INTRODUCTION TO DNS

or, how computers handle our predilection for names over IP addresses

TECH SPECS

- Instructor:

  Nathan Isburgh

  instructor@edgecloud.com

- 4 hours, lecture/lab format
- Two 10 minute breaks

WHAT?

- DNS - Domain Name System - A system of servers and clients which map **host names** to **IP addresses** and vice versa.
- IP Address - Identification information for nodes on a network - “dotted quad”
- Host name - human-friendly name for identification of nodes on a network
- Example:
  - www.rackspace.com -> 72.32.191.88

SO HOW DOES THIS WORK?

- Queries!
- A client of the DNS service puts together a query to send to a name server. This query formats the request in a consistent manner so the DNS server can process and answer with a response.
- What’s a query? A simple query might be requesting the IP address associated with a domain name, as in the previous example. Other queries might ask for the mail exchangers of a domain, to deliver an email message.

DNS CLIENTS

- Really aren't all that interesting. Basically...
- A user wants to go to google.com, so the web browser asks for the IP address of google.com via the *resolver* library. The resolver library takes the request, forms a datagram message and shoots it out to the configured nameserver for the machine. The nameserver works all the magic and sends back the answer, which is then parsed by the resolver library and handed back to the application.
- Even still, DNS is quite elegant in it's solution to the naming problem

QUICK HISTORY

- In the late 60's, ARPAnet was formed by [D]ARPA. This infantile network interconnected a few dozen computer sites around the country, and naming was not a problem because the network was mostly static.
- Then in the 70's, ARPAnet grew to a few hundred hosts, and naming had to be centralized. The Stanford Research Institute (SRI) hosted a file called HOSTS.TXT, which was a flat text database of all machines connected on the ARPAnet.
- Administrators periodically downloaded the HOSTS.TXT file to update their local host to address mappings.

MORE QUICK HISTORY

- Changes to hostnames and addresses were emailed to the SRI Network Information Center (NIC).
- This setup was tolerable while ARPAnet was still a small, friendly community of nerds.
- Then TCP/IP came along in the early 80's, and the ARPAnet population *exploded*.
- All of a sudden, HOSTS.TXT maintained by one group at SRI-NIC was untenable.

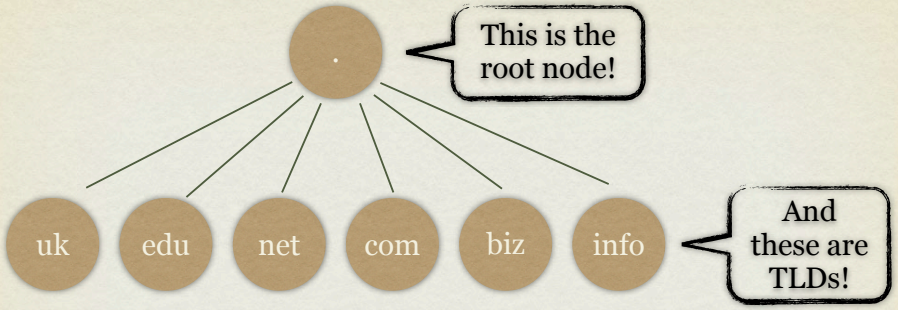
DNS TO THE RESCUE!

- Paul Mockapetris was charged with designing a replacement for the rapidly failing HOSTS.TXT hack.
- In 1984, Paul released two Requests For Comments (RFC) detailing the design of the Domain Name System. Later updated by RFC's 1034 and 1035, as well as many other RFC's covering security and other enhanced features of DNS.

DNS FUNDAMENTALS

- DNS has a very strong hierarchal structure to it, similar to a filesystem - an inverted tree.
- At the top is the trunk of the tree, known as the root node, written as a single dot: .
- Directly beneath the root node are the Top Level Domains (TLDs). The TLDs provide the top-most level of organization of the name space. Think of them as groups or buckets which all domain names fall in to. Some of the TLDs:
  - com, net, org, edu, biz, us, uk, info, travel, museum

PICTURE TIME!

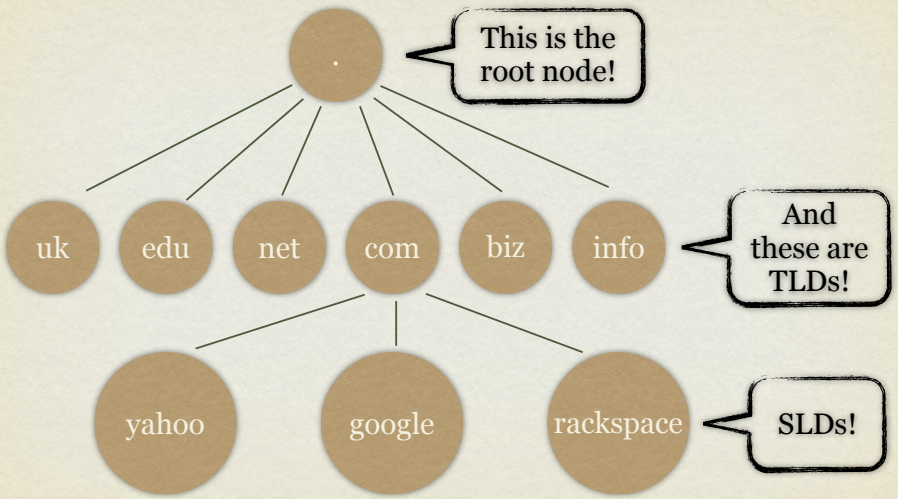


- Now imagine the traditional Unix filesystem.. Put / in for the root node, and usr, bin, etc, var, lib, mnt for the TLDs.
- See the similarity?

SECOND LEVEL DOMAINS

- After the TLD comes the second level domain, only important because this is generally what is “owned” when someone registers a domain name. The combination of the second level domain with the top level domain. Examples:
  - yahoo.com
  - google.com
  - rackspace.com

MORE PICTURES!



DNS

- The similarities between the Domain Name System and a Unix filesystem are numerous:
  - Siblings can not have the same name
  - Inverted tree, ‘trunk’ is root
  - Each node in the tree is a root for a subtree within the overall tree. This is known as a directory in filesystems, and a domain (or subdomain depending on preference) in DNS.

DNS

- Just like a pathname identifies a location in the filesystem, a domain name identifies a location within the DNS database.
  - In a filesystem, the pathname is specified starting from the root node, separating each directory with a ‘/’, as: /usr/local/bin
  - In DNS, the domain is specified in reverse, from the lowest level, back to the root, separating each domain with a ‘.’, as: web1.rackspace.com.
- The final ‘.’ on a domain name is not required*, and if it is not present, is generally assumed to be the root node.

WHY?

- Prevents name collisions! Every node is guaranteed unique!
- Delegation of authority!
- The concepts of sub trees and domains allows for simple demarcation of authorities.
- This is an absolutely critical aspect for the design of DNS. No longer is there one centralized authority for all naming needs (like the old SRI-NIC model).

DELEGATION

- Now, the DNS administrators can delegate authority for a particular *domain* to a separate entity.
- For example, the *rackspace* domain was delegated by the *com* domain to Rackspace. Now Rackspace manages the entire namespace at and below rackspace.com.
- This is great for everyone - the DNS admins no longer have to handle change requests for rackspace.com. Further, since the domain was delegated, they don't have to hold the domain database for that branch of the tree. Requests will be passed along to the Rackspace nameservers!

DELEGATION

- Rackspace has complete control over the rackspace.com domain.
- A host on rackspace.com, called ‘www’, provides a website.
- Perhaps the departments are split up, so you have names like:
  - cloud.rackspace.com
  - managed.rackspace.com
  - apps.rackspace.com

DELEGATION

- Organization is great. Delegation is better. If cloud.rackspace.com starts getting too big for rackspace to handle themselves (as a company-wide IT entity), cloud could be *delegated* and then the Cloud department would be in charge of their own little subdomain.
- Then if Cloud needed to add a new node, wiki, they could do it themselves: wiki.cloud.rackspace.com.
- The authoritative information for wiki would be found on the Cloud nameservers, not the Rackspace nameservers.

AUTHORITY

- “Respect my authori-tie!”
- Authority decides who is responsible for a given domain.
- When a nameserver responds to a request, it can provide *authoritative* or *non-authoritative* information
- *Authoritative* information means that the nameserver is directly responsible for that information.
- *Non-Authoritative* information is what a nameserver passes on from some other name server. Which brings us back to queries..

DNS QUERIES IN DEPTH

- So what really happens when a user requests www.rackspace.com in their web browser?
- Let’s diagram this out on the board..

ROOT NAMESERVERS

- The root nameservers are the starting point for the domain name system. Root, see? Get it? Bah.
- The root nameservers know where all of the top level domain name servers are located. And in some cases, certain root nameservers are authoritative for a TLD.
- As you can see, the root nameservers are the most important parts of the domain name system. There are currently 13 root name servers spread around the globe, on varying networks and some operating in multiple geographic locations through anycast routing.

(NON)RECURSIVE QUERIES

- A recursive query asks a nameserver for the final answer to a question, and relies on the name server to perform the necessary additional queries to provide that information.
- A non-recursive or iterative query simply asks a nameserver to provide the best information it has on the query.
- In the example earlier, the initial request from libresolv to the nameserver was recursive. All other requests were iterative. This is the standard method of operation.

SO, THAT SEEMS INEFFICIENT...

- You're right, if that's all there was, DNS would be terribly slow, inefficient, and a **ridiculous** strain on the root nameservers.
- Fortunately, that problem was considered all the way back at the design phase, and so *caching* is used everywhere.
- Caching is the technique of storing a local copy of some piece of information in order to have faster access to it. There are many ways to keep the cache copy "up to date" with the real copy, and in DNS that is achieved by the Time To Live (TTL).

CACHING AND TTL

- When a nameserver receives a response to a query, it *caches* that response in order to improve performance on future queries that might need that information.
- Consider, in the previous example, if someone had earlier requested yahoo.com. Then when a request for www.rackspace.com is made, the nameserver would have already cached the nameserver information for the com domain, and would not need to query the root nameservers.
- The cached information is held for the Time To Live - maybe 5 hours, maybe 3 seconds - the TTL is decided by the authoritative server.

TTL

- As with everything, setting the TTL is a balancing decision.
- A shorter TTL provides for a more rapid propagation of DNS changes. But, it also increases load on your nameservers, since the cache expires quickly and will have to be refreshed by other nameservers more often.
- A longer TTL is the inverse - lower load on your nameserver, but longer propagation times.

DONE!

- Well, actually, we're just getting started. But that's the Big Picture for DNS host to address resolution.
- What about the reverse? Address to host resolution?

REVERSE MAPPING

- Given an IP address, how to find the domain name? Search the entire database one at a time? Heck no, not possible.
- in-addr.arpa!
- This is a special domain used to provide the reverse mapping.
- Each octet in the dotted quad of an IP address is at a level in the domain tree for in-addr.arpa.

IN-ADDR.ARPA

- From in-addr.arpa, the next level can be one of 256 domains, representing the first octet of the IP address.
- The next level down is the second octet, and so forth. So:
  - 192.168.1.100
  - 100.1.168.192.in-addr.arpa
- Why is it backwards?

IN-ADDR.ARPA

- Simple! IP address space is dolled out in blocks, with the address for the block representing the first portion of the IP address, and the address for the specific host in that block comprising the remaining part of the IP address.
- If the in-addr.arpa domain wishes to delegate domain authority to the corresponding users of the IP blocks, the first level below the in-addr.arpa domain must be the first octet of the IP address, since IP address space is *delegated* from left to right. Think of the inverse tree and domains.
- The other way would “work” as far as DNS is concerned, but delegation of the in-addr.arpa space would be impossible.

YAY THEORY

- Had enough theory?
- Good. Let’s see some tools to play with name servers!

NSLOOKUP & DIG

BIND

Berkeley Internet Name Domain

ZONES AND DOMAINS

- The distinction is small, but important to make clear:
  - A domain represents all of the various branches of a particular subtree in the DNS database, regardless of delegation. For example, the rackspace.com domain would encompass everything, cloud.rackspace.com, www.rackspace.com and apps.rackspace.com.
  - A zone represents all of the information a nameserver might be authoritative for - which includes other domains. So the rackspace.com zone would include www.rackspace.com, but not cloud.rackspace.com or apps.rackspace.com, since those are delegated and represent their own zones.

CONFIG FILES

- BIND basically has two types of configuration files:
  - BIND configuration file, specific to BIND and it's features
  - Database files, or zone files, which contain DNS resource records used to describe all of the DNS information needed in a domain
- Next, a discussion of resource records and the six most common ones: SOA, NS, A, CNAME, PTR, MX

RESOURCE RECORDS

- A resource record contains the DNS *information* about a domain.
- There are several types of resource records, including address records (A), mail exchangers (MX) and name servers (NS).
- Every domain has at least 2 resource records, an SOA and an NS. But that wouldn't be a very useful domain, so there are usually quite a few more records, defining addresses, mail exchangers, canonical names and more.

SOA

- Start Of Authority: This resource record defines authority for a zone.

```
domain IN SOA nameserver adminemail (
      serial
      refresh
      retry
      expire
      negativettl
)
```

We'll discuss these later!

SOA EXAMPLE

```
rackspace.com. IN SOA ns1.rackspace.com. dnsadmin.rackspace.com. (
   2009123004   ; Serial number
   3h           ; Refresh interval
   1h           ; Retry interval
   1w           ; Expires
   1h           ; Negative TTL
)
```

NS

- Name Server: Defines authoritative nameservers for the zone.

  zone **IN NS** *nameserver*

- Example:

  `rackspace.com. IN NS ns1.rackspace.com.`

A

- Address: Maps hostnames to IP addresses

  hostname **IN A** *ipaddress*

- Example:

  ```
  ns1.rackspace.com. IN A 192.168.1.5
  ```

CNAME

- Canonical Name: Maps *alias* hostnames to their *canonical* counterparts.

  aliashostname **IN CNAME** *canonicalhostname*

- Example:

  ```
  ns.rackspace.com. IN CNAME ns1.rackspace.com.
  ```

CANONICAL?

- In layman terms, canonical is another way of saying “real”, “absolute” or “official”.
- So a canonical name refers to the “official” name for a host.
- Creating an alias for a host means that you have to decide on the canonical name, which would be some A record.
- When a resolver performs an A query on a CNAME, the nameserver looks up the canonical name to find out the address to return.

WHY NOT JUST USE A?

- There are basically two reasons to use CNAME records instead of just lots of A records.
  - First, ease of maintenance. If you need 10 names for one machine, defining them with CNAME is easiest if you then need to change the IP address of the machine. Only one change instead of 11.
  - Second, *canonicalization*. Some services, most notably sendmail, will convert all aliases into their canonical names. This simplifies mail configuration.

PTR

- Pointer: Maps an IP address back to a name, specifically the canonical name.

  ipaddressdomain **IN PTR** *canonicalhostname*

- Example:

```
5.1.168.192.in-addr.arpa. IN PTR ns1.rackspace.com.
```

PTR

- Remember, there is only one PTR record for a given IP address, and it should always point to the *canonical hostname*.
- Also, as a side note, make sure your mail servers map both directions exactly. This is important for proper authentication:
  - `mailer.mydomain.com. -> 192.168.1.50`
  - `50.1.168.192.in-addr.arpa. -> mailer.mydomain.com`

MX

- Mail Exchanger: Defines hosts responsible for incoming email for the named zones.

  ```
  zone IN MX preference mailhandler
  ```

- Example:

  ```
  rackspace.com. IN MX 10 mail1.rackspace.com.
  ```

MX RECORDS

- MX records allow for enhanced mail routing functionality.
- When an email is shipped out, the server canonicalizes the delivery address. So, for example, bob@ns.rackspace.com becomes bob@ns1.rackspace.com.
- Then the server looks up the MX records for ns1.rackspace.com, choosing the record with the **lowest** preference and attempting delivery there. If delivery fails, the next lowest is attempted.
- This allows for backup email servers!

BACKUP EMAIL SERVERS

- But, how does the backup server handle forwarding on the message to the primary?
- Preference values to the rescue again!
- The backup server will compare it's own MX preference value with the list, and discard all records at or below it's own level, thereby eliminating the chance for mail delivery loops (assuming everything is set up correctly)
- Quite elegant, no?

PUTTING IT ALL TOGETHER

db.rackspace.com:

```
$TTL 1h
rackspace.com. IN SOA ns1.rackspace.com. dnsadmin.rackspace.com. (
   2009123004   ; Serial number
   3h           ; Refresh interval
   1h           ; Retry interval
   1w           ; Expires
   1h           ; Negative TTL
)

rackspace.com. IN NS ns1.rackspace.com.
ns1.rackspace.com. IN A 192.168.1.5
mail1.rackspace.com. IN A 192.168.1.20
ns.rackspace.com. IN CNAME ns1.rackspace.com.
rackspace.com. IN MX 10 mail1.rackspace.com.
```

TOGETHER...

db.192.168.1:

```
$TTL 1h
1.168.192.in-addr.arpa. IN SOA ns1.rackspace.com. dnsadmin.rackspace.com. (
   2009123004   ; Serial number
   3h           ; Refresh interval
   1h           ; Retry interval
   1w           ; Expires
   1h           ; Negative TTL
)

1.168.192.in-addr.arpa. IN NS ns1.rackspace.com.
5.1.168.192.in-addr.arpa. IN PTR ns1.rackspace.com.
20.1.168.192.in-addr.arpa. IN PTR mail1.rackspace.com.
```

SO WHAT ELSE?

- In addition to the zone files for your domains, you need a couple more zone files to get BIND up and running.
  - Loopback address
  - Root hints

LOOPBACK ADDRESS?

- Someone has to take responsibility for loopback address requests! It's simple enough. `db.127.0.0`:

```
$TTL 1w
0.0.127.in-addr.arpa. IN SOA ns1.rackspace.com. dnsadmin.rackspace.com. (
   2009123004   ; Serial number
   3h           ; Refresh interval
   1h           ; Retry interval
   1w           ; Expires
   1h           ; Negative TTL
)

0.0.127.in-addr.arpa. IN NS ns1.rackspace.com.
1.0.0.127.in-addr.arpa. IN PTR localhost.
```

ROOT HINTS!

- The root hints tell the nameserver where those DNS Root Servers are located, so that requests for hosts outside of your authoritative zones can be resolved.
- This one is the simplest to put together. You don't even have to write it!
- Simply ftp the db.cache file from ftp.rs.internic.net/domain
- Let's take a look at that file real quick..

WHEW

- Finally. All of the zone files are put together and ready. Final step? Configuring BIND.
- The config file is generally `/etc/named.conf` (though for the security-paranoid, `/var/named/chroot/etc/named.conf` might be the place...)
- We'll view this one on the live system in a few minutes...

SHORTCUTS

- There are a few handy shortcuts you can use in zone files:
  - $ORIGIN - automatically set to the zone name specified in the bind config file. Can be changed if desired. Automatically appended to any unqualified name in the zone file:
    - `ns1.rackspace.com. IN A 192.168.1.5`
    - `ns1 IN A 192.168.1.5`

SHORTCUTS

- @ Notation: @ is replaced with $ORIGIN. Generally used in SOA:
  - `@ IN SOA ns1.rackspace.com. dnsadmin.rackspace.com. ( ... )`
- Repeating: When RR name is the same as the previous, it can be left out:
  - `mail1.rackspace.com. IN A 192.168.1.50`
  - `IN A 192.168.1.51`

LIVE CONFIG

- Let's take a look at the live config now, to see the shortcuts in use and complete the circle, if you will, on all of the configuration moving parts.
- Logs, too! Long live live logs letting learners learn log life!

SLAVING

- DNS Slave servers are a critical component to the healthy functioning of DNS.
- Slave servers exist to help shoulder the load of requests, and also provide backup resolution services if other servers go down.
- Slave servers are still *authoritative* for zones, the slave part just means that they pull their data from a master server via a **zone transfer** rather than a local file.

SETTING UP A SLAVE SERVER

- Setting up a slave server is actually quite simple.
- In the `named.conf` file, there are some slight changes. Each `zone` (besides 127.0.0/24 and root zones) section will contain:
  - `type slave;`
  - `masters { 192.168.1.5; };`
  - `file "bak.zonename"`

SOA FIELDS

- Now we can talk about those fields in the SOA record!

```
domain IN SOA nameserver adminemail (
    serial
    refresh
    retry
    expire
    negativettl
)
```

SLAVING IN ACTION

- We'll take a look, now, at a slave nameserver for our theoretical DNS environment.

CHANGES

- Changing the information in a zone is a simple process.
  - On the master, edit the appropriate zone file.
  - First, increment the serial number, so slaves will pick up the changes.
  - Second, make your changes.
  - Third, reload the nameserver (rndc reload)
- The slaves will pick up the changes automatically

SUBDOMAINS

- Creating a subdomain is simple enough.
  - The proper zone file is created for the subdomain
  - The parent domain lists NS and A records for the authoritative nameservers of the new subdomain.

DNS SECURITY

- Some key aspects of securing your nameserver
  - Keep BIND up to date!
  - Run BIND in a chroot-ed environment!
  - Use the allow-transfer option to control what IP addresses can perform zone transfers - make sure it's only your slave servers!
  - Consider implementing TSIG or DNSSEC

FIN

Questions?