Infrastructure Supporting Multiple Servers:
DNS Extension: DNS-SD;
Linda; DNS for RR;
L7 Request Routing; L4 Request Routing--NAT

Y. Richard Yang

http://zoo.cs.yale.edu/classes/cs434/

03/10/2021
Admin

- Schedule of topics
Recap: Roadmap

- We first focus on using multiple largely homogenous (similar) servers providing resource/latency scaling.
Recap: Basic Notification/Direction (Request Routing) Abstractions

Name Abstraction (DNS)

IP/Locator Abstraction (LB, IP anycast)
Recap: Design Features One May Learn from DNS

- Hierarchical name space and hierarchical delegation avoids administrative bottleneck/central control, improving manageability and scalability
- Multiple domain servers improve scalability/robustness
- Native caching (control) reduces workload and improves robustness
- Flexible recursive and iterative query allows structure such as local resolver to simplify client and enable caching
- Using UDP to reduce overhead but also support TCP using the same format
- Same query and response format can simplify result forwarding
- Interesting <length><content> encoding and pointer for compression
- Proactive answers of anticipated queries (server push) reduce # queries on server and latency on client
Recap: Problems/Remaining Issues of Basic DNS

- **Security**
  - Security of DNS information [see DNSSEC]

- **Architecture**
  - Each local domain needs servers, but an ad hoc domain may not have a DNS server
    - Example: typical home network does not have a DNS server

- **Information/data/query model**
  - Limited key (fixed harder to extend qtype; see [1])
  - One dimension (nodes on the tree, not flow from src to dst)

- **Update model**
  - Largely a read data store
    - secondary NS server sync up w/ primary [AXFR query]
    - although theoretically you can update the values of the records, it is rarely enabled

Recap: DNS-Service Discovery and mDNS

- Form the foundation of Apple Bonjour/Airplay
- Often used in home network for zero-touch management (no DNS server set up)

Demo on Mac
  - See list of commands `dns-sd`
  - List Airplay in my home
    - `dns-sd -B _airplay._tcp .`
  - See a particular instance
    - `dns-sd -L <instance> <service> <domain>`
Problem: Each local domain needs DNS servers, but an ad hoc domain may not have a DNS server.

Solution:
- Each node (host) can become a responder.
- Each node (host) can use multicast to announce (write) its values.
  - Utilize IP multicast (broadcast medium).
  - Link-local addressing: send to multicast address: 224.0.0.251 (address as a group name, and any host can specify that it joins the group).
DNS-SERVICE DISCOVERY COMPONENT: DNS SERVER SELECTION (SRV) [RFC2782]

- Problems:
  - Each new service needs to add a new DNS resource record type
  - Basic DNS information model does not provide server information such as port, priority, ...

- Solution:
  - Query: discover servers of a service by query type SRV:
    
    name: <service> . <domain> // <service> := <_service> . <_protocol>
    
    e.g.,
    
    _printer._tcp.example.com. _printer._tcp.local.

  - Result (SRV record type)
    
    _Service._Proto.Name TTL Class SRV Priority Weight Port Target
DNS-Service Discovery Component:
DNS-based Service Discovery [RFC6763]

- Problem:
  - Different instances are not the same (as SRV intended), e.g., different printers should be identified by instance name to allow easier selection

- Solution
  - `<service> . <protocol>. <domain> => <instance> . <service> . <protocol>. <domain>
    e.g., "My Test" _printer._tcp dns-sd.org.
  - Query PTR type returns all instances
    • Provide more info on an instance using SRV, TXT records
Exercise: DNS-SD
Local Service Registration

- Demo: Advertise (register) an LPR printer on port 6789
- Exercise: what messages will be posted on the network?

```
dns-sd -R "My Test" _printer._tcp 6789 pdl=application/postscript
```

- Name of instance providing the service
- `<type_service>`.`<transport>`
- domain (. means default, which is local)
- port
- `Txt` for additional data

○ `dns-sd -B _printer._tcp`
○ Capture packets using wireshark
○ Check System Preference/Printer +
<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>24</td>
<td>172.27.21.251</td>
<td>224.0.0.251</td>
<td>MNIS</td>
<td>232 Standard query response 0x0000 PTR, cache flush Ys-MacBook-Pro.local PTR, cache flush Ys-MacBook-Pro.local NSEC, ca.</td>
</tr>
<tr>
<td>25</td>
<td>fe80::1c9f:22de:19a:ff82:ff</td>
<td></td>
<td>MNIS</td>
<td>252 Standard query response 0x0000 PTR, cache flush Ys-MacBook-Pro.local PTR, cache flush Ys-MacBook-Pro.local NSEC, ca.</td>
</tr>
<tr>
<td>26</td>
<td>172.27.21.251</td>
<td>224.0.0.251</td>
<td>MNIS</td>
<td>172 Standard query 0x0000 ANY Ys-MacBook-Pro.local, &quot;OU&quot; question ANY Ys-MacBook-Pro.local, &quot;OU&quot; question SRV 0</td>
</tr>
<tr>
<td>27</td>
<td>fe80::1c9f:22de:19a:ff82:ff</td>
<td></td>
<td>MNIS</td>
<td>192 Standard query 0x0000 ANY Ys-MacBook-Pro.local, &quot;OU&quot; question ANY Ys-MacBook-Pro.local, &quot;OU&quot; question SRV 0</td>
</tr>
<tr>
<td>28</td>
<td>172.27.21.251</td>
<td>224.0.0.251</td>
<td>MNIS</td>
<td>172 Standard query 0x0000 ANY Ys-MacBook-Pro.local, &quot;OU&quot; question ANY Ys-MacBook-Pro.local, &quot;OU&quot; question SRV 0</td>
</tr>
<tr>
<td>29</td>
<td>fe80::1c9f:22de:19a:ff82:ff</td>
<td></td>
<td>MNIS</td>
<td>192 Standard query 0x0000 ANY Ys-MacBook-Pro.local, &quot;OU&quot; question ANY Ys-MacBook-Pro.local, &quot;OU&quot; question SRV 0</td>
</tr>
<tr>
<td>30</td>
<td>172.27.21.251</td>
<td>224.0.0.251</td>
<td>MNIS</td>
<td>172 Standard query 0x0000 ANY Ys-MacBook-Pro.local, &quot;OU&quot; question ANY Ys-MacBook-Pro.local, &quot;OU&quot; question SRV 0</td>
</tr>
<tr>
<td>31</td>
<td>fe80::1c9f:22de:19a:ff82:ff</td>
<td></td>
<td>MNIS</td>
<td>192 Standard query 0x0000 ANY Ys-MacBook-Pro.local, &quot;OU&quot; question ANY Ys-MacBook-Pro.local, &quot;OU&quot; question SRV 0</td>
</tr>
<tr>
<td>32</td>
<td>172.27.21.251</td>
<td>224.0.0.251</td>
<td>MNIS</td>
<td>291 Standard query response 0x0000 TXT, cache flush PTR »,printer_»,local PTR My Test »,printer_»,local SRV, cache</td>
</tr>
<tr>
<td>33</td>
<td>fe80::1c9f:22de:19a:ff82:ff</td>
<td></td>
<td>MNIS</td>
<td>311 Standard query response 0x0000 TXT, cache flush PTR »,printer_»,local PTR My Test »,printer_»,local SRV, cache</td>
</tr>
<tr>
<td>34</td>
<td>172.27.21.251</td>
<td>224.0.0.251</td>
<td>MNIS</td>
<td>232 Standard query response 0x0000 PTR, cache flush Ys-MacBook-Pro.local PTR, cache flush Ys-MacBook-Pro.local NSEC, ca.</td>
</tr>
<tr>
<td>35</td>
<td>fe80::1c9f:22de:19a:ff82:ff</td>
<td></td>
<td>MNIS</td>
<td>252 Standard query response 0x0000 PTR, cache flush Ys-MacBook-Pro.local PTR, cache flush Ys-MacBook-Pro.local NSEC, ca.</td>
</tr>
<tr>
<td>36</td>
<td>172.27.21.251</td>
<td>224.0.0.251</td>
<td>MNIS</td>
<td>291 Standard query response 0x0000 TXT, cache flush PTR »,printer_»,local PTR My Test »,printer_»,local SRV, cache</td>
</tr>
<tr>
<td>37</td>
<td>fe80::1c9f:22de:19a:ff82:ff</td>
<td></td>
<td>MNIS</td>
<td>311 Standard query response 0x0000 TXT, cache flush PTR »,printer_»,local PTR My Test »,printer_»,local SRV, cache</td>
</tr>
<tr>
<td>38</td>
<td>172.27.21.251</td>
<td>224.0.0.251</td>
<td>MNIS</td>
<td>469 Standard query response 0x0000 PTR, cache flush Ys-MacBook-Pro.local PTR, cache flush Ys-MacBook-Pro.local TXT, cac.</td>
</tr>
<tr>
<td>39</td>
<td>fe80::1c9f:22de:19a:ff82:ff</td>
<td></td>
<td>MNIS</td>
<td>449 Standard query response 0x0000 PTR, cache flush Ys-MacBook-Pro.local PTR, cache flush Ys-MacBook-Pro.local TXT, cac.</td>
</tr>
</tbody>
</table>
Outline

- Admin and recap
- Infrastructure supporting multiple network servers
  - Overview
  - Domain name system (DNS)
    - Basic DNS
    - mDNS and DNS-SD
    - Linda (detour)
Multiple Service Discovery Protocols and Systems

- **Grapevine**: Xerox PARC early 1980’s Birrell, Levin, Needham, Schroeder *CACM* 25(1)

- The MAIN name system, an exercise in centralized computing, Deegan, Crowcroft and Warfield, *ACM SIGCOMM* 35(5), Oct 2005

- **ZooKeeper** as a name/service discovery model

- **Kafka** (key-value store) as a name/service discovery model
General Service/Naming Discovery Paradigm: Linda

- “Distributed workspace” by David Gelernter in the 80’s at Yale

- Very influential in naming and resource discovery

- Key issues
  - How to name services/resources
  - How to write/update into name space
  - How to resolve names
The Linda Paradigm

- Naming scheme:
  - arbitrary tuples (heterogeneous-type vectors)

- Name resolution:
  - Nodes write into shared memory
  - Nodes read matching tuples from shared memory
    - exact matching is required for extraction
Linda: Core API

- **out()**: writes tuples to shared space
  - example: `out("abc", 1.5, 12)`.
  - result: insert (“abc”, 1.5, 12) into space

- **read()**: retrieves tuple copy matching arg list (blocking)
  - result: finds (“abc”, 1.5, 12) and sets local variables
    A = 1.5, B = 12. Tuple (“abc”, 1.5, 12) is still resident in space.

- **in()**: retrieves and deletes matching tuple from space (blocking)
  - example: same as above except (“abc”, 1.5, 12) is deleted

- **eval(expression)**: similar to out except that the tuple argument to eval is evaluated
  - example: `eval("ab", -6, abs(-6))` creates tuple (“ab”, -6, 6)
Linda Extension: JavaSpaces

- Industry took Linda principles and made modifications
  - add transactions, leases, events
  - store Java objects instead of tuples
  - a very comprehensive service discovery system

- Definitive book, “JavaSpaces Principles, Patterns, and Practice”
  - 2 of 3 authors got Ph.D.’s from Yale
Outline

- Admin and recap
- Infrastructure supporting multiple network servers
  - Overview
  - Domain name system (DNS)
    - Basic DNS
    - mDNS and DNS-SD
    - Linda (detour)
    - Use DNS for request routing
Example: Amazon Cloud (AWS)

- AWS uses DNS extensively in request routing
  - Route 53 https://docs.aws.amazon.com/route53/index.html
    - Dig +trace zoom.us
  - CloudFront (CDN)
  - Elastic load balancer
    - See ELB example (newer version uses elbv2)
Example: Amazon Elastic Cloud 2 (EC2) Elastic Load Balancing

- As an example of using your own servers
  - 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,InstanceProtocol=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,

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

**Result:**

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

Outline

- Admin and recap
- Infrastructure supporting multiple network servers
  - Overview
  - Domain name system (DNS)
    - Basic DNS architecture
    - mDNS and DNS-SD
    - Linda (detour)
    - Use DNS for request routing (RR)
      - Amazon DNS examples
      - Akamai CDN as an example
CDN vs DNS

- CDN provider (e.g., Akamai) provides both
  - **edge** servers for hosting content, e.g., Akamai provides (~325,000) servers in 130 countries and 1435 networks
    - Claims 85% Internet users are within a single "network hop" of an Akamai CDN server.
  - **DNS redirection servers** to select edge servers
    - Closeness to client browser, server load, ... (often secret)

- Content publisher (e.g., cnn)
  - Provides base HTML documents
  - Runs **origin** server(s); but delegates heavy-weight content (e.g., images) to CDN servers

Integrating CDN and Content Publisher

- Originally, embedded content directly use CDN domain names (called URL Akamaization), e.g.,

  <IMG SRC= http://www.provider.com/image.gif >
  => <IMG SRC = http://a661.g.akamai.net/hash/image.gif>

- 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 CDN embedding
Exercise Result: Akamai CDN (Offline)

- `dig +trace cdn.cnn.com a`
  - Replied by ns-47.awsdns-05.com

- `dig +trace ion-ma.turner.com.edgekey.net. a`
  - Replied by ns1-66.akam.net
  - Answer: ion-ma.turner.com.edgekey.net. 21600 IN CNAME e12596.dscj.akamaiedge.net.

- `dig +trace e12596.dscj.akamaiedge.net. A`
  - Replied by n7dscj.akamaiedge.net
  - Answer: e12596.dscj.akamaiedge.net. 20 IN A 96.6.28.230
Exercise

- How large is the content server pool that the DNS request routing server need to monitor, if the name is the format `<customer-id>.akamaiedge.net`, e.g., e12596.akamaiedge.net?

- How may you scale the selection process?
Summary: CDN Load-Balancing/Request Routing

- Potential basic techniques (Akamai-like design)
  - One-level of indirection (aliasing, cname), to hide CDN so that publisher completely controls "branding"
  - Hierarchy (multiple levels) DNS names, e.g.,
    - first level: query name server akamaiedge.net. to decide region according to (dscj, clientIP)
    - next level: query e12596 of region name server to choose specific server
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

Day

Night
Server Diversity for Yahoo

Majority of PL nodes see between 10 and 50 Akamai edge-servers.

Nodes far away from Akamai hot-spots.
Redirection Effectiveness: Measurement Methodology

Akamai Low-Level DNS Server

Planet Lab Node

9 Best Akamai Replica Servers

ping

ping
Do redirections reveal network conditions?

- Rank = \( r1 + r2 - 1 \)
  - 16 means perfect correlation
  - 0 means poor correlation

Brazil is poor

MIT and Amsterdam are excellent

Percentage of time Akamai's selection is better or equal to rank
Discussion

- Advantages of request routing using DNS
- Issues of RR using DNS
Outline

- Admin and recap
- Infrastructure supporting multiple network servers
  - Overview
  - Domain name system (DNS)
  - (On-path) request routing
Basic Ideas

- Request router on path (at least initially)
  - RR can directly monitor the status of servers or in addition pull server status
  - RR can inspect request content when selecting the server

- Many names, a relatively confusing field, e.g.,
  - Amazon comparison of 4 types of load balancers

- Our goal: briefly look at the basic structures, using short examples whenever possible

- A potential topic as a project to further explore
Outline

- Admin and recap
- Infrastructure supporting multiple network servers
  - Overview
  - Domain name system (DNS)
  - (On-path) request routing
    - L7 (Application-layer) request routing
Nginx Example

```
http {
    # ...
    # Configuration of the server group
    upstream appservers {
        zone appservers 64k;
        server appserv1.example.com weight=5;
        server appserv2.example.com:8080 fail_timeout=5s;
        server reservel.example.com:8080 backup;
        server reserve2.example.com:8080 backup;
    }
    server {
        # Location that proxies requests to the upstream group
        location / {
            proxy_pass http://appservers;
            health_check;
        }
        # Location for dynamic configuration requests
        location /api {
            limit_except GET {
                auth_basic "NGINX Plus API";
                auth_basic_user_file /path/to/passwd/file;
            }
            api write=on;
            allow 127.0.0.1;
            deny all;
        }
    }
}
```

```
curl -X POST -d '{
    "server": "10.0.0.1:8089",
    "weight": 4,
    "max_conns": 0,
    "max_fails": 0,
    "fail_timeout": "10s",
    "slow_start": "10s",
    "backup": true,
    "down": true
}' -s 'http://127.0.0.1/api/6/http/upstreams/appservers/servers'

curl -X DELETE -s 'http://127.0.0.1/api/6/http/upstreams/appservers/servers/0'

Sticky cookie srv_id expires=1h domain=.example.com path=/

https://docs.nginx.com/nginx/admin-guide/load-balancer/http-load-balancer/
http {
  resolver 10.0.0.1 valid=300s ipv6=off;
  resolver_timeout 10s;
  server {
    location / {
      proxy_pass http://backend;
    }
  }
  upstream backend {
    zone backend 32k;
    least_conn;
    # ...
    server backend1.example.com resolve;
    server backend2.example.com resolve;
  }
}
Haproxy Example

```
# Haproxy Example Configuration

global
    # global settings here

defaults
    # defaults here

frontend
    # a frontend that accepts requests from clients

backend
    # servers that fulfill the requests

defaults
    timeout connect 10s
    timeout client 30s
    timeout server 30s
    log global
    mode http
    option httplog
    maxconn 3000
```

https://www.haproxy.com/blog/the-four-essential-sections-of-an-haproxy-configuration/
https://cbonte.github.io/haproxy-dconv/2.0/configuration.html
Haproxy Example

```plaintext
global
  # global settings here

defaults
  # defaults here

frontend www.mysite.com
  bind 10.0.0.3:80
  bind 10.0.0.3:443 ssl crt /etc/ssl/certs/mysite.pem
  http-request redirect scheme https unless { ssl_fc }
  use_backend api_servers if { path_beg /api/ }
  default_backend web_servers

backend web_servers
  balance roundrobin
  cookie SERVERUSED insert indirect nocache
  option httpchk HEAD /
  default-server check maxconn 20
  server server1 10.0.1.3:80 cookie server1
  server server2 10.0.1.4:80 cookie server2
```

https://www.haproxy.com/blog/the-four-essential-sections-of-an-haproxy-configuration/
https://cbonte.github.io/haproxy-dconv/2.0/configuration.html
Discussion

- Advantages of L7 RR
- Issues of L7 RR
Outline

- Admin and recap
- Infrastructure supporting multiple network servers
  - Overview
  - Domain name system (DNS)
  - (On-path) request routing
    - L7 (application-layer) request routing
    - L4 (network/transport layer) request routing
A single service IP address (naming abstraction) can be used for a cluster of (physical) servers

- Such a single IP is called a virtual IP address (VIP)

```bash
ipvsadm -A -t 192.168.0.1:80 -s rr
ipvsadm -a -t 192.168.0.1:80 -r 172.16.0.1:80 -m
ipvsadm -a -t 192.168.0.1:80 -r 172.16.0.2:80 -m
```
Network Load Balancing (NLB): Basic Structure

Exercise: Can the LB just forward the request to a chosen real server i?
## Problem

- Although the request router can send to a real server (how?), the packet has VIP as destination address, but a real server may use its own RIP
  - if NLB just forwards the packet from client to a real server, the real server drops the packet
  - even if the real server play, the reply from real server to client has real server IP as source -> client will drop the packet

<table>
<thead>
<tr>
<th>State</th>
<th>Address</th>
<th>Sendbuf</th>
<th>Recvbuf</th>
</tr>
</thead>
<tbody>
<tr>
<td>Listening</td>
<td>{&quot;6789, <em>:</em>&quot;}</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>completed connection queue: C1; C2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>State</th>
<th>Address</th>
<th>Sendbuf</th>
<th>Recvbuf</th>
</tr>
</thead>
<tbody>
<tr>
<td>Established</td>
<td>{128.36.232.5:6789, 198.69.10.10:1500}</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>sendbuf:</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>recvbuf:</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Real Server TCP socket space

\[ S=\text{client} \quad D=\text{VIP} \]
Discussion: How May You Address the Issue?
Outline

- Admin and recap
- Infrastructure supporting multiple network servers
  - Overview
  - Domain name system (DNS)
  - (On-path) request routing
    - L7 (application-layer) request routing
    - L4 (network-layer) request routing
      - NAT
Solution 1: Network Address Translation (NAT)

- Assumption:
  - Real servers use RIPv

- Solution: Network Load Balancer (NLB) does rewriting/translation

- Thus, the NLB is similar to a typical NAT gateway with an additional scheduling function
Example Virtual Server via NAT

Table 1: an example of virtual server rules

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Virtual IP Address</th>
<th>Port</th>
<th>Real IP Address</th>
<th>Port</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCP</td>
<td>202.103.106.5</td>
<td>80</td>
<td>172.16.0.2</td>
<td>80</td>
<td>1</td>
</tr>
<tr>
<td>TCP</td>
<td>202.103.106.5</td>
<td>80</td>
<td>172.16.0.3</td>
<td>8000</td>
<td>2</td>
</tr>
<tr>
<td>TCP</td>
<td>202.103.106.5</td>
<td>21</td>
<td>172.16.0.3</td>
<td>21</td>
<td>1</td>
</tr>
</tbody>
</table>
NLB/NAT Flow

- Request packet
  - Source IP: 206.183.42.40
  - Destination IP: 192.168.0.100

- VIP: 192.168.0.100
  - RIP: 192.168.0.201

- Active load balancer
  - Primary heartbeat
  - Backup heartbeat

- Server 1
  - RIP: 192.168.0.205

- Server n
  - RIP: 192.168.0.206
NLB/NAT Flow
Advantages:
- A single public IP address to be realized, transparently, by a set of real servers with private IP addresses
- Real servers need no change and are not aware of load balancing

Problems
- The network load balancer must be on the critical path and hence may become the bottleneck due to load to rewrite request and response packets
  - Typically, rewriting responses has more load because there are more response packets
Backup Slides
Building Block: Layer 2 Forwarding and Address Resolution Protocol (ARP)

- Each network interface card listens to an assigned MAC address
- To send to a device with a given IP, the sender
  - first translates the IP of the destination to its MAC (device) address
    - The translation is done by the Address Resolution Protocol (ARP)
  - then sends the packet with the given MAC address
    - this is called layer 2 forwarding
Recall: ARP Protocol

ARP is “plug-and-play”:
- nodes create their ARP tables without intervention from net administrator

A broadcast protocol:
- Client broadcasts query frame, containing queried IP address
  - all machines on LAN receive ARP query
- Node with queried IP receives ARP frame, replies its MAC address
Demo: ARP

- `ifconfig -a`
  - to show all interfaces and their MAC addresses
- `arp -a`
  - show the binding between IP address and MAC address
- `Wireshark to capture arp traffic`
ARP Format and Features

- **Query:** Layer 2 (Link layer) broadcast: destination `ff:ff:ff:ff:ff:ff` to be received by all hosts at the same local network
- **Response:** Host with the MAC returns its MAC if it has the query IP
- **Gratuitous ARP:** A host sends this message to update other devices if it changes MAC