16. Lampson, B. and Sturgis, H.K. *Crash Recovery in a Distributed System.* (unpublished), Xerox Palo Alto Research Center, 1979.
17. Liskov, Barbara. *Linguistic support for distributed programs: a status report.* Laboratory for Computer Science Computation Structures Group Memo 201, MIT, Cambridge, 1980.
18. Metcalfe, R.M., and Boggs, D.R. Ethernet: distributed packet switching for local computer networks. *Comm. ACM 19*, 7 (July 1976) 395–404.
19. Nelson, Bruce Jay. *Remote Procedure Call.* Ph.D. Dissertation, Report CMU-CS-81-119, Carnegie-Mellon University, Pittsburgh, PA, 1981.
20. Ousterhout, John K., Scelza, Donald, A., and Sindhu, Pradeep. Medusa: an experiment in distributed operating system structure. *Comm. ACM 23*, 2 (Feb. 1980), 92–105.
21. Peterson, James L. Notes on a workshop on distributed computing. *Operating Systems Review 13*, 3 (July 1979), 18–27.
22. Popek, G., et al. Locus: A network transparent, high reliability distributed system. *Proc. 8th Symp. on Operating System Principles*, Dec. 1981, 169–177.
23. Rawson, E.G., and Metcalfe R.M. Fibernet: multimode optical fibers for local computer networks. *IEEE Trans. on Computer Communication COM-26*, 7 (July 1978), 983–990.
24. Saltzer, J.H. End-to-end arguments in system design. *Proc. 2nd Int. Conf. on Operating Systems.* Paris (April 1981).
25. Saltzer, J.H., Clark, D., and Reed, D. *Version Two Ring Network.* Laboratory for Computer Science Report, MIT, Cambridge, 1981.
26. Spector, Alfred Z. *Multiprocessing Architectures for Local Computer Networks.* Ph.D. Dissertation, Report STAN-CS-81-874, Stanford University, 1981.
27. Swan, R.J., Fuller, S.H., and Siewiorek, D.P. Cm* A modular multi-microprocessor. *Proc. of the National Computer Conference.* June 1977, 636–644.
28. Thacker, C.P., McCreight, E.M., Lampson B.W., Sproull, R.F., and Boggs, D.R. Alto: A personal computer. In Siewiorek, O., Bell, G., and Newell, A. *Computer Structures: Readings and Examples.* Second ed. McGraw Hill, New York, 1981.
29. Wilkes, M.V., and Wheeler, D.J. The Cambridge digital communication ring. *Proc. Local Area Communication Network Symposium.* Boston, May 1979.
30. *ALTO: A Personal Computer System Hardware Manual.* Xerox Palo Alto Research Center, 1979.
31. Zimmerman, H. OSI reference model—the ISO model of architecture for open systems interconnection. *IEEE Trans. on Communication COM-28*, 4 (Apr. 1980), 425–432.

Operating Systems

Anita K. Jones
Editor

Grapevine: An Exercise in Distributed Computing

Andrew D. Birrell, Roy Levin,
Roger M. Needham, and Michael D. Schroeder

Xerox Palo Alto Research Center

Grapevine is a multicomputer system on the Xerox research internet. It provides facilities for the delivery of digital messages such as computer mail; for naming people, machines, and services; for authenticating people and machines; and for locating services on the internet. This paper has two goals: to describe the system itself and to serve as a case study of a real application of distributed computing. Part I describes the set of services provided by Grapevine and how its data and function are divided among computers on the internet. Part II presents in more detail selected aspects of Grapevine that illustrate novel facilities or implementation techniques, or that provide insight into the structure of a distributed system. Part III summarizes the current state of the system and the lessons learned from it so far.

CR Categories and Subject Descriptors: C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*distributed applications, distributed databases*; C.4 [**Performance of Systems**]—*reliability, availability and serviceability*; D.4.7 [**Operating Systems**]: Organization and Design—*distributed systems*; H.2.4 [**Database Management**]: Systems—*distributed systems*; H.2.7 [**Database Management**]: Database Administration; H.4.3 [**Information Systems Applications**]: Communications Applications—*electronic mail*

General Terms: Design, Experimentation, Reliability

Part I. Description of Grapevine

1. Introduction

Grapevine is a system that provides message delivery, resource location, authentication, and access control ser-

Authors' Present Addresses: Andrew D. Birrell, Roy Levin, and Michael D. Schroeder, Xerox Palo Alto Research Center, Computer Science Laboratory, 3333 Coyote Hill Road, Palo Alto, CA 94304; Roger M. Needham, University of Cambridge Computer Laboratory, Corn Exchange Street, Cambridge, CB2 3QG, United Kingdom.

vices in a computer internet. The implementation of Grapevine is distributed and replicated. By *distributed* we mean that some of the services provided by Grapevine involve the use of multiple computers communicating through an internet; by *replicated* we mean that some of the services are provided equally well by any of several distinct computers. The primary use of Grapevine is delivering computer mail, but Grapevine is used in many other ways as well. The Grapevine project was motivated by our desire to do research into the structure of distributed systems and to provide our community with better computer mail service.

Plans for the system were presented in an earlier paper [5]. This paper describes the completed system. The mechanisms discussed below are in service supporting more than 1500 users. Designing and building Grapevine took about three years by a team that averaged two to three persons.

1.1 Environment for Grapevine

Figure 1 illustrates the kind of computing environment in which Grapevine was constructed and operates. A large internet of this style exists within the Xerox Corporation research and development community. This internet extends from coast-to-coast in the U.S.A. to Canada, and to England. It contains over 1500 computers on more than 50 local networks.

Most computing is done in personal *workstation* computers [12]; typically each workstation has a modest amount of local disk storage. These workstations may be used at different times for different tasks, although generally each is used only by a single individual. The internet connecting these workstations is a collection of Ethernet local networks [6], gateways, and long distance links (typically telephone lines at data rates of 9.6 to 56 Kbps). Also connected to the internet are *server* computers that provide shared services to the community, such as file storage or printing.

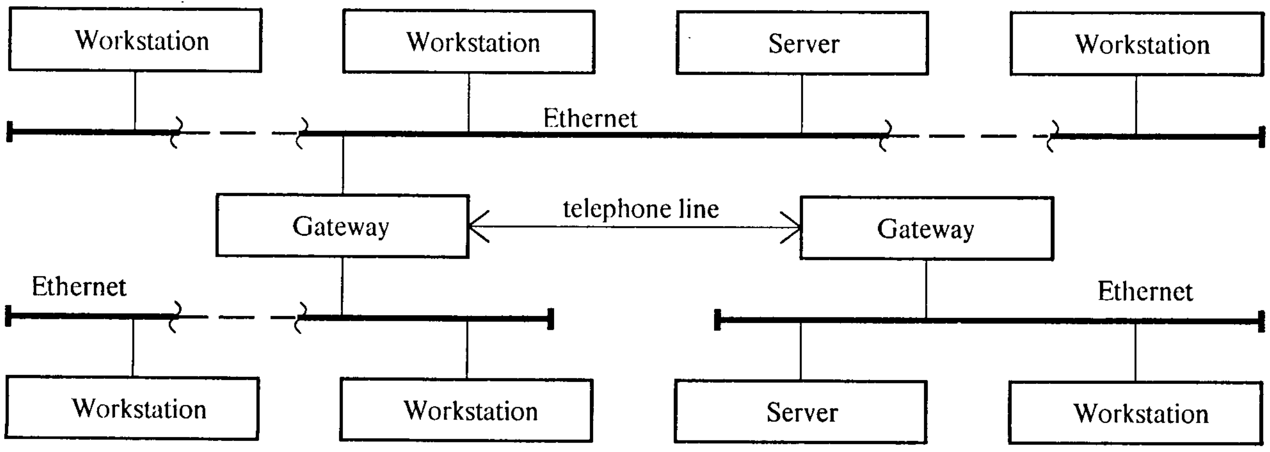
Protocols already exist for communicating between computers attached to the internet [11]. These protocols provide a uniform means for addressing any computer attached to any local network in order to send individual packets or to establish and use byte streams. The individual packets are typically small (up to 532 bytes), and are sent unreliably (though with high probability of success) with no acknowledgment. The byte stream protocols provide reliable, acknowledged, transmission of unlimited amounts of data [1].

1.2 Services and Clients

Our primary consideration when designing and implementing Grapevine was its use as the delivery mechanism for a large, dispersed computer mail system. A computer mail system allows a group of human users to exchange messages of digital text. The sender prepares a message using some sort of text editing facility and names a set of recipients. He then presents the message to a delivery mechanism. The delivery mechanism moves the message from the sender to an internal buffer for each recipient, where it is stored along with other messages for that recipient until he wants to receive them. We call the buffer for a recipient's messages an *inbox*. When ready, the recipient can read and process the messages in his inbox with an appropriate text display program. The recipient names supplied by the sender may identify *distribution lists*: named sets of recipients, each of whom is to receive the message. We feel that computer mail is both an important application of distributed computing and a good test bed for ideas about how to structure distributed systems.

Buffered delivery of a digital message from a sender to one or more recipients is a mechanism that is useful in many contexts: it may be thought of as a general communication protocol, with the distinctive property that the recipient of the data need not be available at the time the sender wishes to transmit the data. Grapevine separates this message delivery function from message creation and interpretation, and makes the delivery function available for a wider range of uses. Grapevine does not interpret the contents of the messages it transports. Interpretation is up to the various message manipulation programs that are software *clients* of Grapevine. A client

Fig. 1. An Example of a Small Internet.



program implementing a computer mail user interface will interpret messages as interpersonal, textual memos. Other clients might interpret messages as print files, digital audio, software, capabilities, or data base updates.

Grapevine also offers *authentication*, *access control*, and *resource location* services to clients. For example, a document preparation system might use Grapevine's resource location service to find a suitable printing server attached to the internet (and then the message delivery service to transfer a document there for printing) or a file server might use Grapevine's authentication and access control services to decide if a read request for a particular file should be honored.

Grapevine's clients run on various workstations and server computers attached to the internet. Grapevine itself is implemented as programs running on server computers dedicated to Grapevine. A client accesses the services provided by Grapevine through the mediation of a software package running on the client's computer. The Grapevine computers cooperate to provide services that are distributed and replicated.

2. Design Goals

We view distributed implementation of Grapevine both as a design goal and as the implementation technique that best meets the other design goals. A primary motivation for the Grapevine project was implementing a useful distributed system in order to understand some system structures that met a real set of requirements. Once we chose message delivery as the functional domain for the project, the following specific design goals played a significant role in determining system structure.

Grapevine makes its services available to many different clients. Thus, it should make no assumptions about message content. Also, the integrity of these services should not in any way depend on correctness of the clients. Though the use of an unsatisfactory client program will affect the service given to its user, it should not affect the service given to others. These two goals help determine the distribution of function between Grapevine and its clients.

Two goals relate to Grapevine's reliability properties. First, a user or client implementor should feel confident that if a message is accepted for delivery then it will either be made available to its intended recipients or returned with an indication of what went wrong. The delivery mechanism should meet this goal in the face of user errors (such as invalid names), client errors (such as protocol violations), server problems (such as disk space congestion or hardware failures), or communication difficulties (such as internet link severance or gateway crashes). Second, failure of a single Grapevine server computer should not mean the unavailability of the Grapevine services to any client.

The typical interval from sending a message to its arrival in a recipient's inbox should be a few minutes at most. The typical interactive delay perceived by a client program when delivering or receiving a message should be a few seconds at most. Since small additions to delivery times are not likely to be noticed by users, it is permissible to improve interactive behavior at the expense of delivery time.

Grapevine should allow decentralized administration. The users of a widespread internet naturally belong to different organizations. Such activities as admission of users, control of the names by which they are known, and their inclusion in distribution lists should not require an unnatural degree of cooperation and shared conventions among administrations. An administrator should be able to implement his decisions by interacting directly with Grapevine rather than by sending requests to a central agency.

Grapevine should work well in a large size range of user communities. Administrators should be able to implement decentralized decisions to adjust storage and computing resources in convenient increments when the shape, size, or load patterns of the internet change.

Grapevine should provide authentication of senders and recipients, message delivery secure from eavesdropping or content alteration, and control on use and modification of its data bases.

3. Overview

3.1 Registration Data Base

Grapevine maintains a *registration data base* that maps names to information about the users, machines, services, distribution lists, and access control lists that those names signify. This data base is used in controlling the message delivery service; is accessed directly for the resource location, access control, and authentication services; and is used to configure Grapevine itself. Grapevine also makes the values in the data base available to clients to apply their own semantics.

There are two types of entries in the registration data base: *individual* and *group*. We call the name of an entry in the registration data base an *RName*.

A group entry contains a set of RNames of other data base entries, as well as additional information that will be discussed later. Groups are a way of naming collections of RNames. The groups form a naming network with no structural constraints. Groups are used primarily as distribution lists: specifying a group RName as a recipient for a message causes that message to be sent to all RNames in that group, and in contained groups. Groups also are used to represent access control lists and collections of like resources.

An individual entry contains an *authenticator* (a password), a list of *inbox sites*, and a *connect site*, as well as additional information that will be discussed later. The inbox site list indicates, in order of preference, the Grapevine computers where the individual's messages may be buffered. The way these multiple inboxes are used is discussed in Sec. 4.2. The connect site is an internet address for making a connection to the individual. Thus, an individual entry specifies ways of authenticating the identity of and communicating with—by message delivery or internet connection—the named entity. Individuals are used to represent human users and servers, in particular the servers that implement Grapevine. Usually the connect site is used only for individuals that represent servers. Specifying an individual RName (either a human or a server) as a recipient of a message causes the message to be forwarded to and buffered in an inbox for that RName.

3.2 Functions

Following is a list of the functions that Grapevine makes available to its clients. Responses to error conditions are omitted from this description. The first three functions constitute Grapevine's *delivery service.*

Accept message:

[sender, password, recipients, message-body] → ok

The client presents a message body from the sender for delivery to the recipients. The sender must be RName of an individual and the password must authenticate that individual (see below). The recipients are individual and group RNames. The individuals correspond directly to message recipients while the groups name distribution lists. After Grapevine acknowledges acceptance of the message the client can go about its other business. Grapevine then expands any groups specified as recipients to produce the complete set of individuals that are to receive the message and delivers the message to an inbox for each.

Message polling:

[individual] → {empty, nonempty}

Message polling is used to determine whether an individual's inboxes contain messages that can be retrieved. We chose not to authenticate this function so it would respond faster and load the Grapevine computers less.

Retrieve messages:

[name, password] → sequence of messages → ok

The client presents an individual's name and password. If the password authenticates the individual then Grapevine returns all messages from the corresponding inboxes. When the client indicates "ok," Grapevine erases these messages from those inboxes.

Grapevine's authentication, access control, and resource location services are implemented by the remaining functions. These are called the *registration service,* because they are all based on the registration data base.

Authenticate:

[individual, password] → {authentic, bogus}

The authentication function allows any client to determine the authenticity of an individual. An individual/password combination is authentic if the password matches the one in the individual's registration data base entry.[1]

Membership:

[name, group] → {in, out}

Grapevine returns an indication of whether the name is included in the group. Usually the client is interpreting the group as an access control list. There are two forms of the membership function. One indicates direct membership in the named group; the other indicates membership in its closure.

Resource location:

[group] → members
[individual] → connect site
[individual] → ordered list of inbox sites

The first resource location function returns a group's membership set. If the group is interpreted as a distribution list, this function yields the individual recipients of a message sent to the distribution list; if the group is interpreted as the name of some service, this function yields the names of the servers that offer the service. For a group representing a service, combining the first function with the second enables a client to discover the internet addresses of machines offering the service, as described in Sec. 5. The third function is used for message delivery and retrieval as described in Sec. 4.

Registration data base update and inquiry:

There are various functions for adding and deleting names in the registration data base, and for inspecting and changing the associated values.

3.3 Registries

We use a partitioned naming scheme for RNames. The partitions serve as the basis for dividing the administrative responsibility, and for distributing the data base among the Grapevine computers. We structure the name space of RNames as a two-level hierarchy. An RName is a character string of the form *F.R* where *R* is a *registry* name and *F* is a name within that registry. Registries can correspond to organizational, geographic, or other arbitrary partitions that exist within the user community. A two-level hierarchy is appropriate for the size and organizational complexity of our user community, but a larger community or one with more organizational diversity would cause us to use a three-level scheme. Using more levels would not be a fundamental change to Grapevine.

[1] This password-based authentication scheme is intrinsically weak. Passwords are transmitted over the internet as clear-text and clients of the authentication service see individuals' passwords. It also does not provide two-way authentication: clients cannot authenticate servers. The Grapevine design includes proper encryption-based authentication and security facilities that use Needham and Schroeder's protocols [9] and the Federal Data Encryption Standard [8]. These better facilities, however, are not implemented yet.

3.4 Distribution of Function

As indicated earlier, Grapevine is implemented by code that runs in dedicated Grapevine computers, and by code that runs in clients' computers. The code running in a Grapevine computer is partitioned into two parts, called the *registration server* and the *message server.* Although one registration server and one message server cohabit each Grapevine computer, they should be thought of as separate entities. (Message servers and registration servers communicate with one another purely by internet protocols.) Several Grapevine computers are scattered around the internet, their placement being dictated by load and topology. Their registration servers work together to implement the registration service. Their message servers work together to implement the delivery service. As we will see in Secs. 4 and 5, message and registration services are each clients of the other.

The registration data base is distributed and replicated. Distribution is at the grain of a registry; that is, each registration server contains either entries for all RNames in a registry or no entries for that registry. Typically no registration server contains all registries. Also, each registry is replicated in several different registration servers. Each registration server supports, by publicly available internet protocols, the registration functions described above for names in the registries that it contains. Any server that contains the data for a registry can accept a change to that registry. That server takes the responsibility for propagating the change to the other relevant servers.

Any message server is willing to accept any message for delivery, thus providing a replicated mail submission service. Each message server will accept message polling and retrieval requests for inboxes on that server. An individual may have inboxes on several message servers, thus replicating the delivery path for the individual.

If an increase in Grapevine's capacity is required to meet expanding load, then another Grapevine computer can be added easily without disrupting the operation of existing servers or clients. If usage patterns change, then the distribution of function among the Grapevine computers can be changed for a particular individual, or for an entire registry. As we shall see later this redistribution is facilitated by using the registration data base to describe the configuration of Grapevine itself.

The code that runs in clients' machines is called the *GrapevineUser package.* There are several versions of the GrapevineUser package: one for each language or operating environment. Their function and characteristics are sufficiently similar, however, that they may be thought of as a single package. This package has two roles: it implements the internet protocols for communicating with particular Grapevine servers; and it performs the resource location required to choose which server to contact for a particular function, given the data distribution and server availability situation of the moment. GrapevineUser thus makes the multiple Grapevine servers look like a single service. A client using the GrapevineUser package never has to mention the name or internet address of a particular Grapevine server. The GrapevineUser package is not trusted by the rest of Grapevine. Although an incorrect package could affect the services provided to any client that uses it, it cannot affect the use of Grapevine by other clients. The implementation of Grapevine, however, includes engineering decisions based on the known behavior of the GrapevineUser package, on the assumption that most clients will use it or equivalent packages.

3.5 Examples of How Grapevine Works

With Fig. 2 we consider examples of how Grapevine works. If a user named *P.Q* were using workstation 1 to send a message to *X.Y.*, then events would proceed as follows. After the user had prepared the message using a suitable client program, the client program would call the delivery function of the GrapevineUser package on workstation 1. GrapevineUser would contact some registration server such as *A* and use the Grapevine resource location functions to locate any message server such as *B*; it would then submit the message to *B*. For each recipient, *B* would use the resource location facilities, and suitable registration servers (such as *A*) to determine that recipient's best inbox site. For the recipient *X.Y*, this might be message server *C*, in which case *B* would forward the message to *C*. *C* would buffer this message locally in the inbox for *X.Y*. If the message had more recipients, the message server *B* might consult other registration servers and forward the message to multiple message servers. If some of the recipients were distribution lists, *B* would use the registration servers to obtain the members of the appropriate groups.

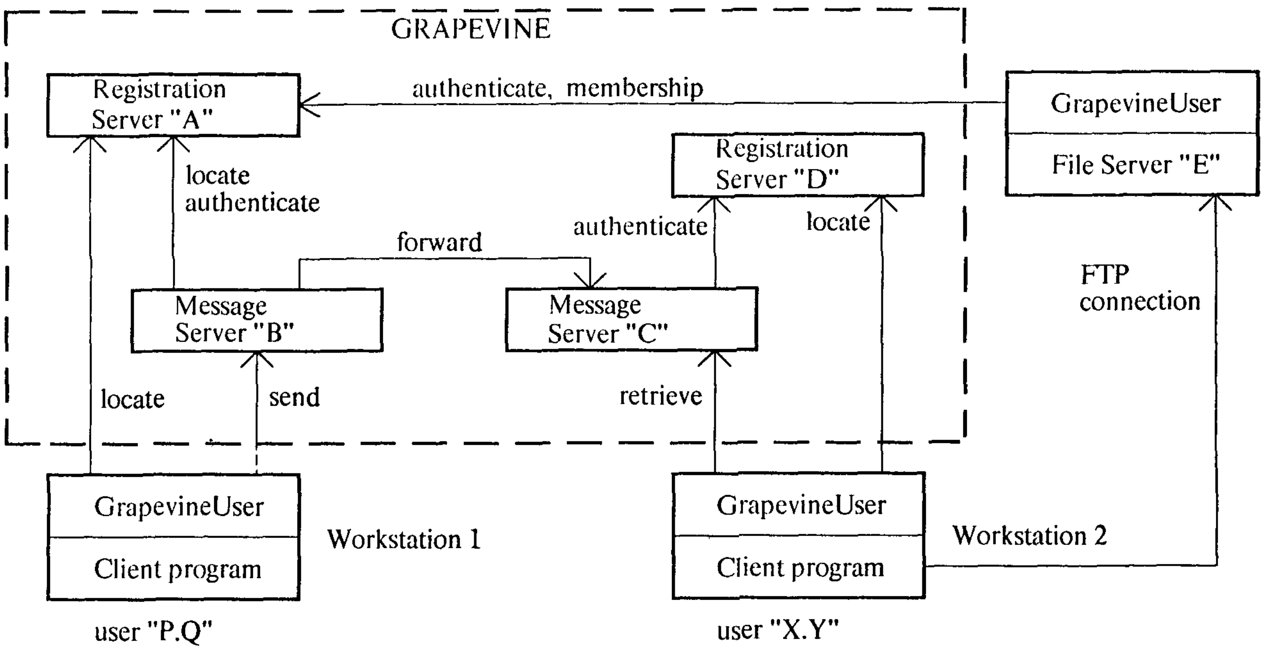
When *X.Y* wishes to use workstation 2 to read his mail, his client program calls the retrieval function of the GrapevineUser package in workstation 2. GrapevineUser uses some registration server (such as *D*) that contains the *Y* registry to locate inbox sites for *X.Y*, then connects to each of these inbox sites to retrieve his messages. Before allowing this retrieval, *C* uses a registration server to authenticate *X.Y*.

If *X.Y* wanted to access a file on the file server *E* through some file transfer program (FTP) the file server might authenticate his identity and check access control lists by communicating with some registration server (such as *A*).

3.6 Choice of Functions

The particular facilities provided by Grapevine were chosen because they are required to support computer mail. The functions were generalized and separated so other applications also could make use of them. If they want to, the designers of other systems are invited to use the Grapevine facilities. Two important benefits occur, however, if Grapevine becomes the *only* mechanism for authentication and for grouping individuals by organization, interest, and function. First, if Grapevine per-

Fig. 2. Distribution of Function.



forms all authentications, then users have the same name and password everywhere, thus simplifying many administrative operations. Second, if Grapevine is used everywhere for grouping, then the same group structure can be used for many different purposes. For example, a single group can be an access control list for several different file servers and also be a distribution list for message delivery. The groups in the registration data base can capture the structure of the user community in one place to be used in many ways.

4. Message Delivery

We now consider the message delivery service in more detail.

4.1 Acceptance

To submit a message for delivery a client must establish an internet connection to a message server; any operational server will do. This resource location step, done by the GrapevineUser package, is described in Sec. 5. Once such a connection is established, the GrapevineUser package simply translates client procedure calls into the corresponding server protocol actions. If that particular message server crashes or otherwise becomes inaccessible during the message submission, then the GrapevineUser package locates another message server (if possible) and allows the client to restart the message submission.

The client next presents the RName and password of the sender, a *returnTo* RName, and a list of recipient RNames. The message server authenticates the sender by using the registration service. If the authentication fails, the server refuses to accept the message for delivery. Each recipient RName is then checked to see if it matches an RName in the registration data base. All invalid recipient names are reported back to the client. In the infrequent case that no registration server for a registry is accessible, all RNames in that registry are presumed for the time being to be valid. The server constructs a *property list* for the message containing the sender name, returnTo name, recipient list, and a *postmark*. The postmark is a unique identification of the message, and consists of the server's clock reading at the time the message was presented for delivery together with the server's internet address. Next, the client machine presents the *message body* to the server. The server puts the property list and message body in reliable storage, indicates that the message is accepted for delivery, and closes the connection. The client may cancel delivery anytime prior to sending the final packet of the message body, for example, after being informed of invalid recipients.

Only the property list is used to direct delivery. A client might obtain the property values by parsing a text message body and require that the parsed text be syntactically separated as a "header," but this happens before Grapevine is involved in the delivery. The property list stays with the message body throughout the delivery process and is available to the receiving client. Grapevine guarantees that the recipient names in the property list were used to control the delivery of the message, and that the sender RName and postmark are accurate.

4.2 Transport and Buffering

Once a message is accepted for delivery, the client may go about its other business. The message server, however, has more to do. It first determines the complete list of individuals that should receive the message by recursively enumerating groups in the property list. It obtains from the registration service each individual's inbox site list. It chooses a destination message server for each on the basis of the inbox site list ordering and its opinion of the present accessibility of the other message servers. The individual names are accumulated in *steering lists*, one for each message server to which the message should be forwarded and one for local recipients. The message server then forwards the message and appropriate steering list to each of the other servers, and places the message in the inboxes for local recipients. Upon receiving a forwarded message from another server, the same algorithm is performed using the individuals in the incoming steering list as the recipients, all of which will have local inboxes unless the registration data base has changed. The message server stores the property list and body just once on its local disk and places references to the disk object in the individual's inboxes. This sharing of messages that appear in more that one local inbox saves a considerable amount of storage in the server.[2]

With this delivery algorithm, messages for an individual tend to accumulate at the server that is first on the inbox site list. Duplicate elimination, required because distribution lists can overlap, is achieved while adding the message into the inboxes by being sure never to add a message if that same message, as identified by its postmark, was the one previously added to that inbox. This duplicate elimination mechanism fails under certain unusual circumstances such as servers crashing or the data base changing during the delivery process, but requires less computation than the alternative of sorting the list of recipient individuals.

In some circumstances delivery must be delayed, for example, all of an individual's inbox sites or a registry's registration servers may be inaccessible. In such cases the message is queued for later delivery.

In some circumstances delivery will be impossible: for example, a recipient RName may be removed from the registration data base between validation and delivery, or a valid distribution list may contain invalid RNames. Occasionally delivery may not occur within a reasonable time, for example, a network link may be down for several days. In such cases the message server mails a copy of the message to an appropriate RName with a text explanation of what the problem was and who did not get the message. The appropriate RName for this error notification may be the returnTo name recorded in the message's property list or the owner of the distribution list that contained the invalid name, as recorded in a group entry in the registration data base. Even this error notification can fail, however, and ultimately such messages end up in a *dead letter* inbox for consideration by a human administrator.

[2] As another measure to conserve disk storage, messages from an inbox not emptied within seven days are copied to a file server and the references in the inbox are changed to point at these archived copies. Archiving is transparent to clients: archived messages are transferred back through the message server when messages from the inbox are retrieved.

4.3 Retrieval

To retrieve new messages for an individual, a client invokes the GrapevineUser package to determine the internet addresses of all inbox sites for the individual, and to poll each site for new messages by sending it a single *inbox check* packet containing the individual's RName. For each positive response, GrapevineUser connects to the message server and presents the individual's name and password. If these are authentic, then the message server permits the client to inspect waiting messages one at a time, obtaining first the property list and then the body. When a client has safely stored the messages, it may send an acknowledgment to the message server. On receipt of this acknowledgment, the server discards all record of the retrieved messages. Closing the retrieval connection without acknowledgment causes the message server to retain these messages. For the benefit of users who want to inspect new messages when away from their personal workstation, the message server also allows the client to specify that some messages from the inbox be retained and some be discarded.

There is no guarantee that messages will be retrieved in the order they were presented for delivery. Since the inbox is read first-in, first-out and messages tend to accumulate in the first inbox of an individual's inbox site list, however, this order is highly likely to be preserved. The postmark allows clients who care to sort their messages into approximate chronological order. The order is approximate because the postmarks are based on the time as perceived by individual message servers, not on any universal time.

4.4 Use of Replication in Message Delivery

Replication is used to achieve a highly available message delivery service. Any message server can accept any message for delivery. Complete replication of this acceptance function is important because the human user of a computer mail client may be severely inconvenienced if he cannot present a message for delivery when he wants to. He would have to put the message somewhere and remember to present it later. Fortunately, complete replication of the acceptance function is cheap and simple to provide. Message transport and buffering, however, are not completely replicated. Once accepted for delivery, the crash of a single message server can delay delivery of a particular message until the server is operational again, by temporarily trapping the message in a forwarding queue or an inbox.[3] Allowing multiple inboxes for an individual replicates the delivery path. Unless all servers containing an individual's inbox sites

[3] The servers are programmed so any crash short of a physical disk catastrophe will not lose information. Writing a single page to the disk is used as the primitive atomic action.

are inaccessible at once, new messages for that individual can get through. We could have replicated messages in several of an individual's inboxes, but the expense and complexity of doing so does not seem to be justified by the extra availability it would provide. If the immediate delivery of a message is important then its failure to arrive is likely to be noticed outside the system; it can be sent again because a delivery path for new messages still exists.

5. The Registration Data Base

The registration data base is used by Grapevine to name registration servers, message servers, and indeed, registries themselves. This recursive use of the registration data base to represent itself results in an implementation that is quite compact.

5.1 Implementing Registries

One registry in the data base is of particular importance, the registry named GV (for Grapevine). The GV registry is replicated in every registration server; all names of the form *.*gv* exist in every registration server. The GV registry controls the distribution and replication of the registration data base, and allows clients to locate appropriate registration servers for particular RNames.

Each registration server is represented as an individual in the GV registry. The connect site for this individual is the internet address where clients of this registration server can connect to it. (The authenticator and inbox site list in the entry are used also, as we will see later.)

The groups of the GV registry are the registries themselves; *reg* is a registry if and only if there exists a group *reg.gv*. The members of this group are the RNames of the registration servers that contain the registry. The GV registry is represented this way too. Since the GV registry is in every registration server, the membership set for *gv.gv* includes the RNames of all registration servers.

5.2 Message Server Names

Each message server is represented as an individual in the MS registry (for message servers). The connect site in this entry is the internet address where clients of this message server can connect to it. (The authenticator and inbox site list in the entry are used also, as we will see later.) It is message server RNames that appear in individuals' inbox site lists.

A group in the MS registry, *Maildrop.ms*, contains as members some subset (usually, but not necessarily, all) of the message server RNames. This group is used to find a message server that will accept a message for delivery.

5.3 Resource Location

The registration data base is used to locate resources. In general, a service is represented as a group in the data base; servers are individuals. The members of the group are the RNames of the servers offering the service; the connect sites of the individuals are the internet addresses for the servers. To contact an instance of the service, a client uses the GrapevineUser package to obtain the membership of the group and then to obtain the connect site of each member. The client then may choose among these addresses, for example, on the basis of closeness and availability.

The GrapevineUser package employs such a resource location strategy to find things in the distributed registration data base. Assume for a moment that there is a way of getting the internet address of some operational registration server, say *Cabernet.gv*. GrapevineUser can find the internet addresses of those registration servers that contain the entry for RName *f.r* by connecting to *Cabernet.gv* and asking it to produce the membership of *r.gv*. GrapevineUser can pick a particular registration server to use by asking *Cabernet.gv* to produce the connect site for each server in *r.gv* and attempting to make a connection until one responds. If *f.r* is a valid name, then any registration server in *r.gv* has the entry for it. At this point GrapevineUser can extract any needed information from the entry of *f.r*, for example, the inbox site list.

Similarly, GrapevineUser can obtain the internet addresses of message servers that are willing to accept messages for delivery by using this resource location mechanism to locate the servers in the group *MailDrop.ms*. Any available server on this list will do.

In practice, these resource location algorithms are streamlined so that although the general algorithms are very flexible, the commonly occurring cases are handled with acceptable efficiency. For example, a client may assume initially that *any* registration server contains the data base entry for a particular name; the registration server will return the requested information or a *name not found* error if this registration server knows the registry, and otherwise will return a *wrong server* error. To obtain a value from the registration data base a client can try any registration server; only in the case of a *wrong server* response does the client need to perform the full resource location algorithm.

We are left with the problem of determining the internet address of some registration server in order to get started. Here it is necessary to depend on some more primitive resource location protocol. The appropriate mechanism depends on what primitive facilities are available in the internet. We use two mechanisms. First, on each local network is a primitive *name lookup server*, which can be contacted by a broadcast protocol. The name lookup server contains an infrequently updated data base that maps character strings to internet addresses. We arrange for the fixed character string *GrapevineRServer* to be entered in this data base and mapped to the internet addresses of some subset of the registration servers in the internet. The GrapevineUser package can get a set of addresses of registration servers using the broadcast name lookup protocol, and send a distinctive packet to each of these addresses. Any accessible registration server will respond to such packets, and the client may then attempt to connect to whichever server responds. Second, we broadcast a distinctive packet on the directly connected local network. Again, any accessible registration server will respond. This second mechanism is used in addition to the first because, when there is a registration server on the local network, the second method gives response faster and allows a client to find a local registration server when the name lookup server is down.

Part II. Grapevine as a Distributed System

6. Updating the Registration Data Base

The choice of methods for managing the distributed registration data base was largely determined by the requirement that Grapevine provide highly available, decentralized administrative functions. Administrative functions are performed by changing the registration data base. Replication of this data base makes high availability of administrative functions possible. An inappropriate choice of the method for ensuring the consistency of copies of the data, however, might limit this potential high availability. In particular, if we demanded that data base updates be atomic across all servers, then most servers would have to be accessible before any update could be started. For Grapevine, the nature of the services dependent on the registration data allows a looser definition of consistency that results in higher availability of the update function. Grapevine guarantees only that the copies of a registration data base entry eventually will have the same new value following an update to one of them. If all servers containing copies are up and can communicate with one another, convergence will occur within a few minutes at most. While an update is converging, clients may detect inconsistency by reading the value of an entry from several servers.

6.1 Representation

The value for each entry in the registration data base is represented mainly as a collection of lists. The membership set of a group is one such list. Each list is represented as two sublists of items, called the *active* sublist and the *deleted* sublist. An item consists of a string and a timestamp. A particular string can appear only once in a list, either in the active or the deleted sublist. A timestamp is a unique identifier whose most significant bits are a time and least significant bits an internet address. The time is that perceived by the server that placed the item in the list; the address is that server's. Because a particular server never includes the same time in two different timestamps, all timestamps from all servers are totally ordered.[4]

Fig. 3. A Group from the Registration Data Base.

Prefix: [1-Apr-81 12:46:45, 3#14], type = group, LaurelImp↑.pa
Remark: (stamp=[22-Aug-80 23:42:14, 3#22]) Laurel Team
Members: Birrell.pa Brotz.pa, Horning.pa, Levin.pa, Schroeder.pa
Stamp-list: [23-Aug-80 17:27:45, 3#22], [23-Aug-80 17:42:35, 3#22], [23-Aug-80 19:04:54, 3#22], [23-Aug-80 19:31:01, 3#22], [23-Aug-80 20:50:23, 3#22]
DelMembers: Butterfield.pa
Stamp-list: [25-Mar-81 14:15:12, 3#14]
Owners: Brotz.pa
Stamp-list: [22-Aug-80 23:43:09, 3#14]
DelOwners: none
Stamp-list: null
Friends: LaurelImp↑.pa
Stamp-list: [1-Apr-81 12:46:45, 3#14]
DelFriends: none
Stamp-list: null

For example, Fig. 3 presents the complete entry for a group named "LaurelImp↑.pa" from the registration data base as it appeared in early April 1981. There are three such lists in this entry: the membership set labeled *members* and two access control lists labeled *owners* and *friends* (see Sec. 6.5 for the semantics of these). There are five current members followed by the corresponding five timestamps, and one deleted member followed by the corresponding timestamp. The owners and friends lists each contain one name and no deletions are recorded from either.

A registration data base entry also contains a version timestamp. This timestamp, which has the same form as an item timestamp, functions as an entry's version number. Whenever anything in an entry changes the version timestamp increases in value, usually to the maximum of the other timestamps in the entry. When interrogating the data base, a client can compare the version timestamp on which it based some cached information with that in the data base. If the cached timestamp matches then the client is saved the expense of obtaining the data base value again and recomputing the cached information. The version timestamp appears in the prefix line in Fig. 3.

6.2 Primitive Operations

Grapevine uses two primitive operations on the lists in a registration data base entry. An *update* operation can add or delete a list item. To add/delete the string *s* to/from a list, any item with the matching string in either of the sublists first is removed. Then a timestamp *t* is produced from the server's internet address and clock. Finally the item (*s*, *t*) is added to the active/deleted sublist. A *merge* operation combines two versions of a complete list to produce a new list with the most recent information from both. Each string that appears in either

[4] The item timestamps in the active sublist are used to imply the preference order for the inbox site list in an individual's entry; older items are preferred. Thus, deleting then adding a site name moves it to the end of the preference ordering.

version will appear precisely once in the result. Each string will be in the active or deleted sublist of the result according to the largest timestamp value associated with that string in either version. That largest timestamp value also provides the timestamp for the string in the result. Keeping the sublists sorted by string value greatly increases the speed with which the merge can be performed. The update and merge operations are atomic in each particular server.

6.3 Propagation

The administrative interface to Grapevine is provided by client software running in an administrator's computer. To make a change to the data of any registry, a client machine uses the resource location facilities of the GrapevineUser package to find and connect to some registration server that knows about that registry. That registration server performs an update operation on the local copy of an entry. Once this update has been completed the client can go about its other business. The server propagates the change to the replicas of the entry in other servers. The means used to propagate the change is Grapevine's delivery service itself, since it gives a guarantee of delivery and provides buffering when other servers are temporarily inaccessible. As described in Sec. 5.1, the members of the group that represent a registry are the registration servers that contain a copy of the data for that registry. Thus, if the change is to an entry in the *reg* registry, the accepting server sends a *change message* to the members, other than itself, of the distribution list *reg.gv*. A change message contains the name of the affected entry and the entire new value for the entry. Registration servers poll their inboxes for new messages every 30 seconds. When a change message is received by a server it uses merge operations to combine the entry from the change message with its own copy.

With this propagation algorithm, the same final state eventually prevails everywhere. When a client makes multiple updates to an entry at the same server, a compatible sequence of entry values will occur everywhere, even if the resulting change messages are processed in different orders by different servers. If two administrators perform conflicting updates to the data base such as adding and removing the same member of a group, initiating the updates at different servers at nearly the same time, it is hard to predict which one of them will prevail; this appears to be acceptable, since the administrators presumably are not communicating with each other outside the system. Also, since copies will be out of step until the change messages are received and acted upon, clients must be prepared to cope with transient inconsistencies. The algorithms used by clients have to be *convergent* in the sense that an acceptable result will eventually ensue even if different and inconsistent versions of the registration data appear at various stages in a computation. The message delivery algorithms have this property. Similar update propagation techniques have been proposed by others who have encountered situations that do not demand instantaneous consistency [10, 13].

If deleted items were never removed from an entry, continued updates would cause the data base to grow. Deleted items are kept in an entry so that out-of-order arrival of change messages involving addition followed by deletion of the same string will not cause the wrong final state. Deleted items also provide a record of recent events for use by human administrators. We declare an upper bound of 14 days upon the clock asynchrony among the registration servers, on message delivery delay, and on administrative hindsight. The Grapevine servers each scan their local data base once a day during inactive periods and purge all deleted items older than the bound.

If a change message gets destroyed because of a software bug or equipment failure, there is a danger that a permanent inconsistency will result. Since a few destroyed messages over the life of the system are inevitable, we must provide some way to resynchronize the data base. At one point we dealt with this problem by detecting during the merge operation whether the local copy of the entry contained information that was missing from the incoming copy. Missing information caused the server to send the result of the merge in a change message to all servers for the registry. While this "anti-entropy" mechanism tended to push the data base back into a consistent state, the effect was too haphazard to be useful; errors were not corrected until the next change to an entry. Our present plan for handling long-term inconsistencies is for each registration server periodically, say once a night, to compare its copy of the data base for a registry with another and to use merges to resolve any inconsistencies that are discovered. The version timestamp in each entry makes this comparison efficient: if two version timestamps are equal then the entries match. Care must be taken that the comparisons span all registration servers for a registry, or else disconnected regions of inconsistency can survive.

6.4 Creating and Deleting Names

The rule that the latest timestamp wins does not deal adequately with the creation of new names. If two administrators connect to two different registration servers at about the same time and try to create a new data base entry with the same name, it is likely that both will succeed. When this data base change propagates, the entry with the latest time timestamp will prevail. The losing administrator may be very surprised, if he ever finds out. Because the later creation could be trapped in a crashed registration server for some time, an administrator could never be sure that his creation had won. For name creation we want the *earlier* creation to prevail. To achieve this effect, we faced the possibility of having to implement one of the known and substantial algorithms for atomic updates to replicated databases [3], which seemed excessive, or of working out a way to make all names unique by appending a hidden timestamp, which seemed complex. We instead fell back on observations about the way in which systems of this nature are used. For each registry there is usually some human-level centralization of name creation, if only to deal with questions of suitability of RNames (not having a junior clerk preempt the RName which everyone would associate with the company president). We consider this centralization enough to solve the problem. Note that there is no requirement that a particular *server* be used for name creation: there is no centralization at the machine level.

Deleting names is straightforward. A deleted entry is marked as such and retained in the data base with a version timestamp. Further updates to a deleted entry are not allowed. Recreation of a deleted entry is not allowed. Sufficiently old deleted entries are removed from the data base by the purging process described in Sec. 6.3.

6.5 Access Controls

An important aspect of system administration is control of who can make which administrative changes. To address this need we associate two access control lists with each group: the *owners list* and the *friends list*. These lists appear in the example entry in Fig. 3. The interpretation of these access lists is the responsibility of the registration server. For ordinary groups the conventions are as follows: membership in the owners list confers permission to add or remove any group member, owner, or friend; membership in the friends list confers permission to add or remove oneself. The names in the owners and friends lists may themselves be the names of groups. Quite separately, clients of the registration server have freedom to use membership in groups for access control purposes about which the registration server itself knows nothing at all. The owners and friends lists on the groups that represent registries are used to control name creation and deletion within registries; these lists also provide the default access controls on groups whose owners list is empty. While we have spent some time adjusting the specific semantics of the Grapevine access controls, we do not present further details here.

6.6 Other Consequences of Changes

The registration servers and message servers are normal clients of one another's services, with no special relationship. Registration servers use message server delivery functions and message servers use the registration service to authenticate clients, locate inboxes, etc. This view, however, is not quite complete. If a change is made to the inbox locations of any individual, notice has to be given to all message servers that are removed, so they can redeliver any messages for that individual buffered in local inboxes. Notice is given by the registration server delivering a message to the message servers in question informing them of the change. Correctness requires that the last registration server that changes its copy of the entry emit the message; we achieve this effect by having each registration server emit such a message as the change is made. A message server receiving an inbox removal message simply redelivers all messages in the affected inbox. Redelivery is sufficient to rebuffer the messages in the proper server. In the system as implemented a simplification is made; inbox removal messages are sent to all inbox sites for the affected individual, not just to removed sites. While this may appear to be wasteful, it is most unusual for any site other than the primary one to have anything to redeliver.

Other registration service clients that use the registration data base to control resource bindings may also desire notification of changes to certain entries. A general notification facility would require allowing a notification list to be associated with any data base entry. Any change to an entry would result in a message being sent to the RNames on its notification list. We have not provided this general facility in the present implementation, but would do so if the system were reimplemented.

7. Finding an Inbox Site

The structure and distribution of the Grapevine registration data base are quite complex, with many indirections. Algorithms for performing actions based on this data base should execute reliably in the face of administrative changes to the registration data base (including those which cause dynamic reconfiguration of the system) and multiple servers that can crash independently. In their full generality such algorithms are expensive to execute. To counter this, we have adopted a technique of using caches and hints to optimize these algorithms. By *cache* we mean a record of the parameters and results of previous calculations. A cache is useful if accessing it is much faster than repeating the calculation and frequently produces the required value. By *hint* we mean a value that is highly likely to be correct and that is faster to check than to recalculate. To illustrate how caches and hints can work, we describe here in some detail how the message server caches hints about individuals' inbox sites.

The key step in the delivery process is mapping the name of an individual receiving a message to the preferred inbox site. The mapping depends upon the current state of the registration data base and the availability of particular message servers. To make this mapping process as efficient as possible, each message server maintains an *inbox site cache* that maps RNames of individuals to a hint for the currently preferred inbox site. Each message server also maintains a *down server list* containing the names of message servers that it believes to be inaccessible at present. A message server is placed on this list when it does not accept connections or fails during a connection. The rules for using the inbox site cache to determine the preferred message server for a recipient *I* are:

1. If an entry for *I* is in the cache and the site indicated for *I* in the cache is not on the down server list, then use that site;
2. Otherwise get the inbox site list for *I* from the registration service; cache and return for use the first site not on the down server list; if the selected site is not first on the list, mark the entry as "secondary."

There has to be a rule for removing message servers from the down server list; this happens when the server shows signs of life by responding to a periodic single packet poll.

When a message server is removed from the down server list, the inbox site cache must be brought up to date. Any entry that is marked as "secondary" and that is not the revived site could be there as a substitute for the revived site; all such entries are removed from the cache. This heuristic removes from the cache a superset of the entries whose preferred inbox site has changed (but not all entries in the cache) and will cause recalculation of the preferred inbox site for those entries the next time they are needed.

We noted earlier that changing an individual's inbox site list may require a message server to redeliver all messages in that individual's inbox, and that this redelivery is triggered by messages from registration servers to the affected message servers. The same changes also can cause site caches to become out-of-date. Part of this problem is solved by having the inbox redelivery messages also trigger appropriate site cache flushing in the servers that had an affected inbox. Unfortunately any message server potentially has a site cache entry made out-of-date by the change. Instead of sending a message to *all* message servers, we correct the remaining obsolete caches by providing feedback from one message server to another when incorrect forwarding occurs as a result of an out-of-date cache. Thus, the site cache really does contain hints.

To summarize the cache flushing and redelivery arrangements, then, registration servers remove servers from an inbox site list and send messages to all servers originally on the list. Each responds by removing any entry for the subject individual from its site cache and redelivering any messages found in that individual's inbox. During this redelivery process, the cache entry will naturally be refreshed. Other message servers with out-of-date caches may continue to forward messages here for the subject individual. Upon receiving any message forwarded from another server, then, the target message server repeats the inbox site mapping for each name in the steering list. If the preferred site is indeed this target message server, then the message is added to the corresponding inbox. If not, then the target site does the following:

1. Forwards the message according to the new mapping result;
2. Sends a cache flush notification for the subject individual back to the server that incorrectly forwarded the message here.

The cache flush notification is a single packet sent unreliably: if it fails to arrive, another one will be provoked in due course. This strategy results in the minimum of cache flush notifications being sent—one to each message server whose cache actually needs attention, sent when the need for attention has become obvious. This mechanism is more economical than the alternative of sending cache flush notifications to all message servers, and even if that were done it would still be necessary to cope with the arrival of messages at old inbox sites.

8. System Configuration

As described in Sec. 5, the configuration of the Grapevine system is controlled by its registration data base. Various entries in the data base define the servers available to Grapevine and the ways in which the data and functions of Grapevine are distributed among them. We now consider procedures for reconfiguring Grapevine.

8.1 Adding and Deleting Registry Replicas

The set of registration servers that contain some registry is defined by the membership set for the corresponding group in the GV registry. When a change occurs to this membership set, the affected server(s) need to acquire or discard a copy of the registry data. To discover such changes, each registration server simply monitors all change messages for groups in the GV registry, watching for additions or deletions of its own name. A registration server responds to being deleted by discarding the local replica of the registry. With the present implementation, a registration server ignores being added to a registry site list. Responding to a registry addition in the obvious way—by connecting to another registration server for the registry and retrieving the registry data—is not sufficient. Synchronization problems arise that can lead to the failure to send change messages to the added server. Solving these problems may require the use of global locks, but we would prefer a solution more compatible with the looser synchronization philosophy of Grapevine. For the present obtaining a registry replica is triggered manually, after waiting for the updates to the GV registry to propagate and after ensuring that other such reconfigurations are not in progress.

8.2 Creating Servers

Installing a new Grapevine computer requires creating a new registration server and a new message server. To create the new registration server named, say, *Zinfandel.gv*, a system administrator first creates that individual (with password) in the registration data base, and gives it a connect site that is the internet address of the new computer. Next, *Zinfandel.gv* is added to the membership set of all registries that are to be recorded in this new registration server. To create the new message server named, say, *Zinfandel.ms*, the administrator creates that individual with the same connect site, then adds *Zinfandel.ms* to *MailDrop.ms.* Both servers are assigned inbox sites.

Once the data base changes have been made, the registration and message servers are started on the new computer. The first task for each is to determine its own name and password so that it may authenticate itself to the other Grapevine servers. A server obtains its name by noting its own internet address, which is always available to a machine, then consulting the data base in a different registration server to determine which server is specified to be at that address: the registration server looks for a name in the group *gv.gv*, the message server looks for a name in the group *MailDrop.ms.* Having found its name, the server asks a human operator to type its password; the operator being able to do this correctly is the fundamental source of the server's authority. The server verifies its password by the authentication protocol, again using a registration server that is already in operation, and then records its name and password on its own disk. The new registration server then consults some other registration server to obtain the contents of the GV registry in order to determine which groups in the GV registry contain its name: these specify which registries the new server should contain. It then contacts appropriate other servers to obtain copies of the data base for these registries. Because the new server can authenticate itself as an individual in the GV registry, other registration servers are willing to give it entire data base entries, including individuals' passwords.

Obtaining the registry replicas for the new registration server suffers from the same synchronization problems as adding a registry replica to an existing server. We solve them the same way, by waiting for the administrative updates to the GV registry to propagate before starting the new computer and avoiding other simultaneous reconfigurations.

8.3 Stopping and Restarting Servers

Stopping a server is very easy. Grapevine computers can be stopped without disturbing any disk write in progress. The message and registration servers are programmed so that, when interrupted between disk page writes, they can be restarted without losing any permanent information. While a message or registration server is not running, messages for it accumulate in its inboxes in message servers elsewhere, to be read after it restarts.

Whenever a message and registration server restart, each verifies its name and password by consulting other servers, and verifies that its internet address corresponds to the connect site recorded for it in the data base; if necessarry it changes the connect site recorded in the data base. Updating the connect site allows a server to be moved to a new machine just by moving the contents of the disk. After restarting, a registration server acts on all accumulated data base change messages before declaring itself open for business.

Using the internet, it is possible, subject to suitable access controls, to load a new software version into a remote running Grapevine computer, stop it, and restart it with the new version.

8.4 Other Reconfigurations

One form of reconfiguration of the system requires great care: changing the location of inbox sites for a registration server. Unless special precautions are taken, the registration server may never encounter the change message telling it about a new inbox site, because that message is waiting for it at the new site. A similar problem arises when we change the internet address of a message server that contains a registration server's inbox. Restrictions on where such data base changes can be initiated appear to be sufficient to solve these problems, but we have not automated them. Although this resolution of this problem is somewhat inelegant, the problem is not common enough to justify special mechanisms.

Part III. Conclusions

9. Present State

The Grapevine system was first made available to a limited number of clients during 1980. At present (Fall 1981) it is responsible for most of the mail traffic and distribution lists on the Xerox research internet. There are five dedicated Grapevine computers, each containing a registration server and a message server. The computers are physically distributed among northern and southern California and New York. The registration data base contains about 1500 individuals and 500 groups, divided mainly into four major registries; there are two other registries used by nonmail clients of the registration service, plus the GV and MS registries. The total message traffic amounts to some 2500 messages each working day, with an average of 4 recipients each; the messages average about 500 characters, and are almost exclusively text.

The registration data base also is used for authentication and configuration of various file servers, for authentication and access control in connection with maintenance of the basic software and data bases that support our internet gateways, and for resource location associated with remote procedure call binding. The registration data base is administered almost exclusively by nontechnical staff. There are at least three separate computer mail interface programs in use for human-readable mail. Most mail system users add and delete themselves from various distribution lists, removing this tiresome job from administrative staff.

The Grapevine registration and message servers are programmed in Mesa [7]. They contain some 33,000 lines of custom written code, together with standard packages for runtime support and PUP-level communications. The Grapevine computers are Altos [12] with 128K bytes of main memory and 5M bytes of disk storage. A running Grapevine computer has between 40 and 70 Mesa processes [4], and can handle 12 simultaneous connections. The peak load of messages handled by a single message server so far exceeds 150 per hour and 1000 messages per day. One server handled 30,000 messages while running for 1000 hours. The maximum number of primary inboxes that have been assigned to a server is 380.

10. Discussion

The fundamental design decision to use a distributed data base as the basis for Grapevine's message delivery services has worked out well. The distributed data base allowed us to meet the design goals specified in Sec. 2, and has not generated operational difficulties. The distributed update algorithms that trade atomic update for increased availability have had the desired effect. The temporary inconsistencies do not bother the users or administrators and the ability to continue data base changes while the internet is partitioned by failed long-distance links is exercised enough to be appreciated.

In retrospect, our particular implementation of the data base for Grapevine was too inflexible. As the use of the system grew, the need for various extensions to the values recorded in individual and group entries has become apparent. Reformatting the existing distributed data base to include space for the new values is difficult operationally. In a new implementation we would consider providing facilities for dynamic extension of the value set in each entry. With value set extension, however, we would keep the present update algorithm and its loose consistency guarantees. These guarantees are sufficient for Grapevine's functional domain, and their simplicity and efficiency are compelling. There is a requirement in a message system for some data base which allows more flexible descriptions of recipients or distribution lists to be mapped onto message system RNames (such as the white or yellow page services of the telephone system), but in our view that service falls outside of Grapevine's domain. A system which provides more flexibility in this direction is described in [2].

Providing all naming semantics by indirection through the registration data base has been very powerful. It has allowed us to separate the concept of *naming* a recipient from that of *addressing* the recipient. For example, the fact that a recipient is named *Birrell.pa* says nothing about where his messages should be sent. This is in contrast to many previous message systems. Indirections also provide us with flexibility in configuring the system.

One feature which recurs in descriptions of Grapevine is the concept of a "group" as a generalization of a distribution list. Our experience with use of the system confirms the utility of use of the single "group" mechanism for distribution lists, access control lists, services, and administrative purposes.

Clients other than computer mail interfaces are beginning to use Grapevine's naming, authentication, and resource location facilities. Their experience suggests that these are an important set of primitives to provide in an internet for constructing other distributed applications. Message transport as a communication protocol for data other than textual messages is a useful addition to our set of communication protocols. The firm separation between Grapevine and its clients was a good decision; it allows us to serve a wide variety of clients and to give useful guarantees to our clients, even if the clients operate in different languages and in different computing environments.

At several points in Grapevine, we have defined and implemented mechanisms of substantial versatility. As a consequence, the algorithms to implement these mechanisms in their full generality are expensive. The techniques of caches and hints are powerful tools that allow us to regain acceptable efficiency without sacrificing "correct" structure. The technique of adding caches and hints to a general mechanism is preferable to the alternative style of using special case short cut mechanisms whose existence complicates algorithmic invariants.

Grapevine was built partly to demonstrate the assertion that a properly designed replicated system can provide a very robust service. The chance of all replicas being unavailable at the same time seems low. Our experience suggests that unavailability due to hardware failure follows this pattern. No more than one Grapevine computer at a time has ever been down because of a hardware problem. On the other hand, some software bugs do not exhibit this independence. Generally all servers are running the same software version. If a client's action provokes a bug that causes a particular server to fail, then in taking advantage of the service replication that client may cause many servers to fail. A client once provoked a protocol bug when attempting to present a message for delivery. By systematically trying again at each server in *MailDrop.ms*, that client soon crashed all the Grapevine computers. Another widespread failure occurred as a result of a malformed registration data base update propagating to all servers for a particular registry. We conclude that it is hard to design a replicated system that is immune from such coordinated software unreliability.

Our experience with Grapevine has reinforced our belief in the value of producing "real" implementations of systems to test ideas. At several points in the implementation, reality forced us to rethink initial design proposals: for example, the arrangements to ensure long-term consistency of the data base in the presence of lost messages. There is no alternative to a substantial user community when investigating how the design performs under heavy load and incremental expansion.

Acknowledgments. Many people have contributed to the success of the Grapevine project. Bob Taylor and Bob Metcalfe recognized early the need for work on computer mail systems and encouraged us to develop Grapevine. Ben Wegbreit participated in the initial system design effort. Many colleagues have helped the project in various ways: Dave Boggs, Doug Brotz, Jeremy Dion, Jim Horning, Robert Kierr, and Ed Taft deserve special mention. Jerry Saltzer and several anonymous referees have made valuable commentaries on earlier drafts of the paper.

Received 9/81; revised 11/81; accepted 12/81

References

1. Boggs, D.R., Shoch, J.F., Taft, E.A., and Metcalfe, R.M. PUP: An internetwork architecture. *IEEE Trans. on Communications 28*, 4 (April 1980), 612–634.
2. Dawes, N., Harris, S., Magoon, M., Maveety, S., and Petty, D. The design and service impact of COCOS—An electronic office system. In *Computer Message Systems.* R.P. Uhlig (Ed.) North-Holland, New York, 1981, pp 373–384.
3. Gifford, D.K. Weighted voting for replicated data. In *Proc. 7th Symposium on Operating Systems Principles.* (Dec. 1979), ACM Order No. 534 790, pp 150–162.
4. Lampson, B.W., and Redell, D.D. Experience with processes and monitors in Mesa. *Comm. ACM 23*, 2 (Feb. 1980), 105–117.
5. Levin, R., and Schroeder, M.D. Transport of electronic messages through a network. *TeleInformatics 79*, North Holland, 1979, pp. 29–33; also available as Xerox Palo Alto Research Center Technical Report CSL-79-4.
6. Metcalfe, R.M., and Boggs, D.R. Ethernet: Distributed packet switching for local computer networks. *Comm. ACM 19*, 7 (July 1976), 395–404.
7. Mitchell, J.G., Maybury, W., and Sweet, R. Mesa language manual (Version 5.0) Technical Report CSL-79-3, Xerox Palo Alto Research Center, 1979.
8. National Bureau of Standards, Data encryption standard. *Federal Information Processing Standards 46*, Jan. 1977.
9. Needham, R.M., and Schroeder, M.D. Using encryption for authentication in large networks of computers. *Comm. ACM 21*, 12 (Dec. 1978), 993–999.
10. Rothnie, J.B., Goodman, N., and Bernstein, P.A. The redundant update methodology of SDD-1: A system for distributed databases (The fully redundant case). Computer Corporation of America, June 1977.
11. Shoch, J.F. Internetwork naming, addressing and routing. In *Proc. 17th IEEE Computer Society International Conference,* Sept. 1978, IEEE Cat. No. 78 CH 1388-8C, pp 72–79.
12. Thacker, C.P., McCreight, E.M., Lampson, B.W., Sproull, R.F., and Boggs, D.R. Alto: A personal computer. In D.P. Siewiorek, C.G. Bell, and A. Newell, *Computer Structures: Principles and Examples.* (2nd Ed.) McGraw-Hill, New York 1981.
13. Thomas, R.H. A solution to the update problem for multiple copy data base which used distributed control. Bolt, Beranek and Newman Technical Report #3340, July 1976.

Operating Systems

Anita K. Jones
Editor

Cryptographic Sealing for Information Secrecy and Authentication

David K. Gifford
Stanford University and
Xerox Palo Alto Research Center

A new protection mechanism is described that provides general primitives for protection and authentication. The mechanism is based on the idea of sealing an object with a key. Sealed objects are self-authenticating, and in the absence of an appropriate set of keys, only provide information about the size of their contents. New keys can be freely created at any time, and keys can also be derived from existing keys with operators that include *Key-And* and *Key-Or*. This flexibility allows the protection mechanism to implement common protection mechanisms such as capabilities, access control lists, and information flow control. The mechanism is enforced with a synthesis of conventional cryptography, public-key cryptography, and a threshold scheme.

CR Categories and Subject Descriptors: D.4.6 **[Operating Systems]**: Security and Protection—access controls, authentication, cryptographic controls, information flow controls; E.3 **[Data]**: Data Encryption—public-key cryptosystems

General Term: Security

Additional Key Words and Phrases: cryptographic sealing, seal, unseal, key, secrecy, conventional cryptosystems

1. Introduction

In order to trust computers with sensitive information it is necessary to take steps to ensure that information stored in them will only be disclosed to authorized users. Personal information is now routinely stored in computers, and thus computer security is a necessary precondition for privacy. In addition, it is widely recognized that information can be sold as a product. In this case, it is important to be able to restrict information access to paying customers.

This research was supported in part by the Fannie and John Hertz Foundation and by the Xerox Corporation.

Author's Present Address: David K. Gifford, Xerox Palo Alto Research Center, 3333 Coyote Hill Road, Palo Alto, CA 94304.