Cycle 4:
Cloud Networking and Services
CS 436
Matthew Caesar
What is a cloud?

• IT model enabling shared, ubiquitous access to compute resources

• Cloud networking: access of networking resources in the cloud

• Examples:
  • Cloud-hosted networking (AWS, Azure)
  • Cloud-hosted management (Meraki, Aerohive)
Types of clouds

- Private Cloud
- Public Cloud
- Hybrid Cloud
Types of clouds

• **Software as a Service (SaaS)**
  • Provider provides software, which is centrally hosted
  • Examples: Google Apps, Salesforce, Cisco WebEx

• **Infrastructure as a Service (IaaS)**
  • Customers lease virtual machines they configure themselves
  • Examples: AWS, Azure, IBM SmartCloud

• **Platform as a Service (PaaS)**
  • Customers work with services created/maintained by provider
  • Examples: AWS, Azure, RedHat OpenShift

• **Function as a Service (FaaS)**
  • Customer provides code, Provider provides functions and event triggers
  • Examples: Google Cloud Functions, AWS Lambda, MS Azure Functions
Public Services
Directing Traffic to Services

- Need to provide services to public users (or internal users)

- Challenges: redundancy, low latency, etc

- Concepts:
  - Anycast/Load Balancing
  - NAT
Anycast and Load Balancing

- Direct users to servers
- Done for various reasons
  - For resiliency
  - To balance load
  - To give differential QoS, to target advertisements, to censor...
- Key approaches
  - Wide-area: DNS anycast
  - Wide- and local-area: IP anycast
  - Local-area: load balancing appliance
How to deploy a big Internet svc

• Solution: Put down a server somewhere
• Problems: Too much load, Far away from some clients, Single point of failure, Easy to DoS
Better: Deploy multiple servers

- Solution: Put down several servers
- Benefits: Improve scaling, resilience, proximity
- Question: How to actually implement this
How about: Tell users where to go

- Solution: Tell users which servers to go to
- Problem: How are you going to enforce that
  - Could filter users who don’t listen
  - How to know where users are located?

- california.vine.com
  - 34.206.30.72

- montana.vine.com
  - 169.45.1.3

- florida.vine.com
  - 72.21.206.80
Better idea: Transparent direction

• Give the illusion of single server

• Benefits:
  • Don’t have to coordinate with humans
  • Can force users to servers in desirable ways

• Approach: Anycast
  • DNS Anycast (typically wide-area)
  • IP Anycast (typically wide- and regional-area)
  • Load balancers (typically local-area)
Approach 1: IP Anycast

• Idea: advertise same server address from multiple locations
  • Configure upstream routers to advertise same/overlapping prefix

• Routers in between choose shortest path
  • Directs traffic towards “closest” server
    • Closest in terms of # of hops
Approach 1: IP Anycast

Idea: advertise same server address from multiple locations

svr1.vine.com
34.206.30.72

svr2.vine.com
34.206.30.72

svr3.vine.com
34.206.30.72

vine.com → 34.206.30.72

rtr1

34.206.30.0/24

34.206.30.0/24

34.206.30.0/24

prefix nexthop
34.306.30/24 rtr7

prefix nexthop
34.306.30/24 rtr3

prefix nexthop
34.306.30/24 rtr7

rtr2

rtr3

rtr4

rtr5

rtr6

rtr7

svr2.vine.com
34.206.30.72

svr3.vine.com
34.206.30.72
IP Anycast: Pros and Cons

• IP Anycast has several nice properties
• Fully transparent to higher layers
• ISPs and network can control
• Fast failover for network failures
  • Networks on server events leads to churn

• Cons:
  • Imprecise control (can’t always control how advertisements flow)
Approach 2: DNS Anycast

• DNS is a huge distributed service
  • But you control the name → address mapping for your domain

• Idea: modify Authoritative DNS to return different mappings for different clients
  • Goal: map clients to “nearby” servers

• Inputs to mapping function
  • Mapping of Internet topology
    • Useful for estimating latency between hosts
  • Server loads
  • Etc
How are users assigned to PoPs

• Through DNS
  • Resolving server can hand back an IP address “close” to requesting user

# Chicago
$ dig +short www.linkedin.com
216.52.242.80

# Spain
$ dig @109.69.8.51 +short www.linkedin.com
91.225.248.80
**Approach 2: DNS Anycast**

*Idea: Configure DNS to return subset of servers, based on network properties*

- **svr1.vine.com**: 190.65.7.7
- **svr2.vine.com**: 59.46.88.22
- **svr3.vine.com**: 34.206.30.72

---

**I am 144.6.155.43.**

**What is the IP Address for vine.com?**

**Which of my servers are nearest to 144.6.155.43?**

*Use 34.206.30.72 first, fall back to 59.46.88.22*
Approach 2: DNS Anycast

*Config DNS to return subset of servers, based on network properties*

**Benefits**
- Tighter control over host-server mappings than IP anycast
  - Can map based on latency, server load
- No need to coordinate with network operator
  - Need to convince providers to advertise prefix

**Cons**
- Harder for ISPs to control traffic
- May require additional infrastructure to probe/survey network
- Need to coordinate with DNS provider
- Clients may be far from their local DNS (where request actually comes from)
- Some local DNS servers don’t conform to specifications
Approach 3: Load-Balancing Devices

• In the local area, we have tighter control over forwarding
  • Can force ingress/egress path to be symmetric
  • Forwarding speeds low enough to deeply inspect packets
  • Network latency less of an issue

• Approach: install load-balancing hardware into the network
  • Device monitors servers, uses load balancing algorithm to distribute traffic
Approach 3: Load-Balancing Devices

Internet

client 188.1.100
MAC: M1

IP: 141.149.65.1
MAC: M5

Src IP: 188.1.1.100
Dst IP: 141.149.65.3
Src MAC: M1
Dst MAC: M2

Src IP: 188.1.1.100
Dst IP: 10.10.10.20
Src MAC: M3
Dst MAC: M4

Src IP: 188.1.1.100
Dst IP: 141.149.65.3
Src MAC: M1
Dst MAC: M2

IP: 69.22.55.44
MAC: M4

Src IP: 188.1.1.100
Dst IP: 141.149.65.3
Src MAC: M1
Dst MAC: M2
DNS
Internet Names & Addresses

• Machine addresses: *e.g.*, **192.17.90.136**
  • router-usable labels for machines
  • conforms to network structure (the “where”)

• Machine names: *e.g.*, **instr.engr.illinois.edu**
  • human-usable labels for machines
  • conforms to organizational structure (the “who”)

• The Domain Name System (DNS) is how we map from one to the other
DNS: History

• Initially all host-address mappings were in a hosts.txt file (in /etc/hosts):
  • Maintained by the Stanford Research Institute (SRI)
  • Changes were submitted to SRI by email
  • New versions of hosts.txt periodically FTP’d from SRI
  • An administrator could pick names at their discretion

• As the Internet grew this system broke down:
  • SRI couldn’t handle the load; names were not unique; hosts had inaccurate copies of hosts.txt

• The Domain Name System (DNS) was invented to fix this
  • First name server implementation done by 4 students
Goals

• No naming conflicts (uniqueness)
• Scalable
  • many names
  • (secondary) frequent updates
• Distributed, autonomous administration
  • Ability to update my own (machines’) names
  • Don’t have to track everybody’s updates
• Highly available
• **Lookups are fast**
• Non-goal?: perfect consistency
Key idea: Hierarchy

Three intertwined hierarchies

• Hierarchical namespace
  • As opposed to original flat namespace

• Hierarchically administered
  • As opposed to centralized

• (Distributed) hierarchy of servers
  • As opposed to centralized storage
Hierarchical Namespace

- "Top Level Domains" are at the top
- Domains are subtrees
  - E.g: .edu, illinois.edu, cs.illinois.edu
- Name is leaf-to-root path
  - www.caesar.cs.illinois.edu
- Depth of tree is arbitrary (limit 128)
- Name collisions trivially avoided
  - each domain is responsible
Hierarchical Administration

- A zone corresponds to an administrative authority that is responsible for that portion of the hierarchy

- E.g., UIUC controls names: *.illinois.edu and *.cs.illinois.edu

- E.g., CS controls names: *.cs.illinois.edu
Server Hierarchy

• Top of hierarchy: Root servers
  • Location hardwired into other servers

• Next Level: Top-level domain (TLD) servers
  • .com, .edu, etc.
  • Managed professionally

• Bottom Level: Authoritative DNS servers
  • Actually store the name-to-address mapping
  • Maintained by the corresponding administrative authority
Server Hierarchy

- Each server stores a (small!) subset of the total DNS database

- An authoritative DNS server stores “resource records” for all DNS names in the domain that it has authority for

- Each server needs to know other servers that are responsible for the other portions of the hierarchy
  - Every server knows the root
  - Root server knows about all top-level domains
DNS Root

- Located in Virginia, USA
- How do we make the root scale?

Verisign, Dulles, VA
DNS Root Servers

- 13 root servers (labeled A-M; see http://www.root-servers.org/)

- A Verisign, Dulles, VA
- C Cogent, Herndon, VA
- D U Maryland College Park, MD
- G US DoD Vienna, VA
- H ARL Aberdeen, MD
- J Verisign
- B USC-ISI Marina del Rey, CA
- F Internet Software Consortium
  - Palo Alto, CA
- E NASA Mt View, CA
- K RIPE London
- I Autonomica, Stockholm
- M WIDE Tokyo
- L ICANN Los Angeles, CA
DNS Root Servers

- 13 root servers (labeled A-M; see http://www.root-servers.org/)
- Replicated via any-casting
Anycast in a nutshell

• Routing finds shortest paths to destination

• If several locations are given the same address, then the network will deliver the packet to the closest location with that address

• This is called “anycast”
  • Very robust
  • Requires no modification to routing algorithms
Using DNS (Client/App View)

- Two components
  - Local DNS servers
  - Resolver software on hosts

- Local DNS server (“default name server”)
  - Usually near the endhosts that use it
  - Hosts configured with local server address (e.g., /etc/resolv.conf) or learn server via a host configuration protocol (e.g., DHCP)

- Client application
  - Obtain DNS name (e.g., from URL)
  - Do `gethostbyname()` to trigger DNS request to its local DNS server
How Does Resolution Happen?

Host at cis.poly.edu wants IP address for www.illinois.edu
Recursive vs. Iterative Queries

- **Recursive** query
  - Ask server to get answer for you
  - E.g., request 1 and response 8
- **Iterative** query
  - Ask server who to ask next
  - E.g., all other request-response pairs
- Usually recursive is disabled (subject to DoS attacks)
DNS Records

• DNS info. stored as resource records (RRs)
  • RR is (name, value, type, TTL)

• Type = A: (⇒ Address)
  • name = hostname
  • value = IP address

• Type = NS: (⇒ Name Server)
  • name = domain
  • value = name of dns server for domain
DNS Records (cont’d)

- **Type = CNAME:** (Canonical **NAME**)
  - name = hostname
  - value = canonical name

- **Type = MX:** (Mail **Xchanger**)
  - name = domain in email address
  - value = canonical name(s) of mail server(s)

- **Example:**

```
<table>
<thead>
<tr>
<th>NAME</th>
<th>TYPE</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>bar.example.com.</td>
<td>CNAME</td>
<td>foo.example.com.</td>
</tr>
<tr>
<td>foo.example.com.</td>
<td>A</td>
<td>192.0.2.23</td>
</tr>
</tbody>
</table>
```
Inserting Resource Records into DNS

- Example: just created company “FooBar”
- Get a block of address space from ISP
  - Say 212.44.9.128/25

- Register foobar.com at registrar (e.g., Network Solutions)
  - Provide registrar with names and IP addresses of your authoritative name server (primary and secondary)
  - Registrar inserts RR pairs into the .com TLD server:
    - (foobar.com, dns1.foobar.com, NS)
    - (dns1.foobar.com, 212.44.9.129, A)

- You store appropriate records in your server dns1.foobar.com:
  - e.g., type A record for www.foobar.com
  - e.g., type MX record for foobar.com
DNS Protocol

• Query and Reply messages; both with the same message format
  • header: identifier, flags, etc.
  • plus resource records
  • see text/section for details

• Client--server interaction on UDP Port 53
  • Spec supports TCP too, but not always implemented
Goals -- how are we doing?

• No naming conflicts (uniqueness)
• Scalable
• Distributed, autonomous administration
• Highly available
• Fast lookups
DNS Caching

• Performing all these queries takes time
  • And all this before actual communication takes place
  • E.g., 1-second latency before starting Web download

• Caching can greatly reduce overhead
  • The top-level servers very rarely change
  • Popular sites (e.g., www.cnn.com) visited often
  • Local DNS server often has the information cached

• How DNS caching works
  • DNS servers cache responses to queries
  • Responses include a “time to live” (TTL) field
  • Server deletes cached entry after TTL expires
Negative Caching

• Remember things that don’t work
  • Misspellings like www.cnn.comm and www.cnnn.com
  • These can take a long time to fail the first time
  • Good to remember that they don’t work
  • … so the failure takes less time the next time around

• But: negative caching is optional
  • And not widely implemented
Reliability

• DNS servers are replicated (primary/secondary)
  • Name service available if at least one replica is up
  • Queries can be load-balanced between replicas

• Usually, UDP used for queries
  • Need reliability: must implement this on top of UDP

• Try alternate servers on timeout
  • Exponential backoff when retrying same server

• Same identifier for all queries
  • Don’t care which server responds
Important Properties of DNS

Administrative delegation and hierarchy results in:

- Easy unique naming
- “Fate sharing” for network failures
- Reasonable trust model
- Caching lends scalability, performance
DNS provides Indirection

- Addresses can change underneath
  - Move www.cnn.com to 4.125.91.21
  - Humans/Apps should be unaffected

- Name could map to multiple IP addresses
  - Enables
    - Load-balancing
    - Reducing latency by picking nearby servers

- **Multiple names** for the same address
  - E.g., many services (mail, www, ftp) on same machine
  - E.g., aliases like www.cnn.com and cnn.com

- But, this flexibility applies only within domain!
DNSSEC

• What if a DNS server gets compromised/intercepted?
  • Can redirect users to other sites
  • E.g., fake bank site that learns your password
  • “Cache poisoning” – filling DNS caches with bad information

• Solution: DNSSEC
  • Digitally sign IP-to-name mappings
    • Signature is cryptographic hash of DNS data
  • If signatures can’t be verified, error to user
  • Doesn’t mitigate DoS or provide data confidentiality
DNSSEC Lookup

1. User makes request for a .AFRICA domain
2. ISP verifies the ROOT's DS key
3. ROOT points the ISP to the .AFRICA TLD and gives the ISP the .AFRICA DS Key
4. ISP verifies .AFRICA's DS Key
5. .AFRICA points the ISP to the EXAMPLE.AFRICA SLD and gives the ISP the EXAMPLE.AFRICA DS Key
6. ISP verifies EXAMPLE.AFRICA's SLD DS key
7. Requested SLD information is retrieved and sent back to ISP
8. ISP sends SLD information back to user
9. User accesses trusted EXAMPLE.AFRICA domain

Figure 1: Example of Trust Chain Using DNSSEC
NAT
NAT: Network Address Translation

All datagrams leaving local network have same single source NAT IP address: 138.76.29.7, different source port numbers

Datagrams with source or destination in this network have 10.0.0/24 address for source, destination (as usual)
NAT: Network Address Translation

• **Motivation**: local network uses just one IP address as far as outside word is concerned:
  • no need to be allocated range of addresses from ISP: just one IP address is used for all devices
  • can change addresses of devices in local network without notifying outside world
  • can change ISP without changing addresses of devices in local network
  • devices inside local net not explicitly addressable, visible by outside world (a security plus).
NAT: Network Address Translation

Implementation: NAT router must:

- **outgoing datagrams: replace** (source IP address, port #) of every outgoing datagram to (NAT IP address, new port #) . . . remote clients/servers will respond using (NAT IP address, new port #) as destination addr.

- **remember (in NAT translation table)** every (source IP address, port #) to (NAT IP address, new port #) translation pair

- **incoming datagrams: replace** (NAT IP address, new port #) in dest fields of every incoming datagram with corresponding (source IP address, port #) stored in NAT table
NAT: Network Address Translation

1: host 10.0.0.1 sends datagram to 128.119.40, 80

2: NAT router changes datagram source addr from 10.0.0.1, 3345 to 138.76.29.7, 5001, updates table

3: Reply arrives dest. address: 138.76.29.7, 5001

4: NAT router changes datagram dest addr from 138.76.29.7, 5001 to 10.0.0.1, 3345
NAT: Network Address Translation

• 16-bit port-number field:
  • 60,000 simultaneous connections with a single LAN-side address!

• NAT is (was?) controversial:
  • routers should only process up to layer 3
  • violates end-to-end argument
    • NAT possibility must be taken into account by app designers, eg, P2P applications
  • address shortage should instead be solved by IPv6
Types of NAT

- PAT: Port-Address Translation
  - Uses port field in addition to address field

- Restricted NAT
  - An external host $H_e$ can only contact an internal host $H_i$ if $H_i$ has previously sent packets to $H_e$
  - See also “hole-punching” source sends packet to create association, informs association to peer (may have to guess port assignment alg)

- Port forwarding
  - Install static mapping in NAT, to make services on internal network publicly reachable
    - E.g., port 80 on NAT address gets mapped to machine X, port 8080

- Carrier-grade NAT
  - NAT run by an ISP, as opposed to an enterprise/end-user
  - NAT444: cust private nw $\rightarrow$ provider private nw $\rightarrow$ public nw
NAT Device Implementation

Pool
4.1.36.42

Translation table

NAT