

DOMAIN NAMES - CONCEPTS AND FACILITIES

1. STATUS OF THIS MEMO

This RFC is an introduction to the Domain Name System (DNS), and omits many details which can be found in a companion RFC, "Domain Names - Implementation and Specification" [RFC-1035]. That RFC assumes that the reader is familiar with the concepts discussed in this memo.

A subset of DNS functions and data types constitute an official protocol. The official protocol includes standard queries and their responses and most of the Internet class data formats (e.g., host addresses).

However, the domain system is intentionally extensible. Researchers are continuously proposing, implementing and experimenting with new data types, query types, classes, functions, etc. Thus while the components of the official protocol are expected to stay essentially unchanged and operate as a production service, experimental behavior should always be expected in extensions beyond the official protocol. Experimental or obsolete features are clearly marked in these RFCs, and such information should be used with caution.

The reader is especially cautioned not to depend on the values which appear in examples to be current or complete, since their purpose is primarily pedagogical. Distribution of this memo is unlimited.

2. INTRODUCTION

This RFC introduces domain style names, their use for Internet mail and host address support, and the protocols and servers used to implement domain name facilities.

2.1. The history of domain names

The impetus for the development of the domain system was growth in the Internet:

- Host name to address mappings were maintained by the Network Information Center (NIC) in a single file (HOSTS.TXT) which was FTPed by all hosts [RFC-952, RFC-953]. The total network

bandwidth consumed in distributing a new version by this scheme is proportional to the square of the number of hosts in the network, and even when multiple levels of FTP are used, the outgoing FTP load on the NIC host is considerable. Explosive growth in the number of hosts didn't bode well for the future.

- The network population was also changing in character. The timeshared hosts that made up the original ARPANET were being replaced with local networks of workstations. Local organizations were administering their own names and addresses, but had to wait for the NIC to change HOSTS.TXT to make changes visible to the Internet at large. Organizations also wanted some local structure on the name space.
- The applications on the Internet were getting more sophisticated and creating a need for general purpose name service.

The result was several ideas about name spaces and their management [IEN-116, RFC-799, RFC-819, RFC-830]. The proposals varied, but a common thread was the idea of a hierarchical name space, with the hierarchy roughly corresponding to organizational structure, and names using "." as the character to mark the boundary between hierarchy levels. A design using a distributed database and generalized resources was described in [RFC-882, RFC-883]. Based on experience with several implementations, the system evolved into the scheme described in this memo.

The terms "domain" or "domain name" are used in many contexts beyond the DNS described here. Very often, the term domain name is used to refer to a name with structure indicated by dots, but no relation to the DNS. This is particularly true in mail addressing [Quarterman 86].

2.2. DNS design goals

The design goals of the DNS influence its structure. They are:

- The primary goal is a consistent name space which will be used for referring to resources. In order to avoid the problems caused by ad hoc encodings, names should not be required to contain network identifiers, addresses, routes, or similar information as part of the name.
- The sheer size of the database and frequency of updates suggest that it must be maintained in a distributed manner, with local caching to improve performance. Approaches that

attempt to collect a consistent copy of the entire database will become more and more expensive and difficult, and hence should be avoided. The same principle holds for the structure of the name space, and in particular mechanisms for creating and deleting names; these should also be distributed.

- Where there tradeoffs between the cost of acquiring data, the speed of updates, and the accuracy of caches, the source of the data should control the tradeoff.
- The costs of implementing such a facility dictate that it be generally useful, and not restricted to a single application. We should be able to use names to retrieve host addresses, mailbox data, and other as yet undetermined information. All data associated with a name is tagged with a type, and queries can be limited to a single type.
- Because we want the name space to be useful in dissimilar networks and applications, we provide the ability to use the same name space with different protocol families or management. For example, host address formats differ between protocols, though all protocols have the notion of address. The DNS tags all data with a class as well as the type, so that we can allow parallel use of different formats for data of type address.
- We want name server transactions to be independent of the communications system that carries them. Some systems may wish to use datagrams for queries and responses, and only establish virtual circuits for transactions that need the reliability (e.g., database updates, long transactions); other systems will use virtual circuits exclusively.
- The system should be useful across a wide spectrum of host capabilities. Both personal computers and large timeshared hosts should be able to use the system, though perhaps in different ways.

2.3. Assumptions about usage

The organization of the domain system derives from some assumptions about the needs and usage patterns of its user community and is designed to avoid many of the the complicated problems found in general purpose database systems.

The assumptions are:

- The size of the total database will initially be proportional

to the number of hosts using the system, but will eventually grow to be proportional to the number of users on those hosts as mailboxes and other information are added to the domain system.

- Most of the data in the system will change very slowly (e.g., mailbox bindings, host addresses), but that the system should be able to deal with subsets that change more rapidly (on the order of seconds or minutes).
- The administrative boundaries used to distribute responsibility for the database will usually correspond to organizations that have one or more hosts. Each organization that has responsibility for a particular set of domains will provide redundant name servers, either on the organization's own hosts or other hosts that the organization arranges to use.
- Clients of the domain system should be able to identify trusted name servers they prefer to use before accepting referrals to name servers outside of this "trusted" set.
- Access to information is more critical than instantaneous updates or guarantees of consistency. Hence the update process allows updates to percolate out through the users of the domain system rather than guaranteeing that all copies are simultaneously updated. When updates are unavailable due to network or host failure, the usual course is to believe old information while continuing efforts to update it. The general model is that copies are distributed with timeouts for refreshing. The distributor sets the timeout value and the recipient of the distribution is responsible for performing the refresh. In special situations, very short intervals can be specified, or the owner can prohibit copies.
- In any system that has a distributed database, a particular name server may be presented with a query that can only be answered by some other server. The two general approaches to dealing with this problem are "recursive", in which the first server pursues the query for the client at another server, and "iterative", in which the server refers the client to another server and lets the client pursue the query. Both approaches have advantages and disadvantages, but the iterative approach is preferred for the datagram style of access. The domain system requires implementation of the iterative approach, but allows the recursive approach as an option.

The domain system assumes that all data originates in master files scattered through the hosts that use the domain system. These master files are updated by local system administrators. Master files are text files that are read by a local name server, and hence become available through the name servers to users of the domain system. The user programs access name servers through standard programs called resolvers.

The standard format of master files allows them to be exchanged between hosts (via FTP, mail, or some other mechanism); this facility is useful when an organization wants a domain, but doesn't want to support a name server. The organization can maintain the master files locally using a text editor, transfer them to a foreign host which runs a name server, and then arrange with the system administrator of the name server to get the files loaded.

Each host's name servers and resolvers are configured by a local system administrator [RFC-1033]. For a name server, this configuration data includes the identity of local master files and instructions on which non-local master files are to be loaded from foreign servers. The name server uses the master files or copies to load its zones. For resolvers, the configuration data identifies the name servers which should be the primary sources of information.

The domain system defines procedures for accessing the data and for referrals to other name servers. The domain system also defines procedures for caching retrieved data and for periodic refreshing of data defined by the system administrator.

The system administrators provide:

- The definition of zone boundaries.
- Master files of data.
- Updates to master files.
- Statements of the refresh policies desired.

The domain system provides:

- Standard formats for resource data.
- Standard methods for querying the database.
- Standard methods for name servers to refresh local data from foreign name servers.

2.4. Elements of the DNS

The DNS has three major components:

- The DOMAIN NAME SPACE and RESOURCE RECORDS, which are specifications for a tree structured name space and data associated with the names. Conceptually, each node and leaf of the domain name space tree names a set of information, and query operations are attempts to extract specific types of information from a particular set. A query names the domain name of interest and describes the type of resource information that is desired. For example, the Internet uses some of its domain names to identify hosts; queries for address resources return Internet host addresses.
- NAME SERVERS are server programs which hold information about the domain tree's structure and set information. A name server may cache structure or set information about any part of the domain tree, but in general a particular name server has complete information about a subset of the domain space, and pointers to other name servers that can be used to lead to information from any part of the domain tree. Name servers know the parts of the domain tree for which they have complete information; a name server is said to be an AUTHORITY for these parts of the name space. Authoritative information is organized into units called ZONES, and these zones can be automatically distributed to the name servers which provide redundant service for the data in a zone.
- RESOLVERS are programs that extract information from name servers in response to client requests. Resolvers must be able to access at least one name server and use that name server's information to answer a query directly, or pursue the query using referrals to other name servers. A resolver will typically be a system routine that is directly accessible to user programs; hence no protocol is necessary between the resolver and the user program.

These three components roughly correspond to the three layers or views of the domain system:

- From the user's point of view, the domain system is accessed through a simple procedure or OS call to a local resolver. The domain space consists of a single tree and the user can request information from any section of the tree.
- From the resolver's point of view, the domain system is composed of an unknown number of name servers. Each name

server has one or more pieces of the whole domain tree's data, but the resolver views each of these databases as essentially static.

- From a name server's point of view, the domain system consists of separate sets of local information called zones. The name server has local copies of some of the zones. The name server must periodically refresh its zones from master copies in local files or foreign name servers. The name server must concurrently process queries that arrive from resolvers.

In the interests of performance, implementations may couple these functions. For example, a resolver on the same machine as a name server might share a database consisting of the the zones managed by the name server and the cache managed by the resolver.

3. DOMAIN NAME SPACE and RESOURCE RECORDS

3.1. Name space specifications and terminology

The domain name space is a tree structure. Each node and leaf on the tree corresponds to a resource set (which may be empty). The domain system makes no distinctions between the uses of the interior nodes and leaves, and this memo uses the term "node" to refer to both.

Each node has a label, which is zero to 63 octets in length. Brother nodes may not have the same label, although the same label can be used for nodes which are not brothers. One label is reserved, and that is the null (i.e., zero length) label used for the root.

The domain name of a node is the list of the labels on the path from the node to the root of the tree. By convention, the labels that compose a domain name are printed or read left to right, from the most specific (lowest, farthest from the root) to the least specific (highest, closest to the root).

Internally, programs that manipulate domain names should represent them as sequences of labels, where each label is a length octet followed by an octet string. Because all domain names end at the root, which has a null string for a label, these internal representations can use a length byte of zero to terminate a domain name.

By convention, domain names can be stored with arbitrary case, but domain name comparisons for all present domain functions are done in a case-insensitive manner, assuming an ASCII character set, and a high order zero bit. This means that you are free to create a node with label "A" or a node with label "a", but not both as brothers; you could refer to either using "a" or "A". When you receive a domain name or

label, you should preserve its case. The rationale for this choice is that we may someday need to add full binary domain names for new services; existing services would not be changed.

When a user needs to type a domain name, the length of each label is omitted and the labels are separated by dots ("."). Since a complete domain name ends with the root label, this leads to a printed form which ends in a dot. We use this property to distinguish between:

- a character string which represents a complete domain name (often called "absolute"). For example, "poneria.ISI.EDU."
- a character string that represents the starting labels of a domain name which is incomplete, and should be completed by local software using knowledge of the local domain (often called "relative"). For example, "poneria" used in the ISI.EDU domain.

Relative names are either taken relative to a well known origin, or to a list of domains used as a search list. Relative names appear mostly at the user interface, where their interpretation varies from implementation to implementation, and in master files, where they are relative to a single origin domain name. The most common interpretation uses the root "." as either the single origin or as one of the members of the search list, so a multi-label relative name is often one where the trailing dot has been omitted to save typing.

To simplify implementations, the total number of octets that represent a domain name (i.e., the sum of all label octets and label lengths) is limited to 255.

A domain is identified by a domain name, and consists of that part of the domain name space that is at or below the domain name which specifies the domain. A domain is a subdomain of another domain if it is contained within that domain. This relationship can be tested by seeing if the subdomain's name ends with the containing domain's name. For example, A.B.C.D is a subdomain of B.C.D, C.D, D, and " ".

3.2. Administrative guidelines on use

As a matter of policy, the DNS technical specifications do not mandate a particular tree structure or rules for selecting labels; its goal is to be as general as possible, so that it can be used to build arbitrary applications. In particular, the system was designed so that the name space did not have to be organized along the lines of network boundaries, name servers, etc. The rationale for this is not that the name space should have no implied semantics, but rather that the choice of implied semantics should be left open to be used for the problem at

hand, and that different parts of the tree can have different implied semantics. For example, the IN-ADDR.ARPA domain is organized and distributed by network and host address because its role is to translate from network or host numbers to names; NetBIOS domains [RFC-1001, RFC-1002] are flat because that is appropriate for that application.

However, there are some guidelines that apply to the "normal" parts of the name space used for hosts, mailboxes, etc., that will make the name space more uniform, provide for growth, and minimize problems as software is converted from the older host table. The political decisions about the top levels of the tree originated in RFC-920. Current policy for the top levels is discussed in [RFC-1032]. MILNET conversion issues are covered in [RFC-1031].

Lower domains which will eventually be broken into multiple zones should provide branching at the top of the domain so that the eventual decomposition can be done without renaming. Node labels which use special characters, leading digits, etc., are likely to break older software which depends on more restrictive choices.

3.3. Technical guidelines on use

Before the DNS can be used to hold naming information for some kind of object, two needs must be met:

- A convention for mapping between object names and domain names. This describes how information about an object is accessed.
- RR types and data formats for describing the object.

These rules can be quite simple or fairly complex. Very often, the designer must take into account existing formats and plan for upward compatibility for existing usage. Multiple mappings or levels of mapping may be required.

For hosts, the mapping depends on the existing syntax for host names which is a subset of the usual text representation for domain names, together with RR formats for describing host addresses, etc. Because we need a reliable inverse mapping from address to host name, a special mapping for addresses into the IN-ADDR.ARPA domain is also defined.

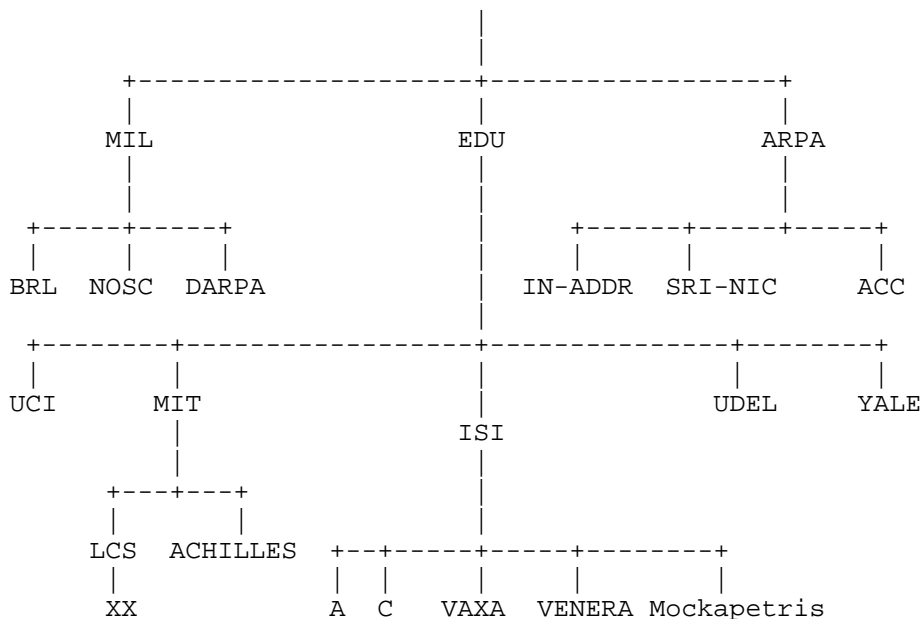
For mailboxes, the mapping is slightly more complex. The usual mail address <local-part>@<mail-domain> is mapped into a domain name by converting <local-part> into a single label (regardless of dots it contains), converting <mail-domain> into a domain name using the usual text format for domain names (dots denote label breaks), and concatenating the two to form a single domain name. Thus the mailbox

HOSTMASTER@SRI-NIC.ARPA is represented as a domain name by HOSTMASTER.SRI-NIC.ARPA. An appreciation for the reasons behind this design also must take into account the scheme for mail exchanges [RFC-974].

The typical user is not concerned with defining these rules, but should understand that they usually are the result of numerous compromises between desires for upward compatibility with old usage, interactions between different object definitions, and the inevitable urge to add new features when defining the rules. The way the DNS is used to support some object is often more crucial than the restrictions inherent in the DNS.

3.4. Example name space

The following figure shows a part of the current domain name space, and is used in many examples in this RFC. Note that the tree is a very small subset of the actual name space.



In this example, the root domain has three immediate subdomains: MIL, EDU, and ARPA. The LCS.MIT.EDU domain has one immediate subdomain named XX.LCS.MIT.EDU. All of the leaves are also domains.

3.5. Preferred name syntax

The DNS specifications attempt to be as general as possible in the rules

for constructing domain names. The idea is that the name of any existing object can be expressed as a domain name with minimal changes. However, when assigning a domain name for an object, the prudent user will select a name which satisfies both the rules of the domain system and any existing rules for the object, whether these rules are published or implied by existing programs.

For example, when naming a mail domain, the user should satisfy both the rules of this memo and those in RFC-822. When creating a new host name, the old rules for HOSTS.TXT should be followed. This avoids problems when old software is converted to use domain names.

The following syntax will result in fewer problems with many applications that use domain names (e.g., mail, TELNET).

<domain> ::= <subdomain> | " "

<subdomain> ::= <label> | <subdomain> "." <label>

<label> ::= <letter> [[<ldh-str>] <let-dig>]

<ldh-str> ::= <let-dig-hyp> | <let-dig-hyp> <ldh-str>

<let-dig-hyp> ::= <let-dig> | "-"

<let-dig> ::= <letter> | <digit>

<letter> ::= any one of the 52 alphabetic characters A through Z in upper case and a through z in lower case

<digit> ::= any one of the ten digits 0 through 9

Note that while upper and lower case letters are allowed in domain names, no significance is attached to the case. That is, two names with the same spelling but different case are to be treated as if identical.

The labels must follow the rules for ARPANET host names. They must start with a letter, end with a letter or digit, and have as interior characters only letters, digits, and hyphen. There are also some restrictions on the length. Labels must be 63 characters or less.

For example, the following strings identify hosts in the Internet:

A.ISI.EDU XX.LCS.MIT.EDU SRI-NIC.ARPA

3.6. Resource Records

A domain name identifies a node. Each node has a set of resource

information, which may be empty. The set of resource information associated with a particular name is composed of separate resource records (RRs). The order of RRs in a set is not significant, and need not be preserved by name servers, resolvers, or other parts of the DNS.

When we talk about a specific RR, we assume it has the following:

owner which is the domain name where the RR is found.

type which is an encoded 16 bit value that specifies the type of the resource in this resource record. Types refer to abstract resources.

This memo uses the following types:

A a host address

CNAME identifies the canonical name of an alias

HINFO identifies the CPU and OS used by a host

MX identifies a mail exchange for the domain. See [RFC-974 for details.

NS the authoritative name server for the domain

PTR a pointer to another part of the domain name space

SOA identifies the start of a zone of authority]

class which is an encoded 16 bit value which identifies a protocol family or instance of a protocol.

This memo uses the following classes:

IN the Internet system

CH the Chaos system

TTL which is the time to live of the RR. This field is a 32 bit integer in units of seconds, and is primarily used by resolvers when they cache RRs. The TTL describes how long a RR can be cached before it should be discarded.

RDATA	which is the type and sometimes class dependent data which describes the resource:
A	For the IN class, a 32 bit IP address For the CH class, a domain name followed by a 16 bit octal Chaos address.
CNAME	a domain name.
MX	a 16 bit preference value (lower is better) followed by a host name willing to act as a mail exchange for the owner domain.
NS	a host name.
PTR	a domain name.
SOA	several fields.

The owner name is often implicit, rather than forming an integral part of the RR. For example, many name servers internally form tree or hash structures for the name space, and chain RRs off nodes. The remaining RR parts are the fixed header (type, class, TTL) which is consistent for all RRs, and a variable part (RDATA) that fits the needs of the resource being described.

The meaning of the TTL field is a time limit on how long an RR can be kept in a cache. This limit does not apply to authoritative data in zones; it is also timed out, but by the refreshing policies for the zone. The TTL is assigned by the administrator for the zone where the data originates. While short TTLs can be used to minimize caching, and a zero TTL prohibits caching, the realities of Internet performance suggest that these times should be on the order of days for the typical host. If a change can be anticipated, the TTL can be reduced prior to the change to minimize inconsistency during the change, and then increased back to its former value following the change.

The data in the RDATA section of RRs is carried as a combination of binary strings and domain names. The domain names are frequently used as "pointers" to other data in the DNS.

3.6.1. Textual expression of RRs

RRs are represented in binary form in the packets of the DNS protocol, and are usually represented in highly encoded form when stored in a name server or resolver. In this memo, we adopt a style similar to that used

in master files in order to show the contents of RRs. In this format, most RRs are shown on a single line, although continuation lines are possible using parentheses.

The start of the line gives the owner of the RR. If a line begins with a blank, then the owner is assumed to be the same as that of the previous RR. Blank lines are often included for readability.

Following the owner, we list the TTL, type, and class of the RR. Class and type use the mnemonics defined above, and TTL is an integer before the type field. In order to avoid ambiguity in parsing, type and class mnemonics are disjoint, TTLs are integers, and the type mnemonic is always last. The IN class and TTL values are often omitted from examples in the interests of clarity.

The resource data or RDATA section of the RR are given using knowledge of the typical representation for the data.

For example, we might show the RRs carried in a message as:

```

ISI.EDU.      MX      10 VENERA.ISI.EDU.
              MX      10 VAXA.ISI.EDU.
VENERA.ISI.EDU. A      128.9.0.32
              A      10.1.0.52
VAXA.ISI.EDU.  A      10.2.0.27
              A      128.9.0.33

```

The MX RRs have an RDATA section which consists of a 16 bit number followed by a domain name. The address RRs use a standard IP address format to contain a 32 bit internet address.

This example shows six RRs, with two RRs at each of three domain names.

Similarly we might see:

```

XX.LCS.MIT.EDU. IN      A      10.0.0.44
                  CH      A      MIT.EDU. 2420

```

This example shows two addresses for XX.LCS.MIT.EDU, each of a different class.

3.6.2. Aliases and canonical names

In existing systems, hosts and other resources often have several names that identify the same resource. For example, the names C.ISI.EDU and USC-ISIC.ARPA both identify the same host. Similarly, in the case of mailboxes, many organizations provide many names that actually go to the same mailbox; for example Mockapetris@C.ISI.EDU, Mockapetris@B.ISI.EDU,

and PVM@ISI.EDU all go to the same mailbox (although the mechanism behind this is somewhat complicated).

Most of these systems have a notion that one of the equivalent set of names is the canonical or primary name and all others are aliases.

The domain system provides such a feature using the canonical name (CNAME) RR. A CNAME RR identifies its owner name as an alias, and specifies the corresponding canonical name in the RDATA section of the RR. If a CNAME RR is present at a node, no other data should be present; this ensures that the data for a canonical name and its aliases cannot be different. This rule also insures that a cached CNAME can be used without checking with an authoritative server for other RR types.

CNAME RRs cause special action in DNS software. When a name server fails to find a desired RR in the resource set associated with the domain name, it checks to see if the resource set consists of a CNAME record with a matching class. If so, the name server includes the CNAME record in the response and restarts the query at the domain name specified in the data field of the CNAME record. The one exception to this rule is that queries which match the CNAME type are not restarted.

For example, suppose a name server was processing a query with for USC-ISIC.ARPA, asking for type A information, and had the following resource records:

```
USC-ISIC.ARPA  IN      CNAME  C.ISI.EDU
C.ISI.EDU     IN      A      10.0.0.52
```

Both of these RRs would be returned in the response to the type A query, while a type CNAME or * query should return just the CNAME.

Domain names in RRs which point at another name should always point at the primary name and not the alias. This avoids extra indirections in accessing information. For example, the address to name RR for the above host should be:

```
52.0.0.10.IN-ADDR.ARPA  IN      PTR      C.ISI.EDU
```

rather than pointing at USC-ISIC.ARPA. Of course, by the robustness principle, domain software should not fail when presented with CNAME chains or loops; CNAME chains should be followed and CNAME loops signalled as an error.

3.7. Queries

Queries are messages which may be sent to a name server to provoke a

response. In the Internet, queries are carried in UDP datagrams or over TCP connections. The response by the name server either answers the question posed in the query, refers the requester to another set of name servers, or signals some error condition.

In general, the user does not generate queries directly, but instead makes a request to a resolver which in turn sends one or more queries to name servers and deals with the error conditions and referrals that may result. Of course, the possible questions which can be asked in a query does shape the kind of service a resolver can provide.

DNS queries and responses are carried in a standard message format. The message format has a header containing a number of fixed fields which are always present, and four sections which carry query parameters and RRs.

The most important field in the header is a four bit field called an opcode which separates different queries. Of the possible 16 values, one (standard query) is part of the official protocol, two (inverse query and status query) are options, one (completion) is obsolete, and the rest are unassigned.

The four sections are:

Question	Carries the query name and other query parameters.
Answer	Carries RRs which directly answer the query.
Authority	Carries RRs which describe other authoritative servers. May optionally carry the SOA RR for the authoritative data in the answer section.
Additional	Carries RRs which may be helpful in using the RRs in the other sections.

Note that the content, but not the format, of these sections varies with header opcode.

3.7.1. Standard queries

A standard query specifies a target domain name (QNAME), query type (QTYPE), and query class (QCLASS) and asks for RRs which match. This type of query makes up such a vast majority of DNS queries that we use the term "query" to mean standard query unless otherwise specified. The QTYPE and QCLASS fields are each 16 bits long, and are a superset of defined types and classes.

The QTYPE field may contain:

<any type> matches just that type. (e.g., A, PTR).
 AXFR special zone transfer QTYPE.
 MAILB matches all mail box related RRs (e.g. MB and MG).
 * matches all RR types.

The QCLASS field may contain:

<any class> matches just that class (e.g., IN, CH).
 * matches aLL RR classes.

Using the query domain name, QTYPE, and QCLASS, the name server looks for matching RRs. In addition to relevant records, the name server may return RRs that point toward a name server that has the desired information or RRs that are expected to be useful in interpreting the relevant RRs. For example, a name server that doesn't have the requested information may know a name server that does; a name server that returns a domain name in a relevant RR may also return the RR that binds that domain name to an address.

For example, a mailer trying to send mail to Mockapetris@ISI.EDU might ask the resolver for mail information about ISI.EDU, resulting in a query for QNAME=ISI.EDU, QTYPE=MX, QCLASS=IN. The response's answer section would be:

```
ISI.EDU.      MX      10 VENERA.ISI.EDU.
              MX      10 VAXA.ISI.EDU.
```

while the additional section might be:

```
VAXA.ISI.EDU. A      10.2.0.27
              A      128.9.0.33
VENERA.ISI.EDU. A    10.1.0.52
              A      128.9.0.32
```

Because the server assumes that if the requester wants mail exchange information, it will probably want the addresses of the mail exchanges soon afterward.

Note that the QCLASS=* construct requires special interpretation regarding authority. Since a particular name server may not know all of the classes available in the domain system, it can never know if it is authoritative for all classes. Hence responses to QCLASS=* queries can

never be authoritative.

3.7.2. Inverse queries (Optional)

Name servers may also support inverse queries that map a particular resource to a domain name or domain names that have that resource. For example, while a standard query might map a domain name to a SOA RR, the corresponding inverse query might map the SOA RR back to the domain name.

Implementation of this service is optional in a name server, but all name servers must at least be able to understand an inverse query message and return a not-implemented error response.

The domain system cannot guarantee the completeness or uniqueness of inverse queries because the domain system is organized by domain name rather than by host address or any other resource type. Inverse queries are primarily useful for debugging and database maintenance activities.

Inverse queries may not return the proper TTL, and do not indicate cases where the identified RR is one of a set (for example, one address for a host having multiple addresses). Therefore, the RRs returned in inverse queries should never be cached.

Inverse queries are NOT an acceptable method for mapping host addresses to host names; use the IN-ADDR.ARPA domain instead.

A detailed discussion of inverse queries is contained in [RFC-1035].

3.8. Status queries (Experimental)

To be defined.

3.9. Completion queries (Obsolete)

The optional completion services described in RFCs 882 and 883 have been deleted. Redesigned services may become available in the future, or the opcodes may be reclaimed for other use.

4. NAME SERVERS

4.1. Introduction

Name servers are the repositories of information that make up the domain database. The database is divided up into sections called zones, which are distributed among the name servers. While name servers can have several optional functions and sources of data, the essential task of a name server is to answer queries using data in its zones. By design,

name servers can answer queries in a simple manner; the response can always be generated using only local data, and either contains the answer to the question or a referral to other name servers "closer" to the desired information.

A given zone will be available from several name servers to insure its availability in spite of host or communication link failure. By administrative fiat, we require every zone to be available on at least two servers, and many zones have more redundancy than that.

A given name server will typically support one or more zones, but this gives it authoritative information about only a small section of the domain tree. It may also have some cached non-authoritative data about other parts of the tree. The name server marks its responses to queries so that the requester can tell whether the response comes from authoritative data or not.

4.2. How the database is divided into zones

The domain database is partitioned in two ways: by class, and by "cuts" made in the name space between nodes.

The class partition is simple. The database for any class is organized, delegated, and maintained separately from all other classes. Since, by convention, the name spaces are the same for all classes, the separate classes can be thought of as an array of parallel namespace trees. Note that the data attached to nodes will be different for these different parallel classes. The most common reasons for creating a new class are the necessity for a new data format for existing types or a desire for a separately managed version of the existing name space.

Within a class, "cuts" in the name space can be made between any two adjacent nodes. After all cuts are made, each group of connected name space is a separate zone. The zone is said to be authoritative for all names in the connected region. Note that the "cuts" in the name space may be in different places for different classes, the name servers may be different, etc.

These rules mean that every zone has at least one node, and hence domain name, for which it is authoritative, and all of the nodes in a particular zone are connected. Given, the tree structure, every zone has a highest node which is closer to the root than any other node in the zone. The name of this node is often used to identify the zone.

It would be possible, though not particularly useful, to partition the name space so that each domain name was in a separate zone or so that all nodes were in a single zone. Instead, the database is partitioned at points where a particular organization wants to take over control of

a subtree. Once an organization controls its own zone it can unilaterally change the data in the zone, grow new tree sections connected to the zone, delete existing nodes, or delegate new subzones under its zone.

If the organization has substructure, it may want to make further internal partitions to achieve nested delegations of name space control. In some cases, such divisions are made purely to make database maintenance more convenient.

4.2.1. Technical considerations

The data that describes a zone has four major parts:

- Authoritative data for all nodes within the zone.
- Data that defines the top node of the zone (can be thought of as part of the authoritative data).
- Data that describes delegated subzones, i.e., cuts around the bottom of the zone.
- Data that allows access to name servers for subzones (sometimes called "glue" data).

All of this data is expressed in the form of RRs, so a zone can be completely described in terms of a set of RRs. Whole zones can be transferred between name servers by transferring the RRs, either carried in a series of messages or by FTPing a master file which is a textual representation.

The authoritative data for a zone is simply all of the RRs attached to all of the nodes from the top node of the zone down to leaf nodes or nodes above cuts around the bottom edge of the zone.

Though logically part of the authoritative data, the RRs that describe the top node of the zone are especially important to the zone's management. These RRs are of two types: name server RRs that list, one per RR, all of the servers for the zone, and a single SOA RR that describes zone management parameters.

The RRs that describe cuts around the bottom of the zone are NS RRs that name the servers for the subzones. Since the cuts are between nodes, these RRs are NOT part of the authoritative data of the zone, and should be exactly the same as the corresponding RRs in the top node of the subzone. Since name servers are always associated with zone boundaries, NS RRs are only found at nodes which are the top node of some zone. In the data that makes up a zone, NS RRs are found at the top node of the

zone (and are authoritative) and at cuts around the bottom of the zone (where they are not authoritative), but never in between.

One of the goals of the zone structure is that any zone have all the data required to set up communications with the name servers for any subzones. That is, parent zones have all the information needed to access servers for their children zones. The NS RRs that name the servers for subzones are often not enough for this task since they name the servers, but do not give their addresses. In particular, if the name of the name server is itself in the subzone, we could be faced with the situation where the NS RRs tell us that in order to learn a name server's address, we should contact the server using the address we wish to learn. To fix this problem, a zone contains "glue" RRs which are not part of the authoritative data, and are address RRs for the servers. These RRs are only necessary if the name server's name is "below" the cut, and are only used as part of a referral response.

4.2.2. Administrative considerations

When some organization wants to control its own domain, the first step is to identify the proper parent zone, and get the parent zone's owners to agree to the delegation of control. While there are no particular technical constraints dealing with where in the tree this can be done, there are some administrative groupings discussed in [RFC-1032] which deal with top level organization, and middle level zones are free to create their own rules. For example, one university might choose to use a single zone, while another might choose to organize by subzones dedicated to individual departments or schools. [RFC-1033] catalogs available DNS software and discusses administration procedures.

Once the proper name for the new subzone is selected, the new owners should be required to demonstrate redundant name server support. Note that there is no requirement that the servers for a zone reside in a host which has a name in that domain. In many cases, a zone will be more accessible to the internet at large if its servers are widely distributed rather than being within the physical facilities controlled by the same organization that manages the zone. For example, in the current DNS, one of the name servers for the United Kingdom, or UK domain, is found in the US. This allows US hosts to get UK data without using limited transatlantic bandwidth.

As the last installation step, the delegation NS RRs and glue RRs necessary to make the delegation effective should be added to the parent zone. The administrators of both zones should insure that the NS and glue RRs which mark both sides of the cut are consistent and remain so.

4.3. Name server internals

4.3.1. Queries and responses

The principal activity of name servers is to answer standard queries. Both the query and its response are carried in a standard message format which is described in [RFC-1035]. The query contains a QTYPE, QCLASS, and QNAME, which describe the types and classes of desired information and the name of interest.

The way that the name server answers the query depends upon whether it is operating in recursive mode or not:

- The simplest mode for the server is non-recursive, since it can answer queries using only local information: the response contains an error, the answer, or a referral to some other server "closer" to the answer. All name servers must implement non-recursive queries.
- The simplest mode for the client is recursive, since in this mode the name server acts in the role of a resolver and returns either an error or the answer, but never referrals. This service is optional in a name server, and the name server may also choose to restrict the clients which can use recursive mode.

Recursive service is helpful in several situations:

- a relatively simple requester that lacks the ability to use anything other than a direct answer to the question.
- a request that needs to cross protocol or other boundaries and can be sent to a server which can act as intermediary.
- a network where we want to concentrate the cache rather than having a separate cache for each client.

Non-recursive service is appropriate if the requester is capable of pursuing referrals and interested in information which will aid future requests.

The use of recursive mode is limited to cases where both the client and the name server agree to its use. The agreement is negotiated through the use of two bits in query and response messages:

- The recursion available, or RA bit, is set or cleared by a name server in all responses. The bit is true if the name server is willing to provide recursive service for the client, regardless of whether the client requested recursive service. That is, RA signals availability rather than use.

- Queries contain a bit called recursion desired or RD. This bit specifies whether the requester wants recursive service for this query. Clients may request recursive service from any name server, though they should depend upon receiving it only from servers which have previously sent an RA, or servers which have agreed to provide service through private agreement or some other means outside of the DNS protocol.

The recursive mode occurs when a query with RD set arrives at a server which is willing to provide recursive service; the client can verify that recursive mode was used by checking that both RA and RD are set in the reply. Note that the name server should never perform recursive service unless asked via RD, since this interferes with trouble shooting of name servers and their databases.

If recursive service is requested and available, the recursive response to a query will be one of the following:

- The answer to the query, possibly preface by one or more CNAME RRs that specify aliases encountered on the way to an answer.
- A name error indicating that the name does not exist. This may include CNAME RRs that indicate that the original query name was an alias for a name which does not exist.
- A temporary error indication.

If recursive service is not requested or is not available, the non-recursive response will be one of the following:

- An authoritative name error indicating that the name does not exist.
- A temporary error indication.
- Some combination of:

RRs that answer the question, together with an indication whether the data comes from a zone or is cached.

A referral to name servers which have zones which are closer ancestors to the name than the server sending the reply.

- RRs that the name server thinks will prove useful to the requester.

4.3.2. Algorithm

The actual algorithm used by the name server will depend on the local OS and data structures used to store RRs. The following algorithm assumes that the RRs are organized in several tree structures, one for each zone, and another for the cache:

1. Set or clear the value of recursion available in the response depending on whether the name server is willing to provide recursive service. If recursive service is available and requested via the RD bit in the query, go to step 5, otherwise step 2.
2. Search the available zones for the zone which is the nearest ancestor to QNAME. If such a zone is found, go to step 3, otherwise step 4.
3. Start matching down, label by label, in the zone. The matching process can terminate several ways:

- a. If the whole of QNAME is matched, we have found the node.

If the data at the node is a CNAME, and QTYPE doesn't match CNAME, copy the CNAME RR into the answer section of the response, change QNAME to the canonical name in the CNAME RR, and go back to step 1.

Otherwise, copy all RRs which match QTYPE into the answer section and go to step 6.

- b. If a match would take us out of the authoritative data, we have a referral. This happens when we encounter a node with NS RRs marking cuts along the bottom of a zone.

Copy the NS RRs for the subzone into the authority section of the reply. Put whatever addresses are available into the additional section, using glue RRs if the addresses are not available from authoritative data or the cache. Go to step 4.

- c. If at some label, a match is impossible (i.e., the corresponding label does not exist), look to see if a the "*" label exists.

If the "*" label does not exist, check whether the name we are looking for is the original QNAME in the query

or a name we have followed due to a CNAME. If the name is original, set an authoritative name error in the response and exit. Otherwise just exit.

If the "*" label does exist, match RRs at that node against QTYPE. If any match, copy them into the answer section, but set the owner of the RR to be QNAME, and not the node with the "*" label. Go to step 6.

4. Start matching down in the cache. If QNAME is found in the cache, copy all RRs attached to it that match QTYPE into the answer section. If there was no delegation from authoritative data, look for the best one from the cache, and put it in the authority section. Go to step 6.
5. Using the local resolver or a copy of its algorithm (see resolver section of this memo) to answer the query. Store the results, including any intermediate CNAMEs, in the answer section of the response.
6. Using local data only, attempt to add other RRs which may be useful to the additional section of the query. Exit.

4.3.3. Wildcards

In the previous algorithm, special treatment was given to RRs with owner names starting with the label "*". Such RRs are called wildcards. Wildcard RRs can be thought of as instructions for synthesizing RRs. When the appropriate conditions are met, the name server creates RRs with an owner name equal to the query name and contents taken from the wildcard RRs.

This facility is most often used to create a zone which will be used to forward mail from the Internet to some other mail system. The general idea is that any name in that zone which is presented to server in a query will be assumed to exist, with certain properties, unless explicit evidence exists to the contrary. Note that the use of the term zone here, instead of domain, is intentional; such defaults do not propagate across zone boundaries, although a subzone may choose to achieve that appearance by setting up similar defaults.

The contents of the wildcard RRs follows the usual rules and formats for RRs. The wildcards in the zone have an owner name that controls the query names they will match. The owner name of the wildcard RRs is of the form "*.<anydomain>", where <anydomain> is any domain name. <anydomain> should not contain other * labels, and should be in the authoritative data of the zone. The wildcards potentially apply to descendants of <anydomain>, but not to <anydomain> itself. Another way

to look at this is that the "*" label always matches at least one whole label and sometimes more, but always whole labels.

Wildcard RRs do not apply:

- When the query is in another zone. That is, delegation cancels the wildcard defaults.
- When the query name or a name between the wildcard domain and the query name is known to exist. For example, if a wildcard RR has an owner name of "*.X", and the zone also contains RRs attached to B.X, the wildcards would apply to queries for name Z.X (presuming there is no explicit information for Z.X), but not to B.X, A.B.X, or X.

A * label appearing in a query name has no special effect, but can be used to test for wildcards in an authoritative zone; such a query is the only way to get a response containing RRs with an owner name with * in it. The result of such a query should not be cached.

Note that the contents of the wildcard RRs are not modified when used to synthesize RRs.

To illustrate the use of wildcard RRs, suppose a large company with a large, non-IP/TCP, network wanted to create a mail gateway. If the company was called X.COM, and IP/TCP capable gateway machine was called A.X.COM, the following RRs might be entered into the COM zone:

X.COM	MX	10	A.X.COM
*.X.COM	MX	10	A.X.COM
A.X.COM	A	1.2.3.4	
A.X.COM	MX	10	A.X.COM
*.A.X.COM	MX	10	A.X.COM

This would cause any MX query for any domain name ending in X.COM to return an MX RR pointing at A.X.COM. Two wildcard RRs are required since the effect of the wildcard at *.X.COM is inhibited in the A.X.COM subtree by the explicit data for A.X.COM. Note also that the explicit MX data at X.COM and A.X.COM is required, and that none of the RRs above would match a query name of XX.COM.

4.3.4. Negative response caching (Optional)

The DNS provides an optional service which allows name servers to distribute, and resolvers to cache, negative results with TTLs. For

example, a name server can distribute a TTL along with a name error indication, and a resolver receiving such information is allowed to assume that the name does not exist during the TTL period without consulting authoritative data. Similarly, a resolver can make a query with a QTYPE which matches multiple types, and cache the fact that some of the types are not present.

This feature can be particularly important in a system which implements naming shorthands that use search lists because a popular shorthand, which happens to require a suffix toward the end of the search list, will generate multiple name errors whenever it is used.

The method is that a name server may add an SOA RR to the additional section of a response when that response is authoritative. The SOA must be that of the zone which was the source of the authoritative data in the answer section, or name error if applicable. The MINIMUM field of the SOA controls the length of time that the negative result may be cached.

Note that in some circumstances, the answer section may contain multiple owner names. In this case, the SOA mechanism should only be used for the data which matches QNAME, which is the only authoritative data in this section.

Name servers and resolvers should never attempt to add SOAs to the additional section of a non-authoritative response, or attempt to infer results which are not directly stated in an authoritative response. There are several reasons for this, including: cached information isn't usually enough to match up RRs and their zone names, SOA RRs may be cached due to direct SOA queries, and name servers are not required to output the SOAs in the authority section.

This feature is optional, although a refined version is expected to become part of the standard protocol in the future. Name servers are not required to add the SOA RRs in all authoritative responses, nor are resolvers required to cache negative results. Both are recommended. All resolvers and recursive name servers are required to at least be able to ignore the SOA RR when it is present in a response.

Some experiments have also been proposed which will use this feature. The idea is that if cached data is known to come from a particular zone, and if an authoritative copy of the zone's SOA is obtained, and if the zone's SERIAL has not changed since the data was cached, then the TTL of the cached data can be reset to the zone MINIMUM value if it is smaller. This usage is mentioned for planning purposes only, and is not recommended as yet.

4.3.5. Zone maintenance and transfers

Part of the job of a zone administrator is to maintain the zones at all of the name servers which are authoritative for the zone. When the inevitable changes are made, they must be distributed to all of the name servers. While this distribution can be accomplished using FTP or some other ad hoc procedure, the preferred method is the zone transfer part of the DNS protocol.

The general model of automatic zone transfer or refreshing is that one of the name servers is the master or primary for the zone. Changes are coordinated at the primary, typically by editing a master file for the zone. After editing, the administrator signals the master server to load the new zone. The other non-master or secondary servers for the zone periodically check for changes (at a selectable interval) and obtain new zone copies when changes have been made.

To detect changes, secondaries just check the SERIAL field of the SOA for the zone. In addition to whatever other changes are made, the SERIAL field in the SOA of the zone is always advanced whenever any change is made to the zone. The advancing can be a simple increment, or could be based on the write date and time of the master file, etc. The purpose is to make it possible to determine which of two copies of a zone is more recent by comparing serial numbers. Serial number advances and comparisons use sequence space arithmetic, so there is a theoretic limit on how fast a zone can be updated, basically that old copies must die out before the serial number covers half of its 32 bit range. In practice, the only concern is that the compare operation deals properly with comparisons around the boundary between the most positive and most negative 32 bit numbers.

The periodic polling of the secondary servers is controlled by parameters in the SOA RR for the zone, which set the minimum acceptable polling intervals. The parameters are called REFRESH, RETRY, and EXPIRE. Whenever a new zone is loaded in a secondary, the secondary waits REFRESH seconds before checking with the primary for a new serial. If this check cannot be completed, new checks are started every RETRY seconds. The check is a simple query to the primary for the SOA RR of the zone. If the serial field in the secondary's zone copy is equal to the serial returned by the primary, then no changes have occurred, and the REFRESH interval wait is restarted. If the secondary finds it impossible to perform a serial check for the EXPIRE interval, it must assume that its copy of the zone is obsolete and discard it.

When the poll shows that the zone has changed, then the secondary server must request a zone transfer via an AXFR request for the zone. The AXFR may cause an error, such as refused, but normally is answered by a sequence of response messages. The first and last messages must contain

the data for the top authoritative node of the zone. Intermediate messages carry all of the other RRs from the zone, including both authoritative and non-authoritative RRs. The stream of messages allows the secondary to construct a copy of the zone. Because accuracy is essential, TCP or some other reliable protocol must be used for AXFR requests.

Each secondary server is required to perform the following operations against the master, but may also optionally perform these operations against other secondary servers. This strategy can improve the transfer process when the primary is unavailable due to host downtime or network problems, or when a secondary server has better network access to an "intermediate" secondary than to the primary.

5. RESOLVERS

5.1. Introduction

Resolvers are programs that interface user programs to domain name servers. In the simplest case, a resolver receives a request from a user program (e.g., mail programs, TELNET, FTP) in the form of a subroutine call, system call etc., and returns the desired information in a form compatible with the local host's data formats.

The resolver is located on the same machine as the program that requests the resolver's services, but it may need to consult name servers on other hosts. Because a resolver may need to consult several name servers, or may have the requested information in a local cache, the amount of time that a resolver will take to complete can vary quite a bit, from milliseconds to several seconds.

A very important goal of the resolver is to eliminate network delay and name server load from most requests by answering them from its cache of prior results. It follows that caches which are shared by multiple processes, users, machines, etc., are more efficient than non-shared caches.

5.2. Client-resolver interface

5.2.1. Typical functions

The client interface to the resolver is influenced by the local host's conventions, but the typical resolver-client interface has three functions:

1. Host name to host address translation.

This function is often defined to mimic a previous HOSTS.TXT

based function. Given a character string, the caller wants one or more 32 bit IP addresses. Under the DNS, it translates into a request for type A RRs. Since the DNS does not preserve the order of RRs, this function may choose to sort the returned addresses or select the "best" address if the service returns only one choice to the client. Note that a multiple address return is recommended, but a single address may be the only way to emulate prior HOSTS.TXT services.

2. Host address to host name translation

This function will often follow the form of previous functions. Given a 32 bit IP address, the caller wants a character string. The octets of the IP address are reversed, used as name components, and suffixed with "IN-ADDR.ARPA". A type PTR query is used to get the RR with the primary name of the host. For example, a request for the host name corresponding to IP address 1.2.3.4 looks for PTR RRs for domain name "4.3.2.1.IN-ADDR.ARPA".

3. General lookup function

This function retrieves arbitrary information from the DNS, and has no counterpart in previous systems. The caller supplies a QNAME, QTYPE, and QCLASS, and wants all of the matching RRs. This function will often use the DNS format for all RR data instead of the local host's, and returns all RR content (e.g., TTL) instead of a processed form with local quoting conventions.

When the resolver performs the indicated function, it usually has one of the following results to pass back to the client:

- One or more RRs giving the requested data.

In this case the resolver returns the answer in the appropriate format.

- A name error (NE).

This happens when the referenced name does not exist. For example, a user may have mistyped a host name.

- A data not found error.

This happens when the referenced name exists, but data of the appropriate type does not. For example, a host address

function applied to a mailbox name would return this error since the name exists, but no address RR is present.

It is important to note that the functions for translating between host names and addresses may combine the "name error" and "data not found" error conditions into a single type of error return, but the general function should not. One reason for this is that applications may ask first for one type of information about a name followed by a second request to the same name for some other type of information; if the two errors are combined, then useless queries may slow the application.

5.2.2. Aliases

While attempting to resolve a particular request, the resolver may find that the name in question is an alias. For example, the resolver might find that the name given for host name to address translation is an alias when it finds the CNAME RR. If possible, the alias condition should be signalled back from the resolver to the client.

In most cases a resolver simply restarts the query at the new name when it encounters a CNAME. However, when performing the general function, the resolver should not pursue aliases when the CNAME RR matches the query type. This allows queries which ask whether an alias is present. For example, if the query type is CNAME, the user is interested in the CNAME RR itself, and not the RRs at the name it points to.

Several special conditions can occur with aliases. Multiple levels of aliases should be avoided due to their lack of efficiency, but should not be signalled as an error. Alias loops and aliases which point to non-existent names should be caught and an error condition passed back to the client.

5.2.3. Temporary failures

In a less than perfect world, all resolvers will occasionally be unable to resolve a particular request. This condition can be caused by a resolver which becomes separated from the rest of the network due to a link failure or gateway problem, or less often by coincident failure or unavailability of all servers for a particular domain.

It is essential that this sort of condition should not be signalled as a name or data not present error to applications. This sort of behavior is annoying to humans, and can wreak havoc when mail systems use the DNS.

While in some cases it is possible to deal with such a temporary problem by blocking the request indefinitely, this is usually not a good choice, particularly when the client is a server process that could move on to

other tasks. The recommended solution is to always have temporary failure as one of the possible results of a resolver function, even though this may make emulation of existing HOSTS.TXT functions more difficult.

5.3. Resolver internals

Every resolver implementation uses slightly different algorithms, and typically spends much more logic dealing with errors of various sorts than typical occurrences. This section outlines a recommended basic strategy for resolver operation, but leaves details to [RFC-1035].

5.3.1. Stub resolvers

One option for implementing a resolver is to move the resolution function out of the local machine and into a name server which supports recursive queries. This can provide an easy method of providing domain service in a PC which lacks the resources to perform the resolver function, or can centralize the cache for a whole local network or organization.

All that the remaining stub needs is a list of name server addresses that will perform the recursive requests. This type of resolver presumably needs the information in a configuration file, since it probably lacks the sophistication to locate it in the domain database. The user also needs to verify that the listed servers will perform the recursive service; a name server is free to refuse to perform recursive services for any or all clients. The user should consult the local system administrator to find name servers willing to perform the service.

This type of service suffers from some drawbacks. Since the recursive requests may take an arbitrary amount of time to perform, the stub may have difficulty optimizing retransmission intervals to deal with both lost UDP packets and dead servers; the name server can be easily overloaded by too zealous a stub if it interprets retransmissions as new requests. Use of TCP may be an answer, but TCP may well place burdens on the host's capabilities which are similar to those of a real resolver.

5.3.2. Resources

In addition to its own resources, the resolver may also have shared access to zones maintained by a local name server. This gives the resolver the advantage of more rapid access, but the resolver must be careful to never let cached information override zone data. In this discussion the term "local information" is meant to mean the union of the cache and such shared zones, with the understanding that

authoritative data is always used in preference to cached data when both are present.

The following resolver algorithm assumes that all functions have been converted to a general lookup function, and uses the following data structures to represent the state of a request in progress in the resolver:

SNAME the domain name we are searching for.

STYPE the QTYPE of the search request.

SCLASS the QCLASS of the search request.

SLIST a structure which describes the name servers and the zone which the resolver is currently trying to query. This structure keeps track of the resolver's current best guess about which name servers hold the desired information; it is updated when arriving information changes the guess. This structure includes the equivalent of a zone name, the known name servers for the zone, the known addresses for the name servers, and history information which can be used to suggest which server is likely to be the best one to try next. The zone name equivalent is a match count of the number of labels from the root down which SNAME has in common with the zone being queried; this is used as a measure of how "close" the resolver is to SNAME.

SBELT a "safety belt" structure of the same form as SLIST, which is initialized from a configuration file, and lists servers which should be used when the resolver doesn't have any local information to guide name server selection. The match count will be -1 to indicate that no labels are known to match.

CACHE A structure which stores the results from previous responses. Since resolvers are responsible for discarding old RRs whose TTL has expired, most implementations convert the interval specified in arriving RRs to some sort of absolute time when the RR is stored in the cache. Instead of counting the TTLs down individually, the resolver just ignores or discards old RRs when it runs across them in the course of a search, or discards them during periodic sweeps to reclaim the memory consumed by old RRs.

5.3.3. Algorithm

The top level algorithm has four steps:

1. See if the answer is in local information, and if so return it to the client.
2. Find the best servers to ask.
3. Send them queries until one returns a response.
4. Analyze the response, either:
 - a. if the response answers the question or contains a name error, cache the data as well as returning it back to the client.
 - b. if the response contains a better delegation to other servers, cache the delegation information, and go to step 2.
 - c. if the response shows a CNAME and that is not the answer itself, cache the CNAME, change the SNAME to the canonical name in the CNAME RR and go to step 1.
 - d. if the response shows a servers failure or other bizarre contents, delete the server from the SLIST and go back to step 3.

Step 1 searches the cache for the desired data. If the data is in the cache, it is assumed to be good enough for normal use. Some resolvers have an option at the user interface which will force the resolver to ignore the cached data and consult with an authoritative server. This is not recommended as the default. If the resolver has direct access to a name server's zones, it should check to see if the desired data is present in authoritative form, and if so, use the authoritative data in preference to cached data.

Step 2 looks for a name server to ask for the required data. The general strategy is to look for locally-available name server RRs, starting at SNAME, then the parent domain name of SNAME, the grandparent, and so on toward the root. Thus if SNAME were Mockapetris.ISI.EDU, this step would look for NS RRs for Mockapetris.ISI.EDU, then ISI.EDU, then EDU, and then . (the root). These NS RRs list the names of hosts for a zone at or above SNAME. Copy the names into SLIST. Set up their addresses using local data. It may be the case that the addresses are not available. The resolver has many choices here; the best is to start parallel resolver processes looking

for the addresses while continuing onward with the addresses which are available. Obviously, the design choices and options are complicated and a function of the local host's capabilities. The recommended priorities for the resolver designer are:

1. Bound the amount of work (packets sent, parallel processes started) so that a request can't get into an infinite loop or start off a chain reaction of requests or queries with other implementations EVEN IF SOMEONE HAS INCORRECTLY CONFIGURED SOME DATA.
2. Get back an answer if at all possible.
3. Avoid unnecessary transmissions.
4. Get the answer as quickly as possible.

If the search for NS RRs fails, then the resolver initializes SLIST from the safety belt SBELT. The basic idea is that when the resolver has no idea what servers to ask, it should use information from a configuration file that lists several servers which are expected to be helpful. Although there are special situations, the usual choice is two of the root servers and two of the servers for the host's domain. The reason for two of each is for redundancy. The root servers will provide eventual access to all of the domain space. The two local servers will allow the resolver to continue to resolve local names if the local network becomes isolated from the internet due to gateway or link failure.

In addition to the names and addresses of the servers, the SLIST data structure can be sorted to use the best servers first, and to insure that all addresses of all servers are used in a round-robin manner. The sorting can be a simple function of preferring addresses on the local network over others, or may involve statistics from past events, such as previous response times and batting averages.

Step 3 sends out queries until a response is received. The strategy is to cycle around all of the addresses for all of the servers with a timeout between each transmission. In practice it is important to use all addresses of a multihomed host, and too aggressive a retransmission policy actually slows response when used by multiple resolvers contending for the same name server and even occasionally for a single resolver. SLIST typically contains data values to control the timeouts and keep track of previous transmissions.

Step 4 involves analyzing responses. The resolver should be highly paranoid in its parsing of responses. It should also check that the response matches the query it sent using the ID field in the response.

The ideal answer is one from a server authoritative for the query which either gives the required data or a name error. The data is passed back to the user and entered in the cache for future use if its TTL is greater than zero.

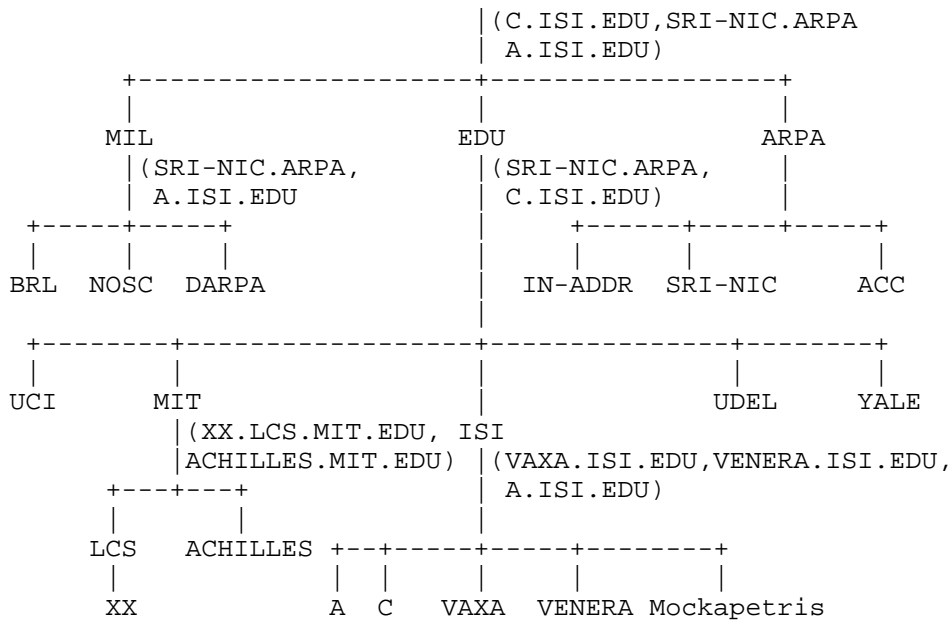
If the response shows a delegation, the resolver should check to see that the delegation is "closer" to the answer than the servers in SLIST are. This can be done by comparing the match count in SLIST with that computed from SNAME and the NS RRs in the delegation. If not, the reply is bogus and should be ignored. If the delegation is valid the NS delegation RRs and any address RRs for the servers should be cached. The name servers are entered in the SLIST, and the search is restarted.

If the response contains a CNAME, the search is restarted at the CNAME unless the response has the data for the canonical name or if the CNAME is the answer itself.

Details and implementation hints can be found in [RFC-1035].

6. A SCENARIO

In our sample domain space, suppose we wanted separate administrative control for the root, MIL, EDU, MIT.EDU and ISI.EDU zones. We might allocate name servers as follows:



In this example, the authoritative name server is shown in parentheses at the point in the domain tree at which it assumes control.

Thus the root name servers are on C.ISI.EDU, SRI-NIC.ARPA, and A.ISI.EDU. The MIL domain is served by SRI-NIC.ARPA and A.ISI.EDU. The EDU domain is served by SRI-NIC.ARPA. and C.ISI.EDU. Note that servers may have zones which are contiguous or disjoint. In this scenario, C.ISI.EDU has contiguous zones at the root and EDU domains. A.ISI.EDU has contiguous zones at the root and MIL domains, but also has a non-contiguous zone at ISI.EDU.

6.1. C.ISI.EDU name server

C.ISI.EDU is a name server for the root, MIL, and EDU domains of the IN class, and would have zones for these domains. The zone data for the root domain might be:

```

.           IN           SOA           SRI-NIC.ARPA. HOSTMASTER.SRI-NIC.ARPA. (
            870611                ;serial
            1800                ;refresh every 30 min
            300                 ;retry every 5 min
            604800              ;expire after a week
            86400)              ;minimum of a day
            NS           A.ISI.EDU.
            NS           C.ISI.EDU.
            NS           SRI-NIC.ARPA.

MIL.       86400         NS           SRI-NIC.ARPA.
            86400         NS           A.ISI.EDU.

EDU.       86400         NS           SRI-NIC.ARPA.
            86400         NS           C.ISI.EDU.

SRI-NIC.ARPA.  A           26.0.0.73
              A           10.0.0.51
              MX          0 SRI-NIC.ARPA.
              HINFO       DEC-2060 TOPS20

ACC.ARPA.    A           26.6.0.65
              HINFO       PDP-11/70 UNIX
              MX          10 ACC.ARPA.

USC-ISIC.ARPA. CNAME     C.ISI.EDU.

73.0.0.26.IN-ADDR.ARPA. PTR     SRI-NIC.ARPA.
65.0.6.26.IN-ADDR.ARPA. PTR     ACC.ARPA.
51.0.0.10.IN-ADDR.ARPA. PTR     SRI-NIC.ARPA.
52.0.0.10.IN-ADDR.ARPA. PTR     C.ISI.EDU.

```

```

103.0.3.26.IN-ADDR.ARPA. PTR    A.ISI.EDU.

A.ISI.EDU. 86400 A      26.3.0.103
C.ISI.EDU. 86400 A      10.0.0.52

```

This data is represented as it would be in a master file. Most RRs are single line entries; the sole exception here is the SOA RR, which uses "(" to start a multi-line RR and ")" to show the end of a multi-line RR. Since the class of all RRs in a zone must be the same, only the first RR in a zone need specify the class. When a name server loads a zone, it forces the TTL of all authoritative RRs to be at least the MINIMUM field of the SOA, here 86400 seconds, or one day. The NS RRs marking delegation of the MIL and EDU domains, together with the glue RRs for the servers host addresses, are not part of the authoritative data in the zone, and hence have explicit TTLs.

Four RRs are attached to the root node: the SOA which describes the root zone and the 3 NS RRs which list the name servers for the root. The data in the SOA RR describes the management of the zone. The zone data is maintained on host SRI-NIC.ARPA, and the responsible party for the zone is HOSTMASTER@SRI-NIC.ARPA. A key item in the SOA is the 86400 second minimum TTL, which means that all authoritative data in the zone has at least that TTL, although higher values may be explicitly specified.

The NS RRs for the MIL and EDU domains mark the boundary between the root zone and the MIL and EDU zones. Note that in this example, the lower zones happen to be supported by name servers which also support the root zone.

The master file for the EDU zone might be stated relative to the origin EDU. The zone data for the EDU domain might be:

```

EDU.  IN SOA SRI-NIC.ARPA. HOSTMASTER.SRI-NIC.ARPA. (
        870729 ;serial
        1800 ;refresh every 30 minutes
        300 ;retry every 5 minutes
        604800 ;expire after a week
        86400 ;minimum of a day
    )
    NS SRI-NIC.ARPA.
    NS C.ISI.EDU.

UCI 172800 NS ICS.UCI
    172800 NS ROME.UCI
ICS.UCI 172800 A 192.5.19.1
ROME.UCI 172800 A 192.5.19.31

```

```
ISI 172800 NS VAXA.ISI
      172800 NS A.ISI
      172800 NS VENERA.ISI.EDU.
VAXA.ISI 172800 A 10.2.0.27
      172800 A 128.9.0.33
VENERA.ISI.EDU. 172800 A 10.1.0.52
      172800 A 128.9.0.32
A.ISI 172800 A 26.3.0.103

UDEL.EDU. 172800 NS LOUIE.UDEL.EDU.
      172800 NS UMN-REI-UC.ARPA.
LOUIE.UDEL.EDU. 172800 A 10.0.0.96
      172800 A 192.5.39.3

YALE.EDU. 172800 NS YALE.ARPA.
YALE.EDU. 172800 NS YALE-BULLDOG.ARPA.

MIT.EDU. 43200 NS XX.LCS.MIT.EDU.
      43200 NS ACHILLES.MIT.EDU.
XX.LCS.MIT.EDU. 43200 A 10.0.0.44
ACHILLES.MIT.EDU. 43200 A 18.72.0.8
```

Note the use of relative names here. The owner name for the ISI.EDU. is stated using a relative name, as are two of the name server RR contents. Relative and absolute domain names may be freely intermixed in a master

6.2. Example standard queries

The following queries and responses illustrate name server behavior. Unless otherwise noted, the queries do not have recursion desired (RD) in the header. Note that the answers to non-recursive queries do depend on the server being asked, but do not depend on the identity of the requester.

6.2.1. QNAME=SRI-NIC.ARPA, QTYPE=A

The query would look like:

```

+-----+
Header   | OPCODE=SQUERY                               |
+-----+
Question | QNAME=SRI-NIC.ARPA., QCLASS=IN, QTYPE=A    |
+-----+
Answer   | <empty>                                     |
+-----+
Authority | <empty>                                     |
+-----+
Additional | <empty>                                     |
+-----+

```

The response from C.ISI.EDU would be:

```

+-----+
Header   | OPCODE=SQUERY, RESPONSE, AA                |
+-----+
Question | QNAME=SRI-NIC.ARPA., QCLASS=IN, QTYPE=A    |
+-----+
Answer   | SRI-NIC.ARPA. 86400 IN A 26.0.0.73         |
          |           86400 IN A 10.0.0.51             |
+-----+
Authority | <empty>                                     |
+-----+
Additional | <empty>                                     |
+-----+

```

The header of the response looks like the header of the query, except that the RESPONSE bit is set, indicating that this message is a response, not a query, and the Authoritative Answer (AA) bit is set indicating that the address RRs in the answer section are from authoritative data. The question section of the response matches the question section of the query.

If the same query was sent to some other server which was not authoritative for SRI-NIC.ARPA, the response might be:

```

+-----+
Header  | OPCODE=SQUERY,RESPONSE |
+-----+
Question | QNAME=SRI-NIC.ARPA., QCLASS=IN, QTYPE=A |
+-----+
Answer  | SRI-NIC.ARPA. 1777 IN A 10.0.0.51 |
        |           1777 IN A 26.0.0.73 |
+-----+
Authority | <empty> |
+-----+
Additional | <empty> |
+-----+

```

This response is different from the previous one in two ways: the header does not have AA set, and the TTLs are different. The inference is that the data did not come from a zone, but from a cache. The difference between the authoritative TTL and the TTL here is due to aging of the data in a cache. The difference in ordering of the RRs in the answer section is not significant.

6.2.2. QNAME=SRI-NIC.ARPA, QTYPE=*

A query similar to the previous one, but using a QTYPE of *, would receive the following response from C.ISI.EDU:

```

+-----+
Header  | OPCODE=SQUERY, RESPONSE, AA |
+-----+
Question | QNAME=SRI-NIC.ARPA., QCLASS=IN, QTYPE=* |
+-----+
Answer  | SRI-NIC.ARPA. 86400 IN  A    26.0.0.73 |
        |                               A    10.0.0.51 |
        |                               MX   0 SRI-NIC.ARPA. |
        |                               HINFO DEC-2060 TOPS20 |
+-----+
Authority | <empty> |
+-----+
Additional | <empty> |
+-----+

```

If a similar query was directed to two name servers which are not authoritative for SRI-NIC.ARPA, the responses might be:

```

+-----+
Header  | OPCODE=SQUERY, RESPONSE |
+-----+
Question | QNAME=SRI-NIC.ARPA., QCLASS=IN, QTYPE=* |
+-----+
Answer  | SRI-NIC.ARPA. 12345 IN      A      26.0.0.73 |
        |                               A      10.0.0.51 |
+-----+
Authority | <empty> |
+-----+
Additional | <empty> |
+-----+

```

and

```

+-----+
Header  | OPCODE=SQUERY, RESPONSE |
+-----+
Question | QNAME=SRI-NIC.ARPA., QCLASS=IN, QTYPE=* |
+-----+
Answer  | SRI-NIC.ARPA. 1290 IN HINFO DEC-2060 TOPS20 |
+-----+
Authority | <empty> |
+-----+
Additional | <empty> |
+-----+

```

Neither of these answers have AA set, so neither response comes from authoritative data. The different contents and different TTLs suggest that the two servers cached data at different times, and that the first server cached the response to a QTYPE=A query and the second cached the response to a HINFO query.

6.2.3. QNAME=SRI-NIC.ARPA, QTYPE=MX

This type of query might be result from a mailer trying to look up routing information for the mail destination HOSTMASTER@SRI-NIC.ARPA. The response from C.ISI.EDU would be:

```

Header      |-----|
            | OPCODE=SQUERY, RESPONSE, AA |
            |-----|
Question    | QNAME=SRI-NIC.ARPA., QCLASS=IN, QTYPE=MX |
            |-----|
Answer      | SRI-NIC.ARPA. 86400 IN      MX      0 SRI-NIC.ARPA. |
            |-----|
Authority    | <empty> |
            |-----|
Additional  | SRI-NIC.ARPA. 86400 IN      A      26.0.0.73 |
            |                A      10.0.0.51 |
            |-----|

```

This response contains the MX RR in the answer section of the response. The additional section contains the address RRs because the name server at C.ISI.EDU guesses that the requester will need the addresses in order to properly use the information carried by the MX.

6.2.4. QNAME=SRI-NIC.ARPA, QTYPE=NS

C.ISI.EDU would reply to this query with:

```

Header      |-----|
            | OPCODE=SQUERY, RESPONSE, AA |
            |-----|
Question    | QNAME=SRI-NIC.ARPA., QCLASS=IN, QTYPE=NS |
            |-----|
Answer      | <empty> |
            |-----|
Authority    | <empty> |
            |-----|
Additional  | <empty> |
            |-----|

```

The only difference between the response and the query is the AA and RESPONSE bits in the header. The interpretation of this response is that the server is authoritative for the name, and the name exists, but no RRs of type NS are present there.

6.2.5. QNAME=SRI-NIC.ARPA, QTYPE=A

If a user mistyped a host name, we might see this type of query.

C.ISI.EDU would answer it with:

```

+-----+
Header   | OPCODE=SQUERY, RESPONSE, AA, RCODE=NE |
+-----+
Question | QNAME=SIR-NIC.ARPA., QCLASS=IN, QTYPE=A |
+-----+
Answer   | <empty> |
+-----+
Authority | . SOA SRI-NIC.ARPA. HOSTMASTER.SRI-NIC.ARPA. |
          |           870611 1800 300 604800 86400 |
+-----+
Additional | <empty> |
+-----+

```

This response states that the name does not exist. This condition is signalled in the response code (RCODE) section of the header.

The SOA RR in the authority section is the optional negative caching information which allows the resolver using this response to assume that the name will not exist for the SOA MINIMUM (86400) seconds.

6.2.6. QNAME=BRL.MIL, QTYPE=A

If this query is sent to C.ISI.EDU, the reply would be:

```

+-----+
Header   | OPCODE=SQUERY, RESPONSE |
+-----+
Question | QNAME=BRL.MIL, QCLASS=IN, QTYPE=A |
+-----+
Answer   | <empty> |
+-----+
Authority | MIL.           86400 IN NS       SRI-NIC.ARPA. |
          |           86400   NS       A.ISI.EDU. |
+-----+
Additional | A.ISI.EDU.           A       26.3.0.103 |
          | SRI-NIC.ARPA.           A       26.0.0.73 |
          |                       A       10.0.0.51 |
+-----+

```

This response has an empty answer section, but is not authoritative, so it is a referral. The name server on C.ISI.EDU, realizing that it is not authoritative for the MIL domain, has referred the requester to servers on A.ISI.EDU and SRI-NIC.ARPA, which it knows are authoritative for the MIL domain.

6.2.7. QNAME=USC-ISIC.ARPA, QTYPE=A

The response to this query from A.ISI.EDU would be:

```

Header      |-----|
| OPCODE=SQUERY, RESPONSE, AA |
|-----|
Question    | QNAME=USC-ISIC.ARPA., QCLASS=IN, QTYPE=A |
|-----|
Answer      | USC-ISIC.ARPA. 86400 IN CNAME      C.ISI.EDU. |
| C.ISI.EDU.      86400 IN A        10.0.0.52 |
|-----|
Authority    | <empty> |
|-----|
Additional  | <empty> |
|-----|

```

Note that the AA bit in the header guarantees that the data matching QNAME is authoritative, but does not say anything about whether the data for C.ISI.EDU is authoritative. This complete reply is possible because A.ISI.EDU happens to be authoritative for both the ARPA domain where USC-ISIC.ARPA is found and the ISI.EDU domain where C.ISI.EDU data is found.

If the same query was sent to C.ISI.EDU, its response might be the same as shown above if it had its own address in its cache, but might also be:

```

Header      |-----|
            | OPCODE=SQUERY, RESPONSE, AA |
            |-----|
Question    | QNAME=USC-ISIC.ARPA., QCLASS=IN, QTYPE=A |
            |-----|
Answer      | USC-ISIC.ARPA. 86400 IN CNAME C.ISI.EDU. |
            |-----|
Authority    | ISI.EDU.      172800 IN NS      VAXA.ISI.EDU. |
            |               | NS      A.ISI.EDU. |
            |               | NS      VENERA.ISI.EDU. |
            |-----|
Additional  | VAXA.ISI.EDU. 172800 A      10.2.0.27 |
            |               | 172800 A      128.9.0.33 |
            | VENERA.ISI.EDU. 172800 A      10.1.0.52 |
            |               | 172800 A      128.9.0.32 |
            | A.ISI.EDU.    172800 A      26.3.0.103 |
            |-----|

```

This reply contains an authoritative reply for the alias USC-ISIC.ARPA, plus a referral to the name servers for ISI.EDU. This sort of reply isn't very likely given that the query is for the host name of the name server being asked, but would be common for other aliases.

6.2.8. QNAME=USC-ISIC.ARPA, QTYPE=CNAME

If this query is sent to either A.ISI.EDU or C.ISI.EDU, the reply would be:

```

Header      |-----|
            | OPCODE=SQUERY, RESPONSE, AA |
            |-----|
Question    | QNAME=USC-ISIC.ARPA., QCLASS=IN, QTYPE=A |
            |-----|
Answer      | USC-ISIC.ARPA. 86400 IN CNAME C.ISI.EDU. |
            |-----|
Authority    | <empty> |
            |-----|
Additional  | <empty> |
            |-----|

```

Because QTYPE=CNAME, the CNAME RR itself answers the query, and the name server doesn't attempt to look up anything for C.ISI.EDU. (Except possibly for the additional section.)

6.3. Example resolution

The following examples illustrate the operations a resolver must perform for its client. We assume that the resolver is starting without a

cache, as might be the case after system boot. We further assume that the system is not one of the hosts in the data and that the host is located somewhere on net 26, and that its safety belt (SBELT) data structure has the following information:

```
Match count = -1
SRI-NIC.ARPA.  26.0.0.73      10.0.0.51
A.ISI.EDU.     26.3.0.103
```

This information specifies servers to try, their addresses, and a match count of -1, which says that the servers aren't very close to the target. Note that the -1 isn't supposed to be an accurate closeness measure, just a value so that later stages of the algorithm will work.

The following examples illustrate the use of a cache, so each example assumes that previous requests have completed.

6.3.1. Resolve MX for ISI.EDU.

Suppose the first request to the resolver comes from the local mailer, which has mail for PVM@ISI.EDU. The mailer might then ask for type MX RRs for the domain name ISI.EDU.

The resolver would look in its cache for MX RRs at ISI.EDU, but the empty cache wouldn't be helpful. The resolver would recognize that it needed to query foreign servers and try to determine the best servers to query. This search would look for NS RRs for the domains ISI.EDU, EDU, and the root. These searches of the cache would also fail. As a last resort, the resolver would use the information from the SBELT, copying it into its SLIST structure.

At this point the resolver would need to pick one of the three available addresses to try. Given that the resolver is on net 26, it should choose either 26.0.0.73 or 26.3.0.103 as its first choice. It would then send off a query of the form:

```

Header      |-----|
            | OPCODE=SQUERY |
            |-----|
Question    | QNAME=ISI.EDU., QCLASS=IN, QTYPE=MX |
            |-----|
Answer      | <empty> |
            |-----|
Authority    | <empty> |
            |-----|
Additional  | <empty> |
            |-----|

```

The resolver would then wait for a response to its query or a timeout. If the timeout occurs, it would try different servers, then different addresses of the same servers, lastly retrying addresses already tried. It might eventually receive a reply from SRI-NIC.ARPA:

```

Header      |-----|
            | OPCODE=SQUERY, RESPONSE |
            |-----|
Question    | QNAME=ISI.EDU., QCLASS=IN, QTYPE=MX |
            |-----|
Answer      | <empty> |
            |-----|
Authority    | ISI.EDU.      172800 IN NS      VAXA.ISI.EDU. |
            |               |                NS      A.ISI.EDU. |
            |               |                NS      VENERA.ISI.EDU. |
            |-----|
Additional  | VAXA.ISI.EDU. 172800  A      10.2.0.27 |
            |               |                A      128.9.0.33 |
            | VENERA.ISI.EDU. 172800  A      10.1.0.52 |
            |               |                A      128.9.0.32 |
            | A.ISI.EDU.    172800  A      26.3.0.103 |
            |-----|

```

The resolver would notice that the information in the response gave a closer delegation to ISI.EDU than its existing SLIST (since it matches three labels). The resolver would then cache the information in this response and use it to set up a new SLIST:

```

Match count = 3
A.ISI.EDU.      26.3.0.103
VAXA.ISI.EDU.  10.2.0.27      128.9.0.33
VENERA.ISI.EDU. 10.1.0.52      128.9.0.32

```

A.ISI.EDU appears on this list as well as the previous one, but that is purely coincidental. The resolver would again start transmitting and waiting for responses. Eventually it would get an answer:


```

Header      |-----+-----|
            | | OPCODE=SQUERY, RESPONSE, AA | |
            |-----+-----|
Question    | | QNAME=ISI.EDU., QCLASS=IN, QTYPE=MX | |
            |-----+-----|
Answer      | | ISI.EDU.                MX 10 VENERA.ISI.EDU. | |
            | |                               MX 20 VAXA.ISI.EDU. | |
            |-----+-----|
Authority    | | <empty> | |
            |-----+-----|
Additional   | | VAXA.ISI.EDU.  172800  A  10.2.0.27 | |
            | |                               172800  A  128.9.0.33 | |
            | | VENERA.ISI.EDU. 172800  A  10.1.0.52 | |
            | |                               172800  A  128.9.0.32 | |
            |-----+-----|

```

The resolver would add this information to its cache, and return the MX RRs to its client.

6.3.2. Get the host name for address 26.6.0.65

The resolver would translate this into a request for PTR RRs for 65.0.6.26.IN-ADDR.ARPA. This information is not in the cache, so the resolver would look for foreign servers to ask. No servers would match, so it would use SBELT again. (Note that the servers for the ISI.EDU domain are in the cache, but ISI.EDU is not an ancestor of 65.0.6.26.IN-ADDR.ARPA, so the SBELT is used.)

Since this request is within the authoritative data of both servers in SBELT, eventually one would return:

```

Header      |-----+-----+-----+-----+-----+-----+-----+-----+-----+
            | OPCODE=SQUERY, RESPONSE, AA                               |
            +-----+-----+-----+-----+-----+-----+-----+-----+-----+
Question    | QNAME=65.0.6.26.IN-ADDR.ARPA.,QCLASS=IN,QTYPE=PTR         |
            +-----+-----+-----+-----+-----+-----+-----+-----+-----+
Answer      | 65.0.6.26.IN-ADDR.ARPA.      PTR      ACC.ARPA.           |
            +-----+-----+-----+-----+-----+-----+-----+-----+-----+
Authority   | <empty>                                                     |
            +-----+-----+-----+-----+-----+-----+-----+-----+-----+
Additional  | <empty>                                                     |
            +-----+-----+-----+-----+-----+-----+-----+-----+-----+

```

6.3.3. Get the host address of poneria.ISI.EDU

This request would translate into a type A request for poneria.ISI.EDU. The resolver would not find any cached data for this name, but would find the NS RRs in the cache for ISI.EDU when it looks for foreign servers to ask. Using this data, it would construct a SLIST of the form:

Match count = 3

```

A.ISI.EDU.      26.3.0.103
VAXA.ISI.EDU.   10.2.0.27      128.9.0.33
VENERA.ISI.EDU. 10.1.0.52

```

A.ISI.EDU is listed first on the assumption that the resolver orders its choices by preference, and A.ISI.EDU is on the same network.

One of these servers would answer the query.

7. REFERENCES and BIBLIOGRAPHY

- [Dyer 87] Dyer, S., and F. Hsu, "Hesiod", Project Athena
 Technical Plan - Name Service, April 1987, version 1.9.
 Describes the fundamentals of the Hesiod name service.
- [IEN-116] J. Postel, "Internet Name Server", IEN-116,
 USC/Information Sciences Institute, August 1979.
 A name service obsoleted by the Domain Name System, but
 still in use.

- [Quarterman 86] Quarterman, J., and J. Hoskins, "Notable Computer Networks", Communications of the ACM, October 1986, volume 29, number 10.
- [RFC-742] K. Harrenstien, "NAME/FINGER", RFC-742, Network Information Center, SRI International, December 1977.
- [RFC-768] J. Postel, "User Datagram Protocol", RFC-768, USC/Information Sciences Institute, August 1980.
- [RFC-793] J. Postel, "Transmission Control Protocol", RFC-793, USC/Information Sciences Institute, September 1981.
- [RFC-799] D. Mills, "Internet Name Domains", RFC-799, COMSAT, September 1981.
- Suggests introduction of a hierarchy in place of a flat name space for the Internet.
- [RFC-805] J. Postel, "Computer Mail Meeting Notes", RFC-805, USC/Information Sciences Institute, February 1982.
- [RFC-810] E. Feinler, K. Harrenstien, Z. Su, and V. White, "DOD Internet Host Table Specification", RFC-810, Network Information Center, SRI International, March 1982.
- Obsolete. See RFC-952.
- [RFC-811] K. Harrenstien, V. White, and E. Feinler, "Hostnames Server", RFC-811, Network Information Center, SRI International, March 1982.
- Obsolete. See RFC-953.
- [RFC-812] K. Harrenstien, and V. White, "NICNAME/WHOIS", RFC-812, Network Information Center, SRI International, March 1982.
- [RFC-819] Z. Su, and J. Postel, "The Domain Naming Convention for Internet User Applications", RFC-819, Network Information Center, SRI International, August 1982.
- Early thoughts on the design of the domain system. Current implementation is completely different.
- [RFC-821] J. Postel, "Simple Mail Transfer Protocol", RFC-821, USC/Information Sciences Institute, August 1980.

- [RFC-830] Z. Su, "A Distributed System for Internet Name Service", RFC-830, Network Information Center, SRI International, October 1982.
- Early thoughts on the design of the domain system. Current implementation is completely different.
- [RFC-882] P. Mockapetris, "Domain names - Concepts and Facilities," RFC-882, USC/Information Sciences Institute, November 1983.
- Superceded by this memo.
- [RFC-883] P. Mockapetris, "Domain names - Implementation and Specification," RFC-883, USC/Information Sciences Institute, November 1983.
- Superceded by this memo.
- [RFC-920] J. Postel and J. Reynolds, "Domain Requirements", RFC-920, USC/Information Sciences Institute October 1984.
- Explains the naming scheme for top level domains.
- [RFC-952] K. Harrenstien, M. Stahl, E. Feinler, "DoD Internet Host Table Specification", RFC-952, SRI, October 1985.
- Specifies the format of HOSTS.TXT, the host/address table replaced by the DNS.
- [RFC-953] K. Harrenstien, M. Stahl, E. Feinler, "HOSTNAME Server", RFC-953, SRI, October 1985.
- This RFC contains the official specification of the hostname server protocol, which is obsoleted by the DNS. This TCP based protocol accesses information stored in the RFC-952 format, and is used to obtain copies of the host table.
- [RFC-973] P. Mockapetris, "Domain System Changes and Observations", RFC-973, USC/Information Sciences Institute, January 1986.
- Describes changes to RFC-882 and RFC-883 and reasons for them. Now obsolete.

- [RFC-974] C. Partridge, "Mail routing and the domain system", RFC-974, CSNET CIC BBN Labs, January 1986.
- Describes the transition from HOSTS.TXT based mail addressing to the more powerful MX system used with the domain system.
- [RFC-1001] NetBIOS Working Group, "Protocol standard for a NetBIOS service on a TCP/UDP transport: Concepts and Methods", RFC-1001, March 1987.
- This RFC and RFC-1002 are a preliminary design for NETBIOS on top of TCP/IP which proposes to base NetBIOS name service on top of the DNS.
- [RFC-1002] NetBIOS Working Group, "Protocol standard for a NetBIOS service on a TCP/UDP transport: Detailed Specifications", RFC-1002, March 1987.
- [RFC-1010] J. Reynolds and J. Postel, "Assigned Numbers", RFC-1010, USC/Information Sciences Institute, May 1987
- Contains socket numbers and mnemonics for host names, operating systems, etc.
- [RFC-1031] W. Lazear, "MILNET Name Domain Transition", RFC-1031, November 1987.
- Describes a plan for converting the MILNET to the DNS.
- [RFC-1032] M. K. Stahl, "Establishing a Domain - Guidelines for Administrators", RFC-1032, November 1987.
- Describes the registration policies used by the NIC to administer the top level domains and delegate subzones.
- [RFC-1033] M. K. Lottor, "Domain Administrators Operations Guide", RFC-1033, November 1987.
- A cookbook for domain administrators.
- [Solomon 82] M. Solomon, L. Landweber, and D. Neuhengen, "The CSNET Name Server", Computer Networks, vol 6, nr 3, July 1982.
- Describes a name service for CSNET which is independent from the DNS and DNS use in the CSNET.

Index

A 12
Absolute names 8
Aliases 14, 31
Authority 6
AXFR 17

Case of characters 7
CH 12
CNAME 12, 13, 31
Completion queries 18

Domain name 6, 7

Glue RRs 20

HINFO 12

IN 12
Inverse queries 16
Iterative 4

Label 7

Mailbox names 9
MX 12

Name error 27, 36
Name servers 5, 17
NE 30
Negative caching 44
NS 12

Opcode 16

PTR 12

QCLASS 16
QTYPE 16

RDATA 13
Recursive 4
Recursive service 22
Relative names 7
Resolvers 6
RR 12

Safety belt	33
Sections	16
SOA	12
Standard queries	22
Status queries	18
Stub resolvers	32
TTL	12, 13
Wildcards	25
Zone transfers	28
Zones	19

Obsoletes: RFCs 882, 883, 973

DOMAIN NAMES - IMPLEMENTATION AND SPECIFICATION

1. STATUS OF THIS MEMO

This RFC describes the details of the domain system and protocol, and assumes that the reader is familiar with the concepts discussed in a companion RFC, "Domain Names - Concepts and Facilities" [RFC-1034].

The domain system is a mixture of functions and data types which are an official protocol and functions and data types which are still experimental. Since the domain system is intentionally extensible, new data types and experimental behavior should always be expected in parts of the system beyond the official protocol. The official protocol parts include standard queries, responses and the Internet class RR data formats (e.g., host addresses). Since the previous RFC set, several definitions have changed, so some previous definitions are obsolete.

Experimental or obsolete features are clearly marked in these RFCs, and such information should be used with caution.

The reader is especially cautioned not to depend on the values which appear in examples to be current or complete, since their purpose is primarily pedagogical. Distribution of this memo is unlimited.

Table of Contents

1. STATUS OF THIS MEMO	1
2. INTRODUCTION	3
2.1. Overview	3
2.2. Common configurations	4
2.3. Conventions	7
2.3.1. Preferred name syntax	7
2.3.2. Data Transmission Order	8
2.3.3. Character Case	9
2.3.4. Size limits	10
3. DOMAIN NAME SPACE AND RR DEFINITIONS	10
3.1. Name space definitions	10
3.2. RR definitions	11
3.2.1. Format	11
3.2.2. TYPE values	12
3.2.3. QTYPE values	12
3.2.4. CLASS values	13

3.2.5. QCLASS values	13
3.3. Standard RRs	13
3.3.1. CNAME RDATA format	14
3.3.2. HINFO RDATA format	14
3.3.3. MB RDATA format (EXPERIMENTAL)	14
3.3.4. MD RDATA format (Obsolete)	15
3.3.5. MF RDATA format (Obsolete)	15
3.3.6. MG RDATA format (EXPERIMENTAL)	16
3.3.7. MINFO RDATA format (EXPERIMENTAL)	16
3.3.8. MR RDATA format (EXPERIMENTAL)	17
3.3.9. MX RDATA format	17
3.3.10. NULL RDATA format (EXPERIMENTAL)	17
3.3.11. NS RDATA format	18
3.3.12. PTR RDATA format	18
3.3.13. SOA RDATA format	19
3.3.14. TXT RDATA format	20
3.4. ARPA Internet specific RRs	20
3.4.1. A RDATA format	20
3.4.2. WKS RDATA format	21
3.5. IN-ADDR.ARPA domain	22
3.6. Defining new types, classes, and special namespaces	24
4. MESSAGES	25
4.1. Format	25
4.1.1. Header section format	26
4.1.2. Question section format	28
4.1.3. Resource record format	29
4.1.4. Message compression	30
4.2. Transport	32
4.2.1. UDP usage	32
4.2.2. TCP usage	32
5. MASTER FILES	33
5.1. Format	33
5.2. Use of master files to define zones	35
5.3. Master file example	36
6. NAME SERVER IMPLEMENTATION	37
6.1. Architecture	37
6.1.1. Control	37
6.1.2. Database	37
6.1.3. Time	39
6.2. Standard query processing	39
6.3. Zone refresh and reload processing	39
6.4. Inverse queries (Optional)	40
6.4.1. The contents of inverse queries and responses	40
6.4.2. Inverse query and response example	41
6.4.3. Inverse query processing	42

6.5. Completion queries and responses	42
7. RESOLVER IMPLEMENTATION	43
7.1. Transforming a user request into a query	43
7.2. Sending the queries	44
7.3. Processing responses	46
7.4. Using the cache	47
8. MAIL SUPPORT	47
8.1. Mail exchange binding	48
8.2. Mailbox binding (Experimental)	48
9. REFERENCES and BIBLIOGRAPHY	50
Index	54

2. INTRODUCTION

2.1. Overview

The goal of domain names is to provide a mechanism for naming resources in such a way that the names are usable in different hosts, networks, protocol families, internets, and administrative organizations.

From the user's point of view, domain names are useful as arguments to a local agent, called a resolver, which retrieves information associated with the domain name. Thus a user might ask for the host address or mail information associated with a particular domain name. To enable the user to request a particular type of information, an appropriate query type is passed to the resolver with the domain name. To the user, the domain tree is a single information space; the resolver is responsible for hiding the distribution of data among name servers from the user.

From the resolver's point of view, the database that makes up the domain space is distributed among various name servers. Different parts of the domain space are stored in different name servers, although a particular data item will be stored redundantly in two or more name servers. The resolver starts with knowledge of at least one name server. When the resolver processes a user query it asks a known name server for the information; in return, the resolver either receives the desired information or a referral to another name server. Using these referrals, resolvers learn the identities and contents of other name servers. Resolvers are responsible for dealing with the distribution of the domain space and dealing with the effects of name server failure by consulting redundant databases in other servers.

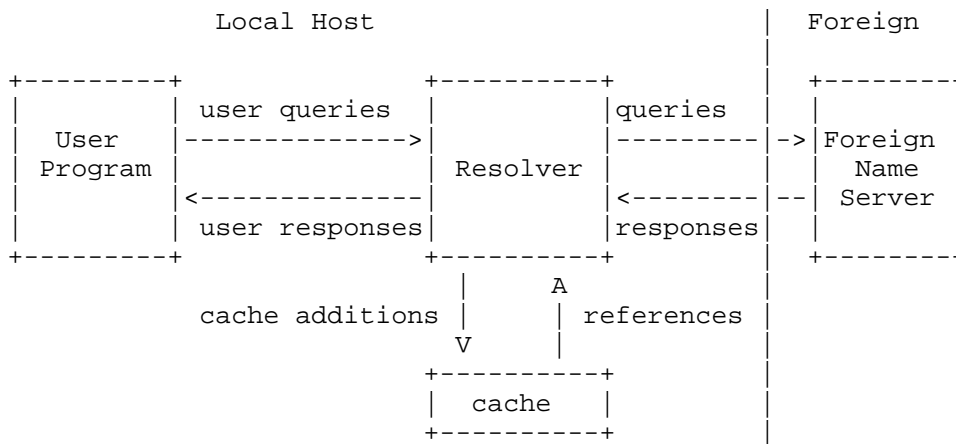
Name servers manage two kinds of data. The first kind of data held in sets called zones; each zone is the complete database for a particular "pruned" subtree of the domain space. This data is called authoritative. A name server periodically checks to make sure that its zones are up to date, and if not, obtains a new copy of updated zones

from master files stored locally or in another name server. The second kind of data is cached data which was acquired by a local resolver. This data may be incomplete, but improves the performance of the retrieval process when non-local data is repeatedly accessed. Cached data is eventually discarded by a timeout mechanism.

This functional structure isolates the problems of user interface, failure recovery, and distribution in the resolvers and isolates the database update and refresh problems in the name servers.

2.2. Common configurations

A host can participate in the domain name system in a number of ways, depending on whether the host runs programs that retrieve information from the domain system, name servers that answer queries from other hosts, or various combinations of both functions. The simplest, and perhaps most typical, configuration is shown below:

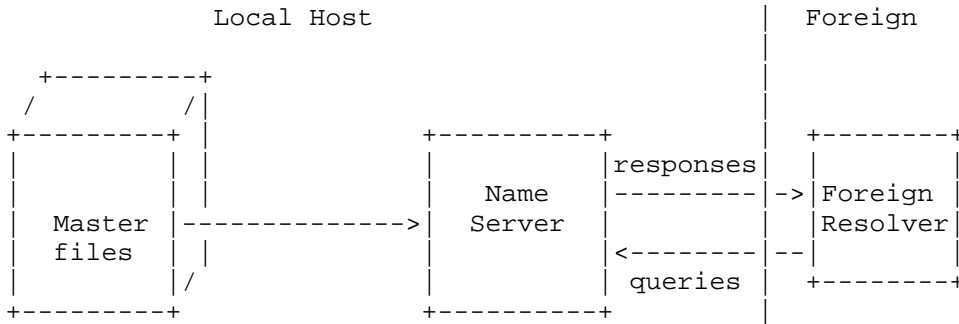


User programs interact with the domain name space through resolvers; the format of user queries and user responses is specific to the host and its operating system. User queries will typically be operating system calls, and the resolver and its cache will be part of the host operating system. Less capable hosts may choose to implement the resolver as a subroutine to be linked in with every program that needs its services. Resolvers answer user queries with information they acquire via queries to foreign name servers and the local cache.

Note that the resolver may have to make several queries to several different foreign name servers to answer a particular user query, and hence the resolution of a user query may involve several network accesses and an arbitrary amount of time. The queries to foreign name servers and the corresponding responses have a standard format described

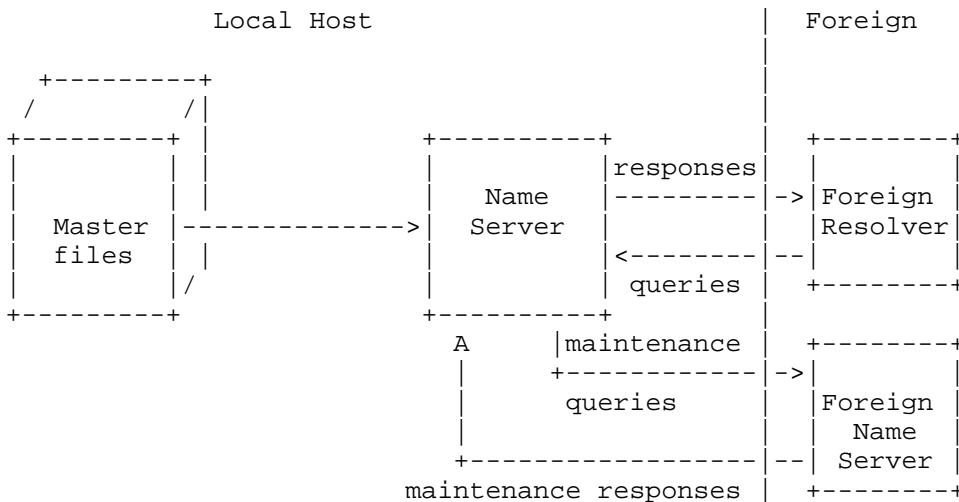
in this memo, and may be datagrams.

Depending on its capabilities, a name server could be a stand alone program on a dedicated machine or a process or processes on a large timeshared host. A simple configuration might be:



Here a primary name server acquires information about one or more zones by reading master files from its local file system, and answers queries about those zones that arrive from foreign resolvers.

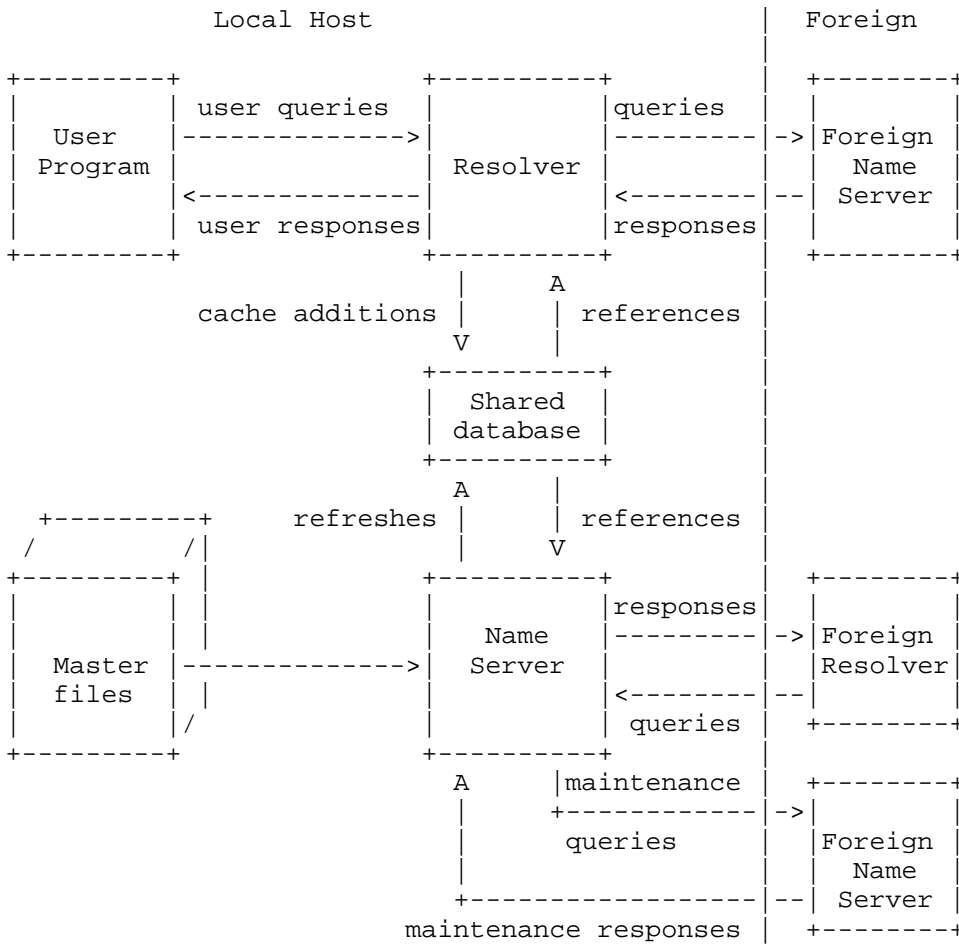
The DNS requires that all zones be redundantly supported by more than one name server. Designated secondary servers can acquire zones and check for updates from the primary server using the zone transfer protocol of the DNS. This configuration is shown below:



In this configuration, the name server periodically establishes a virtual circuit to a foreign name server to acquire a copy of a zone or to check that an existing copy has not changed. The messages sent for

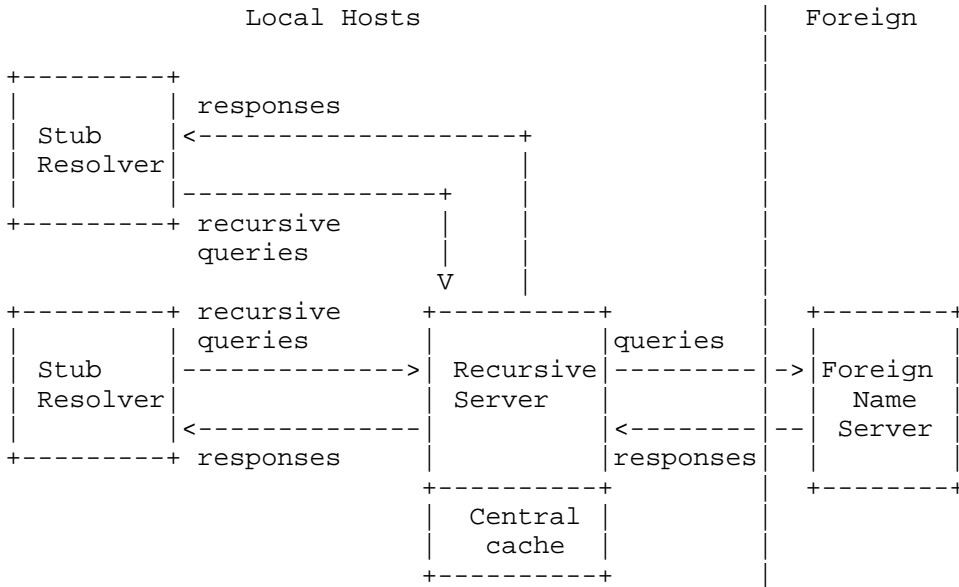
these maintenance activities follow the same form as queries and responses, but the message sequences are somewhat different.

The information flow in a host that supports all aspects of the domain name system is shown below:



The shared database holds domain space data for the local name server and resolver. The contents of the shared database will typically be a mixture of authoritative data maintained by the periodic refresh operations of the name server and cached data from previous resolver requests. The structure of the domain data and the necessity for synchronization between name servers and resolvers imply the general characteristics of this database, but the actual format is up to the local implementor.

Information flow can also be tailored so that a group of hosts act together to optimize activities. Sometimes this is done to offload less capable hosts so that they do not have to implement a full resolver. This can be appropriate for PCs or hosts which want to minimize the amount of new network code which is required. This scheme can also allow a group of hosts can share a small number of caches rather than maintaining a large number of separate caches, on the premise that the centralized caches will have a higher hit ratio. In either case, resolvers are replaced with stub resolvers which act as front ends to resolvers located in a recursive server in one or more name servers known to perform that service:



In any case, note that domain components are always replicated for reliability whenever possible.

2.3. Conventions

The domain system has several conventions dealing with low-level, but fundamental, issues. While the implementor is free to violate these conventions WITHIN HIS OWN SYSTEM, he must observe these conventions in ALL behavior observed from other hosts.

2.3.1. Preferred name syntax

The DNS specifications attempt to be as general as possible in the rules for constructing domain names. The idea is that the name of any existing object can be expressed as a domain name with minimal changes.

However, when assigning a domain name for an object, the prudent user will select a name which satisfies both the rules of the domain system and any existing rules for the object, whether these rules are published or implied by existing programs.

For example, when naming a mail domain, the user should satisfy both the rules of this memo and those in RFC-822. When creating a new host name, the old rules for HOSTS.TXT should be followed. This avoids problems when old software is converted to use domain names.

The following syntax will result in fewer problems with many applications that use domain names (e.g., mail, TELNET).

<domain> ::= <subdomain> | " "

<subdomain> ::= <label> | <subdomain> "." <label>

<label> ::= <letter> [[<ldh-str>] <let-dig>]

<ldh-str> ::= <let-dig-hyp> | <let-dig-hyp> <ldh-str>

<let-dig-hyp> ::= <let-dig> | "-"

<let-dig> ::= <letter> | <digit>

<letter> ::= any one of the 52 alphabetic characters A through Z in upper case and a through z in lower case

<digit> ::= any one of the ten digits 0 through 9

Note that while upper and lower case letters are allowed in domain names, no significance is attached to the case. That is, two names with the same spelling but different case are to be treated as if identical.

The labels must follow the rules for ARPANET host names. They must start with a letter, end with a letter or digit, and have as interior characters only letters, digits, and hyphen. There are also some restrictions on the length. Labels must be 63 characters or less.

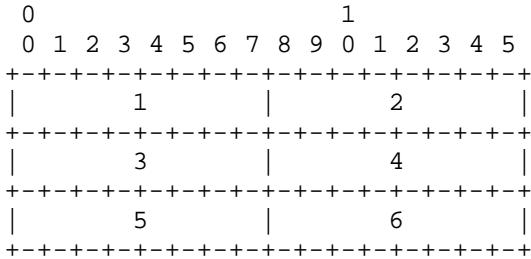
For example, the following strings identify hosts in the Internet:

A.ISI.EDU XX.LCS.MIT.EDU SRI-NIC.ARPA

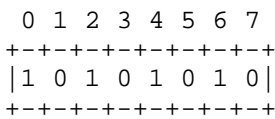
2.3.2. Data Transmission Order

The order of transmission of the header and data described in this document is resolved to the octet level. Whenever a diagram shows a

group of octets, the order of transmission of those octets is the normal order in which they are read in English. For example, in the following diagram, the octets are transmitted in the order they are numbered.



Whenever an octet represents a numeric quantity, the left most bit in the diagram is the high order or most significant bit. That is, the bit labeled 0 is the most significant bit. For example, the following diagram represents the value 170 (decimal).



Similarly, whenever a multi-octet field represents a numeric quantity the left most bit of the whole field is the most significant bit. When a multi-octet quantity is transmitted the most significant octet is transmitted first.

2.3.3. Character Case

For all parts of the DNS that are part of the official protocol, all comparisons between character strings (e.g., labels, domain names, etc.) are done in a case-insensitive manner. At present, this rule is in force throughout the domain system without exception. However, future additions beyond current usage may need to use the full binary octet capabilities in names, so attempts to store domain names in 7-bit ASCII or use of special bytes to terminate labels, etc., should be avoided.

When data enters the domain system, its original case should be preserved whenever possible. In certain circumstances this cannot be done. For example, if two RRs are stored in a database, one at x.y and one at X.Y, they are actually stored at the same place in the database, and hence only one casing would be preserved. The basic rule is that case can be discarded only when data is used to define structure in a database, and two names are identical when compared in a case insensitive manner.

Loss of case sensitive data must be minimized. Thus while data for x.y and X.Y may both be stored under a single location x.y or X.Y, data for a.x and B.X would never be stored under A.x, A.X, b.x, or b.X. In general, this preserves the case of the first label of a domain name, but forces standardization of interior node labels.

Systems administrators who enter data into the domain database should take care to represent the data they supply to the domain system in a case-consistent manner if their system is case-sensitive. The data distribution system in the domain system will ensure that consistent representations are preserved.

2.3.4. Size limits

Various objects and parameters in the DNS have size limits. They are listed below. Some could be easily changed, others are more fundamental.

labels	63 octets or less
names	255 octets or less
TTL	positive values of a signed 32 bit number.
UDP messages	512 octets or less

3. DOMAIN NAME SPACE AND RR DEFINITIONS

3.1. Name space definitions

Domain names in messages are expressed in terms of a sequence of labels. Each label is represented as a one octet length field followed by that number of octets. Since every domain name ends with the null label of the root, a domain name is terminated by a length byte of zero. The high order two bits of every length octet must be zero, and the remaining six bits of the length field limit the label to 63 octets or less.

To simplify implementations, the total length of a domain name (i.e., label octets and label length octets) is restricted to 255 octets or less.

Although labels can contain any 8 bit values in octets that make up a label, it is strongly recommended that labels follow the preferred syntax described elsewhere in this memo, which is compatible with existing host naming conventions. Name servers and resolvers must compare labels in a case-insensitive manner (i.e., A=a), assuming ASCII with zero parity. Non-alphabetic codes must match exactly.

3.2. RR definitions

3.2.1. Format

All RRs have the same top level format shown below:

```

          1 1 1 1 1 1
          0 1 2 3 4 5
+-----+-----+-----+-----+-----+-----+
|                                               |
| /                                               / |
| /                NAME                        / |
|                                               |
+-----+-----+-----+-----+-----+-----+
|                                               |
|                TYPE                          |
+-----+-----+-----+-----+-----+-----+
|                                               |
|                CLASS                         |
+-----+-----+-----+-----+-----+-----+
|                                               |
|                TTL                           |
+-----+-----+-----+-----+-----+-----+
|                                               |
|                RDLENGTH                      |
+-----+-----+-----+-----+-----+-----+
| /                RDATA                       / |
| /                                               / |
+-----+-----+-----+-----+-----+-----+

```

where:

NAME an owner name, i.e., the name of the node to which this resource record pertains.

TYPE two octets containing one of the RR TYPE codes.

CLASS two octets containing one of the RR CLASS codes.

TTL a 32 bit signed integer that specifies the time interval that the resource record may be cached before the source of the information should again be consulted. Zero values are interpreted to mean that the RR can only be used for the transaction in progress, and should not be cached. For example, SOA records are always distributed with a zero TTL to prohibit caching. Zero values can also be used for extremely volatile data.

RDLENGTH an unsigned 16 bit integer that specifies the length in octets of the RDATA field.

RDATA a variable length string of octets that describes the resource. The format of this information varies according to the TYPE and CLASS of the resource record.

3.2.2. TYPE values

TYPE fields are used in resource records. Note that these types are a subset of QTYPES.

TYPE	value and meaning
A	1 a host address
NS	2 an authoritative name server
MD	3 a mail destination (Obsolete - use MX)
MF	4 a mail forwarder (Obsolete - use MX)
CNAME	5 the canonical name for an alias
SOA	6 marks the start of a zone of authority
MB	7 a mailbox domain name (EXPERIMENTAL)
MG	8 a mail group member (EXPERIMENTAL)
MR	9 a mail rename domain name (EXPERIMENTAL)
NULL	10 a null RR (EXPERIMENTAL)
WKS	11 a well known service description
PTR	12 a domain name pointer
HINFO	13 host information
MINFO	14 mailbox or mail list information
MX	15 mail exchange
TXT	16 text strings

3.2.3. QTYPE values

QTYPE fields appear in the question part of a query. QTYPES are a superset of TYPES, hence all TYPES are valid QTYPES. In addition, the following QTYPES are defined:

AXFR	252	A request for a transfer of an entire zone
MAILB	253	A request for mailbox-related records (MB, MG or MR)
MAILA	254	A request for mail agent RRs (Obsolete - see MX)
*	255	A request for all records

3.2.4. CLASS values

CLASS fields appear in resource records. The following CLASS mnemonics and values are defined:

IN	1	the Internet
CS	2	the CSNET class (Obsolete - used only for examples in some obsolete RFCs)
CH	3	the CHAOS class
HS	4	Hesiod [Dyer 87]

3.2.5. QCLASS values

QCLASS fields appear in the question section of a query. QCLASS values are a superset of CLASS values; every CLASS is a valid QCLASS. In addition to CLASS values, the following QCLASSES are defined:

*	255	any class
---	-----	-----------

3.3. Standard RRs

The following RR definitions are expected to occur, at least potentially, in all classes. In particular, NS, SOA, CNAME, and PTR will be used in all classes, and have the same format in all classes. Because their RDATA format is known, all domain names in the RDATA section of these RRs may be compressed.

<domain-name> is a domain name represented as a series of labels, and terminated by a label with zero length. <character-string> is a single length octet followed by that number of characters. <character-string> is treated as binary information, and can be up to 256 characters in length (including the length octet).

3.3.1. CNAME RDATA format

```

+-----+-----+-----+-----+-----+-----+-----+-----+-----+
/                               CNAME                               /
/                               /                                   /
+-----+-----+-----+-----+-----+-----+-----+-----+-----+

```

where:

CNAME A <domain-name> which specifies the canonical or primary name for the owner. The owner name is an alias.

CNAME RRs cause no additional section processing, but name servers may choose to restart the query at the canonical name in certain cases. See the description of name server logic in [RFC-1034] for details.

3.3.2. HINFO RDATA format

```

+-----+-----+-----+-----+-----+-----+-----+-----+-----+
/                               CPU                               /
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
/                               OS                               /
+-----+-----+-----+-----+-----+-----+-----+-----+-----+

```

where:

CPU A <character-string> which specifies the CPU type.

OS A <character-string> which specifies the operating system type.

Standard values for CPU and OS can be found in [RFC-1010].

HINFO records are used to acquire general information about a host. The main use is for protocols such as FTP that can use special procedures when talking between machines or operating systems of the same type.

3.3.3. MB RDATA format (EXPERIMENTAL)

```

+-----+-----+-----+-----+-----+-----+-----+-----+-----+
/                               MADNAME                           /
/                               /                                   /
+-----+-----+-----+-----+-----+-----+-----+-----+-----+

```

where:

MADNAME A <domain-name> which specifies a host which has the specified mailbox.

MB records cause additional section processing which looks up an A type RRs corresponding to MADNAME.

3.3.4. MD RDATA format (Obsolete)

```
+-----+-----+-----+-----+-----+-----+-----+-----+
/                               MADNAME                               /
/                               /                                     /
+-----+-----+-----+-----+-----+-----+-----+-----+
```

where:

MADNAME A <domain-name> which specifies a host which has a mail agent for the domain which should be able to deliver mail for the domain.

MD records cause additional section processing which looks up an A type record corresponding to MADNAME.

MD is obsolete. See the definition of MX and [RFC-974] for details of the new scheme. The recommended policy for dealing with MD RRs found in a master file is to reject them, or to convert them to MX RRs with a preference of 0.

3.3.5. MF RDATA format (Obsolete)

```
+-----+-----+-----+-----+-----+-----+-----+-----+
/                               MADNAME                               /
/                               /                                     /
+-----+-----+-----+-----+-----+-----+-----+-----+
```

where:

MADNAME A <domain-name> which specifies a host which has a mail agent for the domain which will accept mail for forwarding to the domain.

MF records cause additional section processing which looks up an A type record corresponding to MADNAME.

MF is obsolete. See the definition of MX and [RFC-974] for details ofw the new scheme. The recommended policy for dealing with MD RRs found in a master file is to reject them, or to convert them to MX RRs with a preference of 10.

3.3.6. MG RDATA format (EXPERIMENTAL)

```

+-----+
/                MGMNAME                /
/                                          /
+-----+

```

where:

MGMNAME A <domain-name> which specifies a mailbox which is a member of the mail group specified by the domain name.

MG records cause no additional section processing.

3.3.7. MINFO RDATA format (EXPERIMENTAL)

```

+-----+
/                RMAILBX                /
+-----+
/                EMAILBX                /
+-----+

```

where:

RMAILBX A <domain-name> which specifies a mailbox which is responsible for the mailing list or mailbox. If this domain name names the root, the owner of the MINFO RR is responsible for itself. Note that many existing mailing lists use a mailbox X-request for the RMAILBX field of mailing list X, e.g., Msggroup-request for Msggroup. This field provides a more general mechanism.

EMAILBX A <domain-name> which specifies a mailbox which is to receive error messages related to the mailing list or mailbox specified by the owner of the MINFO RR (similar to the ERRORS-TO: field which has been proposed). If this domain name names the root, errors should be returned to the sender of the message.

MINFO records cause no additional section processing. Although these records can be associated with a simple mailbox, they are usually used with a mailing list.

3.3.8. MR RDATA format (EXPERIMENTAL)

```

+-----+-----+-----+-----+-----+-----+-----+-----+
/                               NEWNAME                               /
/                               /                                     /
+-----+-----+-----+-----+-----+-----+-----+-----+

```

where:

NEWNAME A <domain-name> which specifies a mailbox which is the proper rename of the specified mailbox.

MR records cause no additional section processing. The main use for MR is as a forwarding entry for a user who has moved to a different mailbox.

3.3.9. MX RDATA format

```

+-----+-----+-----+-----+-----+-----+-----+-----+
|                               PREFERENCE                               |
+-----+-----+-----+-----+-----+-----+-----+-----+
/                               EXCHANGE                               /
/                               /                                     /
+-----+-----+-----+-----+-----+-----+-----+-----+

```

where:

PREFERENCE A 16 bit integer which specifies the preference given to this RR among others at the same owner. Lower values are preferred.

EXCHANGE A <domain-name> which specifies a host willing to act as a mail exchange for the owner name.

MX records cause type A additional section processing for the host specified by EXCHANGE. The use of MX RRs is explained in detail in [RFC-974].

3.3.10. NULL RDATA format (EXPERIMENTAL)

```

+-----+-----+-----+-----+-----+-----+-----+-----+
/                               <anything>                               /
/                               /                                     /
+-----+-----+-----+-----+-----+-----+-----+-----+

```

Anything at all may be in the RDATA field so long as it is 65535 octets or less.

NULL records cause no additional section processing. NULL RRs are not allowed in master files. NULLs are used as placeholders in some experimental extensions of the DNS.

3.3.11. NS RDATA format

```
+-----+
/                NSDNAME                /
/                                          /
+-----+
```

where:

NSDNAME A <domain-name> which specifies a host which should be authoritative for the specified class and domain.

NS records cause both the usual additional section processing to locate a type A record, and, when used in a referral, a special search of the zone in which they reside for glue information.

The NS RR states that the named host should be expected to have a zone starting at owner name of the specified class. Note that the class may not indicate the protocol family which should be used to communicate with the host, although it is typically a strong hint. For example, hosts which are name servers for either Internet (IN) or Hesiod (HS) class information are normally queried using IN class protocols.

3.3.12. PTR RDATA format

```
+-----+
/                PTRDNAME                /
+-----+
```

where:

PTRDNAME A <domain-name> which points to some location in the domain name space.

PTR records cause no additional section processing. These RRs are used in special domains to point to some other location in the domain space. These records are simple data, and don't imply any special processing similar to that performed by CNAME, which identifies aliases. See the description of the IN-ADDR.ARPA domain for an example.

3.3.13. SOA RDATA format

```

+-----+
/                MNAME                /
/
+-----+
/                RNAME                /
+-----+
|                SERIAL                |
+-----+
|                REFRESH                |
+-----+
|                RETRY                  |
+-----+
|                EXPIRE                  |
+-----+
|                MINIMUM                  |
+-----+

```

where:

MNAME The <domain-name> of the name server that was the original or primary source of data for this zone.

RNAME A <domain-name> which specifies the mailbox of the person responsible for this zone.

SERIAL The unsigned 32 bit version number of the original copy of the zone. Zone transfers preserve this value. This value wraps and should be compared using sequence space arithmetic.

REFRESH A 32 bit time interval before the zone should be refreshed.

RETRY A 32 bit time interval that should elapse before a failed refresh should be retried.

EXPIRE A 32 bit time value that specifies the upper limit on the time interval that can elapse before the zone is no longer authoritative.

MINIMUM The unsigned 32 bit minimum TTL field that should be exported with any RR from this zone.

SOA records cause no additional section processing.

All times are in units of seconds.

Most of these fields are pertinent only for name server maintenance operations. However, MINIMUM is used in all query operations that retrieve RRs from a zone. Whenever a RR is sent in a response to a query, the TTL field is set to the maximum of the TTL field from the RR and the MINIMUM field in the appropriate SOA. Thus MINIMUM is a lower bound on the TTL field for all RRs in a zone. Note that this use of MINIMUM should occur when the RRs are copied into the response and not when the zone is loaded from a master file or via a zone transfer. The reason for this provision is to allow future dynamic update facilities to change the SOA RR with known semantics.

3.3.14. TXT RDATA format

```
+-----+
/                TXT-DATA                /
+-----+
```

where:

TXT-DATA One or more <character-string>s.

TXT RRs are used to hold descriptive text. The semantics of the text depends on the domain where it is found.

3.4. Internet specific RRs

3.4.1. A RDATA format

```
+-----+
|                ADDRESS                |
+-----+
```

where:

ADDRESS A 32 bit Internet address.

Hosts that have multiple Internet addresses will have multiple A records.

A records cause no additional section processing. The RDATA section of an A line in a master file is an Internet address expressed as four decimal numbers separated by dots without any imbedded spaces (e.g., "10.2.0.52" or "192.0.5.6").

3.4.2. WKS RDATA format

```

+-----+
|                                     |
|                ADDRESS             |
+-----+
|          PROTOCOL                   |
+-----+
|                                     |
|          <BIT MAP>                  |
|                                     |
+-----+

```

where:

ADDRESS An 32 bit Internet address

PROTOCOL An 8 bit IP protocol number

<BIT MAP> A variable length bit map. The bit map must be a multiple of 8 bits long.

The WKS record is used to describe the well known services supported by a particular protocol on a particular internet address. The PROTOCOL field specifies an IP protocol number, and the bit map has one bit per port of the specified protocol. The first bit corresponds to port 0, the second to port 1, etc. If the bit map does not include a bit for a protocol of interest, that bit is assumed zero. The appropriate values and mnemonics for ports and protocols are specified in [RFC-1010].

For example, if PROTOCOL=TCP (6), the 26th bit corresponds to TCP port 25 (SMTP). If this bit is set, a SMTP server should be listening on TCP port 25; if zero, SMTP service is not supported on the specified address.

The purpose of WKS RRs is to provide availability information for servers for TCP and UDP. If a server supports both TCP and UDP, or has multiple Internet addresses, then multiple WKS RRs are used.

WKS RRs cause no additional section processing.

In master files, both ports and protocols are expressed using mnemonics or decimal numbers.

3.5. IN-ADDR.ARPA domain

The Internet uses a special domain to support gateway location and Internet address to host mapping. Other classes may employ a similar strategy in other domains. The intent of this domain is to provide a guaranteed method to perform host address to host name mapping, and to facilitate queries to locate all gateways on a particular network in the Internet.

Note that both of these services are similar to functions that could be performed by inverse queries; the difference is that this part of the domain name space is structured according to address, and hence can guarantee that the appropriate data can be located without an exhaustive search of the domain space.

The domain begins at IN-ADDR.ARPA and has a substructure which follows the Internet addressing structure.

Domain names in the IN-ADDR.ARPA domain are defined to have up to four labels in addition to the IN-ADDR.ARPA suffix. Each label represents one octet of an Internet address, and is expressed as a character string for a decimal value in the range 0-255 (with leading zeros omitted except in the case of a zero octet which is represented by a single zero).

Host addresses are represented by domain names that have all four labels specified. Thus data for Internet address 10.2.0.52 is located at domain name 52.0.2.10.IN-ADDR.ARPA. The reversal, though awkward to read, allows zones to be delegated which are exactly one network of address space. For example, 10.IN-ADDR.ARPA can be a zone containing data for the ARPANET, while 26.IN-ADDR.ARPA can be a separate zone for MILNET. Address nodes are used to hold pointers to primary host names in the normal domain space.

Network numbers correspond to some non-terminal nodes at various depths in the IN-ADDR.ARPA domain, since Internet network numbers are either 1, 2, or 3 octets. Network nodes are used to hold pointers to the primary host names of gateways attached to that network. Since a gateway is, by definition, on more than one network, it will typically have two or more network nodes which point at it. Gateways will also have host level pointers at their fully qualified addresses.

Both the gateway pointers at network nodes and the normal host pointers at full address nodes use the PTR RR to point back to the primary domain names of the corresponding hosts.

For example, the IN-ADDR.ARPA domain will contain information about the ISI gateway between net 10 and 26, an MIT gateway from net 10 to MIT's

net 18, and hosts A.ISI.EDU and MULTICS.MIT.EDU. Assuming that ISI gateway has addresses 10.2.0.22 and 26.0.0.103, and a name MILNET-GW.ISI.EDU, and the MIT gateway has addresses 10.0.0.77 and 18.10.0.4 and a name GW.LCS.MIT.EDU, the domain database would contain:

```

10.IN-ADDR.ARPA.      PTR MILNET-GW.ISI.EDU.
10.IN-ADDR.ARPA.      PTR GW.LCS.MIT.EDU.
18.IN-ADDR.ARPA.      PTR GW.LCS.MIT.EDU.
26.IN-ADDR.ARPA.      PTR MILNET-GW.ISI.EDU.
22.0.2.10.IN-ADDR.ARPA. PTR MILNET-GW.ISI.EDU.
103.0.0.26.IN-ADDR.ARPA. PTR MILNET-GW.ISI.EDU.
77.0.0.10.IN-ADDR.ARPA. PTR GW.LCS.MIT.EDU.
4.0.10.18.IN-ADDR.ARPA. PTR GW.LCS.MIT.EDU.
103.0.3.26.IN-ADDR.ARPA. PTR A.ISI.EDU.
6.0.0.10.IN-ADDR.ARPA. PTR MULTICS.MIT.EDU.

```

Thus a program which wanted to locate gateways on net 10 would originate a query of the form QTYPE=PTR, QCLASS=IN, QNAME=10.IN-ADDR.ARPA. It would receive two RRs in response:

```

10.IN-ADDR.ARPA.      PTR MILNET-GW.ISI.EDU.
10.IN-ADDR.ARPA.      PTR GW.LCS.MIT.EDU.

```

The program could then originate QTYPE=A, QCLASS=IN queries for MILNET-GW.ISI.EDU. and GW.LCS.MIT.EDU. to discover the Internet addresses of these gateways.

A resolver which wanted to find the host name corresponding to Internet host address 10.0.0.6 would pursue a query of the form QTYPE=PTR, QCLASS=IN, QNAME=6.0.0.10.IN-ADDR.ARPA, and would receive:

```

6.0.0.10.IN-ADDR.ARPA.  PTR MULTICS.MIT.EDU.

```

Several cautions apply to the use of these services:

- Since the IN-ADDR.ARPA special domain and the normal domain for a particular host or gateway will be in different zones, the possibility exists that that the data may be inconsistent.
- Gateways will often have two names in separate domains, only one of which can be primary.
- Systems that use the domain database to initialize their routing tables must start with enough gateway information to guarantee that they can access the appropriate name server.
- The gateway data only reflects the existence of a gateway in a manner equivalent to the current HOSTS.TXT file. It doesn't replace the dynamic availability information from GGP or EGP.

3.6. Defining new types, classes, and special namespaces

The previously defined types and classes are the ones in use as of the date of this memo. New definitions should be expected. This section makes some recommendations to designers considering additions to the existing facilities. The mailing list NAMEDROPPERS@SRI-NIC.ARPA is the forum where general discussion of design issues takes place.

In general, a new type is appropriate when new information is to be added to the database about an existing object, or we need new data formats for some totally new object. Designers should attempt to define types and their RDATA formats that are generally applicable to all classes, and which avoid duplication of information. New classes are appropriate when the DNS is to be used for a new protocol, etc which requires new class-specific data formats, or when a copy of the existing name space is desired, but a separate management domain is necessary.

New types and classes need mnemonics for master files; the format of the master files requires that the mnemonics for type and class be disjoint.

TYPE and CLASS values must be a proper subset of QTYPEs and QCLASSes respectively.

The present system uses multiple RRs to represent multiple values of a type rather than storing multiple values in the RDATA section of a single RR. This is less efficient for most applications, but does keep RRs shorter. The multiple RRs assumption is incorporated in some experimental work on dynamic update methods.

The present system attempts to minimize the duplication of data in the database in order to insure consistency. Thus, in order to find the address of the host for a mail exchange, you map the mail domain name to a host name, then the host name to addresses, rather than a direct mapping to host address. This approach is preferred because it avoids the opportunity for inconsistency.

In defining a new type of data, multiple RR types should not be used to create an ordering between entries or express different formats for equivalent bindings, instead this information should be carried in the body of the RR and a single type used. This policy avoids problems with caching multiple types and defining QTYPEs to match multiple types.

For example, the original form of mail exchange binding used two RR types one to represent a "closer" exchange (MD) and one to represent a "less close" exchange (MF). The difficulty is that the presence of one RR type in a cache doesn't convey any information about the other because the query which acquired the cached information might have used a QTYPE of MF, MD, or MAILA (which matched both). The redesigned

service used a single type (MX) with a "preference" value in the RDATA section which can order different RRs. However, if any MX RRs are found in the cache, then all should be there.

4. MESSAGES

4.1. Format

All communications inside of the domain protocol are carried in a single format called a message. The top level format of message is divided into 5 sections (some of which are empty in certain cases) shown below:

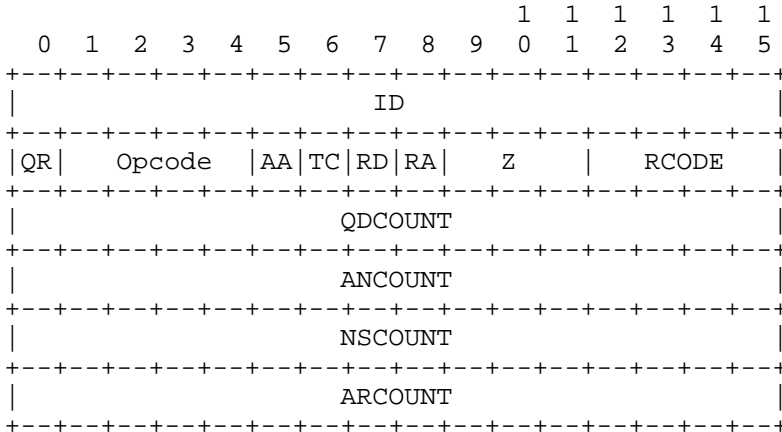
Header	
Question	the question for the name server
Answer	RRs answering the question
Authority	RRs pointing toward an authority
Additional	RRs holding additional information

The header section is always present. The header includes fields that specify which of the remaining sections are present, and also specify whether the message is a query or a response, a standard query or some other opcode, etc.

The names of the sections after the header are derived from their use in standard queries. The question section contains fields that describe a question to a name server. These fields are a query type (QTYPE), a query class (QCLASS), and a query domain name (QNAME). The last three sections have the same format: a possibly empty list of concatenated resource records (RRs). The answer section contains RRs that answer the question; the authority section contains RRs that point toward an authoritative name server; the additional records section contains RRs which relate to the query, but are not strictly answers for the question.

4.1.1.1. Header section format

The header contains the following fields:



where:

- ID A 16 bit identifier assigned by the program that generates any kind of query. This identifier is copied the corresponding reply and can be used by the requester to match up replies to outstanding queries.
- QR A one bit field that specifies whether this message is a query (0), or a response (1).
- OPCODE A four bit field that specifies kind of query in this message. This value is set by the originator of a query and copied into the response. The values are:
- | | |
|------|----------------------------------|
| 0 | a standard query (QUERY) |
| 1 | an inverse query (IQUERY) |
| 2 | a server status request (STATUS) |
| 3-15 | reserved for future use |
- AA Authoritative Answer - this bit is valid in responses, and specifies that the responding name server is an authority for the domain name in question section.
- Note that the contents of the answer section may have multiple owner names because of aliases. The AA bit

corresponds to the name which matches the query name, or the first owner name in the answer section.

- TC TrunCation - specifies that this message was truncated due to length greater than that permitted on the transmission channel.
- RD Recursion Desired - this bit may be set in a query and is copied into the response. If RD is set, it directs the name server to pursue the query recursively. Recursive query support is optional.
- RA Recursion Available - this bit is set or cleared in a response, and denotes whether recursive query support is available in the name server.
- Z Reserved for future use. Must be zero in all queries and responses.
- RCODE Response code - this 4 bit field is set as part of responses. The values have the following interpretation:
- | | |
|---|--|
| 0 | No error condition |
| 1 | Format error - The name server was unable to interpret the query. |
| 2 | Server failure - The name server was unable to process this query due to a problem with the name server. |
| 3 | Name Error - Meaningful only for responses from an authoritative name server, this code signifies that the domain name referenced in the query does not exist. |
| 4 | Not Implemented - The name server does not support the requested kind of query. |
| 5 | Refused - The name server refuses to perform the specified operation for policy reasons. For example, a name server may not wish to provide the information to the particular requester, or a name server may not wish to perform a particular operation (e.g., zone |

transfer) for particular data.

6-15 Reserved for future use.

QDCOUNT an unsigned 16 bit integer specifying the number of entries in the question section.

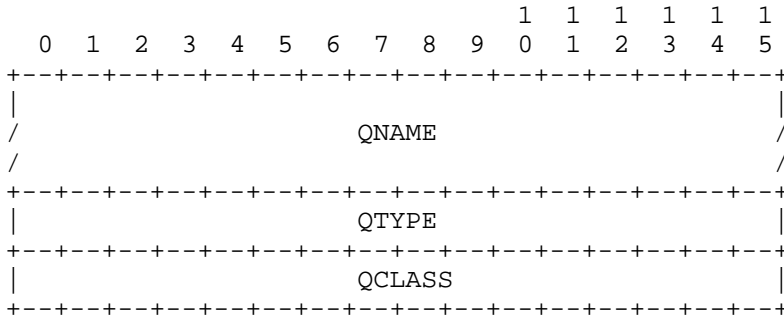
ANCOUNT an unsigned 16 bit integer specifying the number of resource records in the answer section.

NSCOUNT an unsigned 16 bit integer specifying the number of name server resource records in the authority records section.

ARCOUNT an unsigned 16 bit integer specifying the number of resource records in the additional records section.

4.1.2. Question section format

The question section is used to carry the "question" in most queries, i.e., the parameters that define what is being asked. The section contains QDCOUNT (usually 1) entries, each of the following format:



where:

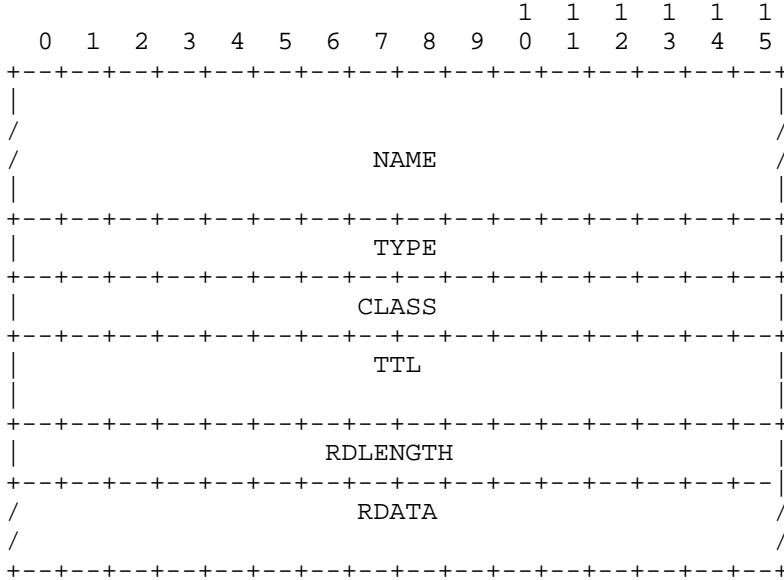
QNAME a domain name represented as a sequence of labels, where each label consists of a length octet followed by that number of octets. The domain name terminates with the zero length octet for the null label of the root. Note that this field may be an odd number of octets; no padding is used.

QTYPE a two octet code which specifies the type of the query. The values for this field include all codes valid for a TYPE field, together with some more general codes which can match more than one type of RR.

QCLASS a two octet code that specifies the class of the query.
 For example, the QCLASS field is IN for the Internet.

4.1.3. Resource record format

The answer, authority, and additional sections all share the same format: a variable number of resource records, where the number of records is specified in the corresponding count field in the header. Each resource record has the following format:



where:

- NAME a domain name to which this resource record pertains.
- TYPE two octets containing one of the RR type codes. This field specifies the meaning of the data in the RDATA field.
- CLASS two octets which specify the class of the data in the RDATA field.
- TTL a 32 bit unsigned integer that specifies the time interval (in seconds) that the resource record may be cached before it should be discarded. Zero values are interpreted to mean that the RR can only be used for the transaction in progress, and should not be cached.

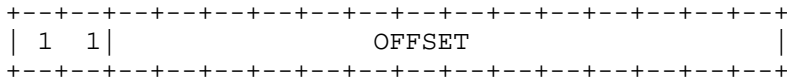
RDLENGTH an unsigned 16 bit integer that specifies the length in octets of the RDATA field.

RDATA a variable length string of octets that describes the resource. The format of this information varies according to the TYPE and CLASS of the resource record. For example, the if the TYPE is A and the CLASS is IN, the RDATA field is a 4 octet ARPA Internet address.

4.1.4. Message compression

In order to reduce the size of messages, the domain system utilizes a compression scheme which eliminates the repetition of domain names in a message. In this scheme, an entire domain name or a list of labels at the end of a domain name is replaced with a pointer to a prior occurrence of the same name.

The pointer takes the form of a two octet sequence:



The first two bits are ones. This allows a pointer to be distinguished from a label, since the label must begin with two zero bits because labels are restricted to 63 octets or less. (The 10 and 01 combinations are reserved for future use.) The OFFSET field specifies an offset from the start of the message (i.e., the first octet of the ID field in the domain header). A zero offset specifies the first byte of the ID field, etc.

The compression scheme allows a domain name in a message to be represented as either:

- a sequence of labels ending in a zero octet
- a pointer
- a sequence of labels ending with a pointer

Pointers can only be used for occurrences of a domain name where the format is not class specific. If this were not the case, a name server or resolver would be required to know the format of all RRs it handled. As yet, there are no such cases, but they may occur in future RDATA formats.

If a domain name is contained in a part of the message subject to a length field (such as the RDATA section of an RR), and compression is

used, the length of the compressed name is used in the length calculation, rather than the length of the expanded name.

Programs are free to avoid using pointers in messages they generate, although this will reduce datagram capacity, and may cause truncation. However all programs are required to understand arriving messages that contain pointers.

For example, a datagram might need to use the domain names F.ISI.ARPA, FOO.F.ISI.ARPA, ARPA, and the root. Ignoring the other fields of the message, these domain names might be represented as:

```

20 |          1          |          F          |
   +-----+-----+-----+-----+-----+-----+
22 |          3          |          I          |
   +-----+-----+-----+-----+-----+-----+
24 |          S          |          I          |
   +-----+-----+-----+-----+-----+-----+
26 |          4          |          A          |
   +-----+-----+-----+-----+-----+-----+
28 |          R          |          P          |
   +-----+-----+-----+-----+-----+-----+
30 |          A          |          0          |
   +-----+-----+-----+-----+-----+-----+

40 |          3          |          F          |
   +-----+-----+-----+-----+-----+-----+
42 |          0          |          0          |
   +-----+-----+-----+-----+-----+-----+
44 | 1  1 |                20 |
   +-----+-----+-----+-----+-----+-----+

64 | 1  1 |                26 |
   +-----+-----+-----+-----+-----+-----+

92 |          0          |
   +-----+-----+-----+-----+-----+-----+

```

The domain name for F.ISI.ARPA is shown at offset 20. The domain name FOO.F.ISI.ARPA is shown at offset 40; this definition uses a pointer to concatenate a label for FOO to the previously defined F.ISI.ARPA. The domain name ARPA is defined at offset 64 using a pointer to the ARPA component of the name F.ISI.ARPA at 20; note that this pointer relies on ARPA being the last label in the string at 20. The root domain name is

defined by a single octet of zeros at 92; the root domain name has no labels.

4.2. Transport

The DNS assumes that messages will be transmitted as datagrams or in a byte stream carried by a virtual circuit. While virtual circuits can be used for any DNS activity, datagrams are preferred for queries due to their lower overhead and better performance. Zone refresh activities must use virtual circuits because of the need for reliable transfer.

The Internet supports name server access using TCP [RFC-793] on server port 53 (decimal) as well as datagram access using UDP [RFC-768] on UDP port 53 (decimal).

4.2.1. UDP usage

Messages sent using UDP use server port 53 (decimal).

Messages carried by UDP are restricted to 512 bytes (not counting the IP or UDP headers). Longer messages are truncated and the TC bit is set in the header.

UDP is not acceptable for zone transfers, but is the recommended method for standard queries in the Internet. Queries sent using UDP may be lost, and hence a retransmission strategy is required. Queries or their responses may be reordered by the network, or by processing in name servers, so resolvers should not depend on them being returned in order.

The optimal UDP retransmission policy will vary with performance of the Internet and the needs of the client, but the following are recommended:

- The client should try other servers and server addresses before repeating a query to a specific address of a server.
- The retransmission interval should be based on prior statistics if possible. Too aggressive retransmission can easily slow responses for the community at large. Depending on how well connected the client is to its expected servers, the minimum retransmission interval should be 2-5 seconds.

More suggestions on server selection and retransmission policy can be found in the resolver section of this memo.

4.2.2. TCP usage

Messages sent over TCP connections use server port 53 (decimal). The message is prefixed with a two byte length field which gives the message

length, excluding the two byte length field. This length field allows the low-level processing to assemble a complete message before beginning to parse it.

Several connection management policies are recommended:

- The server should not block other activities waiting for TCP data.
- The server should support multiple connections.
- The server should assume that the client will initiate connection closing, and should delay closing its end of the connection until all outstanding client requests have been satisfied.
- If the server needs to close a dormant connection to reclaim resources, it should wait until the connection has been idle for a period on the order of two minutes. In particular, the server should allow the SOA and AXFR request sequence (which begins a refresh operation) to be made on a single connection. Since the server would be unable to answer queries anyway, a unilateral close or reset may be used instead of a graceful close.

5. MASTER FILES

Master files are text files that contain RRs in text form. Since the contents of a zone can be expressed in the form of a list of RRs a master file is most often used to define a zone, though it can be used to list a cache's contents. Hence, this section first discusses the format of RRs in a master file, and then the special considerations when a master file is used to create a zone in some name server.

5.1. Format

The format of these files is a sequence of entries. Entries are predominantly line-oriented, though parentheses can be used to continue a list of items across a line boundary, and text literals can contain CRLF within the text. Any combination of tabs and spaces act as a delimiter between the separate items that make up an entry. The end of any line in the master file can end with a comment. The comment starts with a ";" (semicolon).

The following entries are defined:

```
<blank>[<comment>]
```



```
$ORIGIN <domain-name> [<comment>]
$INCLUDE <file-name> [<domain-name>] [<comment>]
<domain-name><rr> [<comment>]
<blank><rr> [<comment>]
```

Blank lines, with or without comments, are allowed anywhere in the file.

Two control entries are defined: \$ORIGIN and \$INCLUDE. \$ORIGIN is followed by a domain name, and resets the current origin for relative domain names to the stated name. \$INCLUDE inserts the named file into the current file, and may optionally specify a domain name that sets the relative domain name origin for the included file. \$INCLUDE may also have a comment. Note that a \$INCLUDE entry never changes the relative origin of the parent file, regardless of changes to the relative origin made within the included file.

The last two forms represent RRs. If an entry for an RR begins with a blank, then the RR is assumed to be owned by the last stated owner. If an RR entry begins with a <domain-name>, then the owner name is reset.

<rr> contents take one of the following forms:

```
[<TTL>] [<class>] <type> <RDATA>
[<class>] [<TTL>] <type> <RDATA>
```

The RR begins with optional TTL and class fields, followed by a type and RDATA field appropriate to the type and class. Class and type use the standard mnemonics, TTL is a decimal integer. Omitted class and TTL values are default to the last explicitly stated values. Since type and class mnemonics are disjoint, the parse is unique. (Note that this order is different from the order used in examples and the order used in the actual RRs; the given order allows easier parsing and defaulting.)

<domain-name>s make up a large share of the data in the master file. The labels in the domain name are expressed as character strings and separated by dots. Quoting conventions allow arbitrary characters to be stored in domain names. Domain names that end in a dot are called absolute, and are taken as complete. Domain names which do not end in a dot are called relative; the actual domain name is the concatenation of the relative part with an origin specified in a \$ORIGIN, \$INCLUDE, or as an argument to the master file loading routine. A relative name is an error when no origin is available.

<character-string> is expressed in one or two ways: as a contiguous set of characters without interior spaces, or as a string beginning with a " and ending with a ". Inside a " delimited string any character can occur, except for a " itself, which must be quoted using \ (back slash).

Because these files are text files several special encodings are necessary to allow arbitrary data to be loaded. In particular:

of the root.

@ A free standing @ is used to denote the current origin.

\X where X is any character other than a digit (0-9), is used to quote that character so that its special meaning does not apply. For example, "\" can be used to place a dot character in a label.

\DDD where each D is a digit is the octet corresponding to the decimal number described by DDD. The resulting octet is assumed to be text and is not checked for special meaning.

() Parentheses are used to group data that crosses a line boundary. In effect, line terminations are not recognized within parentheses.

; Semicolon is used to start a comment; the remainder of the line is ignored.

5.2. Use of master files to define zones

When a master file is used to load a zone, the operation should be suppressed if any errors are encountered in the master file. The rationale for this is that a single error can have widespread consequences. For example, suppose that the RRs defining a delegation have syntax errors; then the server will return authoritative name errors for all names in the subzone (except in the case where the subzone is also present on the server).

Several other validity checks that should be performed in addition to insuring that the file is syntactically correct:

1. All RRs in the file should have the same class.
2. Exactly one SOA RR should be present at the top of the zone.
3. If delegations are present and glue information is required, it should be present.

4. Information present outside of the authoritative nodes in the zone should be glue information, rather than the result of an origin or similar error.

5.3. Master file example

The following is an example file which might be used to define the ISI.EDU zone and is loaded with an origin of ISI.EDU:

```
@ IN SOA      VENERA      Action\.domains (
                                20      ; SERIAL
                                7200    ; REFRESH
                                600     ; RETRY
                                3600000; EXPIRE
                                60)    ; MINIMUM

      NS      A.ISI.EDU.
      NS      VENERA
      NS      VAXA
      MX      10      VENERA
      MX      20      VAXA

A      A      26.3.0.103

VENERA A      10.1.0.52
      A      128.9.0.32

VAXA   A      10.2.0.27
      A      128.9.0.33
```

```
$INCLUDE <SUBSYS>ISI-MAILBOXES.TXT
```

Where the file <SUBSYS>ISI-MAILBOXES.TXT is:

```
MOE      MB      A.ISI.EDU.
LARRY    MB      A.ISI.EDU.
CURLEY   MB      A.ISI.EDU.
STOOGES  MG      MOE
          MG      LARRY
          MG      CURLEY
```

Note the use of the \ character in the SOA RR to specify the responsible person mailbox "Action.domains@E.ISI.EDU".

6. NAME SERVER IMPLEMENTATION

6.1. Architecture

The optimal structure for the name server will depend on the host operating system and whether the name server is integrated with resolver operations, either by supporting recursive service, or by sharing its database with a resolver. This section discusses implementation considerations for a name server which shares a database with a resolver, but most of these concerns are present in any name server.

6.1.1. Control

A name server must employ multiple concurrent activities, whether they are implemented as separate tasks in the host's OS or multiplexing inside a single name server program. It is simply not acceptable for a name server to block the service of UDP requests while it waits for TCP data for refreshing or query activities. Similarly, a name server should not attempt to provide recursive service without processing such requests in parallel, though it may choose to serialize requests from a single client, or to regard identical requests from the same client as duplicates. A name server should not substantially delay requests while it reloads a zone from master files or while it incorporates a newly refreshed zone into its database.

6.1.2. Database

While name server implementations are free to use any internal data structures they choose, the suggested structure consists of three major parts:

- A "catalog" data structure which lists the zones available to this server, and a "pointer" to the zone data structure. The main purpose of this structure is to find the nearest ancestor zone, if any, for arriving standard queries.
- Separate data structures for each of the zones held by the name server.
- A data structure for cached data. (or perhaps separate caches for different classes)

All of these data structures can be implemented an identical tree structure format, with different data chained off the nodes in different parts: in the catalog the data is pointers to zones, while in the zone and cache data structures, the data will be RRs. In designing the tree framework the designer should recognize that query processing will need to traverse the tree using case-insensitive label comparisons; and that

in real data, a few nodes have a very high branching factor (100-1000 or more), but the vast majority have a very low branching factor (0-1).

One way to solve the case problem is to store the labels for each node in two pieces: a standardized-case representation of the label where all ASCII characters are in a single case, together with a bit mask that denotes which characters are actually of a different case. The branching factor diversity can be handled using a simple linked list for a node until the branching factor exceeds some threshold, and transitioning to a hash structure after the threshold is exceeded. In any case, hash structures used to store tree sections must insure that hash functions and procedures preserve the casing conventions of the DNS.

The use of separate structures for the different parts of the database is motivated by several factors:

- The catalog structure can be an almost static structure that need change only when the system administrator changes the zones supported by the server. This structure can also be used to store parameters used to control refreshing activities.
- The individual data structures for zones allow a zone to be replaced simply by changing a pointer in the catalog. Zone refresh operations can build a new structure and, when complete, splice it into the database via a simple pointer replacement. It is very important that when a zone is refreshed, queries should not use old and new data simultaneously.
- With the proper search procedures, authoritative data in zones will always "hide", and hence take precedence over, cached data.
- Errors in zone definitions that cause overlapping zones, etc., may cause erroneous responses to queries, but problem determination is simplified, and the contents of one "bad" zone can't corrupt another.
- Since the cache is most frequently updated, it is most vulnerable to corruption during system restarts. It can also become full of expired RR data. In either case, it can easily be discarded without disturbing zone data.

A major aspect of database design is selecting a structure which allows the name server to deal with crashes of the name server's host. State information which a name server should save across system crashes

includes the catalog structure (including the state of refreshing for each zone) and the zone data itself.

6.1.3. Time

Both the TTL data for RRs and the timing data for refreshing activities depends on 32 bit timers in units of seconds. Inside the database, refresh timers and TTLs for cached data conceptually "count down", while data in the zone stays with constant TTLs.

A recommended implementation strategy is to store time in two ways: as a relative increment and as an absolute time. One way to do this is to use positive 32 bit numbers for one type and negative numbers for the other. The RRs in zones use relative times; the refresh timers and cache data use absolute times. Absolute numbers are taken with respect to some known origin and converted to relative values when placed in the response to a query. When an absolute TTL is negative after conversion to relative, then the data is expired and should be ignored.

6.2. Standard query processing

The major algorithm for standard query processing is presented in [RFC-1034].

When processing queries with QCLASS=*, or some other QCLASS which matches multiple classes, the response should never be authoritative unless the server can guarantee that the response covers all classes.

When composing a response, RRs which are to be inserted in the additional section, but duplicate RRs in the answer or authority sections, may be omitted from the additional section.

When a response is so long that truncation is required, the truncation should start at the end of the response and work forward in the datagram. Thus if there is any data for the authority section, the answer section is guaranteed to be unique.

The MINIMUM value in the SOA should be used to set a floor on the TTL of data distributed from a zone. This floor function should be done when the data is copied into a response. This will allow future dynamic update protocols to change the SOA MINIMUM field without ambiguous semantics.

6.3. Zone refresh and reload processing

In spite of a server's best efforts, it may be unable to load zone data from a master file due to syntax errors, etc., or be unable to refresh a zone within the its expiration parameter. In this case, the name server

should answer queries as if it were not supposed to possess the zone.

If a master is sending a zone out via AXFR, and a new version is created during the transfer, the master should continue to send the old version if possible. In any case, it should never send part of one version and part of another. If completion is not possible, the master should reset the connection on which the zone transfer is taking place.

6.4. Inverse queries (Optional)

Inverse queries are an optional part of the DNS. Name servers are not required to support any form of inverse queries. If a name server receives an inverse query that it does not support, it returns an error response with the "Not Implemented" error set in the header. While inverse query support is optional, all name servers must be at least able to return the error response.

6.4.1. The contents of inverse queries and responses Inverse queries reverse the mappings performed by standard query operations; while a standard query maps a domain name to a resource, an inverse query maps a resource to a domain name. For example, a standard query might bind a domain name to a host address; the corresponding inverse query binds the host address to a domain name.

Inverse queries take the form of a single RR in the answer section of the message, with an empty question section. The owner name of the query RR and its TTL are not significant. The response carries questions in the question section which identify all names possessing the query RR WHICH THE NAME SERVER KNOWS. Since no name server knows about all of the domain name space, the response can never be assumed to be complete. Thus inverse queries are primarily useful for database management and debugging activities. Inverse queries are NOT an acceptable method of mapping host addresses to host names; use the IN-ADDR.ARPA domain instead.

Where possible, name servers should provide case-insensitive comparisons for inverse queries. Thus an inverse query asking for an MX RR of "Venera.isi.edu" should get the same response as a query for "VENERA.ISI.EDU"; an inverse query for HINFO RR "IBM-PC UNIX" should produce the same result as an inverse query for "IBM-pc unix". However, this cannot be guaranteed because name servers may possess RRs that contain character strings but the name server does not know that the data is character.

When a name server processes an inverse query, it either returns:

1. zero, one, or multiple domain names for the specified resource as QNAMEs in the question section

2. an error code indicating that the name server doesn't support inverse mapping of the specified resource type.

When the response to an inverse query contains one or more QNAMEs, the owner name and TTL of the RR in the answer section which defines the inverse query is modified to exactly match an RR found at the first QNAME.

RRs returned in the inverse queries cannot be cached using the same mechanism as is used for the replies to standard queries. One reason for this is that a name might have multiple RRs of the same type, and only one would appear. For example, an inverse query for a single address of a multiply homed host might create the impression that only one address existed.

6.4.2. Inverse query and response example The overall structure of an inverse query for retrieving the domain name that corresponds to Internet address 10.1.0.52 is shown below:

Header		OPCODE=IQUERY, ID=997	
Question		<empty>	
Answer		<anyname> A IN 10.1.0.52	
Authority		<empty>	
Additional		<empty>	

This query asks for a question whose answer is the Internet style address 10.1.0.52. Since the owner name is not known, any domain name can be used as a placeholder (and is ignored). A single octet of zero, signifying the root, is usually used because it minimizes the length of the message. The TTL of the RR is not significant. The response to this query might be:

Header	OPCODE=RESPONSE, ID=997
Question	QTYPE=A, QCLASS=IN, QNAME=VENERA.ISI.EDU
Answer	VENERA.ISI.EDU A IN 10.1.0.52
Authority	<empty>
Additional	<empty>

Note that the QTYPE in a response to an inverse query is the same as the TYPE field in the answer section of the inverse query. Responses to inverse queries may contain multiple questions when the inverse is not unique. If the question section in the response is not empty, then the RR in the answer section is modified to correspond to be an exact copy of an RR at the first QNAME.

6.4.3. Inverse query processing

Name servers that support inverse queries can support these operations through exhaustive searches of their databases, but this becomes impractical as the size of the database increases. An alternative approach is to invert the database according to the search key.

For name servers that support multiple zones and a large amount of data, the recommended approach is separate inversions for each zone. When a particular zone is changed during a refresh, only its inversions need to be redone.

Support for transfer of this type of inversion may be included in future versions of the domain system, but is not supported in this version.

6.5. Completion queries and responses

The optional completion services described in RFC-882 and RFC-883 have been deleted. Redesigned services may become available in the future.

7. RESOLVER IMPLEMENTATION

The top levels of the recommended resolver algorithm are discussed in [RFC-1034]. This section discusses implementation details assuming the database structure suggested in the name server implementation section of this memo.

7.1. Transforming a user request into a query

The first step a resolver takes is to transform the client's request, stated in a format suitable to the local OS, into a search specification for RRs at a specific name which match a specific QTYPE and QCLASS. Where possible, the QTYPE and QCLASS should correspond to a single type and a single class, because this makes the use of cached data much simpler. The reason for this is that the presence of data of one type in a cache doesn't confirm the existence or non-existence of data of other types, hence the only way to be sure is to consult an authoritative source. If QCLASS=* is used, then authoritative answers won't be available.

Since a resolver must be able to multiplex multiple requests if it is to perform its function efficiently, each pending request is usually represented in some block of state information. This state block will typically contain:

- A timestamp indicating the time the request began.
The timestamp is used to decide whether RRs in the database can be used or are out of date. This timestamp uses the absolute time format previously discussed for RR storage in zones and caches. Note that when an RRs TTL indicates a relative time, the RR must be timely, since it is part of a zone. When the RR has an absolute time, it is part of a cache, and the TTL of the RR is compared against the timestamp for the start of the request.
- Note that using the timestamp is superior to using a current time, since it allows RRs with TTLs of zero to be entered in the cache in the usual manner, but still used by the current request, even after intervals of many seconds due to system load, query retransmission timeouts, etc.
- Some sort of parameters to limit the amount of work which will be performed for this request.

The amount of work which a resolver will do in response to a client request must be limited to guard against errors in the database, such as circular CNAME references, and operational problems, such as network partition which prevents the

resolver from accessing the name servers it needs. While local limits on the number of times a resolver will retransmit a particular query to a particular name server address are essential, the resolver should have a global per-request counter to limit work on a single request. The counter should be set to some initial value and decremented whenever the resolver performs any action (retransmission timeout, retransmission, etc.) If the counter passes zero, the request is terminated with a temporary error.

Note that if the resolver structure allows one request to start others in parallel, such as when the need to access a name server for one request causes a parallel resolve for the name server's addresses, the spawned request should be started with a lower counter. This prevents circular references in the database from starting a chain reaction of resolver activity.

- The SLIST data structure discussed in [RFC-1034].

This structure keeps track of the state of a request if it must wait for answers from foreign name servers.

7.2. Sending the queries

As described in [RFC-1034], the basic task of the resolver is to formulate a query which will answer the client's request and direct that query to name servers which can provide the information. The resolver will usually only have very strong hints about which servers to ask, in the form of NS RRs, and may have to revise the query, in response to CNAMEs, or revise the set of name servers the resolver is asking, in response to delegation responses which point the resolver to name servers closer to the desired information. In addition to the information requested by the client, the resolver may have to call upon its own services to determine the address of name servers it wishes to contact.

In any case, the model used in this memo assumes that the resolver is multiplexing attention between multiple requests, some from the client, and some internally generated. Each request is represented by some state information, and the desired behavior is that the resolver transmit queries to name servers in a way that maximizes the probability that the request is answered, minimizes the time that the request takes, and avoids excessive transmissions. The key algorithm uses the state information of the request to select the next name server address to query, and also computes a timeout which will cause the next action should a response not arrive. The next action will usually be a transmission to some other server, but may be a temporary error to the

client.

The resolver always starts with a list of server names to query (SLIST). This list will be all NS RRs which correspond to the nearest ancestor zone that the resolver knows about. To avoid startup problems, the resolver should have a set of default servers which it will ask should it have no current NS RRs which are appropriate. The resolver then adds to SLIST all of the known addresses for the name servers, and may start parallel requests to acquire the addresses of the servers when the resolver has the name, but no addresses, for the name servers.

To complete initialization of SLIST, the resolver attaches whatever history information it has to the each address in SLIST. This will usually consist of some sort of weighted averages for the response time of the address, and the batting average of the address (i.e., how often the address responded at all to the request). Note that this information should be kept on a per address basis, rather than on a per name server basis, because the response time and batting average of a particular server may vary considerably from address to address. Note also that this information is actually specific to a resolver address / server address pair, so a resolver with multiple addresses may wish to keep separate histories for each of its addresses. Part of this step must deal with addresses which have no such history; in this case an expected round trip time of 5-10 seconds should be the worst case, with lower estimates for the same local network, etc.

Note that whenever a delegation is followed, the resolver algorithm reinitializes SLIST.

The information establishes a partial ranking of the available name server addresses. Each time an address is chosen and the state should be altered to prevent its selection again until all other addresses have been tried. The timeout for each transmission should be 50-100% greater than the average predicted value to allow for variance in response.

Some fine points:

- The resolver may encounter a situation where no addresses are available for any of the name servers named in SLIST, and where the servers in the list are precisely those which would normally be used to look up their own addresses. This situation typically occurs when the glue address RRs have a smaller TTL than the NS RRs marking delegation, or when the resolver caches the result of a NS search. The resolver should detect this condition and restart the search at the next ancestor zone, or alternatively at the root.

- If a resolver gets a server error or other bizarre response from a name server, it should remove it from SLIST, and may wish to schedule an immediate transmission to the next candidate server address.

7.3. Processing responses

The first step in processing arriving response datagrams is to parse the response. This procedure should include:

- Check the header for reasonableness. Discard datagrams which are queries when responses are expected.
- Parse the sections of the message, and insure that all RRs are correctly formatted.
- As an optional step, check the TTLs of arriving data looking for RRs with excessively long TTLs. If a RR has an excessively long TTL, say greater than 1 week, either discard the whole response, or limit all TTLs in the response to 1 week.

The next step is to match the response to a current resolver request. The recommended strategy is to do a preliminary matching using the ID field in the domain header, and then to verify that the question section corresponds to the information currently desired. This requires that the transmission algorithm devote several bits of the domain ID field to a request identifier of some sort. This step has several fine points:

- Some name servers send their responses from different addresses than the one used to receive the query. That is, a resolver cannot rely that a response will come from the same address which it sent the corresponding query to. This name server bug is typically encountered in UNIX systems.
- If the resolver retransmits a particular request to a name server it should be able to use a response from any of the transmissions. However, if it is using the response to sample the round trip time to access the name server, it must be able to determine which transmission matches the response (and keep transmission times for each outgoing message), or only calculate round trip times based on initial transmissions.
- A name server will occasionally not have a current copy of a zone which it should have according to some NS RRs. The resolver should simply remove the name server from the current SLIST, and continue.

7.4. Using the cache

In general, we expect a resolver to cache all data which it receives in responses since it may be useful in answering future client requests. However, there are several types of data which should not be cached:

- When several RRs of the same type are available for a particular owner name, the resolver should either cache them all or none at all. When a response is truncated, and a resolver doesn't know whether it has a complete set, it should not cache a possibly partial set of RRs.
- Cached data should never be used in preference to authoritative data, so if caching would cause this to happen the data should not be cached.
- The results of an inverse query should not be cached.
- The results of standard queries where the QNAME contains "*" labels if the data might be used to construct wildcards. The reason is that the cache does not necessarily contain existing RRs or zone boundary information which is necessary to restrict the application of the wildcard RRs.
- RR data in responses of dubious reliability. When a resolver receives unsolicited responses or RR data other than that requested, it should discard it without caching it. The basic implication is that all sanity checks on a packet should be performed before any of it is cached.

In a similar vein, when a resolver has a set of RRs for some name in a response, and wants to cache the RRs, it should check its cache for already existing RRs. Depending on the circumstances, either the data in the response or the cache is preferred, but the two should never be combined. If the data in the response is from authoritative data in the answer section, it is always preferred.

8. MAIL SUPPORT

The domain system defines a standard for mapping mailboxes into domain names, and two methods for using the mailbox information to derive mail routing information. The first method is called mail exchange binding and the other method is mailbox binding. The mailbox encoding standard and mail exchange binding are part of the DNS official protocol, and are the recommended method for mail routing in the Internet. Mailbox binding is an experimental feature which is still under development and subject to change.

The mailbox encoding standard assumes a mailbox name of the form "<local-part>@<mail-domain>". While the syntax allowed in each of these sections varies substantially between the various mail internets, the preferred syntax for the ARPA Internet is given in [RFC-822].

The DNS encodes the <local-part> as a single label, and encodes the <mail-domain> as a domain name. The single label from the <local-part> is prefixed to the domain name from <mail-domain> to form the domain name corresponding to the mailbox. Thus the mailbox HOSTMASTER@SRI-NIC.ARPA is mapped into the domain name HOSTMASTER.SRI-NIC.ARPA. If the <local-part> contains dots or other special characters, its representation in a master file will require the use of backslash quoting to ensure that the domain name is properly encoded. For example, the mailbox Action.domains@ISI.EDU would be represented as Action\.domains.ISI.EDU.

8.1. Mail exchange binding

Mail exchange binding uses the <mail-domain> part of a mailbox specification to determine where mail should be sent. The <local-part> is not even consulted. [RFC-974] specifies this method in detail, and should be consulted before attempting to use mail exchange support.

One of the advantages of this method is that it decouples mail destination naming from the hosts used to support mail service, at the cost of another layer of indirection in the lookup function. However, the addition layer should eliminate the need for complicated "%", "!", etc encodings in <local-part>.

The essence of the method is that the <mail-domain> is used as a domain name to locate type MX RRs which list hosts willing to accept mail for <mail-domain>, together with preference values which rank the hosts according to an order specified by the administrators for <mail-domain>.

In this memo, the <mail-domain> ISI.EDU is used in examples, together with the hosts VENERA.ISI.EDU and VAXA.ISI.EDU as mail exchanges for ISI.EDU. If a mailer had a message for Mockapetris@ISI.EDU, it would route it by looking up MX RRs for ISI.EDU. The MX RRs at ISI.EDU name VENERA.ISI.EDU and VAXA.ISI.EDU, and type A queries can find the host addresses.

8.2. Mailbox binding (Experimental)

In mailbox binding, the mailer uses the entire mail destination specification to construct a domain name. The encoded domain name for the mailbox is used as the QNAME field in a QTYPE=MAILB query.

Several outcomes are possible for this query:

1. The query can return a name error indicating that the mailbox does not exist as a domain name.

In the long term, this would indicate that the specified mailbox doesn't exist. However, until the use of mailbox binding is universal, this error condition should be interpreted to mean that the organization identified by the global part does not support mailbox binding. The appropriate procedure is to revert to exchange binding at this point.

2. The query can return a Mail Rename (MR) RR.

The MR RR carries new mailbox specification in its RDATA field. The mailer should replace the old mailbox with the new one and retry the operation.

3. The query can return a MB RR.

The MB RR carries a domain name for a host in its RDATA field. The mailer should deliver the message to that host via whatever protocol is applicable, e.g., b,SMTP.

4. The query can return one or more Mail Group (MG) RRs.

This condition means that the mailbox was actually a mailing list or mail group, rather than a single mailbox. Each MG RR has a RDATA field that identifies a mailbox that is a member of the group. The mailer should deliver a copy of the message to each member.

5. The query can return a MB RR as well as one or more MG RRs.

This condition means the the mailbox was actually a mailing list. The mailer can either deliver the message to the host specified by the MB RR, which will in turn do the delivery to all members, or the mailer can use the MG RRs to do the expansion itself.

In any of these cases, the response may include a Mail Information (MINFO) RR. This RR is usually associated with a mail group, but is legal with a MB. The MINFO RR identifies two mailboxes. One of these identifies a responsible person for the original mailbox name. This mailbox should be used for requests to be added to a mail group, etc. The second mailbox name in the MINFO RR identifies a mailbox that should receive error messages for mail failures. This is particularly appropriate for mailing lists when errors in member names should be reported to a person other than the one who sends a message to the list.

New fields may be added to this RR in the future.

9. REFERENCES and BIBLIOGRAPHY

- [Dyer 87] S. Dyer, F. Hsu, "Hesiod", Project Athena
Technical Plan - Name Service, April 1987, version 1.9.

Describes the fundamentals of the Hesiod name service.
- [IEN-116] J. Postel, "Internet Name Server", IEN-116,
USC/Information Sciences Institute, August 1979.

A name service obsoleted by the Domain Name System, but
still in use.
- [Quarterman 86] J. Quarterman, and J. Hoskins, "Notable Computer Networks",
Communications of the ACM, October 1986, volume 29, number
10.
- [RFC-742] K. Harrenstien, "NAME/FINGER", RFC-742, Network
Information Center, SRI International, December 1977.
- [RFC-768] J. Postel, "User Datagram Protocol", RFC-768,
USC/Information Sciences Institute, August 1980.
- [RFC-793] J. Postel, "Transmission Control Protocol", RFC-793,
USC/Information Sciences Institute, September 1981.
- [RFC-799] D. Mills, "Internet Name Domains", RFC-799, COMSAT,
September 1981.

Suggests introduction of a hierarchy in place of a flat
name space for the Internet.
- [RFC-805] J. Postel, "Computer Mail Meeting Notes", RFC-805,
USC/Information Sciences Institute, February 1982.
- [RFC-810] E. Feinler, K. Harrenstien, Z. Su, and V. White, "DOD
Internet Host Table Specification", RFC-810, Network
Information Center, SRI International, March 1982.

Obsolete. See RFC-952.
- [RFC-811] K. Harrenstien, V. White, and E. Feinler, "Hostnames
Server", RFC-811, Network Information Center, SRI
International, March 1982.

Obsolete. See RFC-953.

- [RFC-812] K. Harrenstien, and V. White, "NICNAME/WHOIS", RFC-812, Network Information Center, SRI International, March 1982.
- [RFC-819] Z. Su, and J. Postel, "The Domain Naming Convention for Internet User Applications", RFC-819, Network Information Center, SRI International, August 1982.
- Early thoughts on the design of the domain system. Current implementation is completely different.
- [RFC-821] J. Postel, "Simple Mail Transfer Protocol", RFC-821, USC/Information Sciences Institute, August 1980.
- [RFC-830] Z. Su, "A Distributed System for Internet Name Service", RFC-830, Network Information Center, SRI International, October 1982.
- Early thoughts on the design of the domain system. Current implementation is completely different.
- [RFC-882] P. Mockapetris, "Domain names - Concepts and Facilities," RFC-882, USC/Information Sciences Institute, November 1983.
- Superceded by this memo.
- [RFC-883] P. Mockapetris, "Domain names - Implementation and Specification," RFC-883, USC/Information Sciences Institute, November 1983.
- Superceded by this memo.
- [RFC-920] J. Postel and J. Reynolds, "Domain Requirements", RFC-920, USC/Information Sciences Institute, October 1984.
- Explains the naming scheme for top level domains.
- [RFC-952] K. Harrenstien, M. Stahl, E. Feinler, "DoD Internet Host Table Specification", RFC-952, SRI, October 1985.
- Specifies the format of HOSTS.TXT, the host/address table replaced by the DNS.

[RFC-953] K. Harrenstien, M. Stahl, E. Feinler, "HOSTNAME Server", RFC-953, SRI, October 1985.

This RFC contains the official specification of the hostname server protocol, which is obsoleted by the DNS. This TCP based protocol accesses information stored in the RFC-952 format, and is used to obtain copies of the host table.

[RFC-973] P. Mockapetris, "Domain System Changes and Observations", RFC-973, USC/Information Sciences Institute, January 1986.

Describes changes to RFC-882 and RFC-883 and reasons for them.

[RFC-974] C. Partridge, "Mail routing and the domain system", RFC-974, CSNET CIC BBN Labs, January 1986.

Describes the transition from HOSTS.TXT based mail addressing to the more powerful MX system used with the domain system.

[RFC-1001] NetBIOS Working Group, "Protocol standard for a NetBIOS service on a TCP/UDP transport: Concepts and Methods", RFC-1001, March 1987.

This RFC and RFC-1002 are a preliminary design for NETBIOS on top of TCP/IP which proposes to base NetBIOS name service on top of the DNS.

[RFC-1002] NetBIOS Working Group, "Protocol standard for a NetBIOS service on a TCP/UDP transport: Detailed Specifications", RFC-1002, March 1987.

[RFC-1010] J. Reynolds, and J. Postel, "Assigned Numbers", RFC-1010, USC/Information Sciences Institute, May 1987.

Contains socket numbers and mnemonics for host names, operating systems, etc.

[RFC-1031] W. Lazear, "MILNET Name Domain Transition", RFC-1031, November 1987.

Describes a plan for converting the MILNET to the DNS.

[RFC-1032] M. Stahl, "Establishing a Domain - Guidelines for Administrators", RFC-1032, November 1987.

Describes the registration policies used by the NIC to administer the top level domains and delegate subzones.

[RFC-1033]

M. Lottor, "Domain Administrators Operations Guide", RFC-1033, November 1987.

A cookbook for domain administrators.

[Solomon 82]

M. Solomon, L. Landweber, and D. Neuhengen, "The CSNET Name Server", Computer Networks, vol 6, nr 3, July 1982.

Describes a name service for CSNET which is independent from the DNS and DNS use in the CSNET.

Index

* 13

; 33, 35

<character-string> 35

<domain-name> 34

@ 35

\ 35

A 12

Byte order 8

CH 13

Character case 9

CLASS 11

CNAME 12

Completion 42

CS 13

Hesiod 13

HINFO 12

HS 13

IN 13

IN-ADDR.ARPA domain 22

Inverse queries 40

Mailbox names 47

MB 12

MD 12

MF 12

MG 12

MINFO 12

MINIMUM 20

MR 12

MX 12

NS 12

NULL 12

Port numbers 32

Primary server 5

PTR 12, 18

QCLASS 13
QTYPE 12

RDATA 12
RDLENGTH 11

Secondary server 5
SOA 12
Stub resolvers 7

TCP 32
TXT 12
TYPE 11

UDP 32

WKS 12

