DNS+Anycast 均衡负载实战（IPV4）

我们都知道google的公共DNS为:8.8.8.8，甚至我们可以在全球任何地方都能ping通这个IP或者通过dig能解析域名，例如如下操作:

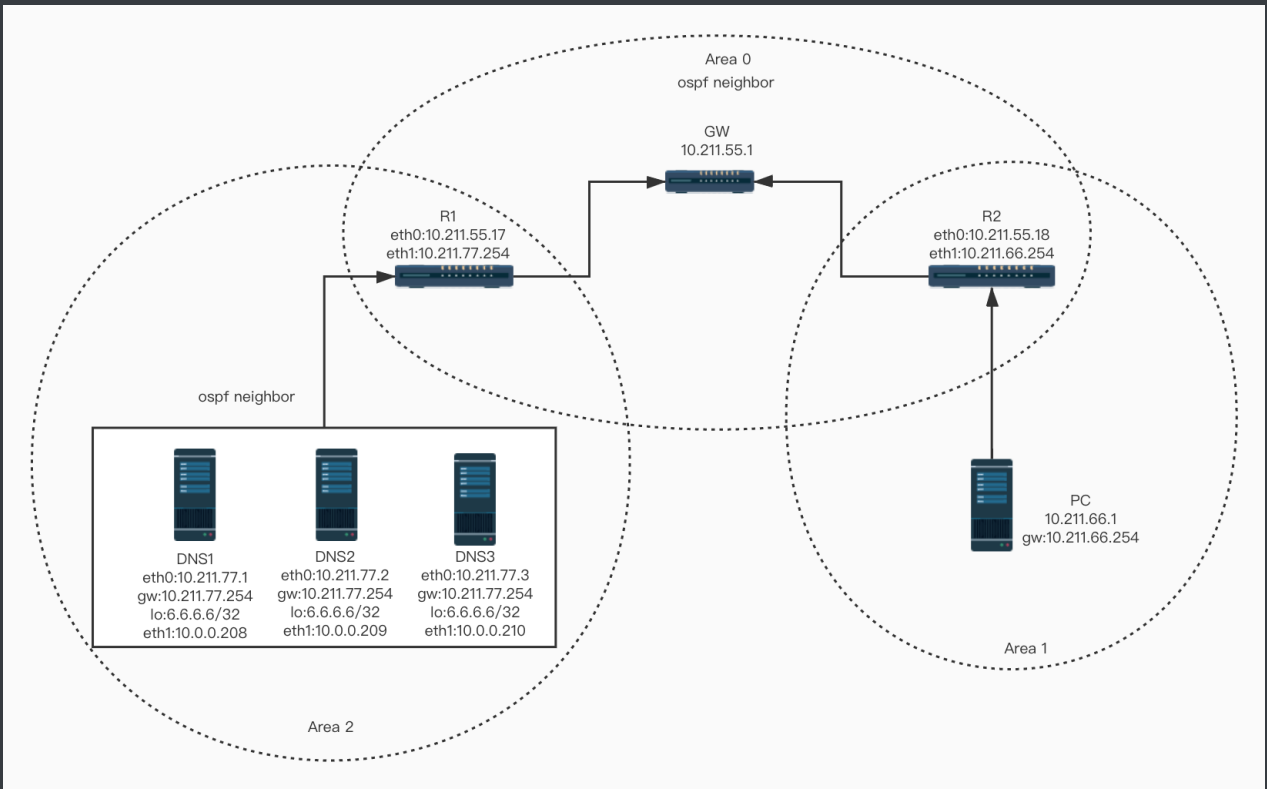
```
# dig www.baidu.com @8.8.8.8 A

; <<>> DiG 9.10.6 <<>> www.baidu.com @8.8.8.8 A
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 53023
;; flags: qr rd ra; QUERY: 1, ANSWER: 4, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 512
;; QUESTION SECTION:
;www.baidu.com.     IN  A

;; ANSWER SECTION:
www.baidu.com.    770 IN  CNAME www.a.shifen.com.
www.a.shifen.com. 53  IN  CNAME www.wshifen.com.
www.wshifen.com.  115 IN  A 104.193.88.77
www.wshifen.com.  115 IN  A 104.193.88.123

;; Query time: 293 msec
;; SERVER: 8.8.8.8#53(8.8.8.8)
;; WHEN: Wed Mar 24 21:57:42 CST 2021
;; MSG SIZE  rcvd: 127
```

通过强制使用8.8.8.8的dns服务器我们可以解析任何域名，这里就涉及到dns+anycast全局均衡负载的技术了。

anycast是什么

anycast准确的说是一种通信方式，其中文名叫：**任播**，最早在rfc1546中提出，只不过rfc1546只是一个概念，并没有实质性的实践，最终是在rfc3513才真正的提出任播地址的格式以及使用方式。但任播地址仅仅使用到了IPV6上，而对于IPV4并没有任播，IPV4只有组播以及单播。而今天讲的则是IPV4的任播方式，虽然没有按照rfc1513的格式实现，但是凭借IPV4这么多年的沉淀，早已利用各种骚操作使用任何场景，下面我们就来解开这IPV4的dns+anycast部署模式。

实战



效果

如上拓扑图，我们建立了一个IP为6.6.6.6内网DNS服务器群，所在网段为10.211.77.0/24，而我们要实现的是在PC端（网段10.211.66.0/24）能ping通6.6.6.6，同时执行命令`dig www.baidu.com @6.6.6.6 A`，能得到文章最开始的dig 8.8.8.8类似的返回结果。

准备工作

硬件准备

6台虚拟机均是centos7，网络模式均采用桥接的方式，其中虚拟机网关IP为：10.211.55.1
R1网卡配置:

```
    eth0:
    IPADDR=10.211.55.17
    PREFIX=24
    GATEWAY=10.211.55.1

    eth1:
    IPADDR=10.211.77.254
    PREFIX=24
    GATEWAY=10.211.55.1
```

R2网卡配置:

```
    eth0:
    IPADDR=10.211.55.18
    PREFIX=24
    GATEWAY=10.211.55.1

    eth1:
    IPADDR=10.211.66.254
    PREFIX=24
    GATEWAY=10.211.55.1
```

PC是直连R2路由器服务器，因此PC是不能上网的，并且PC的网关IP为R2的网卡eth1的IP（10.211.66.254），PC网卡配置如下:

```
    eth0:
    IPADDR=10.211.66.1
    PREFIX=24
    GATEWAY=10.211.66.254
```

DNS服务器组，三台DNS服务器均直连R1路由器，网关都是R1的eth1网卡IP（10.211.77.254），配置eth1网卡只是为了让三台DNS能够上网，这样才能出去递归解析域名，其中10.0.0.0/24是连接了Internet，其网关为10.0.0.2，故DNS群配置分别如下:

```
    DNS1:
      eth0:
        IPADDR=10.211.77.1
        PREFIX=24
        GATEWAY=10.211.77.254
       eth1:
        IPADDR=10.0.0.208
        PREFIX=24
        GATEWAY=10.0.0.1
    DNS2:
      eth0:
        IPADDR=10.211.77.2
        PREFIX=24
        GATEWAY=10.211.77.254
       eth1:
        IPADDR=10.0.0.209
        PREFIX=24
        GATEWAY=10.0.0.1
    DNS3:
      eth0:
```

```
    IPADDR=10.211.77.3
    PREFIX=24
    GATEWAY=10.211.77.254
  eth1:
    IPADDR=10.0.0.210
    PREFIX=24
    GATEWAY=10.0.0.1
```

软件准备

- 配置之前我们先测试一下互通性，我们现在R1上ping 10.211.66.254（R2的eth1 IP），可以看到此时是不通的。

```
[root@localhost quagga]# ping 10.211.66.254
PING 10.211.66.254 (10.211.66.254) 56(84) bytes of data.
^C
--- 10.211.66.254 ping statistics ---
90 packets transmitted, 0 received, 100% packet loss, time 89012ms
```

- 路由器R1、R2配置：
  - 软件只需要准备一款: `quagga` ，这是一个动态路由软件，利用它我们可以在没有路由器的情况下，将linux服务器作为路由器，以达到路由的效果。quagga只是守护进程，真正使用到的服务是:zebra以及ospf，zebra是基础服务，用于静态IP以及路由管理等配置。ospf则是一种路由协议，全称为Open Shortest Path First，开放式最短路径优先协议，这是一个内网网关协议，用于单一自治系统内决策路由。通过ospf协议我们可以将同一路由域内的路由器选定为邻居，并且通过SPF算法计算出最短路由。
    我们利用ospf的特性就将R1与R2建立其邻居关系，这样就可以互相学习对方路由表，这样R1就能学习到R2的10.211.66.0/24网段的所有地址，而R2也可以学习到R1内的10.211.77.0/24网段的所有IP地址。这样一来R2就能ping通10.211.77.0/24网段，例如R1的eth1 IP:10.211.77.254。至于quagga安装不是本文的重点。
  - R1的ospfd.conf配置如下:

```
password foobar
!
interface eth0
!
router ospf
  ospf router-id 10.211.55.17
  network 10.211.55.0/24 area 0.0.0.0
  network 10.211.77.0/24 area 0.0.0.2
  network 6.6.6.6/32 area 0.0.0.2
  network 10.0.0.0/24 area 0.0.0.2
!
line vty
!
log file /var/log/message/quagga/ospfd.log
```

```
然后执行：
systemctl start zebra
systemctl start ospfd
```

- R2的ospfd.conf配置：

```
password foobar
!
interface eth0
!
router ospf
  ospf router-id 10.211.55.18
  network 10.211.55.0/24 area 0.0.0.0
  network 10.211.66.0/24 area 0.0.0.1
!
line vty
!
log file /var/log/message/quagga/ospfd.log

然后执行：
systemctl start zebra
systemctl start ospfd
```

- 观察节点状态:

1. 在R1上执行: `vtysh`

   执行命令： `show ip route` 显示如下信息:

```
centos-7-2-r1.shared# show ip route
Codes: K - kernel route, C - connected, S - static, R - RIP,
       O - OSPF, I - IS-IS, B - BGP, P - PIM, A - Babel, N - NHRP,
       > - selected route, * - FIB route

K>* 0.0.0.0/0 via 10.211.55.1, eth1
O   10.211.55.0/24 [110/10] is directly connected, eth0, 00:00:30
C>* 10.211.55.0/24 is directly connected, eth0
K>* 10.211.55.1/32 is directly connected, eth1
O>* 10.211.66.0/24 [110/20] via 10.211.55.18, eth0, 00:00:20
O   10.211.77.0/24 [110/10] is directly connected, eth1, 00:01:33
C>* 10.211.77.0/24 is directly connected, eth1
C>* 127.0.0.0/8 is directly connected, lo
```

2. 其中O表示ospf邻居网络，O>*则表示学习到的对方的动态路由，这里R1学习到了R2的10.211.66.0/24网段的子网。

   再次输入 `show ip ospf neighbor` ，查看邻居信息：

```
centos-7-2-r1.shared# show ip ospf neighbor

Neighbor ID     Pri State           Dead Time Address         Interface            RXmtL RqstL DBsmL
10.211.55.18      1 Full/DR           32.699s 10.211.55.18    eth0:10.211.55.17        0     0     0
```

   此时显示了10.211.55.18，表示ospf邻居关系已经建立成功。

3. 同理R2上输入 `show ip ospf neighbor` 也能看到如下信息:

```
localhost.localdomain# show ip ospf neighbor

Neighbor ID     Pri State           Dead Time Address         Interface            RXmtL RqstL DBsmL
10.211.55.17      1 Full/Backup       30.422s 10.211.55.17    eth0:10.211.55.18        0     0     0
```

4. 我们看一下路由表，看是否学习到了对方的路由表

   在R1上执行: `show ip ospf database` ，显示信息如下:

```
centos-7-2-r1.shared# show ip ospf database

       OSPF Router with ID (10.211.55.17)

                Router Link States (Area 0.0.0.0)

Link ID         ADV Router      Age  Seq#       CkSum  Link count
10.211.55.17    10.211.55.17     681 0x80000005 0xb2f8 1
10.211.55.18    10.211.55.18     681 0x80000004 0xb2f6 1

                Net Link States (Area 0.0.0.0)

Link ID         ADV Router      Age  Seq#       CkSum
10.211.55.18    10.211.55.18     682 0x80000001 0xbc02

                Summary Link States (Area 0.0.0.0)

Link ID         ADV Router      Age  Seq#       CkSum  Route
10.211.66.0     10.211.55.18     572 0x80000002 0x9c6e 10.211.66.0/24
10.211.77.0     10.211.55.17     744 0x80000001 0x2bd6 10.211.77.0/24

                Router Link States (Area 0.0.0.2)

Link ID         ADV Router      Age  Seq#       CkSum  Link count
10.211.55.17    10.211.55.17     705 0x80000003 0xbe10 1

                Summary Link States (Area 0.0.0.2)

Link ID         ADV Router      Age  Seq#       CkSum  Route
10.211.55.0     10.211.55.17     681 0x80000002 0x1cfa 10.211.55.0/24
10.211.66.0     10.211.55.17     671 0x80000001 0x09f9 10.211.66.0/24
```

在R2上也执行: `show ip ospf database`

```
localhost.localdomain# show ip ospf database

       OSPF Router with ID (10.211.55.18)

                Router Link States (Area 0.0.0.0)

Link ID         ADV Router      Age  Seq#       CkSum  Link count
10.211.55.17    10.211.55.17     804 0x80000005 0xb2f8 1
10.211.55.18    10.211.55.18     803 0x80000004 0xb2f6 1

                Net Link States (Area 0.0.0.0)

Link ID         ADV Router      Age  Seq#       CkSum
10.211.55.18    10.211.55.18     803 0x80000001 0xbc02

                Summary Link States (Area 0.0.0.0)

Link ID         ADV Router      Age  Seq#       CkSum  Route
10.211.66.0     10.211.55.18     693 0x80000002 0x9c6e 10.211.66.0/24
10.211.77.0     10.211.55.17     867 0x80000001 0x2bd6 10.211.77.0/24

                Router Link States (Area 0.0.0.1)

Link ID         ADV Router      Age  Seq#       CkSum  Link count
10.211.55.18    10.211.55.18     803 0x80000003 0x4b8c 1

                Summary Link States (Area 0.0.0.1)

Link ID         ADV Router      Age  Seq#       CkSum  Route
10.211.55.0     10.211.55.18     843 0x80000001 0x18fe 10.211.55.0/24
10.211.77.0     10.211.55.18     798 0x80000001 0x896d 10.211.77.0/24
```

我们可以看到在R1的路由表里已经出现了R2的10.211.66.0/24网段，同时在R2的路由表里也能看到R1的10.211.77.0/24网段。

5. 测试互通性:我们在R1上ping 10.211.66.254（R2的eth1 IP），此时已经实现互通了。

```
[root@localhost quagga]# ping 10.211.66.254
PING 10.211.66.254 (10.211.66.254) 56(84) bytes of data.
64 bytes from 10.211.66.254: icmp_seq=1 ttl=64 time=0.207 ms
64 bytes from 10.211.66.254: icmp_seq=2 ttl=64 time=0.332 ms
64 bytes from 10.211.66.254: icmp_seq=3 ttl=64 time=0.304 ms
64 bytes from 10.211.66.254: icmp_seq=4 ttl=64 time=0.305 ms
64 bytes from 10.211.66.254: icmp_seq=5 ttl=64 time=0.274 ms
64 bytes from 10.211.66.254: icmp_seq=6 ttl=64 time=0.267 ms
64 bytes from 10.211.66.254: icmp_seq=7 ttl=64 time=0.313 ms
64 bytes from 10.211.66.254: icmp_seq=8 ttl=64 time=0.297 ms
64 bytes from 10.211.66.254: icmp_seq=9 ttl=64 time=0.338 ms
^C
```

既然我们已经能ping通10.211.66.254，那此时是否能ping通PC端呢?

在R1上执行 `ping 10.211.66.1` ，显示如下信息，表示成功对接。

```
[root@localhost quagga]# ping 10.211.66.1
PING 10.211.66.1 (10.211.66.1) 56(84) bytes of data.
64 bytes from 10.211.66.1: icmp_seq=1 ttl=63 time=0.469 ms
64 bytes from 10.211.66.1: icmp_seq=2 ttl=63 time=0.525 ms
64 bytes from 10.211.66.1: icmp_seq=3 ttl=63 time=0.555 ms
64 bytes from 10.211.66.1: icmp_seq=4 ttl=63 time=0.511 ms
64 bytes from 10.211.66.1: icmp_seq=5 ttl=63 time=0.458 ms
64 bytes from 10.211.66.1: icmp_seq=6 ttl=63 time=0.493 ms
64 bytes from 10.211.66.1: icmp_seq=7 ttl=63 time=0.480 ms
64 bytes from 10.211.66.1: icmp_seq=8 ttl=63 time=0.474 ms
64 bytes from 10.211.66.1: icmp_seq=9 ttl=63 time=0.461 ms
^C
--- 10.211.66.1 ping statistics ---
9 packets transmitted, 9 received, 0% packet loss, time 8013ms
rtt min/avg/max/mdev = 0.458/0.491/0.555/0.041 ms
```

而PC端的IP如下:

```
[root@localhost ~]# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 00:1c:42:f7:25:b8 brd ff:ff:ff:ff:ff:ff
    inet 10.211.66.1/24 brd 10.211.66.255 scope global noprefixroute eth0
       valid_lft forever preferred_lft forever
    inet6 1888::bb17:ef61:1782:e653/64 scope global noprefixroute dynamic
       valid_lft 2591991sec preferred_lft 604791sec
    inet6 fe80::db94:538:c33:dd32/64 scope link tentative noprefixroute dadfailed
       valid_lft forever preferred_lft forever
    inet6 fe80::baaa:e2e7:32dc:1e66/64 scope link tentative noprefixroute dadfailed
       valid_lft forever preferred_lft forever
    inet6 fe80::6583:f65d:9129:f9ea/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
```

同理我们在PC端执行: `ping 10.211.77.254` ,显示如下信息，完美互通!

```
[root@localhost ~]# ping 10.211.77.254
PING 10.211.77.254 (10.211.77.254) 56(84) bytes of data.
64 bytes from 10.211.77.254: icmp_seq=1 ttl=63 time=0.363 ms
64 bytes from 10.211.77.254: icmp_seq=2 ttl=63 time=0.486 ms
64 bytes from 10.211.77.254: icmp_seq=3 ttl=63 time=0.522 ms
64 bytes from 10.211.77.254: icmp_seq=4 ttl=63 time=0.486 ms
64 bytes from 10.211.77.254: icmp_seq=5 ttl=63 time=0.465 ms
64 bytes from 10.211.77.254: icmp_seq=6 ttl=63 time=0.593 ms
64 bytes from 10.211.77.254: icmp_seq=7 ttl=63 time=0.615 ms
64 bytes from 10.211.77.254: icmp_seq=8 ttl=63 time=0.561 ms
64 bytes from 10.211.77.254: icmp_seq=9 ttl=63 time=0.437 ms
^C
--- 10.211.77.254 ping statistics ---
9 packets transmitted, 9 received, 0% packet loss, time 8005ms
rtt min/avg/max/mdev = 0.363/0.503/0.615/0.075 ms
[root@localhost ~]# _
```

- 接入DNS群组:

1. DNS1上也需要启用ospf，与R1建立邻居关系，这样才能让R1找到最近的dns服务器。DNS1上ospfd.conf配置如下，由于DNS2、DNS3除了router-id不相同以外其余配置都一样，这里就不再贴出。

```
password foobar
!
interface lo
!
```

```
router ospf
  ospf router-id 10.211.77.1
  network 10.211.77.0/24 area 0.0.0.2
  network 6.6.6.6/32 area 0.0.0.2
  network 10.0.0.0/24 area 0.0.0.2
!
line vty
!
log file /var/log/message/quagga/ospfd.log

然后三台DNS上分别执行:
systemctl start zebra
systemctl start ospfd
```

2. 观察状态:
   在DNS1上进入 `vtysh` ， 然后执行: `show ip route` ， 显示如下信息:

```
localhost.localdomain# show ip route
Codes: K - kernel route, C - connected, S - static, R - RIP,
       O - OSPF, I - IS-IS, B - BGP, P - PIM, A - Babel, N - NHRP,
       > - selected route, * - FIB route

K>* 0.0.0.0/0 via 10.211.77.254, eth1
O   6.6.6.6/32 [110/0] is directly connected, lo, 00:04:54
C>* 6.6.6.6/32 is directly connected, lo
O   10.0.0.0/24 [110/10] is directly connected, eth0, 00:04:54
C>* 10.0.0.0/24 is directly connected, eth0
O>* 10.211.55.0/24 [110/20] via 10.211.77.254, eth1, 00:04:41
O>* 10.211.66.0/24 [110/30] via 10.211.77.254, eth1, 00:04:41
O   10.211.77.0/24 [110/10] is directly connected, eth1, 00:04:46
C>* 10.211.77.0/24 is directly connected, eth1
C>* 127.0.0.0/8 is directly connected, lo
```

   可以看到此时DNS1已经连接到了10.211.66.0/24网段已经10.211.77.0/24网段，以及10.211.55.0/24网段。我们在看一下邻居关系，在DNS1上执行 `show ip ospf neighbor`

```
localhost.localdomain# show ip ospf neighbor

Neighbor ID     Pri State           Dead Time Address         Interface            RXmtL RqstL DBsmL
10.211.77.3       1 Full/DROther      35.135s 10.0.0.208      eth0:10.0.0.209          0     0     0
10.211.77.2       1 Full/Backup       38.039s 10.0.0.210      eth0:10.0.0.209          0     0     0
10.211.77.2       1 Full/DROther      38.039s 10.211.77.2     eth1:10.211.77.1         0     0     0
10.211.77.3       1 Full/DROther      35.135s 10.211.77.3     eth1:10.211.77.1         0     0     0
10.211.55.17      1 Full/DR           36.027s 10.211.77.254   eth1:10.211.77.1         0     0     0
localhost.localdomain#
```

此时我们已经可以看到DNS1、DNS2、DNS3都在一个邻居网络上，表示我们的所有网络已经连接成功。此时我们再次测试联通性，在DNS1上执行: `ping 10.211.66.1` （PC的IP），可以看到也是通的。

```
localhost.localdomain# ping 10.211.66.1
PING 10.211.66.1 (10.211.66.1) 56(84) bytes of data.
64 bytes from 10.211.66.1: icmp_seq=1 ttl=62 time=0.633 ms
64 bytes from 10.211.66.1: icmp_seq=2 ttl=62 time=0.676 ms
64 bytes from 10.211.66.1: icmp_seq=3 ttl=62 time=0.718 ms
64 bytes from 10.211.66.1: icmp_seq=4 ttl=62 time=0.708 ms
64 bytes from 10.211.66.1: icmp_seq=5 ttl=62 time=0.689 ms
^C
--- 10.211.66.1 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4007ms
rtt min/avg/max/mdev = 0.633/0.684/0.718/0.044 ms
```

3. 实现anycast

我们刚才都是用的10.211.77.0/24进行访问的，但是我们最终目的是访问6.6.6.6。这里我们在DNS1、DNS2、DNS3的lo环网上添加一个6.6.6.6/32的IP:

```
ip addr add 6.6.6.6/32 dev lo
```

此时的DNS1、DNS2、DNS3的网卡信息分别如下:

```
[root@localhost quagga]# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet 6.6.6.6/32 scope global lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 00:1c:42:eb:67:f2 brd ff:ff:ff:ff:ff:ff
    inet 10.0.0.209/24 brd 10.0.0.255 scope global noprefixroute eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::db94:538:c33:dd32/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 00:1c:42:5f:b1:3b brd ff:ff:ff:ff:ff:ff
    inet 10.211.77.1/24 brd 10.211.77.255 scope global noprefixroute eth1
       valid_lft forever preferred_lft forever
    inet6 1888::4368:7a37:cc9f:b9b2/64 scope global noprefixroute dynamic
       valid_lft 2591806sec preferred_lft 604606sec
    inet6 fe80::ac87:b813:4ee8:a681/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
```

```
[root@localhost ~]# ip addr add 6.6.6.6/32 dev lo
[root@localhost ~]# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet 6.6.6.6/32 scope global lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 00:1c:42:e6:8f:ec brd ff:ff:ff:ff:ff:ff
    inet 10.0.0.210/24 brd 10.0.0.255 scope global noprefixroute eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::db94:538:c33:dd32/64 scope link tentative noprefixroute dadfailed
       valid_lft forever preferred_lft forever
    inet6 fe80::baaa:e2e7:32dc:1e66/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 00:1c:42:89:54:61 brd ff:ff:ff:ff:ff:ff
    inet 10.211.77.2/24 brd 10.211.77.255 scope global noprefixroute eth1
       valid_lft forever preferred_lft forever
    inet6 1888::3e68:b0:8f69:10d1/64 scope global noprefixroute dynamic
       valid_lft 2591724sec preferred_lft 604524sec
    inet6 fe80::ac87:b813:4ee8:a681/64 scope link tentative noprefixroute dadfailed
       valid_lft forever preferred_lft forever
    inet6 fe80::10c3:5ef6:d39b:b6c0/64 scope link tentative noprefixroute dadfailed
       valid_lft forever preferred_lft forever
    inet6 fe80::9128:5509:2e5b:f5ec/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
```

```
[root@localhost ~]# ip addr add 6.6.6.6/32 dev lo
[root@localhost ~]# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet 6.6.6.6/32 scope global lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 00:1c:42:f6:d8:10 brd ff:ff:ff:ff:ff:ff
    inet 10.0.0.208/24 brd 10.0.0.255 scope global noprefixroute eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::db94:538:c33:dd32/64 scope link tentative noprefixroute dadfailed
       valid_lft forever preferred_lft forever
    inet6 fe80::baaa:e2e7:32dc:1e66/64 scope link tentative noprefixroute dadfailed
       valid_lft forever preferred_lft forever
    inet6 fe80::6583:f65d:9129:f9ea/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 00:1c:42:0f:c4:c6 brd ff:ff:ff:ff:ff:ff
    inet 10.211.77.3/24 brd 10.211.77.255 scope global noprefixroute eth1
       valid_lft forever preferred_lft forever
    inet6 fe80::ac87:b813:4ee8:a681/64 scope link tentative noprefixroute dadfailed
       valid_lft forever preferred_lft forever
    inet6 fe80::10c3:5ef6:d39b:b6c0/64 scope link tentative noprefixroute dadfailed
       valid_lft forever preferred_lft forever
    inet6 fe80::9128:5509:2e5b:f5ec/64 scope link tentative noprefixroute dadfailed
       valid_lft forever preferred_lft forever
```

此时我们再通过PC端来ping 6.6.6.6，看看效果:

```
[root@localhost ~]# ping 6.6.6.6
PING 6.6.6.6 (6.6.6.6) 56(84) bytes of data.
64 bytes from 6.6.6.6: icmp_seq=1 ttl=62 time=0.665 ms
64 bytes from 6.6.6.6: icmp_seq=2 ttl=62 time=0.723 ms
64 bytes from 6.6.6.6: icmp_seq=3 ttl=62 time=0.638 ms
64 bytes from 6.6.6.6: icmp_seq=4 ttl=62 time=0.791 ms
64 bytes from 6.6.6.6: icmp_seq=5 ttl=62 time=0.695 ms
64 bytes from 6.6.6.6: icmp_seq=6 ttl=62 time=0.628 ms
64 bytes from 6.6.6.6: icmp_seq=7 ttl=62 time=0.868 ms
^C
--- 6.6.6.6 ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 6005ms
rtt min/avg/max/mdev = 0.628/0.715/0.868/0.084 ms
[root@localhost ~]#
```

由此我们的整个网络拓扑已经建立起来，PC端成功的ping通6.6.6.6网段。也实现我们预期的效果。

4. 搭建dns服务器

   我们这里使用的是bind9作为DNS服务器，分别在DNS1、DNS2、DNS3上执行: `./named -c ./named.conf -g` ，这样我们就建立起三个dns服务器，监控日志打印如下信息表示成功启动:

```
24-Mar-2021 13:00:21.530 automatic empty zone: view default: 106.100.IN-ADDR.ARPA
24-Mar-2021 13:00:21.531 automatic empty zone: view default: 107.100.IN-ADDR.ARPA
24-Mar-2021 13:00:21.531 automatic empty zone: view default: 108.100.IN-ADDR.ARPA
24-Mar-2021 13:00:21.531 automatic empty zone: view default: 109.100.IN-ADDR.ARPA
24-Mar-2021 13:00:21.531 automatic empty zone: view default: 110.100.IN-ADDR.ARPA
24-Mar-2021 13:00:21.531 automatic empty zone: view default: 111.100.IN-ADDR.ARPA
24-Mar-2021 13:00:21.531 automatic empty zone: view default: 112.100.IN-ADDR.ARPA
24-Mar-2021 13:00:21.531 automatic empty zone: view default: 113.100.IN-ADDR.ARPA
24-Mar-2021 13:00:21.532 automatic empty zone: view default: 114.100.IN-ADDR.ARPA
24-Mar-2021 13:00:21.532 automatic empty zone: view default: 115.100.IN-ADDR.ARPA
24-Mar-2021 13:00:21.532 automatic empty zone: view default: 116.100.IN-ADDR.ARPA
24-Mar-2021 13:00:21.532 automatic empty zone: view default: 117.100.IN-ADDR.ARPA
24-Mar-2021 13:00:21.532 automatic empty zone: view default: 118.100.IN-ADDR.ARPA
24-Mar-2021 13:00:21.532 automatic empty zone: view default: 119.100.IN-ADDR.ARPA
24-Mar-2021 13:00:21.532 automatic empty zone: view default: 120.100.IN-ADDR.ARPA
24-Mar-2021 13:00:21.533 automatic empty zone: view default: 121.100.IN-ADDR.ARPA
24-Mar-2021 13:00:21.533 automatic empty zone: view default: 122.100.IN-ADDR.ARPA
24-Mar-2021 13:00:21.533 automatic empty zone: view default: 123.100.IN-ADDR.ARPA
24-Mar-2021 13:00:21.533 automatic empty zone: view default: 124.100.IN-ADDR.ARPA
24-Mar-2021 13:00:21.533 automatic empty zone: view default: 125.100.IN-ADDR.ARPA
24-Mar-2021 13:00:21.533 automatic empty zone: view default: 126.100.IN-ADDR.ARPA
24-Mar-2021 13:00:21.534 automatic empty zone: view default: 127.100.IN-ADDR.ARPA
24-Mar-2021 13:00:21.534 automatic empty zone: view default: 0.IN-ADDR.ARPA
24-Mar-2021 13:00:21.534 automatic empty zone: view default: 127.IN-ADDR.ARPA
24-Mar-2021 13:00:21.534 automatic empty zone: view default: 254.169.IN-ADDR.ARPA
24-Mar-2021 13:00:21.535 automatic empty zone: view default: 2.0.192.IN-ADDR.ARPA
24-Mar-2021 13:00:21.535 automatic empty zone: view default: 100.51.198.IN-ADDR.ARPA
24-Mar-2021 13:00:21.535 automatic empty zone: view default: 113.0.203.IN-ADDR.ARPA
24-Mar-2021 13:00:21.535 automatic empty zone: view default: 255.255.255.255.IN-ADDR.ARPA
24-Mar-2021 13:00:21.535 automatic empty zone: view default: 0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.
.ARPA
24-Mar-2021 13:00:21.536 automatic empty zone: view default: 1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.
.ARPA
24-Mar-2021 13:00:21.536 automatic empty zone: view default: D.F.IP6.ARPA
24-Mar-2021 13:00:21.536 automatic empty zone: view default: 8.E.F.IP6.ARPA
24-Mar-2021 13:00:21.537 automatic empty zone: view default: 9.E.F.IP6.ARPA
24-Mar-2021 13:00:21.537 automatic empty zone: view default: A.E.F.IP6.ARPA
24-Mar-2021 13:00:21.537 automatic empty zone: view default: B.E.F.IP6.ARPA
24-Mar-2021 13:00:21.538 automatic empty zone: view default: 8.B.D.0.1.0.0.2.IP6.ARPA
24-Mar-2021 13:00:21.538 automatic empty zone: view default: EMPTY.AS112.ARPA
24-Mar-2021 13:00:21.538 automatic empty zone: view default: HOME.ARPA
24-Mar-2021 13:00:21.540 none:98: 'max-cache-size 90%' - setting to 728MB (out of 809MB)
24-Mar-2021 13:00:21.572 command channel listening on 127.0.0.1#953
24-Mar-2021 13:00:21.572 not using config file logging statement for logging due to -g option
24-Mar-2021 13:00:21.580 managed-keys-zone/default: loaded serial 0
24-Mar-2021 13:00:21.589 all zones loaded
24-Mar-2021 13:00:21.590 running
```

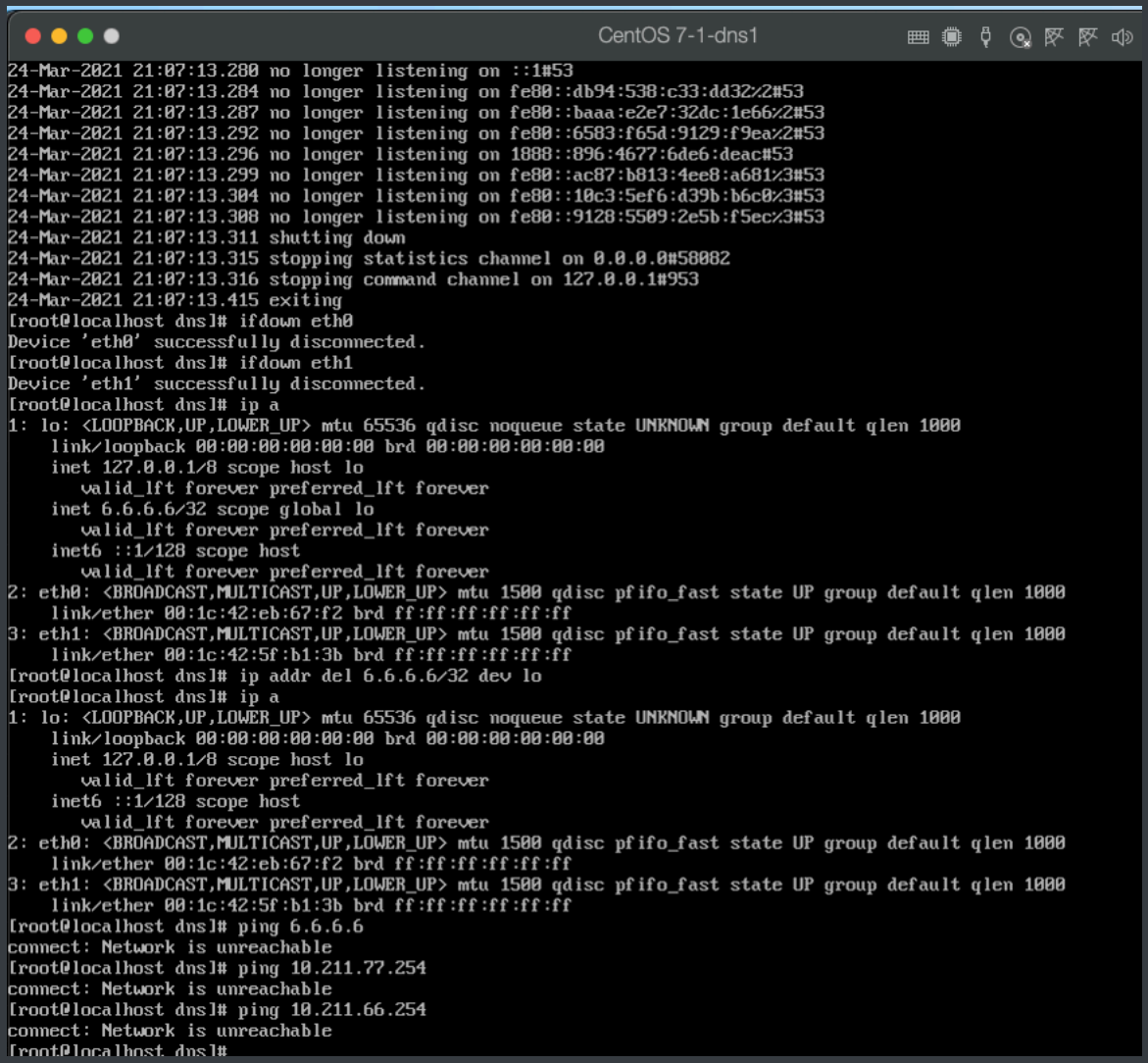
此时我们在PC端执行: `dig www.baidu.com @6.6.6.6 A`

```
[root@localhost ~]# dig www.baidu.com @6.6.6.6 A

; <<>> DiG 9.16.5 <<>> www.baidu.com @6.6.6.6 A
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 13835
;; flags: qr rd ra; QUERY: 1, ANSWER: 3, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 291a3a2a9b631ccb01000000605b70db62238a78122ce0a1 (good)
;; QUESTION SECTION:
;www.baidu.com.                 IN      A

;; ANSWER SECTION:
www.baidu.com.          1200    IN      CNAME   www.a.shifen.com.
www.a.shifen.com.       300     IN      A       14.215.177.38
www.a.shifen.com.       300     IN      A       14.215.177.39

;; Query time: 2519 msec
;; SERVER: 6.6.6.6#53(6.6.6.6)
;; WHEN: Wed Mar 24 13:24:04 EDT 2021
;; MSG SIZE  rcvd: 132
```

此时我们已经成功的部署好DNS服务器并且能正常提供递归服务。

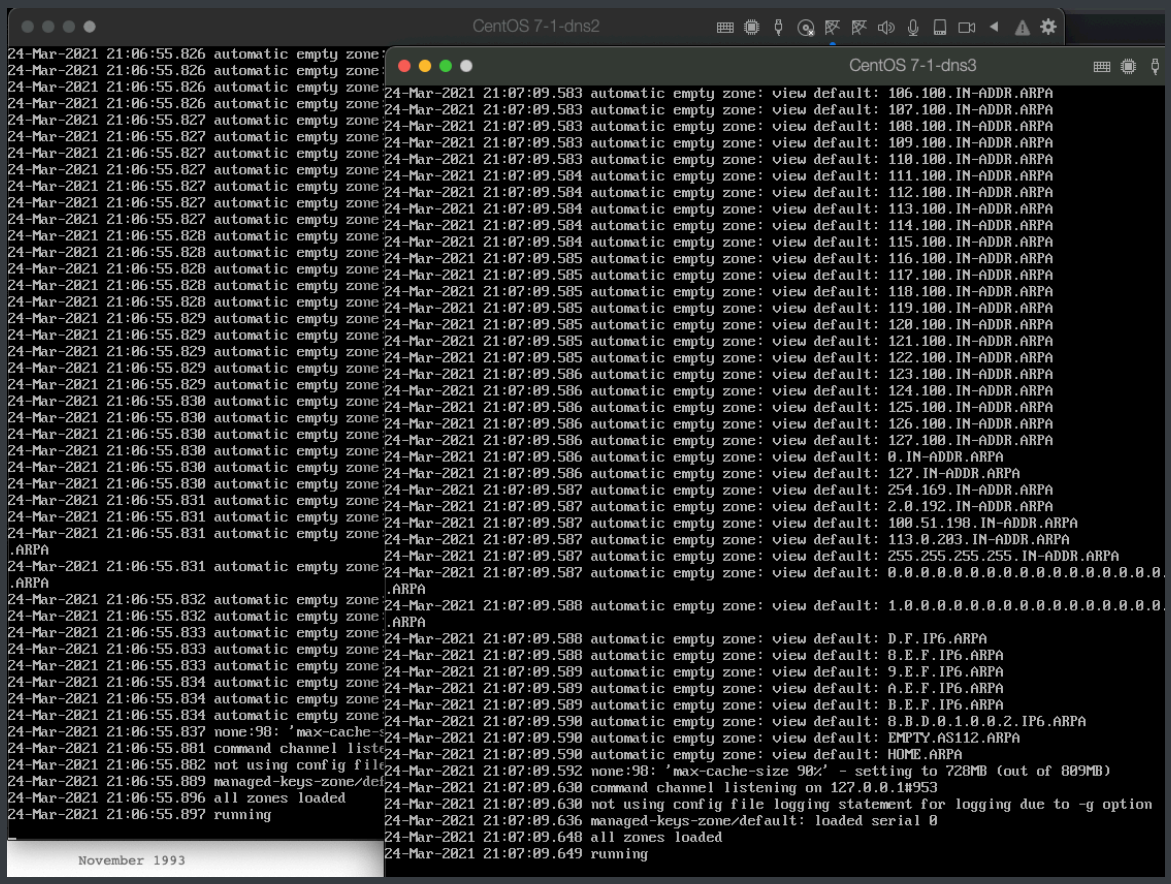
5. 均衡负载

我们刚才虽然已经将三台DNS架设起来，而且PC端也能成功的dig请求6.6.6.6解析域名，但是其实现在每次dig的请求都是到了DNS1这台服务器，与我们想要的均衡负载似乎并没有太大的关系，并且我们本可以使用DNS1的eth1网卡IP:10.211.77.1，为何又大费周章的去做一个6.6.6.6/32的地址来进行解析呢？这似乎跟均衡以及anycast没半毛钱关系。
我们做一个实验，如果把DNS1服务器关掉或者关闭网卡，会是怎么样的呢?

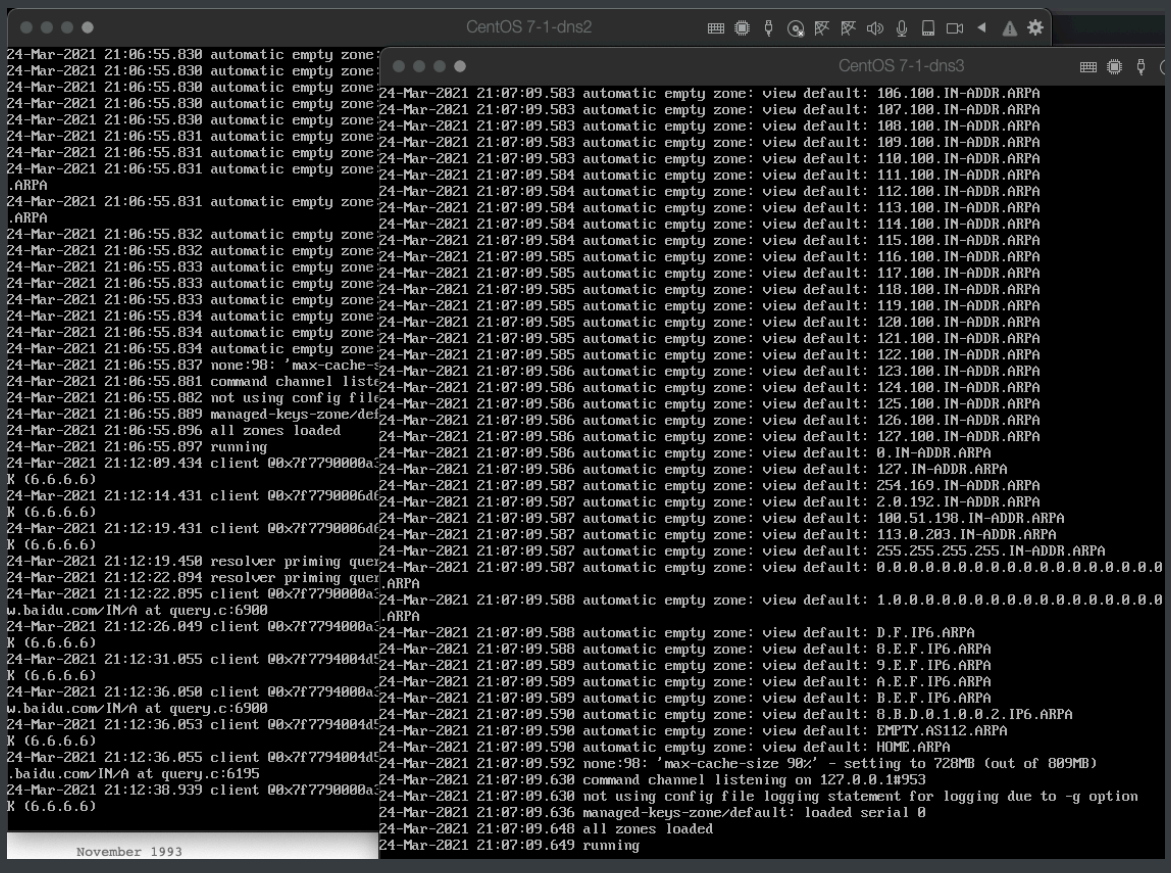

CentOS 7-1-dns1
24-Mar-2021 21:07:13.280 no longer listening on ::1#53
24-Mar-2021 21:07:13.284 no longer listening on fe80::db94:538:c33:dd32%2#53
24-Mar-2021 21:07:13.287 no longer listening on fe80::baaa:e2e7:32dc:1e66%2#53
24-Mar-2021 21:07:13.292 no longer listening on fe80::6583:f65d:9129:f9ea%2#53
24-Mar-2021 21:07:13.296 no longer listening on 1888::896:4677:6de6:deac#53
24-Mar-2021 21:07:13.299 no longer listening on fe80::ac87:b813:4ee8:a681%3#53
24-Mar-2021 21:07:13.304 no longer listening on fe80::10c3:5ef6:d39b:b6c0%3#53
24-Mar-2021 21:07:13.308 no longer listening on fe80::9128:5509:2e5b:f5ec%3#53
24-Mar-2021 21:07:13.311 shutting down
24-Mar-2021 21:07:13.315 stopping statistics channel on 0.0.0.0#58082
24-Mar-2021 21:07:13.316 stopping command channel on 127.0.0.1#953
24-Mar-2021 21:07:13.415 exiting
[root@localhost dns]# ifdown eth0
Device 'eth0' successfully disconnected.
[root@localhost dns]# ifdown eth1
Device 'eth1' successfully disconnected.
[root@localhost dns]# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
inet 127.0.0.1/8 scope host lo
valid_lft forever preferred_lft forever
inet 6.6.6.6/32 scope global lo
valid_lft forever preferred_lft forever
inet6 ::1/128 scope host
valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
link/ether 00:1c:42:eb:67:f2 brd ff:ff:ff:ff:ff:ff
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
link/ether 00:1c:42:5f:b1:3b brd ff:ff:ff:ff:ff:ff
[root@localhost dns]# ip addr del 6.6.6.6/32 dev lo
[root@localhost dns]# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
inet 127.0.0.1/8 scope host lo
valid_lft forever preferred_lft forever
inet6 ::1/128 scope host
valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
link/ether 00:1c:42:eb:67:f2 brd ff:ff:ff:ff:ff:ff
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
link/ether 00:1c:42:5f:b1:3b brd ff:ff:ff:ff:ff:ff
[root@localhost dns]# ping 6.6.6.6
connect: Network is unreachable
[root@localhost dns]# ping 10.211.77.254
connect: Network is unreachable
[root@localhost dns]# ping 10.211.66.254
connect: Network is unreachable
[root@localhost dns]#

我们在DNS1上分别关掉eth0、eth1以及删掉6.6.6.6/32，此时 `ip a` 可以看到DNS1已经是断网的状态，也ping不通任何服务器。

而此时的DNS2以及DNS3的状态如下,都已经开启了DNS服务。



此时我们再次从PC端执行 `dig www.baidu.com @6.6.6.6 A` ，此时的DNS2以及DNS3的日志打印:



可以看到DNS2已经收到了PC端的A记录解析请求。同时PC端也收到了响应:

CentOS 7-2-pc

```
[root@localhost ~]# dig www.baidu.com @6.6.6.6 A

; <<>> DiG 9.16.5 <<>> www.baidu.com @6.6.6.6 A
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 17865
;; flags: qr rd ra; QUERY: 1, ANSWER: 3, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: f0a73761cac7319a01000000605be38b7d19c610dd859f02 (good)
;; QUESTION SECTION:
;www.baidu.com.                 IN      A

;; ANSWER SECTION:
www.baidu.com.          1200    IN      CNAME   www.a.shifen.com.
www.a.shifen.com.       300     IN      A       14.215.177.38
www.a.shifen.com.       300     IN      A       14.215.177.39

;; Query time: 4436 msec
;; SERVER: 6.6.6.6#53(6.6.6.6)
;; WHEN: Wed Mar 24 13:40:40 EDT 2021
;; MSG SIZE  rcvd: 132
```

如此一来，DNS的均衡负载就能正常运转，同样的道理如果我们把DNS1再次启动，此时的PC端的请求会回到DNS1上。这样就可以水平拓展DNS服务器以达到均衡负载的目的。

6. 原理

   以上就是DNS+Anycast搭建均衡负载的实战，这其中比较关键的点就是ospf协议以及在lo环网上添加6.6.6.6/32作为均衡负载IP。我们在刚才的实验中可以看到，当DNS1断开以后，请求能自动达到了DNS2上，这一步其实就是ospf协议做的工作。我们在搭建的时候将DNS1、DNS2、DNS3以及R1一起通过ospf建立了邻居关系。

   此时的DNS1、DNS2、DNS3就是R1路由器的邻居，同时DNS三台服务器上的所有网段都能被R1路由器学习到，因此在三台DNS服务器上添加了6.6.6.6/32也能被R1路由器学习到。而我们都知道ospf协议是开放最短路径最优协议，也就是在邻居里面会计算出最近的邻居，一般这个路径是根据路由跳数计算，而三台DNS处于一个路由内网中，因此就只需要记录谁最先响应hello组播消息，那么谁就是最近的邻居。因此就有了每次请求6.6.6.6的时候，都是DNS1响应。而如果DNS1断开以后则是DNS2响应。这样就通过路由协议以实现一个均衡负载的场景。

   在lookback回环网卡上添加6.6.6.6/32是因为lookback网卡的状态永远是up的，即是没有配置地址，他都会存在，这就保证了服务的可用。另外就是将相同的IP添加到lookback上就不会存在IP冲突的情况，因为回环地址只会作为主机解析使用，它不会讲任何数据传输给网络接口。也就是说要访问本机的6.6.6.6/32其实是通过了网卡eth0或者eth1入口，然后解析到回环上的。所以就不存在同一局域网内IP冲突的情况。

7. 缺点

   讲了这么多anycast的部署以及优点，但它有一个致命的弱点，那就是因为它本身就是通过IP协议逐跳寻址的特性，讲数据包导向不通的目的地，但由于逐跳的路由收敛和端到端的五元组连接互相没有同步，导致anycast不能用于TCP长链接。例如在请求的过程中，数据还没返回，但DNS服务器断开，此时路由会收敛计算到其他路径，这就会导致返回路径不一致，从而造成TCP主机之间会出现断开的情况，因此anycast只适合例如UDP这样一问一答的情景，例如DNS，而且google的公共DNS就是利用了这个特性进行搭建的，只不过google搭建的是一个全球性的公共DNS，利用的是BGP边界路由协议，实现的是不通路由之间的宣告，而本例只是利

用了ospf协议搭建的内网的DNS，仅仅是google公共DNS的一个很小的单元罢了。

但管中窥豹，我们通过小型的内网anycast场景的搭建也能窥视全球DNS的布局模式，这也不为一种学习方式。