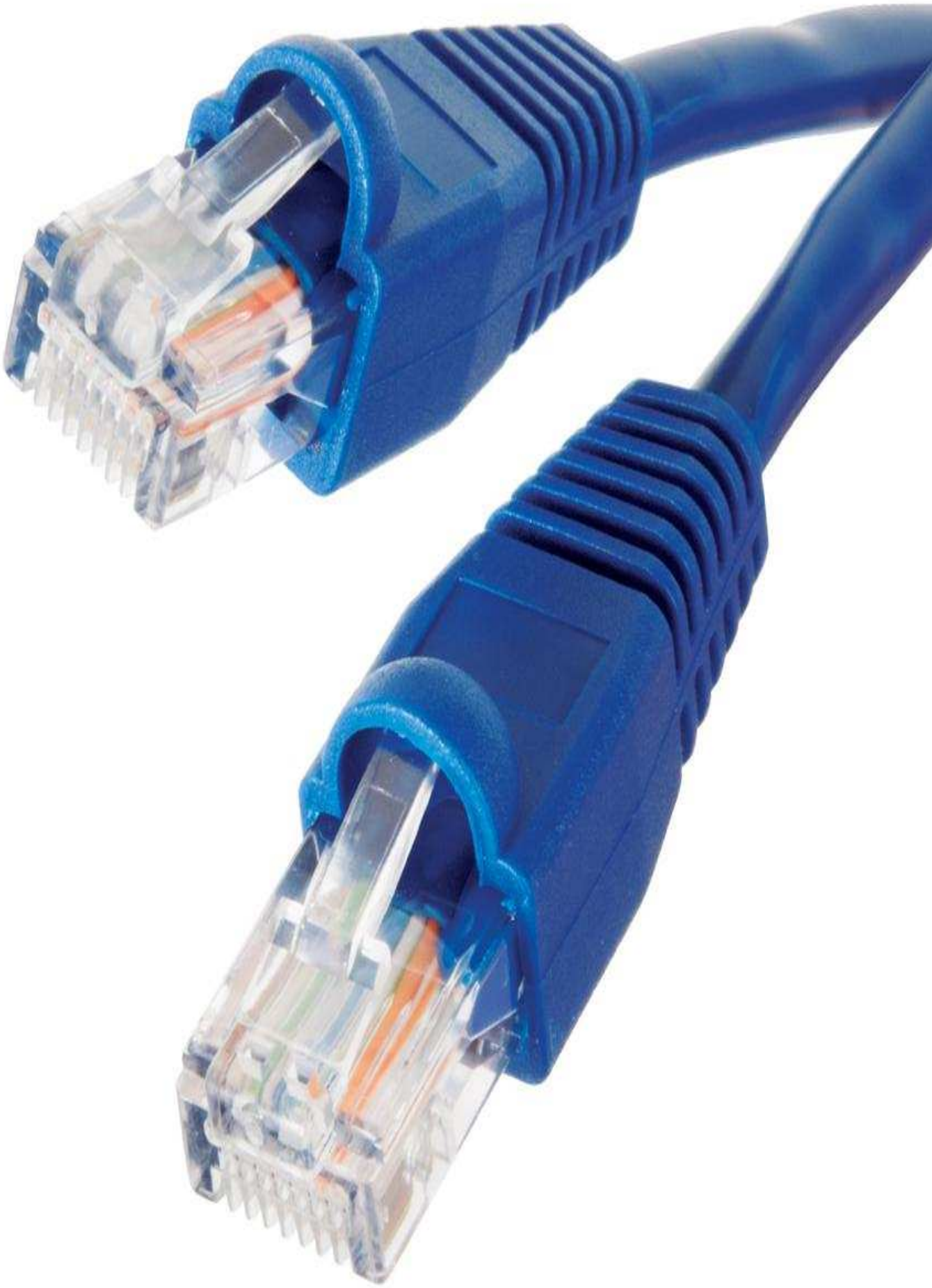


Anycast

ljm

1. Goal

We're introducing anycast as mechanism to increase the availability of a server, in this case a DNS server. We're interested in seeing anycast workin and at the moment we're not so much interested in which routing protocol is being used. We'll use RIP because it is easy. We're also not so much interested in security (at first at least), so forget the enable password, the banner and ssh for the moment. Also, we're using /24 networks for simplification so I don't have to calculate netmasks.

1.1 Network layout

Two data centres are connected through an ethernet-like interface (I don't need the trouble of a serial link in this lab). Both data centres have a DNS server (xenial1 and xenial2) and both have a DNS client (xenial3 and xenial4). Between the datacentres there are routers, and the DNS server is behind a second router.

1.2 Steps

We'll go through the configuration in different steps:

— set-up the network in GNS3

— set-up the DNS-servers on the Debian

— verify that it works with unicast DNS calls

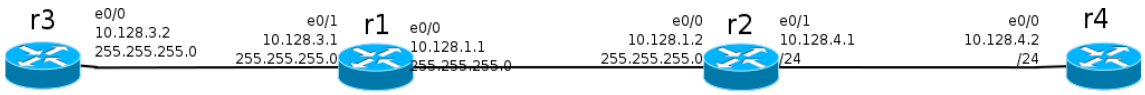
— set-up any-cast

1.3 Versions

1	february 2016	based on debian
2	april 2018	introduced vagrant; precise server and fedora clients
3	february 2020	all machines to xenial; no more tap-devices (all vboxnet)

2. The Network

First set-up a basic network.



I used 3640 as routers. I put a NM-4E in slot 0. Step one is to set-up the IP addresses; as an example for r1:

```
enable
conf t
interface Ethernet0/0
 ip address 10.128.1.1 255.255.255.0
 no shut
!
interface Ethernet0/1
 ip address 10.128.3.1 255.255.255.0
 no shut
!
^Z
wr
```

Once it is all set, verify that you can ping the neighbours:

```
Router#ping 10.128.1.2
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 10.128.1.2, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 4/4/8 ms
Router#ping 10.128.3.2
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 10.128.3.2, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 4/4/8 ms
Router#
```

Next step is to get routing going on all routers.

```
enable
conf t
router rip
version 2
network 10.128.0.0
no auto-summary
```

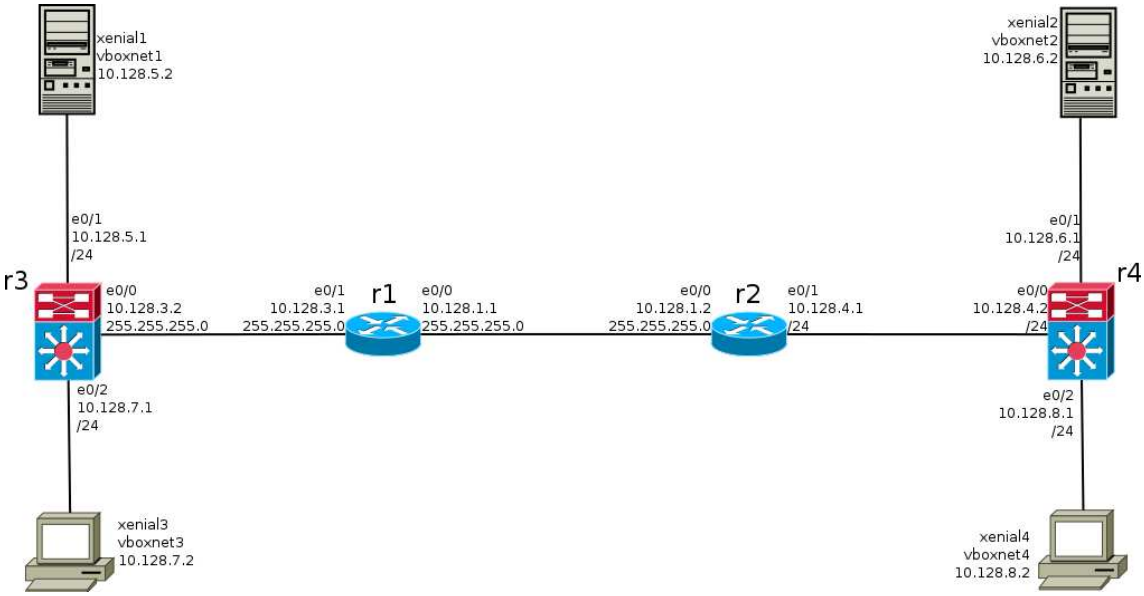
Verify that routing works; on r3:

```
Router#traceroute 10.128.4.2
Type escape sequence to abort.
Tracing the route to 10.128.4.2
  1 10.128.3.1 4 msec 0 msec 4 msec
  2 10.128.1.2 12 msec 4 msec 8 msec
  3 10.128.4.2 12 msec 12 msec *
Router#
```

Next are the servers.

host	function	adapter	IP	network
xenial1	DNS server	eth1	10.128.5.2	vboxnet1
xenial2	DNS server	eth1	10.128.6.2	vboxnet2
xenial3	DNS client	eth1	10.128.7.2	vboxnet3
xenial4	DNS client	eth1	10.128.8.2	vboxnet4

The networks are defined in VirtualBox.



As a first `Vagrantfile` file, use the following content:

```
# -*- mode: ruby -*-
# vi: set ft=ruby :
# Vagrantfile API/syntax version. Don't touch unless you know what you're doing!
VAGRANTFILE_API_VERSION = "2"
Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|
        config.vm.define :xenial1 do |t|
                t.vm.box = "ubuntu/xenial64"
                t.vm.box_url = "file:////links/virt_comp/vagrant/boxes/xenial64"
                t.vm.provider "virtualbox" do |prov|
                        prov.customize ["modifyvm", :id, "--nic2", "hostonly", "--hos-
tonlyadapter2", "vboxnet1" ]
                end
                t.vm.provision "shell", path: "./setup.xenial1.sh"
        end
        config.vm.define :xenial2 do |t|
                t.vm.box = "ubuntu/xenial64"
                t.vm.box_url = "file:////links/virt_comp/vagrant/boxes/xenial64"
                t.vm.provider "virtualbox" do |prov|
                        prov.customize ["modifyvm", :id, "--nic2", "hostonly", "--hos-
tonlyadapter2", "vboxnet2" ]
                end
                t.vm.provision "shell", path: "./setup.xenial2.sh"
        end
        config.vm.define :xenial3 do |t|
                t.vm.box = "ubuntu/xenial64"
                t.vm.box_url = "file:////links/virt_comp/vagrant/boxes/xenial64"
                t.vm.provider "virtualbox" do |prov|
                        prov.customize ["modifyvm", :id, "--nic2", "hostonly", "--hos-
tonlyadapter2", "vboxnet3" ]
                end
                t.vm.provision "shell", path: "./setup.xenial3.sh"
        end
        config.vm.define :xenial4 do |t|
                t.vm.box = "ubuntu/xenial64"
                t.vm.box_url = "file:////links/virt_comp/vagrant/boxes/xenial64"
                t.vm.provider "virtualbox" do |prov|
                        prov.customize ["modifyvm", :id, "--nic2", "hostonly", "--hos-
tonlyadapter2", "vboxnet4" ]
                end
                t.vm.provision "shell", path: "./setup.xenial4.sh"
        end
end
```

The set-up scripts look like this:

`setup.xenial1.sh:`

```
 #!/bin/bash
 echo "Setup xenial1"
 ETH1=$(dmesg | grep -i 'renamed from eth1' | sed -n 's/: renamed from eth1//;s/.*
//p')

 ifconfig $ETH1 10.128.5.2 netmask 255.255.255.0
 route add -net 10.128.0.0 netmask 255.255.0.0 gw 10.128.5.1
 netstat -rn
```

Noteworthy is, that Ubuntu now uses unpredictable interface names, which systemd zealots seem to prefer, and the effort needed to assign the IP address to the right interface.

setup.xenial2.sh:

```
 #!/bin/bash
 echo "Setup xenial2"
 ETH1=$(dmesg | grep -i 'renamed from eth1' | sed -n 's/: renamed from eth1//;s/.*
//p')
 ifconfig $ETH1 10.128.6.2 netmask 255.255.255.0
 route add -net 10.128.0.0 netmask 255.255.0.0 gw 10.128.6.1
 netstat -rn
```

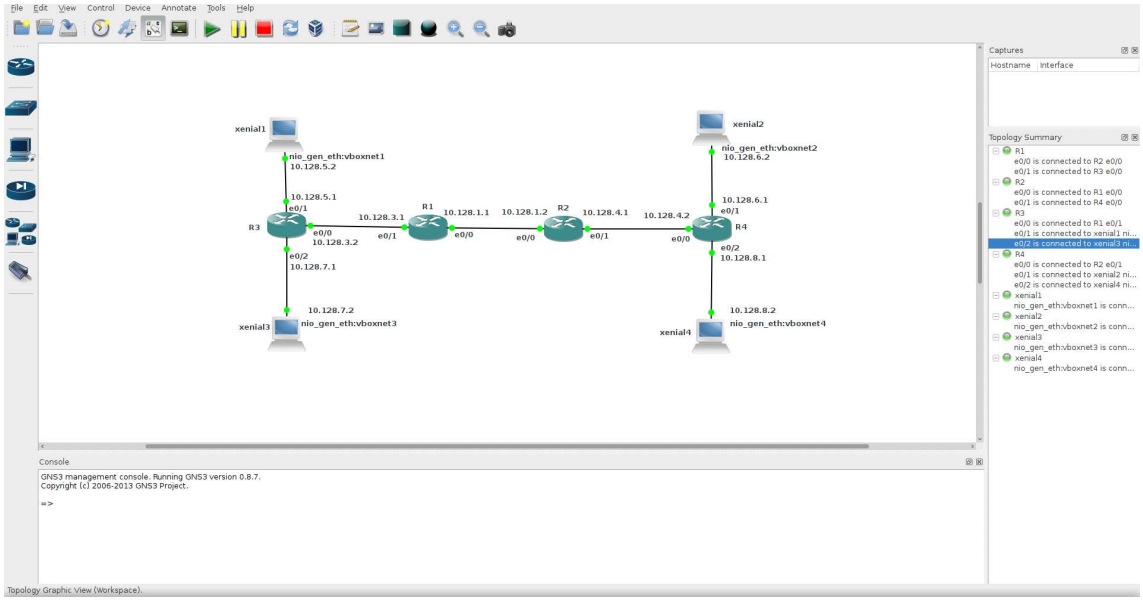
setup.xenial3.sh:

```
 #!/bin/bash
 ETH1=$(dmesg | grep -i 'renamed from eth1' | sed -n 's/: renamed from eth1//;s/.*
//p')
 ifconfig $ETH1 10.128.7.2 netmask 255.255.255.0
 ifconfig -a
 route add -net 10.128.0.0 netmask 255.255.0.0 gw 10.128.7.1
 netstat -rn
 hostname xenial3
 domainname home
```

setup.xenial4.sh

```
 #!/bin/bash
 ETH1=$(dmesg | grep -i 'renamed from eth1' | sed -n 's/: renamed from eth1//;s/.*
//p')
 ifconfig $ETH1 10.128.8.2 netmask 255.255.255.0
 route add -net 10.128.0.0 netmask 255.255.0.0 gw 10.128.8.1
 netstat -rn
```

Finally, if all that work is done, the network in GNS3 should look like this:



You should now be able to ping from xenial3 and xenial4 to xenial1 and xenial2.

```
ljm[anycast]$ vagrant ssh xenial3 -t -- ping -c1 10.128.5.2
PING 10.128.5.2 (10.128.5.2) 56(84) bytes of data.
64 bytes from 10.128.5.2: icmp_seq=1 ttl=63 time=14.6 ms
--- 10.128.5.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 14.616/14.616/14.616/0.000 ms
ljm[anycast]$
```

3. The DNS servers

3.1 Installing a DNS server

Install the software:

```
apt-get install bind9 dnsutils
```

Next is a bit of a cheat. For my normal network I configure DNS with a script and I do not feel like redoing that by hand. The head of the script is specific for the DNS-server:

```
$i_am="xenial1";
$i_am_domain="home";
$i_am_long="xenial1.home";
$i_am_ip="10.128.5.2";
```

So I got the script to xenial1 and xenial2 and provided the input-file:

```
xenial1 10.128.5.2
xenial2 10.128.6.2
xenial3  10.128.7.2
xenial4  10.128.8.2
r1       10.128.3.1
r2       10.128.4.1
r3       10.128.3.2
r4       10.128.4.2
```

Both files are available on `/vagrant` so they can ve used directly from the setup-script:

```
 #!/bin/bash
 echo "Setup xenial1"
 ETH1=$(dmesg | grep -i 'renamed from eth1' | sed -n 's/: renamed from eth1//;s/.*
//p')

 ifconfig $ETH1 10.128.5.2 netmask 255.255.255.0
 route add -net 10.128.0.0 netmask 255.255.0.0 gw 10.128.5.1
 netstat -rn

 apt-get update
 echo "apt-get -y install bind9 dnsutils"
 apt-get -y install bind9 dnsutils

 cd /etc/bind
 perl /vagrant/make_config.1.perl /vagrant/dns-input-file

 cat > /etc/resolv.conf <<EOF
 domain home
 search home
 nameserver 127.0.0.1
 EOF
 cat > /etc/hosts <<EOF
 127.0.0.1    localhost
 10.128.5.2   xenial1.home xenial1
 EOF
 hostname xenial1
 domainname home

 hostname
 domainname
 echo  /etc/resolv.conf
 cat  /etc/resolv.conf
 echo /etc/hosts
 cat /etc/hosts
```

If you use the standard debian Bind9, your name-lookups will give the message that the name-server does not allow recursion. To fix that, add in `/etc/bind/named.conf.options` the line

```
allow-recursion {10.128.0.0/16;};
```

And then it works:

```
ljm[anycast]$ vagrant ssh xenial2 -t -- nslookup r1
Server:         127.0.0.1
Address:        127.0.0.1#53
Name:   r1.home
Address: 10.128.3.1
```

3.2 The clients.

Here is the set-up script for xenial3:

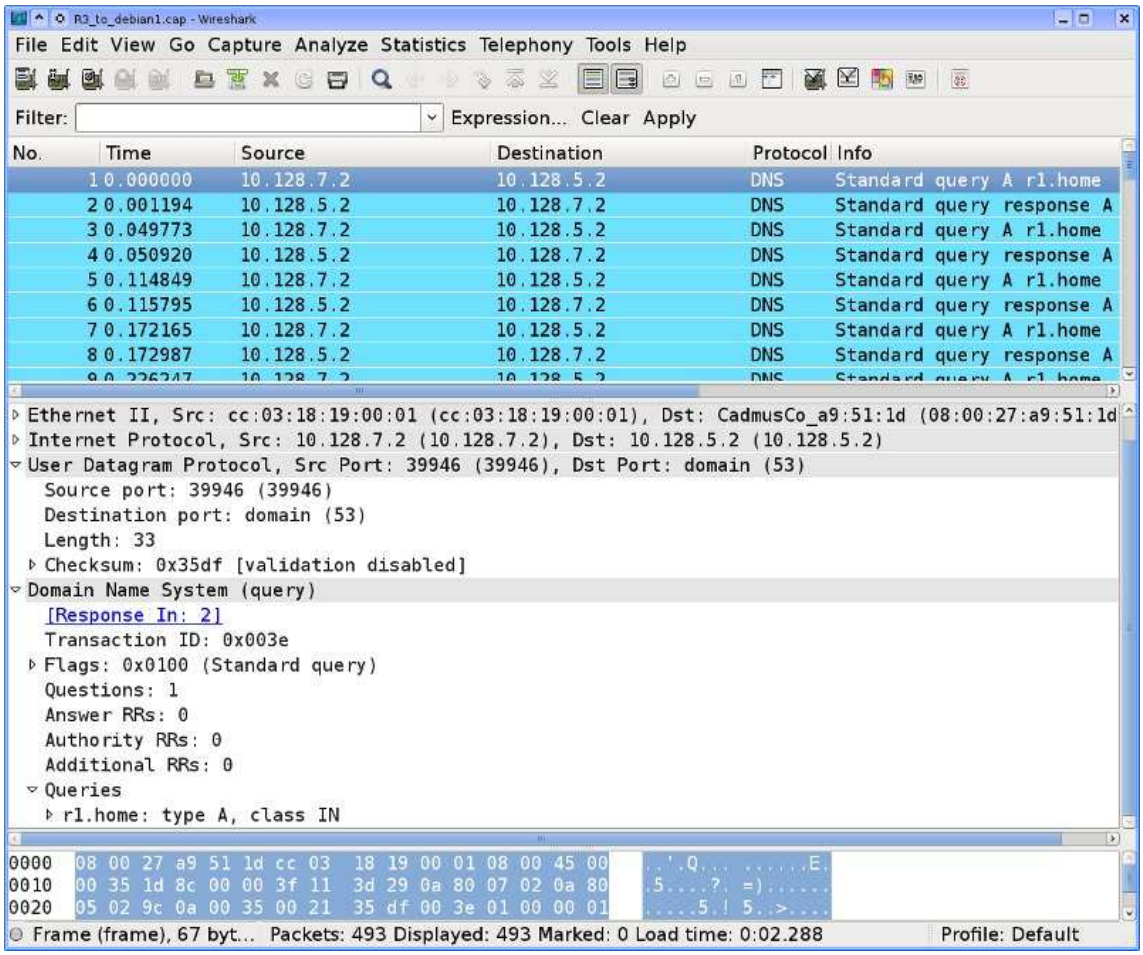
```
 #!/bin/bash

 ETH1=$(dmesg | grep -i 'renamed from eth1' | sed -n 's/: renamed from eth1//;s/.*
//p')


 ifconfig $ETH1 10.128.7.2 netmask 255.255.255.0
 ifconfig -a
 route add -net 10.128.0.0 netmask 255.255.0.0 gw 10.128.7.1
 netstat -rn
 hostname xenial3
 domainname home
 apt-get update
 apt-get install sysvbanner

 banner hosts
 cat >/etc/hosts <<EOF
 127.0.0.1        localhost
 10.128.7.2        xenial3.home        xenial3
 EOF

 banner resolv
 cat > /etc/resolv.conf <<EOF
 domain home
 search home
 nameserver 10.128.5.2
 nameserver 10.128.6.2
 EOF
 banner result
 hostname
 domainname
 netstat -rn
 echo "/etc/hosts:"
 cat /etc/hosts
 echo "/etc/resolv.conf:"
 cat /etc/resolv.conf

 ping -c1 10.128.7.1
```

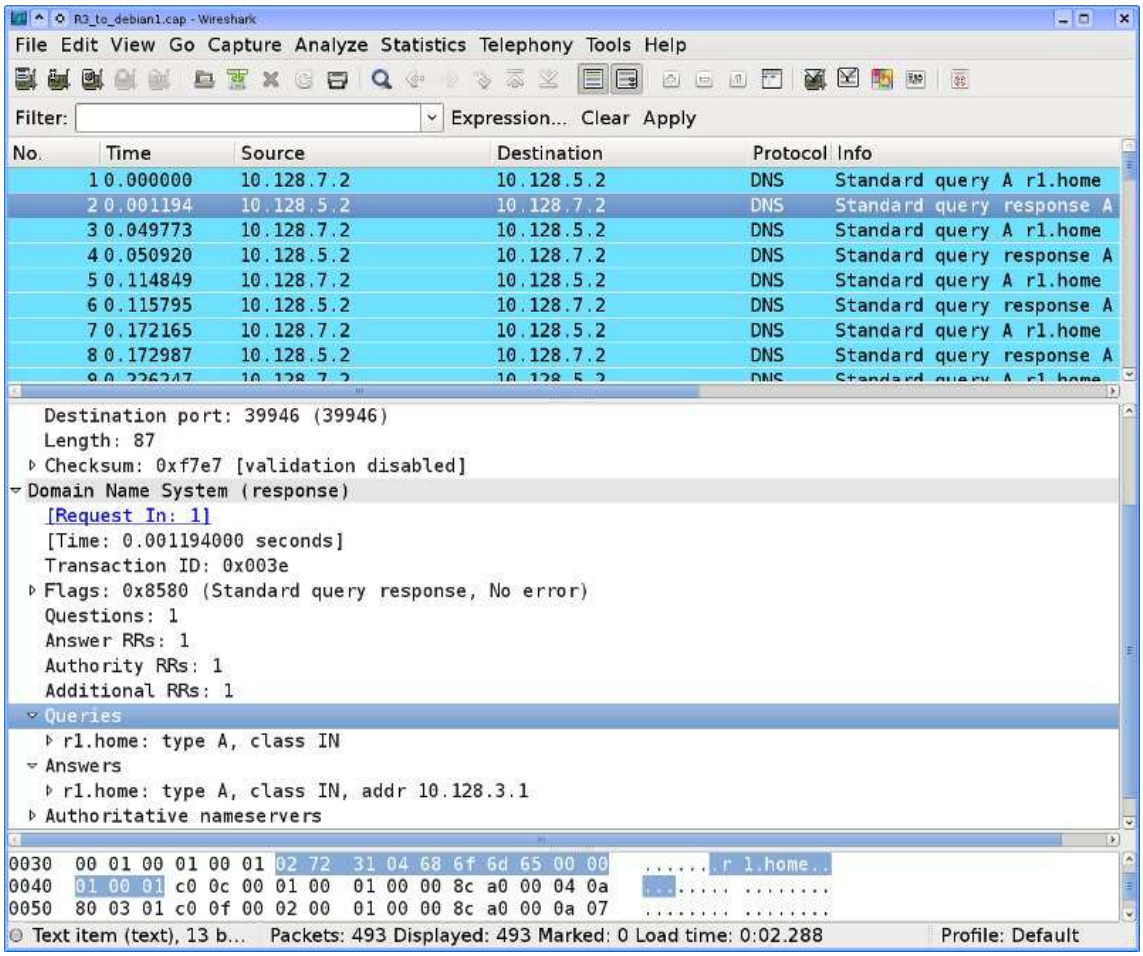
3.3 Testing it

OK. fire-up the routers and the virtual machines and try on xenial3:

```
ljm[anycast]$ vagrant ssh xenial3 -t -- nslookup r1
Server:         10.128.224.2
Address:        10.128.224.2#53
Name:   r1.home
Address: 10.128.3.1
```

And put a wireshark on the line to see the packets:



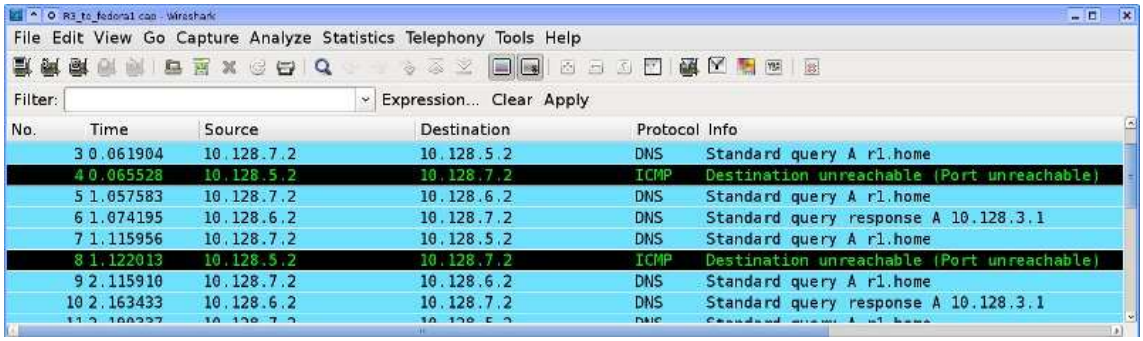


3.4 The problem

This works very nice, but if the first server is down, it will take some time before the second server is used. To show the effect, stop the name server on xenial1 and do an nslookup.

```
ljm@verlaine anycast]$ vagrant ssh xenial4 -c 'time nslookup r1'
Server:         10.128.5.2
Address:        10.128.5.2#53
Name:   r1.home
Address: 10.128.3.1
real    0m0.103s
user    0m0.007s
sys     0m0.002s
Connection to 127.0.0.1 closed.
[ljm@verlaine anycast]$ vagrant halt xenial1
   xenial1: Attempting graceful shutdown of VM...
[ljm@verlaine anycast]$ vagrant ssh xenial4 -c 'time nslookup r1'
Server:         10.128.6.2
Address:        10.128.6.2#53
Name:   r1.home
Address: 10.128.3.1
real    0m1.036s
user    0m0.006s
sys     0m0.003s
Connection to 127.0.0.1 closed.
[ljm@verlaine anycast.vagrant]$ vagrant ssh xenial3 -c 'time nslookup r1'
Server:  10.128.6.2
Address: 10.128.6.2#53
Name:    r1.home
Address: 10.128.3.1
real     0m1.045s
user     0m0.006s
sys      0m0.011s
```

This takes a full second longer. So what happens is this:



The cycle starts at number 3 with a request from xenial3. Because only bind9 is down, there is an ICMP reply. Then, after a time-out, xenial3 tries the next server (destination 10.128.6.2) and gets a reply.

Unavailability of a name server will not stop the rest of the network from working but it will slow down all the applications. For example, if a web-page contains adds from 4 different servers, it will take up to 5 seconds longer to load. That may not be acceptable.

4. Anycast

4.1 Introduction

A solution for failing servers is anycast. With anycasst, we basically do not care which server replies. We let the network determine which server is best in terms of availability or speed. The routing protocol (in this case RIP) will solve all our availability issues. Furthermore, client configuration will be easier; we no longer have to determine by hand which DNS server is closest.

There are some drawbacks. The routing protocol will work at layer 3. That means that, if the server is available, but the service is not, the routing will still route to that server. So some sort of monitoring in the DNS server must be done. Furthermore, it is best if the DNS host participates in the routing protocol. This make the configuration of the server more complicated.

4.2 Anycast on the DNS servers

On Linux, routing is done via Quagga. Quagga has a RIP daemon. So first install Quagga on xenial1 and xenial2:

```
apt-get install quagga
```

Next, we'll define our DNS service address on xenial1 and xenial2:

```
ifconfig lo:0 10.128.224.2 netmask 255.255.255.255
```

On the xenials 1 and 2 in `/etc/quagga` create a file `zebra.conf` with the following content:

```
hostname xenial1
password password
enable password password
interface lo
interface lo:0
ip address 10.128.244.2/32
interface eth1
ip address 10.128.5.2/24
line vty
```

For lo:0 use the same IP address for both name servers. For eth1, use the actual IP address. Next, tell quagga which daemons to start. Create a file `daemons` with the following content:

```
zebra=yes
bgpd=no
ospfd=no
ospf6d=no
ripd=yes
ripngd=no
isisd=no
```

And, since we're using RIP, create `ripd.conf` with:

```
hostname xenial1
password password
log stdout
router rip
 version 2
 network 10.128.0.0/16
 network lo:0
 network eth1
```

It would also be nice if we could switch on/off anycast with a simple flag at start-up, so an if-then is put around the anycast configuration. This gives us the following setup-file:

```
echo "Setup xenial1"
ETH1=$(dmesg | grep -i 'renamed from eth1' | sed -n 's/: renamed from eth1//;s/.*
//p')
ifconfig $ETH1 10.128.5.2 netmask 255.255.255.0
route add -net 10.128.0.0 netmask 255.255.0.0 gw 10.128.5.1
netstat -rn
apt-get update
echo "apt-get -y install bind9 dnsutils"
apt-get -y install bind9 dnsutils
apt-get -y install quagga
cd /etc/bind
perl /vagrant/make_config.1.perl /vagrant/dns-input-file
cat > /etc/resolv.conf <<EOF
domain home
search home
nameserver 127.0.0.1
EOF
cat > /etc/hosts <<EOF
127.0.0.1    localhost
10.128.5.2   xenial1.home xenial1
EOF
hostname xenial1
domainname home
if [ -f /vagrant/do_anycast ] ; then
ifconfig lo:0 10.128.224.2 netmask 255.255.255.255
cat > /etc/quagga/zebra.conf <<EOF
hostname xenial1
password password
enable password password
interface lo
interface lo:0
ip address 10.128.244.2/32
interface eth1
ip address 10.128.5.2/24
line vty
EOF
cat > /etc/quagga/daemons <<EOF
zebra=yes
bgpd=no
ospfd=no
ospf6d=no
```

```
ripd=yes
ripngd=no
isisd=no
EOF
cat > /etc/quagga/ripd.conf <<EOF
hostname xenial1
password password
log stdout
router rip
 version 2
 network 10.128.0.0/16
 network lo:0
 network eth1
EOF
ed /etc/default/bind9 << EOF
1
/-u bind
d
w
q
EOF
/etc/init.d/quagga restart
/etc/init.d/bind9 restart
ping -c1 10.128.244.2
cat > /etc/resolv.conf <<EOF
domain home
search home
nameserver 10.128.224.2
EOF
fi
hostname
domainname
echo  /etc/resolv.conf
cat  /etc/resolv.conf
echo /etc/hosts
cat /etc/hosts
ping -c1 10.128.5.1
```

and then, according to all the websites, it should work....

4.3 Debugging

... and of course it does not.

I had to do some debugging and this is what I found.

4.3.1 Strategy

The debugging strategy works as follows:

- first make sure that the name-servers work as expected
- then verify the routing from/to the servers (network part)
- then make sure that the clients are configured correctly

4.3.2 Provisioning problems

First, make sure you've created `do_anycast` (do a `touch do_anycast` to create the file).

The replacement of `/etc/resolv.conf` and `/etc/hosts` by the provisioning script did not work as expected. The DHCP-client overwrites these files, apparently. Moving the creation of these files to the ens of the set-up script seemed to work. The alternative is to change the system interface-section.

4.3.3 Bind specific problems

There are two things that I see:

— nslookup does not give answers on 10.128.224.1 (not even from the connecting router), but ping replies correctly

— when a nameserver is shut-down, even the ping does not seem to get redirected.

4.3.4 DNS

Bind9 on Ubuntu is, as a default, run with `-u bind` as option. That is for security reasons. From the man-page (BSD, not debian, but that doesn't make much difference):

```
-u      Specifies the user the server should run as after
        it initializes. The value specified may be either
        a username or a numeric user ID. If the -g flag is
        not specified, then the group ID used will be the
        primary group of the user specified (initgroups()
        is called, so all of the user's groups will be
        available to the server).
        Note: normally, named will rescan the active Ether-
        net interfaces when it receives SIGHUP. Use of the
        -u option makes this impossible since the default
        port that named listens on is a reserved port that
        only the superuser may bind to.
```

This suggests that BIND cannot rescan ports after a SIGHUP. But it should be able to register the ports at start-up. BIND binds to port 53 which should only be possible for root.

However, it also suggests that running as `root` instead of `bind` should make it easier to debug, because I don't have to restart every time.

Running as root is done by removing the option `-u bind` from `/etc/default/bind9`.

Restart, and suddenly it works. A quick look at `anycast/rc` solves the mistery. First, bind is started, and after that the `lo:0` is created. If bind has already started and did the setuid() to run as bind, it cannot listen to a port <1023 on the new interface. So either start bind after creating lo:0 or run as root.

4.3.5 No take-over

On the Internet I read that RIP anycast make take some time to do the take-over.

So, we'll make a script to test this theory. We'll use ping to determine if the host is reachable. First, add a host 'test' with different IP address to both nameservers. This enables us to see which nameserver is being used. Then start the following script:

```
while :; do
    if ping -c1 10.128.224.2 ; then
        date >> log
        nslookup test >> log
        echo -n .
    fi
done
```

When the dots start running, shutdown the DNS server (xenial1 if you're doing this on xenial3). And sure enough, after a while you'll see the take-over:

```
<snip>
Wed Feb 20 15:55:04 CET 2013
Server:  10.128.224.2
Address: 10.128.224.2#53
Name:    test.home
Address: 10.128.1.1
Wed Feb 20 15:55:04 CET 2013
Server:  10.128.224.2
Address: 10.128.224.2#53
Name:    test.home
Address: 10.128.1.1
Wed Feb 20 15:55:04 CET 2013
;; connection timed out; no servers could be reached
Wed Feb 20 15:58:54 CET 2013
Server:  10.128.224.2
Address: 10.128.224.2#53
Name:    test.home
Address: 10.128.2.2
<snip>
```

From the IP address of test, you can see that 10.128.224.2 is now routed to the other name server. You can also see that in this case, it took around 4 minutes for the take-over. This is a normal period.

Because the routing is l4 protocol un-aware you can also do this with a ping:

```
[ljm@verlaine anycast.vagrant]$ vagrant ssh xenial3
Last login: Sun Nov 30 21:13:13 2014 from 10.0.2.2
[vagrant@xenial3 ~]$ ping 10.128.224.2
PING 10.128.224.2 (10.128.224.2) 56(84) bytes of data.
64 bytes from 10.128.224.2: icmp_seq=1 ttl=63 time=16.9 ms
64 bytes from 10.128.224.2: icmp_seq=2 ttl=63 time=11.9 ms
64 bytes from 10.128.224.2: icmp_seq=3 ttl=63 time=13.7 ms
64 bytes from 10.128.224.2: icmp_seq=4 ttl=63 time=19.1 ms
64 bytes from 10.128.224.2: icmp_seq=5 ttl=63 time=16.2 ms
64 bytes from 10.128.224.2: icmp_seq=6 ttl=63 time=11.5 ms
64 bytes from 10.128.224.2: icmp_seq=7 ttl=63 time=18.4 ms
64 bytes from 10.128.224.2: icmp_seq=8 ttl=63 time=15.7 ms
64 bytes from 10.128.224.2: icmp_seq=9 ttl=63 time=11.5 ms
64 bytes from 10.128.224.2: icmp_seq=10 ttl=63 time=16.1 ms
64 bytes from 10.128.224.2: icmp_seq=11 ttl=63 time=12.4 ms
64 bytes from 10.128.224.2: icmp_seq=12 ttl=63 time=18.0 ms
64 bytes from 10.128.224.2: icmp_seq=13 ttl=63 time=12.3 ms
64 bytes from 10.128.224.2: icmp_seq=14 ttl=63 time=16.9 ms
From 10.128.7.1 icmp_seq=232 Destination Host Unreachable
From 10.128.7.1 icmp_seq=233 Destination Host Unreachable
From 10.128.7.1 icmp_seq=234 Destination Host Unreachable
From 10.128.7.1 icmp_seq=235 Destination Host Unreachable
From 10.128.7.1 icmp_seq=236 Destination Host Unreachable
From 10.128.7.1 icmp_seq=237 Destination Host Unreachable
From 10.128.7.1 icmp_seq=238 Destination Host Unreachable
From 10.128.7.1 icmp_seq=239 Destination Host Unreachable
From 10.128.7.1 icmp_seq=240 Destination Host Unreachable
From 10.128.7.1 icmp_seq=241 Destination Host Unreachable
From 10.128.7.1 icmp_seq=242 Destination Host Unreachable
From 10.128.7.1 icmp_seq=243 Destination Host Unreachable
From 10.128.7.1 icmp_seq=244 Destination Host Unreachable
From 10.128.7.1 icmp_seq=245 Destination Host Unreachable
From 10.128.7.1 icmp_seq=246 Destination Host Unreachable
From 10.128.7.1 icmp_seq=247 Destination Host Unreachable
From 10.128.7.1 icmp_seq=248 Destination Host Unreachable
From 10.128.7.1 icmp_seq=249 Destination Host Unreachable
From 10.128.7.1 icmp_seq=250 Destination Host Unreachable
From 10.128.7.1 icmp_seq=251 Destination Host Unreachable
From 10.128.7.1 icmp_seq=252 Destination Host Unreachable
From 10.128.7.1 icmp_seq=253 Destination Host Unreachable
From 10.128.7.1 icmp_seq=254 Destination Host Unreachable
From 10.128.7.1 icmp_seq=255 Destination Host Unreachable
From 10.128.7.1 icmp_seq=256 Destination Host Unreachable
64 bytes from 10.128.224.2: icmp_seq=257 ttl=60 time=53.3 ms
64 bytes from 10.128.224.2: icmp_seq=258 ttl=60 time=46.9 ms
64 bytes from 10.128.224.2: icmp_seq=259 ttl=60 time=42.3 ms
64 bytes from 10.128.224.2: icmp_seq=260 ttl=60 time=37.8 ms
64 bytes from 10.128.224.2: icmp_seq=261 ttl=60 time=47.5 ms
64 bytes from 10.128.224.2: icmp_seq=262 ttl=60 time=42.8 ms
64 bytes from 10.128.224.2: icmp_seq=263 ttl=60 time=49.2 ms
```

```
64 bytes from 10.128.224.2: icmp_seq=264 ttl=60 time=36.6 ms
64 bytes from 10.128.224.2: icmp_seq=265 ttl=60 time=50.9 ms
```

Take-over when the server reboots is more seamles:

```
[ljm@xenial4 Documents]$ while :; do date; nslookup test.home; sleep 10; done
Thu May  9 13:02:59 CEST 2013
Server:         10.128.224.2
Address:        10.128.224.2#53
Name:   test.home
Address: 10.128.1.1
Thu May  9 13:03:09 CEST 2013
Server:         10.128.224.2
Address:        10.128.224.2#53
Name:   test.home
Address: 10.128.1.1
Thu May  9 13:03:19 CEST 2013
Server:         10.128.224.2
Address:        10.128.224.2#53
Name:   test.home
Address: 10.128.2.2
Thu May  9 13:03:29 CEST 2013
Server:         10.128.224.2
Address:        10.128.224.2#53
Name:   test.home
Address: 10.128.2.2
```

4.4 Conclusions

Using anycast as a fail-over mechanism with RIP as routing protocol is not a feasible option. Some optimalization may produce better results, but as fail-over it is not sufficient.

Server configuration is uniform, which is nice if you want to deploy a standard image without much customization.

CONTENTS