Network Applications: Operational Analysis; Load Balancing among Homogeneous Servers

Qiao Xiang, Congming Gao, Qiang Su

https://sngroup.org.cn/courses/cnns-xmuf25/index.shtml

10/14/2025

This deck of slides are heavily based on CPSC 433/533 at Yale University, by courtesy of Dr. Y. Richard Yang.

Outline

- Admin and recap
- HTTP
  - HTTP "acceleration"
  - Operational analysis

Admin

- Lab assignment 2 due today
- Lab assignment 3 to be posted

Recap: Substantial Efforts to Speedup Basic HTTP/1.0

- Reduce the number of objects fetched [Browser cache]

- Reduce data volume [Compression of data]
- Header compression [HTTP/2]

- Reduce the latency to the server to fetch the content [Proxy cache]
- Remove the extra RTTs to fetch an object [Persistent HTTP, aka HTTP/1.1]

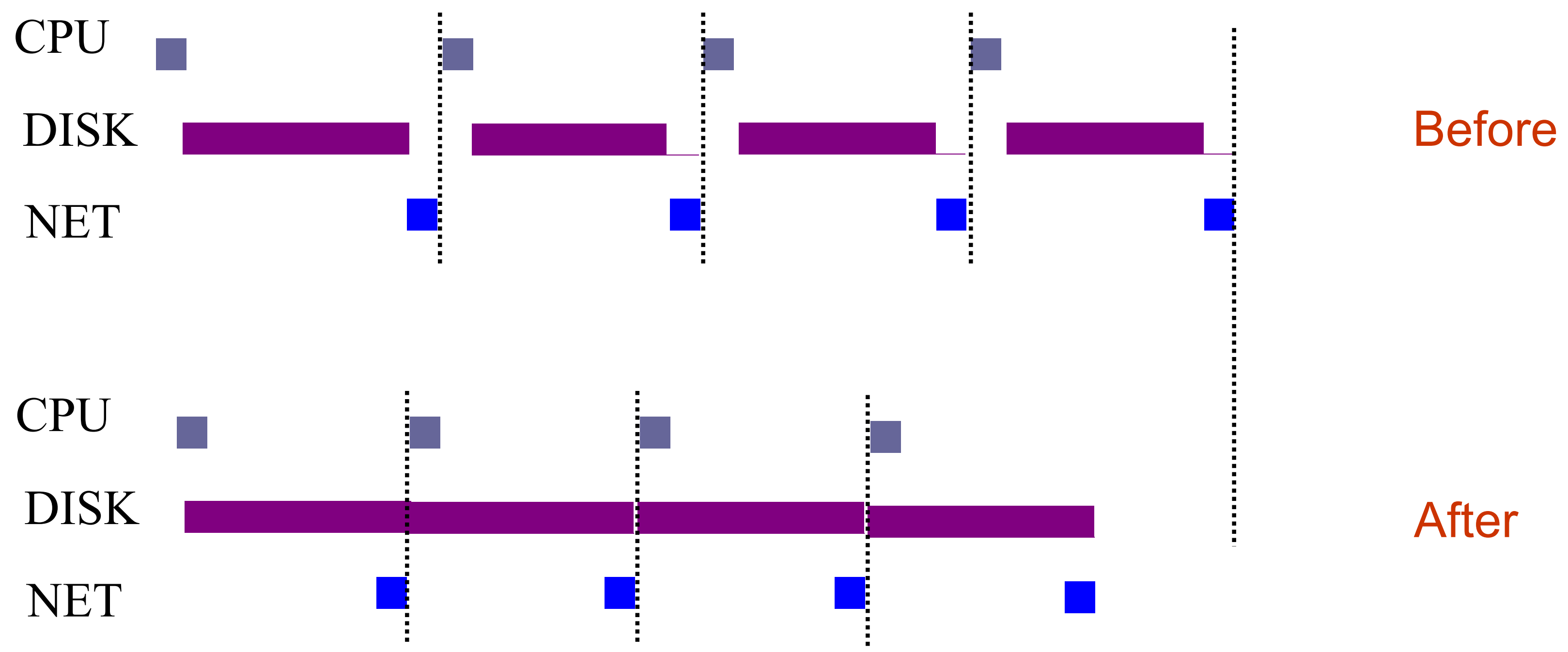
- Increase concurrency [Multiple TCP connections]
- Asynchronous fetch (multiple streams) using a single TCP [HTTP/2]

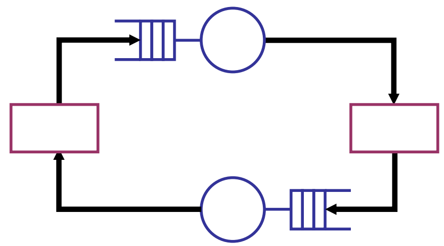
- Server push [HTTP/2]

Outline

- Admin and recap
- HTTP
  - HTTP "acceleration"
  - *Operational analysis*

Goal: Best Server Design Limited Only by Resource Bottleneck





Some Questions

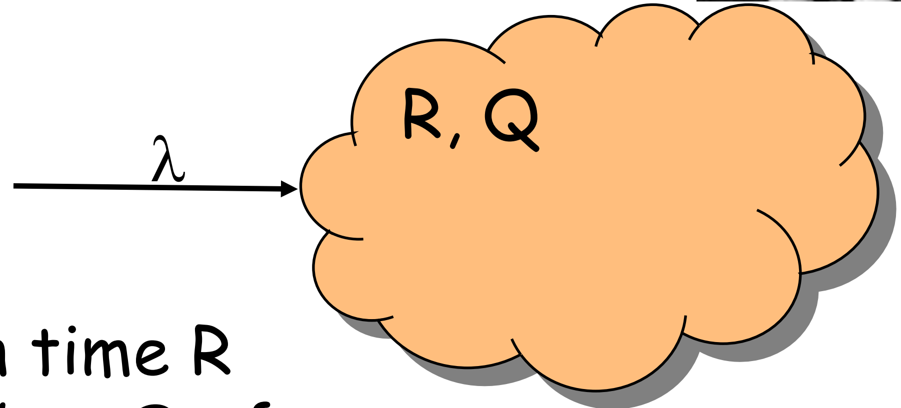
- When is CPU the bottleneck for scalability?
    - So that we need to add helper threads
- How do we know that we are reaching the limit of scalability of a single machine?

- These questions drive network server architecture design

- Some basic performance analysis techniques are good to have

Background: Little's Law (1961)

- For any system with no or (low) loss.
- Assume
  - mean arrival rate λ, mean time R at system, and mean number Q of requests at system



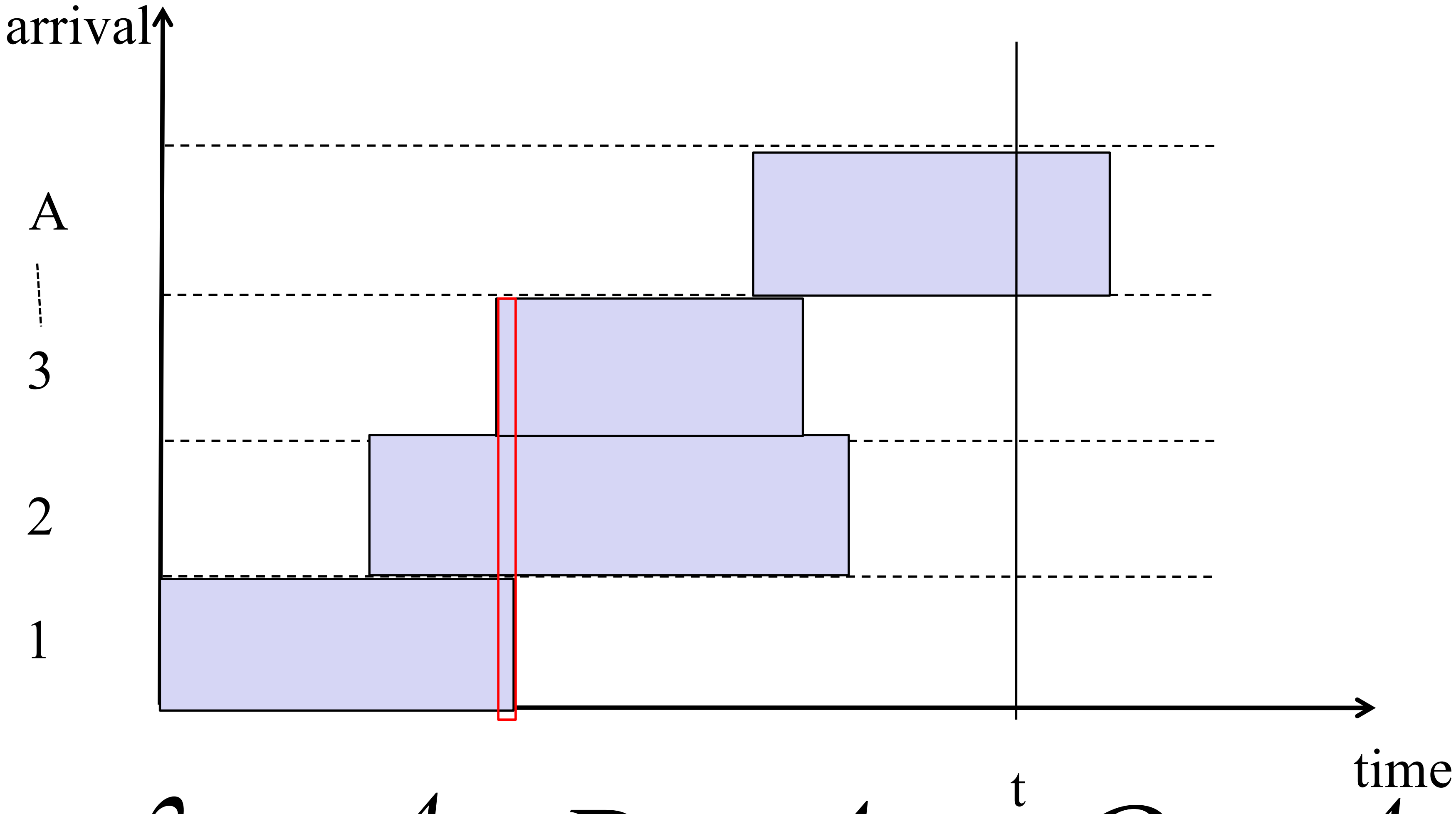
- Then relationship between Q, λ, and R:

$$Q = \lambda R$$

Example: XMU admits 5000 students each year, and mean time a student stays is 4 years, how many students are enrolled?

Little's Law: Proof

$$Q = \lambda R$$



$$\lambda = \frac{A}{t} \qquad R = \frac{Area}{A} \qquad Q = \frac{Area}{t}$$

Operational Analysis: Resource Demand of a Request

CPU

V_{CPU} visits for S_{CPU} units of resource time per visit

Network

V_{Net} visits for S_{Net} units of resource time per visit

Disk

V_{Disk} visits for S_{Disk} units of resource time per visit

Memory

V_{Mem} visits for S_{Mem} units of resource time per visit

Operational Quantities

- T: observation interval
- Bi: busy time of device i
- i = 0 denotes system

Ai: # arrivals to device i
Ci: # completions at device i

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

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

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

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

Utilization Law

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

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

$$= X_i S_i$$

- The law is independent of any assumption on arrival/service process
- Example: Suppose NIC processes 125 pkts/sec, and each pkt takes 2 ms. What is utilization of the network NIC?

Deriving Relationship Between R, U, and S for one Device

- Assume flow balanced (arrival=throughput), Little's Law:

$$Q = \lambda R = XR$$

- Assume PASTA (Poisson arrival--memory-less arrival--sees time average), a new request sees Q ahead of it, and FIFO

$$R = S + QS = S + XRS$$

- According to utilization law, U = XS

$$R = S + UR \longrightarrow R = \frac{S}{1-U}$$

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

$$\text{Utilization } \mathrm{U}_i = X V_i S_i \leq 1$$

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

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

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

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

- 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:

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

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

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):

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

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

In Practice: Common Bottlenecks

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

[Aaron Bannert]

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

YouTube Design Alg.

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

Outline

- Admin and recap
- HTTP: single 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)

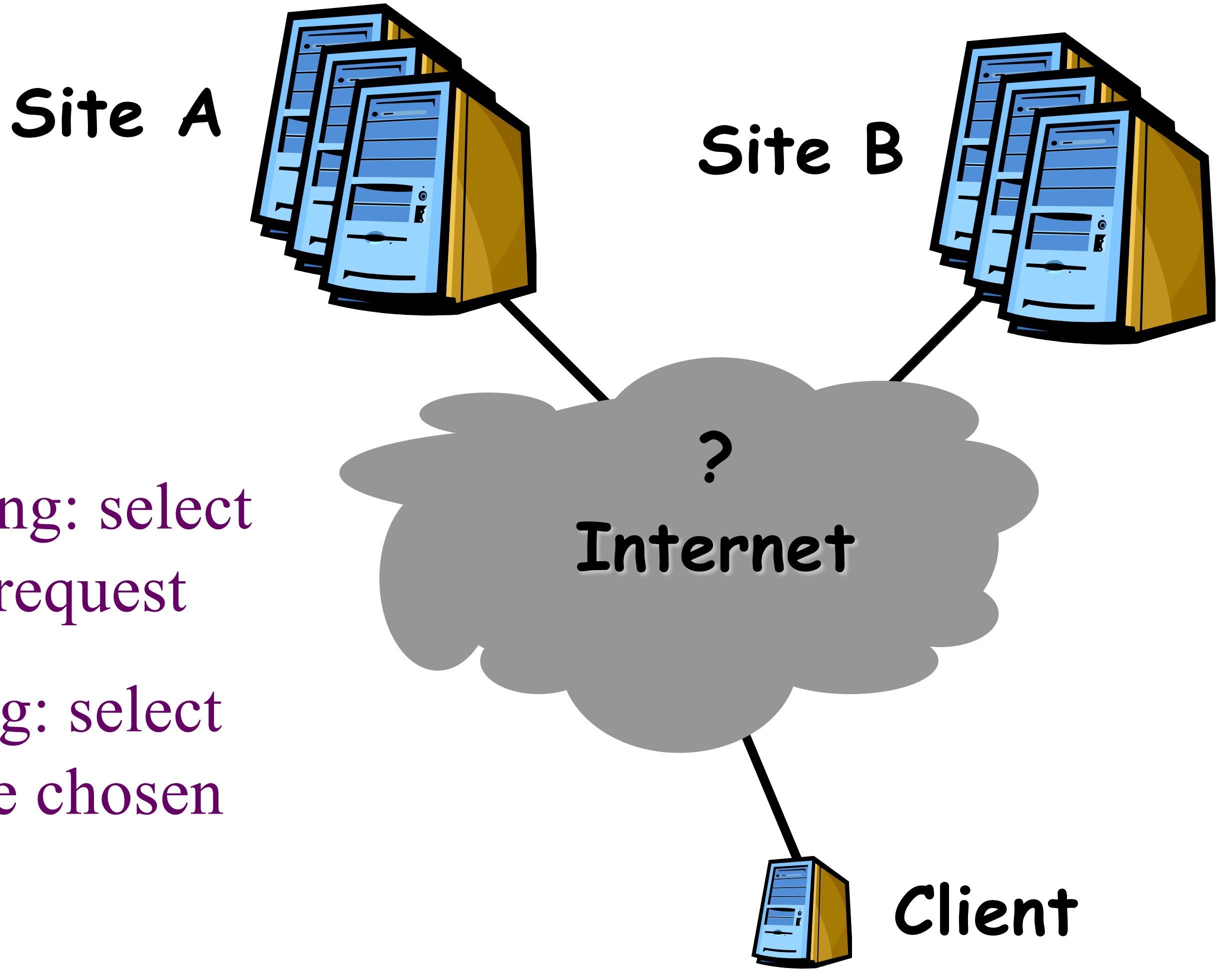
Why Multiple Servers?

- 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

Outline

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

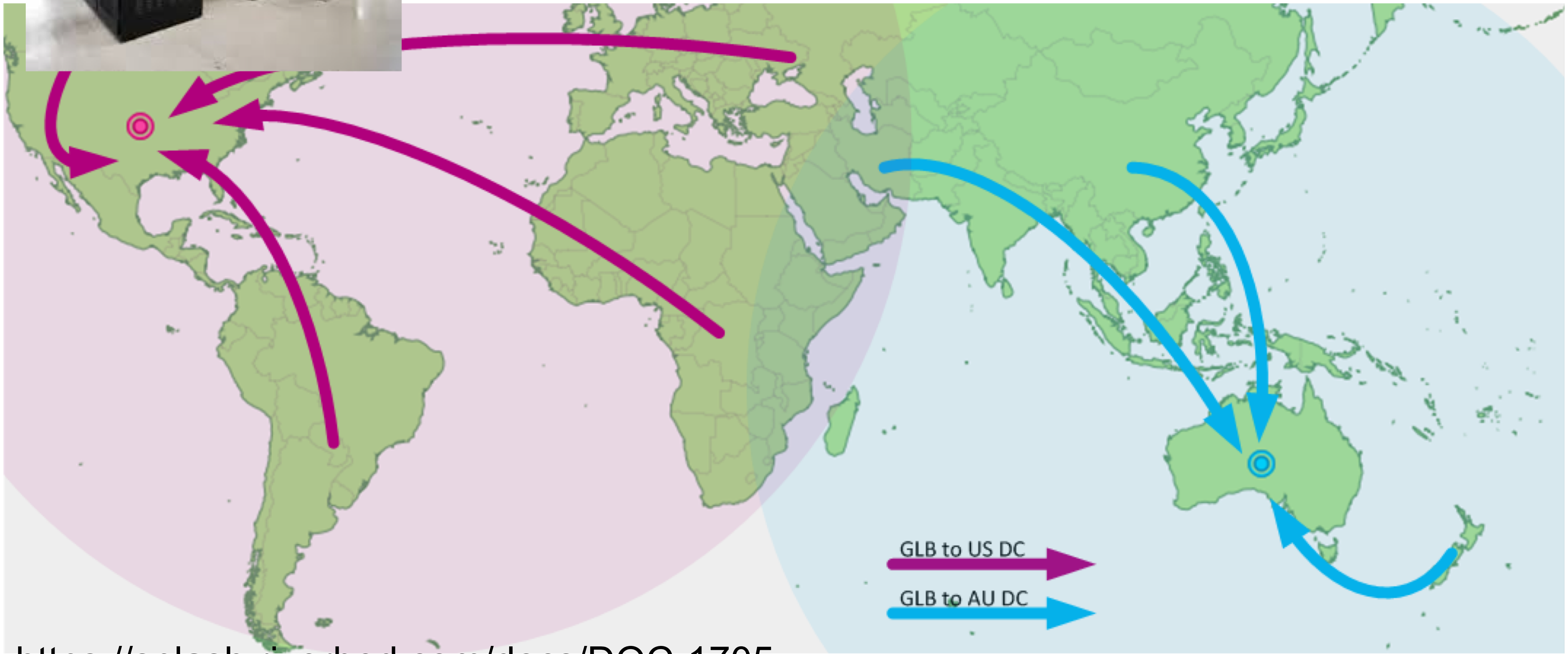
Request Routing: Overview



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

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

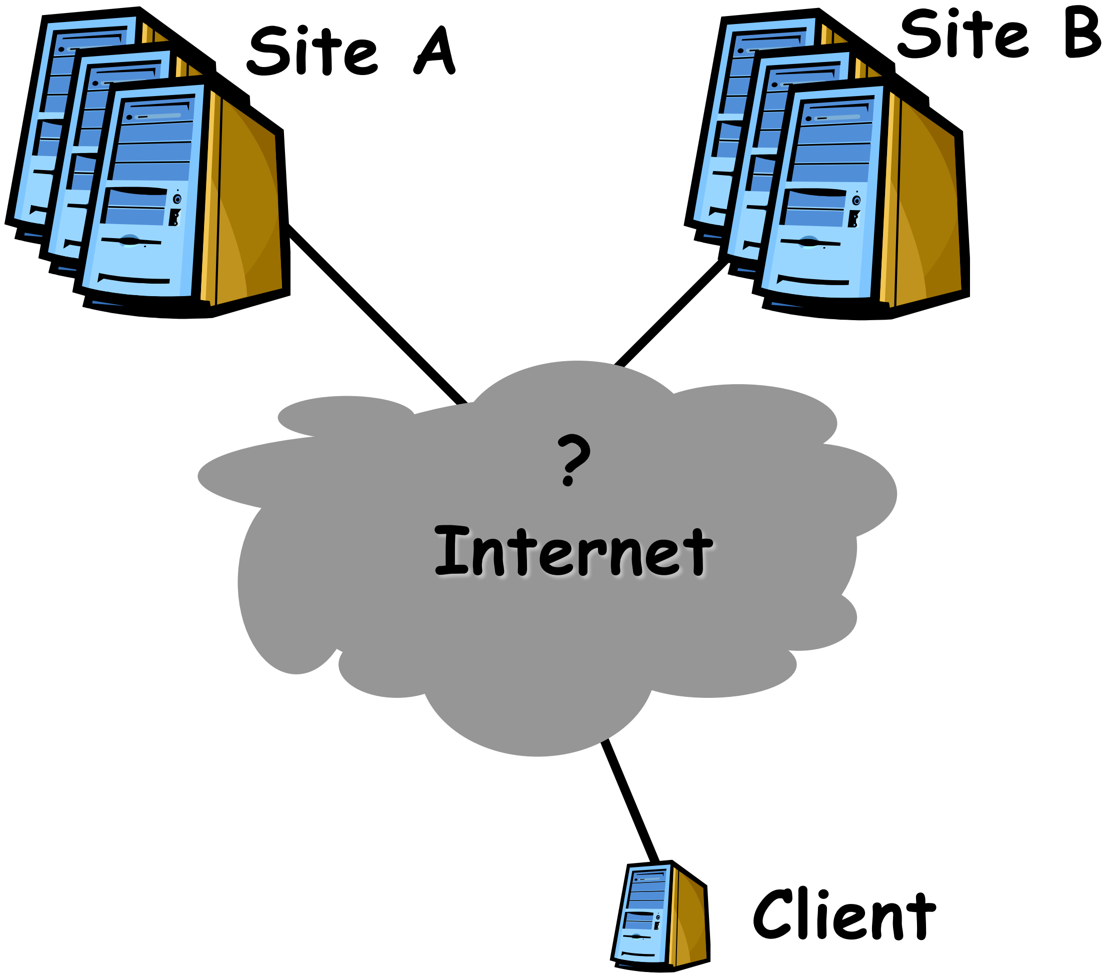
Request Routing: Overview



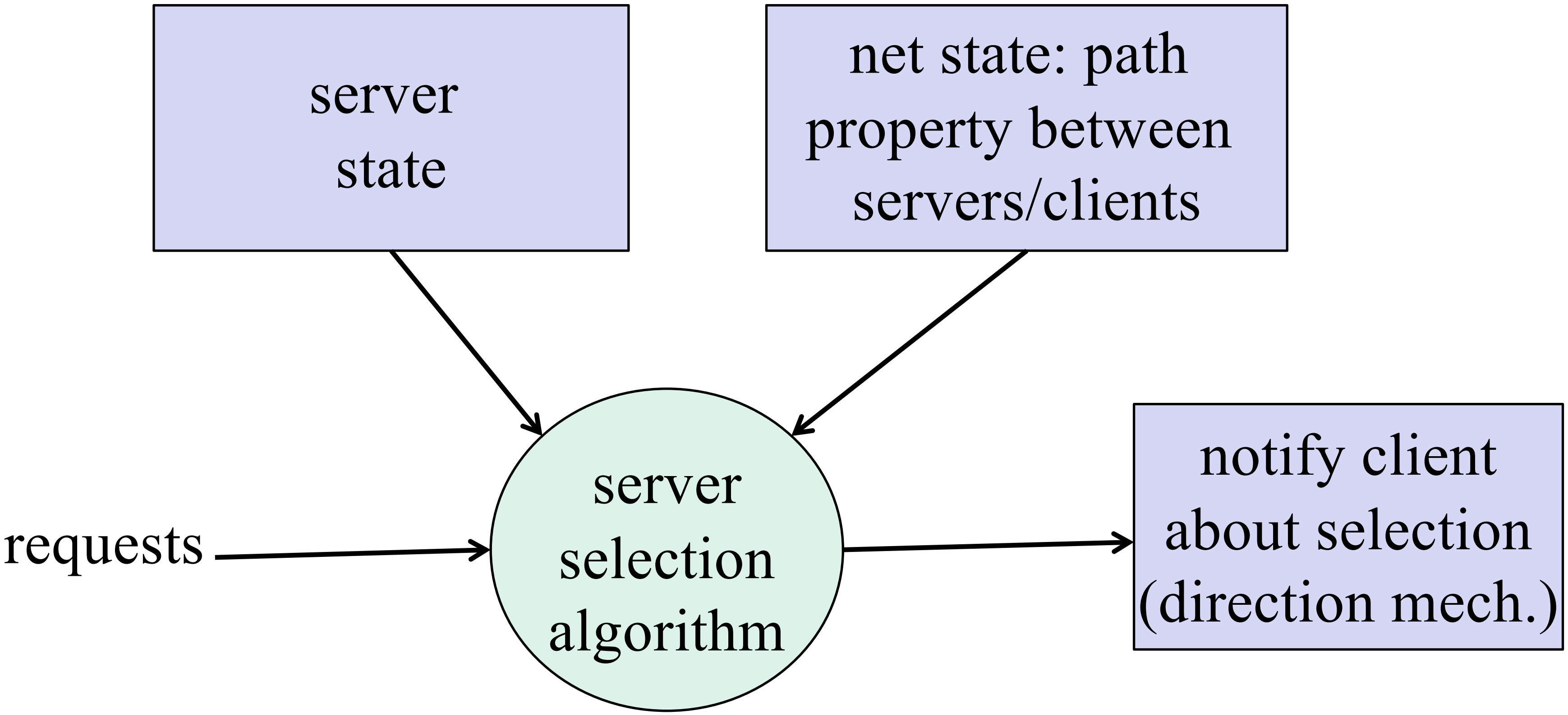
https://splash.riverbed.com/docs/DOC-1705

Request Routing: Basic Architecture

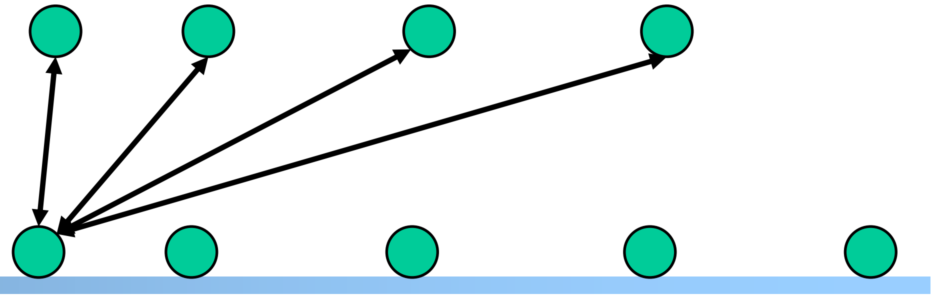
- Major components
  - Server state monitoring
    - Load (incl. failed or not); what requests it can serve
  - Network path properties estimation
    - E.g., bw, delay, loss, network cost between clients and servers
  - Server assignment alg.
  - Request direction mechanism



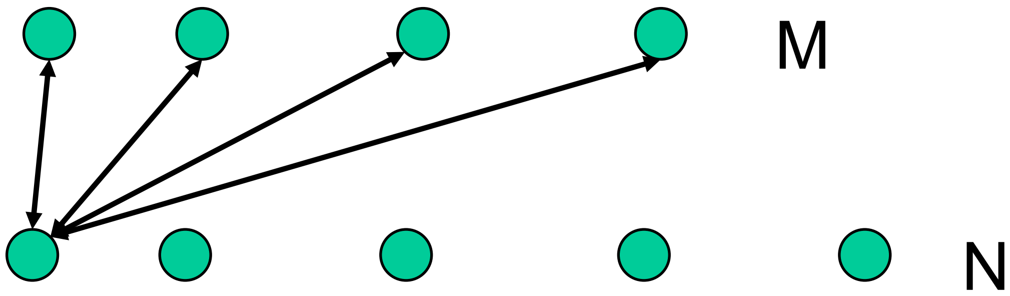
Request Routing: Basic Architecture



Network Path Properties



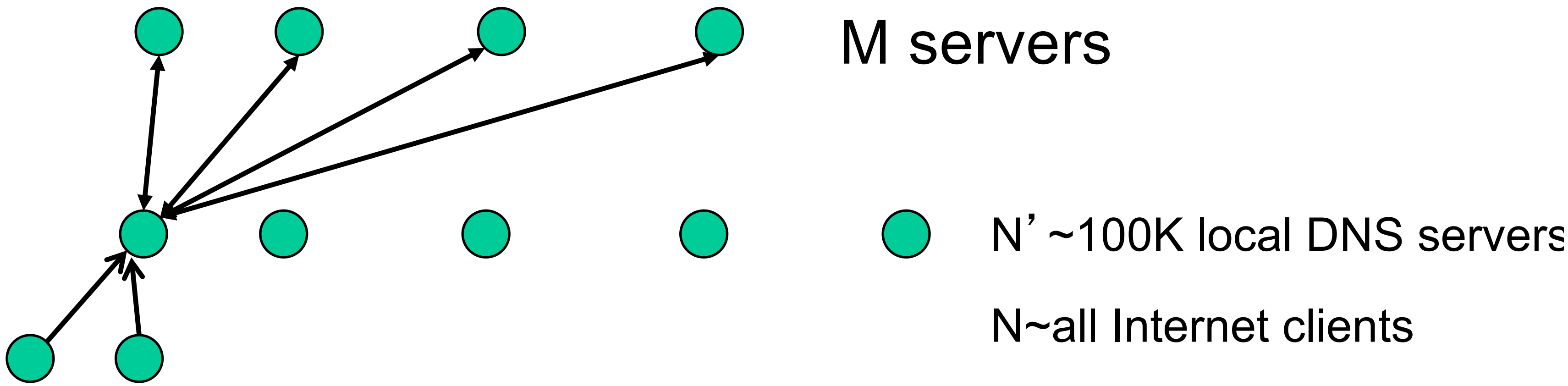
- Why is the problem difficult?
  - Scalability: if do measurements, complete measurements grow with N * M, where
    - N is # of clients
    - M is # of servers



  - Complexity/feasibility in computing path metrics

Network Path Properties: Improve Scalability

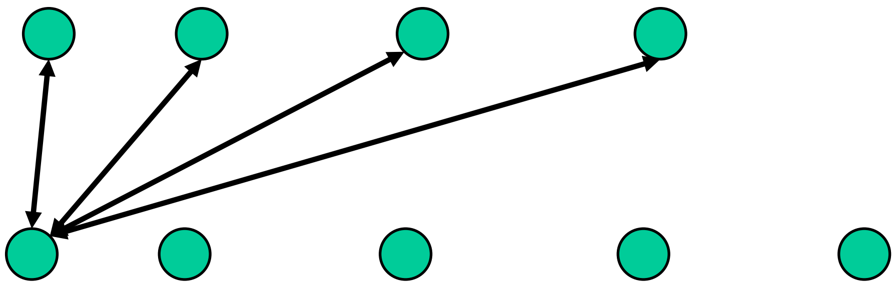
- Aggregation:
  - merge a set of IP addresses (reduce N and M)
    - E.g., when computing path properties, aggregates all clients sharing the same local DNS server



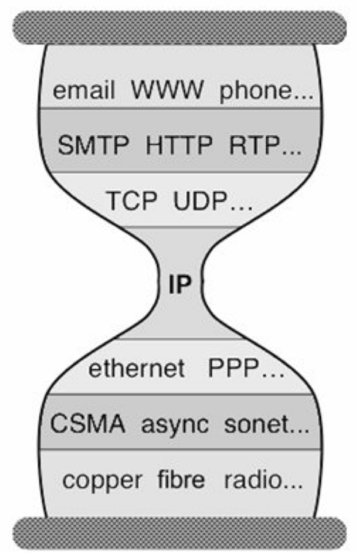
- Sampling and prediction
  - Instead of measuring N*M entries, we measure a subset and predict the unmeasured paths
  - We will cover it later in the course

Server Assignment

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

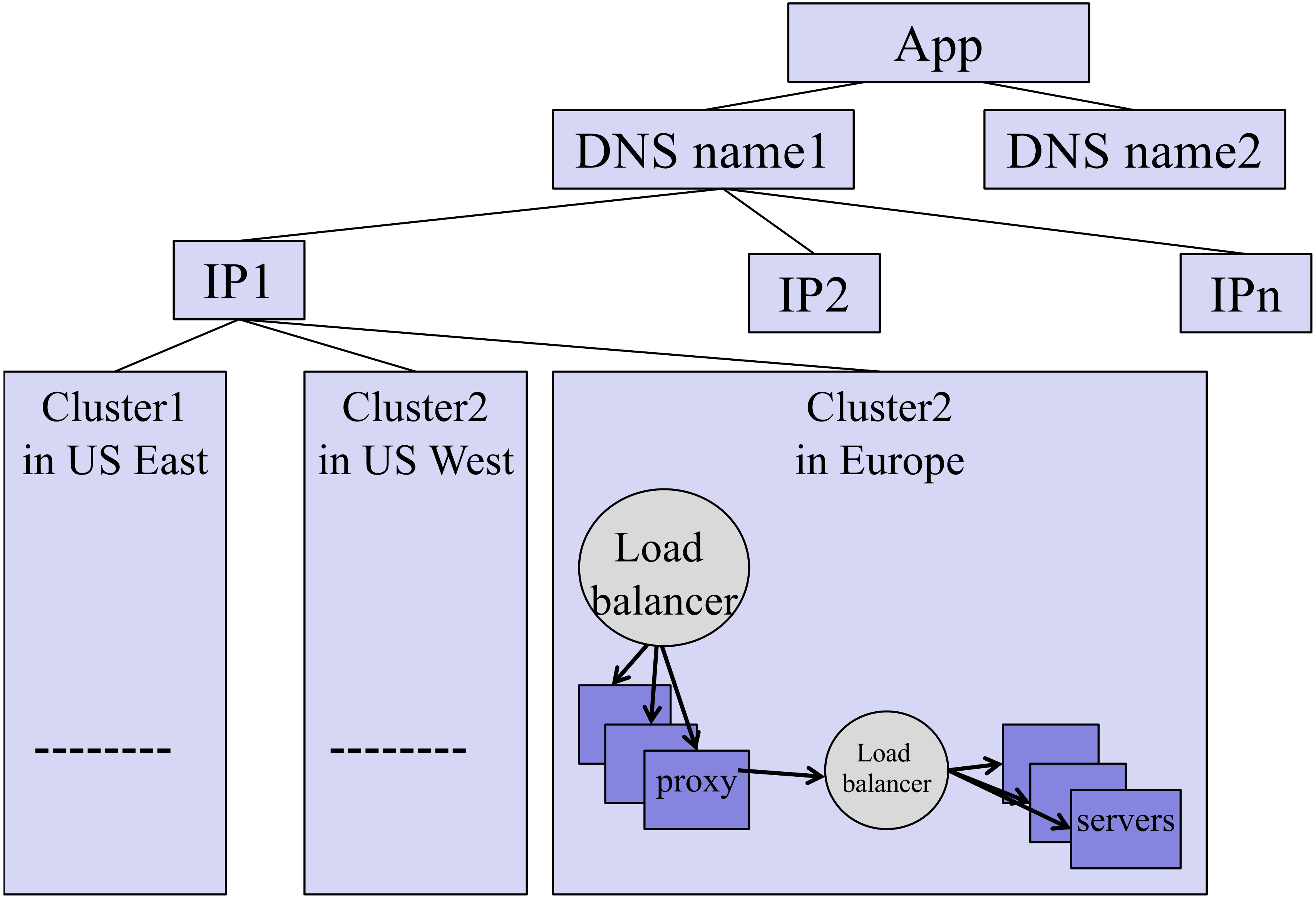


Client Direction Mechanisms

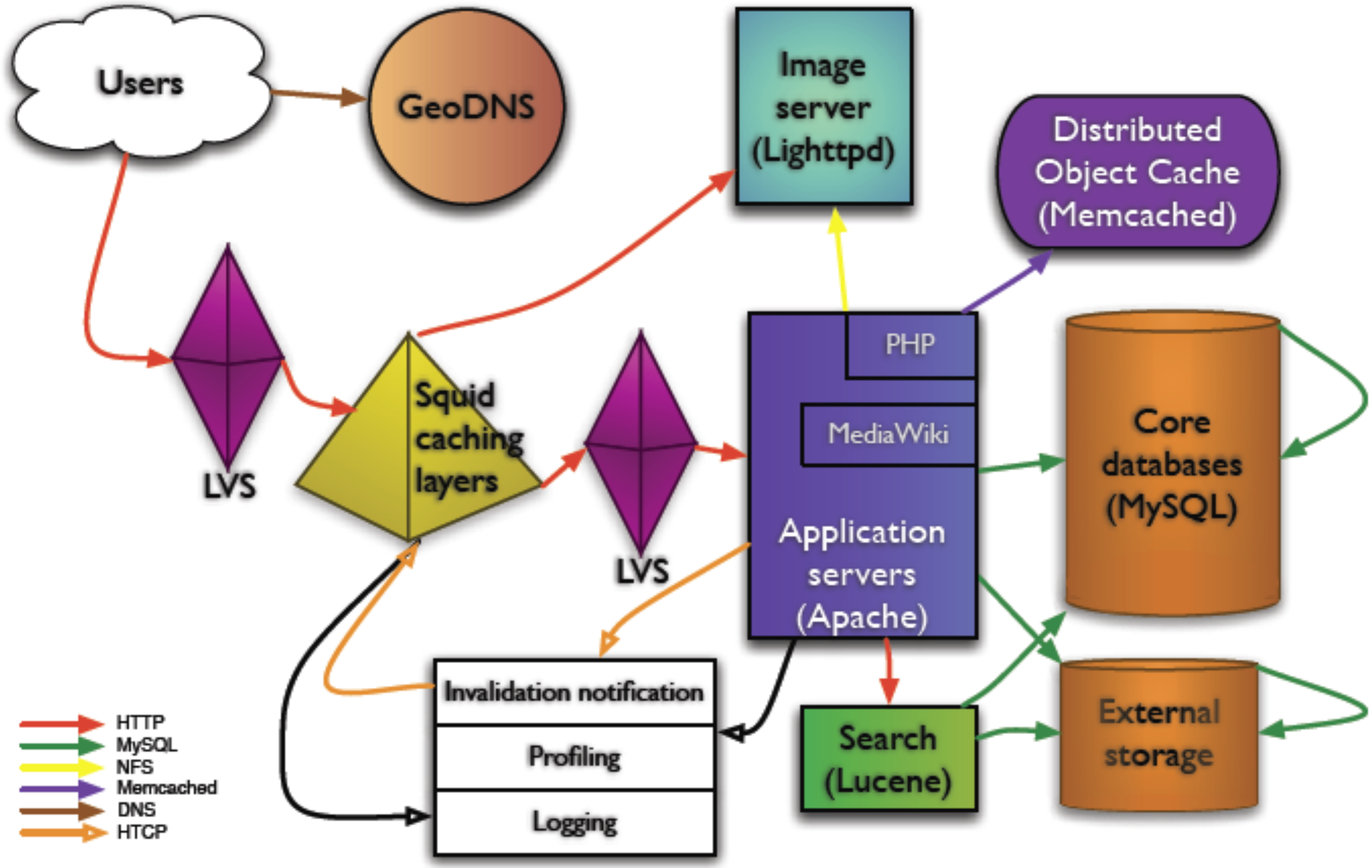


- Key difficulty
  - May need to handle a large of clients
- Basic types of mechanisms
  - Application layer, e.g.,
    - App/user is given a list of candidate server names
    - HTTP redirector
  - DNS: name resolution gives a list of server addresses
  - IP layer: Same IP address represents multiple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



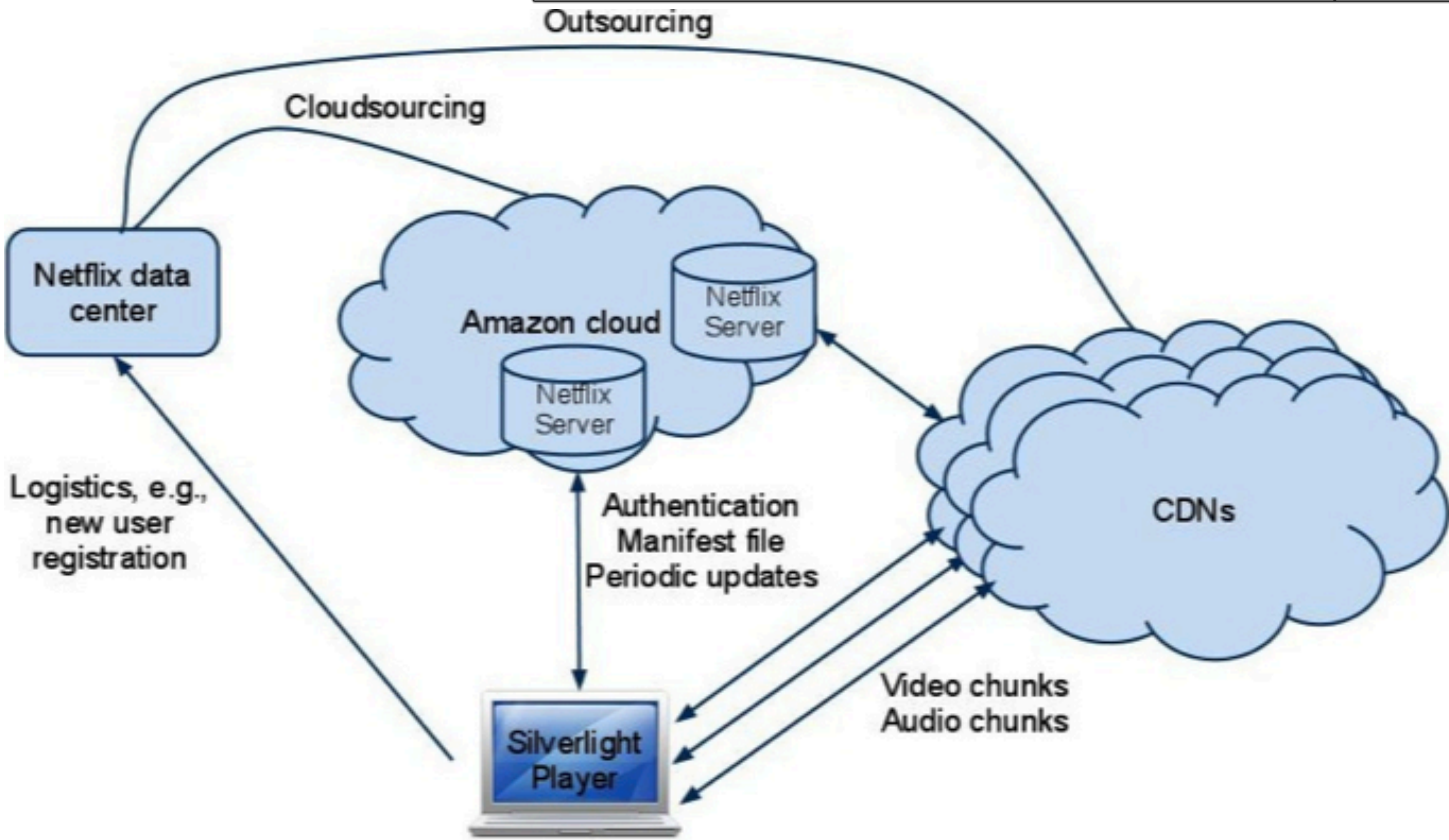
http://wikitech.wikimedia.org/images/8/81/Bergsma_-_Wikimedia_architecture_-_2007.pdf

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



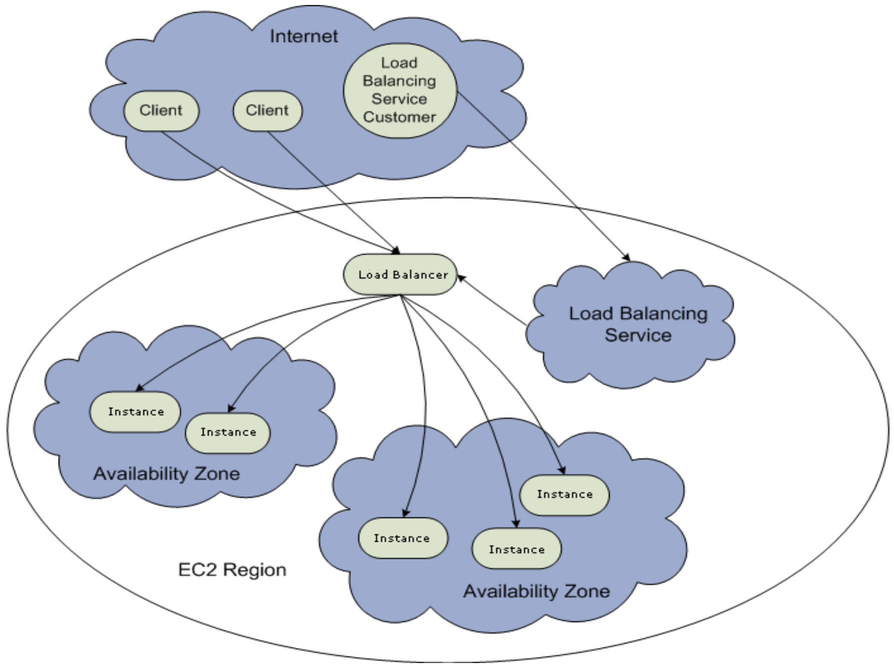
Example: Netflix Manifest File

- 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.



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,UnhealthyThre
shold=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:

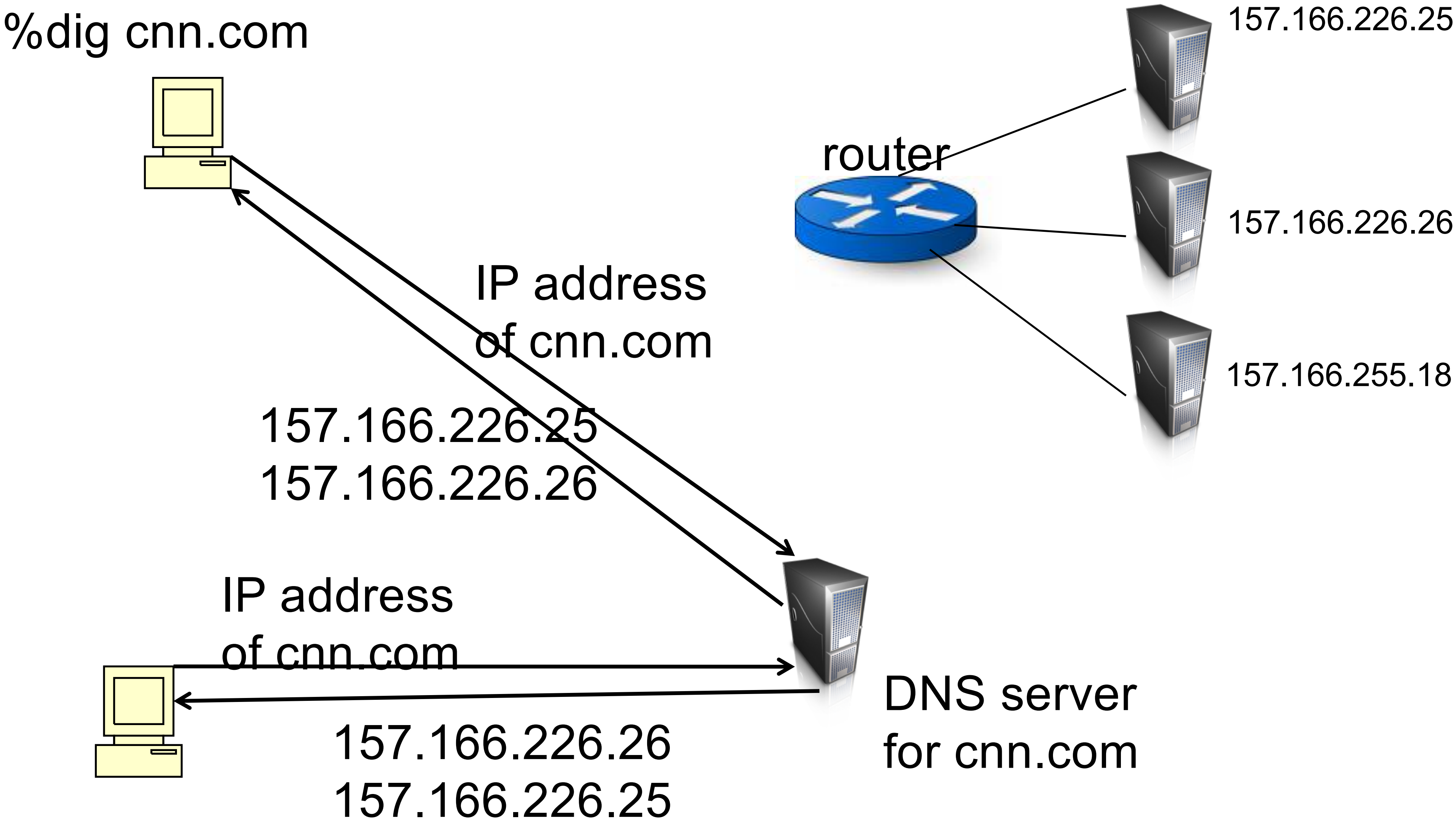

```
{  "Instances": [
    {"InstanceId": "i-d6f6fae3‟},
    {"InstanceId": "i-207d9717‟},
    {"InstanceId": "i-afefb49b‟}
   ]
}
```


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., 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

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

  <IMG SRC = 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

Exercise

https://cdn.nba.com/manage/2021/08/GettyImages-1234610091-scaled-e1665274852371-784x441.jpg

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

  ```
  %dig [+trace]
  ```

  to see DNS load direction

https://www.nba.com/news/reports-lakers-extend-gm-rob-pelinka-through-2025-26-season

Recap: CNAME based DNS Name

- Typical design
  - Use cname to create aliases, e.g., https://cdn.nba.com/manage/2021/08/GettyImages-1234610091-scaled-e1665274852371-1568x882.jpg
  - cname: e8017.dscb.akamaiedge.net
    - why two levels in the name?

- https://cdn.nba.com/manage/2025/10/16x9-43.jpg?w=1568&h=882