Network Working Group
Request for Comments: 882

P. Mockapetris
ISI

November 1983

DOMAIN NAMES - CONCEPTS and FACILITIES

| This RFC introduces domain style names, their use for ARPA Internet mail and host address support, and the protocols and servers used to implement domain name facilities.
|
| This memo describes the conceptual framework of the domain system and some uses, but it omits many uses, fields, and implementation details. A complete specification of formats, timeouts, etc. is presented in RFC 883, "Domain Names - Implementation and Specification". That RFC assumes that the reader is familiar with the concepts discussed in this memo.

INTRODUCTION

The need for domain names

As applications grow to span multiple hosts, then networks, and finally internets, these applications must also span multiple administrative boundaries and related methods of operation (protocols, data formats, etc). The number of resources (for example mailboxes), the number of locations for resources, and the diversity of such an environment cause formidable problems when we wish to create consistent methods for referencing particular resources that are similar but scattered throughout the environment.

The ARPA Internet illustrates the size-related problems; it is a large system and is likely to grow much larger. The need to have a mapping between host names (e.g., USC-ISIF) and ARPA Internet addresses (e.g., 10.2.0.52) is beginning to stress the existing mechanisms. Currently hosts in the ARPA Internet are registered with the Network Information Center (NIC) and listed in a global table (available as the file <NETINFO>HOSTS.TXT on the SRI-NIC host) [1]. The size of this table, and especially the frequency of updates to the table are near the limit of manageability. What is needed is a distributed database that performs the same function, and hence avoids the problems caused by a centralized database.

The problem for computer mail is more severe. While mail system implementers long ago recognized the impossibility of centralizing

mailbox names, they have also created an increasingly large and irregular set of methods for identifying the location of a mailbox. Some of these methods involve the use of routes and forwarding hosts as part of the mail destination address, and consequently force the mail user to know multiple address formats, the capabilities of various forwarders, and ad hoc tricks for passing address specifications through intermediaries.

These problems have common characteristics that suggest the nature of any solution:

The basic need is for 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 contain 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.

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.

Because we want the name space to be useful in dissimilar networks, it is unlikely that all users of domain names will be able to agree on the set of resources or resource information that names will be used to retrieve. Hence names refer to a set of resources, and queries contain resource identifiers. The only standard types of information that we expect to see throughout the name space is structuring information for the name space itself, and resources that are described using domain names and no nonstandard data.

We also want the name server transactions to be independent of the communications system that carries them. Some systems may wish to use datagrams for simple 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.

Elements of the solution

The proposed solution has three major components:

The DOMAIN NAME SPACE, which is a specification for a tree structured name space. 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 ARPA Internet uses some of its domain names to identify hosts; queries for address resources return ARPA Internet host addresses. However, to preserve the generality of the domain mechanism, domain names are not required to have a one-to-one correspondence with host names, host addresses, or any other type of information.

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; these parts are called ZONEs; a name server is an AUTHORITY for these parts of the name space.

RESOLVERS are programs that extract information from name servers in response to user 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 simple procedure or OS calls to resolvers. 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 using the local zones.

In the interests of performance, these layers blur a bit. For example, resolvers on the same machine as a name server may share a database and may also introduce foreign information for use in later queries. This cached information is treated differently from the authoritative data in zones.

Database model

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 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 though 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.

Some users will wish to access the database via datagrams; others will prefer virtual circuits. The domain system is designed so that simple queries and responses can use either style, although refreshing operations need the reliability of virtual circuits. The same overall message format is used for all communication. The domain system does not assume any special properties of the communications system, and hence could be used with any datagram or virtual circuit protocol.

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 optional recursive style is discussed in [14], and omitted from further discussion in this memo.

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 to users of the domain system. A standard format for these files is given in [14].

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

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

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

The system administrators provide:

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

The domain system provides:

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

DOMAIN NAME SPACE

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). Each node and leaf has an associated label. Labels are NOT guaranteed to be unique, with the exception of the root node, which has a null label. The domain name of a node or leaf is the path from the root of the tree to the node or leaf. By convention, the labels that compose a domain name are read left to right, from the most specific (lowest) to the least specific (highest).

Internally, programs that manipulate domain names 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. When domain names are printed, labels in a path are separated by dots ("."). The root label and its associated dot are omitted from printed domain names, but the root can be named by a null domain name (" " in this memo).

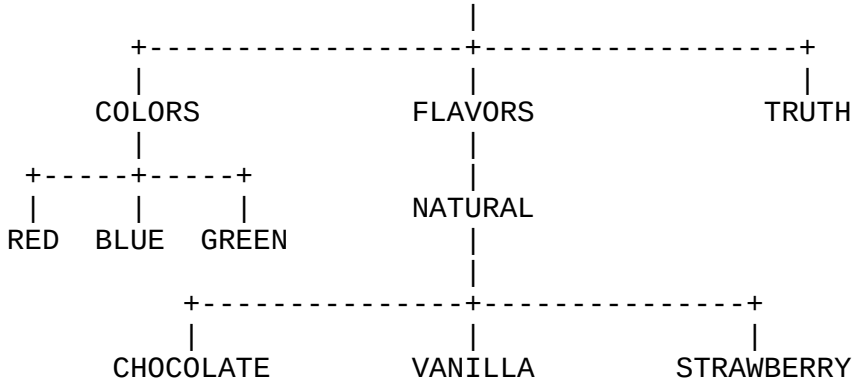
To simplify implementations, the total number of octets that represent label octets and label lengths is limited to 255. Thus a printed domain name can be up to 254 characters.

A special label is defined that matches any other label. This label is the asterisk or "*". An asterisk matches a single label. Thus *.ARPA matches FOO.ARPA, but does not match FOO.BAR.ARPA. The asterisk is mainly used to create default resource records at the boundary between protocol families, and requires prudence in its use.

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 has the containing domain's name as the right part of its name. For example, A.B.C.D is a subdomain of B.C.D, C.D, D, and " ".

This tree structure is intended to parallel the administrative organization and delegation of authority. Potentially, each node or leaf on the tree can create new subdomains ad infinitum. In practice, this delegation can be limited by the administrator of the name servers that manage the domain space and resource data.

The following figure shows an example of a domain name space.



In this example, the root domain has three immediate subdomains: COLORS, FLAVORS, and TRUTH. The FLAVORS domain has one immediate subdomain named NATURAL.FLAVORS. All of the leaves are also

domains. This domain tree has the names " "(the root), COLORS, RED.COLORS, BLUE.COLORS, GREEN.COLORS, FLAVORS, NATURAL.FLAVORS, CHOCOLATE.NATURAL.FLAVORS, VANILLA.NATURAL.FLAVORS, STRAWBERRY.NATURAL.FLAVORS, and TRUTH. If we wished to add a new domain of ARTIFICIAL under FLAVORS, FLAVORS would typically be the administrative entity that would decide; if we wished to create CHIP and MOCHA names under CHOCOLATE, CHOCOLATE.NATURAL.FLAVORS would typically be the appropriate administrative entity.

Resource set information

A domain name identifies a set of resource information. The set of resource information associated with a particular name is composed of separate resource records (RRs).

Each resource record has the following major components:

The domain name which identifies resource set that holds this record, and hence the "owner" of the information. For example, a RR that specifies a host address has a domain name the specifies the host having that address. Thus F.ISI.ARPA might be the owner of a RR which specified an address field of 10.2.0.52. Since name servers typically store their resource information in tree structures paralleling the organization of the domain space, this information can usually be stored implicitly in the database; however it is always included in each resource record carried in a message.

Other information used to manage the RR, such as length fields, timeouts, etc. This information is omitted in much of this memo, but is discussed in [14].

A resource type field that specifies the type of the resource in this resource record. Types refer to abstract resources such as host addresses or mail delivery agents. The type field is two octets long and uses an encoding that is standard throughout the domain name system.

A class field identifies the format of the resource data, such as the ARPA Internet format (IN) or the Computer Science Network format (CSNET), for certain RR types (such as address data). Note that while the class may separate different protocol families, networks, etc. it does not do so in all cases. For example, the IN class uses 32 bit IP addresses exclusively, but the CSNET class uses 32 bit IP addresses, X.25 addresses, and phone numbers. Thus the class field should be used as a guide for interpreting the resource data. The class field is two octets long and uses an encoding that is standard throughout the domain name system.

Resource data that describes the resource. The format of this data can be determined given the type and class fields, but always starts with a two octet length field that allows a name server or resolver to determine the boundaries of the resource data in any transaction, even if it cannot "understand" the resource data itself. Thus name servers and resolvers can hold and pass on records which they cannot interpret. The format of the internal data is restricted only by the maximum length of 65535 octets; for example the host address record might specify a fixed 32 bit number for one class, and a variable length list of addresses in another class.

While the class field in effect partitions the resource data in the domain name system into separate parallel sections according to class, services can span class boundaries if they use compatible resource data formats. For example, the domain name system uses compatible formats for structure information, and the mail data decouples mail agent identification from details of how to contact the agent (e.g. host addresses).

This memo uses the following types in its examples:

A - the host address associated with the domain name

MF - identifies a mail forwarder for the domain

MD - identifies a mail destination for the domain

NS - the authoritative name server for the domain

SOA - identifies the start of a zone of authority

CNAME - identifies the canonical name of an alias

This memo uses the following classes in its examples:

IN - the ARPA Internet system

CS - the CSNET system

The first type of resource record holds a host name to host address binding. Its fields are:

```
+--------+--------+--------+--------------//----------------------+
|<owner> |   A    | <class>| <class specific address>information  |
+--------+--------+--------+--------------//----------------------+
```

The content of the class specific information varies according to the value in the CLASS field; for the ARPA Internet, it is the 32 bit ARPA Internet address of the host, for the CSNET it might be the phone number of the host. For example, F.ISI.ARPA might have two A records of the form:

```
+----------+--------+--------+----------------------------+
|F.ISI.ARPA|   A    |   IN   |          10.2.0.52         |
+----------+--------+--------+----------------------------+
                           and
+----------+--------+--------+----------------------------+
|F.ISI.ARPA|   A    |   CS   |         213-822-2112       |
+----------+--------+--------+----------------------------+
```

Note that the data formats for the A type are class dependent, and the Internet address and phone number formats shown above are for purposes of illustration only. The actual data formats are specified in [14]. For example, CS class data for type A records might actually be a list of Internet addresses, phone numbers and TELENET addresses.

The mail forwarder (MF) and mail delivery (MD) records have the following format:

```
 +--------+--------+--------+----------------------------+
 |<owner> | MD/MF  | <class>|       <domain name>        |
 +--------+--------+--------+----------------------------+
```

The <domain name> field is a domain name of the host that will handle mail; note that this domain name may be completely different from the domain name which names the resource record. For example, F.ISI.ARPA might have two records of the form:

```
+----------+--------+--------+----------------------------+
|F.ISI.ARPA|  MD    |   IN   |         F.ISI.ARPA         |
+----------+--------+--------+----------------------------+
                           and
+----------+--------+--------+----------------------------+
|F.ISI.ARPA|  MF    |   IN   |         B.ISI.ARPA         |
+----------+--------+--------+----------------------------+
```

These records mean that mail for F.ISI.ARPA can either be delivered to the host F.ISI.ARPA or forwarded to B.ISI.ARPA, which will accept responsibility for its eventual delivery. In principle, an additional name lookup is required to map the domain name of the host to the appropriate address, in practice this information is usually returned in the response to the mail query.

The SOA and NS types of resource records are used to define limits

of authority. The domain name given by the owner field of a SOA record is the start of a zone; the domain name given by the owner field of a NS record identifies a point in the name space where authority has been delegated, and hence marks the zone boundary. Except in the case where a name server delegates authority to itself, the SOA identifies the top limit of authority, and NS records define the first name outside of a zone. These resource records have a standard format for all of the name space:

```
+----------+--------+--------+-----------------------------+
| <owner>  |   SOA  | <class>|       <domain name, etc>    |
+----------+--------+--------+-----------------------------+

+----------+--------+--------+-----------------------------+
| <owner>  |   NS   | <class>|       <domain name>         |
+----------+--------+--------+-----------------------------+
```

The SOA record marks the start of a zone when it is present in a database; the NS record both marks the end of a zone started by an SOA (if a higher SOA is present) and also points to a name server that has a copy of the zone specified by the <owner. field of the NS record.

The <domain name, etc> in the SOA record specifies the original source of the information in the zone and other information used by name servers to organize their activities. SOA records are never cached (otherwise they would create false zones); they can only be created in special name server maintenance operations.

The NS record says that a name server which is authoritative for records of the given CLASS can be found at <domain name>.

Queries

Queries to a name server must include a domain name which identifies the target resource set (QNAME), and the type and class of desired resource records. The type and class fields in a query can include any of the corresponding type and class fields that are defined for resource records; in addition, the query type (QTYPE) and query class (QCLASS) fields may contain special values that match more than one of the corresponding fields in RRs.

For example, the QTYPE field may contain:

MAILA - matches all mail agent RRs (e.g. MD and MF).

* - matches any RR type.

The QCLASS field may contain:

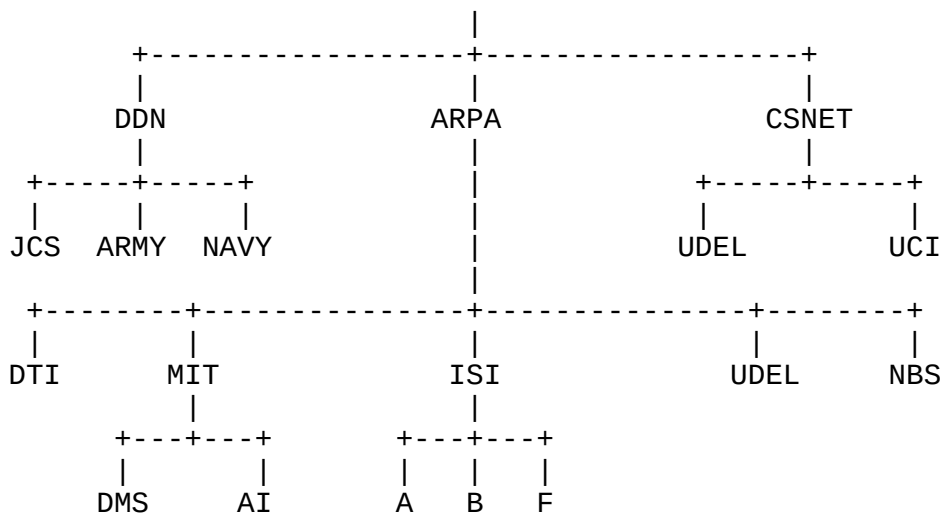
* - matches any RR class.

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

Note that the QCLASS=* construct requires special interpretation regarding authority. Since a 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.

Example space

For purposes of exposition, the following name space is used for the remainder of this memo:



NAME SERVERS

Introduction

Name servers store a distributed database consisting of the structure of the domain name space, the resource sets associated with domain names, and other information used to coordinate actions between name servers.

In general, a name server will be an authority for all or part of a particular domain. The region covered by this authority is called a zone. Name servers may be responsible for no authoritative data, and hence have no zones, or may have several zones. When a name server has multiple zones, the zones may have no common borders or zones may be contiguous.

While administrators should not construct overlapping zones, and name servers must defend against overlapping zones, overlapping is regarded as a non-fatal flaw in the database. Hence the measures taken to protect against it are omitted for the remainder of this memo. A detailed discussion can be found in [14].

When presented with a query for a domain name over which it has authority, a name server returns the desired resource information or an indication that the query refers to a domain name or resource that does not exist. If a name server is presented with a query for a domain name that is not within its authority, it may have the desired information, but it will also return a response that points toward an authoritative name server. If a name server is not an authority for a query, it can never return a negative response.

There is no requirement that a name server for a domain reside in a host which has a name in the same domain, although this will usually be the case. There is also no restriction on the number of name servers that can have authority over a particular domain; most domains will have redundant authoritative name servers. The assumption is that different authoritative copies are identical, even though inconsistencies are possible as updates are made.

Name server functions are designed to allow for very simple implementations of name servers. The simplest name server has a static set of information and uses datagrams to receive queries and return responses.

More sophisticated name server implementations can improve the performance of their clients by caching information from other domains. Although this information can be acquired in a number of ways, the normal method is to store the information acquired by a

resolver when the resolver consults other name servers. In a sophisticated host, the resolver and name server will coordinate their actions and use a shared database. This cooperation requires the incorporation of a time-to-live (TTL) field in all cached resource records. Caching is discussed in the resolver section of this memo; this section is devoted to the actions of a name servers that don't cache.

In order to free simple name servers of the requirement of managing these timeouts, simple name servers should only contain resource records that are expected to remain constant over very long periods or resource records for which the name server is an authority. In the following discussion, the TTL field is assumed to be stored in the resource record but is omitted in descriptions of databases and responses in the interest of clarity.

Authority and administrative control of domains

Although we want to have the potential of delegating the privileges of name space management at every node, we don't want such delegation to be required.

Hence we introduce the concept of authority. Authority is vested in name servers. A name server has authority over all of its domain until it delegates authority for a subdomain to some other name server.

Any administrative entity that wishes to establish its own domain must provide a name server, and have that server accepted by the parent name server (i.e. the name server that has authority over the place in the domain name space that will hold the new domain). While the principles of authority allow acceptance to be at the discretion of parent name servers, the following criteria are used by the root, and are recommended to all name servers because they are responsible for their children's actions:

1. It must register with the parent administrator of domains.

2. It must identify a responsible person.

3. In must provide redundant name servers.

The domain name must be registered with the administrator to avoid name conflicts and to make the domain related information available to other domains. The central administrator may have further requirements, and a domain is not registered until the central administrator agrees that all requirements are met.

There must be a responsible person associated with each domain to

be a contact point for questions about the domain, to verify and update the domain related information, and to resolve any problems (e.g., protocol violations) with hosts in the domain.

The domain must provide redundant (i.e., two or more) name servers to provide the name to address resolution service. These name servers must be accessible from outside the domain (as well as inside) and must resolve names for at least all the hosts in the domain.

Once the central administrator is satisfied, he will communicate the existence to the appropriate administrators of other domains so that they can incorporate NS records for the new name server into their databases.

Name server logic

The processing steps that a name server performs in responding to a query are conceptually simple, although implementations may have internal databases that are quite complex.

For purposes of explanation, we assume that the query consists of a type QTYPE, a class QCLASS, and a domain name QNAME; we assume that the name server stores its RRs in sets where each set has all of the RRs for a particular domain. Note that this database structure and the following algorithms are meant to illustrate one possible implementation, rather than a specification of how all servers must be implemented.

The following notation is used:

ord(DOMAIN-NAME)	returns the number of labels in DOMAIN-NAME.
findset(DOMAIN-NAME)	returns a pointer to the set of stored RRs for DOMAIN-NAME, or NULL if there is no such information.
set(POINTER)	refers to a set located previously by findset, where POINTER is the value returned by findset.
relevant(QTYPE,TYPE)	returns true if a RR of the specified TYPE is relevant to the specified QTYPE. For example, relevant(MAILA,MF) is true and relevant(MAILA,NS) is false.
right(NAME,NUMBER)	returns a domain name that is the rightmost NUMBER labels in the string NAME.

```
copy(RR)            copies the resource record specified by RR
                    into the response.

The name server code could be represented as the following
sequence of steps:

{    find out whether the database makes this server
     authoritative for the domain name specified by QNAME   }

for i:=0 to ord(QNAME) { sequence through all nodes in QNAME }
do   begin
     ptr:=findset(right(QNAME,i));
     if ptr<>NULL
     then { there is domain data for this domain name }
          begin
          for all RRs in set(ptr)
          do   if type(RR)=NS and class(RR)=QCLASS
               then begin
                    auth=false;
                    NSptr:=ptr
                    end;
          for all RRs in set(ptr)
          do   if type(RR)=SOA and class(RR)=QCLASS
               then auth:=true
               end
          end;
      end;

{    copy out authority search results }

if auth
then { if authority check for domain found }
     if ptr=null
     then return(Name error)
     else
else { if not authority, copy NS RRs }
     for all RRs in set(nsptr)
     do   if (type(RR)=NS and class(RR)=QCLASS)
                            or
                        (QCLASS=*)
          then copy(RR);

{    Copy all RRs that answer the question }

for all RRs in set(ptr)
do   if class(RR)=QCLASS and relevant(QTYPE,type(RR))
     then copy(RR);
```

The first section of the code (delimited by the for loop over all

of the subnodes of QNAME) discovers whether the name server is authoritative for the domain specified by QNAME. It sequences through all containing domains of QNAME, starting at the root. If it encounters a SOA it knows that the name server is authoritative unless it finds a lower NS RR which delegates authority. If the name server is authoritative, it sets auth=true; if the name server is not authoritative, it sets NSptr to point to the set which contains the NS RR closest to the domain specified by QNAME.

The second section of the code reflects the result of the authority search into the response. If the name server is authoritative, the code checks to see that the domain specified by QNAME exists; if not, a name error is returned. If the name server is not authoritative, the code copies the RRs for a closer name server into the response.

The last section of the code copies all relevant RRs into the response.

Note that this code is not meant as an actual implementation and is incomplete in several aspects. For example, it doesn't deal with providing additional information, wildcards, QCLASS=*, or with overlapping zones. The first two of these issues are dealt with in the following discussions, the remaining issues are discussed in [14].

Additional information

When a resolver returns information to a user program, the returned information will often lead to a second query. For example, if a mailer asks a resolver for the appropriate mail agent for a particular domain name, the name server queried by the resolver returns a domain name that identifies the agent. In general, we would expect that the mailer would then request the domain name to address binding for the mail agent, and a new name server query would result.

To avoid this duplication of effort, name servers return additional information with a response which satisfies the anticipated query. This information is kept in a separate section of the response. Name servers are required to complete the appropriate additional information if such information is available, but the requestor should not depend on the presence of the information since the name server may not have it. If the resolver caches the additional information, it can respond to the second query without an additional network transaction.

The appropriate information is defined in [14], but generally

consists of host to address bindings for domain names in returned RRs.

Aliases and canonical names

In existing systems, hosts and other resources often have several names that identify the same resource. For example, under current ARPA Internet naming support, USC-ISIF and ISIF 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@ISIF, Mockapetris@ISIB, etc., 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 name and all others are aliases.

The domain system provides a similar feature using the canonical name (CNAME) RR. When a name server fails to find a desired RR in a set associated with some domain name, it checks to see if the resource set contains a CNAME record with a matching class. If so, the name server includes the CNAME record in the response, and continues the query at the domain name specified in the data field of the CNAME record.

Suppose a name server was processing a query with QNAME=ISIF.ARPA, QTYPE=A, and QCLASS=IN, and had the following resource records:

```
ISIF.ARPA     CNAME   IN     F.ISI.ARPA
F.ISI.ARPA    A       IN     10.2.0.52
```

Both of these RRs would be returned in the response.

In the above example, because ISIF.ARPA has no RRs other than the CNAME RR, the resources associated with ISIF.ARPA will appear to be exactly those associated with F.ISI.ARPA for the IN CLASS. Since the CNAME is effective only when the search fails, a CNAME can also be used to construct defaults. For example, suppose the name server had the following set of RRs:

```
F.ISI.ARPA    A       IN     10.2.0.52
F.ISI.ARPA    MD      IN     F.ISI.ARPA
XXXX.ARPA     CNAME   IN     F.ISI.ARPA
XXXX.ARPA     MF      IN     A.ISI.ARPA
```

Using this database, type A queries for XXXX.ARPA would return the XXXX.ARPA CNAME RR and the F.ISI.ARPA A RR, but MAILA or MF queries to XXXX.ARPA would return the XXXX.ARPA MF RR without any information from F.ISI.ARPA. This structure might be used to send

mail addressed to XXXX.ARPA to A.ISI.ARPA and to direct TELNET for XXXX.ARPA to F.ISI.ARPA.

Wildcards

In certain cases, an administrator may wish to associate default resource information for all or part of a domain. For example, the CSNET domain administrator may wish to establish IN class mail forwarding for all hosts in the CSNET domain without IN capability. In such a case, the domain system provides a special label "*" that matches any other label. Note that "*" matches only a single label, and not zero or more than one label. Note also that the "*" is distinct from the "*" values for QCLASS and QTYPE.

The semantics of "*" depend upon whether it appears in a query domain name (QNAME) or in a RR in a database.

When an "*" is used in a QNAME, it can only match a "*" in a resource record.

When "*" appears in a RR in a database, it can never override an existing exact match. For example, if a name server received a query for the domain UDEL.CSNET, and had appropriate RRs for both UDEL.CSNET and *.CSNET, the UDEL.CSNET RRs would be used and the *.CSNET RRs would be ignored. If a query to the same database specified FOO.CSNET, the *.CSNET RR would be used, but the corresponding labels from the QNAME would replace the "*". Thus the FOO.CSNET query would match the *.CSNET RR and return a RR for FOO.CSNET rather than *.CSNET.

RRs containing "*" labels are copied exactly when zones are transfered via name server maintenance operations.

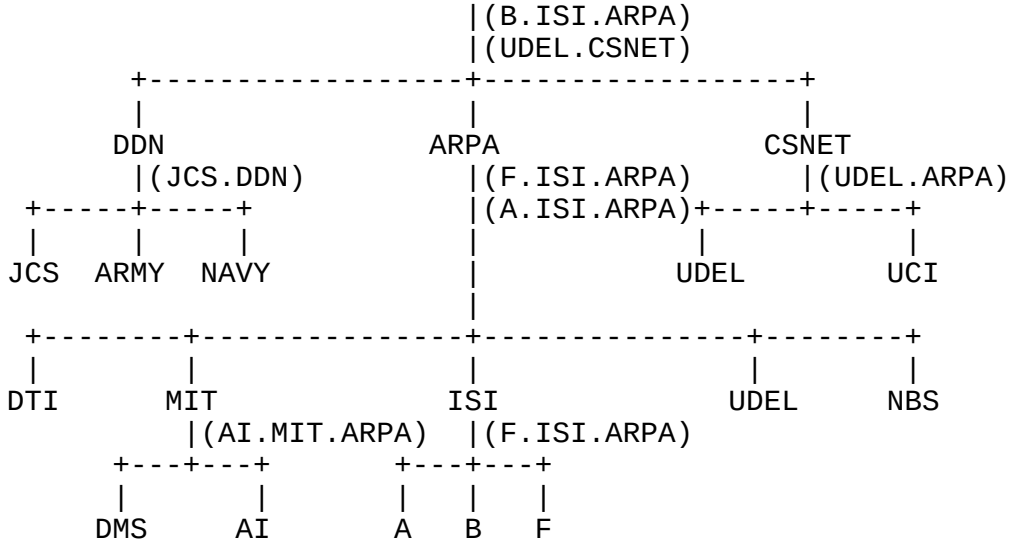
These semantics are easily implemented by having the name server first search for an exact match for a query, and then replacing the leftmost label with a "*" and trying again, repeating the process until all labels became "*" or the search succeeded.

TYPE=* in RRs is prohibited. If it were to be allowed, the requestor would have no way of interpreting the data in the RR because this data is type dependent.

CLASS=* is also prohibited. Similar effects can be achieved using QCLASS=*, and allowing both QCLASS=* and CLASS=* leads to complexities without apparent benefit.

A scenario

In our sample domain space, suppose we wanted separate administrative control for the root, DDN, ARPA, CSNET, MIT and ISI domains. We might allocate name servers as follows:



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

Thus the root name servers are on B.ISI.ARPA and UDEL.CSNET, the DDN name server is on JCS.DDN, the CSNET domain server is on UDEL.ARPA, etc.

In an actual system, all domains should have redundant name servers, but in this example only the ARPA domain has redundant servers A.ISI.ARPA and F.ISI.ARPA. (The B.ISI.ARPA and UDEL.CSNET name servers happen to be not redundant because they handle different classes.) The F.ISI.ARPA name server has authority over the ARPA domain, but delegates authority over the MIT.ARPA domain to the name server on AI.MIT.ARPA. The A.ISI.ARPA name server also has authority over the ARPA domain, but delegates both the ISI.ARPA and MIT.ARPA domains to other name servers.

B.ISI.ARPA Name server for " "

B.ISI.ARPA has the root name server for the IN class. Its database might contain:

Domain	Resource Record		
" "	SOA	IN	A.ISI.ARPA
DDN	NS	IN	JCS.DDN
ARPA	NS	IN	F.ISI.ARPA
CSNET	NS	IN	UDEL.ARPA
" "	NS	IN	B.ISI.ARPA
" "	NS	CS	UDEL.CSNET
JCS.DDN	A	IN	9.0.0.1
F.ISI.ARPA	A	IN	10.2.0.52
UDEL.CSNET	A	CS	302-555-0000
UDEL.ARPA	A	IN	10.0.0.96

The SOA record for the root is necessary so that the name server knows that it is authoritative for the root domain for class IN. The contents of the SOA resource record point back to A.ISI.ARPA and denote that the master data for the zone of authority is originally from this host. The first three NS records denote delegation of authority. The NS root entry for the B.ISI.ARPA name server is necessary so that this name server knows about itself, and can respond correctly to a query for NS information about the root (for which it is an authority). The root entry for class CS denotes that UDEL.CSNET is the authoritative name server for the CS class root. UDEL.CSNET and UDEL.ARPA may or may not refer to the same name server; from this information it is impossible to tell.

If this name server was sent a query specifying QTYPE=MAILA, QCLASS=IN, QNAME=F.ISI.ARPA, it would begin processing (using the previous algorithm) by determining that it was not an authority for F.ISI.ARPA. The test would note that it had authority at " ", but would also note that the authority was delegated at ARPA and never reestablished via another SOA. Thus the response would return the NS record for the domain ARPA.

Any queries presented to this server with QCLASS=CS would result in the UDEL.CSNET NS record being returned in the response.

F.ISI.ARPA Name server for ARPA and ISI.ARPA

In the same domain space, the F.ISI.ARPA database for the domains ARPA and ISI.ARPA might be:

Domain	Resource Record		
" "	NS	IN	B.ISI.ARPA
" "	NS	CS	CSNET.UDEL
ARPA	SOA	IN	B.ISI.ARPA
ARPA	NS	IN	A.ISI.ARPA
ARPA	NS	IN	F.ISI.ARPA
MIT.ARPA	NS	IN	AI.MIT.ARPA
ISI.ARPA	SOA	IN	F.ISI.ARPA
ISI.ARPA	NS	IN	F.ISI.ARPA
A.ISI.ARPA	MD	IN	A.ISI.ARPA
ISI.ARPA	MD	IN	F.ISI.ARPA
A.ISI.ARPA	MF	IN	F.ISI.ARPA
B.ISI.ARPA	MD	IN	B.ISI.ARPA
B.ISI.ARPA	MF	IN	F.ISI.ARPA
F.ISI.ARPA	MD	IN	F.ISI.ARPA
F.ISI.ARPA	MF	IN	A.ISI.ARPA
DTI.ARPA	MD	IN	DTI.ARPA
NBS.ARPA	MD	IN	NBS.ARPA
UDEL.ARPA	MD	IN	UDEL.ARPA
A.ISI.ARPA	A	IN	10.1.0.32
F.ISI.ARPA	A	IN	10.2.0.52
B.ISI.ARPA	A	IN	10.3.0.52
DTI.ARPA	A	IN	10.0.0.12
AI.MIT.ARPA	A	IN	10.2.0.6
DMS.MIT.ARPA	A	IN	10.1.0.6
NBS.ARPA	A	IN	10.0.0.19
UDEL.ARPA	A	IN	10.0.0.96

For the IN class, the SOA RR for ARPA denotes that this name server is authoritative for the domain ARPA, and that the master file for this authority is stored on B.ISI.ARPA. This zone extends to ISI.ARPA, where the database delegates authority back to this name server in another zone, and doesn't include the domain MIT.ARPA, which is served by a name server on AI.MIT.ARPA.

This name server is not authoritative for any data in the CS class. It has a pointer to the root server for CS data which could be use to resolve CS class queries.

Suppose this name server received a query of the form QNAME=A.ISI.ARPA, QTYPE=A, and QCLASS=IN. The authority search

would notice the NS record for " ", its SOA at ARPA, a delegation at ISI.ARPA, and the reassumption of authority at ISI.ARPA. Hence it would know that it was an authority for this query. It would then find the A record for A.ISI.ARPA, and return a datagram containing this record.

Another query might be QNAME=B.ISI.ARPA, QTYPE=MAILA, QCLASS=*. In this case the name server would know that it cannot be authoritative because of the "*" value of QCLASS, and would look for records for domain B.ISI.ARPA that match. Assuming that the name server performs the additional record inclusion mentioned in the name server algorithm, the returned datagram would include:

```
ISI.ARPA      NS      IN     F.ISI.ARPA
" "           NS      CS     UDEL.CSNET
B.ISI.ARPA    MD      IN     B.ISI.ARPA
B.ISI.ARPA    MF      IN     F.ISI.ARPA
B.ISI.ARPA    A       IN     10.3.0.52
F.ISI.ARPA    A       IN     10.2.0.52
```

If the query were QNAME=DMS.MIT.ARPA, QTYPE=MAILA, QCLASS=IN, the name server would discover that AI.MIT.ARPA was the authoritative name server and return the following:

```
MIT.ARPA      NS      IN     AI.MIT.ARPA
AI.MIT.ARPA   A       IN     10.2.0.6
```

In this case, the requestor is directed to seek information from the MIT.ARPA domain name server residing on AI.MIT.ARPA.

UDEL.ARPA and UDEL.CSNET name server

In the previous discussion of the sample domain, we stated that UDEL.CSNET and UDEL.ARPA might be the same name server. In this example, we assume that this is the case. As such, the name server is an authority for the root for class CS, and an authority for the CSNET domain for class IN.

This name server deals with mail forwarding between the ARPA Internet and CSNET systems. Its RRs illustrate one approach to solving this problem. The name server has the following resource records:

```
" "           SOA     CS     UDEL.CSNET
" "           NS      CS     UDEL.CSNET
" "           NS      IN     B.ISI.ARPA
CSNET         SOA     IN     UDEL.ARPA
CSNET         NS      IN     UDEL.ARPA
ARPA          NS      IN     A.ISI.ARPA

*.CSNET       MF      IN     UDEL.ARPA
UDEL.CSNET    MD      CS     UDEL.CSNET
UCI.CSNET     MD      CS     UCI.CSNET
UDEL.ARPA     MD      IN     UDEL.ARPA

B.ISI.ARPA    A       IN     10.3.0.52
UDEL.ARPA     A       IN     10.0.0.96
UDEL.CSNET    A       CS     302-555-0000
UCI.CSNET     A       CS     714-555-0000
```

Suppose this name server received a query of the form QNAME=UCI.CSNET, QTYPE=MAILA, and QCLASS=IN. The name server would discover it was authoritative for the CSNET domain under class IN, but would find no explicit mail data for UCI.CSNET. However, using the *.CSNET record, it would construct a reply:

```
UCI.CSNET     MF      IN     UDEL.ARPA
UDEL.ARPA     A       IN     10.0.0.96
```

If this name server received a query of the form QNAME=UCI.CSNET, QTYPE=MAILA, and QCLASS=CS, the name server would return:

```
UCI.CSNET     MD      CS     UCI.CSNET
UCI.CSNET     A       CS     714-555-0000
```

Note that although this scheme allows for forwarding of all mail addressed as <anything>.CSNET, it doesn't help with names that have more than two components, e.g. A.B.CSNET. Although this problem could be "fixed" by a series of MF entries for *.*.CSNET,

..*.CSNET, etc, a more tasteful solution would be to introduce a cleverer pattern matching algorithm in the CSNET name server.

Summary of requirements for name servers

The requirements for a name server are as follows:

1. It must be recognized by its parent.

2. It must have complete resource information for all domain names for which it is the authority.

3. It must periodically refresh authoritative information from a master file or name server which holds the master.

4. If it caches information it must also handle TTL management for that information.

5. It must answer simple queries.

Inverse queries

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 query might map a domain name to a host address, the corresponding inverse query might map the address 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 an 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. In general, a resolver or other program that wishes to guarantee that an inverse query will work must use a name server that is known to have the appropriate data, or ask all name servers in a domain of interest.

For example, if a resolver wishes to perform an inverse query for an arbitrary host on the ARPA Internet, it must consult a set of name servers sufficient to know that all IN data was considered. In practice, a single inverse query to a name server that has a fairly comprehensive database should satisfy the vast majority of inverse queries.

A detailed discussion of inverse queries is contained in [14].

Completion services

Some existing systems provide the ability to complete partial specifications of arguments. The general principle is that the user types the first few characters of the argument and then hits an escape character to prompt the system to complete the rest. Some completion systems require that the user type enough of the argument to be unique; others do not.

Other systems allow the user to specify one argument and ask the system to fill in other arguments. For example, many mail systems allow the user to specify a username without a host for local mail delivery.

The domain system defines name server completion transactions that perform the analogous service for the domain system. Implementation of this service is optional in a name server, but all name servers must at least be able to understand a completion request and return an error response.

When a resolver wishes to request a completion, it sends a name server a message that sets QNAME to the partial string, QTYPE to the type of resource desired, and QCLASS to the desired class. The completion request also includes a RR for the target domain. The target domain RR identifies the preferred location of the resource. In completion requests, QNAME must still have a null label to terminate the name, but its presence is ignored. Note that a completion request is not a query, but shares some of the same field formats.

For example, a completion request might contain QTYPE=A, QNAME=B, QCLASS=IN and a RR for ISI.ARPA. This request asks for completion for a resource whose name begins with "B" and is "close" to ISI.ARPA. This might be a typical shorthand used in the ISI community which uses "B" as a way of referring to B.ISI.ARPA.

The first step in processing a completion request is to look for a "whole label" match. When the name server receives the request mentioned above, it looks at all records that are of type A, class IN, and whose domain name starts (on the left) with the labels of QNAME, in this case, "B". If multiple records match, the name server selects those whose domain names match (from the right) the most labels of the preferred domain name. If there are still multiple candidates, the name server selects the records that have the shortest (in terms of octets in the name) domain name. If several records remain, then the name server returns them all.

If no records are found in the previous algorithm, the name server assumes that the rightmost label in QNAME is not complete, and

looks for records that match but require addition of characters to the rightmost label of QNAME. For example, the previous search would not match BB.ARPA to B, but this search would. If multiple hits are found, the same discarding strategy is followed.

A detailed discussion of completion can be found in [14].

RESOLVERS

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.

Because a resolver may need to consult several name servers, the amount of time that a resolver will take to complete can vary. This variance is part of the justification for the split between name servers and resolvers; name servers may use datagrams and have a response time that is essentially equal to network delay plus a short service time, while resolvers may take an essentially indeterminate amount of time.

We expect to see two types of resolvers: simple resolvers that can chain through multiple name servers when required, and more complicated resolvers that cache resource records for use in future queries.

Simple resolvers

A simple resolver needs the following capabilities:

1. It must know how to access a name server, and should know the authoritative name server for the host that it services.

2. It must know the protocol capabilities for its clients so that it can set the class fields of the queries it sends to return information that is useful to its clients. If the resolver serves a client that has multiple protocol capabilities, it should be able to support the preferences of the client.

   The resolver for a multiple protocol client can either collect information for all classes using the * class value, or iterate on the classes supported by the client. Note that in either case, the resolver must understand the preferences of the host. For example, the host that supports both CSNET and ARPA

Internet protocols might prefer mail delivery (MD) to mail forwarding (MF), regardless of protocol, or might prefer one protocol regardless of whether MD or MF is required. Care is required to prevent loops.

3. The resolver must be capable of chaining through multiple name servers to get to an authoritative name server for any query. The resolver should guard against loops in referrals; a simple policy is to discard referrals that don't match more of the query name than the referring name server, and also to avoid querying the same name server twice (This test should be done using addresses of name servers instead of domain names to avoid problems when a name server has multiple domain names or errors are present in aliases).

4. The resolver must be able to try alternate name servers when a name server doesn't respond.

5. The resolver must be able to communicate different failure conditions to its client. These failure conditions include unknown domain name, unknown resource for a know domain name, and inability to access any of the authoritative name servers for a domain.

6. If the resolver uses datagrams for queries, it must recover from lost and duplicate datagrams.

Resolvers with cache management

Caching provides a tool for improving the performance of name service, but also is a potential source of incorrect results. For example, a database might cache information that is later changed in the authoritative name servers. While this problem can't be eliminated without eliminating caching, it can be reduced to an infrequent problem through the use of timeouts.

When name servers return resource records, each record has an associated time-to-live (TTL) field. This field is expressed in seconds, and has 16 bits of significance.

When a resolver caches a returned resource record it must also remember the TTL field. The resolver must discard the record when the equivalent amount of time has passed. If the resolver shares a database with a name server, it must decrement the TTL field of imported records periodically rather than simply deleting the record. This strategy is necessary to avoid exporting a resource record whose TTL field doesn't reflect the amount of time that the resource record has been cached. Of course, the resolver should

not decrement the TTL fields of records for which the associated name server is an authority.

Appendix 1 - Domain Name Syntax Specification

The preferred syntax of domain names is given by the following BNF rules. Adherence to this syntax will result in fewer problems with many applications that use domain names (e.g., mail, TELNET). Note that some applications described in [14] use domain names containing binary information and hence do not follow this syntax.

```
<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 ARPA Internet:

```
F.ISI.ARPA     LINKABIT-DCN5.ARPA     UCL-TAC.ARPA
```

REFERENCES and BIBLIOGRAPHY

[1] E. Feinler, K. Harrenstien, Z. Su, and V. White, "DOD Internet Host Table Specification", RFC 810, Network Information Center, SRI International, March 1982.

[2] J. Postel, "Computer Mail Meeting Notes", RFC 805, USC/Information Sciences Institute, February 1982.

[3] Z. Su, and J. Postel, "The Domain Naming Convention for Internet User Applications", RFC 819, Network Information Center, SRI International, August 1982.

[4] Z. Su, "A Distributed System for Internet Name Service", RFC 830, Network Information Center, SRI International, October 1982.

[5] K. Harrenstien, and V. White, "NICNAME/WHOIS", RFC 812, Network Information Center, SRI International, March 1982.

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

[7] K. Harrenstien, "NAME/FINGER", RFC 742, Network Information Center, SRI International, December 1977.

[8] J. Postel, "Internet Name Server", IEN 116, USC/Information Sciences Institute, August 1979.

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

[10] J. Postel, "Transmission Control Protocol", RFC 793, USC/Information Sciences Institute, September 1981.

[11] J. Postel, "User Datagram Protocol", RFC 768, USC/Information Sciences Institute, August 1980.

[12] J. Postel, "Simple Mail Transfer Protocol", RFC 821, USC/Information Sciences Institute, August 1980.

[13] J. Reynolds, and J. Postel, "Assigned Numbers", RFC 870, USC/Information Sciences Institute, October 1983.

[14] P. Mockapetris, "Domain Names - Implementation and Specification", RFC 883, USC/Information Sciences Institute, November 1983.