The Berkeley Internet Name Domain Server

Douglas B. Terry, Mark Painter, David W. Riggle, and Songnian Zhou

Computer Systems Research Group
Computer Science Division
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley

ABSTRACT

The Berkeley Internet Name Domain (BIND) Server allows a standard way of naming the many types of objects and resources that exist in distributed UNIX environments, and provides operations for storing and retrieving information about these objects. BIND Servers collectively manage a hierarchical name space that is partitioned into domains reflecting administrative entities. Many existing UNIX applications, particularly mail facilities, will benefit greatly from such a service.

1. INTRODUCTION

Computers running the UNIX operating system are no longer small, stand-alone time sharing systems, but exist as part of larger distributed computing communities. For instance, environments in which a moderate number of computers running Berkeley UNIX are connected by a local internet are becoming quite common, especially within universities. Desires to share information with others outside of our local environments, typically through electronic mail, are also quite prevalent. The latest version of Berkeley UNIX, 4.2 BSD, has incorporated facilities to support distributed applications, such as network communication protocols, into the kernel.[2] Current efforts are underway in the Computer Systems Research Group at U. C. Berkeley to evolve Berkeley UNIX into a more transparent distributed operating system.[1]

In order to accomplish transparent sharing of objects and resources in a distributed environment, a uniform means of naming and locating the different types of resources must be established. The Berkeley Internet Name Domain (BIND) Server is being implemented to provide such a service to Berkeley UNIX users. BIND Servers running on various machines collectively manage a global database of information about named objects such as host addresses, user mailboxes, and

This work was partially sponsored by the Defense Advanced Research Projects Agency (DoD), ARPA Order No. 4031, and monitored by the Naval Electronics Systems Command under Contract No. N00039-C-0235.

server ports. The goal is to achieve a more transparent and less troublesome distributed computing environment by permitting objects to be referenced independent of their physical locations.

2. DESIGN OVERVIEW

The BIND Servers maintain a tree-structured name space, complying with the DARPA Internet Naming Convention,[6,9] in which each node of the tree has an associated label. The name of a *domain*, or subpart of the tree, is simply the concatenation of all the labels of the domains from the root to the top node of the domain, listed from right to left and separated by dots. The labels need only be unique within the same domain. No limitation exists on the number of levels of the domain space and the number of subdomains that a domain may have.

The authority for managing parts of the name space could potentially be delegated at every domain. Specifically, the whole space is partitioned into a number of areas called *zones* that start at a domain and extend down to the leaf nodes or to domains where other zones start. Zones usually represent human administrative boundaries and associated authorities. For example, Berkeley may have a zone "ucb.arpa" and the computing center at Berkeley may have a zone "cc.ucb.arpa" under "ucb.arpa", each being maintained independently.

Servers store information about objects and resources in *resource records* consisting of a domain name, class, type and data fields.[3,4] Each domain name may have a number of resource records associated with it. The value of the type field, along with the class, specify the format of the data field. The set of allowed types is well defined; for instance, resource records for host addresses have a type value "A" and those for user mailboxes have a type value "MB".

BIND Servers consist of facilities for database management and mechanisms for managing the name space in cooperation with other name servers. The database management subsystem provides utilities for storing and retrieving resource records for a single server. The name management subsystem is responsible for adding semantics to the database, such as recognizing aliases or answering queries, as well as maintaining consistency among replicated data. The interface to the BIND Servers is through *resolvers*, a group of subroutines that users call to access the name servers. In processing user queries, for instance, resolvers are responsible for locating a server with the desired authoritative information. The relationships between the resolvers, the databases, and the name management facilities are depicted in Figure 1. These three aspects of the naming service are discussed in more detail in the next three sections. Section 6 then discusses some possible uses for BIND Servers in a distributed UNIX environment.

3. DISTRIBUTED NAME MANAGEMENT AND QUERY PROCESSING

Each BIND Server is responsible for managing parts of the name space, called zones. A general mapping between zones and servers exists, that is, a zone may be stored at one or more servers, and a server may contain zero or more zones. A

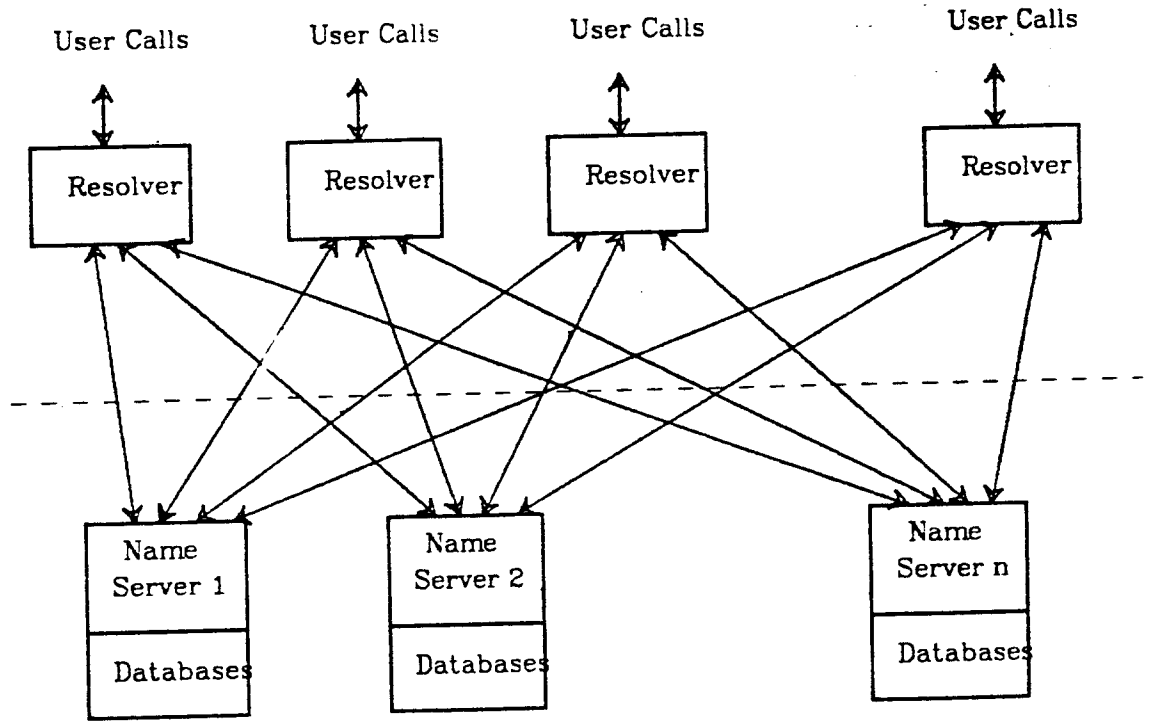

Figure 1. Relationship between resolvers, name servers, and databases.

name server containing a zone is said to be an *authoritative name server* for that zone. To enhance performance and availability, each zone is generally stored in several servers; the degree of replication depends on the frequency of its use and its importance for network operation. The zone at the root of the domain tree, for instance, may be highly replicated to avoid frequent remote queries and to enable operation to continue when servers fail.

Authoritative servers are classified into a *primary name server* and *secondary name servers* for each zone. The primary name server stores the truly authoritative copy of the zone database, whereas the other servers get their zone information via a zone transfer operation from the primary server at system startup time. Note that the terms primary and secondary are meaningful only with respect to a particular zone. While a server is secondary for one zone, it may well be primary for another.

3.1. Retrieval Queries

The first step in processing a client's request for information about a particular domain name, deciding which of possibly many zones to start searching, is accomplished by comparing the domain name in question with the top domain of each zone maintained by the queried server.[4,10] Assuming zones do not overlap, the zone with the closest match is the only one on this server that may contain the

domain being sought. For example, suppose the requested domain is "d.c.b.a", and the name server contains zones with origins "a","e.c.b.a", and "b.a"; then the zone "b.a" is selected. In this example, zone "a" delegated part of its authority to zone "b.a", and zone "e.c.b.a" borders the domain being sought.

Once the closest zone has been selected, the server must search down the path from the zone origin to the domain being sought to check if authority has been delegated to another zone at some domain along the path. If the current name server is determined to be authoritative for the queried domain, then the requested resource record(s) are looked up in the zone database and returned to the resolver if found. On the other hand, if the selected zone ends before the desired domain is reached, indicating that authority had been delegated to a different zone, the domain names and addresses of the servers to which authority was delegated are returned. If no reasonable zone to search resides at the server, information about the authoritative servers for the root domain will be returned to the resolver to allow it to continue its search for the desired domain.

3.2. Update Operations

Three types of update operations are supported by the BIND servers: addition, deletion, and modification of resource records.[10,5] For modification operations, which are meant to be atomic, both the old and the new records are provided to facilitate complete error checking. The old and new records may differ only in their data fields. If different domain names and/or classes were allowed, a modification could involve two zones managed by different servers, requiring more elaborate transaction mechanisms to ensure atomicity.

All updates to a zone must be directed to the primary name server for that zone. While this restriction reduces the availability of updates, it drastically simplifies the algorithms for concurrency control and replicated data consistency. Secondary authoritative servers get their initial data as well as all the updates from the primary server. Data propagation has a radiation pattern, flowing out of the primary server to all the secondary servers. Absolute data consistency among all the copies of the zone data is not guaranteed; instead, only the eventual convergence of the data is ensured.

Incremental refresh operations, instead of whole zone transfers, are used to propagate updates after the initial zone transfer.[10] When an update arrives at a primary name server, in addition to performing the update to its database, the server also records the update in a data structure called the *update list*. Each secondary server periodically establishes a communication connection with the primary server and sends a refresh query identifying itself and the zone desired. The primary server then responds with all of the updates since the last refresh for this secondary server. Pointers into the update list are maintained for each secondary server to keep track of how much of the update list this server has received; the storage occupied by update records that have been distributed to all of the secondary servers is reclaimed.

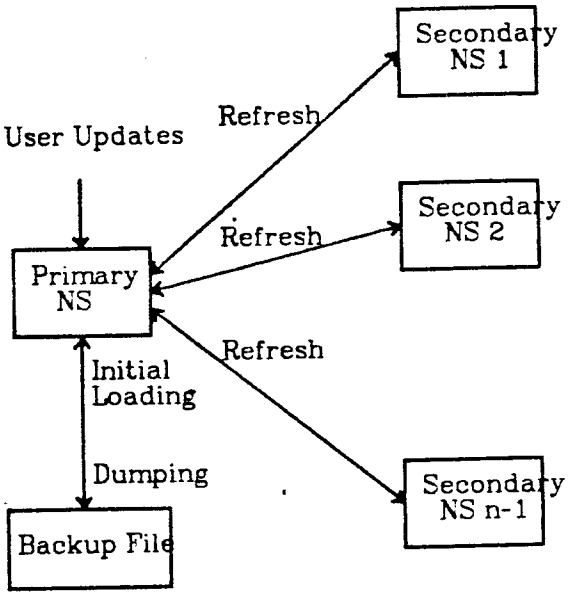

Figure 2. Name server structure in relation to a particular zone.

Figure 2 shows the name servers' structure in relation to a particular zone, and the data propagation pattern. The incremental refresh scheme has two major performance advantages. One is that whole zone transfers are avoided, thereby reducing network and host load significantly. The other is that the updates are propagated in batches, and their frequencies are controlled by each secondary server so that frequent connection costs are avoided. These advantages are particularly noticeable for servers connected over slow dialup lines.

3.3. Performance and Protection Issues

If name server operations are processed serially then a client's response time could be seriously impaired since a zone refresh may take a tremendous amount of time, especially an initial zone transfer. This involves requesting a connection, sending out the query, waiting for the response, and performing all the updates in the response. In the mean time, all the incoming queries are queued up. For performance reasons, a name server should be able to answer user queries and perform maintenance operations, such as zone refreshes, concurrently. Concurrent operations would require that multiple processes share zone databases and name server data structures. For example, the user query process could perform user updates and store them in the update list to be used later by the maintenance query process for propagation. Unfortunately, such sharing of information is very difficult to implement in Berkeley UNIX since each process has its own private address space and no memory sharing between processes is possible. To avoid expensive communication between processes, either through shared files or by

messages, a BIND server runs as a single process; the various activities are multiplexed within that process.

Lastly, permitting interactive user updates necessitates authentication and access control mechanisms in order for the name servers to be usable in most environments. Although no access controls have been implemented in the current version of the BIND servers, access lists could easily be stored in the BIND server database to prevent unauthorized operations. Facilities for remote authentication should probably be provided by separate authentication servers.

4. DATABASE FACILITIES

The use of a large general purpose database system to support a name server would be unnecessarily expensive since only simple retrieval and update operations are required. Mechanisms for joins, selects, and even elaborate crash recovery would be superfluous. Hence, special purpose data manipulation facilities were built with simplicity and speed as the major goals. [8]

An important design decision was to keep the entire database resident in a process' virtual address space. The amount of data stored by a BIND Server is small enough to make a memory resident database feasible on a VAX. The complete database is stored in a collection of fixed-sized buffers making re-use of storage very easy, I/O and memory operations uniform, and debugging easier. Data that is longer than one full buffer, currently 32 bytes, can be stored in several buffers which are then linked together. Two pointers provided in each buffer allow hierarchical structures to be built. Since buffers are dynamically allocated in sequential blocks of a thousand, traversing lists built from fresh buffers should not result in many page faults. A garbage collector periodically reclaims unused buffers and groups them on a free list.

A hash table maps full domain names (treated as a flat name space) into a hierarchical class and type structure built from the single sized data buffers (see Figure 3). Hash table collisions are resolved by chaining. Associated with each domain name is a zone name and a list of classes, where each class has a list of types and each type has a list of data with its time-to-live (TTL).

The database management subsystem answers a standard query for a particular domain name as follows: the domain name is hashed and the collision chain examined until a match for both domain name and zone is found; the desired class is found by a linear search through the class list. From that class the search continues down its type list for the desired type. After the type is located, the data fields bei[illegible]re returned to the name server process.

To survive crashes, a parallel version of the database is maintained reliably on disk. Every time a data buffer is modified, its disk buffer is similarly modified. Updates are always performed by switching a single pointer from the old data buffers to the new data buffers to render the operation atomic. Unfortunately, the UNIX operating system makes it difficult to force data onto the disk. All update routines call *fflush* before returning, but no more assurance of the data actually

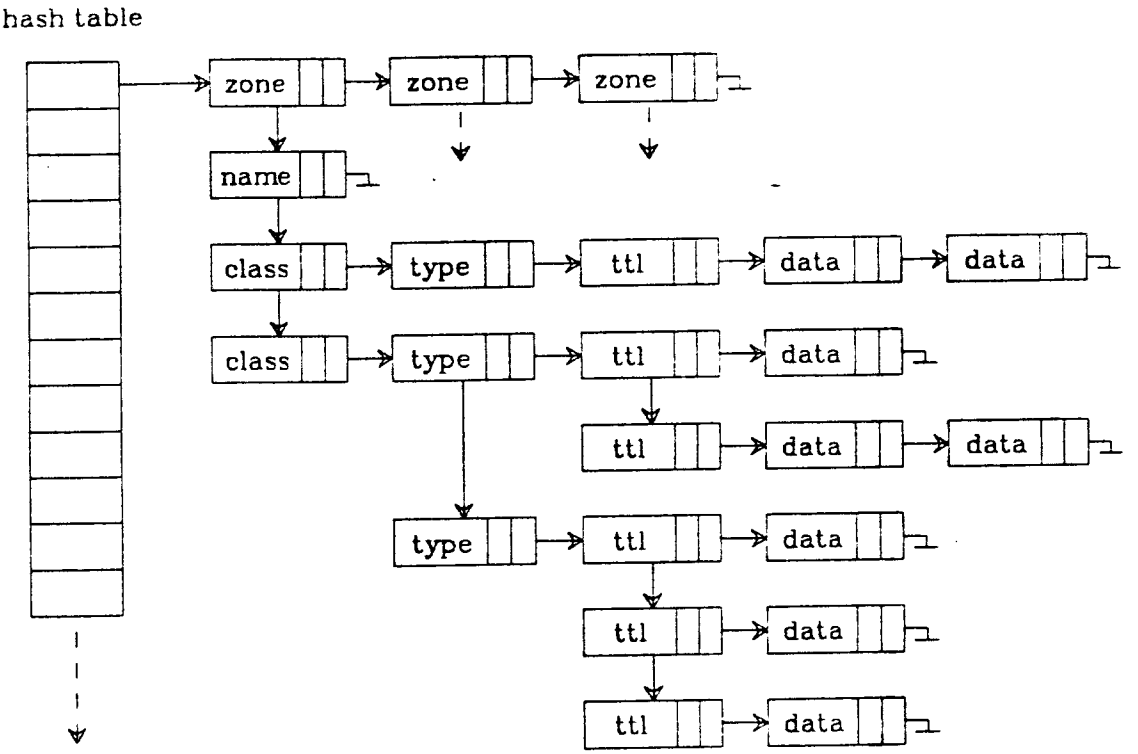

Figure 3. A sample domain name database entry.

being written to disk is provided. A consistency checker is run on the database at system startup time to guarantee that the database is in a consistent state.

5. RESOLVERS

A resolver insulates users from much of the complexity of communicating with BIND servers. Specifically, a resolver handles the details of transmitting a user's query to a name server, including the retransmission of lost queries or responses and the recovery from duplicate, or otherwise stale, responses. Resolvers use an iterative process to locate an authoritative server for a given domain. If a queried BIND server is unable to provide the desired information about the particular domain, the response contains a referral to another name server, or an error indication. Eventually, either the query will be satisfied, or the resolver will be unable to proceed. Initially, the resolver obtains the address of at least one name server from a configuration file.

The current resolver used with the BIND servers may be accessed from C language programs as a collection of subroutines.[5] These user level routines, whose semantics reflect the update and retrieval operations supported within BIND servers, are listed in the Table 1. The present implementation required no modifications to the Berkeley UNIX 4.2 BSD kernel and has the advantage that the cost of communicating with the resolver is simply that of a subroutine call. However, a process using a resolver cannot benefit from data that has been retrieved and cached by another process. Alternatively, the resolver's functions

could be placed in the kernel so that a shared cache can be maintained.

Routine Name	Purpose
std_query	User routine for forming general standard queries.
dn_in_addr	Convert a domain name to the Berkeley UNIX 4.2 BSD structure which is used to manipulate file descriptors for IP communications objects.
inv_query	User routine for forming general inverse queries.
in_to_dname	Convert Berkeley UNIX 4.2 BSD structure which is used to manipulate file descriptors for IP communications objects to a set of corresponding domain names.
com_query	User routine for forming general completion queries.
dnc_in_addr	Identical to "dn_in_addr" except that the domain name need not be completely specified.
answer_i	Return an individual resource record from the set of resource records which are returned by "std_query", "com_query", "inv_query" or "in_to_dname".
add_rr	Ensure that a resource record is in the database of the primary name server for its domain name.
delete_rr	Ensure that a resource record is not in the database of the primary name server for its domain name.
modify_rr	Ensure that the new version of a resource record is in the database of the primary name server for its domain name, and the old version is not. (fails if neither present)
set_resopt	Set and clear a number of options for the resolver.
set_domain	Set the domain name of the name server to which inverse and completion queries will be directed.

Table 1. Current resolver interface

6. UNIX APPLICATIONS

One can envision many possible uses of name servers in a Berkeley UNIX environment. The most obvious use is in support of mail facilities. The procedure of editing */usr/lib/aliases* and waiting several minutes for *newaliases* to massage the file is exceedingly annoying. If stored in a BIND Server, this information about aliases and distribution lists would be incrementally updatable and more readily available. An alias could simply be a domain name with a resource record of type "ALIAS" whose data field contained the recipient's standard name. Distribution lists could either be stored in a single resource record or as a collection of resource records, one for each member of the group. Other information, such as a description of the list's purpose (e.g. the "Doctor Who mailing list at Berkeley") or

the person in charge of maintaining it, could be stored with the distribution list in resource records of various types.

UNIX users often have an entry in the file */usr/lib/aliases* on most, if not all, machines in their local environment that directs mail to the person's home machine. Ideally, mail should be addressed independent of mailbox locations with the BIND Server mapping users to mailbox sites. For example, mail could be sent to "terry@Berkeley.ARPA" instead of "terry@ucbarpa.Berkeley.ARPA" and get properly forwarded to the "ucbarpa" machine by consulting a BIND Server that is authoritative for the "Berkeley.ARPA" domain. Not only would this relieve senders from having to remember and specify machines, but it would also give recipients convenient control over where their mail is received since the name server could simply be updated when a user migrates to a new machine.

Gains in managing a distributed computing environment can also be made by storing information such as the */etc/hosts* file, the *finger* database, and possibly the password file in BIND Servers. Typically this information is replicated on all machines that are under a common administrative authority; often updates must be manually performed on each machine. With BIND Servers, the routines *gethostent*, *getnetent*, *getprotoent*, *getservent*, *etc.* would be simple name server queries. This is only a sample of the range of possible uses of BIND Servers in a local distributed computing environment based on Berkeley UNIX.

7. SUMMARY

The Berkeley Internet Name Domain Server has been designed and implemented to serve the needs of distributed computing communities. It allows a standard way of naming the many types of objects and resources that exist in such an environment, and provides operations for storing and retrieving information about these objects. A number of important decisions have been made in the design of the BIND Servers. These include the incorporation of interactive user update queries, the distinction between primary and secondary name servers, the management scheme for replicated zones, and the protocol for incremental zone refresh. Emphasis was placed on being able to incorporate hosts of various types into the distributed community, including those connected over slow speed telephone lines or under different administrative authorities. Special care was taken to make the BIND Server's external interface upward compatible with the naming service planned for the DARPA Internet.[7,3,4]

The current version of the BIND Server, as described in this paper, is written in the C programming language and runs on 4.2 BSD UNIX. Applications that make use of the service must now be implemented to test out its design. Several existing UNIX utilities could easily be converted to using the resolver routines, and many new applications should be developed to exploit the uniform distributed name space.

References

1. D. Ferrari, "The Evolution of Berkeley UNIX," *Proceedings COMPCON Spring '84*, pp. 502-505, February-March 1984.
2. W. Joy, E. Cooper, R. Fabry, S. Leffler, K. McKusick, D. Mosher, "4.2BSD System Manual," Technical Report 5, Computer Systems Research Group, University of California, Berkeley, Draft of September 1, 1982.
3. P. Mockapetris, "Domain Names -- Concepts and Facilities," RFC 882, USC/Information Sciences Institute, November 1983.
4. P. Mockapetris, "Domain Names -- Implementation and Specification," RFC 883, USC/Information Sciences Institute, November 1983.
5. M. Painter, "The Design and Implementation of a "Domain Names" Resolver," Report No. UCB/CSD 84/176 (Masters Report), Computer Science Division, University of California, Berkeley, May 1984.
6. J. Postel, "Computer Mail Meeting Notes," RFC 805, USC/Information Sciences Institute, February 1982.
7. J. Postel, "The Domain Names Plan and Schedule," RFC 881, USC/Information Sciences Institute, November 1983.
8. D. W. Riggle, "A Name Server Database," Report No. UCB/CSD 84/174 (Masters Report), Computer Science Division, University of California, Berkeley, May 1984.
9. Z. Su, J. Postel, "The Domain Naming Convention for Internet User Applications," RFC 819, Network Information Center, SRI International, August 1982.
10. S. Zhou, "The Design and Implementation of the Berkeley Internet Name Domain (BIND) Servers," Report No. UCB/CSD 84/177 (Masters Report), Computer Science Division, University of California, Berkeley, May 1984.