

ECE 792/CSC 791 - LINUX NETWORKING

VIRTUAL PRIVATE CLOUD – DNS AS A SERVICE

FINAL PROJECT REPORT

TEAM MEMBERS

Harish Annamalai Palaniappan - hpalani

Jagadeesh Saravanan – jsarava

Ragavi Swarnalatha Raman – rraman2

Sampreetha Naattamai Sureshbabu - snaatta

DNS as a Service:

Our Cloud DNS is a reliable,scalable and managed hierarchial DNS service that can be used in a Virtual Private Cloud Environment to set up Private Hosted Zones ie)DNS servers at various points in the private L2 network. The created DNS system enables the usage of private domain names for the servers/machines inside the VPC and translates the requests for the domain names into their corresponding IP addresses.

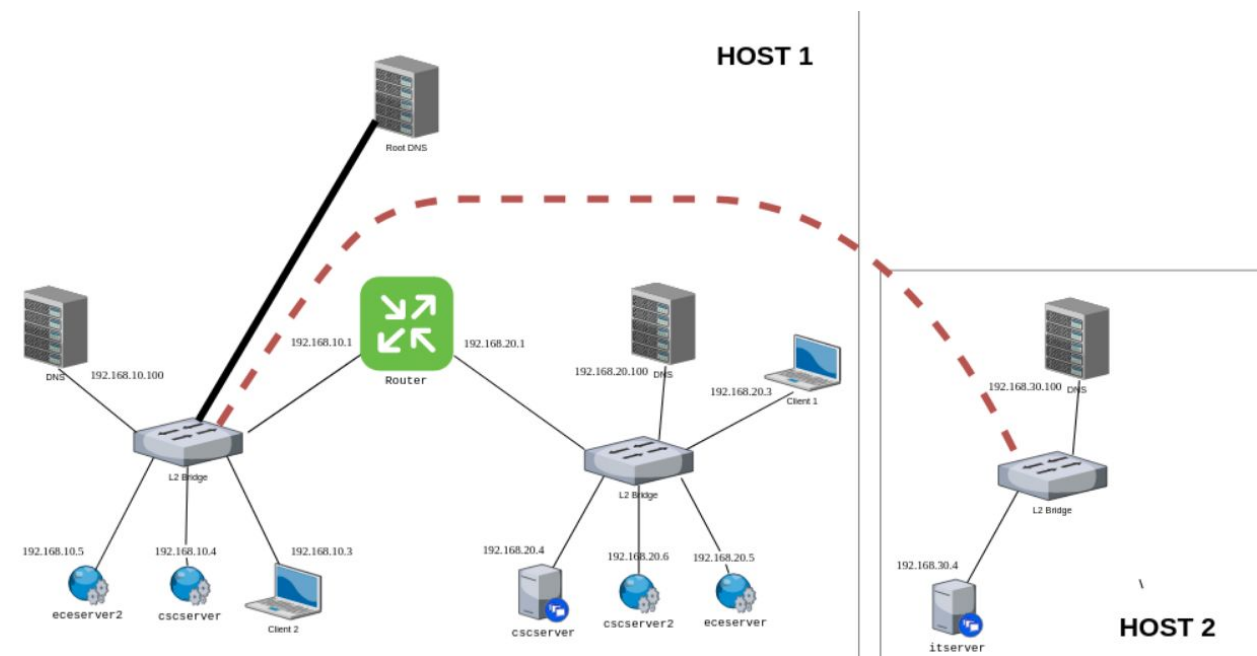
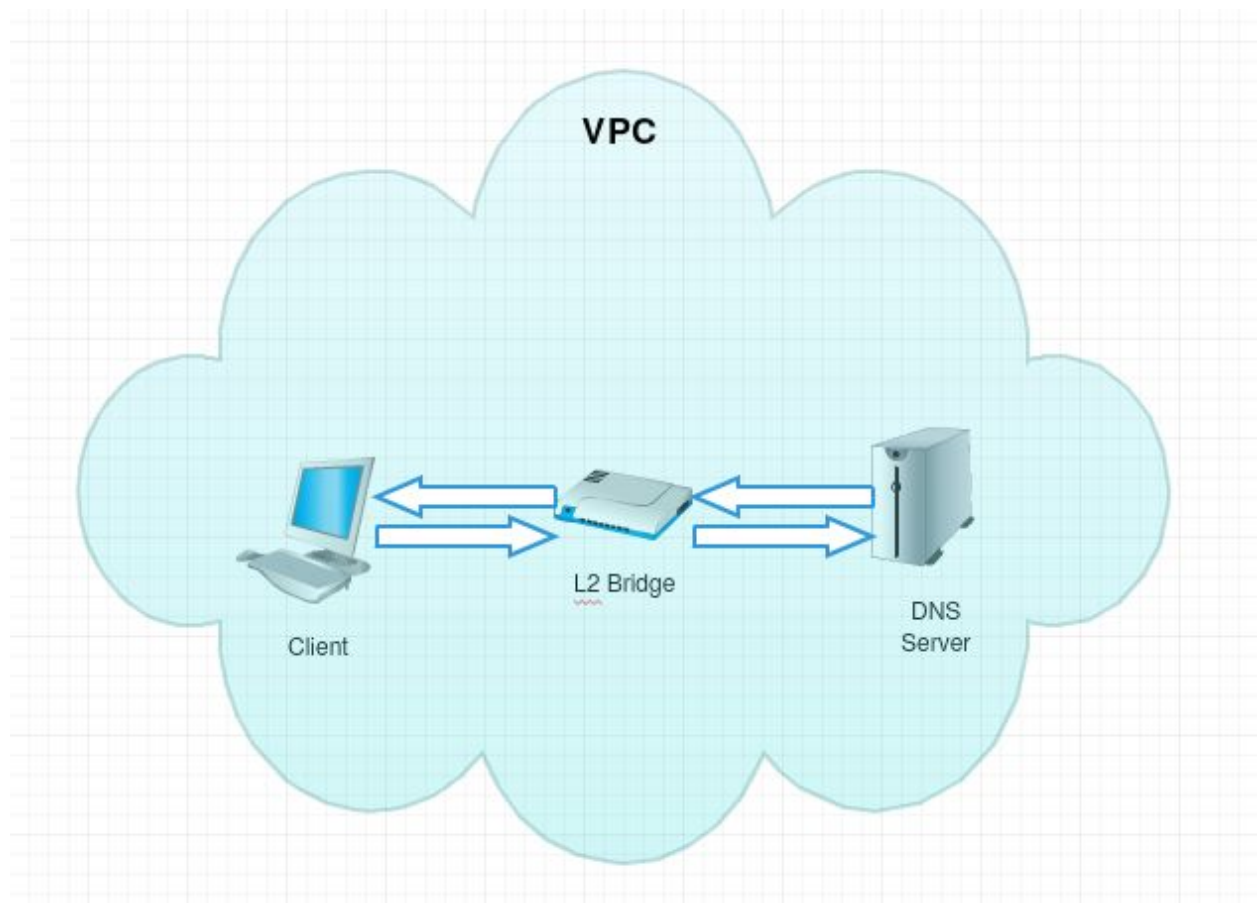
Our DNS solution is an automated application that enables a VPC owner to configure DNS servers as per requirement in various subnets. The application auto-generates Forward and Reverse zone files for the subnets in the DNS server and enables access of servers in the VPC through URLs in addition to their IP addresses. Our application can also configure DNS service across multiple sites that form a part of the VPC through GRE Tunneling.

Project Proposal Document: [Click Here](#)

System Architecture:

- The system architecture contains two separate hypervisors with L2 Bridges to which Virtual Machines and namespaces are connected to.
- The Virtual Machines and Namespaces act as the clients , servers and DNS servers.
- GRE Tunneling is used to connect the two hypervisors to make it appear as if they exist in the same Private Network.

A general overview of the system interaction during the Name resolution is as follows:



Feature Architecture:

1. Name Resolution within Zones

DNS server resolves the name and provides corresponding IP address configured. For example, Client 1 contacts the DNS to reach the ece server using www.ece.univ.edu. DNS resolves the address by giving the IP of any of the configured ece server. Now the client can directly reach the ece server using the obtained IP address.

2. Load Balancing of Servers

This feature balances traffic between several web servers that carry the same content/run the same application. The DNS server to which the request is made does load balancing and distributes the request by returning an IP address from the set mapped to the same domain name in a round robin basis.

3. Location/Source IP Based Aware Resolution

When the VPC offers a service / runs an application on a number of identically configured servers that are distributed across several subnets for redundancy / easy access purpose and the servers need to be reachable using a single domain name , but at the same time the DNS request needs to point users to the server that is closer / present in the same subnet as the client itself we use a Location Aware / Content Based Resolution.

4. High Availability / DNS Failover

When the client queries a DNS and finds that it is shutdown/ failed, then the next entry of DNS in /etc/resolv.conf file is queried for the name resolution.

Implementation:

Clients and Web Servers :

Clients and Web Servers are either VMs installed using libvirt module on the L2 bridges or namespaces created in the hypervisor and attached to the L2 bridge using veth pairs

DNS Server:

BIND name server software is used to set up an internal DNS server which can resolve private host names to their IP addresses. Clients within the same or different subnet contact this DNS server to reach the web servers.

Steps to configure BIND as private network DNS Server:

1. Install BIND9 on DNS servers
2. Configure BIND

Edit /etc/named.conf file to create a new Access Control Lists which includes clients/subnets that can query the DNS. List the forward and reverse zone files that are to be fetched during resolution. Add the private address of Nameserver (NS) to list of addresses that listen on port 53.

4. Create Forward zone file

The forward zone file defines the DNS records for forward DNS lookups. That is, when the DNS receives a name query, it looks in the forward zone file to resolve the server's corresponding private IP address. Edit the file db.<forward zone> file to add SOA, name server and A records (for nameserver and hosts).

5. Create Reverse zone file

Reverse zone file is used to define DNS PTR records for reverse DNS lookups. That is, when the DNS receives a query by IP address, it looks in the reverse zone file(s) to resolve the corresponding Fully Qualified Domain Name

(FQDN). Edit the db.<reverse zone> file to add SOA, name server and PTR records (for all servers whose IP addresses are on same subnet of zone file).

6. Check BIND configuration syntax

7. Start BIND

Primary DNS server is now setup and ready to respond to DNS queries.

8. Configure DNS clients

Before the servers identified in the ACL can query the DNS servers, they must be configured to use new nameservers.

9. Test Clients

In the DNS server, test the forward and reverse lookups using nslookup utility.

Router:

The Router is implemented as a virtual machine with static routes that enable routing between different subnets in the VPC.

Feature Implementation:

1. Resolution within Zones:

The DNS server that resides in a subnet is configured such that it is capable of resolving the IP addresses of Web Servers that reside in other subnets within the Private Network.

The steps involved are:

1. The different zones/subnets are listed in the named.conf file.
2. Separate forward and reverse resolution files for each subnet are created with mapping entries of the servers present in that subnet.
3. Replicate the forward and reverse resolution files in all the DNS servers that needs to do the DNS resolution
4. Configure the client's resolv.conf file to point to the name server present in the same subnet / different subnet.

2. Load Balancing of Servers:

1. Add the IP addresses of the servers under the same domain name as records in the Forward and reverse resolution zone files.
2. Replicate the forward and reverse resolution files in all the DNS servers that needs to do the DNS resolution
3. Configure the client's resolv.conf file to point to the required DNS server.

3. Location / Source IP Based Resolution :

We implement the Location Aware resolution with the help of views. Views allow to define different virtual configurations within the same server and also to specify which client should be sent which IP addresses as response for a DNS resolution request. The steps involved are :

1. Add the subnets that need to be resolved under separate ACLs in the named.conf file
2. Create views that match the acl and the resolution database that much be fetched on a request from the clients in the corresponding acl.

```
acl net1 {
    192.168.10.0/24;
};

acl net2 {
    192.168.20.0/24;
};

view net1 {
    match-clients { net1; };
    include "/etc/named.rfc1912.zones";
    include "/etc/named.root.key";

    zone "univ.edu" {
        type master;
        file "/etc/named/zones/db.univ.edu-net1.zone";
    };
    zone "10.168.192.in-addr.arpa" {
        type master;
        file "/etc/named/zones/db.192.168.10";
    };
};

view net2 {
    match-clients { net2; };

    zone "univ.edu" {
        type master;
        file "/etc/named/zones/db.univ.edu-net2.zone";
    };
    zone "20.168.192.in-addr.arpa" {
        type master;
        file "/etc/named/zones/db.192.168.20";
    };
};
```

3. Create forward and reverse resolution files for the views created in the named.conf with the IP and domain name mapping.

```
$TTL 604800
@      IN      SOA      univ.edu. admin.univ.edu. (
                        3      ; Serial
                        604800 ; Refresh
                        86400  ; Retry
                        2419200 ; Expire
                        604800 ) ; Negative Cache TTL
;
; name servers - NS records
IN      NS      dns1.univ.edu.

; name servers - A records
dns1.univ.edu.      IN      A      192.168.20.100

; 192.168.10.0/24 - A records
www.csc      IN      A      192.168.10.4
www.ece      IN      A      192.168.10.5
```

```
$TTL 604800
@      IN      SOA      univ.edu. admin.univ.edu. (
                        4      ; Serial
                        604800 ; Refresh
                        86400  ; Retry
                        2419200 ; Expire
                        604800 ) ; Negative Cache TTL
;
; name servers - NS records
IN      NS      dns1.univ.edu.

; name servers - A records
dns1.univ.edu.      IN      A      192.168.20.100

; 192.168.20.0/24 - A records
www.csc      IN      A      192.168.20.4
www.csc      IN      A      192.168.20.6
www.ece      IN      A      192.168.20.5
```

3. Replicate the forward and reverse resolution files in all the DNS servers that needs to do the DNS resolution.
4. Configure the client's resolv.conf file to point to the required DNS server.

4. High Availability / DNS Failover

1. A secondary DNS server is configured with same configurations in the named.conf file and the database files for each zone.

2. The client's resolv.conf should contain both the name server's IP address as records to look for whenever a resolution request is sent out.

5. Phishing Prevention:

1. A namespace that acts a client whose DNS resolution requests should be dropped is created.

2. The ip tables of the DNS server is modified by adding an INPUT chain rule.

```
Iptables -I INPUT -p udp --dport 53 -s <client ip> -j DROP
```

```
Iptables -I INPUT -p tcp --dport 53 -s <client ip> -j DROP
```

3. Configure the client's resolv.conf file to point to the DNS server

4. When the client makes a request, the DNS server now drops the request.

Containers :

DNS servers are containers with a custom Centos image that has all tools installed including bind-utils, systemctl etc

```
rraman2@bn16-204:~/dns_image$ cat dockerfile
FROM centos:7

MAINTAINER TEAM2-CLOUDDNS

include /etc/named.rfc1912.zones;

RUN (cd /lib/systemd/system/sysinit.target.wants/; for i in *; do [ $i == \
systemd-tmpfiles-setup.service ] || rm -f $i; done); \
rm -f /lib/systemd/system/multi-user.target.wants/*;\
rm -f /etc/systemd/system/*.wants/*;\
rm -f /lib/systemd/system/local-fs.target.wants/*; \
rm -f /lib/systemd/system/sockets.target.wants/*udev*; \
rm -f /lib/systemd/system/sockets.target.wants/*initctl*; \
rm -f /lib/systemd/system/basic.target.wants/*;\
rm -f /lib/systemd/system/anaconda.target.wants/*; cat unlv.edu-net1.zone
RUN yum update
RUN yum -y install iproute
RUN yum -y install iputils
RUN yum -y install openssh-server
RUN yum -y install traceroute
RUN yum -y install iptables
RUN yum -y install tcpdump
RUN yum -y install python
RUN yum -y install epel-release
RUN yum -y install python-pip
RUN pip install paramiko
RUN yum -y install bind bind-utils
RUN yum -y install initscripts
VOLUME [ "/sys/fs/cgroup" ]
CMD ["/usr/sbin/init"]

rraman2@bn16-204:~/dns_image$
```

The servers and client in the VPC are base ubuntu images with ip tools and ping tool installed.

GRE tunnel in OVS Bridges:

```
snaatta@bn17-157:~$ sudo ovs-vsctl show
70a68a99-87f0-4750-b6de-e267fd66d1d0
    Bridge "l2br"
        Port "gre1"
            Interface "gre1"
                type: gre
                options: {remote_ip="152.46.16.204"}
        Port "dns30l2br1"
            Interface "dns30l2br1"
        Port "l2br"
            Interface "l2br"
                type: internal
        Port "its1l2br1"
            Interface "its1l2br1"
    ovs_version: "2.5.4"
snaatta@bn17-157:~$
```

```

framan2@bn16-204:~$ sudo ovs-vsctl show
398093bc-8c72-4793-ab85-1309a7eb70a1
    Bridge "l2br"
        Port "child11"
            Interface "child11"
        Port "l2br"
            Interface "l2br"
                type: internal
        Port "c1l21"
            Interface "c1l21"
        Port "veth3"
            Interface "veth3"
        Port "gre0"
            Interface "gre0"
                type: gre
                options: {remote_ip="152.46.17.157"}
        Port "child21"
            Interface "child21"
        Port "parent1"
            Interface "parent1"
        Port "c2l21"
            Interface "c2l21"
        Port "cscs1l21"
            Interface "cscs1l21"
        Port "routerl211"
            Interface "routerl211"
        Port "veth1"
            Interface "veth1"
        Port "dns1l2br1"
            Interface "dns1l2br1"
        Port "eces1l21"
            Interface "eces1l21"
    Bridge "l2br1"
        Port "c3l21"
            Interface "c3l21"
        Port "dns2l2br11"
            Interface "dns2l2br11"
        Port "cscs2l21"
            Interface "cscs2l21"
        Port "routerl221"
            Interface "routerl221"
            options: {remote_ip="152.46.16.204"}
        Port "l2br1"
            Interface "l2br1"
                type: internal
        Port "cscs3l21"
            Interface "cscs3l21"
        Port "eces2l21"
            Interface "eces2l21"
    ovs_version: "2.5.4"
framan2@bn16-204:~$

```

Automation:

To automate the entire DNS configuration inside the VPC, a Python application is implemented that gets input from the user through a Northbound interface and communicates the data to the DNS configuration Ansible through a south bound interface.

First Level of Automation:

- Assuming that the customer has a fully setup VPC ready and reachable and ready to configure additional DNS servers, the application gets the subnets in which the DNS servers need to be configured followed by the number of DNS servers.
- The IP address plan of the existing VPC is read by the application as a config file in python
- A parent DNS that creates a hierarchy DNS service and that forwards the DNS request to the authoritative DNS servers in each subnet.
- Authoritative DNS servers that resolve the URLs to IP addresses are created
- Both the DNS servers are containers in the hypervisors.

```
rraman2@bn16-204:~/jaga/try_awsible/ansible_files$ python startup.py
Welcome to our Cloud DNS service. We help you provide DNS service to your Private Cloud setup. Enjoy the Ride!!
Enter the number of subnets in your private cloud
2
Enter subnet/s in which to create DNS seperating each subnet by comma Ex:192.168.10.0/24,192.168.20.0/24
192.168.10.0/24,192.168.20.0/24
Enter Zone Name
univ.edu
zone name: univ.edu.
Enter the bridge name to which your subnet is connected to eg:l2,l2br
l2br
Enter the bridge name to which your subnet is connected to eg:l2,l2br
l2brl1
Allow query parameter to ansible192.168.10.0/24; 192.168.20.0/24;

Forward zone names: ['univ.edu.-net1.zone', 'univ.edu.-net2.zone']

Net server list:1[OrderedDict([('hostname', 'www.csc'), ('ip', '192.168.10.5')]), OrderedDict([('hostname', 'www.ece'), ('ip', '192.168.10.4')])]

Net server list:2[OrderedDict([('hostname', 'www.csc'), ('ip', '192.168.20.4')]), OrderedDict([('hostname', 'www.csc'), ('ip', '192.168.20.6')]), OrderedDict([('hostname', 'www.ece'), ('ip', '192.168.20.5')])]

Networks Dict: OrderedDict([('net', 'net1'), ('ip', '192.168.10.0/24'), ('view', 'net1-view'), ('forward_zone', 'univ.edu.-net1.zone'), ('rev_zone', '10.168.192')])

Networks Dict: OrderedDict([('net', 'net2'), ('ip', '192.168.20.0/24'), ('view', 'net2-view'), ('forward_zone', 'univ.edu.-net2.zone'), ('rev_zone', '20.168.192')])

Networks list: [OrderedDict([('net', 'net2'), ('ip', '192.168.20.0/24'), ('view', 'net2-view'), ('forward_zone', 'univ.edu.-net2.zone'), ('rev_zone', '20.168.192')]), OrderedDict([('net', 'net2'), ('ip', '192.168.20.0/24'), ('view', 'net2-view'), ('forward_zone', 'univ.edu.-net2.zone'), ('rev_zone', '20.168.192')])]
```

Figure : The Python Application that gets VPC info and subnet plan from User

Second Level of Automation:

Ansible:

- Ansible receives the IP address plan and the DNS config data from the Python application
- Bind9 module of Ansible is used to install these configurations in the created DNS servers.
- The yaml file is ran with the host as the ip address of the DNS container
- DNS service is started and ensured that NSlookup happens.

```
[root@9002a14738e8 ansible_files]# ansible-playbook main.yaml
[WARNING]: provided hosts list is empty, only localhost is available. Note
that the implicit localhost does not match 'all'
Welcome to Ubuntu 16.04.3 LTS (GNU/Linux 4.4.0-109-generic x86_64)
PLAY [Configure DNS using bind] *****
TASK [Gathering Facts] *****
ok: [localhost]
TASK [Install bind and bind-utils package] *****
ok: [localhost] => (item=[u'bind', u'bind-utils'])
TASK [Create custom named.conf with desired zone] *****
ok: [localhost]
TASK [Copy zone forward files for all zones to /var/named] *****
ok: [localhost] => (item=univ.edu-net1.zone)
TASK [Copy zone reverse files for all zones to /var/named] *****
ok: [localhost]
TASK [Disable IPv6 support] *****
ok: [localhost]
TASK [Start and enable bind service] *****
ok: [localhost]
PLAY RECAP *****
localhost : ok=7    changed=0    unreachable=0    failed=0
[root@9002a14738e8 ansible_files]#
```

Validation:

1. Resolution from same Subnet:

```
snaatta@bn19-196:~$ sudo ip netns exec client bash
root@bn19-196:~# cat /etc/resolv.conf
# Dynamic resolv.conf(5) file for glibc resolver(3) generated by resolvconf(8)
#     DO NOT EDIT THIS FILE BY HAND -- YOUR CHANGES WILL BE OVERWRITTEN
nameserver 192.168.10.100
search univ.edu
root@bn19-196:~# ping www.csc.univ.edu -c 5
PING www.csc.univ.edu (192.168.10.4) 56(84) bytes of data.
64 bytes from www.csc.univ.edu (192.168.10.4): icmp_seq=1 ttl=64 time=0.153 ms
64 bytes from www.csc.univ.edu (192.168.10.4): icmp_seq=2 ttl=64 time=0.070 ms
64 bytes from www.csc.univ.edu (192.168.10.4): icmp_seq=3 ttl=64 time=0.050 ms
64 bytes from www.csc.univ.edu (192.168.10.4): icmp_seq=4 ttl=64 time=0.054 ms
64 bytes from www.csc.univ.edu (192.168.10.4): icmp_seq=5 ttl=64 time=0.082 ms

--- www.csc.univ.edu ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 3999ms
rtt min/avg/max/mdev = 0.050/0.081/0.153/0.039 ms
root@bn19-196:~# ping www.ece.univ.edu -c 5
PING www.ece.univ.edu (192.168.10.5) 56(84) bytes of data.
64 bytes from www.ece.univ.edu (192.168.10.5): icmp_seq=1 ttl=64 time=0.122 ms
64 bytes from www.ece.univ.edu (192.168.10.5): icmp_seq=2 ttl=64 time=0.351 ms
64 bytes from www.ece.univ.edu (192.168.10.5): icmp_seq=3 ttl=64 time=0.073 ms
64 bytes from www.ece.univ.edu (192.168.10.5): icmp_seq=4 ttl=64 time=0.060 ms
64 bytes from www.ece.univ.edu (192.168.10.5): icmp_seq=5 ttl=64 time=0.098 ms

--- www.ece.univ.edu ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 3999ms
rtt min/avg/max/mdev = 0.060/0.140/0.351/0.108 ms
root@bn19-196:~#
```

When the client pings `www.csc.univ.edu`, DNS resolves the domain name by providing the IP address (192.168.10.4). Similarly when the client pings `www.ece.univ.edu`, it reaches 192.168.10.5. Therefore, the IP address are properly resolved by the DNS server that shows all the configurations in client and DNS server are perfect.

2. Load Balancing among Servers:

```
[root@localhost ~]# cat /etc/resolv.conf
# Generated by NetworkManager
nameserver 192.168.20.100
nameserver 192.168.10.100
[root@localhost ~]# ping www.csc.univ.edu -c 5
PING www.csc.univ.edu (192.168.20.4) 56(84) bytes of data.
64 bytes from www.csc.univ.edu (192.168.20.4): icmp_seq=1 ttl=64 time=3.01 ms
64 bytes from www.csc.univ.edu (192.168.20.4): icmp_seq=2 ttl=64 time=0.668 ms
64 bytes from www.csc.univ.edu (192.168.20.4): icmp_seq=3 ttl=64 time=1.51 ms
64 bytes from www.csc.univ.edu (192.168.20.4): icmp_seq=4 ttl=64 time=1.38 ms
64 bytes from www.csc.univ.edu (192.168.20.4): icmp_seq=5 ttl=64 time=1.01 ms

--- www.csc.univ.edu ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4006ms
rtt min/avg/max/mdev = 0.668/1.518/3.012/0.803 ms
[root@localhost ~]# ping www.csc.univ.edu -c 5
PING www.csc.univ.edu (192.168.20.6) 56(84) bytes of data.
64 bytes from www.csc.univ.edu (192.168.20.6): icmp_seq=1 ttl=64 time=0.679 ms
64 bytes from www.csc.univ.edu (192.168.20.6): icmp_seq=2 ttl=64 time=0.484 ms
64 bytes from www.csc.univ.edu (192.168.20.6): icmp_seq=3 ttl=64 time=0.578 ms
64 bytes from www.csc.univ.edu (192.168.20.6): icmp_seq=4 ttl=64 time=0.744 ms
64 bytes from www.csc.univ.edu (192.168.20.6): icmp_seq=5 ttl=64 time=0.377 ms

--- www.csc.univ.edu ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4004ms
rtt min/avg/max/mdev = 0.377/0.572/0.744/0.133 ms
[root@localhost ~]# ping www.ece.univ.edu -c 5
PING www.ece.univ.edu (192.168.20.5) 56(84) bytes of data.
64 bytes from www.ece.univ.edu (192.168.20.5): icmp_seq=1 ttl=64 time=0.449 ms
64 bytes from www.ece.univ.edu (192.168.20.5): icmp_seq=2 ttl=64 time=0.366 ms
64 bytes from www.ece.univ.edu (192.168.20.5): icmp_seq=3 ttl=64 time=0.311 ms
64 bytes from www.ece.univ.edu (192.168.20.5): icmp_seq=4 ttl=64 time=0.485 ms
64 bytes from www.ece.univ.edu (192.168.20.5): icmp_seq=5 ttl=64 time=0.350 ms

--- www.ece.univ.edu ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4002ms
rtt min/avg/max/mdev = 0.311/0.392/0.485/0.065 ms
[root@localhost ~]#
```

Here load balancing scenario is depicted. Above screenshot shows that when a client first tries to reach `www.csc.univ.edu`, DNS resolves this query with `192.168.20.4` as the IP address. Next time when the client tries to reach the same address, DNS provides the IP as `192.168.20.6`. Round-robin fashion is followed by the DNS in resolving the names.

3. Logs for the DNS Resolution (/var/log/messages)

```
Apr 10 20:05:46 localhost named[20168]: client 192.168.10.3#42456 (www.csc.univ.edu): view net1: query:
www.csc.univ.edu IN A + (192.168.10.100)
Apr 10 20:05:46 localhost named[20168]: client 192.168.10.3#48736 (4.10.168.192.in-addr.arpa): view net1
: query: 4.10.168.192.in-addr.arpa IN PTR + (192.168.10.100)
Apr 10 20:07:44 localhost named[20168]: received control channel command 'querylog'
Apr 10 20:07:44 localhost named[20168]: query logging is now off
Apr 10 20:14:11 localhost named[20168]: received control channel command 'querylog'
Apr 10 20:14:11 localhost named[20168]: query logging is now on
Apr 10 20:14:26 localhost named[20168]: client 192.168.20.3#50662 (www.csc.univ.edu): view net2: query:
www.csc.univ.edu IN A + (192.168.10.100)
Apr 10 20:14:31 localhost named[20168]: client 192.168.20.3#40537 (4.20.168.192.in-addr.arpa): view net2
: query: 4.20.168.192.in-addr.arpa IN PTR + (192.168.10.100)
```

Logs for forward and reverse look up is recorded using name server control utility, `RNDC`. When the client from different subnet queries the DNS, appropriate A records are fetched.

4. High Availability/ DNS Failover:

```
root@localhost:~# ping 192.168.20.100
PING 192.168.20.100 (192.168.20.100) 56(84) bytes of data.
From 192.168.20.3 icmp_seq=1 Destination Host Unreachable
From 192.168.20.3 icmp_seq=2 Destination Host Unreachable
From 192.168.20.3 icmp_seq=3 Destination Host Unreachable
From 192.168.20.3 icmp_seq=4 Destination Host Unreachable
^C
--- 192.168.20.100 ping statistics ---
5 packets transmitted, 0 received, 100% packet loss, time 4001ms
pipe 4
[root@localhost ~]# ping www.csc.univ.edu -c 5
PING www.csc.univ.edu (192.168.20.6) 56(84) bytes of data.
64 bytes from www.csc.univ.edu (192.168.20.6): icmp_seq=1 ttl=64 time=1.01 ms
64 bytes from www.csc.univ.edu (192.168.20.6): icmp_seq=2 ttl=64 time=0.340 ms
64 bytes from www.csc.univ.edu (192.168.20.6): icmp_seq=3 ttl=64 time=0.332 ms
64 bytes from www.csc.univ.edu (192.168.20.6): icmp_seq=4 ttl=64 time=0.369 ms
64 bytes from www.csc.univ.edu (192.168.20.6): icmp_seq=5 ttl=64 time=0.266 ms

--- www.csc.univ.edu ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 8012ms
rtt min/avg/max/mdev = 0.266/0.405/1.019/0.279 ms
root@localhost:~#

snaatta@bn19-196:~$ sudo tcpdump -i vnet1
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on vnet1, link-type EN10MB (Ethernet), capture size 262144 bytes
19:34:05.794697 ARP, Request who-has 192.168.20.100 tell 192.168.20.3, length 28
19:34:05.911522 IP6 fe80::7c94:fdff:fea6:913f.mdns > ff02::fb.mdns: 0 PTR (QM)? 100.20.168.192.in-addr.arpa. (45)
19:34:06.795979 ARP, Request who-has 192.168.20.100 tell 192.168.20.3, length 28
19:34:06.912146 IP6 fe80::7c94:fdff:fea6:913f.mdns > ff02::fb.mdns: 0 PTR (QM)? 100.20.168.192.in-addr.arpa. (45)
19:34:07.797967 ARP, Request who-has 192.168.20.100 tell 192.168.20.3, length 28
19:34:08.914383 IP6 fe80::7c94:fdff:fea6:913f.mdns > ff02::fb.mdns: 0 PTR (QM)? 100.20.168.192.in-addr.arpa. (45)
19:34:09.796201 ARP, Request who-has 192.168.20.100 tell 192.168.20.3, length 28
19:34:16.312116 ARP, Request who-has 192.168.20.100 tell 192.168.20.3, length 28
19:34:17.313939 ARP, Request who-has 192.168.20.100 tell 192.168.20.3, length 28
19:34:20.321753 IP 192.168.20.3.35701 > 192.168.10.100.domain: 46670+ A? www.csc.univ.edu. (34)
19:34:20.321753 IP 192.168.10.100.domain > 192.168.20.3.35701: 46670* 2/1/1 A 192.168.20.4 (101)
19:34:20.323968 IP 192.168.20.3 > 192.168.20.6: ICMP echo request, id 6123, seq 1, length 64
19:34:20.325031 ARP, Request who-has 192.168.20.100 tell 192.168.20.3, length 28
19:34:20.418866 IP6 fe80::7c94:fdff:fea6:913f.mdns > ff02::fb.mdns: 0 PTR (QM)? 100.10.168.192.in-addr.arpa. (45)
19:34:21.325928 ARP, Request who-has 192.168.20.100 tell 192.168.20.3, length 28
19:34:21.422720 IP6 fe80::7c94:fdff:fea6:913f.mdns > ff02::fb.mdns: 0 PTR (QM)? 100.10.168.192.in-addr.arpa. (45)
19:34:25.329870 ARP, Request who-has 192.168.20.1 tell 192.168.20.3, length 28
19:34:25.330615 IP 192.168.20.3.34396 > 192.168.10.100.domain: 9982+ PTR? 6.20.168.192.in-addr.arpa. (43)
```

To demonstrate HA, the link to DNS server (192.168.20.100) is set down and the client still able to ping www.csc.univ.edu via the other DNS server. This depicts our DNS services are made highly available by having multiple DNS servers. Log screenshot is also attached to show that DNS 192.168.20.100 is unavailable and 192.168.10.100 is contacted regarding resolution of www.csc.univ.edu that provides the 2 addresses 192.168.20.4 and 192.168.20.6. Hence HA is achieved.

5. Phishing Prevention:

```
root@bn19-196:~# sudo ip addr show
1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN group default qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
47: client20@if46: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
    group default qlen 1000
    link/ether 5a:56:1c:12:ee:e4 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 192.168.10.2/24 scope global client20
        valid_lft forever preferred_lft forever
    inet6 fe80::5856:1cff:fe12:eee4/64 scope link
        valid_lft forever preferred_lft forever
root@bn19-196:~# ping www.csc.univ.edu
PING www.csc.univ.edu (192.168.10.4) 56(84) bytes of data.
64 bytes from www.csc.univ.edu (192.168.10.4): icmp_seq=1 ttl=64 time=0.330 ms
64 bytes from www.csc.univ.edu (192.168.10.4): icmp_seq=2 ttl=64 time=0.107 ms
64 bytes from www.csc.univ.edu (192.168.10.4): icmp_seq=3 ttl=64 time=0.072 ms
64 bytes from www.csc.univ.edu (192.168.10.4): icmp_seq=4 ttl=64 time=0.072 ms
64 bytes from www.csc.univ.edu (192.168.10.4): icmp_seq=5 ttl=64 time=0.075 ms
^C
--- www.csc.univ.edu ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 3997ms
rtt min/avg/max/mdev = 0.072/0.131/0.330/0.100 ms
root@bn19-196:~# ping www.csc.univ.edu
ping: unknown host www.csc.univ.edu
root@bn19-196:~# cat /etc/resolv.conf
# Dynamic resolv.conf(5) file for glibc resolver(3) generated by resolvconf(8)
#     DO NOT EDIT THIS FILE BY HAND -- YOUR CHANGES WILL BE OVERWRITTEN
nameserver 192.168.10.100
search univ.edu
root@bn19-196:~#

root@localhost:~# iptables -I INPUT -p udp --dport 53 -s 192.168.10.2 -j DROP
root@localhost:~# iptables -I INPUT -p tcp --dport 53 -s 192.168.10.2 -j DROP
root@localhost:~#
```

```
Apr 10 20:34:39 localhost named[20168]: client 192.168.10.2#50934 (www.csc.univ.edu): view net1: query: www.csc.univ.edu IN A + (192.168.10.100)
Apr 10 20:34:39 localhost named[20168]: client 192.168.10.2#35050 (4.10.168.192.in-addr.arpa): view net1: query: 4.10.168.192.in-addr.arpa IN PTR + (192.168.10.100)
Apr 10 20:35:44 localhost kernel: ip_tables: (C) 2000-2006 Netfilter Core Team
```

Iptables modification on the DNS server to drop incoming DNS request from a specific client (192.168.10.2) and the log message captured at the DNS server.