                        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
5   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
10   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.
15

·   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
20   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
25   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:
30

·   - The definition of zone boundaries.
·
·   - Master files of data.
·
35   - Updates to master files.
·
·   - Statements of the refresh policies desired.
·
·   The domain system provides:
40

·   - Standard formats for resource data.
·
·   - Standard methods for querying the database.
·
45   - 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 (regardles 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.

```
                                        |
                                        |
                +--------------------+-----------------+
                |                    |                 |
                MIL                  EDU               ARPA
                |                    |                 |
                |                    |                 |
            +-----+-----+           |      +------+-----+-----+
            |     |     |           |      |      |     |     |
            BRL   NOSC  DARPA       |   IN-ADDR  SRI-NIC   ACC
                                    |
            +--------+-----------------+--------------+--------+
            |        |                 |              |        |
            UCI      MIT               |              UDEL     YALE
                     |                ISI
                     |                 |
                 +---+---+             |
                 |       |             |
                LCS   ACHILLES  +--+-----+-----+--------+
                 |             |  |     |     |        |
                 XX           A   C   VAXA  VENERA Mockapetris
```

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, an 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).
·
5   AXFR   special zone transfer QTYPE.
·
·   MAILB   matches all mail box related RRs (e.g. MB and MG).
·
·   *   matches all RR types.
10
·   The QCLASS field may contain:
·
·   <any class>   matches just that class (e.g., IN, CH).
·
15   *   matches aLL RR classes.
·
17   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
20   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.
25
·   For example, a mailer tying 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:
30
·   ISI.EDU.   MX   10 VENERA.ISI.EDU.
·   MX   10 VAXA.ISI.EDU.
·
·   while the additional section might be:
35
·   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
40
·   Because the server assumes that if the requester wants mail exchange
·   information, it will probably want the addresses of the mail exchanges
·   soon afterward.
·
45   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.
5
· 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.
10
· 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
15 so that the requester can tell whether the response comes from
· authoritative data or not.
·
· 4.2. How the database is divided into zones
·
20 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
25 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
30 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
35 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
40 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.
·
45 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
5 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
10 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
12 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.
15 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

$21 - 12 - 8,7 \longrightarrow s'$

·
20 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
25 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 an discusses administration procedures.
30
· 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
35 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
40 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
45 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 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

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.

 5   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.

10   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.
15

       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.
20

       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
25

   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
30 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
35 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
40 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
45 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:
 5
·   - 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
10   the query name is know 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.
15
·   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.
20
·   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
25 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
30
·   *.X.COM   MX   10   A.X.COM
·
·   A.X.COM   A   1.2.3.4
·   A.X.COM   MX   10   A.X.COM
35
·   *.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
40 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.
·
45 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 beacuse 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 an 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 occurances. 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:

                                        |(C.ISI.EDU,SRI-NIC.ARPA
                                        | A.ISI.EDU)
                +---------------------+-----------------+
                | | |
               MIL   EDU   ARPA
                |(SRI-NIC.ARPA, |(SRI-NIC.ARPA, |
                | A.ISI.EDU | C.ISI.EDU) |
            +-----+-----+ | +------+-----+-----+
            | | | | | | | |
           BRL   NOSC DARPA | IN-ADDR SRI-NIC   ACC
                                       |
            +--------+------------------+--------------+--------+
            | | | | |
           UCI   MIT | UDEL   YALE
                     |(XX.LCS.MIT.EDU, ISI
                     |ACHILLES.MIT.EDU) |(VAXA.ISI.EDU,VENERA.ISI.EDU,
                  +---+---+ | A.ISI.EDU)
                  | | |
                 LCS   ACHILLES +--+-----+-----+--------+
                  | | | | | |
                 XX   A C   VAXA VENERA Mockapetris

In this example, the authoritative name server is shown in parentheses
at the point in the domain tree at which is 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
 5
· 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
10 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
15 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
20 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.
25
· 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.
30
· 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. (
35                              870729 ;serial
·                              1800 ;refresh every 30 minutes
·                              300 ;retry every 5 minutes
·                              604800 ;expire after a week
·                              86400 ;minimum of a day
40                              )
·                      NS SRI-NIC.ARPA.
·                      NS C.ISI.EDU.
·
·      UCI 172800 NS ICS.UCI
45                    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
5                       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
·
10      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
·
15      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.
20      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.
25 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.
30 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:
·
·   +---+
5   Header | OPCODE=SQUERY, RESPONSE |
·   +---+
·   Question | QNAME=SRI-NIC.ARPA., QCLASS=IN, QTYPE=* |
·   +---+
·   Answer | SRI-NIC.ARPA. 12345 IN   A   26.0.0.73 |
10 | A   10.0.0.51 |
·   +---+
·   Authority | <empty> |
·   +---+
·   Additional | <empty> |
15   +---+
·
·   and
·
·   +---+
20   Header | OPCODE=SQUERY, RESPONSE |
·   +---+
·   Question | QNAME=SRI-NIC.ARPA., QCLASS=IN, QTYPE=* |
·   +---+
·   Answer | SRI-NIC.ARPA. 1290 IN HINFO DEC-2060 TOPS20 |
25   +---+
·   Authority | <empty> |
·   +---+
·   Additional | <empty> |
·   +---+
30
·   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
35   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=SIR-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

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:
5

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

·
10 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.
·
15 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.
·
20 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
25 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
30 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
35 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       |
                  |                 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      |
                  +----------------------------------------------------+
```

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.
·
5   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
10   Institute, November 1983.
·
·   Superceeded by this memo.
·
·   [RFC-883]   P. Mockapetris, "Domain names - Implementation and
15   Specification," RFC-883, USC/Information Sciences
·   Institute, November 1983.
·
·   Superceeded by this memo.
·
20 [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.
25
·   [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
30   table replaced by the DNS.
·
·   [RFC-953]   K. Harrenstien, M. Stahl, E. Feinler, "HOSTNAME Server",
·   RFC-953, SRI, October 1985.
·
35   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.
40
·   [RFC-973]   P. Mockapetris, "Domain System Changes and
·   Observations", RFC-973, USC/Information Sciences
·   Institute, January 1986.
·
45   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
5   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",
10   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.
15
·   [RFC-1002]   NetBIOS Working Group, "Protocol standard for a NetBIOS
·   service on a TCP/UDP transport: Detailed
·   Specifications", RFC-1002, March 1987.
·
20 [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.
25
·   [RFC-1031]   W. Lazear, "MILNET Name Domain Transition", RFC-1031,
·   November 1987.
·
·   Describes a plan for converting the MILNET to the DNS.
30
·   [RFC-1032]   M. K. Stahl, "Establishing a Domain - Guidelines for
·   Administrators", RFC-1032, November 1987.
·
·   Describes the registration policies used by the NIC to
35   administer the top level domains and delegate subzones.
·
·   [RFC-1033]   M. K. Lottor, "Domain Administrators Operations Guide",
·   RFC-1033, November 1987.
·
40   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.
·
45   Describes a name service for CSNET which is independent
·   from the DNS and DNS use in the CSNET.

· Index
·
·   A   12
·   Absolute names   8
5   Aliases   14, 31
·   Authority   6
·   AXFR   17
·
·   Case of characters   7
10   CH   12
·   CNAME   12, 13, 31
·   Completion queries   18
·
·   Domain name   6, 7
15
·   Glue RRs   20
·
·   HINFO   12
·
20   IN   12
·   Inverse queries   16
·   Iterative   4
·
·   Label   7
25
·   Mailbox names   9
·   MX   12
·
·   Name error   27, 36
30   Name servers   5, 17
·   NE   30
·   Negative caching   44
·   NS   12
·
35   Opcode   16
·
·   PTR   12
·
·   QCLASS   16
40   QTYPE   16
·
·   RDATA   13
·   Recursive   4
·   Recursive service   22
45   Relative names   7
·   Resolvers   6
·   RR   12