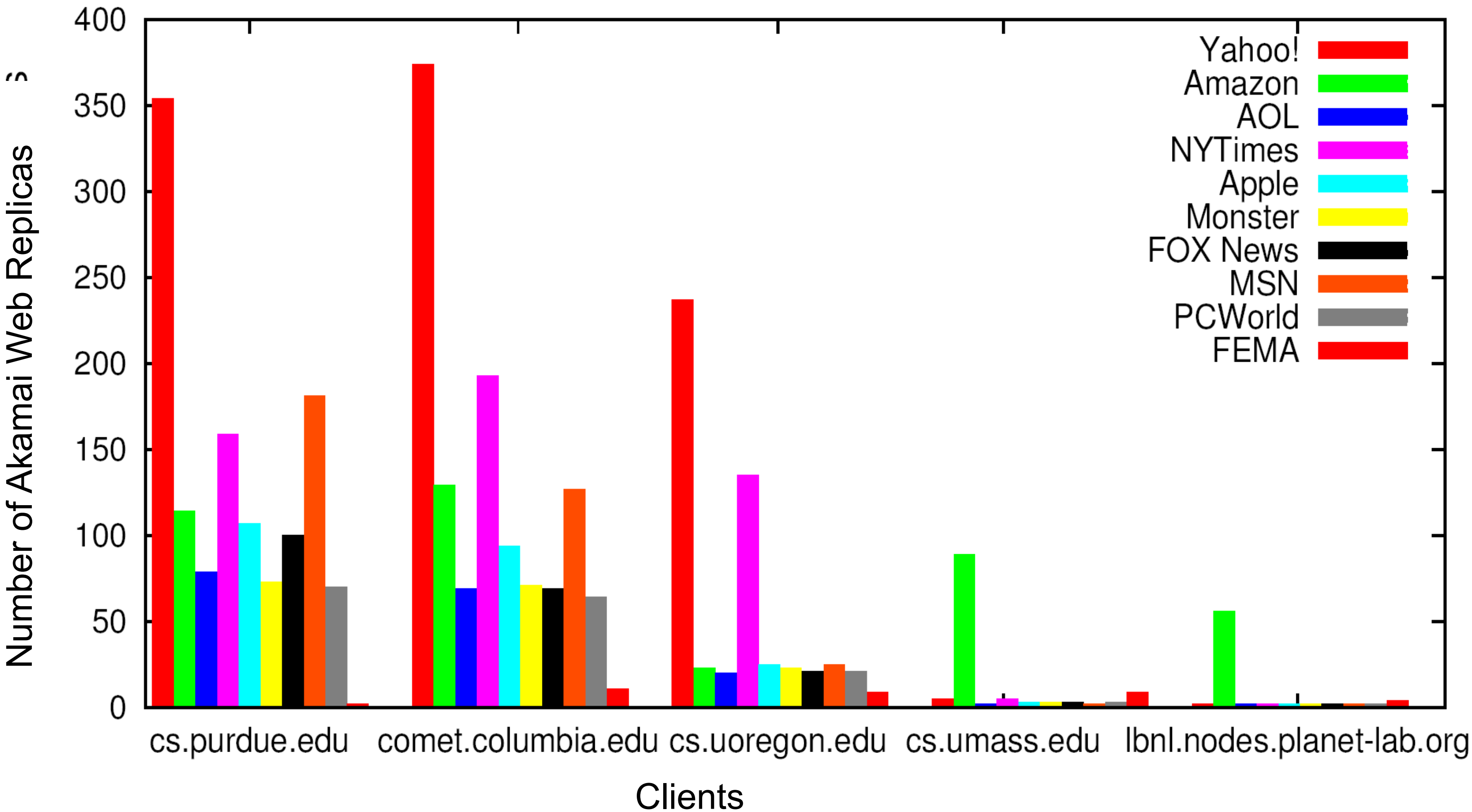
Client 2: Purdue



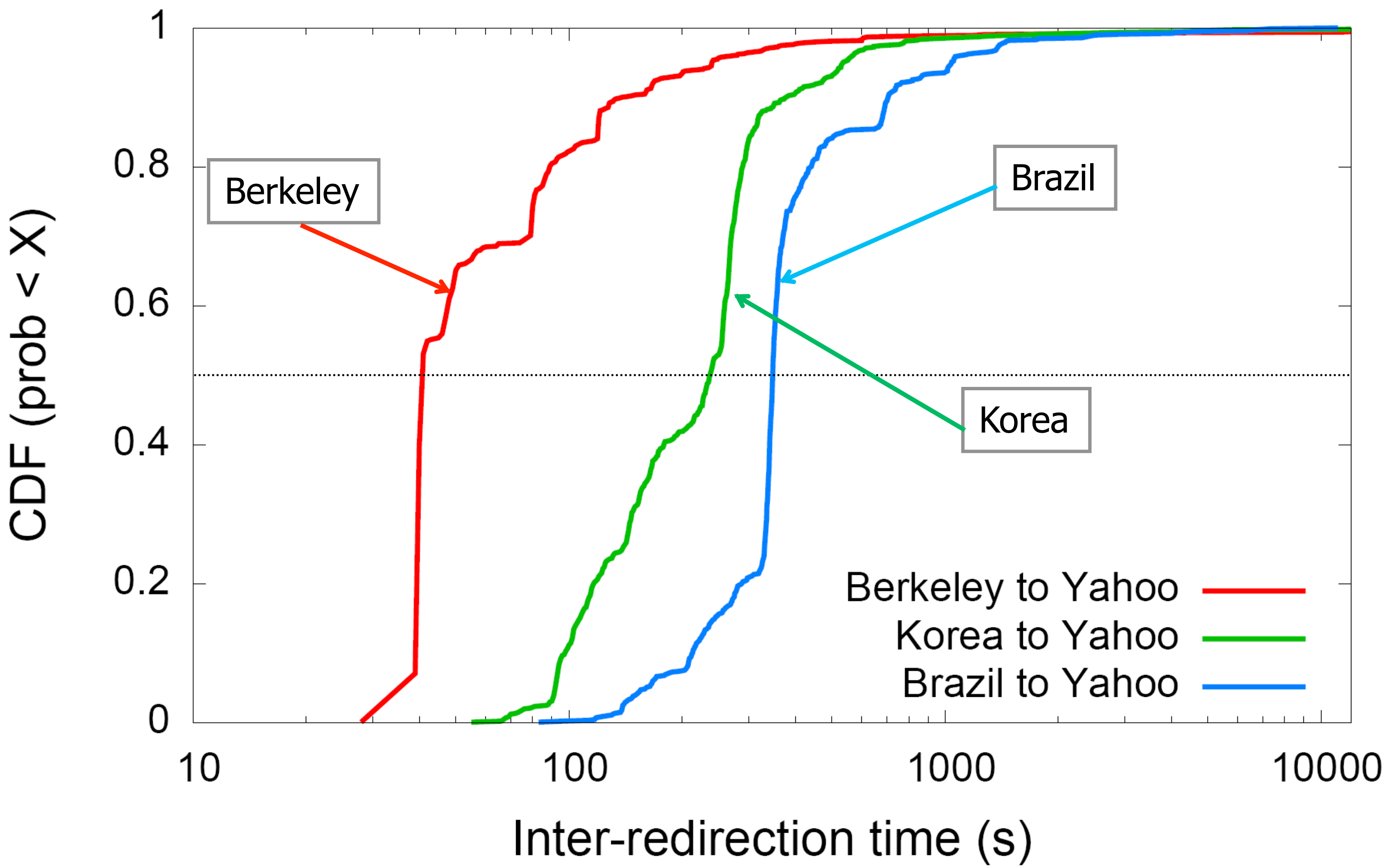
Server Diversity for Yahoo



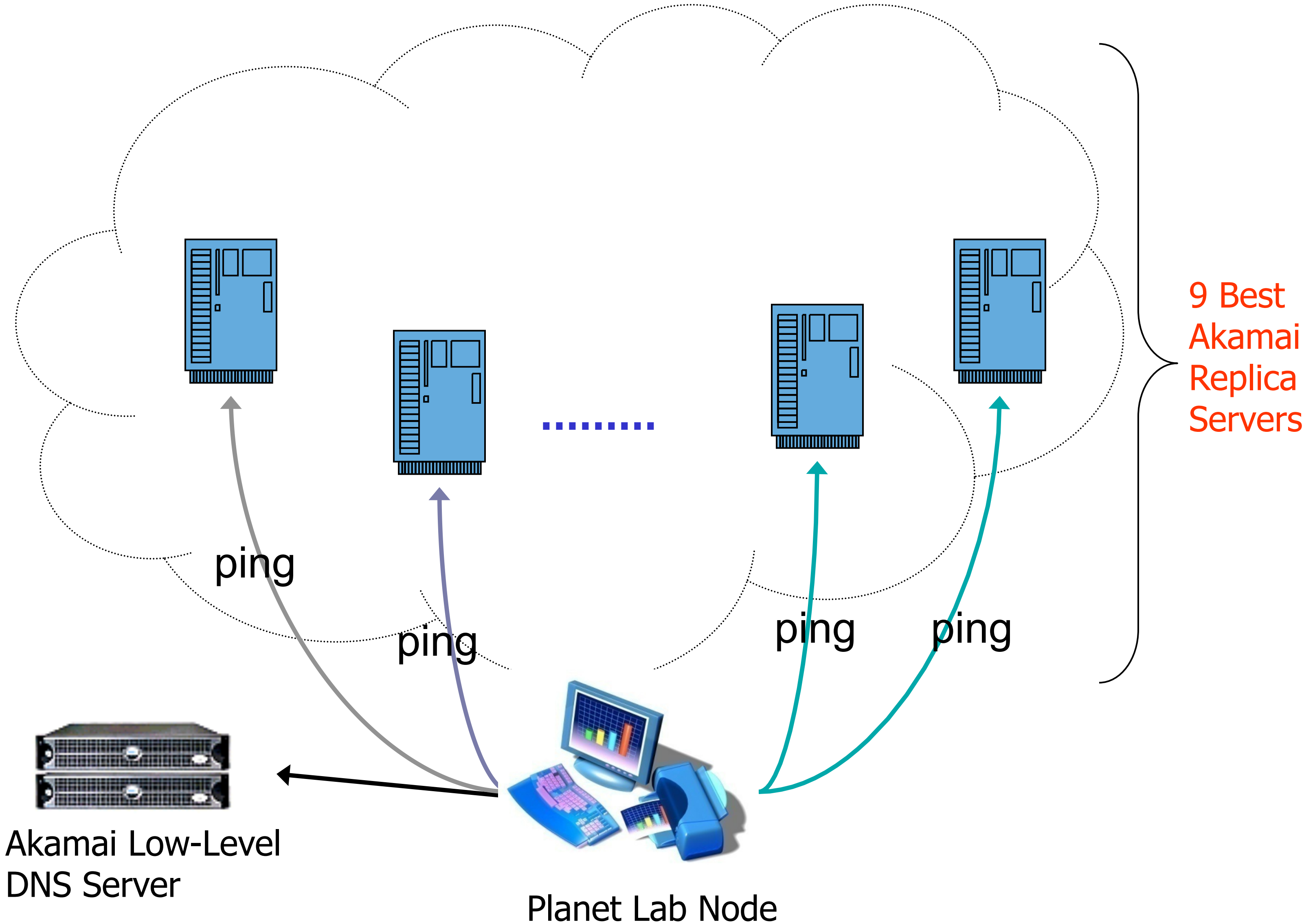
Server Pool: Multiple Akamai Hosted Sites



Load Balancing Dynamics

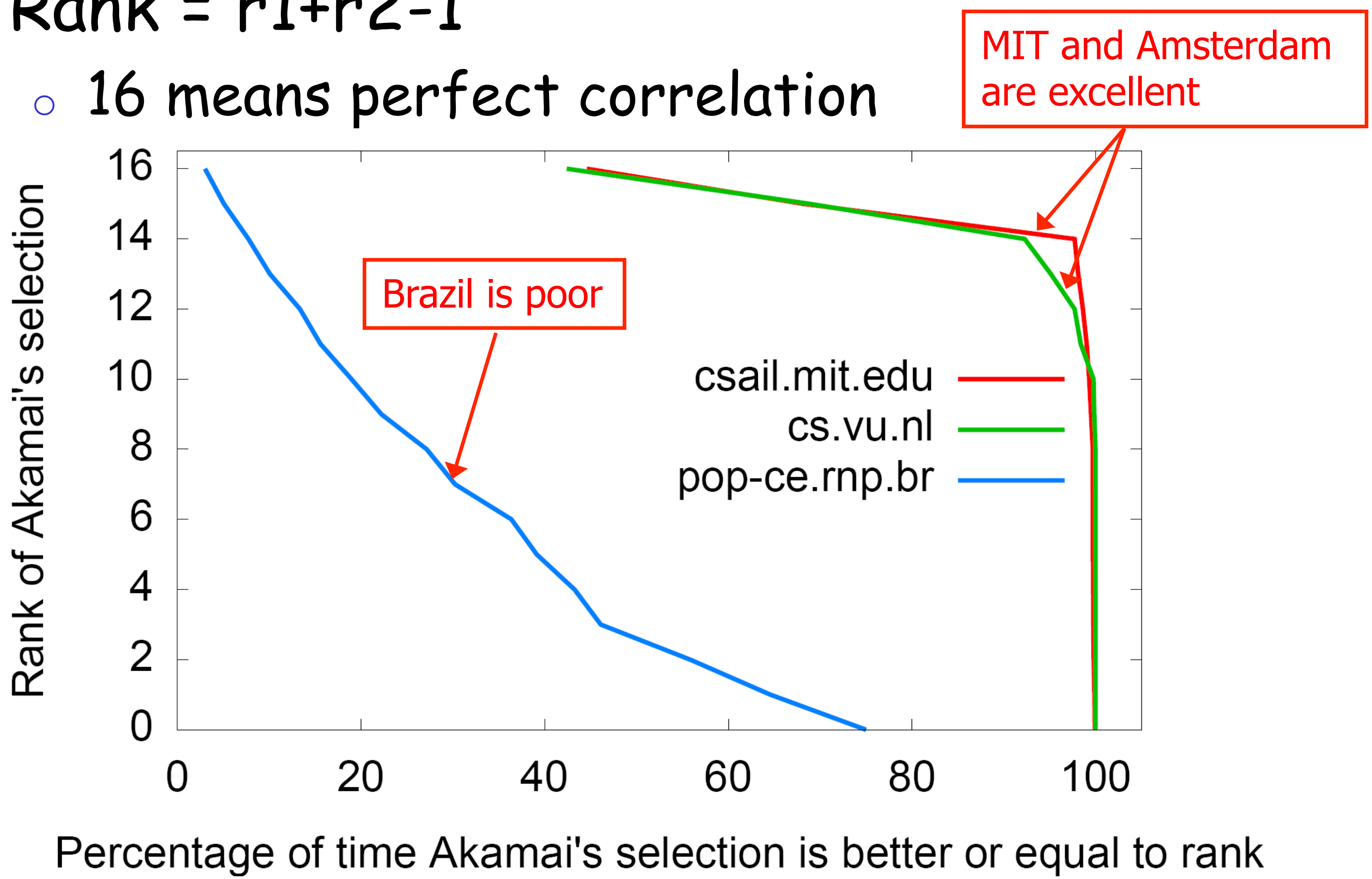


Redirection Effectiveness: Measurement Methodology



Do redirections reveal network conditions?

- Rank = r1+r2-1
  - 16 means perfect correlation



(Offline Read) Facebook DNS Load Direction

- A system named Cartographer (written in Python) processes measurement data and configures the DNS maps of individual DNS servers (open source tinydns)

Discussion

- Advantages and disadvantages of using DNS