
Network Applications: Operational Analysis; Load Balancing among Homogeneous Servers

Qiao Xiang

<https://qiaoxiang.me/courses/cnns-xmuf21/index.shtml>

11/2/2021

Outline

- Admin and recap
- High performance servers
 - Thread design
 - Asynchronous design
 - Operational analysis
- Multi-servers
- Application overlays (peer-to-peer networks)

Admin

- Lab assignment 3 status and questions
 - Move the due date from Nov. 11 to Nov. 14?

Future

```
SocketAddress address
    = new InetSocketAddress(args[0], port);
AsynchronousSocketChannel client
    = AsynchronousSocketChannel.open();
Future<Void> connected
    = client.connect(address);

ByteBuffer buffer = ByteBuffer.allocate(100);

// wait for the connection to finish
connected.get();

// read from the connection
Future<Integer> future = client.read(buffer);

// do other things...

// wait for the read to finish...
future.get();

// flip and drain the buffer
buffer.flip();
WritableByteChannel out
    = Channels.newChannel(System.out);
out.write(buffer);
```

CompletionHandler

```
class LineHandler implements
CompletionHandler<Integer, ByteBuffer> {

    @Override
    public void completed(Integer result, ByteBuffer buffer)
    {
        buffer.flip();
        WritableByteChannel out
            = Channels.newChannel(System.out);
        try {
            out.write(buffer);
        } catch (IOException ex) {
            System.err.println(ex);
        }
    }

    @Override
    public void failed(Throwable ex,
                      ByteBuffer attachment) {
        System.err.println(ex.getMessage());
    }
}

ByteBuffer buffer = ByteBuffer.allocate(100);
CompletionHandler<Integer, ByteBuffer>
    handler = new LineHandler();
channel.read(buffer, buffer, handler);
```

Recap: Operational Quantities

- T : observation interval A_i : # arrivals to device i
- B_i : busy time of device i C_i : # completions at device i
- $i = 0$ denotes system

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

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

$$\text{Utilization } U_i = \frac{B_i}{T}$$

$$\text{Mean service time } S_i = \frac{B_i}{C_i}$$

Recap: Operational Analysis— Fundamental Laws

- Utilization law: $U_i = X_i S_i$
- Forced flow law: $X_i = V_i X$
- Bottleneck device: Largest $D_i = V_i S_i$
- Little's Law: $Q_i = X_i R_i$

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:

- $D_{CPU} = 10 \text{ ms}$ (e.g. 100 requests/s)

□ Network:

- $D_{Net} = 1 \text{ Mbytes} / 100 \text{ Mbps} = 80 \text{ ms}$ (e.g., 12.5 requests/s)

□ Disk I/O:

- $D_{disk} = 0.5 * 1 \text{ ms} * 1M/8K = 62.5 \text{ ms}$
(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 $X = \text{Total\# req./T}$

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

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

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

Bottleneck Analysis

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

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

- Here D is the sum of D_i

Proof

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

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

□ We know

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

$$R(N) \geq D$$

Using interactive response time law:

$$R = \frac{N}{X} - Z \quad \longrightarrow \quad R \geq ND_{\max} - Z$$

$$X = \frac{N}{R+Z} \quad \longrightarrow \quad X \leq \frac{N}{D+Z}$$

Summary: Operational Laws

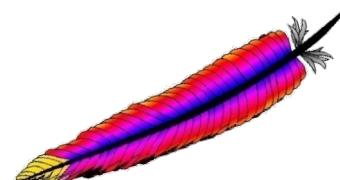
- Utilization law: $U = XS$
- Forced flow law: $X_i = V_i X$
- Bottleneck device: largest $D_i = V_i S_i$
- Little's Law: $Q_i = X_i R_i$
- Bottleneck bound of interactive response
(for the given closed model):

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

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

In Practice: Common Bottlenecks

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



[Aaron Bannert]

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; asynchronous channel/Future/Handler
- Extensibility/robustness
 - Language support/Design for interfaces
- System modeling and measurements
 - Queueing analysis, operational analysis

Outline

- Admin and recap
- Single, high-performance network server
- Multiple network servers
 - *Why multiple network servers*

Why Multiple Servers?

- Scalability
 - Scaling beyond single server throughput
 - There is a fundamental limit on what a single server can
 - process (CPU/bw/disk throughput)
 - store (disk/memory)
 - Scaling beyond single geo location latency
 - 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)
- Today we focus mostly on the first benefit, for homogeneous (replica) servers

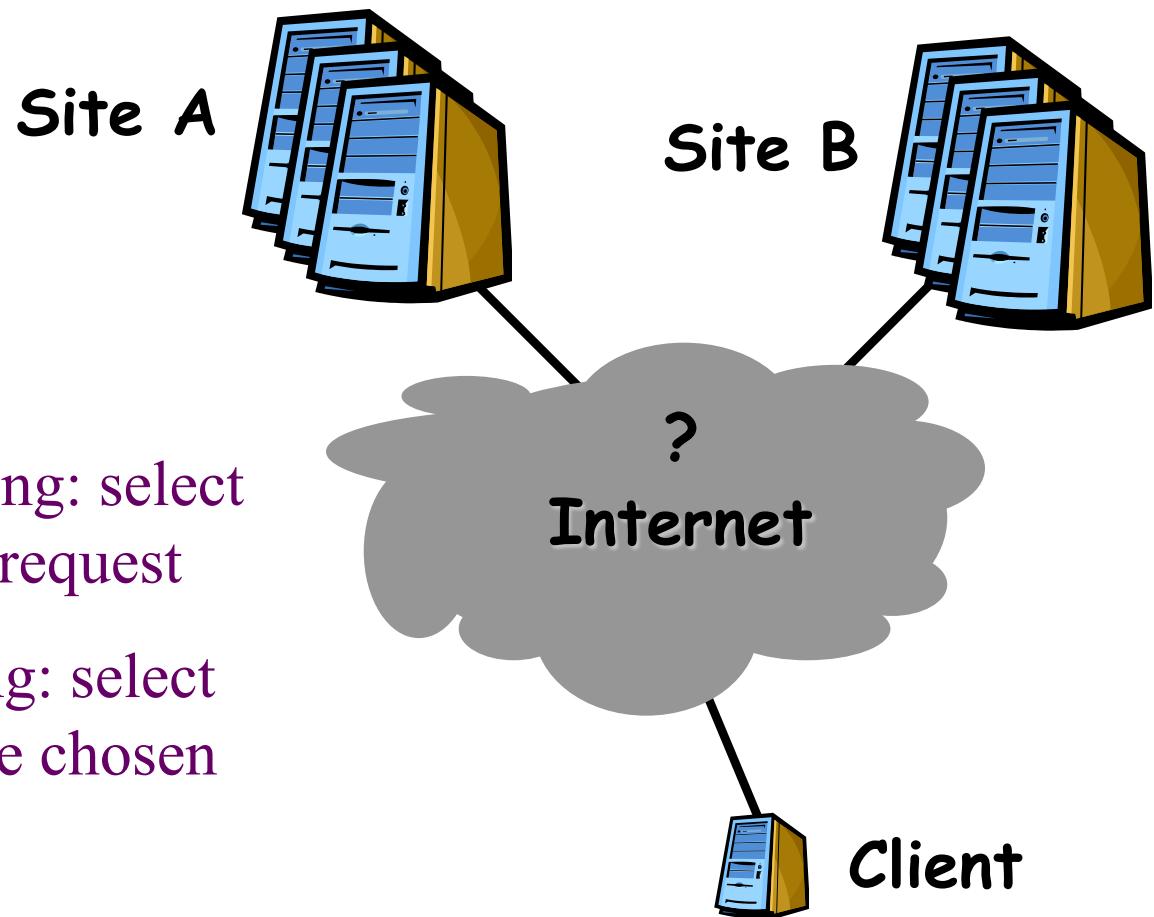
Discussion: Key Technical Challenges in Using Multiple Servers

Outline

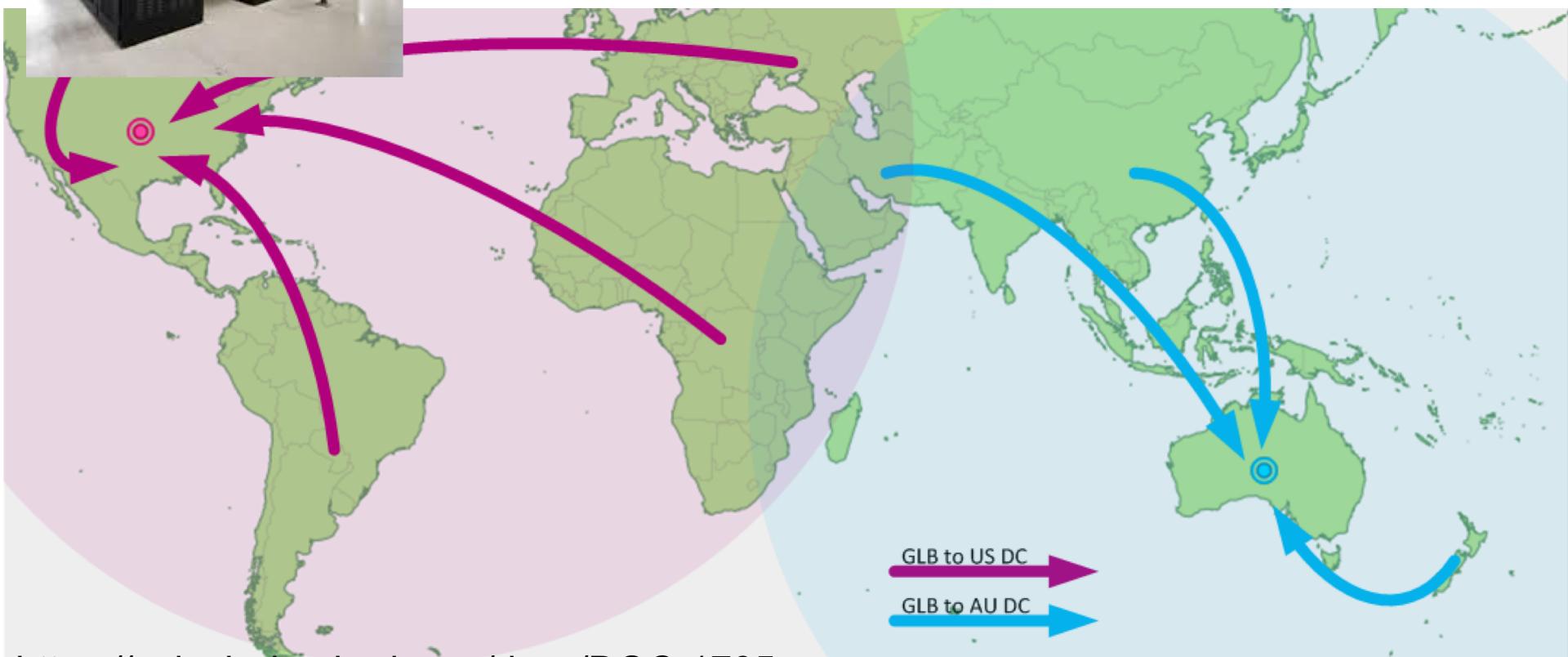
- Recap
- Single network server
- Multiple network servers
 - Why multiple servers
 - *Multiple servers basic mechanism: request routing*

Request Routing: Overview

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



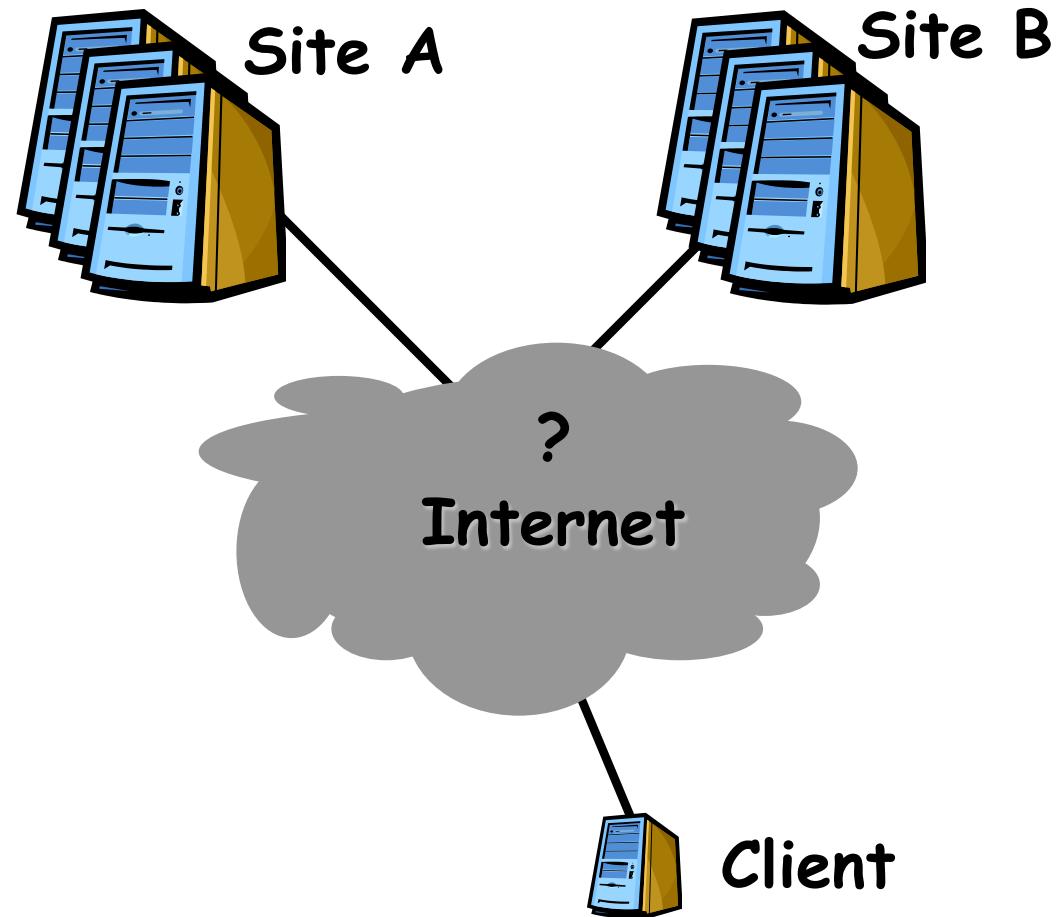
Request Routing: Overview



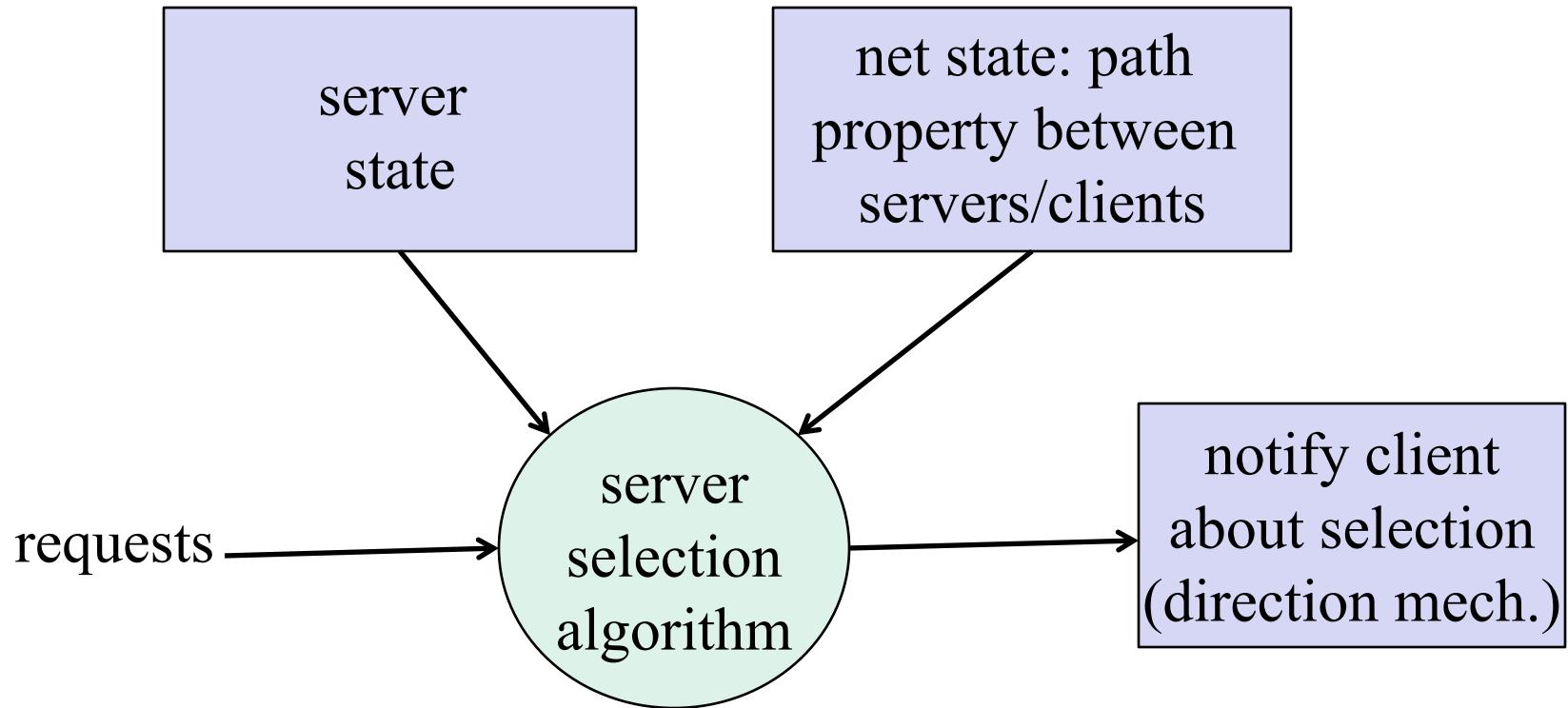
Request Routing: Basic Architecture

□ Major components

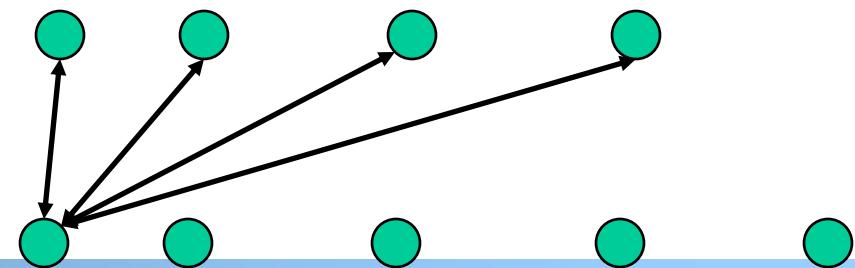
- Server state monitoring
 - Load (incl. failed or not); what requests it can serve
- Network path properties estimation
 - E.g., bw, delay, loss, network cost between clients and servers
- Server assignment alg.
- Request direction mechanism



Request Routing: Basic Architecture

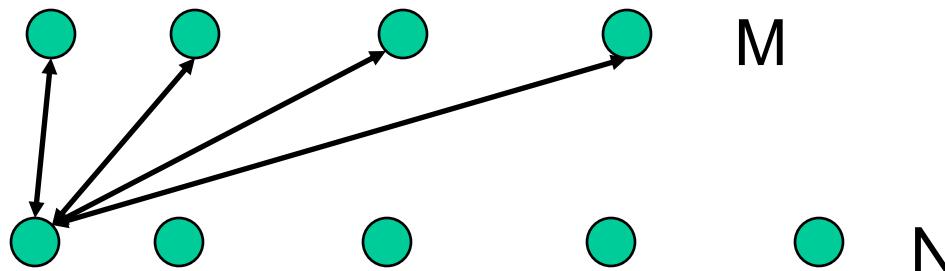


Network Path Properties



□ Why is the problem difficult?

- Scalability: if do measurements, complete measurements grow with $N * M$, where
 - N is # of clients
 - M is # of servers

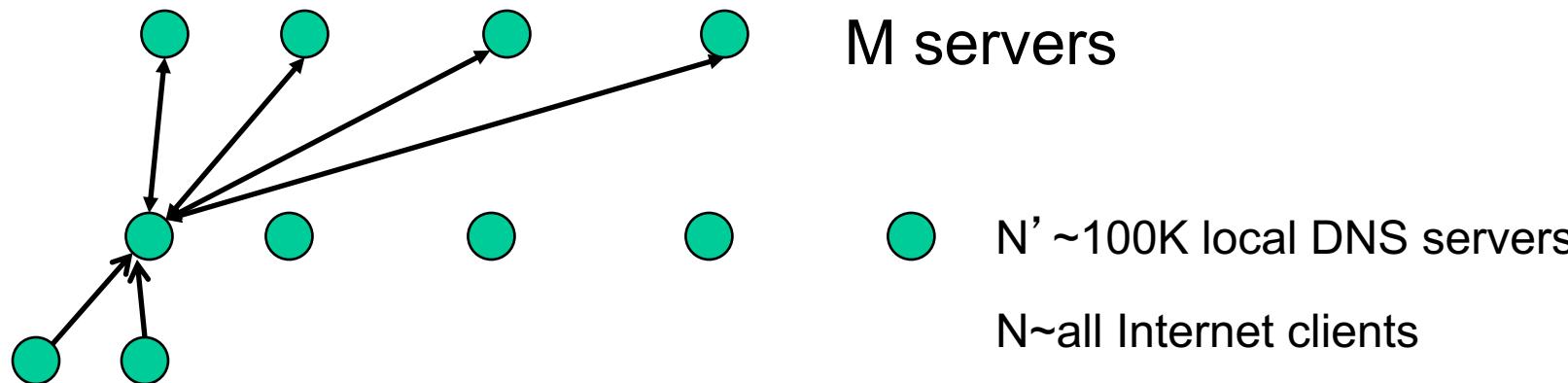


- Complexity/feasibility in computing path metrics

Network Path Properties: Improve Scalability

❑ Aggregation:

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

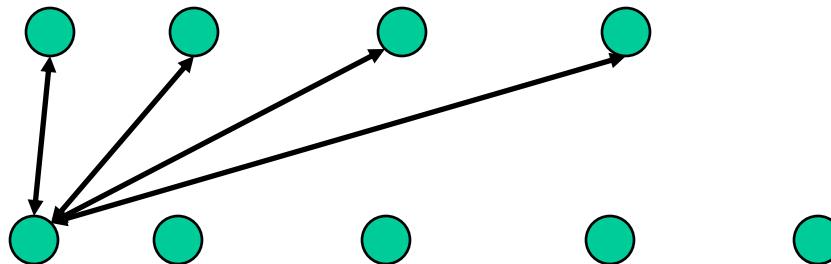


❑ Sampling and prediction

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

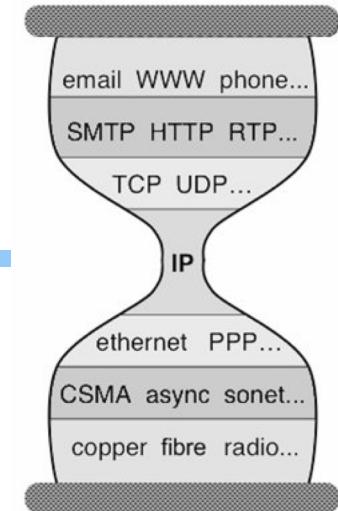
Server Assignment

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

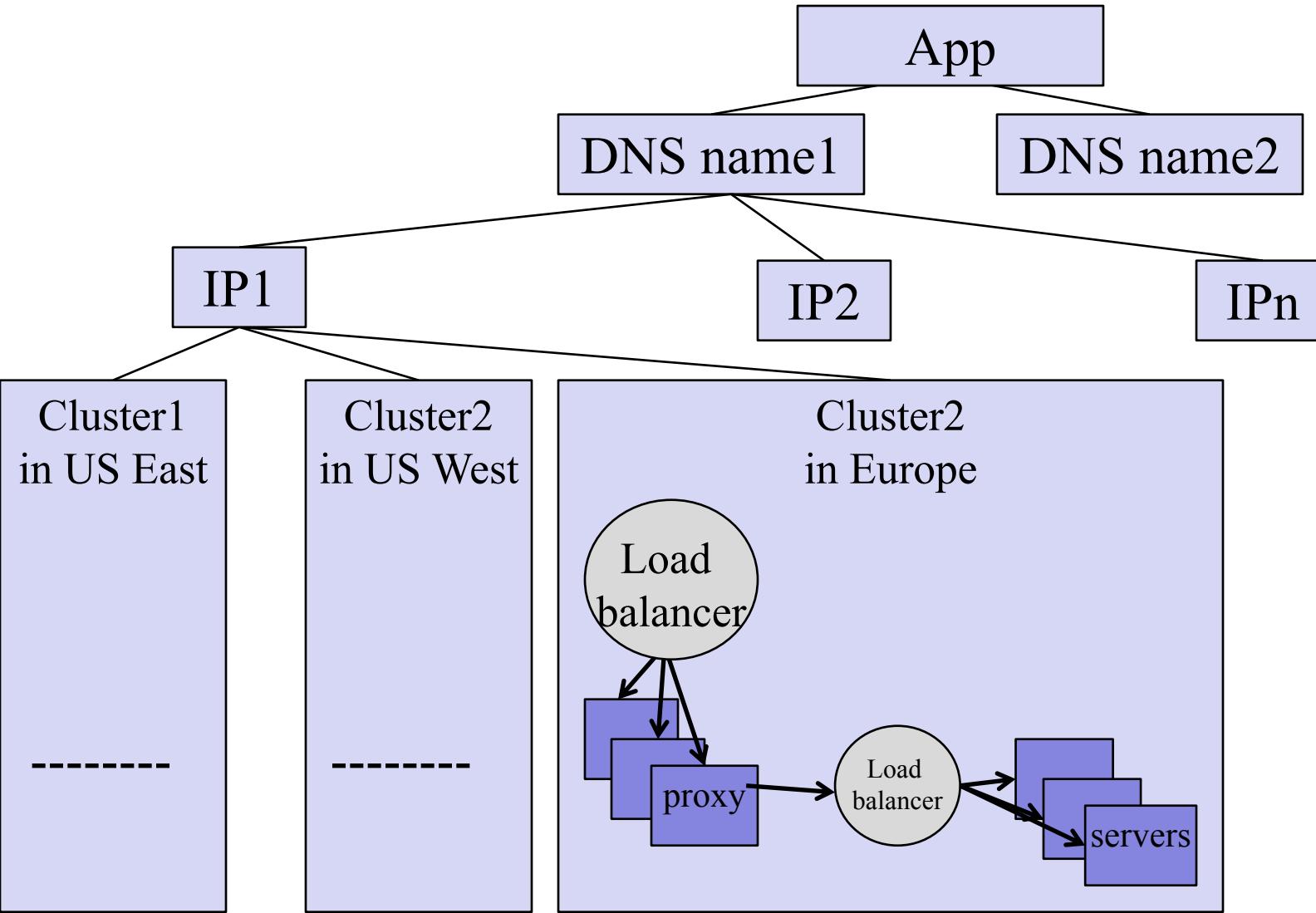


Client Direction Mechanisms

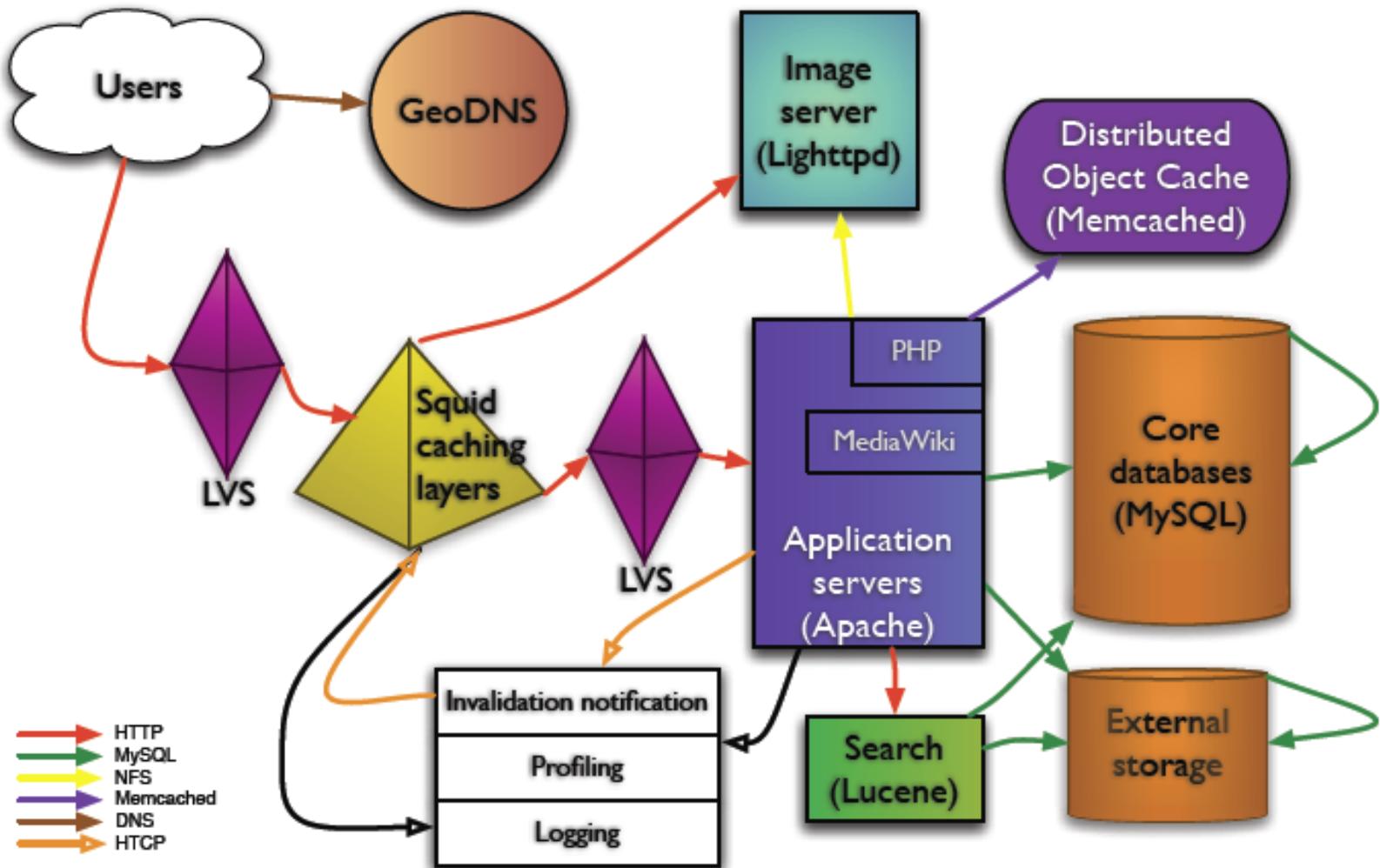
- Key difficulty
 - May need to handle a large of clients
- Basic types of mechanisms
 - Application layer, e.g.,
 - App/user is given a list of candidate server names
 - HTTP redirector
 - DNS: name resolution gives a list of server addresses
 - IP layer: Same IP address represents multiple physical 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



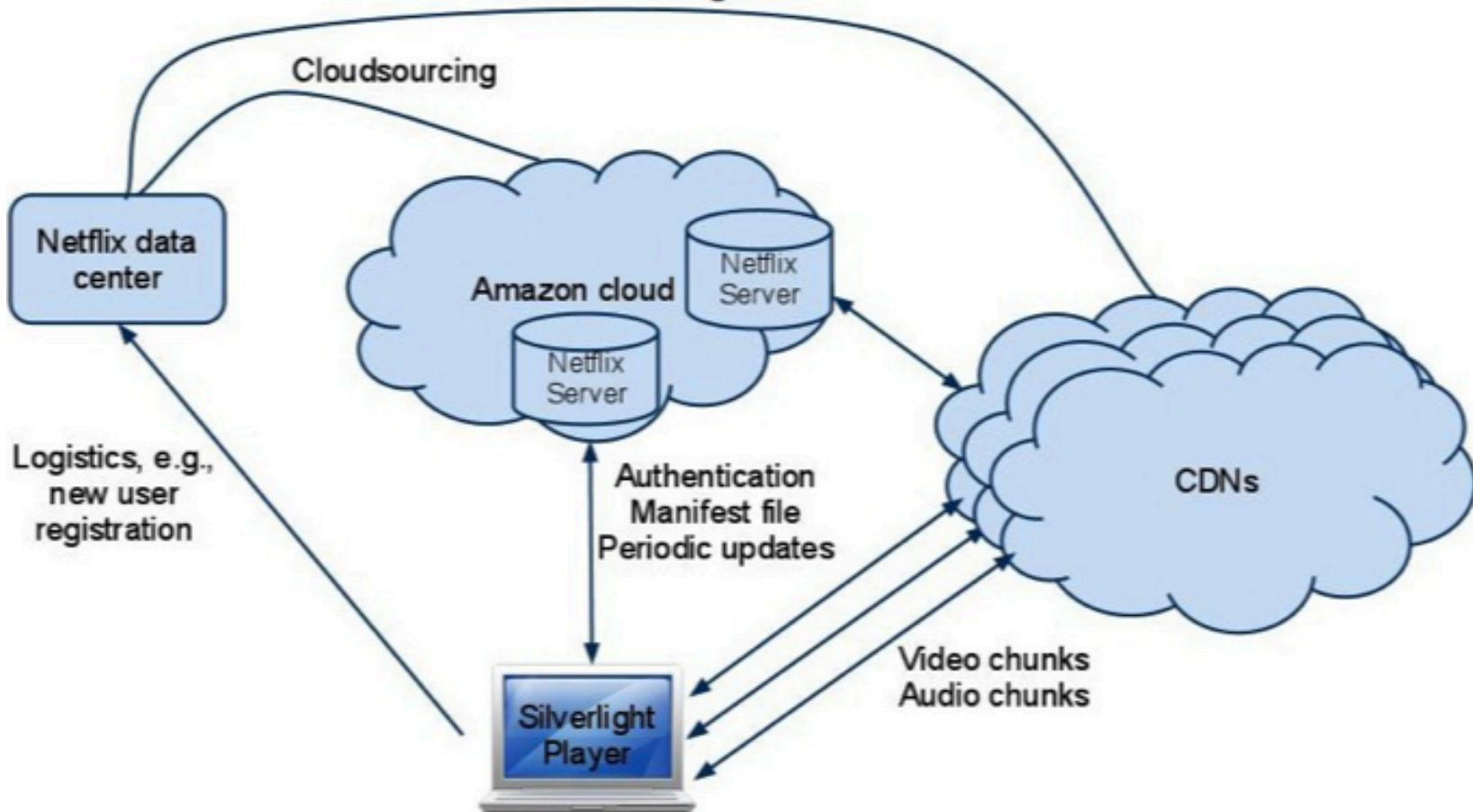
Outline

- Recap
- Single, high-performance 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 |

Outsourcing



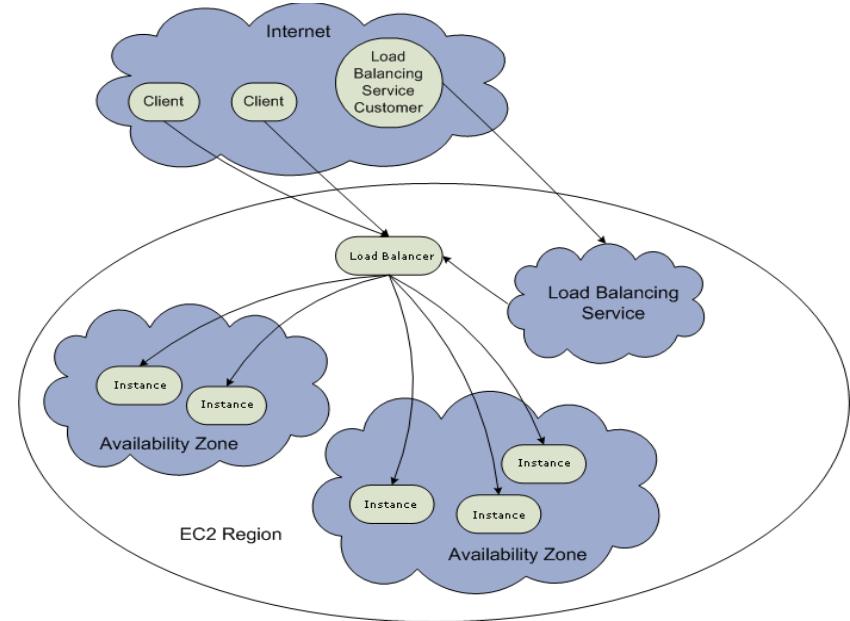
Example: Netflix Manifest File

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

```
<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: 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 register the 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.



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,InstanceP
rotocol=HTTP,InstancePort=80" --
availability-zones us-west-2a us-west-2b
```

Result:

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

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  
    }  
}
```

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"}  

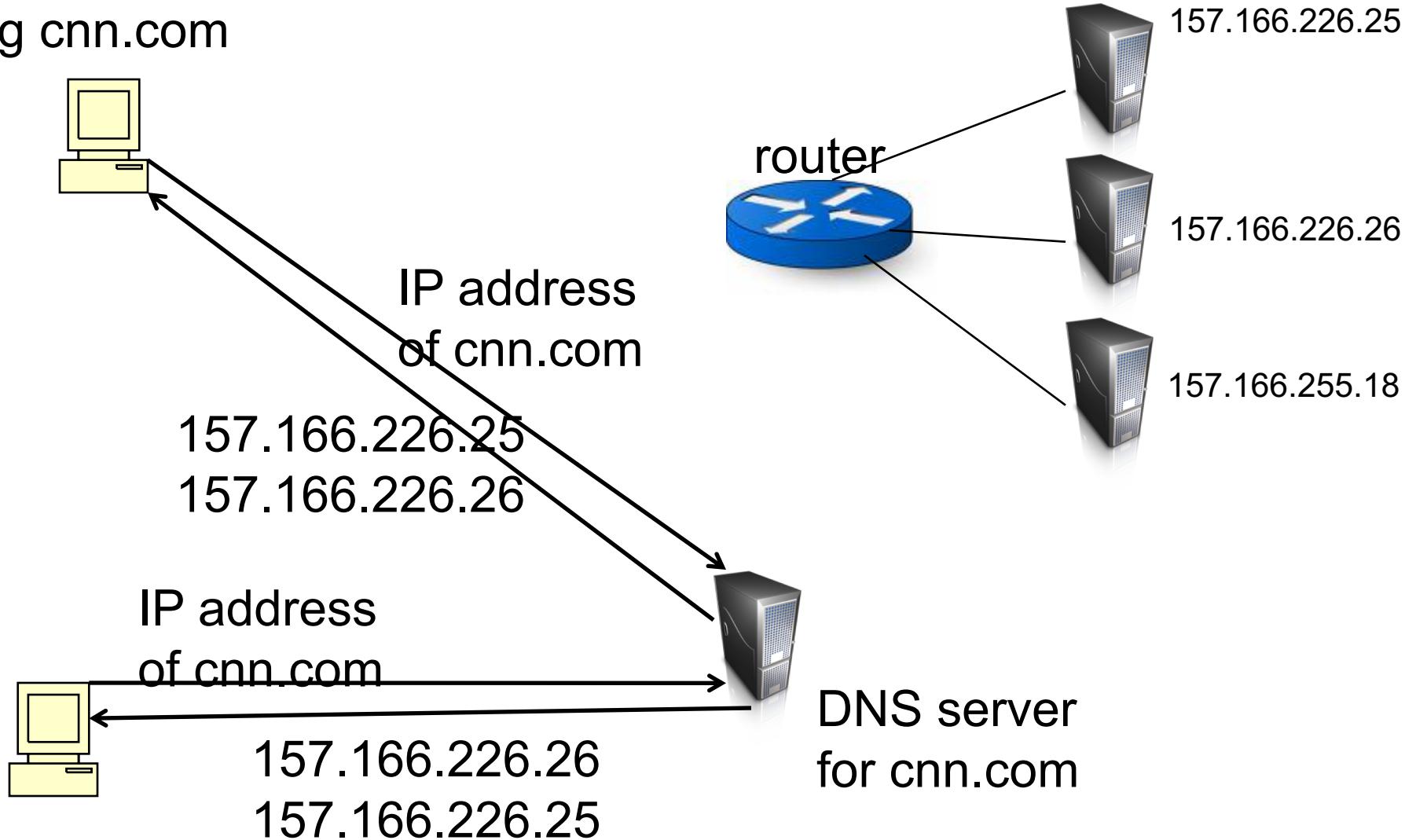
]  
}
```

Outline

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

Basic DNS Indirection and Rotation

%dig cnn.com



CDN Using DNS (Akamai Architecture as an Example)

- Content publisher (e.g., cnn)
 - provides base HTML documents
 - runs **origin** server(s); but delegates heavy-weight content (e.g., images) to CDN
- Akamai runs
 - (~240,000) **edge** servers for hosting content
 - Deployment into 130 countries and 1600 networks
 - Claims 85% Internet users are within a single "network hop" of an Akamai CDN server.
 - customized **DNS redirection servers** to select edge servers based on
 - closeness to client browser, server load

Linking to Akamai

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

changed to

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

- ❑ URL Akamaization is becoming obsolete and supported mostly for legacy reasons

Exercise

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

```
%dig [+trace]  
to see DNS load direction
```

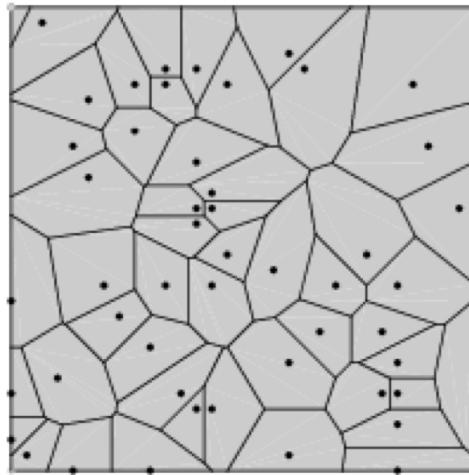
<https://cdn.cnn.com/cnnnext/dam/assets/211031110243-aguero-chest-pain-exlarge-169.jpg>

Recap: CNAME based DNS Name

□ Typical design

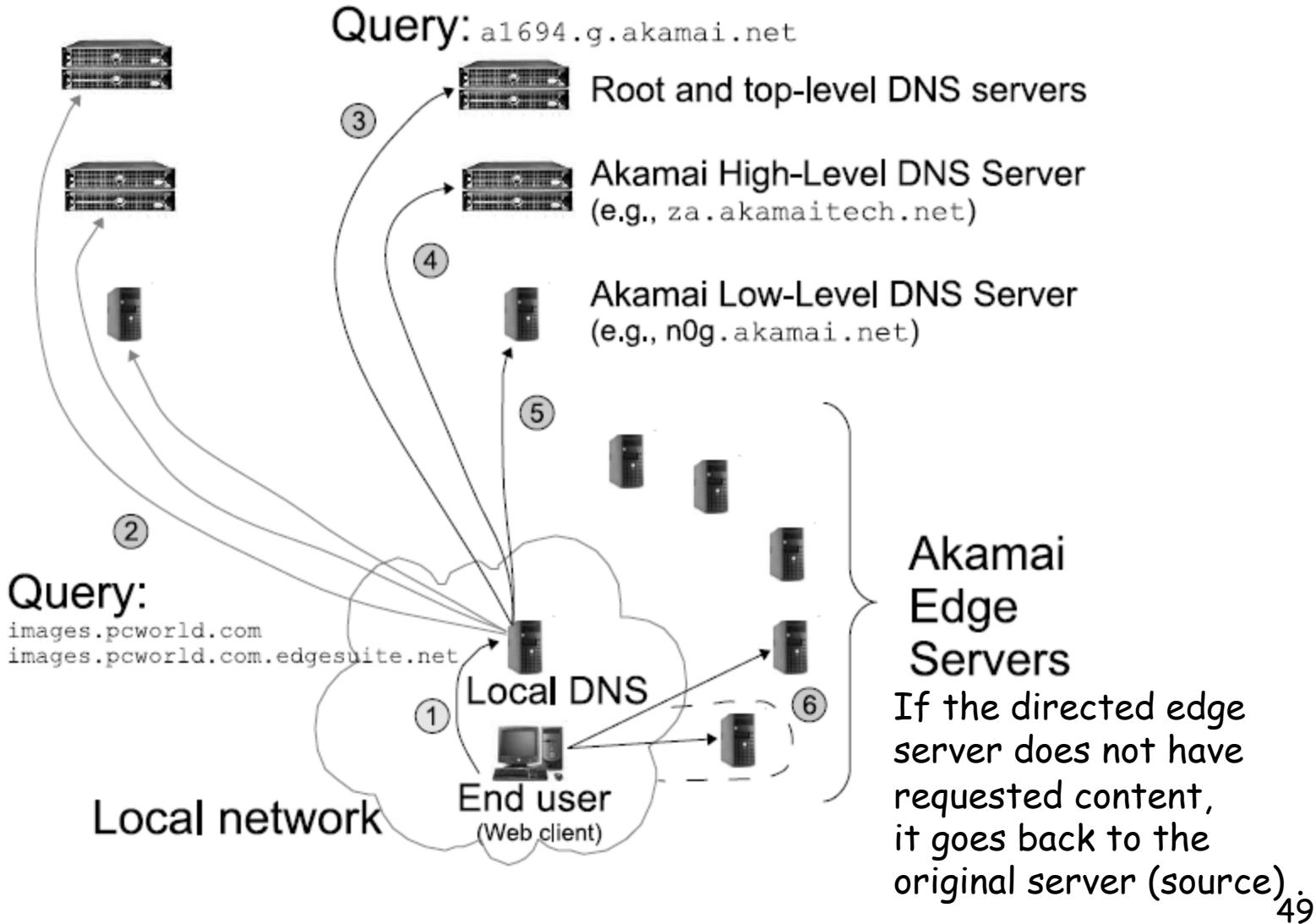
- Use cname to create aliases, e.g.,
<http://cdn.cnn.com/cnnnext/dam/assets/170807134426-forest-bathing-00004819-medium-plus-169.jpg>
- cname: e12596.dscj.akamaiedge.net
 - why two levels in the name?

Two-Level Direction



- high-level DNS determines proximity, directs to low-level DNS;
Input: dscj.akamaiedge.net & and client IP,
Output: region (low-level) DNS
- low-level DNS: who manages a close-by cluster of servers with different IP addresses
Input: e12596.dscj.akamaiedge.net & and client IP
Output: specific servers

Akamai Load Direction



Two-Level DNS Mapping Alg

- High-level
 - Typically geo-location matching from client to reg
- Low-level
 - Typically secret, e.g., details of Akamai algorithms are proprietary
 - Typical goal: load balancing among servers

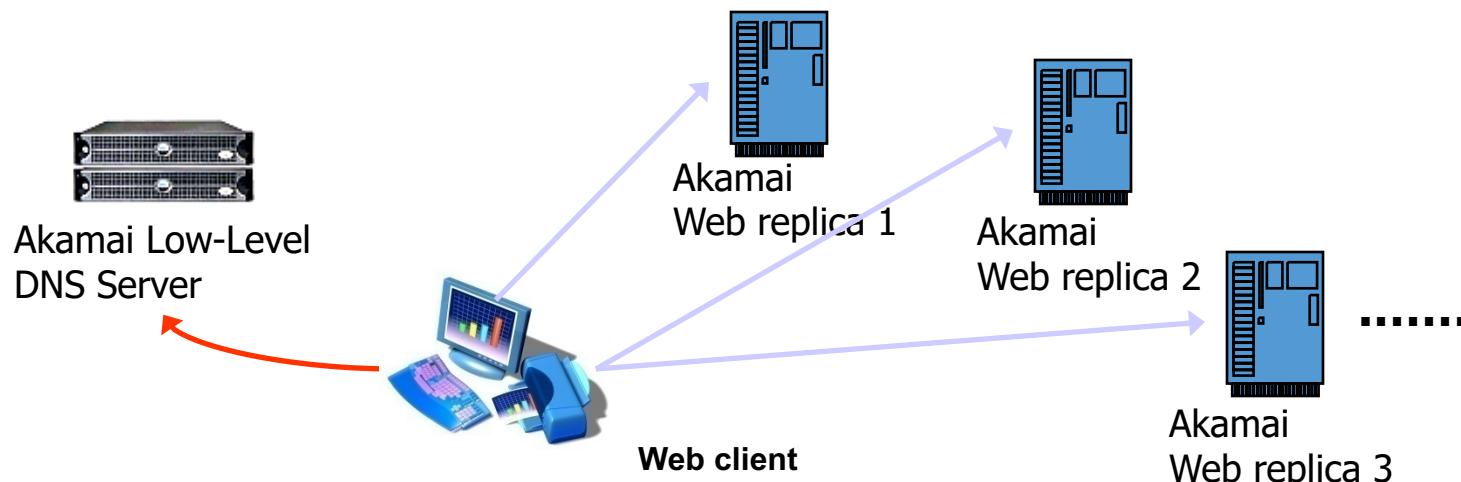
Akamai Local DNS LB Alg

- 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
 - Hash range may be increasing, e.g., first hash [0,1], second [0, 3]
 - Assign the publisher to the first server that does not overload

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.

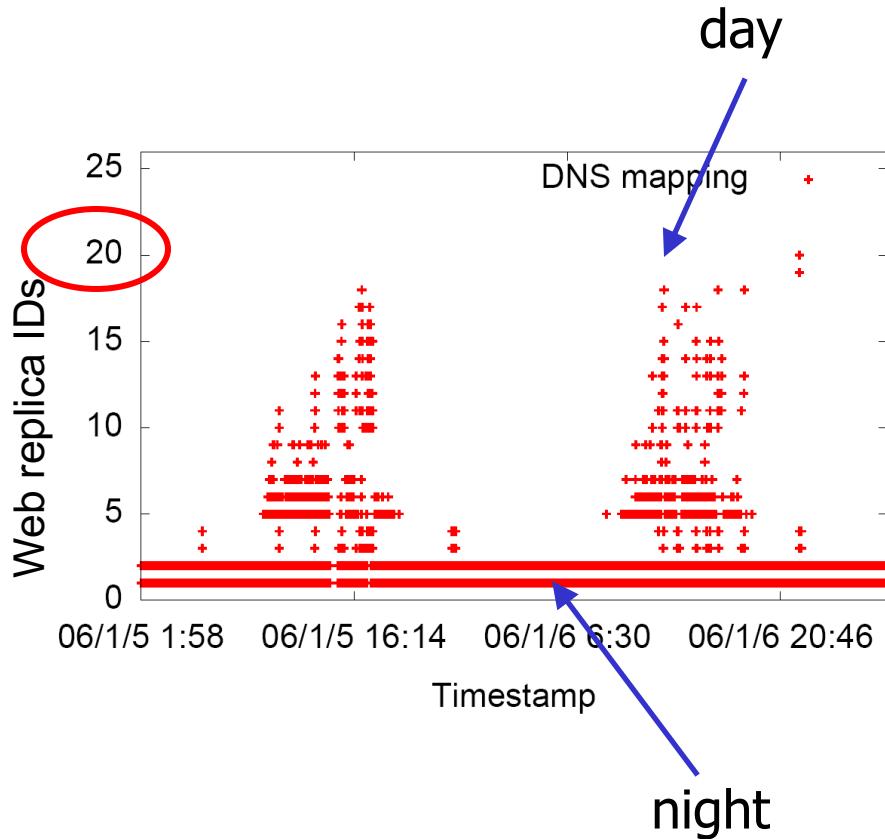


See <http://www.aqualab.cs.northwestern.edu/publications/Ajsu06DBA.pdf>

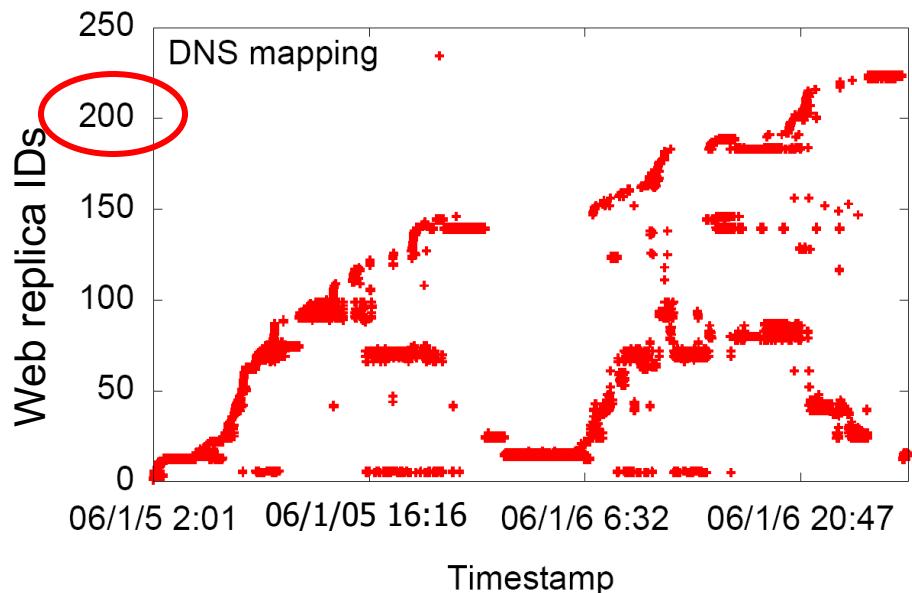
Server Pool: to Yahoo

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

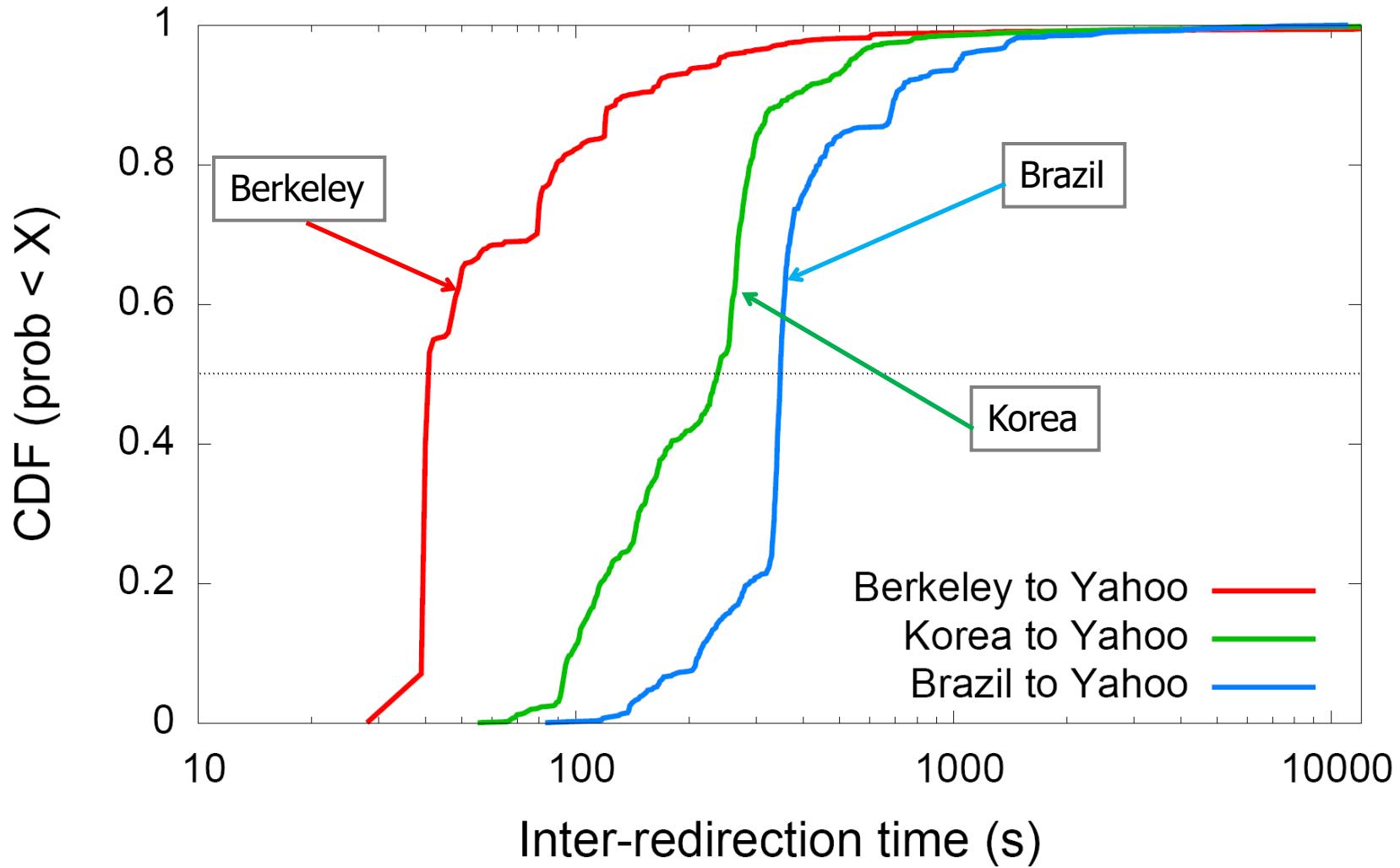
Client 1: Berkeley



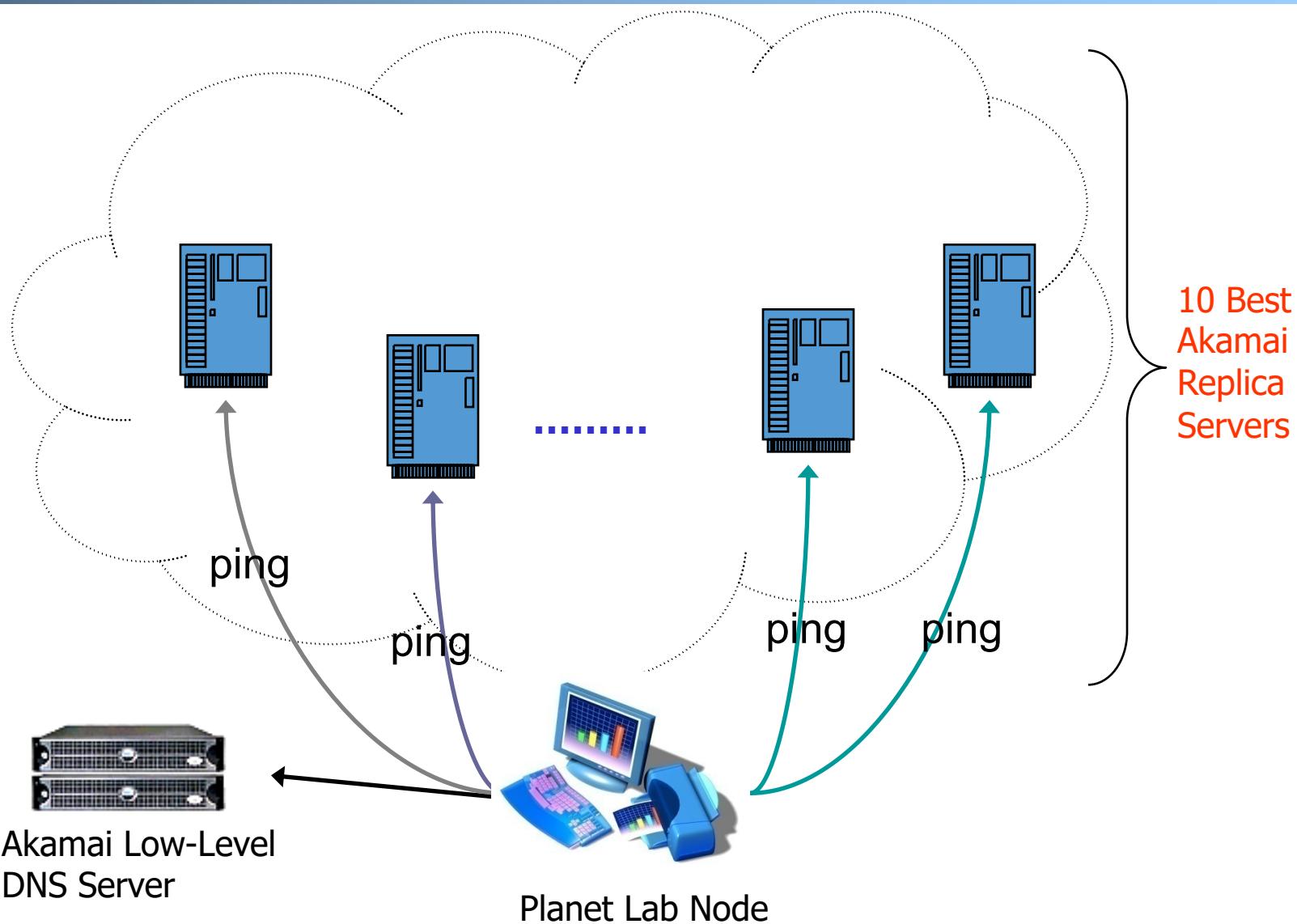
Client 2: Purdue



Load Balancing Dynamics

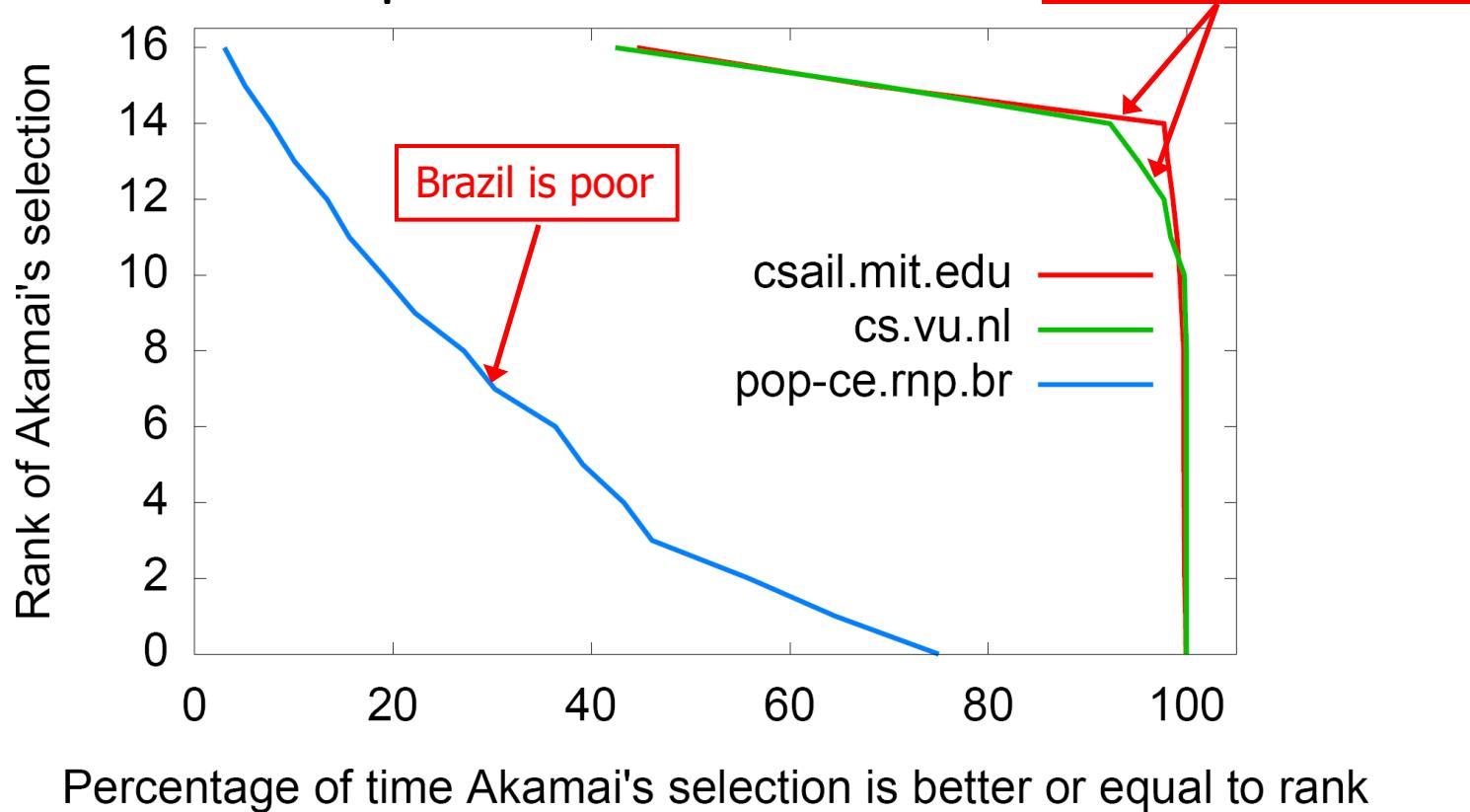


Redirection Effectiveness: Measurement Methodology



Do redirections reveal network conditions?

- Rank = $r_1 + r_2 - 1$, where r_i is rank of server i
 - 16 means perfect correlation



(Offline Read) Facebook DNS Load Direction

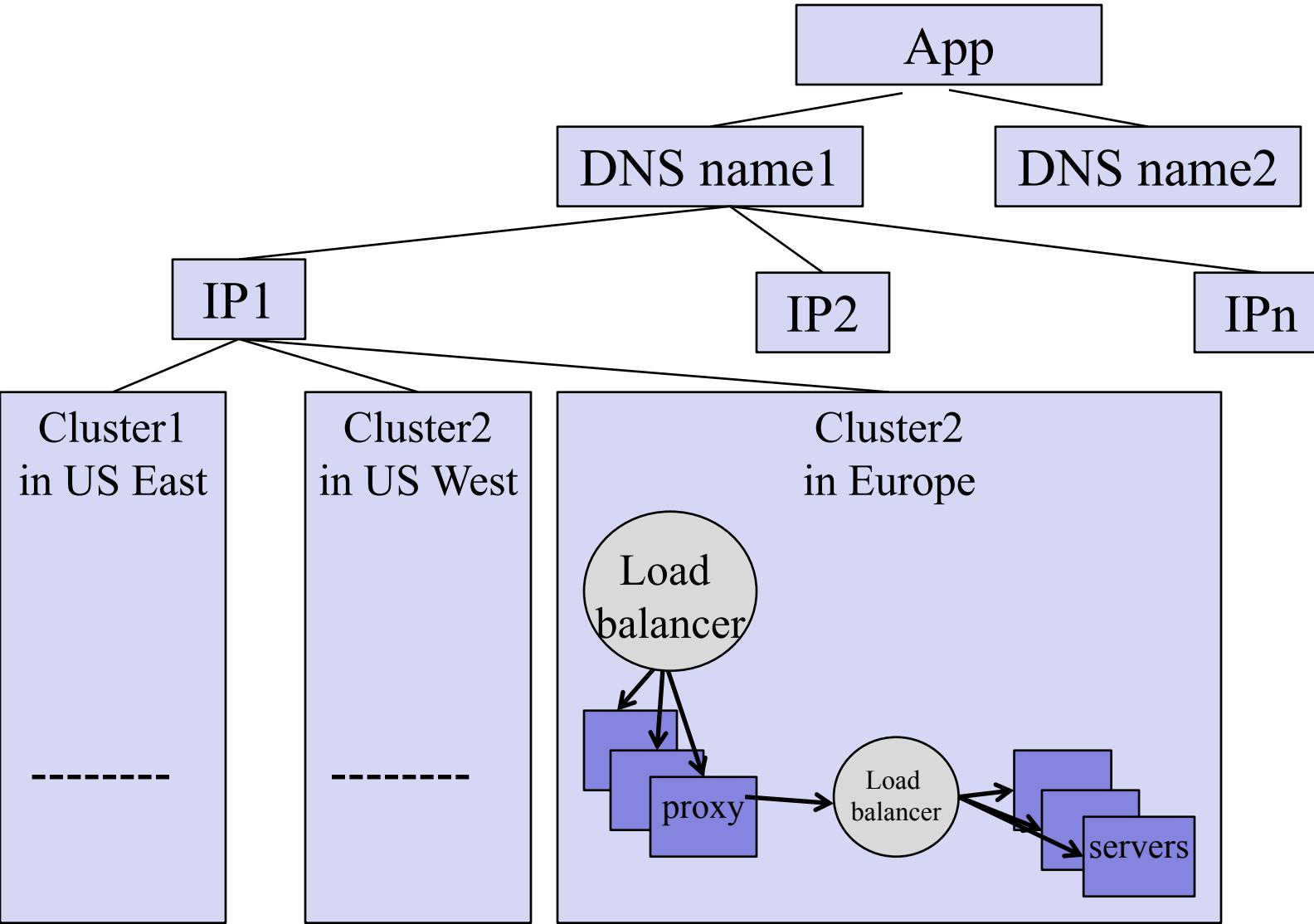
- A system named Cartographer (written in Python) processes measurement data and configures the DNS maps of individual DNS servers (open source tinydns)

Discussion

- Advantages of using DNS for using multiple servers (LB)
 - Leveraging existing DNS features (e.g., cname, hierarchy name for natural hierarchical redirection)
 - Leveraging existing DNS deployment/optimization

- Disadvantages of using DNS
 - Distributed caching may lead to slow response
 - Only in the unit of IP addresses

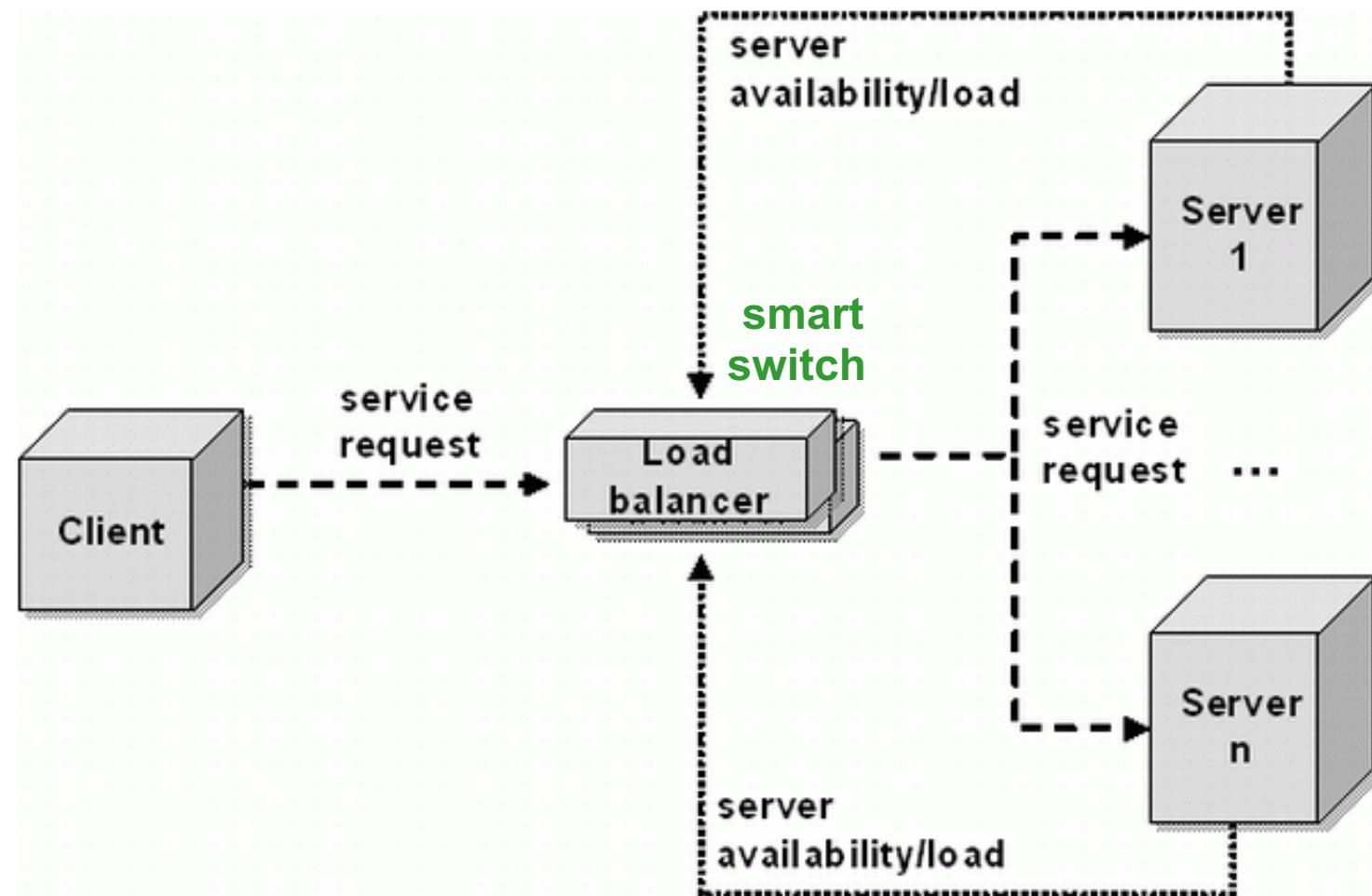
Recap: Direction Mechanisms



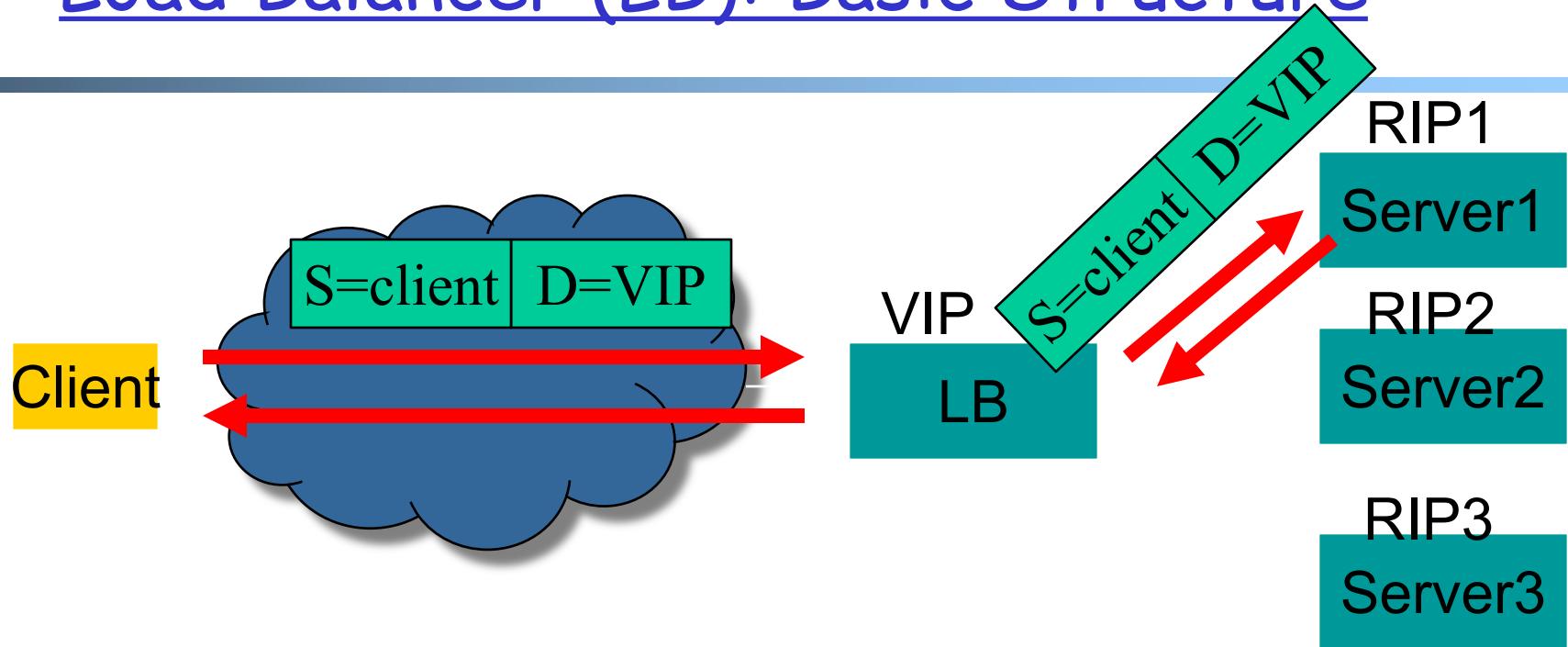
Outline

- Recap
- Single, high-performance network server
- Multiple network servers
 - Basic issues
 - Load direction
 - DNS (IP level)
 - Load balancer/smart switch (sub-IP level)

Smart Switch: Big Picture



Load Balancer (LB): Basic Structure



Problem of the basic structure?

S=client

D=VIP

Problem

- Client to server packet has VIP as destination address, but real servers use RIPv
 - if LB just forwards the packet from client to a real server, the real server drops the packet
 - reply from real server to client has real server IP as source -> client will drop the packet

state: listening
address: {RealIP:**6789**, *:*****}
completed connection queue: C1; C2
sendbuf:
recvbuf:

state:
address: {RealIP:**6789**, 198.69.10.10.**1500**}
sendbuf:
recvbuf:

state:
address: {VIP:**6789**, 198.69.10.10.**1500**}
sendbuf:
recvbuf:

...

...

server

client