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.

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.