Citrix NetScaler Global Server Load Balancing Primer:

Theory and Implementation

Background

Global Server Load Balancing (GSLB) describes a range of technologies to distribute resources around the Internet for various purposes. Depending on the user and nature of deployment, different goals are targeted, which include:

1. Disaster recovery--Providing an alternate location for accessing a resource in the event of a failure, or to provide a means of shifting traffic easily to simplify maintenance (or both)
2. Load sharing--Distributing traffic between multiple locations:
    a. To minimize bandwidth costs
    b. Limit the capacity used at a given location
    c. Limit exposure to various issues, including outages, geographic disruption, and so on.
3. Performance--To position content closer to users, which enhances the user's experience
4. Legal Obligations-- Present users with different versions of resources based on political location

Over time, various techniques have been developed to meet these requirements, although three techniques have withstood the test of time:

1. DNS based redirection to different locations;
2. Content level redirection to different locations, using HTTP redirection;
3. Route Injection to advertise the same IP address to multiple locations.

Route health injection or RHI, is generally uses different methods more applicable to large companies than the first two, and is not covered in this guide. The other two techniques, that of DNS and Content level redirection, are covered, with examples and various configurations.

This guide covers the basics of DNS and GSLB configuration that are necessary when deploying GSLB on the Citrix NetScaler Application Delivery Appliance. While the basics of the commands used for configuring GSLB are detailed in this document, additional information on each command and a complete set of options can be found in the Citrix NetScaler Command Reference Guide, and additional details on various configurations are also found in the Traffic Management Guide.

DNS Overview

To fully understand GSLB, it is first necessary to understand, at a high level, DNS (Domain Name Server) functionality as it relates to GSLB.

DNS is composed of several components:

1. A client, and in particular a client's DNS "Resolver" or the piece of code that is responsible for interacting with the rest of the DNS infrastructure and caching results for the client.
2. A client's "Resolving Nameserver", which the client resolver depends on to perform DNS resolution, and in general, also provides an additional layer of caching to speed up responses for users.
3. Authoritative Nameservers, which include the Root Nameservers and TLD (Top Level Domain) Nameservers.

Caching complicates the understanding of the full process that results in DNS resolution so it is ignored with the assumption that every query results in a full DNS resolution. Consider the following example, as shown in Diagram 1:

1. The user's resolver generates a packet to the local resolving nameserver.
2. The resolving nameserver sends a query to the root nameserver, which handles resolution for all domains on the internet.
3. The root nameserver determines that it does not know the answer, but knows another server that might know; this would be the TLD nameservers for the ".com" subdomain. The root nameserver then replies with a delegation to the proper TLD nameserver.
4. The resolving nameserver then generates a new request to the TLD nameserver.
5. The TLD nameserver decides that, while it does not know the answer, another nameserver might know, which is the authoritative nameserver for the "domain.com" name. As a result, the TLD nameserver sends a delegation response to the resolving nameserver indicating where to find this nameserver.
6. The resolving nameserver again generates a request to the new nameserver.
7. The new nameserver decides that it knows the appropriate records to send back for www.domain.com and does so.
8. Response in hand, the resolving nameserver returns a response to the user's resolver and the application can now establish a connection to the proper location.

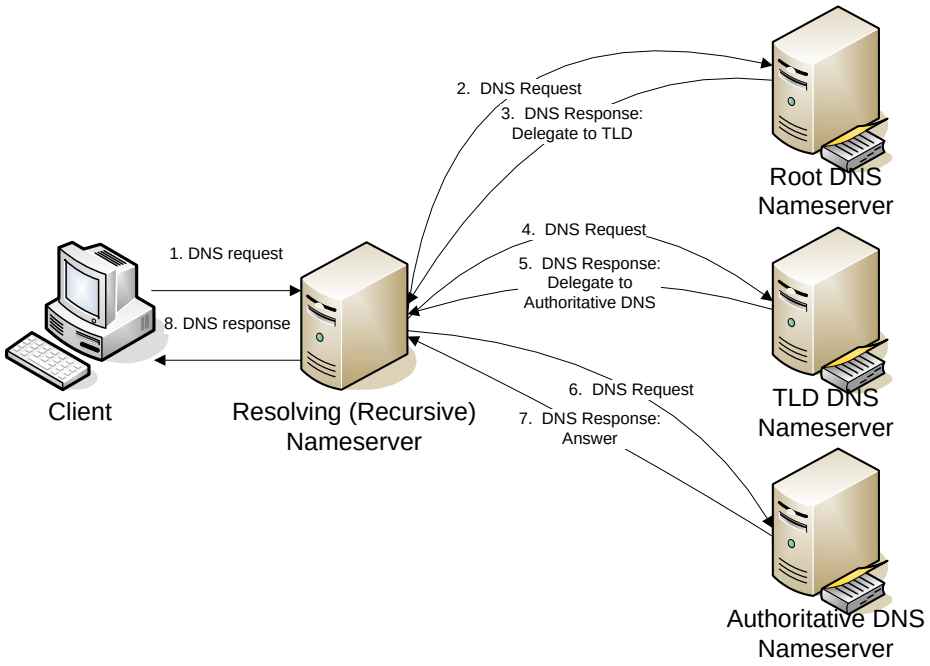


Diagram 1: Normal DNS Packet flow

There are two common misconceptions about how this process works:

1. Each hop in the process makes the query to the next server in the loop when delegation occurs. This is false.
2. The client's resolver performs the iterative process, and therefore the final nameserver would see the actual client's IP address. This is also false.

Additionally, there are further issues that make this process more complicated:

1. Each client may change which resolving nameserver is used based on various parameters (such as timeouts). As such, different IP addresses may be present at a given authoritative domain nameserver for a given client.
2. Resolving nameserver cache responses, and can deliver them to many different clients.
3. Resolving nameservers may override the time to live value set in a response, which in theory is the time that a response may be cached. Therefore, a response that is intended to be cached for 10 seconds may in fact be cached for an hour.
4. Client side resolvers and applications have different cache behaviors to consider as well.
5. Clients may be use DNS resolvers that are geographically dispersed. The DNS request may come from a different location than the client.

Because of these complications, testing and verifying the functionality of GSLB can be a complex task, but is necessary to ensure that business needs are met.

How DNS level GSLB works

GSLB based on DNS works exactly the same way as normal DNS, with the exception that more logic is used to determine what addresses to return. The logic in most situations is based on:

1. The load and capacity of resources on the network.
2. The originating IP address of the request or interface it was received on.
3. Previous requests made from the same IP or network.
4. The health state of resources.

To ensure the various pieces of information are in place, the NetScaler makes use of the following methods to determine the state for proper decision making:

1. Explicit monitors that check for availability of remote resources by accessing the resource itself.
2. Metric Exchange Protocol (MEP), which is a channel of communication between distinct NetScaler devices to share state information with each other.
3. SNMP based load monitors, which poll a remote resource for statistics such as CPU load, network load, and so on.

Multiple methods can be used to provide for more system redundancy. MEP and monitors can both determine the availability of a resource. Details of how to configure each of these appear later in this guide.

To apply the above logic, the system must first receive the DNS request in order to generate an appropriate response. There are several topologies that allow this, the first being an ADNS setup, where the system behaves as the authoritative nameserver. This is shown in Diagram 2.

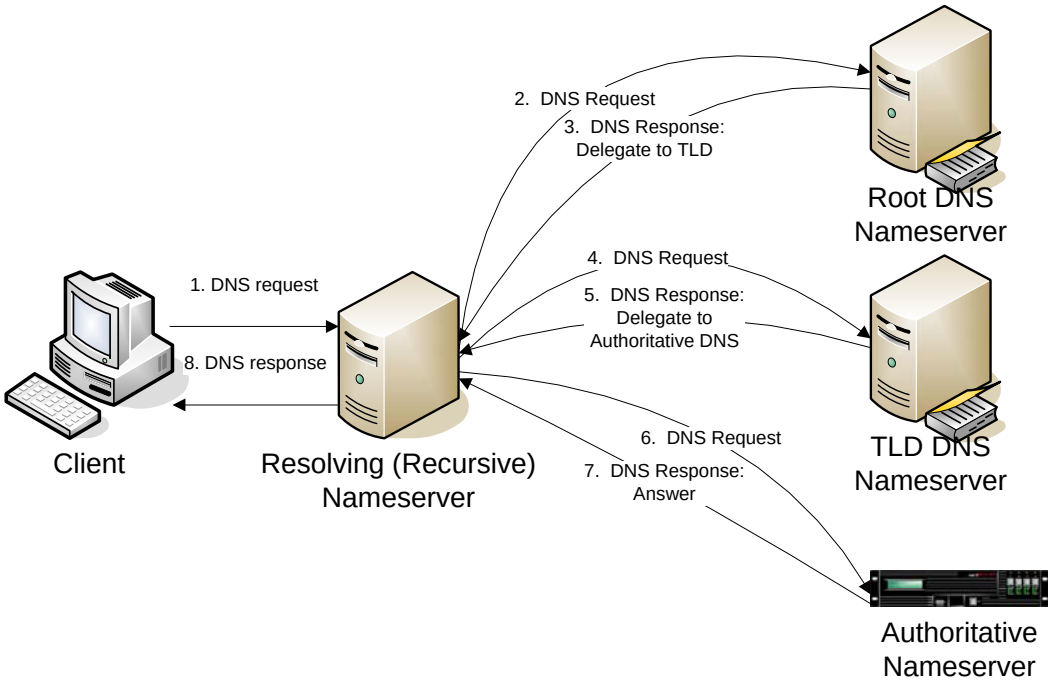


Diagram 2: Authoritative DNS configuration

The next topology, shown in Diagram 3, is widely used and is a modification of the authoritative nameserver configuration above, but uses a sub-domain for GSLB functions. Here, the main authoritative DNS server is not replaced by the NetScaler, but simply delegates a sub-domain to the system for resolution. For example, to globally load balance the name “www” in the DNS domain “domain.com”, the authoritative nameserver is configured as follows (Note: this same set of names is leveraged in later examples, and builds upon this configuration):

```
www    IN    NS    nyc1
       IN    NS    sjc1
       IN    NS    hk1
       IN    NS    lon1
nyc1   IN    A     10.0.1.1
sjc1   IN    A     10.0.2.1
hk1    IN    A     10.0.3.1
lon1   IN    A     10.0.4.1
```

The first line specifies that the name “www”, inside the zone “domain.com”, is delegated to four different nameservers specified by the names nyc1, sjc1, hk1, and lon1. The IP addresses are also included in the zone, and are provided to the requesting nameserver. Once the resolving nameserver receives this, it picks one or more of the IP addresses, and sends the request to that IP next.

An alternate nameserver configuration would delegate www using a cname to a distinct globally load balanced subdomain. A cname is a pointer to another name, so in this scenario, we point “www” to “www.gslb”, then delegate the “gslb” subdomain:

```
www    IN    CNAME www.gslb
GSLB   IN    NS    nyc1
       IN    NS    sjc1
       IN    NS    hk1
       IN    NS    lon1
nyc1   IN    A     10.0.1.1
sjc1   IN    A     10.0.2.1
hk1    IN    A     10.0.3.1
lon1   IN    A     10.0.4.1
```

The advantage of using a subdomain such as “gslb” is that, using cname, you can provide many different names to the NetScaler, and not have to explicitly delegate them all. For example, you could add the following configuration elements to the previous example without having to duplicate the delegations for each name:

```
www1   IN    CNAME www1.gslb
ftp    IN    CNAME ftp.gslb
qa     IN    CNAME qa.gslb
```

Either directly delegating names or using cnames ensures that full control of what is and is not handled by the NetScaler GSLB is controlled by the authoritative DNS server. At the same time, redundancy is provided so that if one nameserver (NetScaler) fails, then the remote nameserver automatically issues another request to another NetScaler, preventing the website from going down.

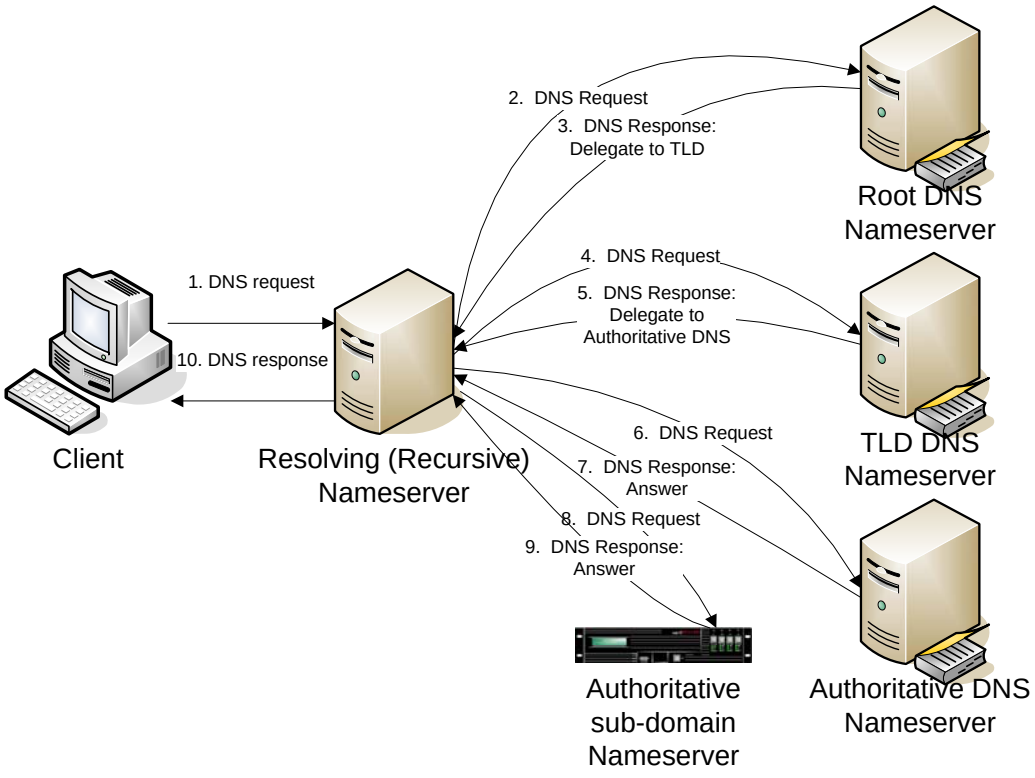


Diagram 3: Authoritative for GSLB sub-domain

Another topology is shown in Diagram 4. The NetScaler functions as a proxy to the authoritative nameserver, typically in conjunction with doing load balancing. This allows the user to leverage multiple capabilities of the system, while providing the GSLB functionality. The advantage of this configuration is that only domains that must be intercepted for GSLB are configured on the system. Any request for information that is not configured on the system, such as for unsupported record types, is passed to the back-end proxy for resolution.

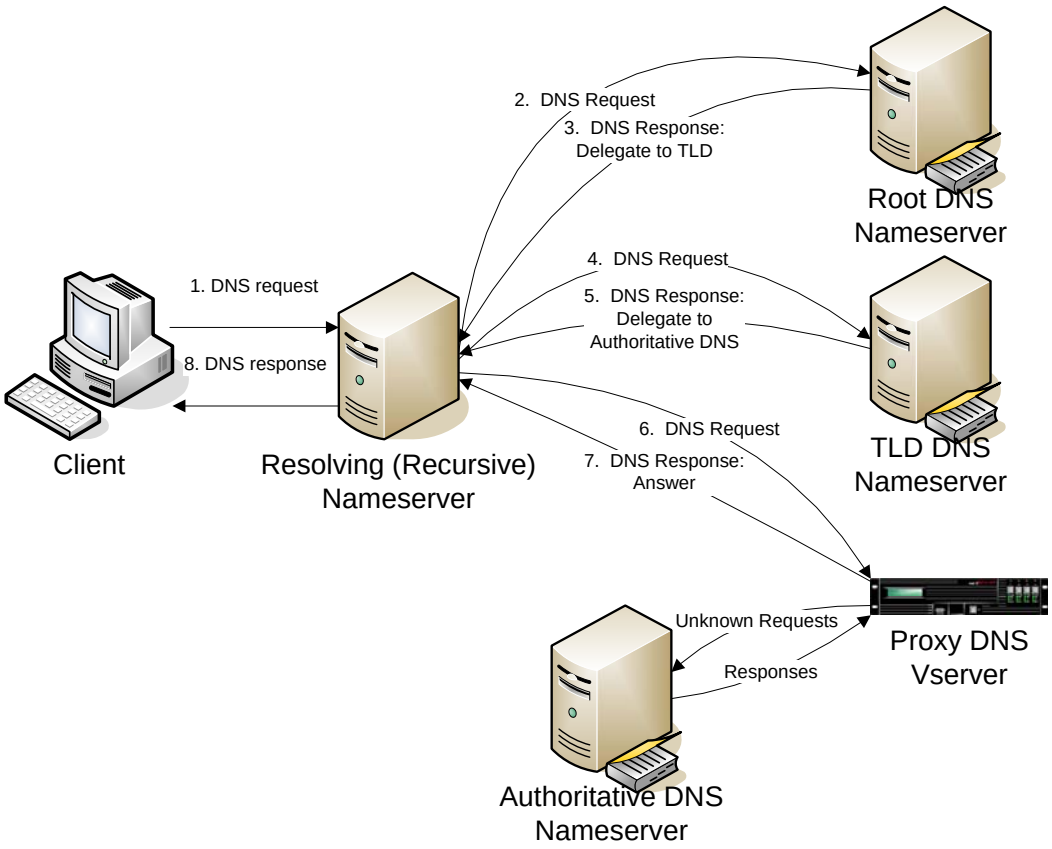


Diagram 4: DNS Proxy based GSLB configuration

A final, but less common configuration is to behave as a recursive DNS itself, as shown in diagram 5. The diagram depicts a scenario where multiple systems are working together to perform various functions. This topology is most useful in larger enterprises, where an existing topology uses a centralized DNS configuration which all the clients point to, and GSLB is being added for reliability. Here, the NetScaler takes the role of the resolving nameserver, and also participates in the GSLB mesh with other systems that are performing monitoring and load balancing. By using this configuration, GSLB can uniquely identify each client by actual IP address, when normally it can only identify the resolving nameserver that the client is using. In large enterprise environments, this scenario can provide faster failover, improved load balancing, and proximity detection for distributed applications.

To minimize the cost, the NetScaler VPX may be used for this type of configuration, as the VPX could be hosted on the same hardware as the original recursive nameserver using virtualization. Additional reliability can be gained through the use of RHI (Route Health Injection) on the internal network, while providing high redundancy and improved performance.

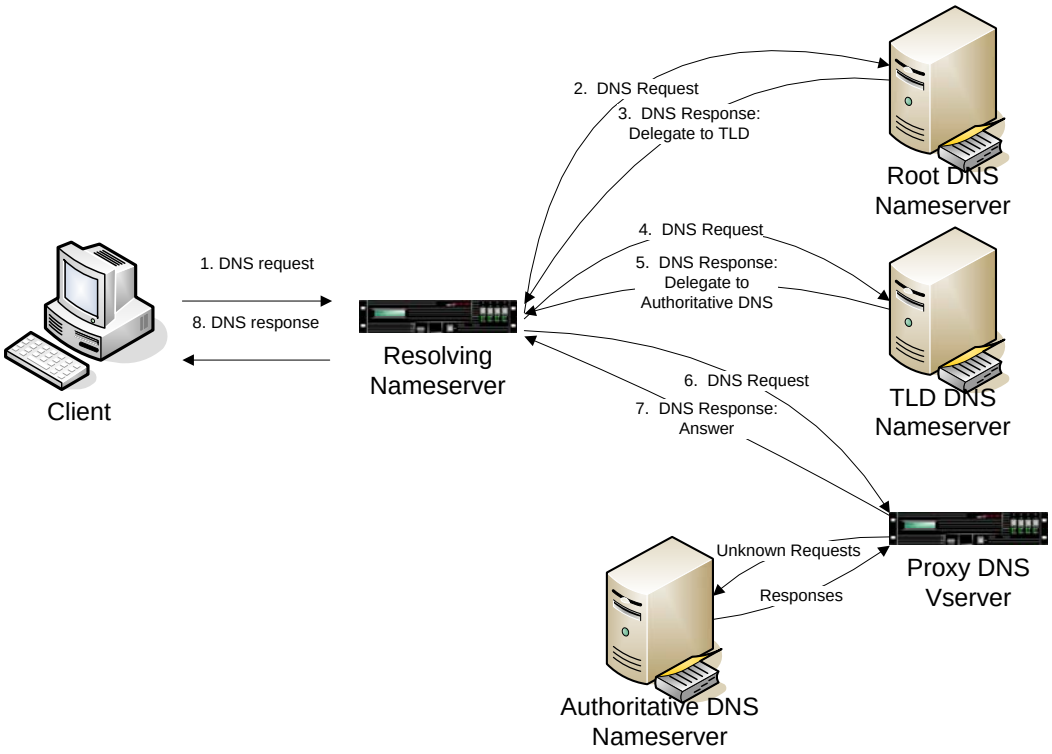


Diagram 5: NetScaler as a resolving nameserver for improved GSLB proximity

Basic NetScaler GSLB Configuration

There are three fundamental entities that must be configured: 1) the "site", 2) the GSLB vServer and 3) the GSLB service. A GSLB site represents a NetScaler or High Availability (HA) pair of NetScalers that contain GSLB state information, and provide information about how the NetScaler nodes should communicate. The GSLB vServer represents a grouping of resources to where users can be directed, along with the logic used to select one resource verses another. The GSLB service represents a target resource, and is bound to the GSLB vServer. The target resource may be a load balancing vServer on a NetScaler, or it could represent a third party server. Sites and services are inherently linked together to indicate proximity between the two, that is, all services must belong to a site, and are assumed to be in the same location as the "site" NetScaler for proximity purposes. Likewise, services and vServers are linked together, so as to link the logic to the resources that are available.

At the most basic level, each NetScaler participating in GSLB must have a site definition, which is used as part of the communication between each NetScaler, and provides overall options on what communication can occur to that site and how it should behave. For example:

```
add gslb site nyc 10.0.1.1
add gslb site sjc 10.0.2.1
add gslb site hk  10.0.3.1
add gslb site lon 10.0.4.1
```

Options that can be configured on GSLB sites include:

1. The internal and external IP addresses used for communication (in case there is a translating firewall between sites)
2. Usage of MEP to and from a given site
3. What level of monitoring should be done to resources at that site
4. Tiering (parent/child) relationship between sites (this applies to very large and advanced configurations, so it is not detailed here)

In general, the GSLB site IP address should be either a MIP or SNIP on the NetScaler, although it can be separate as well, if desired.

The next piece of the configuration puzzle is the GSLB service, which instructs the system as to where resources are located, and how to access them. Each GSLB service

Sample service definitions for these sites:

```
add gslb service nyc1 10.0.1.10 HTTP 80 –siteName nyc
add gslb service sjc1 10.0.2.10 HTTP 80 –siteName sjc
add gslb service hk1  10.0.3.10 HTTP 80 –siteName hk
add gslb service lon1 10.0.4.10 HTTP 80 –siteName lon
```

In some cases, services need to reside in a datacenter where there is no NetScaler performing local load balancing. In such a case, create a placeholder site, and disable MEP so that no data exchange is attempted, then associate the service with the placeholder site:

```
Add gslb site placeholder 1.1.1.1 –metricexchange disabled
Add gslb service remote1 10.0.10.10 HTTP 80 –siteName Placeholder
```

Each site that does exist is forced to poll the remote service for state information instead of using MEP with this configuration. In such cases, persistence options are limited, but in many simple cases, this is sufficient.

The third piece necessary for GSLB configuration is the GSLB vServer. Like in local load balancing, the GSLB vServer object is where GSLB services are bound together, and other options about the load balancing behavior are set. Additionally, the DNS names that should be used for resolution should also be bound to the GSLB vServer. For example:

```
add gslb vserver app_name HTTP
bind gslb vserver app_name –serviceName nyc1
bind gslb vserver app_name –serviceName sjc1
bind gslb vserver app_name –serviceName hk1
bind gslb vserver app_name –serviceName lon1
bind gslb vserver app_name –domainName www.domain.com
bind gslb vserver app_name –domainName www.gslb.domain.com
```

Any name that could show up at the system at either the HTTP or DNS level should be listed here, for example, the name www.domain.com may be a cname pointer to domain.com so DNS at the system level does not see www.domain.com, but at the HTTP level, it is still in the host header. Therefore it should be bound as shown.

With the creation of the site, the GSLB services and the GSLB vServer, and the proper bindings, a simple GSLB configuration is complete. However, configuration is far from over.

Accepting DNS requests

The next step is to configure a means of receiving the DNS requests. The three methods are configuring an ADNS service, configuring a DNS vServer, or creating a recursive listener. For an ADNS service:

```
add service adns_svc 10.0.1.1 adns 53
```

This assumes that 10.0.1.1 is a NetScaler owned IP address, and the NetScaler should only receive requests that it is configured to answer directly, such as if all records reside on the NetScaler for a domain or if a GSLB subdomain is delegated to that IP address.

For a DNS vServer in a proxy type setup, a simple example is:

```
add service dns_svc1 10.0.1.2 DNS 53 –svrTimeout 0
add lb vserver dns_vsvr DNS 10.0.1.1 53
bind lb vserver dns_vsvr dns_svc1
```

For a full recursive configuration as shown in diagram 5 above, the following configuration applies to the NetScaler being used as the recursive nameserver:

```
add dns nameServer 10.0.1.1 -local
set dns parameter –recursion ENABLED
```

The IP specified in the “add dns nameserver” command is the IP address that is listened for on DNS requests, and from there, all DNS caching and GSLB configurations also kick in.

Testing Basic GSLB functionality

Once the basic configuration is in place, the "host" command can be used on the NetScaler itself to verify if the appropriate IP address is replying with the correct information, as shown in the following example (using the recursive example above):

```
root@ns# host www.domain.com 10.0.1.1
Using domain server:
Name: 10.0.1.1
Address: 10.0.1.1#53
Aliases:

www.domain.com has address 10.0.4.10
```

At this point, a basic GSLB configuration is complete.

Persistence

Most applications require that some level of persistence be configured, either to ensure that the same user goes to the same server for the life of their session, or to the same site if state information is maintained in a local database. Therefore, understanding persistence is key to getting the configuration correct.

DNS level persistence can be used to ensure that an IP or subnet covering a range of IP addresses return with the same answer for a given period, until a period of time has passed without any queries from that IP address range. In general, when this is configured, it should be configured with at least a /24 subnet range, as often several DNS servers are configured on a client that are in the same /24 range. This helps ensure that even if one server fails or another server is used for some reason, the persistence is not broken. As an example:

```
set gslb vserver app_name –persistenceType sourceip -persistMask
255.255.255.0 –timeout 60 –persistenceID 42
```

This source IP persistence synchronizes between sites as long as MEP is up and operating normally. As DNS does not have any other information other than the source of the DNS request, no other method of persistence is available except source IP at this level of processing. The persistenceID keeps track of the persistence that is occurring. As such, if you use the same persistence ID on multiple vServers, they share a persistence table.

The second type of persistence, site cookie persistence, is configured at the GSLB service level, and requires protocol support for persistence (HTTP or HTTPS). When GSLB is active, if a request is received with a matching host header value to the names bound to the GSLB vServer, a site persistence cookie is extracted and compared with the local site. If the cookie indicates that another site should have received the request, then the request is sent to the remote location through either a connection proxy or redirect.

To configure the connection proxy setting, the "–sitePersistence ConnectionProxy" option should be set on all remote GSLB services. To configure redirection, two options must be set, the "–sitePersistence HTTPRedirect" and the "–sitePrefix <prefix>" option. The way the site prefix works is to take the http hostname that was received from the user, say www.domain.com, and append the prefix to that name, so if the prefix is sjc, then the result is to redirect to sjcwww.domain.com. This name should be configured as a static record on the authoritative nameserver for domain.com although, if the NetScaler receives the request, it implicitly processes it as well. One way to simplify the configuration is to make use of a prefix that ends with a ".", for example, if set to "sjc.", it results in "sjc.www.domain.com", like in the previous example. If the subdomain "www" was delegated to the NetScaler, this helps ensure that the NetScaler receives all redirects as well, and maintains all names on the NetScaler.

Examples for connection proxy (building on the earlier example configuration):

```
set gslb service nyc1 –sitePersistence ConnectionProxy
set gslb service sjc1 –sitePersistence ConnectionProxy
set gslb service hk1  –sitePersistence ConnectionProxy
set gslb service lon1 –sitePersistence ConnectionProxy
```

Example for connection redirection:

```
set gslb service nyc1 –sitePersistence HTTPRedirect –sitePrefix nyc1.
set gslb service sjc1 –sitePersistence HTTPRedirect –sitePrefix sjc1.
set gslb service hk1  –sitePersistence HTTPRedirect –siteprefix hk1.
set gslb service lon1 –sitePersistence HTTPRedirect –siteprefix lon1.
```

Now, if you use your host command on nyc1.www.domain.com you get:

```
root@ns# host nyc1.www.domain.com 10.0.1.1
Using domain server:
Name: 10.0.1.1
Address: 10.0.1.1#53
Aliases:

nyc1.www.domain.com has address 10.0.1.10
```

One limitation of the GSLB connection proxy mechanism is that it is limited to having one protocol associated with a given GSLB vServer. This initially seems to indicate that you cannot do GSLB properly for both HTTP and HTTPS on the same name, however it really is not a limitation, but it does impact how connection proxying is done. In the case of HTTP and HTTPS, whichever protocol is desired to be used for the proxy operation should be configured at the GSLB level, and a "dangling" GSLB services should be created but left unbound to the GSLB vServer. For example, assuming you have NYC and SJC as your sites, the overall configuration is:

```
add gslb vserver app_name SSL –PersistenceType sourceIP
add gslb service nyc1-https 10.0.1.10 SSL 443 -siteName nyc –
sitePersistence ConnectionProxy
add gslb service nyc1-http 10.0.1.10 HTTP 80 -siteName nyc –
sitePersistence ConnectionProxy

bind gslb vserver app_name –serviceName nyc1-https
```

The GSLB services detect inbound requests to their IP address and Port without even being bound when local load balancing occurs, then they do a lookup on the host header found in the request, and match it to the proper vServer. When found, the request uses the GSLB service bound to the matching GSLB vServer for the proxy request, so it does not matter if the request was received as HTTP or HTTPS, it is proxied over a secure channel to the remote site. If the link is trusted, the reverse could also be done to proxy it using HTTP instead.

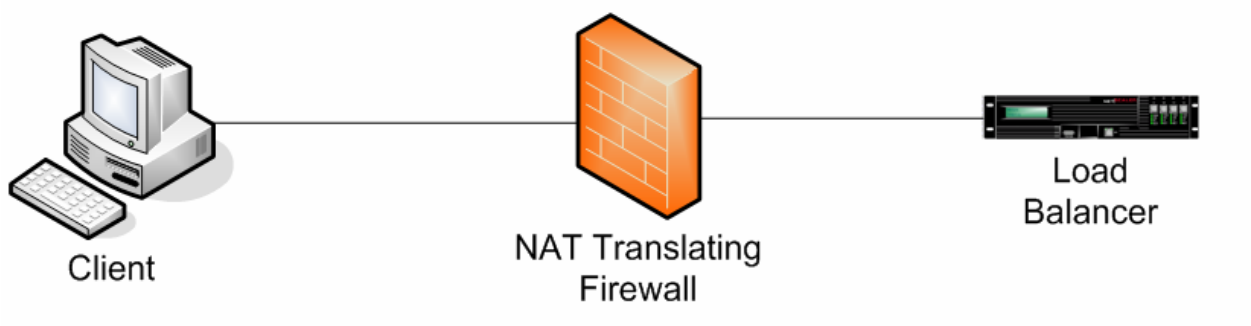
When configuring persistence for HTTPS, there is a further issue of SSL certificates. If redirection is configured, a user connecting to www.domain.com might be sent to nyc.www.domain.com, and an SSL certificate warning occur. To resolve this, there are two generally used options:

1. Wildcard certificates—these certificates have domains in the format *.domain.com for example, so any domain that matches can be directed to without a warning. The advantage of wildcard certificates is that the names used do not need to be known up front; but wildcard certificates are more expensive.
2. Subject Alternative Name certificates—In this case, the certificate would have a primary name of www.domain.com, but list all the potential names that may be redirected to, such as nyc.www.domain.com, sjc.www.domain.com, and so on. The drawback to this approach is that if a new site or name is added, a new certificated must be purchased but these certificates are cheaper than wildcard certificates.

Other side-effects of proxying can also occur based on cookies if they are associated with one domain but not another. As such, thorough testing is always recommended to ensure that the content level persistence is working as expected with an application and under all failure scenarios.

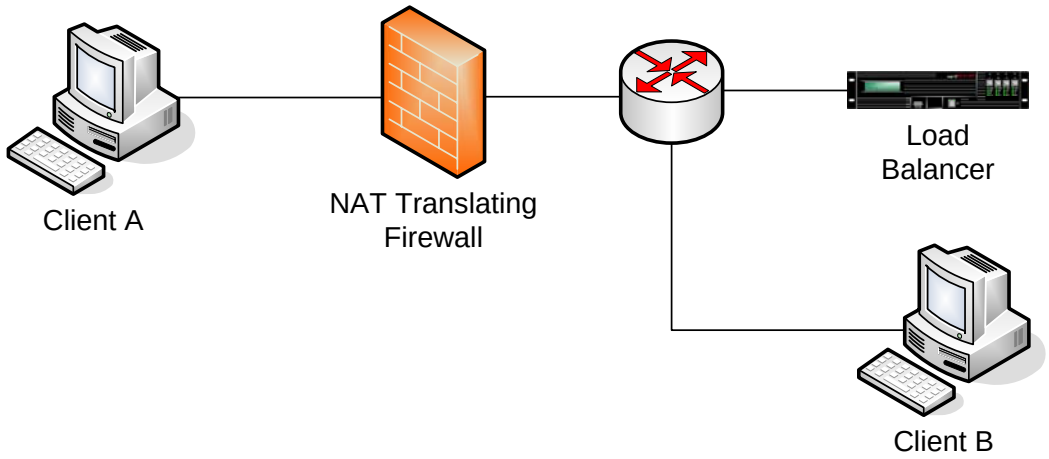
Public versus Private IP Ranges

On each GSLB service, there is a normal IP address and port and a "Public" IP address and port. The public values default to the normal IP address and port, but can be different in the case of a firewall. Consider the following topology:



In this scenario, a client and the NetScaler are separated by a firewall that is translating IP addresses. NetScaler has to understand the IP addresses that are local to itself, and what IP addresses to expose to the clients on the other side of the firewall so it can answer with the public facing IP addresses. This is where the public IP address comes in. Likewise, if there is another remote NetScaler monitoring the service through the firewall, it must know the public port that is exposed, in case that address is translated as well. By setting the public IP address and port values to the appropriate settings, the remote and local NetScalers can behave properly.

When extending the functionality of the public IP addresses and ports, many networks are very complicated in their interactions, and depending on where a client is accessing content from, different IP address ranges might need to be used. Consider the topology with external and internal clients:



Here, we have the same situation that required the public IP address and Port option, but we also have another client that should be given the internal IP address instead. This is where DNS views come in. A DNS view creates a policy that analyzes the DNS request, and if it matches, specifies that a user should receive a different "view" of the network. Then, the views can be bound to the GSLB service so that an alternate IP address is functional. Imagine that the above diagram is for the NYC1 location in the earlier examples. To ensure user requests from the local network (10.0.1.0/24) are given the local private IP address, the following configuration is used:

```
add dns view nyc–local
add dns policy nyc–local "client.IP.SRC.IN_SUBNET(10.0.1.0/24)" –
viewName nyc–local
bind gslb service nyc1 –viewName nyc–local 10.0.1.10
```

You still want to set the publicIP and publicPort configuration properly, because it allows remote monitoring to be done easily, and all other users get the public IP address handed to them, but local users in the proper subnet receive the internal IP address instead. This however, does not change service selection; it just ensures that if the local GSLB is selected, it does not have to bounce off the firewall. More on advanced site selection later.

MEP and Monitoring

Once the basic GSLB configuration is in place, the next step is to ensure that the IP addresses that are provided are healthy. In networks where all the target IP addresses are other NetScaler devices, MEP is the easiest way to exchange states. When properly configured, each node periodically asks the target node what the state of the

resource is on that node. Because the remote node is doing local load balancing, it makes use of the LB vServer state to answer this. Through this process, the local monitoring is leveraged to provide the state to other devices.

In many cases however, this simple state exchange mechanism is not sufficient. There could be a break in communication between two nodes for MEP, while in fact, the remote services are actually up and operational. In other cases, the target IP address and port do not reside on a NetScaler, and explicit monitoring is required. To support the various situations, there are options to modify how to handle the MEP exchanged state. This is done through the "set gslb site" command:

```
> set gslb site nyc
        –metricExchange ( ENABLED | DISABLED )
        –nwMetricExchange ( ENABLED | DISABLED )
        –sessionExchange ( ENABLED | DISABLED )
        –triggerMonitor <triggerMonitor>
```

The first option, –metricExchange enables or disables MEP completely. The second option determines if performance metrics should be exchanged or just the state. In large GSLB configurations, attempting to exchange metrics can add significant overhead that might not be necessary. Likewise, the –sessionExchange dictates whether to synchronize persistence records, such as when USIP persistence is desired. Finally, the –triggerMonitor option determines how monitors bound to the GSLB services behave. This option has the following configuration choices:

1. ALWAYS—Bound monitors should always take priority over MEP exchanged states. While this is the default, if no monitor is bound, then the MEP state is always honored.

2. MEPDOWN—Monitors should only activate if the MEP communication itself fails; honor the MEP state when available, otherwise use the monitor.

3. MEPDOWN_SVCDOWN—Monitors should become active if MEP indicates that the resource is down or if MEP goes down. This basically indicates that MEP should not be trusted completely to remove a resource from consideration, and to use monitors as a failsafe.

At the GSLB service level, monitors can be bound just like with local load balancing, so the details of how this operates are not discussed here.

Disaster Recover

In many GSLB scenarios, the target is to strictly provide a disaster recovery configuration, in other words, no load sharing is performed. The simplest case of this is where a single IP is used unconditionally if the normal resource is not available. This can be configured by using the backupIP option when binding GSLB domains:

```
bind gslb vserver app_name -domainName www.domain.com -TTL 5 -backupIP
1.1.1.1
```

The next model of disaster recovery is when a set of alternate resources are used, so you may have "X" IP addresses that are normally used, but another set of "Y" IP addresses if none of the normal ones are available. In this scenario, a backup vServer is used, just like with local load balancing. For example, if you had vServer "app_name" and "app_name_backup" you would use:

```
set gslb vserver app_name -backupVserver app_name_backup
```

Extending this one step further, if once the primary vServer should not become active again after it fails, then the option "–disablePrimaryOnDown ENABLED" can be set, which forces the backup to continue processing traffic once the primary is done. This is often needed when two sites have unidirectional database synchronization configured, and manual synchronization from the disaster recover site is needed before the primary can become active again.

Load Balancing Methods

Now that the basic configuration is in place, persistence is configured and the state is handled properly, the next part to consider is how to distribute the traffic. Like with local load balancing, there are various load balancing methods that can be configured, which are:

- ROUNDROBIN
- LEASTCONNECTION
- LEASTRESPONSETIME
- SOURCEIPHASH
- LEASTBANDWIDTH
- LEASTPACKETS
- STATICPROXIMITY
- RTT
- CUSTOMLOAD

Most of these behave as they do with local load balancing, and are not discussed, but there are two unique GSLB load balancing methods: RTT (Round Trip Time) and Static Proximity. RTT is a dynamic probing mechanism that calculates the distance between the local NetScaler and the users resolving DNS nameserver. The round trip time is then used to determine the closest site, and as such the closest IP is returned to the user. Static proximity is based on using a pre-configured table of IP address ranges. Both have strengths and weaknesses, and are often used together through the use of the backupLBMethod. Before explaining how both can be used together however, each should be understood individually. One note before explaining these though—as part of the proximity discussion, remember that the IP address that is being tested against is not an actual client IP address in general (unless the NetScaler is in front of the resolving nameserver), instead it is the nameserver that is resolving the IP address on behalf of the client. As such, we refer to this as the "LDNS" IP address for the client's local DNS IP address.

With the static method, tables are created in advance, and then the NetScaler uses them to direct specific IP address ranges to specific GSLB services. As an example of small table, we could use:

```
add location 10.0.1.0 10.0.1.255 "internal.nyc.ns1.*.*.*″
add location 10.0.2.0 10.0.2.255 "internal.sjc.ns1.*.*.*″
add location 10.0.3.0 10.0.3.255 "internal.hk.ns1.*.*.*″
add location 10.0.4.0 10.0.4.255 "internal.lon.ns1.*.*.*″
add location 10.0.10.0 10.0.10.255 "internal.nyc.*.*.*.*″
```

As part of the logic, both the LDNS IP address and each GSLB service are looked up in this table. In it, if a LDNS IP address is 10.0.10.15, then the lookup sets the location to "internal.nyc.*.*.*.*". Assuming the basic configuration used earlier in this guide, the NYC GSLB service of 10.0.1.10 is set to "internal.nyc.ns1.*.*.*". Upon receiving a request from the LDNS, this is found to be a best match, so the NYC location is preferred. This search is performed from left to right, and the longest match determines the "best" match. As an example of how this operates, consider an alternate table:

```
add location 10.0.1.0 10.0.1.255 "NA.nyc.ns1.*.*.*″
add location 10.0.2.0 10.0.2.255 "NA.sjc.ns1.*.*.*″
add location 10.0.3.0 10.0.3.255 "Asia.hk.ns1.*.*.*″
add location 10.0.4.0 10.0.4.255 "EUROPE.lon.ns1.*.*.*″

add location 10.0.10.0 10.0.10.255 "NA.nyc.*.*.*.*″
```

In this case, the best match for IP address 10.0.10.10 would be "NA.nyc.ns1.*.*.*". The reason for the left to right versus right to left matching can be illustrated with the following location table:

```
add location 10.0.1.0 10.0.1.255 "NA.va.bedford.*.*.*″
add location 10.0.2.0 10.0.2.255 "NA.tx.bedford.*.*.*″
add location 10.0.4.0 10.0.4.255 "UK.beds.bedford.*.*.*.*″

add location 10.0.10.0 10.0.10.255 "NA.tx.bedford.*.*.*″
```

If right to left matching was performed, the IP adddress 10.0.10.10 would match on all three locations, when clearly this is not the desired behavior. Instead, the second location entry, for "NA.tx.bedford.*.*.*" is the match, and the proper behavior observed. In the event that no match occurs or several matches are equally “best”, then round robin is used to select between either all GSLB services OR those that match.

Dynamic RTT load balancing makes use of direct measurements from each “site” NetScaler to the remote LDNS IP address to find the site closest to the user’s LDNS. As a result of this, RTT can reduce maintenance on the configurations versus the static table method of proximity, and in theory provide more accurate locations for users. To measure the performance, three monitors are activated on the remote IP address in the following order:

- First a ping of the IP address (LDNS-PING), if that fails; then
- A DNS TCP probe (the syn-ack response time is used as the round trip time) (LDNS-TCP), and if that also fails;
- A DNS UDP query for the domain “.” (LDNS-DNS).

In addition, the monitor behaviors can be changed by adjusting the monitor. As an example, if you wish to change the domain being queried for UDP DNS queries, you can use:

```
set monitor ldns–dns ldns–dns –query "proximitytest.domain.com"
```

This is a commonly changed item, because, in theory, a query for “.” can be used as part of a traffic amplification attack. As such, it is good to change the domain to include the domain name, so that a firewall administrator can more easily understand the purpose of the query. Other values, including the delay between tests and timeout can also be changed.

As a result of the fact that measuring the round trip time would cause a delay, by default, an immediate response is used based on the backup load balancing method, or if not configured (or if RTT is the backup), based on round robin. For internet facing sites, this is generally not an issue because of the frequency that LDNS servers probably query and prime the RTT metric table, but for enterprise environments, it can cause problems if a large number of users end up using the wrong location. To force a delay on the response until data can be gathered, the following nsapimgr command can be used:

```
nsapimgr –ys dropLdnsReqFor2NMEP=1
```

This results in a slight delay on the initial DNS request, but provides time for probes to occur and metrics gathered before providing a response to a user. One final caveat with RTT monitoring—unlike with static tables, proximity is not dictated by the IP address of the GSLB service, but instead by the proximity to the NetScaler representing a site. This can have an impact if a site definition covers resources that are geographically divergent, although this is an unusual situation to encounter.

It is rare that either RTT or static proximity alone is ideal for a given set of clients. The reality is that it is normally a combination of the two that is ideal, such as, directing a known set of IP addresses to pre-configured sites, then leaving dynamic RTT to work cleanly on the rest. To do this, you specify staticproximity as the primary load balancing method, and RTT as the backup, for example:

```
set gslb vserver app_name -lbMethod STATICPROXIMITY backupLBMethod RTT
```

This allows a small table of exceptions to be handled (such as locations known to have a firewall that prevents RTT from functioning properly), then let RTT do the heavy lifting. An alternative way to look at it is to use the reverse—that is, RTT first, then static, because RTT falls back to static when it does not have proximity data, then when it does, it effectively updates the preferred location. This, in effect, makes the static table a set of “hints” to ensure that a reasonably good guess is made initially, but without the cost of a delayed response, although this breaks if source persistence is also configured.