Configuring Anycast DNS

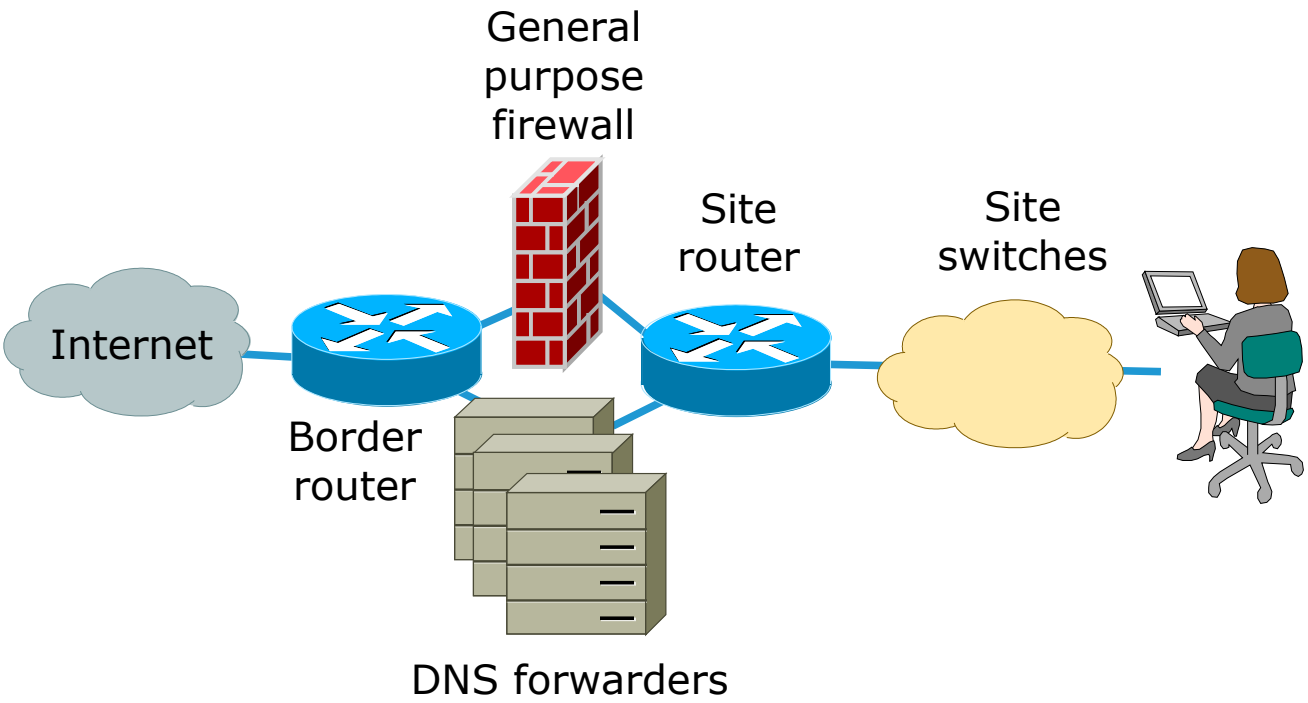
What is the domain name system?

```
vangogh.art.example.edu.au. IN A     1.2.3.4
vangogh.art.example.edu.au. IN AAAA  0102:0304:0506:0708:090a:dead:dead:dead
4.3.2.1.in-addr.arpa.       IN PTR   vangogh.art.example.edu.au.
d.e.a.d.d.e.a.d.d.e.a.d.a.0.9.0.8.0.7.0.6.0.5.0.4.0.3.0.2.0.1.0.ip6.arpa. IN PTR
      vangogh.art.example.edu.au.
www.example.edu.au.         IN CNAME vangogh.art.example.edu.au.
example.edu.au.             IN NS    ns1.example.edu.au.
ns1.example.edu.au.         IN A     1.1.2.1
```

DNS is a database which translates names to other names. it translates names to IPv4 addresses, names to IPv6 addresses, names to names and even names to DNS database servers. Database geeks will see that the DNS is really a hierarchically distributed database, non-geeks can see that each dot marks a point where another database can be used, this allows shared management and sharing of the load. The load is also shared by extensive use of caching: every entry in the DNS has an explicit “time to live” which controls how long the item can be stored in a cache. The optimum TTL value is about three days. The load is also shared by simple load-sharing: multiple DNS servers for each part of the hierarchy are allowed; replication is done by “zone transfers”. Unlike a general-purpose distributed database the DNS has fixed record types: the `A`, `AAAA` and so on in the example. New record types are introduced by upgrading server and client software.

DNS servers fall into four types. *Masters* are where a part of the hierarchy is maintained, by editing “zone files”. *Slaves* use a “zone transfer” to obtain a copy of the master's zone file. The masters and slaves answer queries about their part of the hierarchy (they are found by `NS` records in the part of the hierarchy above them). A *stealth slave* has “zone transfer” but is not listed as an official name server. This is useful for improving performance where many DNS queries are made for a particular zone; for example, mail servers do many DNS lookups, so they will often run a DNS server with a stealth secondary.

Forwarders are DNS servers with no zones. All they do is cache responses to queries. This improves performance, as the walking the entire DNS hierarchy to find a name takes a long time. On AARNet3 resolving a DNS name can be the largest fraction of the time spent displaying a web page. Forwarders also allow lighter host software, as the hosts need not have complicated name caching software to use the DNS. Forwarders also act as an application-specific firewall. If your router blocks all DNS other than through the forwarder then no DNS packet from the outside can reach your hosts on the inside.



Using a DNS forwarder

A forwarder is easy to use. Edit */etc/resolv.conf* adding the IP addresses of up to three forwarders. These are tried in order if the forwarder before it sends no response.

```
search example.edu.au
nameserver 1.1.1.1
nameserver 1.1.1.2
nameserver 1.1.1.3
```

We need to use IP addresses, otherwise there is a chicken-and-egg problem. Hardcoded IP addresses are notoriously difficult to change. And over time we do want to change the addresses; for example, we might grow unfond of having our three DNS forwarders in one subnetwork. How do we deal with that?

```
option domain-name "example.edu.au";
option domain-name-servers 1.1.1.1 1.1.1.2 1.1.1.3;
```

The addresses could be automatically configured, and the Dynamic Host Configuration Protocol can do that. But not all machines can or should use DHCP.

```
ip route add 1.1.1.1/32 dev eth0 gw 1.5.2.254
```

We could put host routes into our routers. But this tends to be difficult to get right, as each and every router needs to be configured correctly, and every host on the same subnet as the DNS forwarder needs to be configured too. Chance of correct deployment is close to zero. And it doesn't work too well on multiple interface machines: in the example if `eth0` goes down then the DNS forwarder is unreachable.

We could inject the route using a routing protocol, such as OSPF. Routing daemons generally will not create a route out of nothing, so let's create a dummy interface on the DNS forwarder to hold the route. For Red Hat we might use */etc/sysconfig/network-scripts/ifcfg-lo:1*.

```
NAME=dns1
DEVICE=lo:1
IPADDR=1.1.1.1
NETMASK=255.255.255.255
BOOTPROTO=none
FIREWALL_MODS=no
USERCTL=no
PEERDNS=no
ONBOOT=no
IPV6INIT=no
NOZEROCONF=yes
```

Injecting a route which binds to a service

Let's use Quagga as the routing daemon. It needs two components installed: the *zebra* daemon to maintain the hosts kernel routing tables, and the *ospfd* component to run the OSPF protocol.

```
QCONFDIR="/etc/quagga"
ZEBRA_OPTS="-A 127.0.0.1 -f ${QCONFDIR}/zebra.conf"
OSPFD_OPTS="-A 127.0.0.1 -f ${QCONFDIR}/ospfd.conf"
```

Zebra needs to be configured to describe the interfaces on the machine. If the machine has multiple interfaces then it should not route between them.

Then OSPF needs to be configured to advertise those interfaces and the networks reachable through them.

```
hostname dns1.example.edu.au:ospfd

password …
enable password …

log file /var/log/quagga/ospfd.log
log record-priority

service advanced-tty
service password-encryption

interface eth0
 ip ospf authentication message-digest
 ip ospf message-digest-key … md5 …
 ip ospf priority 0

interface lo

interface lo.1
 ip opsf cost 1

router ospf
 ospf router-id 1.2.3.4
 auto-cost reference-bandwidth 40000
 passive-interface lo:1
 network 1.1.1.1/32 area 0.0.0.0
 network 1.2.3.4/24 area 0.0.0.0

access-list VTY permit 127.0.0.0 255.0.0.0
ipv6 access-list VTY6 permit ::1/128
line vty
 access-class VTY
 ipv6 access-class VTY6
```

This technique can be used for any service where the IP address needs to be hardcoded. AARNet3 uses it for the addresses of RADIUS gateways used for access point authentication, since we want to be able to place that server anywhere in our network without asking eduroam users to update all their access points.

Note that you can view interface `lo:1` as describing the availability of the service. When `lo:1` is up a 1.1.1.1/32 route is advertised. When lo:1 goes down the 1.1.1.1/32 route is withdrawn.

A side note about route selection and OSPF

A router selects a matching route from a forwarding table by finding the most specific match to the destination address. So 1.1.1.1/32 beats 1.1.1.0/24 beats 0.0.0.0/0. If there are no matching entries in the forwarding table then the packet is discarded and the occasional ICMP Network Unreachable sent to the source address.

If the routing protocol knows of multiple matching destinations then the routing table entry with the lowest cost is installed from the routing table into the forwarding table. So the forwarding table will contain 1.1.1.1/32 cost 1 rather than 1.1.1.1/32 cost 2.

In Quagga the *zebra* daemon populates the kernel's forwarding table from the *zebra* routing table. The *zebra* routing table is populated from the routing information bases maintained by

ospfd, *bgpd* and the other routing protocol daemons.

In OSPF the cost to a destination is calculated by adding the cost of all the links to the destination. The cost is traditionally the clocking delay of the link, with fast ethernet having cost 1 (our configuration has a `reference-bandwidth` which makes a 40Gbps interface have a cost of 1). Larger networks will use explicit costs, often the tens of milliseconds of delay.

OSPF routing is computationally expensive. So on a link which supports broadcasts only two of the routers maintain the OSPF routing information base. We stop *ospfd* from becoming one of these two “designated routers” by using the special `priority` of 0, leaving the job of maintaining the RIB to real routers.

The OSPF router identifier is a 32-bit unique number identifying this OSPF-speaking router. It cannot change without a temporary loss of connectivity. Use the IPv4 address of the major interface, say `eth0`. A real router will have a distinct “loopback” address specifically for this purpose. Linux can do something similar using the `dummy0` interface, but there's no point for this configuration.

The passive-interface keyword means “don't send OSPF packets on this interface.” OSPF only needs to run on the `eth0` interface, so passive out all other interfaces shown by `ifconfig -a`. Note that making an interface passive does not prevent routes about that interface being advertised out of the non-passive interfaces.

A route is advertised by OSPF if it is learned from another OSPF router on another interface (which can't happen here, as we made all of those interfaces passive) or if the exact route is the forwarding table and there is an exactly matching `network` statement. So if the interface `lo:1` is up (and thus has a entry in the routing table) and there is a network statement for the subnet the interface is attached to (in this case `network 1.1.1.1/32`) then the route will be advertised.

Anycast services

DNS forwarders only get short-lived sessions: the client sends a Query, the server sends a Response. This usually uses just two packets; in some bizaare scenarios it uses eight.

So what would happen if we put two of these route-speaking servers on the same network using the same address? In normal operation the traffic would go to the nearest server. If that server was unreachable then the traffic would go to the reachable server with the next highest cost.

Because of the short-lived nature of DNS queries the odds of a routing incident in the middle of a query is low. And if that does happen then the network will be stable when the next address in *resolv.conf* is tried.

Since DNS does not retry, but simply moves to the next entry in *resolv.conf* the DNS server needs to have at least two addresses, say 1.1.1.1 and 1.1.1.2. It makes sense to have 1.1.1.3 as a single machine, but still with an injected route, as the worst case.

The addresses 1.1.1.1 and 1.1.1.2 should flip-flop in OSPF cost between the servers. This

```
interface lo.2
 link-detect
 ip address 1.1.1.2/32
```

means that a second attempt goes to a different server than the first attempt.

```
interface lo.2
 ip opsf cost 2

router ospf
 passive-interface lo:2
 network 1.1.1.2/32 area 0.0.0.0
```

It is important that the `lo:1` and `lo:2` interfaces are only running when the name server is answering queries. A simple *named-anycast* init script which does `service named start`, `ifup lo:1`, `ifup lo:2`, when starting and `ifdown lo:1`, `ifdown lo:2`, `service named stop` when stopping will do that in normal operation.

If *named* dies unexpectedly then there should be some what to detect this and down the `lo:1` and `lo:2` interfaces . This will then cause *ospfd* to withdraw the 1.1.1.1/32 and 1.1.1.2/32 routes and let that traffic go to servers which are still running. This is tricky to detect, and the best strategy at the moment is to periodically test for */var/run/named.pid*, grab the process ID from that file, and make sure that the process ID is in the process table and not a zombie.

The problem for testing for the actual operation of the DNS service is that you could drop all of your anycast servers on a targeted denial of service attack. Better to have one server stand up to the attack and leave users in other parts of the network happily using their different anycast server.

Configuring named

```
redhat# yum install bind-chroot bind-utils
```

Let's configure 1.2.3.4 as a DNS forwarder with the anycast DNS services on 1.1.1.1 and 1.1.1.2. We'll take a stealth slave feed from the master for example.edu.au sicne we know that people will have lots of Queries for that zone and it allows that zone to operate even if the Internet connection drops.

Firstly we need a pretty standard forwarder configuration. Notice how we only limit use of the forwarder to our network 1.0.0.0/8: this prevents some nasty denial of service attacks.

```
// We accept zone transfers from
acl "masters-example-edu-au" {
  1.1.2.1/32;  // ns1.example.edu.au
};
acl "masters" {
  masters-example-edu-au;
};

// We accept Query from
acl "clients" {
  localhost;          // My /etc/resolv.conf
  masters;            // For ease of debugging
```

```
  1.0.0.0/8           // Client networks
};

// We accept Queries about our software version from
acl "versions" {
  localhost;          // Local system users, hopefully only sysadmins
  masters;            // For ease of debugging
};

// We ignore this lot entirely
acl "bogon" {
  0.0.0.0/8;          // Null address
  1.0.0.0/8;          // IANA reserved, popular fakes
  2.0.0.0/8;
  192.0.2.0/24;       // Test address
  224.0.0.0/3;        // Multicast addresses
  // Enterprise networks may or may not be bogus.
  10.0.0.0/8;
  172.16.0.0/12;
  192.168.0.0/16;
};

options {
  directory "/var/named";  // Actually /var/named/chroot/var/named
  dump-file "/var/named/data/cache_dump.db";  // Actually /var/named/chroot/...
  statistics-file "/var/named/data/named_stats.txt";  // Actually /var/named/chroot/...
  // Default security stance is for forwarding
  allow-query {
    clients;
  };
  allow-recusrsion {
    clients;
  };
  allow-transfer {
    none;
  };
  allow-notify {
    none;
  };
  allow-update {
    none;
  };
  blackhole {
    bogon;
  };

  // We are a forwarder, so give the client heaps of info so they
  // don't bother us again.
  recursion yes;
  additional-from-auth yes;
  additional-from-cache yes;
  minimal-responses no;
  // Efficient zone transfers are fine
  transfer-format many-answers;
  // This is the IP address used
  query-source address 1.2.3.4;
  notify-source 1.2.3.4;
  transfer-source 1.2.3.4;

  // Tell everyone my verison, not!
  version none;

  // Performance tuning
```

```
  // 0.5GB for client connections (20KB each)
  recursive-clients 25000;
  // Each TCP client uses a socket, so stop them scarfing the lot
  tcp-clients 2000;
  // Cache of 1GB, half the RAM on this box
  max-cache-size 1000000000;

  // Caching control
  // Lame servers get sin-binned for 30 minutes
  lame-ttl 1800;
  // Positive responses up to 2 weeks
  max-cache-ttl 50400;
  // Negative responses up to 3 hours
  max-ncache-ttl 18000;
  // Stop stupid SOA values invalidating the cache
  // Cache stuff for an hour whatever they say
  min-refresh-time 3600;
  min-retry-time 3600;
};

// Forwarders can't do anything about other's misconfigurations
querylog no;
logging {
  category lame-servers {
    null;
  };
};

// rndc works from localhost
controls {
  inet 127.0.0.1 allow {
    localhost;
  } keys {
    rndckey;
  };
};
include /etc/rndckey;

// Root name servers.
zone "." IN {
  type hint;
  file "named.ca";
};

// Zones with fixed content.
zone "localdomain" IN {
  type master;
  file "localdomain.zone";
  allow-update {
    none;
  };
};
zone "localhost" IN {
  type master;
  file "localhost.zone";
  allow-update {
    none;
  };
};
zone "0.0.127.in-addr.arpa" IN {
  type master;
  file "named.local";
  allow-update {
```

```
    none;
  };
};
zone "0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.ip6.arpa" IN {
  type master;
  file "named.ip6.local";
  allow-update {
    none;
  };
};
zone "255.in-addr.arpa" IN {
  type master;
  file "named.broadcast";
  allow-update {
    none;
  };
};
zone "0.in-addr.arpa" IN {
  type master;
  file "named.zero";
  allow-update {
    none;
  };
};

// We are stealth slaves for these zones
zone "example.edu.au" {
  type slave;
  file "slaves/example.edu.au";
  masters {
    1.1.2.1;       // ns1.example.edu.au
  };
  allow-notify {
    masters-example-edu-au;
  };
  allow-transfer {
    masters-example-edu-au;
  };
};
```

Configuring named for anycast

The alterations to be a anycast server are pretty simple. Listen on the anycast addresses, but my queries and zone transfers should use a non-anycast address (otherwise the responses could go to one of the other anycast servers).

```
options {
  listen-on {
    1.2.3.4;    // eth0, responses to my queries
    1.1.1.1;    // lo:1, anycast cost = 1
    1.1.1.2;    // lo.2, anycast cost = 2
    localhost;  // lo, for my /etc/resolv.conf
  };
  listen-on-v6 {
    none;
  };
  query-source address 1.2.3.4; // eth0
  notify-source 1.2.3.4;        // eth0
  transfer-source 1.2.3.4;      // eth0
};
```

If a anycast DNS server is faulty, how do we tell which one we are using?

```
options {
  server-id hostname;
};
```

Then the user can say

```
$ dig ID.SERVER chaos txt
```

Kim Hawtin, The University of Adelaide.
Glen Turner, Australia's Academic and Research Network.