[Schneider90] F.B. Schneider, “Implementing Fault-tolerant Services using the State Machine Approach: A Tutorial,” *ACM Computing Surveys* ***22***, December 1990.

[Alsberg76] P.A. Alsberg, and J.D. Day, “A Principle for Resilient Sharing of Distributed Resources,” *Proceedings of the Second International Conference on Software Engineering,* 1990, San Francisco, CA, 562-570.

[Albitz92] P. Albitz and C. Liu, “DNS and BIND,” *O’Reilly & Associate, Inc.,* October 1994.

[OSF92] Open Software Foundation, “Introduction to OSF DCE,” *Prentice Hall, Inc.,* 1994.

[Birrell82] A.D. Birrell, R. Levin, R.M.Needham and M.D. Schroeder, “Grapevine: An Exercise in Distributed Computing,” *Communications of the ACM 25,* pp 260-274.

[Birrell84] A.D. Birrell, and B.J. Nelson, “Implementing Remote Procedure Calls,” *Communications of the ACM Transaction on Compute Systems,* Vol. 2, No.1, pp 39-59, February 1984.

[Lampson86] B.W. Lampson, “Designing a Global Name Service,” *Proc. 5th ACM Annual Symposium on Principles of Distributed Computing,* Calgary, Canada.

[ISIS92] “The ISIS Distributed Toolkit Version 3.0 User Reference Manual,” *ISIS Distributed Systems, Inc*. 1992.

[Sykas91] E.D. Sykas, and G.L. Lyberopoulos, “Overview of the CCITT X.500 Recommendations series,” *Computer Communications Review*, November 1991.

[Stonebraker91] M. Stonebraker and G. Kemnitz, “The POSTGRES Next-Generation Database Management System,” *Comm. of the ACM*, Vol. 34, No. 10, October, 1991, pp.78-92.

[Rowe92] L.A. Rowe, and B.C. Smith, “A Continuous Media Play,” *Proc. 3rd Int. Workshop on Network and OS Support for Digital Audio and Video,* San Diego CA, November 1992. Also available as ftp://mm-ftp.cs.-berkeley.edu/pub/multimedia/papers/CM-Player.ps.Z.

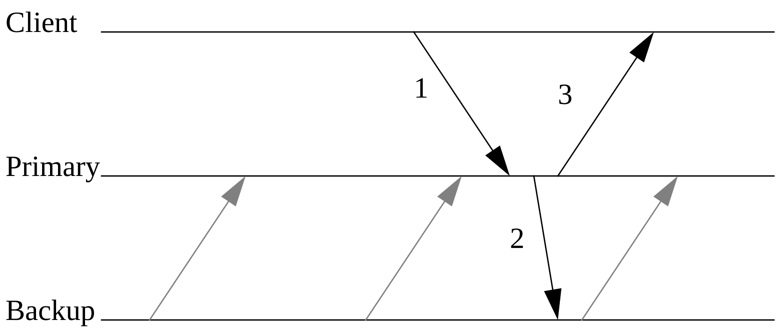


FIGURE 9. Primary-Backup Protocol

lease 3.3.

The name server package is composed of approximately 3500 lines of Tcl code:

(1) 1000 lines in the name server
(2) 700 lines in the client library
(3) 1800 lines in the name server monitor and installation interface.

The most difficult part of the implementation was debugging the backup servers because each backup server runs as a daemon and there is no Tcl debugger that allows you to attach to a process and debug it. Testing the auto-start feature also turns out to be difficult because servers can fail to start and tools to monitor remote process do not exist. Despite the implementation difficulties, the experience with the name server so far has been positive. Its advantage is particular obvious in a demonstration environment in which things are unstable.

Currently, we are considering whether to extend the name server to handle service load-balancing. As mentioned before, a client can query the name server for a list of servers that support particular service. However, it is up to the client to pick the server to which it wants to connect. It would be very useful for the client to have access to data such as the load of each service so it can make more intelligent decisions. One way to implement this feature is to have each server periodically report its load average to the name server in a format that clients can understand. Or, the name server can periodically monitor the load average of the machine on which the server is running. It remains to be investigated which approach is better. Another extension being considered is to use the backup servers as replicated servers giving the name server as a whole higher throughput. The idea is to direct all read-only requests to the backup servers and all updates to the primary server. The issues involved are how to distribute the requests evenly among the backup servers and what happens when a backup server crashes.

6. References

[Smith93] B.C. Smith, L.A. Rowe, and S. Yen, “Tcl Distributed Programming,” *Proc. 1993 Tcl/Tk Workshop,* Berkeley, CA, June 1993.

[Ousterhout94] J.K. Ousterhout, “Tcl and the Tk Toolkit,” *Addison-Wesley Professional Computing Series,* April 1994.

[Rowe94] L.A. Rowe, “Continuous Media Applications,” *Computer Science Division - EECS, University of California at Berkeley,* November 1992. Also available as ftp://mm-ftp.cs.berkeley.edu/pub/multimedia/papers/CMApps94.ps.Z.

[Federighi94] C. Federighi and L.A. Rowe, “The Design and Implementation of the UCB Distributed Video On Demand System,” *Proc. of IS&T/SPIE 1994 Int’l Symp. on Elec. Imaging: Science and Technology,* San Jose, CA, February 1994. Also available as ftp://mm-ftp.cs.berkeley.edu/pub/multimedia/papers/VodsArch-SPIE94.ps.Z

[Rowe95] L.A. Rowe, et. al., “The Berkeley Distributed VOD System,” *Proc. 6th NEC Research Symposium on Multimedia Computing,* Tokyo, Japan, June 1995.

[Ritchie74] D.M. Ritchie, and K. Thompson, “The UNIX Time-Sharing System,” *Communications of the ACM 17:7,* July 1974.

[Stiner88] J.G. Steiner, C. Neuman, and J. I. Schiller, “Kerberos: An Authentication Service for Open Network Systems,” *USENIX Winter Conference,* February 9-12 1988, Dallas, Texas.

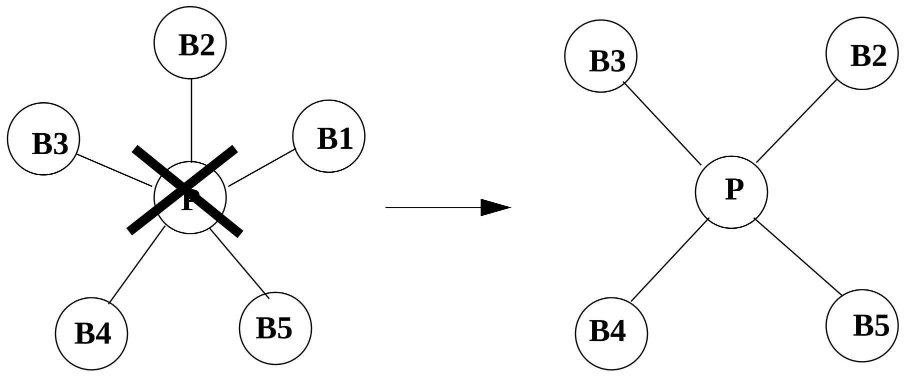


FIGURE 8. Backup Scheme

down the entire system. In backup mode, a system administrator can specify a list of machines on which the name server should run. One machine is picked as the primary server, and the rest are backup servers. Also, each server has an implicit server identification number (SID) that is inferred from the machine list. Each backup server maintains a private connection with the primary, forming a star configuration as shown in Figure 8. When the primary goes down, each backup server will detect it by noticing the bad connection. The backup server with the SID one greater that the primary's SID will then designate itself as the new primary and the others will re-form the star configuration around the new primary. In Figure 8, backup server one (B1) takes over as the primary server. Once the new primary takes over, it will try to restart the dead server. The bad-connection detection is implemented by setting the ***keepAlive*** flag using `dp_socketOption` which requests the system to send periodic messages on a tcp socket. Should the connected party fail to respond to these messages, the connection is considered broken and will be closed automatically. The `dp_atclose` command, which allows a list of commands to be executed when connections are closed, is called to re-form the start configuration.

In backup mode, the primary server maintains consistency among the backup servers. Figure 9 shows the simple primary-backup protocol used by the Tcl-DP name server. The dotted arrows are the ***keepAlive*** messages mentioned above. The client sends an update command to the primary name server. The name server processes the request and updates its state. It then sends an update-state request to the backup server. Without waiting for acknowledgment from the backup, the primary sends its response to the client. Each server also writes its state onto a local disk. The justification for writing to local disk is to avoid writing onto an NFS-mounted file system which can hang the server when it goes down. This scheme makes crash recovery almost instantaneous because the major source of delay is detecting the bad connection.

The external-module interface is currently implemented using the auto-loading feature of Tcl and the RPC command checking feature of DP. The routines from an external module must all have the same unique prefix that is known to the name server. When an RPC request is sent to the name server, the command check procedure will match the routine prefix with the ones known to the name server. If it matches and the routine does not exist, the name server will automatically source the module containing the routine. This approach prevents naming conflicts among different modules. However, it does not prevent a module from renaming procedures and corrupting data. A better approach is to create a separate Tcl interpreter for each external module with read-only access to name server data. Again, each module is identified by its unique prefix and the correct interpreter will be selected to interpret a RPC. We plan to modify the implementation of name server extension using this technique in a future release.

5. Discussion

A prototype of the Tcl-DP name server has been implemented. The Berkeley Plateau Multimedia Project has used it in the implementation of CMT and BVODS systems. The name server has been included in Tcl-DP re-

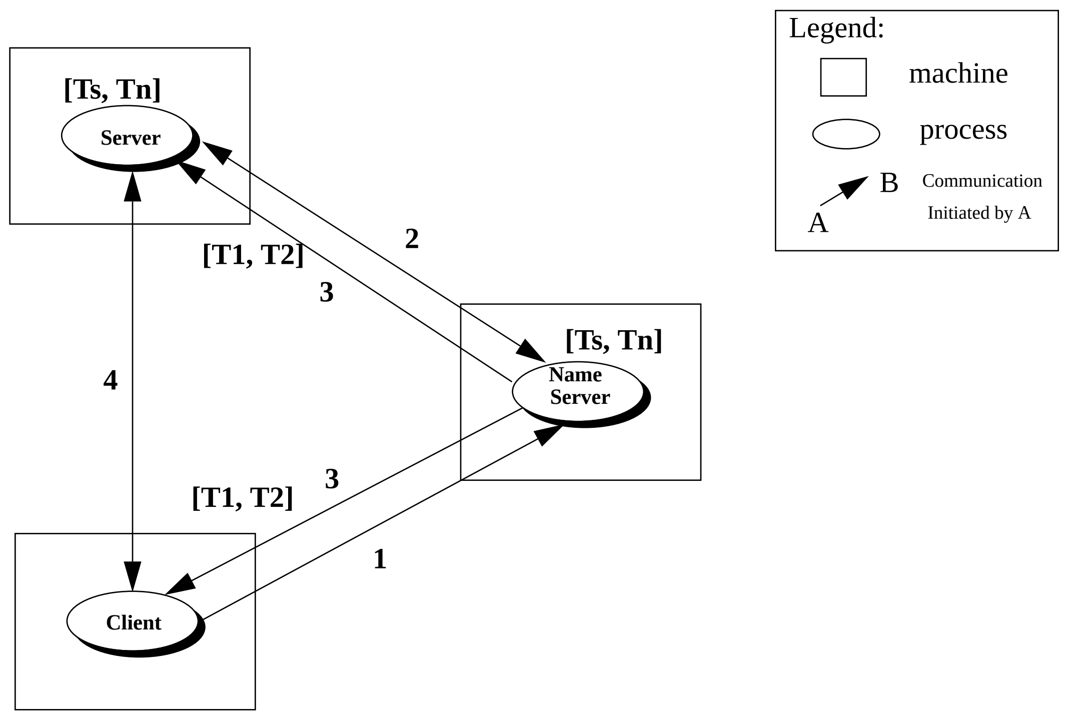


FIGURE 7. Authentication Scheme

6. The new server process connects to the name server process and register its host name and port number to the name server. Note that the server process gets the name from its command line used to start it. The service name is appended to the command line when the name server starts up the server.
7. The client gets the host address and port number of the server process from the name server
8. The client connects to the server process.

Launching servers using `rsh` is similar except that `inetd` and `launchd` are bypassed.

Every auto-started servers shares a pair of random tickets with the name server. The server can retrieve these tickets from `stdin` which is written by the name server. The tickets are used to authenticate the server to the name server and vice versa. For example, when a server registers itself to the name server, it will send its half of the ticket pair to the name server. The name server then verifies the ticket with its internally maintained table of tickets. If the ticket is not valid, the name server will ignore the server

A client application can also ask the name server to authenticate a particular server by using the technique described above. If a server is authentic, the name server will generate a new pair of tickets to be shared between the application and the server. The server and the application can then use the tickets to authenticate each other. Figure 7 is a diagram that depicts the name server security scheme:

1. The client requests the name server to authenticate a server.
2. The name server and the service server exchanges tickets [Tn,Ts]. If the server has the right ticket, a new pair of tickets are generated by the name server [T1,T2].
3. The name server sends the new ticket pair to the server and the client.
4. The server and the client can then use the tickets to authenticate each other.

Currently, tickets are generated by the system clock, and they are not encrypted. This authentication strategy is similar to the one employed by Kerberos [Steiner88]. An authentication is done at the name server level because it is easy to implement and does not require changes to Tcl-DP. However, if Tcl-DP were modified to run on Kerberos, this feature would become obsolete.

The Tcl-DP name server can be configured to run in either stand-alone or backup mode. In stand-alone mode, the name server is a single point of failure that can bring

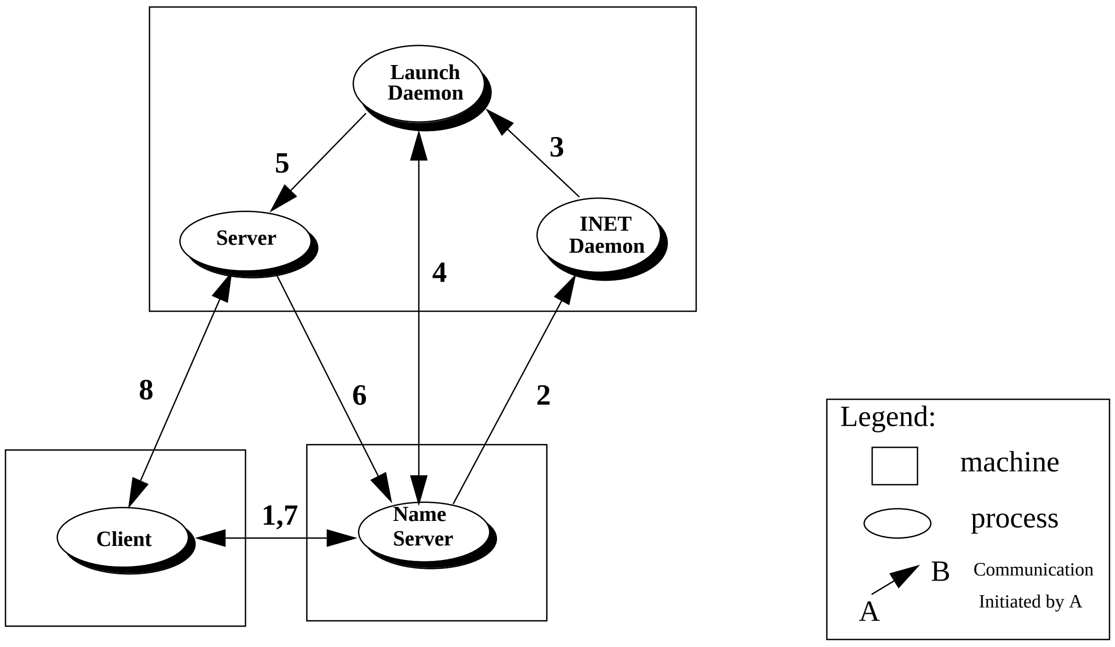


FIGURE 6. Auto-Starting Services

minister the name server including routines to add, delete, alias and edit services. These commands are used to implement the name server monitor.

Lastly, there is a set of commands for authentication between the name server and its servers, and between a server and its clients. The following code segment illustrates its usage by a client application:

```
set ticketPair [ns_AuthenticateService /
cms/bugs-bunny]
set myTicket [lindex $ticketPair 0]
    # get client's ticket
set srvcTicket [lindex $ticketPair 1]
    # get service's ticket
if {$srvcTicket!= [dp_RPC $srvc_fd
ns_Authenticate $myTicket]} {
    error "Service authentication failed!"
}
```

The remainder of this section describes the implementation of the following functions: 1) auto-starting services, 2) using authentication, 3) configuring a reliable name server, and 4) extending name server functionality. When an application needs to connect to a service, it will ask the name server for the host address and port number of the server. If a server is not currently running for an *auto-start* service, the client can request the server to be started either using `rsh` or `inetd`.

Using `rsh` to launch a server has the disadvantage that when the name server crashes, the server may go down as well because `rsh` maintains a connection between the process and the process it starts. `Inetd` avoids this problem by not maintaining a connection. A server is launched using `inetd` as follow (see Figure 6):

1. Client process requests the name server to launch a service.
2. The name server connects to the inet daemon (`inetd`) running on the host on which the server is to be started.
3. `Inetd` executes the launch daemon (`launchd`) which opens a connection with the name server for `launchd`, and exits.
4. The name server requests `launchd` to start the server. `Launchd` is a Tcl script that executes the server on behalf of the name server. It also checks whether a server is legal so a malicious user cannot request a dangerous program such as `rm`. The success or failure of the launching procedure is reported back to the name server.
5. Once the service is started, `launchd` exits which closes the connection with the name server.

Category	**Command**	**Description**
Client	*ns_ListServices*	return a list of services
	ns_FindServices	return hosts and ports
	ns_LaunchServices	auto-start services
	ns_ServiceState	return the states of services
Service	*ns_AdvertiseService*	advertise the service to the name server at start-up
	ns_UnadvertiseService	unadvertise the service
Administration	*ns_Register addService*	add a new auto-start service
	ns_Register deleteService	delete an auto-start service
	ns_Register editService	modify an existing service
	ns_Register aliasService	create an alias to an existing service
	ns_Register infoService	return administration info about a service
Authentication	*ns_AuthenticateService*	authenticate the service and return new tickets to the client and the service
	dp_RPC ns_Authenticate	authentication between a service and a client using the tickets returned by ns_AuthenticateService

Table 1: Tcl-DP Name Server Commands

of Tcl-DP. The client library package provides a wrapper around the actually RPCs which hides the remote nature of the name server. The commands are grouped into four categories shown in Table 1. The first category includes the client routines. These routines include commands for listing, locating and launching services, plus routines for inquiring and resetting service status. By intermixing the different commands, a client can implement different behaviors. For example, a client can first list all services and pick the ones to which it wishes to connect. It can then get the host address and port number on which the service is running and connect to it. If the service crashes, the client can ask the name server to reset the server and launch it again. The following code segment illustrates this case:

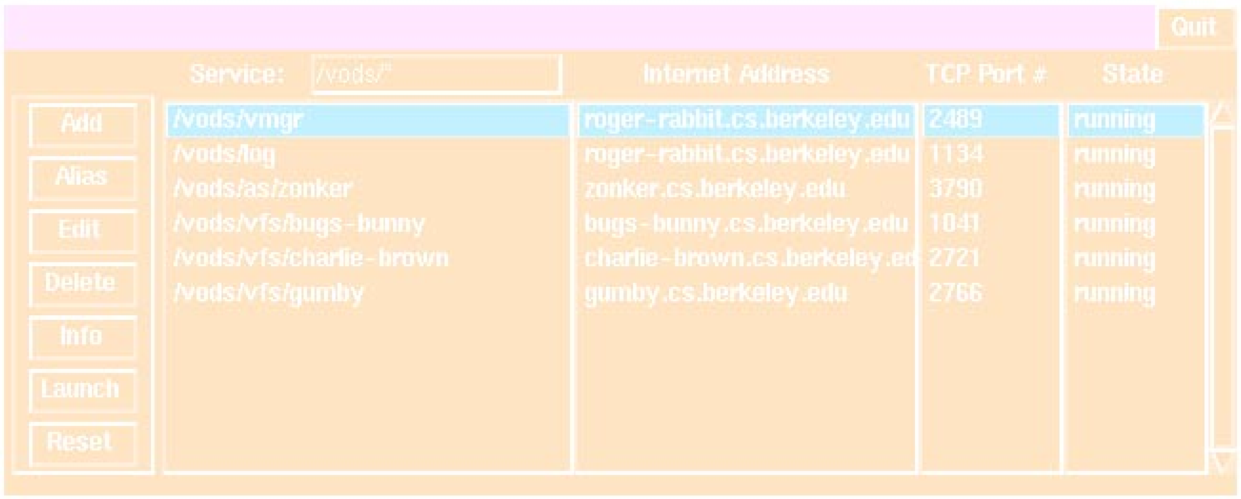
```
set srvc [lindex [ns_ListServices *] 0]
    # get a list of all the services and pick
the first one
set hp [ns_FindServices $srvc]
    # get the host port
if [catch "dp_MakeRPCClient $hp"] {
    # if the service has crashed
    ns_LaunchServices $srvc
    # ask the name server to launch it
again
    # wait for the service to come up
}
```

The second category is composed of routines for the service interface to the name server which are used by a server to advertise services. For example, the following Tcl code has to be executed by a server when it starts up:

```
ns_AdvertiseService /cms/bugs-bunny
bugs-bunny 1234 $ticket
```

"/cms/bugs-bunny" is the service name. *Bugs-bunny* is the host address of the service, and *1234* is the port number. *$ticket* is the authentication ticket issued by the name server.

The third category is composed of routines used to ad-



Service: /vods/*

Service	Internet Address	TCP Port #	State
/vods/vmgr	roger-rabbit.cs.berkeley.edu	2489	running
/vods/log	roger-rabbit.cs.berkeley.edu	1134	running
/vods/as/zonker	zonker.cs.berkeley.edu	3790	running
/vods/vfs/bugs-bunny	bugs-bunny.cs.berkeley.edu	1041	running
/vods/vfs/charlie-brown	charlie-brown.cs.berkeley.ed	2721	running
/vods/vfs/gumby	gumby.cs.berkeley.edu	2766	running

FIGURE 5. Name Server Monitor

Also notice that all processes communicate with the name server running on the Primary machine. If the primary name server goes down, the name server running on the Backup machine will take over. Subsequent queries will be directed to the backup server. This example illustrates the backup capability of the Tcl-DP name server. The connection maintained between the two servers is used for crash detection and update propagation.

As seen from the BVODS example, the main advantage of using this naming scheme is that it avoids naming conflicts. As long as an application can specify a unique name, such as BVODS, services provided by the application can be assigned globally unique names.

Another advantage of using this service naming scheme is that a client can easily find available services under BVODS by having the name server do a glob-style expansion on the pattern "/BVODS/*". One can think of situations in which this feature can be very useful. For example, we can have services named for the machines on which they run (i.e., "/Service/roger-rabbit", "/Service/zonker", etc.) A client first queries for the name of all services running on different machines by issuing a list services command on the pattern "/Service/*", and picking the one to which it wants to connect. If a particular service is overloaded, it can pick another one from the list and switch to it. This capability allows a simple client approach to load-balancing. One can also think of a situation in which this capability can be used as a backup scheme. If a service crashes on a machine and cannot be restarted, a client can pick a server running on a different machine. In addition, new services can be easily added without having to make major changes.

Having the Tcl-DP name server launch servers has two advantages. First, the service is guaranteed to be available as long as the machines are up. Second, the name server can issue authentication tickets to authenticate servers.

In summary, the Tcl-DP name server uses a hierarchical naming scheme that can be used to organize and manage their services as well as providing fault-tolerance and simple load-balancing. The fault-tolerant design of the name server makes it highly reliable.

4. Name Server Implementation

This section describes the implementation of the Tcl-DP name server. The name server is written in Tcl using the DP extension for all interprocess communication. Several new Tcl commands have also been added to the *dpsh* to support running the name server as a daemon and generating random authentication tickets. The Tk windowing toolkit for building Motif-style user interfaces is used to implement the name server monitor, shown in Figure 5, which displays the current status of registered services. Users can restrict the services displayed by specifying a glob pattern, such as "/vods/*" in the **Service** entry box. The monitor also allows users to add, alias, edit, delete, launch, and reset services.

The Tcl-DP name server is composed of a small set of commands that clients invoke through the RPC facility

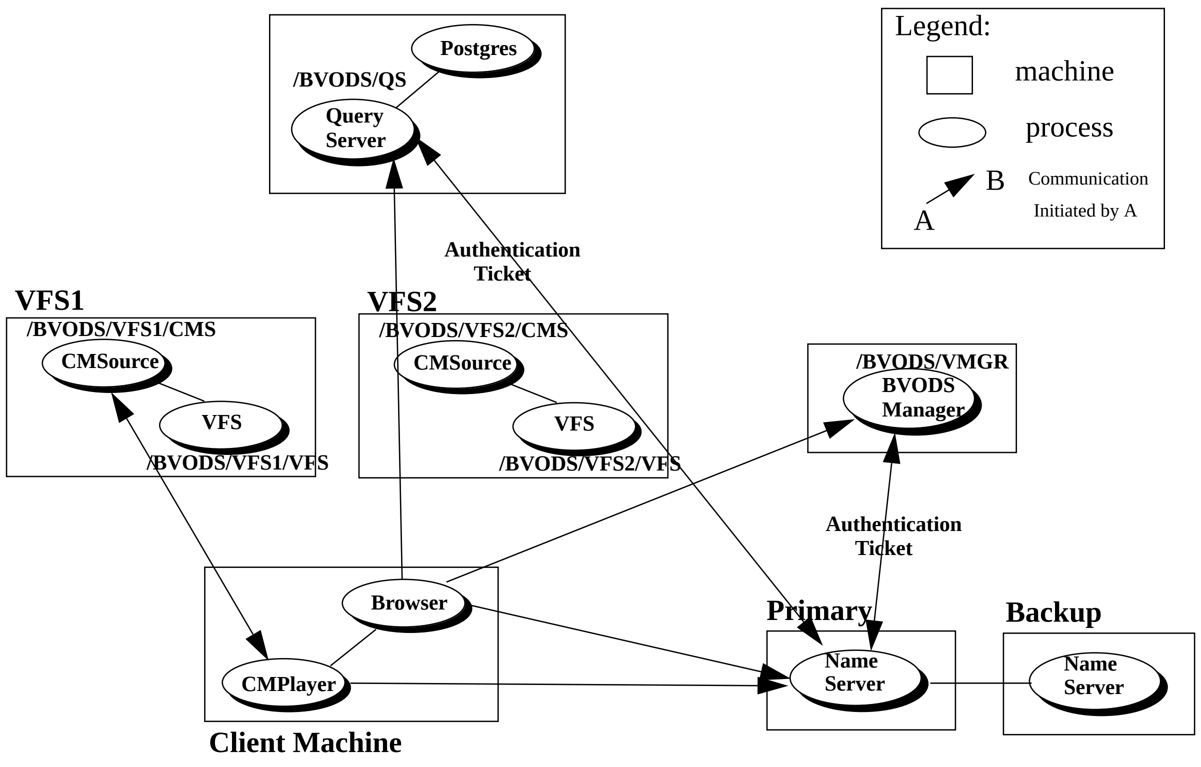


FIGURE 4. BVODS Process Architecture

server achieves this flexibility and generality by allowing applications to organize their services in a hierarchical manner as if they are files in a directory. In addition, aliases can be created for services shared by different applications.

Before we discuss the advantages of the naming scheme, an example of how the name server can be used in a real distributed system is given. Figure 4 shows an example of using the name server to manage BVODS. Boxes represent machines, ovals represent processes, and edges represent communication paths. The labels in the process ovals are service names. The labels next to the process ovals are the service names registered with the name server. As can be seen from the figure, all service names have the prefix "/BVODS/" indicating that they are BVODS services. Also, communication to the name server is initiated through client library routines that are wrappers around the actual RPC calls.

BVODS is a distributed video-on-demand system that is suitable for large video libraries. Users search for videos through a video browser. The browser queries information about a video by connecting to a remote query server (QS) which issues commands to a POSTGRES database that contains indexes to the video library [Stonebraker91]. The browser locates the QS by first issuing a lookup query to the name server using "/BVODS/QS" as the key. If the QS is running, the name server will return the host address and the port number of the server which the browser can use to connect to the QS. If the QS is not running, the name server will start the QS server on behalf of the browser. The authentication ticket, which is a random number generated by the name server, is passed on to the service. Once the QS is up and running, it will register its existence with the name server using the authentication ticket as a proof of legitimacy. When the user identifies a movie he or she wants to view, the browser locates the BVODS Manager (VMGR) using the service name "/BVODS/VMGR" and connects to it. It then asks the VMGR to determine if the movie is available on one of the local video file servers. If so, the browser asks the VMGR for the machine and path names for all components of the video and launches the CMPlayer to play them. The CMPlayer process opens a video window and contacts the CMSource processes, on each of the machines participating in the playback [Rowe92]. The CMSources can be located through the name server using the name "/BVODS/VFS1/CMS" and "/BVODS/VFS2/CMS" which, in reality, are aliases for the services "/CMS/Host1" and "/CMS/Host2".

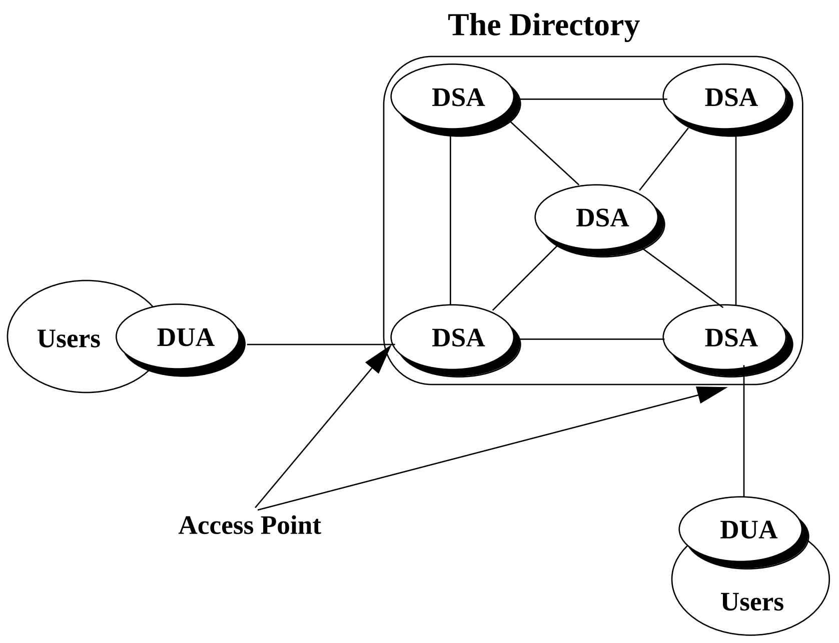


FIGURE 3. CCITT X.500 Directory Services

the address of a server and auto-starting server if necessary.

The CCITT X.500 Directory System provides a framework for the development of a worldwide standard for directory services [Sykas91]. The information contained in the Directory System is called the Directory Information Base (DIB). It is organized in a tree, the Directory Information Tree (DIT), similar to the DNS database. The information about each entity is contained in an entry. Each entry consists of a set of attributes. Examples of attributes are country name, street address, title, postal address, telephone number, ISDN address and OSI address. Each entry is identified by its unique Distinguished Name (DN). An example of a DN is "/.../C=US/O=Berkeley". The prefix "/..." is the name of the global root. *C* stands for country, and *O* stands for Organization. Aliases can also be created to reference other entries. The user interacts with a Directory User Agent (DUA) which is served by a Directory System Agent. Figure 3 depicts the Directory System as a whole. The Directory may also be composed of a centralized DSA rather than distributed ones as shown in Figure 3. Each DSA holds a fragment of the DIB which comprises one or more naming contexts. A naming context is a partial subtree of the DIT. A DSA may use information stored in its local database or interact with other DSAs to carry out requests. Three modes of DSA interaction are defined: chaining, multicasting and referral. A mixture of the three is also possible. Chaining allows one DSA to pass on a remote operation to another DSA when the former has specific knowledge about naming contexts held by the latter. Multicasting allows a DSA to pass on an identical remote operation in parallel or sequentially to one or more DSAs. Referral allows a reference to another DSA to be returned to the DUA or DSA.

The Tcl-DP name server discussed in this paper is designed to be a local name service with backups similar to the CDS in DCE. It is not designed to be a global directory service like DNS and DCE. It only maintains information about processes running on machines distributed in a local domain. However, certain ideas used in the design of the name server are similar to other directory services (i.e. the naming convention). Also, the Tcl-DP name server can be extended using the external-module interface to support higher level abstractions found in directory services.

3. Name Server Design

This section describes the naming scheme used by the Tcl-DP name server and how it can be used in a distributed application such as the Berkeley Distributed Video-On-Demand System (BVODS) [Rowe95].

The Tcl-DP name server is designed to be general-purpose and flexible. We want different applications to share the name server and use it for all types of services they provide. We also want new services to be easy to add without affecting other applications. The Tcl-DP name

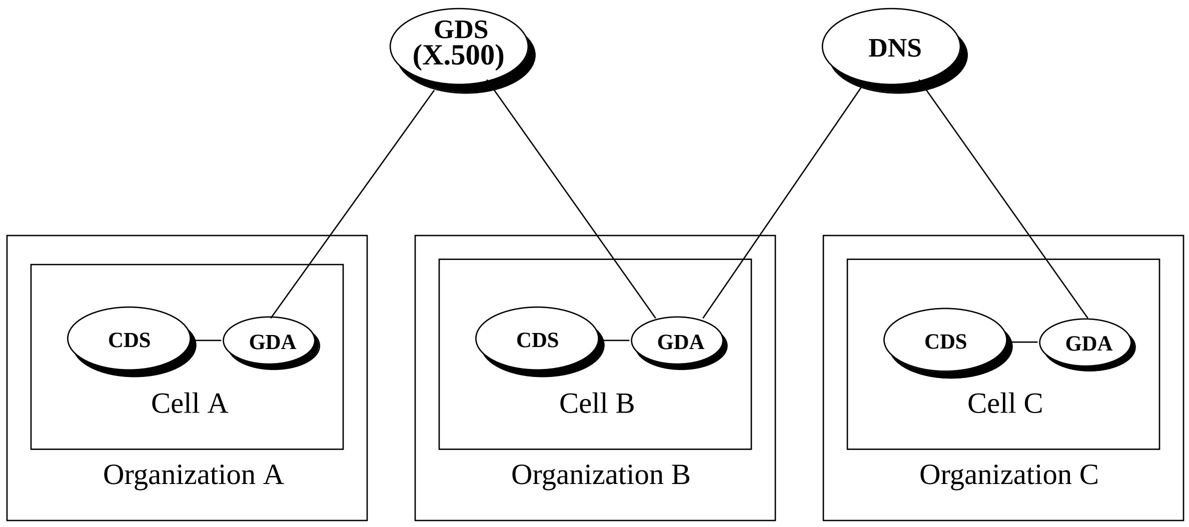


FIGURE 2. DCE Directory Services

servers usually maintain more than one level of information, and they cache complete or partial results of queries. Also, name servers can be replicated to increase availability and throughput. Two types of name servers are used: *primary masters* and *secondary masters*. A primary master loads data from files on the host on which it runs. A secondary master loads data by periodically querying a primary master.

Another well-known name server is the DCE Directory Service which is comprised of three components: 1) Cell Directory Service (CDS), 2) Global Directory Service (GDS) and 3) Global Directory Agent (GDA) [OSF92]. CDS is a local directory service that stores names and attributes of resources located in a DCE cell. It is optimized for local access and replicated for reliability. GDS is a distributed, replicated directory service based on the CCITT X.500 international standard discussed below. It is used when looking up a name outside of the local DCE cell. GDS and DNS together can act as the high-level connectors that allows independent cells to interact with one another. GDS also works with other X.500 implementation which means it can participate in the worldwide X.500 directory service. GDA is the intermediary between a cell's CDS and the rest of the world. It takes a name that cannot be found in its local cell and queries foreign cells to find it using GDS or DNS, depending on where the foreign cell is registered. Figure 2 shows the relationship between the different components of the DCE directory system. The DCE namespace is also hierarchical. It supports names used by DNS and typed names using the X.500 syntax discussed below.

Grapevine is a distributed database running at Xerox Palo Alto Research Center that has been used to provide name service lookup for the Xerox Remote Procedure Call (RPC) package [Birrell82, Birrell84]. It is highly reliable and can be configured to replicate data on different servers. There are two types of entries in Grapevine: *individuals* and *groups*. As far as the RPC package is concerned, for each individual entry there is a *connect-site*, which is a network address, and for each group there is a *member-list* which is a list of individual and group entries. Using these two types of entries, one can create complex hierarchies of services similar to the domain hierarchy in DNS. To avoid long delays during client requests, each client operation is performed at a single site and the result is later propagated to other name servers in the background. The disadvantage of this tactic is that inconsistency may occasionally occur. A more recent system design, such as Global Name Service (GNS), restricts this nondeterminism by periodically sweeping all replicas to ensure all updates are properly propagated [Lampson86].

Other name servers are provided by distributed programming systems (e.g. ISIS, ASN.1, etc.). The Distributed Resource Manager provided by ISIS manages the execution of remote jobs on a local network [ISIS92]. It includes "recycling" idle workstations for remote use and managing a pool of "compute servers." It can also be used to manage a reliable replicated service by ensuring that some desired number of copies of the service are always running, despite machine failures. The Resource Manager supports a subroutine interface for looking up

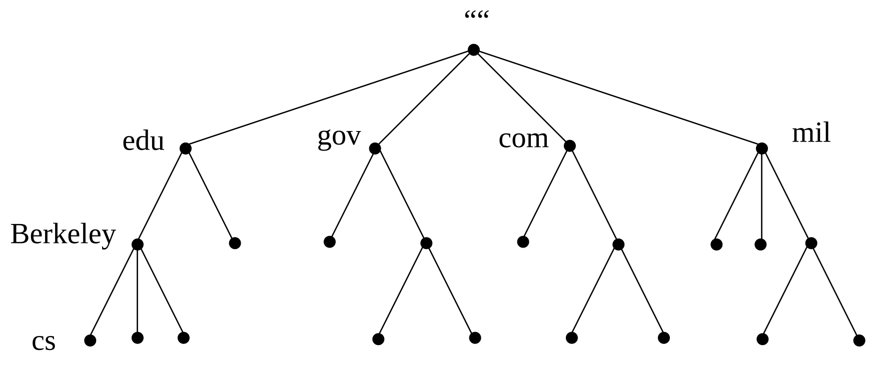


FIGURE 1. DNS Database

For example, a directory service for a conferencing application can be used to keep track of the users currently logged onto the different conferencing servers running on the network. A user can lookup a person from the directory service and ask for the location of his/her conferencing server. Without the external-module interface, the user will have to first query the name server for the location of the directory service and then ask the directory service for the location of the person he/she wants to talk to. The directory service will in turn ask the name server for the location of the desired conferencing server. With the external-module interface, since the directory service runs as part of the name server and has direct access to its data, the user only has to contact the name server to get the desired information, thus eliminating two levels of indirection. Therefore, the interface mechanism can significantly reduce the response time of certain applications. The purpose of these external modules should be to extend the name server functionality or provide higher-level abstractions on top of the one provided by the name server.

Considerable interest exists to develop an international standard for name services or directory services due to the diverse requirements raised by the developments in future communication networks. The requirements include integration of new services and sharing of network resources by various communication services. Possible applications of the directory service can be found in Open System Interconnection (OSI), internetworking, broadband networks and mobile communication. For example, in future broadband networks, name services will be used to manage addressing information of all communication entities such as telephones and computer terminals, and Help-Desk information such as information about hotels, hospitals, and airlines. In future mobile networks, the need for a name service is even more pronounced because the current location of all subscribers as well as information about networks and terminals have to be maintained. In the following paragraphs, several well-known name services including the Domain Name System (DNS), DCE Directory Service, Grapevine and others, will be described, followed by an overview of the CCITT X.500 Directory System standard.

The most widely used name servers are the ones that comprise the Domain Name System’s (DNS) client-server mechanism [Albitz92]. DNS is a distributed database that holds the name-to-address mapping for every host connected to the Internet. It is organized in a tree structure similar to a file directory as shown in Figure 1. The DNS namespace is partitioned into domains and subdomains. In DNS, the name of the root is the null label (““). A typical name in the DNS database is **cs.berkeley.ed**u. Figure 1 shows how this name can be stored in the DNS database. A DNS name server contains information about a segment of the database. A client that accesses the name server, called a *resolver*, issues queries by calling a library routine. Given a query about any domain name, the root name servers can provide the names and addresses for the top-level domain in which the name is located. Top-level domain name servers can provide the list of name servers responsible for the second-level domain. The process continues until the entire query is resolved. For example, to resolve the name **cs.-berkeley.edu**, the resolver first sends a query to the root server for the location of the **edu** server. The **edu** server will then be used to retrieve information about the location of the **berkeley** server. The process continues until the actual location is found. In this case, the resolver will be given the IP address of the CS Division machine which is **128.32.34.35**. To speed up the process, name to access them. It should provide an interface to query such information using unique name associated with each service. Consequently, the name server must enforce a naming convention on services. The Tcl-DP name server uses a hierarchical namespace, similar to other name servers and the structure of a UNIX file system [Ritchie74]. Section 3 will discuss the naming convention in more detail. Every server that offers a service under the name server's management must inform the name server of its existence at start-up time and its demise when it terminates. When a client needs to connect to a service, it queries the name server for the host and port number on which the service is running using the service name as the key. The only information the client needs to know in advance is the host address and port number of the name server plus the names of the services to which it wants to connect. As new services are added to a distributed system, only the service names need to be advertised to the clients. Also, services can be freely moved around on the local network without affecting the clients.

In a distributed environment, services must be highly available because the livelihood of the clients may depend on them. In certain cases, services must run 24 hours a day and 7 days a week. The name server can provide the illusion of 24-by-7 service by restarting the service after it fails. In essence, the name server acts as a service launcher. It assumes that services are either stateless or can recover from disk. If a server process is detected by a client to have crashed, the client can ask the name server to restart the server process. If the server cannot be restarted on a particular machine because that machine has crashed, the client can request connection to the same service running on a different machine. This assumes that multiple processes offering the same service are distributed across the local network. With this capability, failure recovery in case of server crash or a stale network connection is simple. Therefore, unless the network is partitioned or the name server goes down, services are guaranteed to be available at all times. We call this feature auto-starting.

As a dual feature to the auto-starting capability, the name server can also terminate services on demand. One can think of situations in which this capability is required. For example, a service can get into a bad state in which it must be killed before it can be restarted. The name server is the perfect agent to carry out this task because it maintains information such as the process identifier and location of the server that provides the service.

Since the name server is capable of launching and terminating services, it should also have the ability to authenticate such requests to prevent malicious programs from taking over as legitimate services. Therefore, an authentication protocol is needed between the name server and the services it manages. A similar protocol is needed between clients and servers in case they do not trust one another. Effectively, the name server is acting as a security checkpoint in the distributed system. A client can request the name server to check the server on its behalf. The name server can also set things up between the client and server so they can authenticate each other. However, it is still the responsibility of the application to restrict commands and connections to clients.

The Tcl-DP name server uses a ticket-based scheme for authentication. When a service is auto-started, the name server issues a random ticket to the server process. The server process must present this ticket when it reports to the name server. Servers and clients can also authenticate one another using tickets issued by the name server. This approach is similar to the one used by Kerberos [Steiner88].

To implement the functions mentioned in the preceding paragraphs, the name server must itself be highly available and fault-tolerant. One way to implement a fault-tolerant name server is to use multiple servers that fail independently. The state of a service provided by the name server is replicated and distributed among these servers, and updates are coordinated so that when a subset of the servers fail, the service remains available. There are generally two approaches to this fault-tolerant architecture. One approach is to replicate the service state in all servers and to present client requests, in the same order, to all servers. This approach is commonly called *active replication* or the *state-machine* approach [Schneider90]. The other approach is to designate one server as the *primary* and all other servers as *backups*. This approach is called the *primary-backup* or *primary-copy* approach [Alsberg76]. Clients send requests only to the primary. If the primary fails, a *failover* occurs and one of the backups takes over as the new primary. The state-machine approach is more costly because it requires all client requests to be presented to each server, and each server must process the requests in the same relative order. The advantage is that no requests are lost when a subset of the servers fail. The primary-backup approach, on the other hand, is simpler and less costly but suffers from lost requests when failure occurs. The Tcl-DP name server employs the second approach because it is simple and we assume that most applications can tolerate occasional lost requests.

The name server should also provide mechanisms for dynamically interfacing with external modules without interrupting its own operations. The idea is to have the name server auto-load these modules into its address space at run time and allow them direct access to its data.

Tcl-DP Name Server

Peter T. Liu
Brian Smith
Lawrence Rowe
Computer Science Division - EECS
University of California
Berkeley, CA 94720-1776
(pliu@CS.Berkeley.EDU)

Abstract

This paper describes a general purpose name server for Tcl-DP. This name server maintains host addresses and port numbers of services running in a distributed environment and allows clients to query about them. It starts services on demand so services are guaranteed to be available, and it provides a simple authentication protocol for better security. The Tcl-DP name server is also designed to be fault-tolerant. Multiple backup servers can be started on different hosts, and a failover occurs when the main server goes down. In addition, the name server provides mechanisms to interface with external modules for extending its functionality.

1. Introduction

Tcl-DP [Smith93] is a distributed programming extension to Tcl [Ousterhout94]. It provides TCP and IP connection management, blocking and nonblocking remote procedure call (RPC), and a simple distributed object system. It is a high-level scripting language for building distributed client-server applications such as the Berkeley Continuous Media Toolkit (CMT) [Rowe94] and the Berkeley Distributed Video-On-Demand System (BVODS) [Federighi94, Rowe95].

Despite its convenience, Tcl-DP has several shortcomings. First, the system does not provide service management to client-server applications. Tcl-DP clients must use explicit host addresses and port numbers to locate services available to them, unless a customized service registry is implemented. This approach leads to system management and reliability problems in a large and rapidly changing distributed system. Second, Tcl-DP does not provide automatic failure recovery in the event of a server crash or a broken network connection. Services offered by a server process will be unavailable if the process crashes. Therefore, there is no guarantee of service availability in Tcl-DP applications. Finally, Tcl-DP provides limited security. The current release (Tcl-DP v3.2) allows applications to restrict commands that can be called remotely and limit connections to clients running on specific hosts. However, it does not prevent malicious programs from taking over as legitimate services nor does it support finer granularity access control. An authentication protocol is needed to provide better security.

The Tcl-DP name server described in this paper is designed to address these problems. It solves the explicit host address and port number and availability issues, and it provides a simple authentication protocol for applications to increase security. In addition, it is fault tolerant and provides mechanisms for extending its functionality. The remainder of this paper is organized as follow. Section 2 describes the functionality of the Tcl-DP name server and other name services in existence. Section 3 presents an overview of the name server design and describes how it fits in a real application. Section 4 discusses the implementation of the system. And, section 5 describes the current status of the system, the experiences with the current implementation and possible future extensions.

2. Name Server Functionality

This section describes the functionality of the Tcl-DP name server. First, the name server must maintain information about services and respond to client queries about them. Second, it must start services on demand so they are guaranteed to be available at all times. Third, an authentication protocol is needed to protect the system from malicious users. Fourth, the name server itself must be fault-tolerant which means it must recover successfully if the name server itself or the host it runs on crashes. Finally, the name server should provide mechanisms to interface with external modules without interfering with its normal operations. These issues are discussed in this section followed by descriptions of other name servers.

The name server must maintain information about available services running on the system, such as the service names and the host addresses and port numbers needed