Development of the Domain Name System*

Paul V. Mockapetris
USC Information Sciences Institute, Marina del Rey, California

Kevin J. Dunlap
Digital Equipment Corp., DECwest Engineering, Washington

(Originally published in the Proceedings of SIGCOMM '88, Computer Communication Review Vol. 18, No. 4, August 1988, pp. 123–133.)

Abstract

The Domain Name System (DNS) provides name service for the DARPA Internet. It is one of the largest name services in operation today, serves a highly diverse community of hosts, users, and networks, and uses a unique combination of hierarchies, caching, and datagram access.

This paper examines the ideas behind the initial design of the DNS in 1983, discusses the evolution of these ideas into the current implementations and usages, notes conspicuous surprises, successes and shortcomings, and attempts to predict its future evolution.

1. Introduction

The genesis of the DNS was the observation, circa 1982, that the HOSTS.TXT system for publishing the mapping between host names and addresses was encountering or headed for problems. HOSTS.TXT is the name of a simple text file, which is centrally maintained on a host at the SRI Network Information Center (SRI-NIC) and distributed to all hosts in the Internet via direct and indirect file transfers.

The problems were that the file, and hence the costs of its distribution, were becoming too large, and that the centralized control of updating did not fit the trend toward more distributed management of the Internet.

Simple growth was one cause of these problems; another was the evolution of the community using HOSTS.TXT from the NCP-based original ARPANET to the IP/TCP-based Internet. The research ARPANET's role had changed from being a single network connecting large timesharing systems to being one of the several long-haul backbone networks linking local networks which were in turn populated with workstations. The number of hosts changed from the number of timesharing systems (roughly organizations) to the number of workstations (roughly users). This increase was directly reflected in the size of HOSTS.TXT, the rate of change in HOSTS.TXT, and the number of transfers of the file, leading to a much larger than linear increase in total resource use for distributing the file. Since organizations were being forced into management of local network addresses, gateways, etc., by the technology anyway, it was quite logical to want to partition the database and allow local control of local name and address spaces. A distributed naming system seemed in order.

Existing distributed naming systems included the DARPA Internet's IEN116 [IEN 116] and the XEROX Grapevine [Birrell 82] and Clearinghouse systems [Oppen 83]. The IEN116 services seemed excessively limited and host specific, and IEN116 did not provide much benefit to justify the costs of renovation. The XEROX system was then, and may still be, the most sophisticated name service in existence, but it was not

*This research was supported by the Defense Advanced Research Projects Agency under contract MDA903-87-C-0719. Views and conclusions contained in this report are the authors' and should not be interpreted as representing the official opinion or policy of DARPA, the U.S. government, or any person or agency connected with them.

clear that its heavy use of replication, light use of caching, and fixed number of hierarchy levels were appropriate for the heterogeneous and often chaotic style of the DARPA Internet. Importing the XEROX design would also have meant importing supporting elements of its protocol architecture. For these reasons, a new design was begun.

The initial design of the DNS was specified in [RFC 882, RFC 883]. The outward appearance is a hierarchical name space with typed data at the nodes. Control of the database is also delegated in a hierarchical fashion. The intent was that the data types be extensible, with the addition of new data types continuing indefinitely as new applications were added. Although the system has been modified and refined in several areas [RFC 973, RFC 974], the current specifications [RFC 1034, RFC 1035] and usage are quite similar to the original definitions.

Drawing an exact line between experimental use and production status is difficult, but 1985 saw some hosts use the DNS as their sole means of accessing naming information. While the DNS has not replaced the HOSTS.TXT mechanism in many older hosts, it is the standard mechanism for hosts, particularly those based on Berkeley UNIX, that track progress in network and operating system design.

2. DNS Design

The base design assumptions for the DNS were that it must:

- provide at least all of the same information as HOSTS.TXT.
- Allow the database to be maintained in a distributed manner.
- Have no obvious size limits for names, name components, data associated with a name, etc.
- Interoperate across the DARPA Internet and in as many other environments as possible.
- Provide tolerable performance.

Derivative constraints included the following:

- The cost of implementing the system could only be justified if it provided extensible services. In particular, the system should be independent of network topology, and capable of encapsulating other name spaces.
- In order to be universally acceptable, the system should avoid trying to force a single OS, architecture, or organizational style onto its users. This idea applied all the way from concerns about case sensitivity to the idea that the system should be useful for both large timeshared hosts and isolated PCs. In general, we wanted to avoid any constraints on the system due to outside influences and permit as many different implementation structures as possible.

The HOSTS.TXT emulation requirement was not particularly severe, but it did cause an early examination of schemes for storing data other than name-to-address mappings. A hierarchical name space seemed the obvious and minimal solution for the distribution and size requirements. The interoperability and performance constraints implied that the system would have to allow database information to be buffered between the client and the source of the data, since access to the source might not be possible or timely.

The initial DNS design assumed the necessity of striking a balance between a very lean service and a completely general distributed database. A lean service was desirable because it would result in more implementation efforts and early availability. A general design would amortize the cost of introduction across more applications, provide greater functionality, and increase the number of environments in which the DNS would eventually be used. The “leanness” criterion led to a conscious decision to omit many of the functions one might expect in a state-of-the-art database. In particular, dynamic update of the database with the related atomicity, voting, and backup considerations was omitted. The intent was to add these eventually, but it was believed that a system that included these features would be viewed as too complex to be accepted by the community.

2.1 The architecture

The active components of the DNS are of two major types: name servers and resolvers. Name servers are repositories of information, and answer queries using whatever information they possess. Resolvers interface to client programs, and embody the algorithms necessary to find a name server that has the information sought by the client.

These functions may be combined or separated to suit the needs of the environment. In many cases, it is useful to centralize the resolver function in one or more special name servers for an organization. This structure shares the use of cached information, and also allows less capable hosts, such as PCs, to rely on the resolving services of special servers without needing a resolver in the PC.

2.2 The name space

The DNS internal name space is a variable-depth tree where each node in the tree has an associated label. The domain name of a node is the concatenation of all labels on the path from the node to the root of the tree. Labels are variable-length strings of octets, and each octet in a label can be any 8-bit value. The zero length label is reserved for the root. Name space searching operations (for operations defined at present) are done in a case-insensitive manner (assuming ASCII). Thus the labels “Paul”, “paul”, and “PAUL”, would match each other. This matching rule effectively prohibits the creation of brother nodes with labels having equivalent spelling but different case. The rational for this system is that it allows the sources of information to specify its canonical case, but frees users from having to deal with case. Labels are limited to 63 octets and names are restricted to 256 octets total as an aid to implementation, but this limit could be easily changed if the need arose.

The DNS specification avoids defining a standard printing rule for the internal name format in order to encourage DNS use to encode existing structured names. Configuration files in the domain system represent names as character strings separated by dots, but applications are free to do otherwise. For example, host names use the internal DNS rules, so VENERA.ISI.EDU is a name with four labels (the null name of the root is usually omitted). Mailbox names, stated as USER@DOMAIN (or more generally as local-part@organization) encode the text to the left of the “@” in a single label (perhaps including “.”) and use the dot-delimiting DNS configuration file rule for the part following the @. Similar encodings could be developed for file names, etc.

The DNS also decouples the structure of the tree from any implicit semantics. This is not done to keep names free of all implicit semantics, but to leave the choices for these implicit semantics wide open for the application. Thus the name of a host might have more or fewer labels than the name of a user, and the tree is not organized by network or other grouping. Particular sections of the name space have very strong implicit semantics associated with a name, particularly when the DNS encapsulates an existing name space or is used to provide inverse mappings (e.g. IN-ADDR.ARPA, the IP addresses to host name section of the domain space), but the default assumption is that the only way to tell definitely what a name represents is to look at the data associated with the name.

The recommended name space structure for hosts, users, and other typical applications is one that mirrors the structure of the organization controlling the local domain. This is convenient since the DNS features for distributing control of the database is most efficient when it parallels the tree structure. An administrative decision [RFC 920] was made to make the top levels correspond to country codes or broad organization types (for example EDU for educational, MIL for military, UK for Great Britain).

2.3 Data attached to names

Since the DNS should not constrain the data that applications can attach to a name, it can’t fix the data’s format completely. Yet the DNS did need to specify some primitives for data structuring so that replies to queries could be limited to relevant information, and so the DNS could use its own services to keep track of servers, server addresses, etc. Data for each name in the DNS is organized as a set of resource records (RRs); each RR carries a well-known type and class field, followed by applications data. Multiple values of the same type are represented as separate RRs.

Types are meant to represent abstract resources or functions, for example, host addresses and mailboxes. About 15 are currently defined. The class field is meant to divide the database orthogonally from type, and specifies the protocol family or instance. The DARPA Internet has a class, and we imagined that classes might be allocated to CHAOS, ISO, XNS or similar protocol families. We also hoped to try setting up function-specific classes that would be independent of protocol (e.g. a universal mail registry). Three classes are allocated at present: DARPA Internet, CHAOS, and Hessiod.

The decision to use multiple RRs of a single type rather than including multiple values in a single RR differed from that used in the XEROX system, and was not a clear choice. The space efficiency of the single RR with multiple values was attractive, but the multiple RR option cut down the maximum RR size. This appeared to promise simpler dynamic update protocols, and also seemed suited to use in a limited-size datagram environment (i.e. a response could carry only those items that fit in a maximum size packet without regard to partial RR transport).

2.4 Database distribution

The DNS provides two major mechanisms for transferring data from its ultimate source to ultimate destination: zones and caching. Zones are sections of the system-wide database which are controlled by a specific organization. The organization controlling a zone is responsible for distributing current copies of the zones to multiple servers which make the zones available to clients throughout the Internet. Zone transfers are typically initiated by changes to the data in the zone. Caching is a mechanism whereby data acquired in response to a client's request can be locally stored against future requests by the same or other client.

Note that the intent is that both of these mechanisms be invisible to the user who should see a single database without obvious boundaries.

Zones

A zone is a complete description of a contiguous section of the total tree name space, together with some "pointer" information to other contiguous zones. Since zone divisions can be made between any two connected nodes in the total name space, a zone could be a single node or the whole tree, but is typically a simple subtree.

From an organization's point of view, it gets control of a zone of the name space by persuading a parent organization to delegate a subzone consisting of a single node. The parent organization does this by inserting RRs in its zone which mark a zone division. The new zone can then be grown to arbitrary size and further delegated without involving the parent, although the parent always retains control of the initial delegation. For example, the ISI.EDU zone was created by persuading the owner of the EDU domain to mark a zone boundary between EDU and ISI.EDU.

The responsibilities of the organization include the maintenance of the zone's data and providing redundant servers for the zone. The typical zone is maintained in a text form called a master file by some system administrator and loaded into one master server. The redundant servers are either manually reloaded, or use an automatic zone refresh algorithm which is part of the DNS protocol. The refresh algorithm queries a serial number in the master's zone data, then copies the zone only if the serial number has increased. Zone transfers require TCP for reliability.

A particular name server can support any number of zones which may or may not be contiguous. The name server for a zone need not be part of that zone. This scheme allows almost arbitrary distribution, but is most efficient when the database is distributed in parallel with the name hierarchy. When a server answers from zone data, as opposed to cached data, it marks the answer as being authoritative.

A goal behind this scheme is that an organization should be able to have a domain, even if it lacks the communication or host resources for supporting the domain's name service. One method is that organizations with resources for a single server can form buddy systems with another organization of similar means. This can be especially desirable to clients when the organizations are far apart (in network terms), since it makes the data available from separated sites. Another way is that servers agree to provide name service for large communities such as CSNET and UUCP, and receive master files via mail or FTP from their subscribers.

Caching

In addition to the planned distribution of data via zone transfers, the DNS resolvers and combined name server/resolver programs also cache responses for use by later queries. The mechanism for controlling caching is a time-to-live (TTL) field attached to each RR. This field, in units of seconds, represents the length of time that the response can be reused. A zero TTL suppresses caching. The administrator defines TTL values for each RR as part of the zone definition; a low TTL is desirable in that it minimizes periods of transient inconsistency, while a high TTL minimizes traffic and allows caching to mask periods of server unavailability due to network or host problems. Software components are required to behave as if they continuously decremented TTLs of data in caches. The recommended TTL value for host names is two days.

Our intent is that cached answers be as good as answers from an authoritative server, excepting changes made within the TTL period. However, all components of the DNS prefer authoritative information to cached information when both are available locally.

3. Current Implementation Status

The DNS is in use throughout the DARPA Internet. [RFC 1031] catalogs a dozen implementations or ports, ranging from the ubiquitous support provided as part of Berkeley UNIX, through implementations for IBM-PCs, Macintoshes, LISP machines, and fuzzballs

[Mills 88]. Although the HOSTS.TXT mechanism is still used by older hosts, the DNS is the recommended mechanism. Hosts available through HOSTS.TXT form an ever-dwindling subset of all hosts; a recent measurement [Stahl 87] showed approximately 5,500 host names in the present HOSTS.TXT, while over 20,000 host names were available via the DNS.

The current domain name space is partitioned into roughly 30 top level domains. Although a top level domain is reserved for each country (approximately 25 in use, e.g. US, UK), the majority of hosts and subdomains are named under six top level domains named for organization types (e.g. educational is EDU, commercial is COM). Some hosts claim multiple names in different domains, though usually one name is primary and others are aliases. The SRI-NIC manages the zones for all of the non-country, top-level domains, and delegates lower domains to individual universities, companies, and other organizations who wish to manage their own name space.

The delegation of subdomains by the SRI-NIC has grown steadily. In February of 1987, roughly 300 domains were delegated. As of March 1988, over 650 domains are delegated. Approximately 400 represent normal name spaces controlled by organizations other than the SRI-NIC, while 250 of these delegated domains represent network address spaces (i.e. parts of IN-ADDR.ARPA) no longer controlled by the NIC.

Two good examples of contemporary DNS use are the so called “root servers” which are the redundant name servers that support the top levels of the domain name space, and the Berkeley subdomain, which is one of the domains delegated by the SRI-NIC in the EDU domain.

3.1 Root servers

The basic search algorithm for the DNS allows a resolver to search “downward” from domains that it can access already. Resolvers are typically configured with “hints” pointing at servers for the root node and the top of the local domain. Thus if a resolver can access any root server it can access all of the domain space, and if the resolver is in a network partitioned from the rest of the Internet, it can at least access local names.

Although a resolver accesses root servers less as the resolver builds up cached information about servers for lower domains, the availability of root servers is an important robustness issue, and root server activity monitoring provides insights into DNS usage.

Since access to the root and other top level zones is so important, the root domain, together with other top-level domains managed by the SRI-NIC, is supported by seven redundant name servers. These root servers are scattered across the major long haul backbone networks of the Internet, and are also redundant in that three are TOPS-20 systems running JEEVES and four are UNIX systems running BIND.

The typical traffic at each root server is on the order of a query per second, with correspondingly higher rates when other root servers are down or otherwise unavailable. While the broad trend in query rate has generally been upward, day-to-day and month-to-month comparisons of load are driven more by changes in implementation algorithms and timeout tuning than growth in client population. For example, one bad release of popular domain software drove averages to over five times the normal load for extended periods. At present, we estimate that 50% of all root server traffic could be eliminated by improvements in various resolver implementations to use less aggressive retransmission and better caching.

The number of clients which access root servers can be estimated based on measurement tools on the TOPS-20 version. These root servers keep track of the first 200 clients after root server initialization, and the first 200 clients typically account for 90% or more of all queries at any single server. Coordinated measurements at the three TOPS-20 root servers typically show approximately 350 distinct clients in the 600 entries. The number of clients is falling as more organizations adopt strategies that concentrate queries and caching for accesses outside of the local organization.

The clients appear to use static priorities for selecting which root server to use, and failure of a particular root server results in an immediate increase in traffic at other servers. The vast majority of queries are four types: all information (25 to 40%), host name to address mappings (30–40%), address to host mappings (10 to 15%), and new style mail information called MX (less than 10%). Again, these numbers vary widely as new software distributions spread. The root servers refer 10–15% of all queries to servers for lower level domains.

3.2 Berkeley

UNIX support for the DNS was provided by the University of California, Berkeley, partially as research in distributed systems, and partially out of necessity due to growth in the campus network [Dunlap 86a, Dunlap 86b]. The result is the Berkeley Internet Name

Domain (BIND) server. Berkeley serves as an example of a large delegated domain, though it is certainly more sophisticated and has more experience than most.

With BIND, Berkeley became the first organization on the DARPA Internet to bring up machines with all their network applications solely dependent on DNS for doing network host and address resolution. Berkeley started to install machines on campus dependent on the name server in the spring of 1985. In the fall of 1985, the two mail gateways to the DARPA Internet were converted to depend on the DNS, this meant the entire campus had to adopt domain-style mail addresses.

Educating even the sophisticated Berkeley user community on the new form of addressing turned out to be a major task. The single biggest objection from the user community was due to mail addresses which became obsolete, closely followed by the initial lack of shorthands and search rules in the initial implementation.

While the DNS transition was painful, the need was clear, as shown in the following table which gives the number of hosts, subnets, and finally subdomains in use at Berkeley over the last three years. For example, from January 1986 to February 1987, Berkeley added 735 hosts in 250 working days, an average of three new hosts each working day.

Date	Hosts	Subnets	Subdomains
January 1986	267	14	
February 1987	1002	44	
March 1988	1991	86	5

Note that Berkeley has recently divided its domain into multiple zones for administrative convenience.

4. Surprises

Operation of the DNS has revealed several issues that came as surprises to the developers, but on reflection seem quite unsurprising.

4.1 Refinement of semantics

The main role of the DNS is to act as a repository for information, and the initial assumption was that the form and content of that information was well-understood. This turned out to be a bad assumption. Even existing common concepts such as IP host addresses were sources of problems; we knew that we would have to support multiple addresses for a single host, but we were drawn into long discussions of whether the addresses attached to a host name should be ordered, and if so, by what metric.

4.2 Performance

The performance of the underlying network was much worse than the original design expected. Growth in the number of networks overtaxed gateway mechanisms for keeping track of connectivity, leading to lost paths and unidirectional paths. At the same time, growth in load plus the addition of many lower speed links led to longer delays. These problems were manifest at the root servers, where logs reveal many instances of repeated copies of the same query from the same source. Even though the TOPS-20 root servers take less than 100 milliseconds to process the vast majority of queries, clients typically see response times of 500 milliseconds to 5 seconds, even for the closest root server, depending on their location in the Internet. The situation for queries to the delegated domains is often much worse, both because of network troubles, and because the name servers for these domains are often on heavily loaded hosts on less-central networks. Queries from the ARPANET to delegated domains typically take 3 to 10 seconds during prime time, with 30 to 60 second times as occasional worst cases. It is interesting to note that these times to access a remote name server are similar to those seen for the XEROX homogeneous name service [Larson 85].

A related surprise was the difficulty in making reasonable measurements of DNS performance. We had planned to measure the performance of DNS components in order to estimate costs for future enhancement and growth, and to guide tuning of existing retransmission intervals, but the measurements were often swamped by unrelated effects due to gateway changes, new DNS software releases, and the like. Many of the servers perform better as their load increases due to fewer page faults, but this is clearly not a stable situation over the long term, leading to concerns about behavior should network performance improve and be able to deliver higher loads to the servers.

The performance of lookups for queries that did not need network access was a pleasant surprise. We were replacing a fairly simple host table lookup with a more complicated database, so even if cache access worked very well, we might slow existing applications down a great deal. However, the new mechanisms are typically as good or better than the old, regardless of implementation. The reason for this is that the old mechanisms were created for a much smaller database and were not adjusted as the size of database grew explosively, while the new software was based on the assumption of a very large database.

4.3 Negative caching

The DNS provides two negative responses to queries. One says that the name in question does not exist, while the other says that while the name in question exists, the requested data does not. The first might be expected if a name were misspelled, while the second might result if a query asked for the host type of a mailbox or the mailing list members of a host. These responses were expected to be rare.

Initial monitoring of root server activity showed a very high percentage (20 to 60%) of these responses. Logs revealed that many of these queries were generated by programs using old-style host names, or names from other mail internets (e.g. UUCP). In the latter case, mailers would often use a call to the name to address conversion routines to test whether an address was valid in the DARPA Internet, even though this might be easily determined by other means. Since few UUCP mail addresses are valid domain names, this resulted in a negative response from a root server, coupled with a delay for the non-local query.

We expected that the negative responses would decrease, and perhaps vanish, as hosts converted their names to domain-name format and as we asked mail software maintainers to modify their programs. Even though these steps were taken, negative responses stayed in the 10–50% range, with a typical percentage of 25%.

The reason is that the corrective measures were offset by the spread of programs which provided shorthand names through a search list mechanism. The search lists produce a steady stream of bad names as they try alternatives; a mistyped name may now lead to several name errors rather than one. Our conclusion is that any naming system that relies on caching for performance may need caching for negative results as well. Such a mechanism has been added to the DNS as an optional feature, with impressive performance gains in cases where it is supported in both the involved name servers and resolvers. This feature will probably become standard in the future.

5. Successes

5.1 Variable depth hierarchy

The variable-depth hierarchy is used a great deal and was the right choice for several reasons:

- The spread of workstation and local network technology meant that organizations participating in the Internet were finding a need to organize within themselves.
- The organizations were of vastly different size, and hence needed different numbers of organizational levels. For example, both large international companies and small startups are registered in the domain system.
- The variable depth hierarchy makes it possible to encapsulate any fixed level or variable level system. For example, the UK's own name service (NRS) and the DNS mutually encapsulate each other's name space. This scheme may also be used in the future to interoperate with the directory service under development by the ISO and CCITT.

Many networks that do not use the DNS protocols and datatypes have standardized on the DNS hierarchical name syntax for mail addressing [Quarterman 86].

5.2 Organizational structuring of names

While the particular top-level organizational structure used by the current DNS is quite controversial, the principle that names are independent of network, topology, etc. is quite popular. The future structure of the top levels is likely to continue to be a subject of debate. Most proposals generate a roughly equivalent amount of support and condemnation. In the authors' opinion, the only real possibility for wholesale change is a political decision to change the structure of the domain name space to resemble the name space proposed for the ISO/CCITT directory service. This is not a technical issue as the DNS is flexible enough to accommodate almost any political choice.

5.3 Datagram access

The use of datagrams as the preferred method for accessing name servers was successful and probably was essential, given the unexpectedly bad performance of the DARPA Internet. The restriction to approximately 512 bytes of data turns out not to be a problem, performance is much better than that achieved by TCP circuits, and OS resources are not tied up.

The only obvious drawback to datagram access is the need to develop and refine retransmission strategies that are already quite well developed for TCP. Much unnecessary traffic is generated by resolvers that were developed to the point of working, but whose authors lost interest before tuning, or by systems that imported well known versions of code but do not track tuning updates.

5.4 Additional section processing

When a name server answers a query, in addition to whatever information it uses to answer the question, it is free to include in the response any other information it sees fit, as long as the data fits in a single datagram. The idea was to allow the responding server to anticipate the next logical request and answer it before it was asked without significant added communication cost. For example, whenever the root servers pass back the name of a host, they include its address (if available), on the assumption that the host address is needed to use other information. Experiments show that this feature cuts query traffic in half.

5.5 Caching

The caching discipline of the DNS works well, and given the unexpectedly bad performance of the Internet, was essential to the success of the system.

The only problems with caching relate to databases and query strategies that make it less reliable or useful. For example, RRs of the same type at a particular node should have the same TTL so that they will time out simultaneously, but administrators sometimes assign TTLs in the mistaken idea that they are assigning some sort of priority. Administrators also are very fond of picking short TTLs so that their changes take effect rapidly, even if changes are very rare and do not need the timeliness.

A related concern is the security and reliability problems caused by indiscriminate caching. Several existing resolvers cache all information in responses without regard to its reasonableness. This has resulted in numerous instances where bad information has circulated and caused problems. Similar difficulties were encountered when one administrator reversed the TTL and data values, resulting in the distribution of bad data with a TTL of several years. While various measures have reduced the vulnerability to error, the security of the present system does depend on the integrity of the network addressing mechanism, and this is questionable in an era of local networks and PCs.

5.6 Mail address cooperation

Agreement between representatives of the CSNET, BITNET, UUCP, and DARPA Internet communities led to an agreement to use organizationally structured domain names for mail addressing and routing. While the transition from the messy multiply-encoded mail addresses of the past is far from complete, the possibility of cleaning up mail addresses has been clearly demonstrated.

6. Shortcomings

6.1 Type and class growth

When the draft DNS specifications were made available in 1983, the one nearly unanimous criticism was that the type and class data specifiers, which were 8 bits in the draft, should be expanded to 16, or even 32 bits, to allow for new definitions. Over the first five years of DNS use, two new types have been adopted, two types have been dropped, and two new classes have been allocated. Clearly, either the demand for new types and classes was completely misunderstood, or the current DNS makes new definitions too difficult.

While one problem is that almost all existing software regards types and classes as compile-time constants, and hence requires recompilation to deal with changes, a less tractable problem is that new data types and classes are useless until their semantics are carefully designed and published, applications created to use them, and a consensus is reached to use the new system across the Internet. This means that new types face a series of technical and political hurdles.

A methodology or guidelines to aid in the design of new types of information is needed. This is more complicated than just listing the values of interest for an application, since it often involves the design of special name space sections, TTL selections to produce acceptable performance and semantics, and decisions whether to produce a desired binding through one lookup or a sequence of smaller bindings. The single lookup method often seems overwhelmingly attractive to a particular application designer despite the fact that it may overlap or conflict with another application's data. Another factor is that members of the Internet have different views on the proper assumptions or approach for a particular problem.

Mail is an example. After much debate, the MX data type and system [RFC 974] defined a standard method for routing mail, based on the DOMAIN part or a LOCAL-PART@DOMAIN mail address. MX represented a simple addition to the DNS itself, but required changes to all mail servers, and its benefits required a “critical mass” of mailers. Numerous suggestions have been made to extend the DNS to provide mail destination registry down to the individual user level, and the basics of such a service are within our understanding, but consensus for a single plan remains elusive. Part of the constituency demands that user level mail binding be an option on top of MX, while others advocate a fresh start, with lots of features for mail forwarding, list maintenance, etc. The best choice seems to be one in which agent binding is always a choice, but that a mailer which chooses to map to the mailbox level can do so if the mailbox data is also available.

6.2 Easy upgrading of applications

Converting network applications to use the DNS is not a simple task. It would be ideal if all the applications converting from HOSTS.TXT could be recompiled to use the DNS and have everything work, but this is rarely the case.

Part of the problem is transient failure. A distributed naming system, by its very nature, has periods that it can not access particular information. Applications must handle this condition appropriately. Mailers looking up mail destinations should not discard mail due to these transient failures, and can not afford to wait indefinitely. Even if such failures are anticipated to be quite rare once the DNS stabilizes, we face a chicken-and-egg problem in converting mailers to use the new software.

Another part of the problem is that access to the naming system needs to be integrated into the operating system to a much greater degree than providing system call to the resolver. Users need to be able to access these services at the shell level and specify search lists and defaults in a manner consistent with other system operations.

6.3 Distribution of control vs. distribution of expertise or responsibility

Distributing authority for a database does not distribute a corresponding amount of expertise. Maintainers fix things until they work, rather than until they work well, and want to use, not understand, the systems they are provided. Systems designers should anticipate this, and try to compensate by technical means. The DNS furnishes several examples of this principle:

- The initial policy was that we would delegate a domain to any organization which filled out a form listing its redundant servers and other essentials. Instead we should have required that the organization demonstrate redundant servers with real data in them before we delegated the domain, and probably should have insisted that they be on different networks, rather than trusting assurances that the servers did not represent a single point of failure.
- The documentation for the system used examples which were easily explained in the narration. Sample TTL values which mapped to an hour were always copied; text that said the values should be a few days was ignored. Documentation should always be written with the assumption that only the examples are read.
- Debugging of the system was hampered by questions about software versions and parameters. These values should be accessible via the protocol.

7. Conclusions

Just as the classification of many of the previous issues into “successes”, “surprises”, and “shortcomings” is open to debate based on the perspective of the reader, so too is the question “Was the DNS a good idea?”

Modifications to the HOSTS.TXT scheme could have postponed the need for a new system, and reduced the quantitative arguments for the DNS. The DNS has probably not yet reduced the community-wide administrative, communication, or support load. However, the need to distribute functionality was, we believe, inexorable. This need, together with the new functionality and opportunities for future services must be the key criteria for judgment. From the authors’ perspective, they justify the DNS.

There are a lot of choices we might make differently if we were starting over, but the main pieces of advice which would have been valuable when we were starting are:

- Caching can work in a heterogeneous environment, but should include features for caching negative responses as well.
- It is often more difficult to remove functions from systems than it is to get a new function added. All of a community would not convert to

a new service; instead some will stay with the old, some will convert to the new, and some will support both. This has the unfortunate effect of making all functions more complex as new features are added.

- The most capable implementors lose interest once a new system delivers the level of performance they expect; they are not easily motivated to optimize their use of others' resources or provide easily used guidelines for the administrators that use the systems. Distributed software should include a version number and table of parameters which can be interrogated. If possible, systems should include technical means for transferring tuning parameters, or at least defaults, to all installations without requiring the attention of system maintainers.
- Allowing variations in the implementation structure used to provide service is a great idea; allowing variation in the provided service causes problems.

8. Directions for future work

Although the DNS is in production use and hence difficult to change, other research in naming systems, particularly the emerging ISO X.500 directory services, may provide the impetus for additions:

- Support for X.500 style addresses for mail, etc. could be constructed as a layer on top of the DNS, albeit without the sophisticated protection, update, and structuring rules of X.500. Use of the data description techniques from the ISO standards might provide a better mechanism for adding data types than the present data structuring rules, while the proven DNS infrastructure could speed prototyping of ISO applications.
- The value of a ubiquitous name service and consistent name space at all levels of the protocol suite and operating system seems obvious, but it is equally obvious that tradeoffs between performance, generality, and distribution require at least different styles of use at different levels. For example, a system suitable for managing file names on a local disk would be substantially different from a system for maintaining an internet wide mailing list. The challenge here is to develop an approach which, at least conceptually, structures the total task into layers or some other coherent organization.
- Research in naming systems has typically resulted in proposals for systems which could replace or encapsulate all other systems, or systems which allow translations between separate name spaces, data formats, etc. Both approaches have advantages and drawbacks. The present DNS and efforts to unify its name space without special domains for specific networks, etc. place the DNS in the first category. However, its success is universal enough to be encouraging while not enough to solve the user's difficulty with obscure encodings from other systems. Technical and/or political solutions to the growing complexity of naming will be a growing need.

References

[Birrell 82] Birrell, A. D., Levin, R., Needham, R. M., and Schroeder, M. D., "Grapevine: An Exercise in Distributed Computing", Communications of ACM 25, 4:260–274, April 1982.

[Dunlap 86a] Dunlap, K. J., Bloom, J. M., "Experiences Implementing BIND, A Distributed Name Server for the DARPA Internet", Proceedings USENIX Summer Conference, Atlanta, Georgia, June 1986, pages 172–181.

[Dunlap 86b] Dunlap, K. J., "Name Server Operations Guide for BIND", Unix System Manager's Manual, SMM-11. 4.3 Berkeley Software Distribution, Virtual VAX-11 Version. University of California, April 1986.

[IEN 116] Postel, Jon, "Internet Name Server", IEN 116, August 1979.

[Larson 85] Larson, Personal communication.

[Mills 88] Mills, D. L., "The Fuzzball", Proceedings ACM SIGCOMM 88 Symposium, August, 1988.

[Oppen 83] D. C. Oppen and Y. K. Dalal, "The Clearinghouse: A decentralized agent for locating named objects in a distributed environment", ACM

Transactions on Office Information Systems 1(3):230–253, July 1983. An expanded version of this paper is available as Xerox Report OPD-T8103, October 1981.

[Quarterman 86] Quarterman, John S., and Hoskins, Josiah C., "Notable Computer Networks", Communications of the ACM, October 1986, volume 29, number 10.

[RFC 882] P. Mockapetris, "Domain names—Concepts and Facilities," RFC 882, USC/Information Sciences Institute, November 1983. (Obsolete, superseded by RFC 1034.)

[RFC 883] P. Mockapetris, "Domain names—Implementation and Specification," RFC 883, USC/Information Sciences Institute, November 1983. (Obsolete, superseded by RFC 1035.)

[RFC 920] Postel, Jon, and Reynolds, Joyce, "Domain Requirements", RFC 920, October 1984.

[RFC 973] Mockapetris, Paul V., "Domain System Changes and Observations", RFC 973, January 1986.

[RFC 974] Partridge, Craig, "Mail Routing and the Domain System", RFC 974, January 1986.

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

[RFC 1034] P. Mockapetris, "Domain names - Concepts and Facilities," RFC 1034, USC/Information Sciences Institute, November 1987.

[RFC 1035] P. Mockapetris, "Domain names - Implementation and Specification," RFC 1035, USC/Information Sciences Institute, November 1987.

[Stahl 87] M. Stahl, "DDN Domain Naming—Administration, Registration, Procedures and Policy", Second TCP/IP Interoperability Conference, December, 1987.

Note: In the above references, "RFC" refers to papers in the Request for Comments series and "IEN" refers to the DARPA Internet Experiment Notes. Both the RFCs and IENs may be obtained from the Network Information Center, SRI International, Menlo Park, CA 94025, or from the authors of the papers.