1.0 Terminology

Term	Definition and Usage
domain	Common usage. Technically, a domain is the area covered by an arc from any node in the DNS hierarchical structure (a directed graph). Each domain will have an associated Domain Name and may in turn have 'subdomains' each node of which is also a domain and will have a Domain Name. Frequently used as a shortform for Domain Name as in 'my domain' which more correctly would be 'my Domain Name'.
Domain Name	Common usage. Always shown in bold as two proper nouns (Capitalised). Defines that part of a domain name that is delegated by ICANN, one of its accredited registrars, a delegated country code authority or one of its appointed registrars or agents as defined by the country code authority policy. A **Domain Name** has a registered owner and the owner is both Authoritative and Responsible for DNS information. example.com, example.de, example.ny.us, example.co.uk and example.montreal.qc.ca are all examples of **Domain Names**.
domain name	Always shown as two simple nouns. Defines any part of a domain which fully includes the **Domain Name** e.g. www.example.com is a domain name.
Fully Qualified Domain Name (FQDN)	Common Usage. Unambiguously defines a domain name to the root. A FQDN MUST therefore include the root which in turn means it must have a final DOT on the extreme right of the domain name, for instance, www.example.com. is a fully qualified domain name whereas www.example.com is not (it does not terminate with a DOT). Because of the arbitrary starting point of any domain name this term does not specify or mandate any starting point only that it must end with the root. There is no commonly used term that describes a host-to-root domain name.
host name	Fully defines a host within a domain, for example, fred.example.com is a host name. **Note:** *fred* in this context is only a host name because we know it to be a host or have discovered it to be a host via interrogation of a DNS, that is, it has an A or AAAA RR. At its face value it could equally well be a subdomain name.
qualified domain name	Fairly meaningless term. Defines a part of a domain name to the root. A qualified domain name should ALWAYS include the root which in turn means it must have a final DOT on the extreme right of the domain name, for instance, example.com. is a qualified domain name whereas example.com is not (it does not terminate with a DOT).
Second Level Domain (SLD)	Common usage in conjunction with gTLDs to describe that part of the Domain Name uniquely registered by the Domain owner. Defines that part of the Domain Name below the TLD in the domain hierarchy, for instance, in example.com, **example** is the Second Level Domain. Term much less useful with ccTLDs since the registered domain name may in fact be the Third Level domain Name, for instance, example.co.uk.
sub-domain name	An arbitrary name that is allocated by the owner of the **Domain Name**. A sub-domain name will fully include the **Domain Name**. us.example.com is a valid sub-domain name of example.com. Technically, the name of a subdomain is also a Domain Name.
Top Level Domain (TLD)	Common usage. Defines the TLD part of a **Domain Name** (q.v.) Top Level Domains are split into Generic Top Levels Domains (gTLDs), for example, .com, .int etc. and Country Code Top Level Domains (ccTLDs), for example, .us, .ca, .de etc. and Sponsored Top Level Domains (sTLDs), for example, .aero, .travel etc..
unqualified domain name	Imprecise term. Common usage to describe something that is not a Fully Qualified Domain Name (FQDN), that is, it does not end with a dot. Frequently used to mean what this book calls a 'domain name'.

Zone Name	Any part of a domain that is configured in a DNS server and which fully contains the **Domain Name** for which the owner is authoritative, for instance, example.com, us.example.com (a delegated sub-domain) are Zone Names. A zone is an operational convenience for DNS software and not part of the domain naming hierarchy. **RFC 1034** describes sub-domains zones as subzones rather than re-use the term zone and to the process of creating sub-domains as 'cuts' in the name space. The term subzone appears to have been lost in history.
clause	In an attempt to be consistent we use the term **clause** to describe the top level organization in the named.conf file, for example,the zone clause. A clause groups together related **statements**. What we call a clause is variously called a section, a clause, a statement or an option in other documents. Webster defines a clause to be "a separate section of a discourse or writing; specifically : a distinct article in a formal document" which is good enough for us.
statement	We use the term **statement** to describe an item in a clause of a named.conf file, for example, the **allow-transfer** statement may be used in a view, options or zone clause. What we call a statement is variously called a clause, a statement, an option, a substatement and a phrase in other documents. Webster defines a statement to be "a single declaration or remark" which is good enough for us.

2. DNS Concepts

1. A DNS translates (or maps) the name of a resource to its physical IP address - typically referred to as forward mapping
2. A DNS can also translate the physical IP address to the name of a resource - typically called reverse mapping.

Without DNS every host (PC) which wanted to access a resource on the network (Internet), say a simple web page, for example, www.thing.com, would need to know its **physical IP address**.

With a **Name Server** present in the network any host only needs to know the **physical address of a Name Server** and the **name** of the resource it wishes to access. Using this data it can find the address (or any other stored attribute or property) of the resource by interrogating (**querying**) the Name Server.

If the Primary Name Server does not respond a host can use the Secondary (or tertiary etc.).

As our network grows we start to build up a serious number of Names in our Name Server (database). This gives rise to three new problems.

1. Finding any entry in the database of names becomes increasingly slow as we power through many millions of names looking for the one we want. We need a way to index or organize the names.
2. If every host is accessing our Name Servers the load becomes very high. Maybe we need a way to spread the load across a number of servers.
3. With many Name (resource) records in our database the management problem becomes increasingly difficult as everyone tries to update all the records at the same time. Maybe we need a way to separate (or **delegate**) the administration of these Name (**resource**) records.

The Internet's Domain Name System (DNS) is just a specific implementation of the Name Server concept optimized for the prevailing conditions on the Internet.

From our brief history of Name Servers we saw how three needs emerged:

1. The need for a hierarchy of names
2. The need to spread the operational loads on our name servers
3. The need to delegate the administration of our Name servers

Domains and Delegation

The Domain Name System uses a tree (or hierarchical) name structure. At the top of the tree is the root followed by the Top Level Domains (TLDs) then the domain-name and any number of lower levels each separated with a dot.

NOTE: The root of the tree is represented most of the time as a silent dot ('.') but there are times as we shall see later when it very important.

Top Level Domains (TLDs) were split into two types:

1. Generic Top Level Domains (gTLD), for example, .com, .edu, .net, .org, .mil etc.
2. Country Code Top Level Domain (ccTLD), for example .us, .ca, .tv , .uk etc.

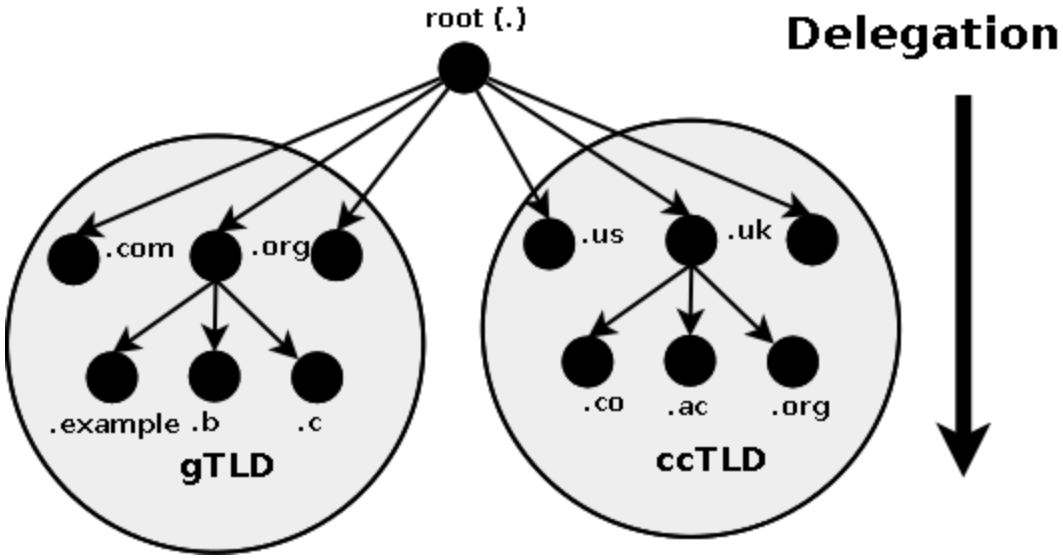


Figure 1-1 Domain Structure and Delegation

What is commonly called a **Domain Name** is actually a combination of a domain-name and a TLD and is written from LEFT to RIGHT with the lowest level in the hierarchy on the left and the highest level on the right.

```
domain-name.tld # example.com
```

In the case of the gTLDs, such as .com, .net etc., the user part of the delegated name - the name the user registered - is a Second Level Domain (SLD). It is the second level in the hierarchy. The user part is therefore frequently simply referred to as the SLD. So the the Domain Name in the example above can be re-defined to consist of:

```
sld.tld # example.com
```

The term Second Level Domain (SLD) is much less useful with ccTLDs where the user registered part is typically the Third Level Domain, for example:

```
example.co.uk
example.com.br
```

The term Second Level Domain (SLD) provides technical precision but can be confusing when applied to a generic concept like a user domain - unless the precision is required we will continue to use the generic term Domain Name or simply Domain to describe the whole name, for instance, what this guide calls a Domain Name would be **example.com** or **example.co.uk**.

The **www** part was chosen by the owner of the domain since they are now the delegated authority for the **example.com** name. They own EVERYTHING to the LEFT of the delegated **Domain Name**.

The leftmost part, **www** in this case, is called a host name.

2.2.3 DNS Organization and Structure

The Internet's DNS exactly maps the 'Domain Name' delegation structure described above. There is a DNS server running at each level in the delegated hierarchy and the responsibility for running the DNS lies with the AUTHORITATIVE control at that level.

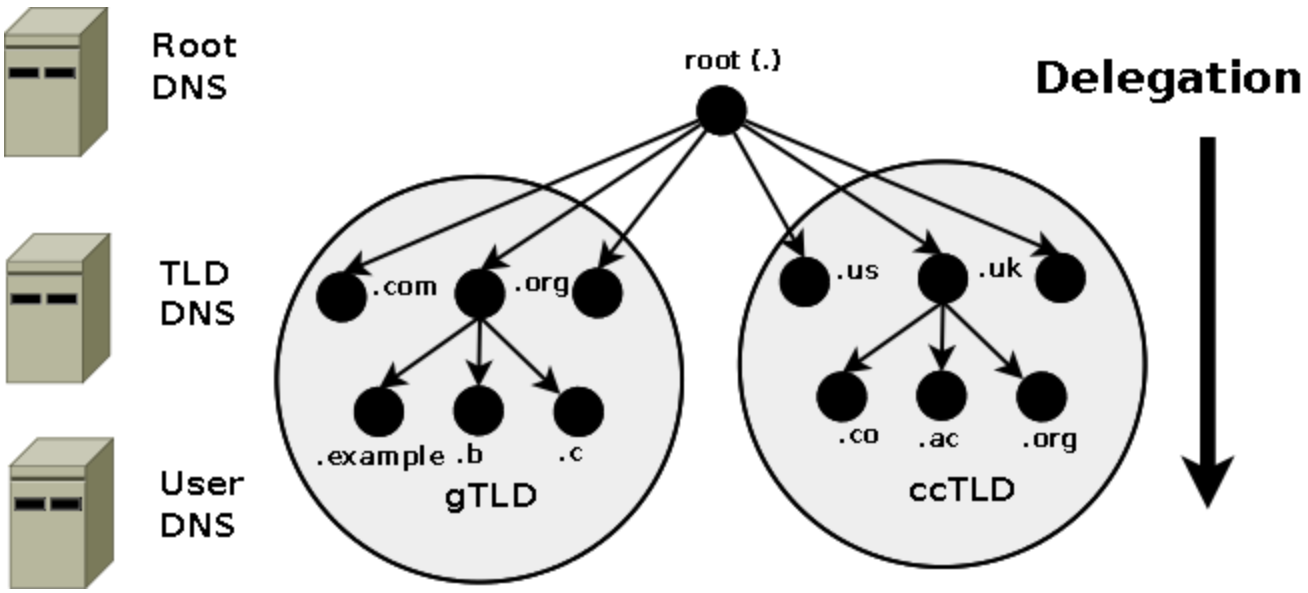


Figure 1-2 DNS mapped to Domain Delegation

The Root-Servers are known to every public DNS server in the world and are the starting point for every name lookup operation (or query). To create additional resilience each root-server typically has multiple **instances** (copies) spread throughout the world. Each instance has the same IP address but data is sent to the closest instance using a process called **anycasting**.

The Authority and therefore the responsibility for the User (or **Domain Name**) DNS servers lies with the owner of the domain. In many cases this responsibility is delegated by the owner of the Domain to an ISP, Web Hosting company or increasingly a registrar. Many companies, however, elect to run their own DNS servers and even delegate the Authority and responsibility for sub-domain DNS servers to separate parts of their organization.

When any DNS cannot answer (resolve) a request (a **query**) for a domain name from a client, for instance, example.com, the query is passed to a **root-server** which will direct (**refer**) the query to the appropriate TLD DNS server (for .com) which will in turn direct (**refer**) it to the appropriate Domain (User) DNS server.

2.2.4 DNS System Components

A Domain Name System (DNS) includes three parts:

1. Data which describes the domain(s)
2. One or more Name Server programs.
3. A resolver program or library.

A single DNS server may support many domains. The data for each domain describes global properties of the domain and its hosts (or services). This data is defined in the form of textual Resource Records organized in Zone Files.

The Name Server program typically does three things:

1. It will read a configuration file which defines the zones for which it is responsible.
2. Depending on the Name Servers functionality a configuration file may describe various behaviours, for instance, to cache or not. Some DNS servers are very specialized and do not provide this level of control.
3. Respond to questions (**queries**) from local or remote hosts.

The resolver program or library is located on each host and provides a means of translating a users request for, say, www.thing.com into one or more queries to DNS servers using UDP (or TCP) protocols.

2.2.5 Zones and Zone Files

Zone files contain Resource Records that describe a domain or sub-domain. A zone file will consist of the following types of data:

1. Data that indicates the top of the zone and some of its general properties (a SOA Record).
2. Authoritative data for all nodes or hosts within the zone (typically A (IPv4) or AAAA (IPv6) Records).

3. Data that describes global information for the zone (including mail MX Records and Name Server NS Records).
4. In the case of sub-domain delegation the name servers responsible for this sub-domain (one or more NS Records).
5. In the case of sub-domain delegation one or more **glue** records that allows a name server to reach the sub-domain (typically one or more A or AAAA Records) for the sub-domain name servers.

2.2.6 DNS Queries

The major task carried out by a DNS server is to respond to queries (questions) from a local or remote resolver or other DNS acting on behalf of a resolver. A query would be something like 'what is the IP address of fred.example.com'.

A DNS server may receive such a query for any domain. DNS servers may be configured to be authoritative for some domains, slaves for others, forward queries or other combinations.

Most of the queries that a DNS server will receive will be for domains for which it has no knowledge, that is, for which it has no local zone files. DNS software typically allows the name server to respond in different ways to queries about which it has no knowledge.

There are three types of queries defined for DNS:

1. A recursive query - the complete answer to the question is always returned. DNS servers are not required to support recursive queries.
2. An Iterative (or non-recursive) query - where the complete answer MAY be returned or a **referral** provided to another DNS. All DNS servers must support Iterative queries.
3. An Inverse query - where the user wants to know the domain name given a resource record.

Note: The process called Reverse Mapping (returns a host name given an IP address) does not use Inverse queries but instead uses Recursive and Iterative (non-recursive) queries using the special domain name IN-ADDR.ARPA.

2.2.6.1 Recursive Queries

A recursive query is one where the DNS server will fully answer the query (or give an error). DNS servers are not required to support recursive queries and both the resolver (or another DNS acting recursively on behalf of another resolver) negotiate use of recursive service using a bit (RD) in the query header.

There are three possible responses to a recursive query:

1. The answer to the query accompanied by any CNAME records (aliases) that may be useful. The response will indicate whether the data is authoritative or cached.
2. An error indicating the domain or host does not exist (NXDOMAIN). This response may also contain CNAME records that pointed to the non-existing host.
3. An temporary error indication - for instance, can't access other DNS's due to network error etc..

In a recursive query a DNS Resolver will, on behalf of the client (stub-resolver), chase the trail of DNS system across the universe to get the real answer to the question. The journey of a simple query such as 'what is the IP address of www.example.com' to a DNS Resolver which supports recursive queries but is not authoritative for example.com is shown in Diagram 1-3 below:

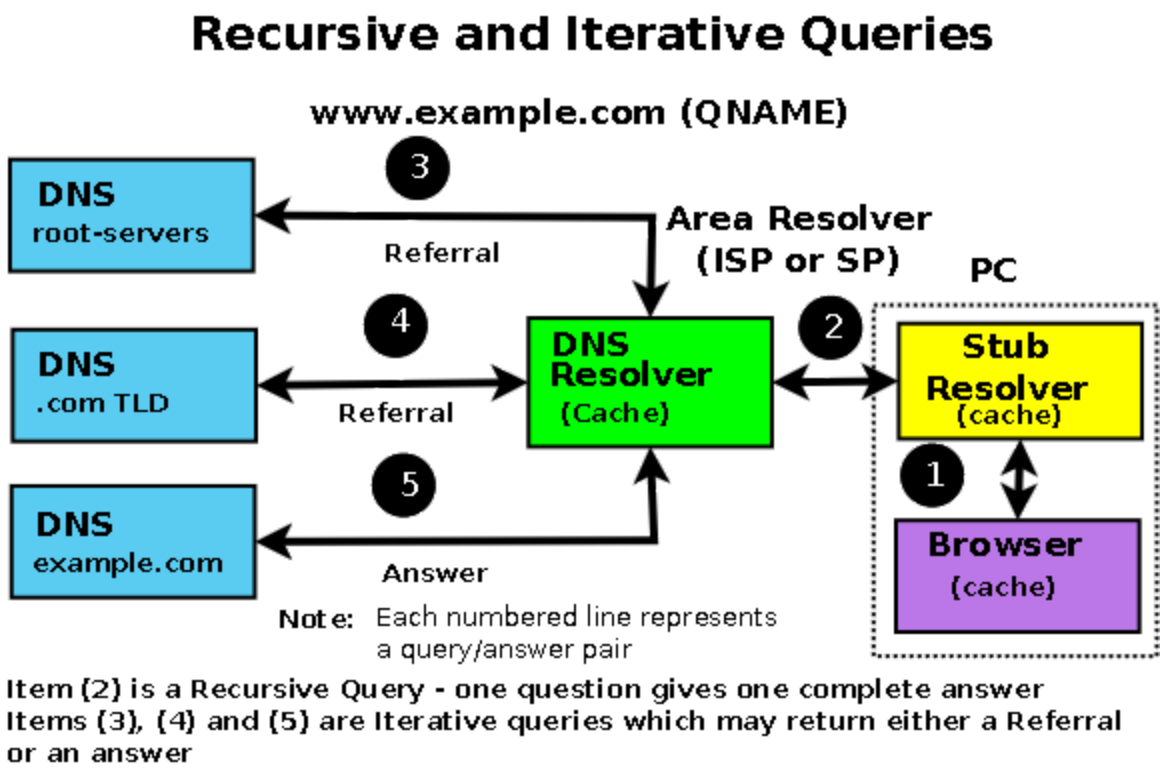


Diagram 1-3 Recursive Query Processing

1. The user types www.example.com into their browser address bar. The browser issues a standard function library call (1) to the local stub-resolver.
2. The stub-resolver sends a query (2) 'what is the IP address of www.example.com' to locally configured DNS resolver (aka recursive name server). This is a standard DNS query requesting recursive services (RD (Recursion Desired) = 1).
3. The DNS Resolver looks up the address of www.example.com in its local tables (its **cache**) and does not find it. (If it were found it would be returned immediately to the Stub-resolver in an answer message and the transaction would be complete.)
4. The DNS resolver sends a query (3) to a root-server (every DNS resolver is configured with a file that tells it the names and IP addresses of the root servers) for the IP of www.example.com. (Root-servers, TLD servers and correctly configured user name servers do not, a matter of policy, support recursive queries so the Resolver will, typically, not set Recursion Desired (RD = 0) - this query is, in fact, an Iterative query.)
5. The root-server knows nothing about example.com, let alone the www part, but it does know about the next level in the hierarchy, in this case, the .com part so it replies (answers) with a referral (3) pointing at the TLD servers for .com.
6. The DNS Resolver sends a new query (4) 'what is the IP address of www.example.com' to one of the .com TLD servers. Again it will use, typically, an Iterative query.
7. The TLD server knows about example.com, but knows nothing about www so, since it cannot supply a complete response to the query, it replies (4) with a referral to the name servers for example.com.

8. The DNS Resolver sends yet another query (5) 'what is the IP address www.example.com' to one of the name servers for example.com. Once again it will use, typically, an Iterative query.
9. The example.com zone file defines a A (IPv4 address) record so the authoritative server for example.com returns (5) the A record for www.example.com (it fully answers the question).
10. The DNS Resolver sends the response (answer) www.example.com=x.x.x.x to the client's stub-resolver (2) and then places this information in its cache.
11. The stub-resolver places the information www.example.com=x.x.x.x in its cache and responds to the original standard library function call (1) with www.example.com = x.x.x.x.
12. The browser receives the response to its standard function call, places the information in its cache (really) and initiates an HTTP session to the address x.x.x.x. DNS transaction complete. Quite simple really, not much could possibly go wrong.

In summary, the stub-resolver demands recursive services from the DNS Resolver. The DNS Resolver provides a recursive service but uses, typically, Iterative queries to achieve it.

2.2.6.2 Iterative (non-recursive) Queries

A Iterative (or non-recursive) query is one where the DNS server may provide an answer or a partial answer (a referral) to the query (or give an error). All DNS servers must support non-recursive (Iterative) queries. An Iterative query is technically simply a normal DNS query that does not request Recursive Services.

There are four possible responses to a non-recursive query:

1. The answer to the query accompanied by any CNAME records (aliases) that may be useful (in a Iterative Query this will ONLY occur if the requested data is already available in the cache). The response will indicate whether the data is authoritative or cached.
2. An error indicating the domain or host does not exist (NXDOMAIN). This response may also contain CNAME records that pointed to the non-existing host.
3. An temporary error indication, for instance, can't access other DNS's due to network error etc..
4. A **referral**: If the requested data is not available in the cache then the name and IP addess(es) of one or more name server(s) that are closer to the requested domain name (in all cases this is the next lower level in the DNS hierarchy) will be returned. This referral may, or may not be, to the authoritative name server for the target domain.

In Diagram 1-3 above the transactions (3), (4) and (5) are normally all Iterative queries. Even if the DNS server requested Recursion (RD=1) it would be denied and a normal referral (or answer) returned. Why use Iterative queries? They are much faster, the DNS server receiving the query either already has the answer in its cache, in which case it sends it, or not, in which case it sends a referral. No messing around. Iterative queries give the requestor greater control. A referral typically contains a list of name servers for the next level in the DNS hierarchy. The requestor may have additional information about one or more of these name servers in its cache (including which is the fastest) from which it can make a better decision about which name server to use. Iterative queries are also extremely useful in diagnostic situations.

2.2.6.3 Inverse Queries

Historically, an Inverse query mapped a resource record to a domain. An example Inverse query would be 'what is the domain name for this MX record'. Inverse query support was optional and it was permitted for the DNS server to return a response **Not Implemented**.

Inverse queries are NOT used to find a host name given an IP address. This process is called Reverse Mapping (Look-up) uses recursive and Iterative (non-recursive) queries with the special domain name IN-ADDR.ARPA.

2.2.7 Zone Updates

Warning: While zone transfers are generally essential for the operation of DNS systems they are also a source of threat. A A slave Name Server can become **poisoned** if it accepts zone updates from a malicious source. Care should be taken during configuration to ensure that, as a minimum, the 'slave' will only accept transfers from known sources. The example configurations provide these minimum precautions. Security Overview outlines some of the potential threats involved.

2.2.7.1 Full Zone Update (AXFR)

The original DNS specifications envisaged that Slave (or secondary) Name Servers would 'poll' the Domain (or zone) Master. The time between such 'polling' is determined by the **refresh** value on the domain's SOA Resource Record

The polling process is accomplished by the Slave sending a query to the Master requesting its current SOA resource record (RR). If the **serial number** of this RR is higher than the current one maintained by the Slave, a zone transfer (AXFR) is requested. This is why it is vital to be very disciplined about updating the SOA serial number every time anything changes in ANY of the zone records.

2.2.7.2 Incremental Zone Update (IXFR)

Transferring very large zone files can take a long time and waste bandwidth and other resources. This is especially wasteful if only a single RR has been changed! RFC 1995 introduced Incremental Zone Transfers (IXFR) which as the name suggests allows the Slave and Master to transfer only those records that have changed.

The process works as for AXFR. The Slave sends a query for the domain's SOA RR every **refresh** interval. If the **serial number** of the SOA record is higher than the current one maintained by the Slave it requests a zone transfer and indicates whether or not it is capable of accepting an Incremental Transfer (IXFR). If both Master and slave support the feature an Incremental Transfer (IXFR) takes place otherwise a Full Transfer (AXFR) takes place.

2.2.7.3 Notify (NOTIFY)

RFC 1912 recommends a REFRESH interval of up to 12 hours on the REFRESH interval of an SOA Resource Record. This means that, in the worst case, changes to the Master Name Server may not be visible at the Slave Name Server(s) for up to 12 hours. In a dynamic environment this may be unacceptable.

RFC 1996 introduced a scheme whereby the **Master** will send a NOTIFY message to the **Slave** Name Server(s) that a change MAY have occurred in the domain records. The Slave(s) on receipt of the NOTIFY will request the latest SOA Resource Record and if the **serial number** of the SOA RR is greater than its current value it will initiate a zone transfer using either a Full Zone Transfer (AXFR) or an Incremental Zone Transfer (IXFR).

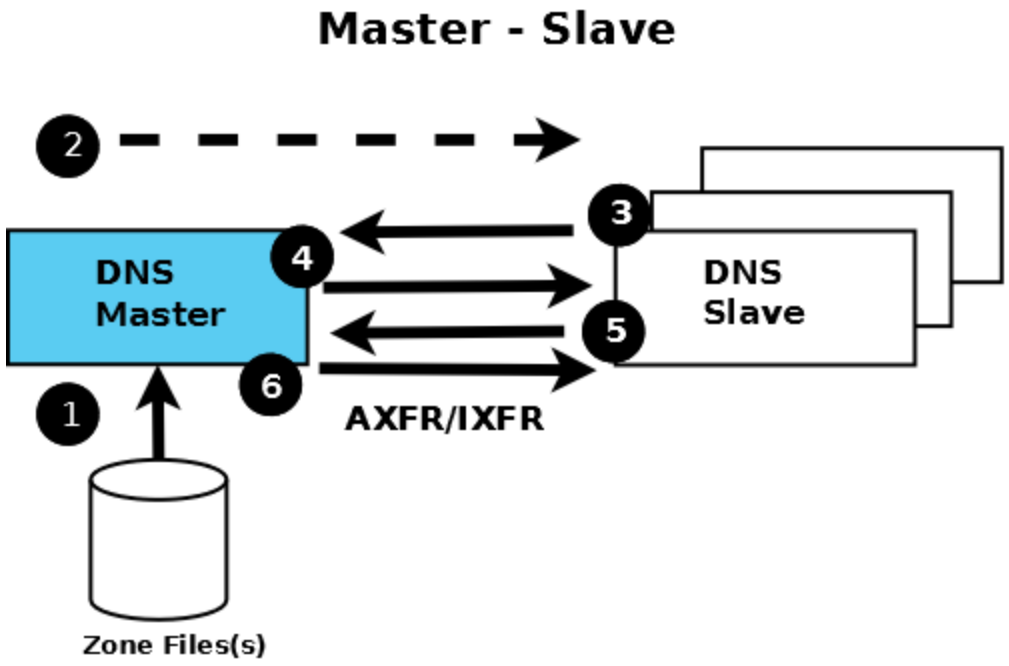


Diagram 1-4 Master - Slave Interaction and Zone Transfer

Zone Transfer Process Description

The time taken to propagate zone changes throughout the Internet is determined by two major factors. First, the time taken to update all the Domain's Name servers when any zone change occurs. This, in turn, is determined by the method used to initiate zone transfers to all Slave Name Servers which may be passive (the Slave will periodically poll the Master) or Active (the Master will send a NOTIFY to its configured Slave(s)). Both methods are described below. Second, the current TTL value (prior to its change) on any changed zone record will determine when Resolvers will refresh their caches by interrogating the Authoritative Name Server.

1. If the Master has been configured to support NOTIFY messages then whenever the status of the Master's zone file (1) changes it will send a NOTIFY message (2) to each configured Slave. A NOTIFY message does not necessarily indicate that the zone file has changed, for example, if the Master or the zone is reloaded then a NOTIFY message is triggered even if no changes have occured. When the Slave receives a NOTIFY message it follows the procedure defined in Step 3 below.
2. Irrespective of whether the Master has been configured to support NOTIFY messages or not the Slave will always use the passive or 'polling' process described in this step. (While on its face this seem superflous in cases where the Master has been configured to use NOTIFY, however, it does provide protection again lost NOTIFY messages due to mal-configuration or malicious attack.) When a Slave server is loaded it will read any current saved zone file (see file statement) or immediately intitate a zone transfer if there is no saved zone file. It then starts a timer using the **refresh** value in the zone's SOA RR. When this timer expires the Slave follows the procedure defined in Step 3 below.

3. If the Slave's **refresh** timer expires OR it receives a NOTIFY message the Slave will immediately issue a query for the zone Master's SOA RR (3).
4. When the answer arrives (4) the Slave compares the **serial number** of its current SOA RR with that of the answer (the Master's SOA RR). If the value of the Master SOA RR **serial number** is greater than the current **serial number** in the Slave's SOA copy then a zone transfer (5) is intiated by the Slave. (The gruesome details of the serial number arithmetic is defined in RFC 1982 and clarified in RFC 2181, the date based convention used for serial numbers is defined here). If the slave fails to read the Master's SOA RR (or fails to intiate the zone transfer) then it will try again after the **retry** time defined in the zone's SOA RR but will continue answering Authoritatively for the Domain (or zone). The retry procedure will be repeated (every **retry** interval) either until it succeeds (in which case the process continues at step 5 below) or until the **expiry** timer of the zone's SOA RR is reached, at which point the Slave will stop answering queries for the Domain.
5. The Slave always intitiates (5) a zone transfer operation (using AXFR or IXFR) using TCP on Port 53 (this can be configured using the transfer-source statement).
6. The Master will transfer the requested zone file (6) to the slave. On completion the Slave will reset its **refresh** and **expiry** timers.

2.2.7.4 Dynamic Update

The classic method of updating Zone Resource Records is to manually edit the zone file and then stop and start the name server to propagate the changes. When the volume of changes reaches a certain level this can become operationally unacceptable - especially considering that in organisations which handle large numbers of Zone Files, such as service providers, BIND itself can take a long time to restart at it plows through very large numbers of zone statements.

The 'holy grail' of DNS is to provide a method of dynamically changing the DNS records while DNS continues to service requests.

There are two architectural approaches to solving this problem:

1. Allow 'run-time' updating of the Zone Records from an external source/application.
2. Directly feed the DNS from a database which can be dynamically updated. BIND provides two APIs to enable this, other DNS implemenations provide embedded database access.

RFC 2136 takes the first approach and defines a process where zone records can be updated from an external source. The key limitation in this specification is that a new domain cannot be added dynamically. All other records within an existing zone can be added, changed or deleted. This limitation is also true for both of BIND's APIs as well.

As part of RFC 2136 the term **Primary Master** was coined to describe the Name Server defined in the SOA Resource Record for the zone. The significance of this term is that when dynamically updating records it is essential to update only one server even though there may be multiple **master** servers for the zone. In order to solve this problem a 'boss' server must be selected, this 'boss' server (the **Primary Master**) has no special characteristics other than it is defined as the Name Server in the SOA record and may appear in an **allow-update** clause to control the update process.

DDNS is normally associated with Secure DNS. Dynamic. By enabling Dynamic DNS you are also opening up the possibility of **master** zone file corruption or poisoning.

2.2.7.5 Alternative Dynamic DNS Approaches

As noted above the major limitation in the standard Dynamic DNS (RFC 2136) approach is that new domains cannot be created dynamically.

BIND-DLZ takes a much more radical approach and, using a serious patch to BIND, allows replacement of all zone files with a single zone file which defines a database entry. The database support, which includes most of the major databases (MySQL, PostgreSQL, BDB and LDAP among others) allows the addition of new domains as well as changes to pre-existing domains without the need to stop and start BIND.

PowerDNS an authoritative only name server takes a similar approach with its own (non-BIND) code base by referring all queries to the database back-end and thereby allow new domains to be added dynamically.

2.3 Security Overview

The critical point is to first understand what you want to secure - or rather what threat level you want to secure against. This will be very different if you run a root server rather than running a modest in-house DNS serving a couple of low volume web sites.

There are at least three types of DNS security, two of which are - relatively - painless and DNSSEC which is - relatively - painful.

2.3.1 Security Threats

In order to be able to assess both the potential threats and the possible counter-measures it is first and foremost necessary to understand the normal data flows in a DNS system. Diagram 1-5 below shows this flow.

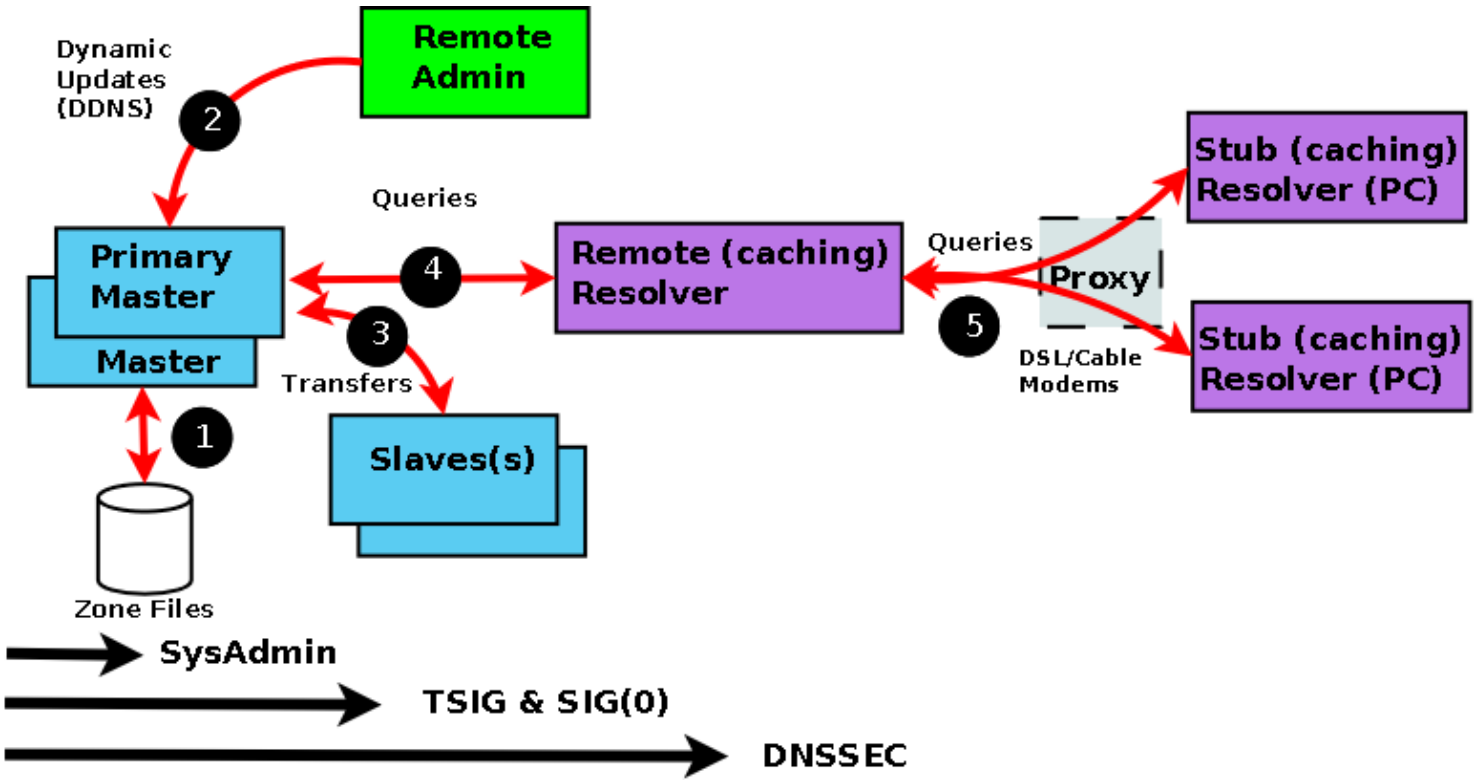


Diagram 1-5 DNS Data Flow

Every data flow (each RED line above) is a potential source of threat! Using the numbers from the above diagram here is what can happen at each flow:

Number	Area	Threat
(1)	Zone Files	File Corruption (malicious or accidental). Local threat.
(2)	Dynamic Updates	Unauthorized Updates, IP address spoofing (impersonating update source). Server to Server (TSIG Transaction) threat.
(3)	Zone Transfers	IP address spoofing (impersonating update source). Server to Server (TSIG Transaction) threat.
(4)	Remote Queries	Cache Poisoning by IP spoofing, data interception, or a subverted Master or Slave. Server to Client (DNSSEC) threat.
(5)	Resolver Queries	Data interception, Poisoned Cache, subverted Master or Slave, local IP spoofing. Remote Client-client (DNSSEC) threat.

The first phase of getting a handle on the problem is to figure (audit) what threats are applicable and how seriously do YOU rate them or do they even apply.

2.3.2 Security Types

The numbering used below relates to diagram 1-3.

(1) The primary source of Zone data is normally the Zone Files. This data should be secure and securely backed up. This threat is classified as **Local** and is typically handled by good system administration.

(2) The BIND default is to **deny** Dynamic Zone Updates. If you have enabled this service or require to it poses a serious threat to the integrity of your Zone files and should be protected. This is classified as a **Server-Server (Transaction)** threat.

(3) If you run slave servers you will do zone transfers. **Note:** You do NOT have to run with slave servers, you can run with multiple masters and eliminate the transfer threat entirely. This is classified as a **Server-Server (Transaction)** threat.

(4) The possibility of Remote Cache Poisoning due to IP spoofing, data interception and other hacks is a judgement call if you are running a simple web site. If the site is high profile, open to competitive threat or is a high revenue earner you have probably implemented solutions already. This is classified as a **Server-Client** threat.

(5) The current DNSSEC standards define a security aware resolver and this concept is under active development by an number of groups round the world. This is classified as a **Server-Client** threat.

2.3.3 Security - Local

Normal system administration practices such as ensuring that files (configuration and zone files) are securely backed-up, proper read and write permissions applied and sensible physical access control to servers may be sufficient.

Implementing a Stealth (or Split) DNS server provides a more serious solution depending on available resources.

2.3.4 Server-Server (TSIG Transactions)

Zone transfers. If you have slave servers you will do zone transfers. BIND provides **Access Control Lists (ACLs)** which allow simple IP address protection. While IP based **ACLs** are relatively easy to subvert using IP addreess spoofing they are a **lot** better than nothing and require very little work. You can run with multiple masters (no slaves) and eliminate the threat entirely. You will have to manually synchronise zone file updates but this may be a simpler solution if changes are not frequent.

Dynamic Updates. If you must run with this service it should be secured. BIND provides **Access Control Lists (ACLs)** which allow simple IP address protection but this is probably not adequate unless you can secure the IP addresses, that is, all systems are behind a firewall/DMZ/NAT or the updating hosts are using private IP addresses.

TSIG/TKEY If all other solutions fail DNS specifications (RFC 2845 - TSIG and RFC 2930 - TKEY) provide authentication protocol enhancements to secure these Server-Server transactions.

TSIG and **TKEY** implementations are messy but not too complicated simply because of the scope of the problem. With **Server-Server** transactions there is a finite and normally small number of hosts involved. The protocols depend on a **shared secret** between the master and the slave(s) or updater(s). It is further assumed that you can get the **shared secret** securely to the peer server by some means not covered in the protocol itself. This process, known as **key exchange**, may not be trivial (typically long random strings of base64 characters are involved) but you can use the telephone(!), mail, fax or PGP email among other methods.

The **shared-secret** is open to **brute-force** attacks so frequent (monthly or more) changing of **shared secrets** will become a fact of life. **TKEY** allows automation of **key-exchange** using a Diffie-Hellman algorithm but starts with a **shared secret**! TKEY appears to have very limited, if any, usage.

2.3.5 Server-Client (DNSSEC)

The classic Remote Poisoned cache problem is not trivial to solve simply because there may be an infinitely large number of Remote Caches involved. It is not reasonable to assume that you can use a **shared secret**.

Instead **DNSSEC** relies on **public/private key authentication**. The DNSSEC specifications attempt to answer three questions:

1. Authentication - the DNS responding really is the DNS that the request was sent to.
2. Integrity - the response is complete and nothing is missing or changed.
3. Proof of non-existance - if the DNS returns a status that the name does not exist (NXDOMAIN) this response can be proven to have come from the authoritative server.

3. DNS Reverse Mapping

3.1 Reverse Mapping Overview

A normal DNS query would be of the form 'what is the IP of host=www in domain=mydomain.com'. There are times however when we want to be able to find out the name of the host whose IP address = x.x.x.x (or x:x:x:x:x for IPv6). Sometimes this is required for diagnostic purposes, more frequently these days it is used for security purposes to trace a hacker or spammer. Indeed, most modern mailing systems use reverse mapping to provide simple, first-cut, authentication using a dual look-up process - IP to name and name to IP. IPv4 reverse mapping is not mandatory though, as indicated by the mail example, it is essential for hosts that send mail, using either a Mail Transfer Agent (MTA) or a Mail User Agent (MUA). In the case of IPv6 reverse-mapping was originally mandatory but as part of the seemingly relentless move to relax the goals of the original specifications (doubtless for good operational reasons) it is no longer a mandatory requirement. It seems to have fallen into the Jolly Useful ™ category.

In order to perform Reverse Mapping using normal recursive and Iterative (non-recursive) queries the DNS designers defined the special (reserved) **Domain Name** of IN-ADDR.ARPA for IPv4 addresses and IP6.ARPA for IPv6 addresses.

3.1.1 IP Address Allocation and Delegation

IP Addresses are allocated in blocks from IANA (currently managed by ICAAN) through the five Regional Internet Registries (RIRs - ARIN, RIPE, AFRNIC, APNIC, LACNIC), which it turn may allocate them to National Internet Registries (NIRs - though this national level is not always present) and so down to what are called Local Internet Registries (LIRs) which are typically ISPs/SPs. These LIRs are typically responsible for allocating blocks of addresses to customers. The allocation of IP address is essentially a delegation of the authority to reverse map these addresses. Thus, as an RIR is allocated a block if IP addresses it picks up the authoritative responsibility to delegate the revesre map as well (and operate the reverse map DNS infrastructure at its level). Similarly the LIR (and/or NIR) continues the delegation until it reaches the end customer. Figure 3.0 shows this relationship.

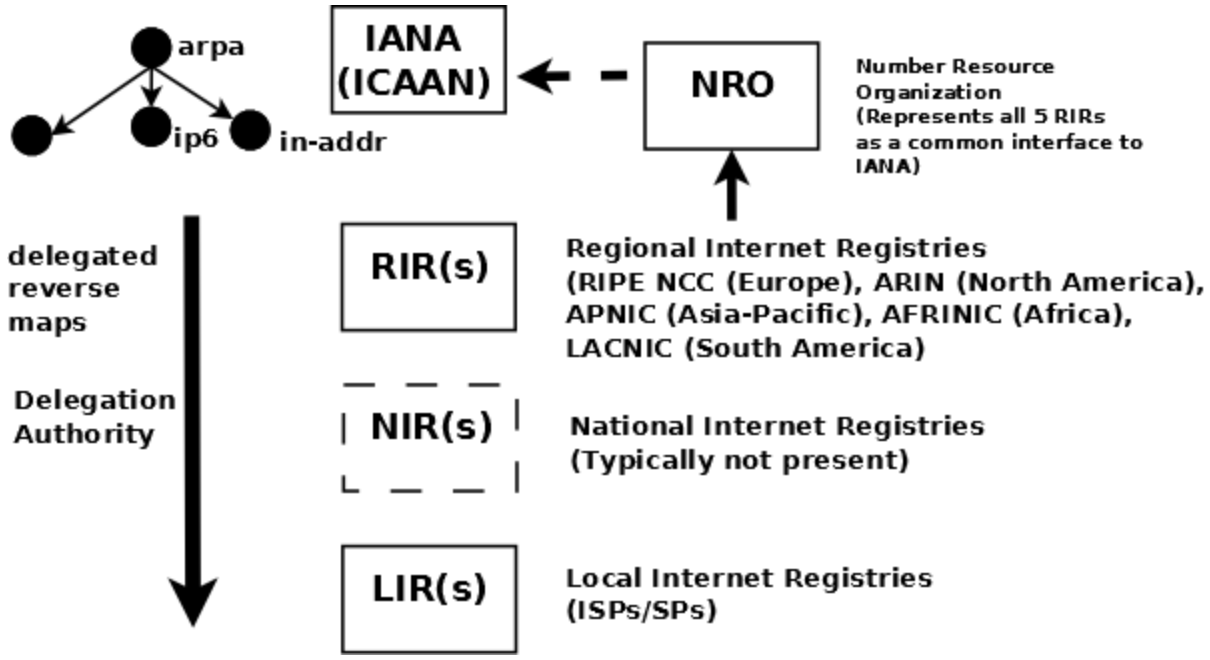


Figure 3.0 IP Address Allocation and Delegation

3.2 IPv4 IN-ADDR.ARPA Reverse Mapping Domain

Reverse Mapping looks horribly complicated. It is not. As with all things when we understand what is being done and why - all becomes as clear as mud.

We defined the normal domain name structure as a tree starting from the root. We write a normal domain name LEFT to RIGHT but the hierarchical structure is RIGHT to LEFT.

```
domain name = www.example.com
highest node in tree is = .com (technically the normally silent . for root
is the highest but...)
next (lower) = .example
next (lower) = www
```

An IPv4 address is written as:

```
192.168.23.17
```

This IPv4 address defines a host (17) which happens to be in a Class C address range (192.168.23.x). In this case the most important part (the highest node in the address hierarchy) is on the LEFT (192) not the RIGHT. This is a tad awkward and would make it impossible to construct a sensible tree structure that could be searched in a single lifetime.

The solution is to reverse the order of the address and place the result under the special domain IN-ADDR.ARPA (you will see this also written as in-addr.arpa which is perfectly legitimate since domain names are case insensitive but the case should be preserved between query and response. You may elect to use whatever you wish including IN-addr.Arpa if that is your preference).

The last part of the IPv4 Address (17) is the host address and hosts, from our previous reading, are typically defined inside a zone file so we will ignore it and only use the Class C address base. The result of our manipulations are:

```
IP address =192.168.23.17
Class C base = 192.168.23 ; omits the host address = 17
Reversed Class C base = 23.168.192
Added to IN-ADDR.ARPA domain = 23.168.192.IN-ADDR.ARPA
```

This is shown in figure 3.1 below.

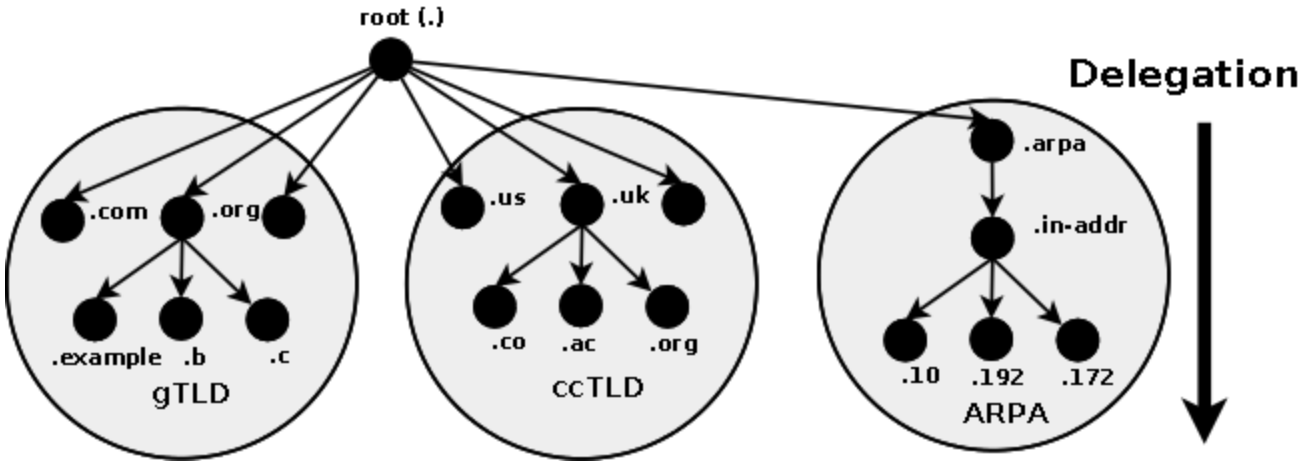


Figure 3.1 IN-ADDR.ARPA Reverse Mapping

Finally, we construct a zone file to describe all the hosts (nodes) in the Reverse Mapped zone using PTR Records. The resulting file will look something like this:

```
$TTL 2d  ; 172800 seconds
$ORIGIN 23.168.192.IN-ADDR.ARPA.
@             IN      SOA   ns1.example.com. hostmaster.example.com. (
                              2003080800 ; serial number
                              3h         ; refresh
                              15m        ; update retry
                              3w         ; expiry
                              3h         ; nx = nxdomain ttl
                              )
              IN      NS      ns1.example.com.
              IN      NS      ns2.example.com.
1             IN      PTR     www.example.com. ; qualified name
2             IN      PTR     joe.example.com.
.....
17            IN      PTR     bill.example.com.
.....
74            IN      PTR     fred.example.com.
....
```

Notes:

We must use qualified names ending with a dot (in fact they are Fully Qualified Domain Names - FQDNs) with the PTR target (left-hand) name in reverse mapped zone files because if we did not our $ORIGIN directive would lead to some strange results. For example, if we wrote an unqualified name such as:

```
74            IN      PTR fred
```

1. Using the $ORIGIN substitution rule the above would expand to fred.23.168.192.IN-ADDR.ARPA. which is probably not what we intended.
2. If a reverse-map file is not included in our DNS configuration then, as normal, the query will pass to the DNS hierarchy. In the case where local IP addreses are globally routable (increasingly rare) an ISP or service provider may be responsible for maintaining the reverse map in which case such reverse-map queries must access the DNS hierarchy. Most frequently, private IP addresses are used, such as 192.168.x.x, 10.x.x.x or some IP ranges in 172.x.x.x, in local networks (and exclusively with home networks) with a NAT configuration translating to a global address before being routed to the public network. In such configurations it is imperative that a reverse-map covering the range of private IP addresses be included in the local DNS configuration. Failure to do so may cause local applications to fail or give strange results (none of them good) because the reverse-map request will be responded to by the DNS hierarchy where such private addresses are meaningless. In order to limit the negative effect of such mis-configuration BIND introduced the empty-zones-enable statement which defaults to a state that will minimise damage to the DNS hierarchy. However, BIND's default option does not solve the problem for local applications which may depend upon correct local results from a reverse-map query. It is, and remains, a serious mis-configuration of DNS to not include a reverse-map for all local (RFC 1918) addresses.

3. There are no A RRs for the defined NS names (respectively ns1.example.com and ns2.example.com) since both are out-of-zone names. Any lookup is done via the forward zone file for example.com in which suitable A RRs for these names must exist.

3.3 Reverse Map Delegation

Classless Reverse Map Delegation is defined by RFC 2317 which has Best Current Practice status and should be regarded as a definitive reference. Classless routing allows allocation of sub-nets on non-octet boundaries, that is, less that 256 addresses from a Class C address may be allocated and routed. The technique defined in the RFC is attributed to Glen A. Herrmannsfeldt.

Normal domain name mapping as we have seen maps the domain name to an IP address. This process is independent of the ISP or other authority that allocated the IP name space. If the addresses were to change then the owner of the domain that maps these addresses would be able to make the necessary changes directly with either the relevant registrar i.e. change the IP address of DNS's for the domain or change the zone file(s) that describe the domain.

The rule is that entities can be delegated only once in the domain name tree this includes IN-ADDR.ARPA. When a Class C subnet is assigned by an ISP or other authority, for example, 192.168.23.64/27 (a 32 IP address subnet) the responsibility for reverse mapping for the whole Class C address has already been assigned to the ISP or Authority. If you want to change the host names in the assigned subnet they must be notified to the authority for that Class C address. Generally, this is unacceptable since such requests may encounter indifference, cost or questions. It is most desirable that responsibility for reverse mapping be delegated when the IP address subnet is assigned though this does require support and co-operation with the currently delegated reverse map authority (ISP or other organization).

The technique defined in RFC 2317 provides for such delegation to take place using CNAME Resource Records (rather than the more normal PTR Resource Records) in an expanded IN-ADDR.ARPA name space.

The IPv4 Reverse Map Generator can be used to generate all the necessary RRs or even just let you experiment with various delegation scenarios.

Note: RFC 2317 uses the CNAME RR to implement the solution as shown in the following zone file fragments. For reverse delegations of >= /24 (for example /24 or /26 - essentially a sub Class C block) the CNAME solution is the most efficient and understandable. For reverse map delegations of < /24 (for example /20 or /18 - essentially a sub Class B or Class A block) the DNAME RR may also be used to provide the same functionality while creating a significantly smaller and more flexible configuration.

The following fragment shows our 192.168.23.64/27 subnet as a fragment of the reverse mapping zone file located at the ISP or other Authority that assigned the subnet:

```
$TTL 2d ; 172800 seconds
$ORIGIN 23.168.192.IN-ADDR.ARPA.
@            IN  SOA   ns1.isp.com. hostmaster.isp.com. (
                                2003080800 ; serial number
                                3h         ; refresh
                                15m        ; update retry
                                3w         ; expiry
                                3h         ; nx = nxdomain ttl
                                )
              IN  NS      ns1.isp.com.
              IN  NS      ns2.isp.com.
; definition of other IP address 0 - 31
....

; definition of our target 192.168.23.64/27 subnet
; name servers for subnet reverse map
64/27         IN  NS  ns1.example.com.
64/27         IN  NS  ns2.example.com.
; IPs addresses in the subnet - all need to be defined
; except 64 and 95 since they are the subnets
; network and broadcast addresses not hosts/nodes
65            IN  CNAME   65.64/27.23.168.192.IN-ADDR.ARPA. ;qualified
66            IN  CNAME   66.64/27 ;unqualified name
67            IN  CNAME   67.64/27
....
93            IN  CNAME   93.64/27
94            IN  CNAME   94.64/27
; end of 192.168.23.64/27 subnet
.....
; other subnet definitions
; which may be delegated or local
....
; local IP definitions
; CNAME and PTR RRs may be mixed in the same file
129           IN  PTR bill.isp.com.
....
```

Notes:

1. Only the zone file at the ISP/Delegation Authority is changed. No change is required to the named.conf file. Reason: If, say, the CNAME 65.64/27.23.168.192.IN-ADDR.ARPA is obtained during the search of the zone file 23.168.192.IN-ADDR.ARPA, DNS software will, as normal, attempt to follow the CNAME since the domain name part is within the current zone. In this case the NS RRs for 64/27 are encountered resulting in a referral to the name servers ns1.example.com and ns2.example.com being returned but with a CNAME

(essentially a new query name) of, for instance, of 65.64/27.23.168.192.IN-ADDR.ARPA..

2. The extended name space ,say, 64/27.23.168.192.IN-ADDR.ARPA (as in the example above) is only 'visible' from the parent zone (which contains the CNAME RRs) to the sibling reverse delegation map zones which implement the extended addresses. The DNS hierarchy continues to see a normal IP address reverse lookup and delegates normally through the IN-ADDR.ARPA domain chain. Thus, to do a reverse lookup in the example above for 192.168.23.66 a normal query is issued to the DNS hierachy for a PRT RR at 66.23.168.192.IN-ADDR.ARPA..

The zone file at the DNS serving the Reverse Map (ns1.example.com in the above example) looks like this:

```
$TTL 2d ; 172800
$ORIGIN 64/27.23.168.192.IN-ADDR.ARPA.
@            IN  SOA   ns1.example.com. hostmaster.example.com. (
                                2003080800 ; serial number
                                3h         ; refresh
                                15m        ; update retry
                                3w         ; expiry
                                3h      ; nx = nxdomain ttl
                                )
               IN  NS      ns1.example.com.
               IN  NS      ns2.example.com.
; IPs addresses in the subnet - all need to be defined
; except 64 and 95 since they are the subnets
; network and broadcast addresses not hosts/nodes
65             IN  PTR   fred.example.com. ;qualified
66             IN  PTR   joe.example.com.
67             IN  PTR   bill.example.com.
....
93             IN  PTR   web.example.com.
94             IN  PTR   ftp.example.com.
; end of 192.168.23.64/27 subnet
```

3.4 IPv6 Reverse Mapping

IPv6 addresses are forward-mapped using AAAA RRs and reverse mapped using normal PTR RRs.

Unlike IPv4, where reverse mapping is frequently not delegated to the end user, IPv6 allows and encourages delegated reverse mapping. The end user can therefore be responsible for creation of reverse-mapping zone files using the IP6.ARPA domain for the address range they have been assigned, depending on the policies of the IPv6 address assignor - RIR/LIR(ISP/SP) (Regional Internet Registry/Local Internet Registry (Internet Service Provider/Service Provider)). The IP6.ARPA domain is similar to the IN-ADDR.ARPA domain used for reverse mapping of IPv4 addresses and is shown in Figure 3.2 below.

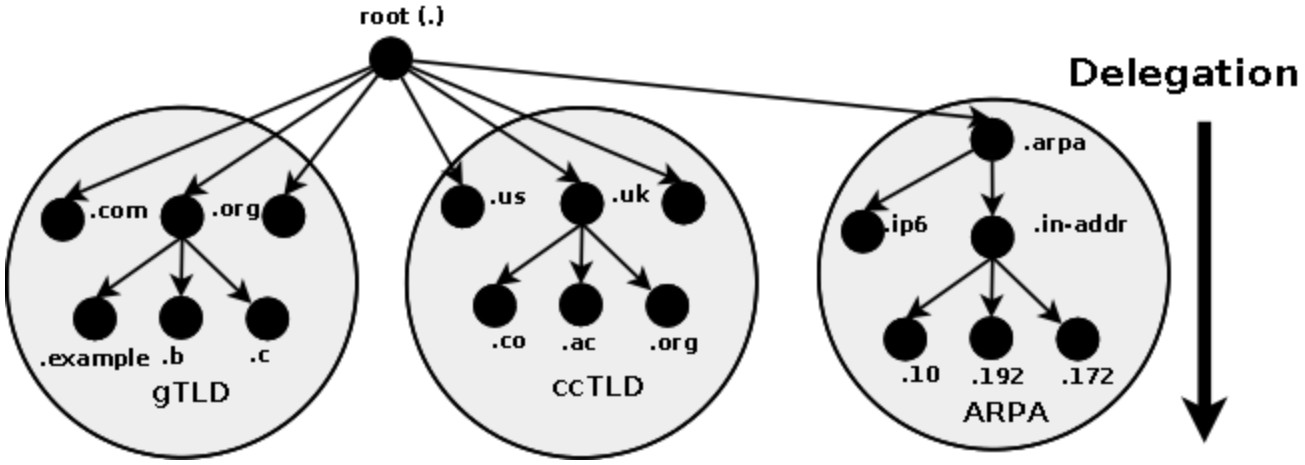


Figure 3.2 IP6.ARPA Reverse Mapping

Assume the user has been allocated from a RIR (Regional Internet Registry) or an LIR(ISP/SP) a fairly typical IPv6 range:

```
2001:db8:0::/48
```

Note IPv6 block allocations, like many other issues in IPv6, are still relatively fluid but are typically /48 or /56 - exceptionally /64.

Assume also, that the user has been delegated, by the RIR/LIR(ISP/SP), the responsibility for reverse mapping the 80-bit addresses in this range (128 - 48 = 80 = user part). IPv6 reverse mapping uses the normal principle of reversing the address and placing the result, in the case of IPv6, under the domain IP6.ARPA. The key difference from the IPv4 IN-ADDR.ARPA domain (described previously) is that a nibble (4 bits) is the unit of delegation. In the context of reverse mapping, each hexadecimal character in the IPv6 address string constitutes a nibble. To illustrate how this works, we must write each character - with no zero elimination - of the assigned addresses range:

```
2001:db8:0::/48
```

Each character is reversed and separated with the normal dot notation to give a reverse-map domain name as shown here:

```
0.0.0.0.8.b.d.0.1.0.0.2.IP6.ARPA
```

Now assume, purely for the sake of simplicity, that only 256 active host addresses are used:

```
2001:db8:0::/120 # 256 hosts
# range 2001:db8:0::0 to 2001:db8:0::FF
```

Based on this we can construct a zone file to contain the definitions as shown here:

```
; reverse IPV6 zone file for example.com
$TTL 2d    ; default TTL for zone
$ORIGIN 0.0.0.0.8.b.d.0.1.0.0.2.IP6.ARPA.
; Start of Authority RR defining the key characteristics of the zone (domain)
@         IN      SOA   ns1.example.com. hostmaster.example.com. (
                        2010121500 ; sn = serial number
                        12h        ; refresh = refresh
                        15m        ; retry = refresh retry
                        3w         ; expiry = expiry
                        2h         ; nx = nxdomain ttl
                        )
; name server RRs for the domain
          IN      NS      ns1.example.com.
; the second name server is
; external to this zone (domain).
          IN      NS      ns2.example.net.
; PTR RR maps a IPv6 address to a host name

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         IN      PTR     ns1.example.com.
2.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0         IN      PTR     mail.example.com.
....
F.F.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         IN      PTR     fred.example.com.
```

IPv6 reverse zone names (and skeleton reverse zone files) may be generated using the IPv6 address tool.

The individual PTR address labels can become brutally long. The constructed domain name, however, does not have to reflect the address segmentation between the global routing prefix and the end-user part of the address as shown in the preceding example. If we assume that we will only ever have a maximum of 65,535 hosts (uses only the right-most 16 bits of the Interface ID or 2001:db8::/112), then we can move some more of the end-user address into the zone domain name, which is written once, to reduce the address part in each PTR line, which may be written many hundreds of times. Thus, the IPv6 address splits in the table below achieve the same result.

Zone Name	PTR Part	Note
0.0.0.0.8.b.d.0.1.0.0.2.IP6.ARPA.	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	Uses a split based on the global routing prefix.
0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.b.d.0.1.0.0.2.IP6.ARPA.	1.0.0.0	Uses a split based on user convenience to reduce the size of each PTR RR.

The zone file to implement this alternate structure is shown next:

```
; reverse IPV6 zone file for example.com
$TTL 2d    ; default TTL for zone
$ORIGIN 0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.b.d.0.1.0.0.2.IP6.ARPA.
; Start of Authority RR defining the key characteristics of the zone (domain)
@         IN      SOA   ns1.example.com. hostmaster.example.com. (
                        2010121500 ; sn = serial number
                        12h        ; refresh = refresh
                        15m        ; retry = refresh retry
                        3w         ; expiry = expiry
                        2h         ; nx =  = nxdomain ttl
                        )
; name server RRs for the domain
          IN      NS      ns1.example.com.
```

```
; the second name server is
; external to this zone (domain).
          IN      NS      ns2.example.net.
; PTR RR maps a IPv6 address to the hostnames

; PTR RR maps a IPv6 address to the hostnames
1.0.0.0.            IN      PTR     joe.example.com.
2.0.0.0             IN      PTR     www.example.com.
....
F.F.0.0             IN      PTR     fred.example.com.
```

Note: Any address not defined in either of the above zone files will, as normal, generate an NXDOMAIN (name error) response. In the case of IPv6 reverse-mapping (and forward-mapping) may not be a simple process as discussed in the next section.

3.5 IPv6 Reverse Mapping Considerations

In the case of an IPv6 end-user delegation, the normal expectation of IPv6 is that any address that is forward mapped using an AAAA (colloquially a Quad A) RR is also reverse mapped using a PTR RR. Specifically, RFC 1912 (an INFORMATIONAL RFC) says "PTR records must point back to a valid A record" and that administrators should "Make sure your PTR and A records match." For the following reasons this may not be as simple as it sounds when using IPv6:

1. In many cases IPv6 addresses are configured using SLAAC (StateLess Address Auto Configuration) or DHCPv6 and therefore may not be known other than by manual inspection. Methods involving DHCPv6 may be configured to provide Dynamic DNS (DDNS) updates to the forward and reverse-mapped zones though in the case of very large networks even this can be problematic due to the potentially large number of hosts involved.
2. Some modern OSes (notably the Windows family) can generate essentially random IPv6 addresses for each session to provide some level of end-user/host privacy (defined by RFC 4941). These changes can only be mapped using DDNS since they are by nature both dynamic and transient. However, RFC 4941 also notes that a DNS entry defeats the object of privacy and instead recommends to either not register in the forward or reverse map, or to used some randomized host name if forced to do so by operational needs (some security systems do a reverse DNS lookup and reject failures).

This topic is currently the subject of considerable discussion (Dec 2010). It is worth noting however, that as with the equivalent IPv4 addresses, the only current applications which are known to use reverse mapping consistently are mail systems. Thus, steps should be taken to ensure that, at least, these hosts have a valid IPv6 reverse-map.

4. DNS Configuration Types

Most DNS may be masters (authoritative) for some zones, slaves for others and provide caching or forwarding for all others. The following terms are commonly used to describe the primary function or requirement of DNS servers.

Notes:

1. **Running any DNS server that supports recursive queries received from any or all users (an Open DNS) is an Extremely Bad Idea ™. While an Open DNS may look like a friendly and neighbourly thing to do such a server may be used in DDoS attacks and carries a significantly increased risk of cache poisoning. It is always possible to define the range of IP addresses that are allowed or permitted to use the recursive feature of any DNS server. Such a server is termed Closed. The various configurations have been modified to ensure that the DNS stays Closed to non-permitted users.**
2. **One of the basic rules of security is that only the minimum services necessary to meet the objectives should be deployed. This means that a secure DNS server should provide only a single function, for instance, authoritative only, or caching only, not both capabilities in the same server. This is a correct but idealistic position, generally possible only in larger organizations. In practice many of us run mixed mode DNS servers. While much can be done to mitigate any security implications it must always be accepted that, in mixed configurations, increased risk is the downside of flexibility.**

4.1 Master (Primary) Name Servers

A Master DNS defines one or more zone files for which this DNS is **Authoritative** ('type master'). The zone has been delegated (via an NS Resource Record) to this DNS.

The term **master** was introduced with BIND 8.x and replaced the term 'primary'.

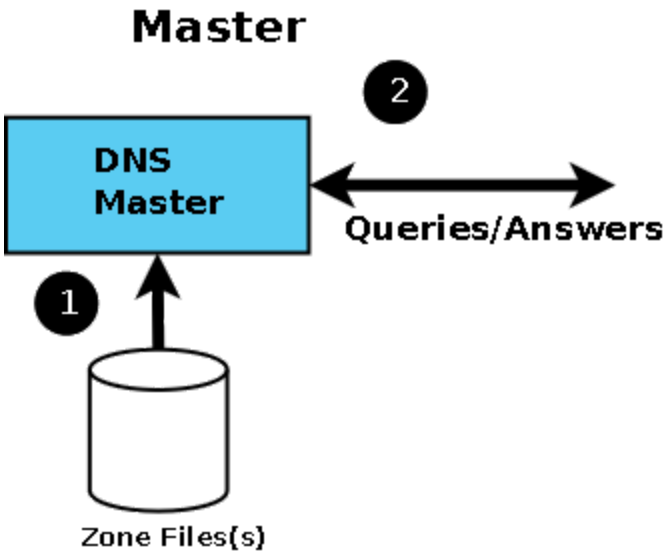


Diagram 1 DNS Master

Master status is defined in BIND by including 'type master' in the zone declaration section of the named.conf file as shown by the following fragment.

```
// example.com fragment from named.conf
// defines this server as a zone master
zone "example.com" in{
     type master;
     file "pri.example.com";
};
```

Notes:

1. It is important to understand that a zone 'master' is simply a server which gets its zone data from a local source as opposed to a 'slave' which gets its zone data from an external (networked) source (typically the 'master' but not always). This apparently trivial point means that you can have any number of 'master' servers for any zone if it makes operational sense. You have to ensure (by a manual or other process) that the zone files are synchronised but apart from this there is nothing to prevent it.
2. Just to confuse things still further you may run across the term 'Primary Master' this has a special meaning in the context of dynamic DNS updates and is defined to be the name server that appears in the SOA RR record.

When a master DNS receives Queries for a zone for which it is authoritative then it will respond as 'Authoritative' (AA bit is set in a query response).

If a DNS server receives a query for a zone for which it is neither a Master nor a Slave then it will act as configured (in BIND this behaviour is defined in the named.conf file):

1. If caching behaviour is permitted and recursive queries are allowed the server will completely answer the request or return an error.
2. If caching behaviour is permitted and Iterative (non-recursive) queries are allowed the server can respond with the complete answer (if it is already in the cache because of another request), a referral or return an error.
3. If caching behaviour is NOT permitted (an 'Authoritative Only' DNS server) the server will return a referral or an error.

A master DNS server can NOTIFY zone changes to defined (typically slave) servers - this is the default behaviour. NOTIFY messages ensure zone changes are rapidly propagated to the slaves (interrupt driven) rather than rely on the slave server periodically polling for changes. The BIND default is to notify the servers defined in NS records for the zone - except itself, obviously.

A zone master can be 'hidden' (only one or more of the slaves know of its existence). There is no requirement in such a configuration for the master server to appear in an NS RR for the domain. The only requirement is that two (or more) name servers support the zone. Both servers could be any combination of master-slave, slave-slave or even master-master.

4.2 Slave Name Servers

A Slave DNS gets its zone data using a zone transfer operation (typically from a zone master) and it will respond as authoritative for those zones for which it is defined to be a 'slave' and for which it has a currently valid zone configuration. It is impossible to determine from a query result that it came from a zone master or slave.

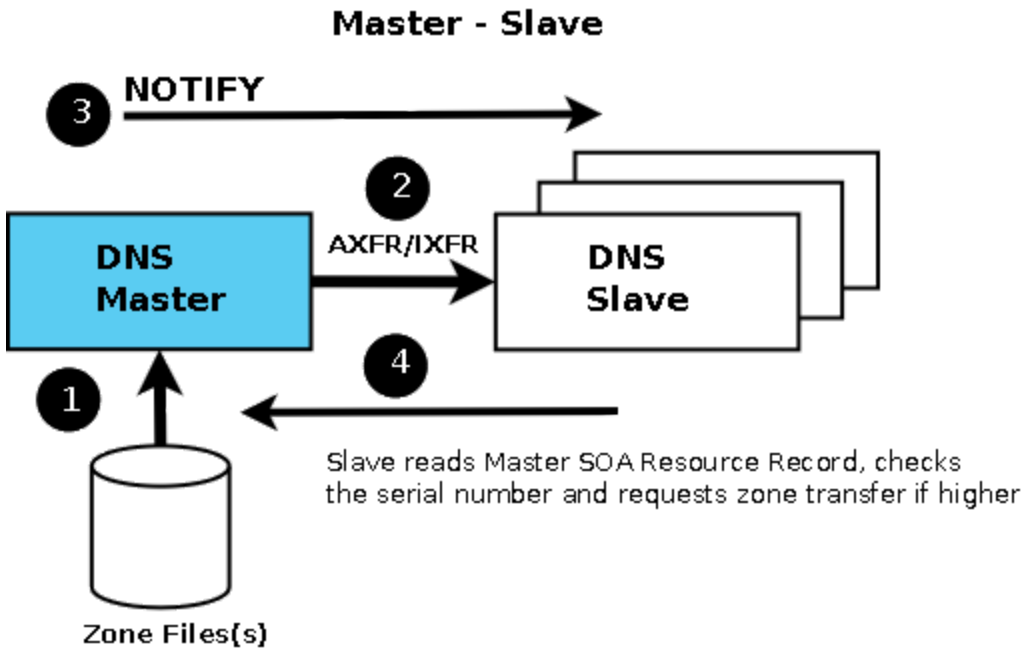


Diagram 2 DNS Slave Server

The term 'slave' was introduced with BIND 8.x and replaced the term 'secondary'.

There can be any number of slave DNS's for any given zone.

Slave status is defined in BIND by including 'type slave' in the zone declaration section of the named.conf file as shown by the following fragment.

```
// example.com fragment from named.conf
// defines this server as a zone slave
zone "example.com" in{
     type slave;
     file "sec/sec.example.com";
     masters {192.168.23.17;};
};
```

Notes:

1. The master DNS for each zone is defined in the 'masters' statement of the zone clause and allows slaves to refresh their zone record when the 'expiry' parameter of the SOA Record is reached. If a slave cannot reach the master DNS when the 'expiry' time has been reached it will stop responding to requests for the zone. It will not use time-expired data.
2. The file parameter is optional and allows the slave to write the transferred zone to disc and hence if BIND is restarted before the 'expiry' time the Slave server will use the saved data. In large DNS systems this can save a considerable amount of network traffic.

Assuming NOTIFY is allowed in the master DNS for the zone (the default behaviour) then zone changes are propagated to all the servers defined with NS Records in the zone file. Other acceptable NOTIFY sources can be defined using the also-notify parameter in named.conf.

4.2.1 But Slaves can also be Masters

The definition of a slave server is simply that it gets its zone data via zone transfer, whereas a master gets its zone data from a local file system. The source of the zone transfer could just as easily be another slave as a master.

1. Assume you want to hide your master servers in, say, a stealth configuration then at least one slave server will sit on the public side of a firewall, or similar configuration, providing perimeter defence. To provide resilience you would need two or more such public slaves. The second slave can be updated from the same master as the first or it could be updated from the slave server - we'll call it the 'boss' slave to avoid getting into tortuous terminology (is it a master-slave or a slave-master?). To configure this miracle the second slave server would define the 'boss' slave's IP in its masters statement. When the 'boss' slave has sucessfully transfered a zone file (from the master) it will send out NOTIFY messages (the default) unless configured not to do so. This type of configuration will marginally increase latency for updating the zone on the second slave - but that may be more than offset by increased stealth.
2. In a DNSSEC environment the master will likely have all kinds of whizzo dodads concerned with keeping keys secure. Whereas DNSSEC slaves simply send the data in the zone file in response to queries and have no requirements for secure key

maintenance. Hidden master configurations will become increasingly the norm in this environment.

4.3 Caching Name Servers (Resolver)

A DNS Caching Server (frequently called a Resolver) obtains information from another server (a Zone Master) in response to a host query and then saves (caches) the data locally. On a second or subsequent request for the same data the Caching Server (Resolver) will respond with its locally stored data (the cache) until the time-to-live (TTL) value of the response expires, at which time the server will refresh the data from the zone master.

If the caching server (resolver) obtains its data directly from a zone master it will respond as 'authoritative', if the data is supplied from its cache the response is 'non-authoritative'.

The default BIND behaviour is to cache and this is associated with the recursion parameter (the default is 'recursion yes'). There are many configuration examples which show caching behaviour being defined using a *type hint* statement in a zone declaration. These configurations confuse two distinct but related functions. If a server is going to provide caching services then it must support recursive queries and recursive queries need access to the root servers which is provided via the 'type hint' statement.

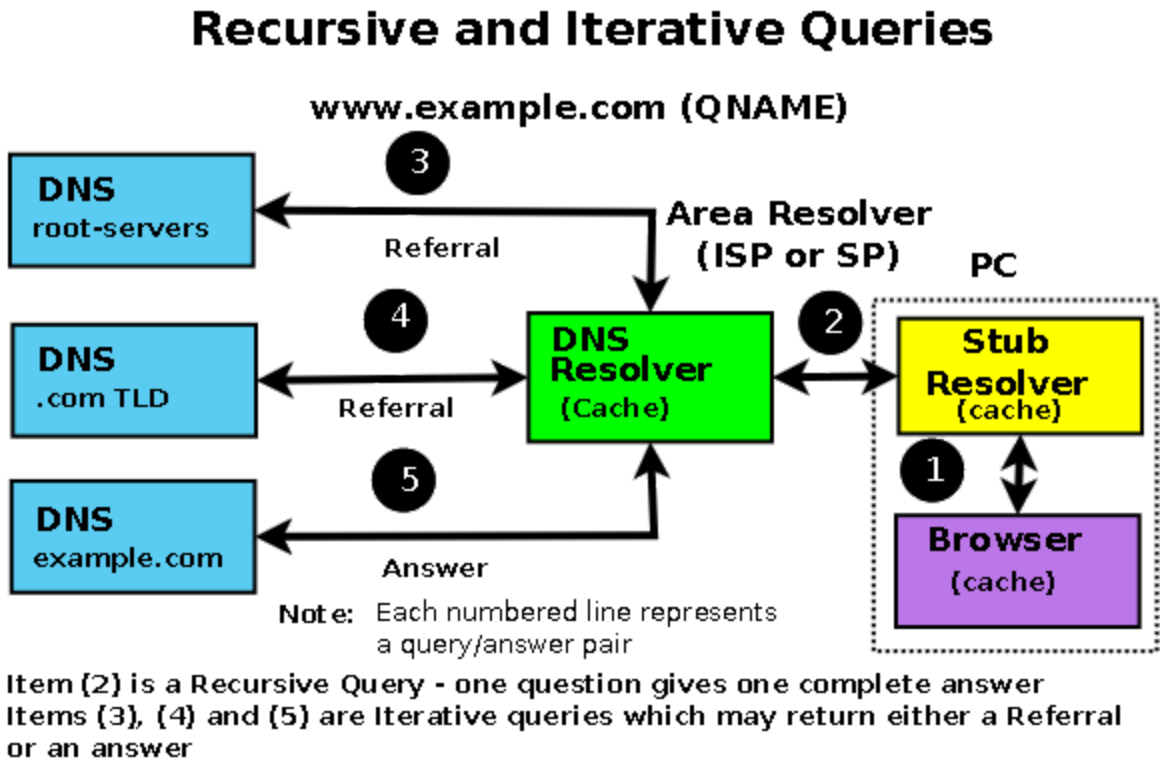


Diagram 3 DNS Resolver (Recursive Server)

```
// options section fragment of named.conf
// recursion yes is the default and may be omitted
options {
     directory "/var/named";
     version "not currently available";
     recursion yes;
};
// zone section
....
// the DOT indicates the root domain = all domains
zone "." IN {
     type hint;
     file "root.servers";
};
```

Notes:

1. BIND defaults to recursive queries which by definition provides caching behaviour. The named.conf recursion parameter controls this behaviour.
2. The zone '.' is shorthand for the root domain which translates to 'any domain not defined as either a master or slave in this named.conf file'.
3. cache data is discarded when BIND is restarted.

The most common DNS server caching configurations are:

- A DNS server acting as master or slave for one or more zones (domains) and as cache server for all other requests. A general purpose DNS server.
- A caching only local server - typically used to minimise external access or to compensate for slow external links. This is sometimes called a Proxy server though we prefer to associate the term with a Forwarding server

To cache or not is a crucial question in the world of DNS. BIND is regarded as the reference implementation of the DNS specification. As such it provides excellent - if complex to configure - functionality. The down side of generality is suboptimal performance on any single function - in particular caching involves a non-trivial performance overhead.

Note: The response to a query is Authoritative under three conditions:

1. The response is received from a Zone master.
2. The response is received from a Zone slave with non time-expired zone data.
3. The response is received by a caching server directly from either a Zone master or slave. If the response is supplied from the cache it is not authoritative.

4.4 Forwarding (a.k.a Proxy) Name Servers

A forwarding (a.k.a. Proxy, Client, Remote) server is one which simply forwards requests to another DNS and caches the results. It is undervalued and extremely useful configuration in a number of situations:

1. Where access to the external network is slow or expensive:
    1. Local DNS caching - results are cached in the forwarding server so that frequently requested domains will provide fast results from the cache.
    2. The Remote (forwarded to) DNS server provides recursive query support resulting in a single query across the network (from the forwarding DNS to the 'forwared to' DNS) thus reducing traffic congestion (on busy networks), traffic volume (on expensive networks) and increasing performance (on slow networks).
2. Forwarding servers also can be used to ease the burden of local administration by providing a single point at which changes to remote name servers may be managed, rather than having to update all hosts. Thus, all hosts in a particular network section or area can be configured to point to a fixed forwarding DNS which can be configured to stream DNS traffic as desired and changed over time with minimal effort.
3. Sanitizing traffic. Especially in larger private networks it may be sensible to stream DNS traffic for local domain access by forwarding to the local DNS servers while forwarding external DNS requests to a **dirty** or hardened caching DNS (or resolver).
4. Forwarding can also be used as part of a Split Server configuration for perimeter defence.

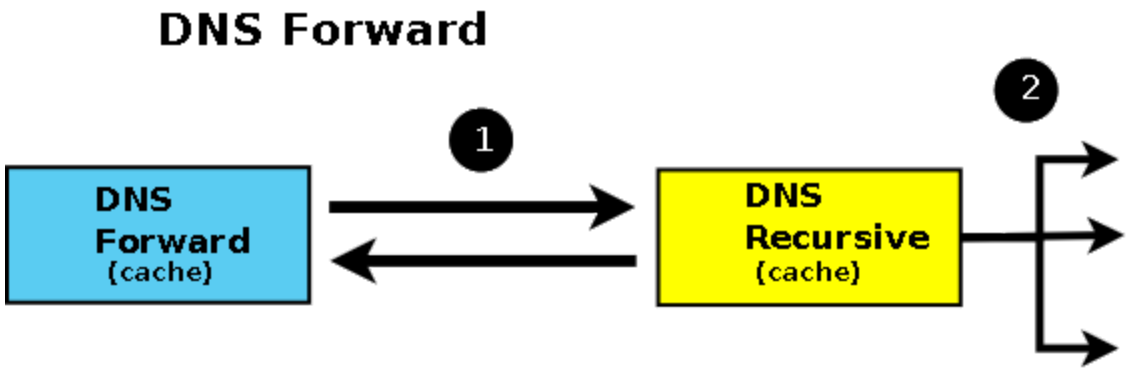


Diagram 4 - DNS Forvarding Server

Ver en el libro los comandos para forwarding en bind.

4.5 Stealth (a.k.a. DMZ or Hidden Master) Name Server

A stealth server is defined as being a name server which does not appear in any **publicly visible** NS Records for the domain. The stealth server can be roughly defined as having the following characteristics:

1. The organisation needs a public DNS to enable access to its public services e.g. web, mail ftp etc..
2. The organisation does not want the world to see any of its internal hosts either by interrogation (query or zone transfer) or should the DNS service be compromised.

A Stealth configuration is shown in Figure 4-5.

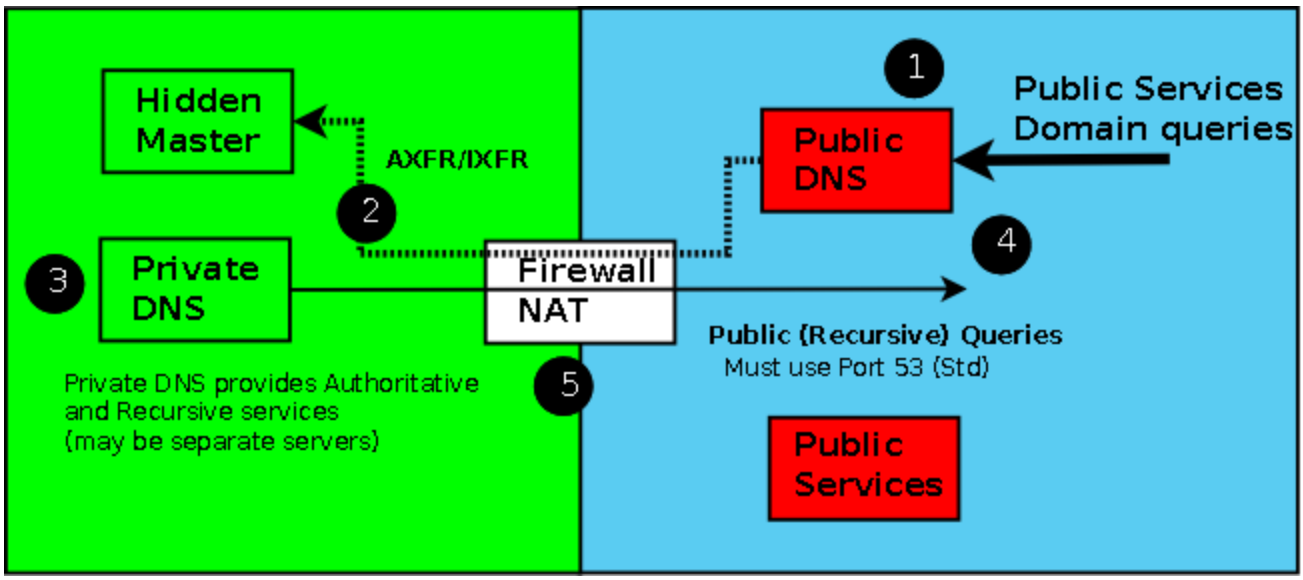


Figure 4-5 Stealth Server Topology

The external server(s) is(are) configured to provide Authoritative Only responses and no caching (no recursive queries accepted). The zone file for this server would be unique and would contain ONLY those systems or services that are publicly visible e.g. SOA, NS records for the public (not stealth) name servers, MX record(s) for mail servers and www and ftp service A records. Zone transfers can be allowed between the public servers as required but they MUST NOT transfer or accept transfers from the Stealth server. While this may seem to create more work, the concern is that should the host running the external service be compromised then inspection of the named.conf or zone files must provide no more information than is already publically visible. If 'master', 'allow-notify','allow-transfer' options are present in named.conf (each of which will contain a private IP) then the attacker has gained more knowledge about the organisation - they have penetrated the 'veil of privacy'.

A minimal public zone file is shown below:

```
; public zone master file
; provides minimal public visibility of external services
example.com.   IN      SOA   ns.example.com. root.example.com. (
                               2003080800 ; se = serial number
```

```
                              3h         ; ref = refresh
                              15m        ; ret = update retry
                              3w         ; ex = expiry
                              3h      ; min = minimum
                              )
              IN      NS      ns1.example.com.
              IN      NS      ns2.example.com.
              IN      MX  10  mail.example.com.
ns1           IN      A       192.168.254.1
ns2           IN      A       192.168.254.2
mail          IN      A       192.168.254.3
www           IN      A       192.168.254.4
ftp           IN      A       192.168.254.5
```

The internal server (the Stealth Server) can be configured to make visible internal and external services, provide recursive queries and all manner of other services. This server would use a private zone master file which could look like this:

```
; private zone master file used by stealth server(s)
; provides public and private services and hosts
example.com.  IN      SOA   ns.example.com. root.example.com. (
                              2003080800 ; se = serial number
                              3h         ; ref = refresh
                              15m        ; ret = update retry
                              3w         ; ex = expiry
                              3h         ; min = minimum
                              )
              IN      NS      ns1.example.com.
              IN      NS      ns2.example.com.
              IN      MX  10  mail.example.com.
; public hosts
ns1           IN      A       192.168.254.1
ns2           IN      A       192.168.254.2
mail          IN      A       192.168.254.3
www           IN      A       192.168.254.4
ftp           IN      A       192.168.254.5
; private hosts
joe           IN      A       192.168.254.6
bill          IN      A       192.168.254.7
fred          IN      A       192.168.254.8
....
accounting    IN      A       192.168.254.28
payroll       IN      A       192.168.254.29
```

4.6 Authoritative Only Server

The term **Authoritative Only** is normally used to describe two concepts:

1. The server will deliver Authoritative Responses - it is a zone master or slave for one or more domains.
2. The server will NOT cache.

There are two configurations in which Authoritative Only servers are typically used:

1. As the public or external server in a Stealth (a.k.a. DMZ or Hidden Master) DNS used to provide perimeter security.
2. High Performance DNS servers.

You cannot completely turn off caching in BIND but you can control it and provide the functionality described above by simply turning off recursion in the 'option' section of named.conf as shown in the example below.

4.7 Split Horizon DNS Server

The term **Split Horizon** is normally used to describe a DNS server that will give different responses (IP addresses) based on the source address, or some other characteristic, of the query. While it has similar configuration properties to the Stealth Server it can also be used in a varity of unique situations such as:

1. **Geographic Mapping:** Assume that, for example, a web service is replicated in a number of locations (for either performance or access latency reasons) then a specific IP address may be returned based on the source address of the query to ensure the shortest possible path from the user to the service. For those familiar with **anycast** you could consider this as a poor man's anycast service.
2. **Naming Consistency:** Assume that you have, say, a corporate in-house LDAP service and that you want to keep certain highly secure data on one server only accessible to certain individuals or organizational sections, which have unique or identifiable IP addresses or address ranges, but for reasons of consistency (scripts, configuration files etc) you want both the secure and insecure LDAP services to be named, say, ldap.example.com.
3. **Load Balancing:** Assume that an analysis of incoming service users shows that their source-ip addresses can be separated into contiguous ranges: 50% from a to b, 50% from b to c. In this case rather than simply provide multiple A/AAAA RRs (where load balancing is essentially random) it may be more effective to use a split-horizon strategy.

Chapter 8. DNS Resource Records (RRs)

DNS Resource Record (RR) Types

RR	Value	RFC	Description
A	1	RFC 1035	IPv4 Address record. An IPv4 address for a host.
AAAA	28	RFC 3596	IPv6 Address record. An IPv6 address for a host.
CNAME	5	RFC 1035	Canonical Name. An alias name for a host. Causes redirection for a single RR at the owner-name.
MX	15	RFC 1035	Mail Exchanger. A preference value and the host name for a mail server/exchanger that will service this zone. RFC 974 defines valid names.
NS	2	RFC 1035	Name Server. Defines the authoritative name server(s) for the domain (defined by the SOA record) or the subdomain.
PTR	12	RFC 1035	IP address (IPv4 or IPv6) to host. Used in reverse maps.
TXT	16	RFC 1035	Text information associated with a name. An SPF record should be defined using a TXT record. DKIM (RFC 4871 also makes use of the TXT RR for authenticaing email.

IPv4 Address Record (A)

Defined in RFC 1035. Forward maps a host name to IPv4 address. The only parameter is an **ipv4** field which is a single IPv4 address in dotted decimal format. The **ipv4** field is an address not a label (name) and therefore is not terminated with a '.' (dot). Valid owner-name format (a.k.a 'label' in the DNS jargon). If host name is BLANK (or space) then the last valid name (or label) is substituted.

Format

```
owner-name  ttl  class   rr     ipv4
joe             IN      A      192.168.254.3
```

IPv6 Address Record (AAAA)

The current IETF recommendation is to use AAAA (Quad A) RR for forward mapping and PTR RRs for reverse mapping when defining IPv6 networks. The IPv6 AAAA RR is defined in RFC 3596. RFC 6563 plunged the dagger into the heart of the short-lived and ill-fated A6 RR by moving it to historical status. (Tech Stuff - IPv6 Protocol.) IPv6 Reverse Mapping is defined in Chapter 3. IPv6 Address Calculator and reverse map generator.

Syntax of AAAA RR

```
name  ttl  class   rr     ipv6
joe       IN      A      2001:db8::1
```

Canonical Name Record (CNAME)

A CNAME record maps a single alias or nickname to the real or Canonical name which may lie outside the current zone. Canonical simply means the expected or real name.

Format

```
name  ttl  class   rr     canonical name
www       IN      CNAME  joe.example.com.
```

Mail Exchange Record (MX)

Defined in RFC 1035. Specifies the name and relative preference of mail servers (mail exchangers in the DNS jargon) for the zone. The MX RR is used by external SMTP (Mail) Agents to route incoming mail for the domain.

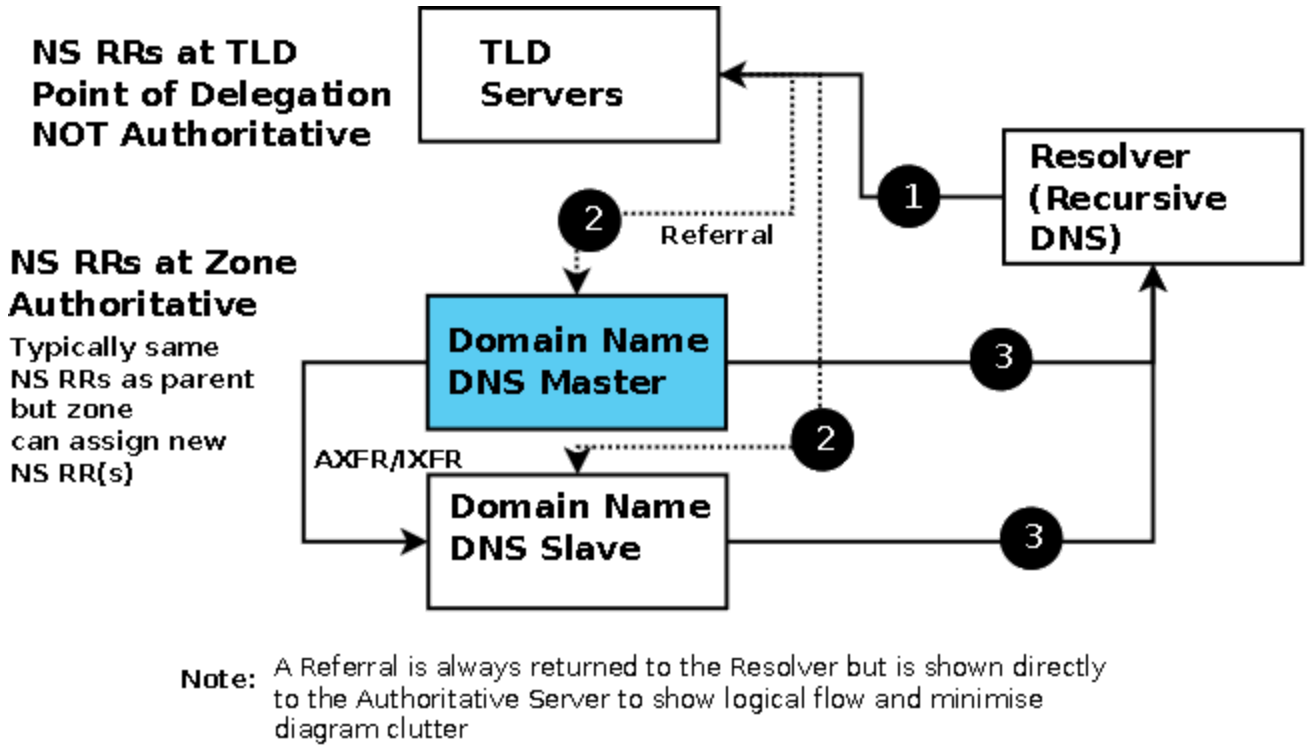
Format

```
owner-name          ttl  class   rr  pref name
example.com.        3w   IN      MX  10   mail.example.com.
```

Name Server Record (NS)

Defined in RFC 1035. NS RRs appear in two places. Within the zone file, in which case they are authoritative records for the zone's name servers. At the *point of delegation* for either a subdomain of the zone or in the zone's parent in which case they are non-authoritative. Thus, the zone example.com's parent zone (.com) will contain non-authoritative NS RRs for the zone example.com at its **point of delegation** (point of delegation is the term frequently used to describe the NS RRs in the parent that delegate a zone of subdomain) and subdomain.example.com will have non-authoritative NS RRS in the zone example.com

at its *point of delegation*. NS RRs at a *point of delegation* are never authoritative only NS RRs within the zone are regarded as authoritative. While this may look a fairly trivial point, is has important implications for DNSSEC. This relationship is shown in the Diagram below.



NS Parent - Child Delegation

NS RRs are required because DNS queries respond with an authority section listing all the authoritative name servers and for queries to sub-domains (including the zone's parent) where they are required to allow referral to take place.

Format

```
owner-name      ttl   class    rr      target-name
example.com.          IN       NS       ns1.example.com.
```

Pointer Record (PTR)

Pointer records are the opposite of A and AAAA RRs and are used in Reverse Map zone files to map an IP address (IPv4 or IPv6) to a host name.

Format

```
name ttl   class    rr      name
15          IN      PTR     www.example.com.
```

The value '15' (the base IP address) in the above example is actually a **name** (an owner-name or left-hand name) and because it does not teminate with a 'dot' BIND

appends the $ORIGIN (or if an $ORIGIN is missing the zone name that referenced this zone file).

Text Record (TXT)

Provides the ability to associate some arbirary and unformatted text with a host or other name. The TXT record is used to define the Sender Policy Framework (SPF) (RFC 7208) and DomainKeys Signed Mail (DKIM) (RFC 4871 and RFC 5617) information records which may be used to validate legitimate email sources from a domain. In both the case of SPF and DKIM it is the application entity that defines and interprets the text format - as far as the DNS is concerned the records are simply unformatted text.

Format

```
name  ttl  class   rr     text
joe        IN      TXT    "Located in a black hole"
```