Experience with Grapevine: The Growth of a Distributed System

MICHAEL D. SCHROEDER, ANDREW D. BIRRELL,
and ROGER M. NEEDHAM
Xerox Palo Alto Research Center

Grapevine is a distributed, replicated system that provides message delivery, naming, authentication, resource location, and access control services in an internet of computers. The system, described in a previous paper [1], was designed and implemented several years ago. We now have had operational experience with the system under substantial load. In this paper we report on what we have learned from using Grapevine.

Categories and Subject Descriptors: C.2.4 [**Computer Communication Networks**]: Distributed Systems—*distributed applications, distributed databases*; C.4 [**Computer Systems Organization**]: Performance of Systems—*reliability, availability, serviceability*; 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

Additional Key Words and Phrases: Grapevine

1. INTRODUCTION

Grapevine is a distributed, replicated system that provides message delivery, naming, authentication, resource location, and access control services in an internet of computers. The system, described in a previous paper [1], was designed and implemented several years ago. Operational experience with the system under substantial load has proved the original design sound in most aspects, although there have been some surprises. In this paper we report on what we have learned from using Grapevine. Our experience may offer some help to designers of new systems.

The utility of computer mail systems is widely recognized. Most computing environments provide a mail service. Local systems with similar properties become interconnected through internets like Arpanet [13], UUCP [14] and CSNET [6]. Recent work by IFIPS WG6.5 [8] addresses standards for interconnecting dissimilar systems. Some work has been reported on homogeneous

Authors' addresses: M.D. Schroeder and A.D. Birrell: Xerox Corporation, Palo Alto Research Center, 3333 Coyote Hill Road, Palo Alto, CA 94304; R.M. Needham: Computer Laboratory, Cambridge University, Corn Exchange Street, Cambridge CB2 3QG, UK.
Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.
© 1984 ACM 0734-2071/84/0200-0003 $00.75

systems that provide a richer set of functions, such as COCOS [9]. No reported message system provides a combination of services or exhibit a homogeneous distributed implementation similar to Grapevine's.

After a brief structural overview of Grapevine in Sections 2 and 3 we present observations in Sections 4–9 based on operational experience with the system. The observations are divided into six general topics that are relevant to the design of most distributed systems. We conclude in Section 10 with a brief description of the changes made to adapt the prototype Grapevine system for use as the Xerox 8000 NS product message system and clearinghouse.

2. STRUCTURAL OVERVIEW

A brief description of the system structure will provide a framework for the discussion that follows. (The reader is encouraged to refer to [1] for a detailed discussion of the objectives and structure of Grapevine.) Grapevine operates in the Xerox research internet. This computer communications network is a collection of Ethernet local networks [10, 11], gateways, and long distance links that interconnects personal workstation computers and shared server computers. Grapevine is implemented as a Mesa [12] program that runs on a set of dedicated server computers. The usual server computer is an Alto [17] with 128 Kbytes of main memory and 5 Mbytes of disk storage. The processor can perform a simple Mesa procedure call in about 30 microseconds. The disks were known from the start of the project to be too small, but larger replacements were not easily available at the time.

Figure 1 shows the location, as of the summer of 1983, of the Grapevine servers in a stylized topology of the internet. Client programs of Grapevine run on various workstation and server computers attached to the internet. Communication among Grapevine servers, and among clients and servers, is built on internet protocols that 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 [3, 16].

The services provided by Grapevine are divided into the message service and the registration service. Each Grapevine server is a combination of a message server program and a registration server program. All message servers coöperate to provide Grapevine's message service, and all registration servers cooperate to provide Grapevine's registration service. Each service is a client of the other.

The message service accepts messages prepared by clients for delivery to individual recipients and distribution lists. Messages are buffered in inboxes on message servers until the recipient requests them. Messages need not be human readable text; Grapevine never looks at the content of messages. Any message server can accept any message for delivery, thus providing a replicated submission service. Each message server will accept retrieval requests for inboxes on that server. A user of the computer mail system has inboxes on at least two message servers, thus replicating the delivery path.

The registration service provides naming, authentication, access control and resource location functions to clients; it is based on a registration database that maps names to information about the users, machines, services, distribution lists,

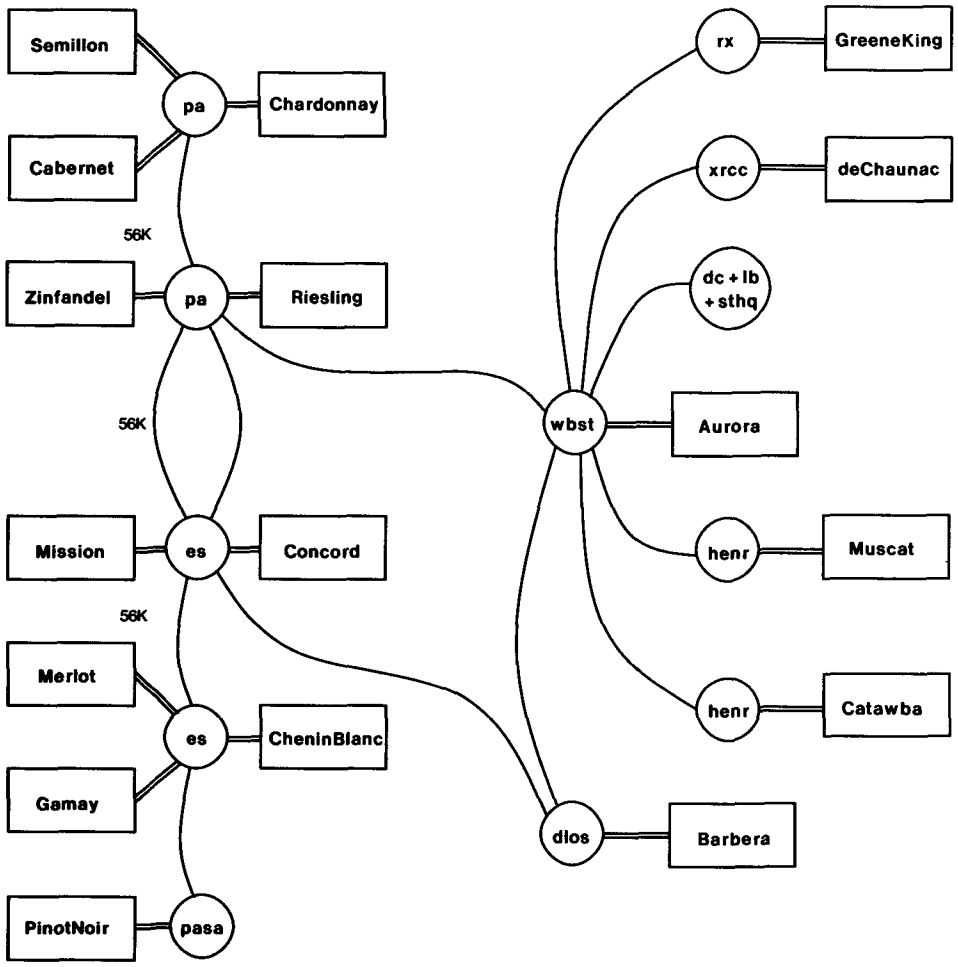

Fig. 1. The location of Grapevine servers in the internet. This diagram shows the placement of the 17 Grapevine servers, represented by rectangles, in a stylized internet topology. Each circle represents a collection of Ethernets serving a local area; a circle contains the names of the registries for most mail system users in that area. The long distance links are shown as curved lines labelled with their data rates. Unlabelled lines are 9.6K bps.

and access control lists that those names signify. The registration data is used to control the message service; it is accessed directly for the resource location, access control and authentication services, and is used to configure Grapevine itself.

There are two types of entries in the registration database: group entries and individual entries. We call the name of an entry in the registration database an RName. A group entry contains a set of RNames of groups and individuals, along with other information. Groups are a way of naming collections of RNames. The groups form a naming network with no structural constraints. They are used primarily as distribution lists, but are also used to represent access control lists

and collections of similar resources. An individual entry contains a password, an ordered list of inbox sites, and a connect site, as well as additional information. The inbox site list indicates, in order of preference, the message servers where the individual's messages may be buffered. The connect site is an internet address for making a connection to the individual. Individuals are used to represent human users and servers, in particular the servers that implement Grapevine. For mail users, the inbox site list normally contains a primary and a secondary inbox site.

We use a partitioned naming scheme for RNames. The partitions serve as a basis for dividing the administrative responsibility and for distributing the database among the registration servers. 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 correspond to locations, organizations, and applications that exist within the user community.

The registration database is distributed and replicated. Distribution is at the level of an entire registry, that is, each registration server contains either entries for all RNames in a registry or no entries for that registry. No registration server contains all registries. Also, each registry is replicated in several different registration servers. A registration server provides access to the information about the names in the registries that it contains. Any server that contains a replica of a registry can accept a change to that registry from a client. That server takes the responsibility for propagating the change to the other relevant servers. The updates are propagated by Grapevine messages.

Associated with each registry is a group of human administrators, called registrars, who are responsible for creating, updating, and deleting individuals and groups in the registry. The registrars for different registries operate independently of one another, except in unusual circumstances. An important part of the administrative burden for a registry is assumed by the users themselves, each of whom can add or remove his own RName as a member of certain groups. A user can also become the owner of a group; the owner can add and remove arbitrary member names. All these functions are carried out by making changes in the registry data through administrative interface programs that are clients of Grapevine.

A client program of Grapevine generally obtains services through code, called the GrapevineUser package, that runs in the client's computer. We have provided versions of this package for several language and operating environments. 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 primary clients of Grapevine are various mail system interface programs, of which Laurel [4] is the most widely used. Some other clients of Grapevine

implement file server authentication and access controls, remote-procedure-call binding [2][1] and process controls for an integrated circuit facility [5].

3. STATE OF THE SYSTEM

As of the fall of 1981 there were 5 Grapevine servers [1]. The registration database contained about 1500 individuals and 500 groups. The total number of messages presented for delivery was about 2500 each working day. Since then the system has grown considerably. By the summer of 1983 there were 17 Grapevine servers. The registration data base contained about 4400 individuals and 1500 groups. Over 8500 messages were presented for delivery in a typical work day, leading to over 35,000 message receptions. Table I and Figure 1 give more details on the size of the system at that time. The growth and heavy use of the system have enabled us to see how well the design has lived up to expectations in actual operation.

4. EFFECTS OF SCALE

An objective of Grapevine design was the ability to increase system capacity over a large range by adding more servers of fixed power, rather than by using more powerful servers. Ideally one should be able to 1) measure the total load placed on the system by a particular community, 2) know how much load a single server can carry, and then 3) compute the number of servers required by using simple division. To meet the ideal the cost of any computation performed by a single server should have a fixed upper bound, and not grow as a function of the total system load or the number of servers used to meet the load. This way the power of an individual server will not limit the growth of the system. The task of the system designer is to meet the ideal well enough to reach a specific maximum system size and total load.

Grapevine design reflects careful attention to this principle of distributed system design. Our specification was a maximum system size of 30 Grapevine servers and a total load generated by 10,000 users. With the exceptions discussed below, the original design appears to meet this specification. Different approaches would be necessary in some cases to expand the system much beyond that specification.

The only direct manifestation of the total system size in an individual server is the space required to store the configuration information that is known by every registration server and the time required to execute resource location algorithms. Every registration server knows the names and addresses of all message and registration servers. Every registration server also knows the names of all registries, and knows for each the names of all servers that contain registry replicas. The amount of disk space required to store this configuration information (about 15 Kbytes for the present system or less than one percent of the disk space available on a server) will not prevent the system from meeting the specification. To go beyond the specification might require the added complexity

[1] Reference 2 appears in this issue on pages 39–59.

Table I. Distribution of Inboxes and Registry Replicas[a]

SERVERS \ REGISTRIES	dc+lb+ sthq		dlos		es		henr		pa		pasa		rx		wbst		xrcc	
	p	s	p	s	p	s	p	s	p	s	p	s	p	s	p	s	p	s
Aurora	137	0	0	286	.	.	.	.	.	.	11	0	4	45	354	17	1	27
Barbera	.	.	290	0	.	.	.	.	0	0	.	.	.	.	0	353	0	* 1
Cabernet	.	.	.	.	.	.	.	.	219	263	0	*11	.	.	0	* 1	.	.
Catawba	.	.	.	.	.	.	44	250	.	.	.	.	.	.	17	0	.	.
Chardonnay	.	.	.	.	.	.	.	.	67	* 2	.	.	.	.	.	.	.	.
CheninBlanc	.	.	.	.	1	792	.	.	3	0	.	.	.	.	.	.	.	.
Concord	.	.	0	0	0	161	0	0	.	.	.	.	.	.	0	0	0	0
deChaunac	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	27	0
Gamay	.	.	.	.	294	0	.	.	.	.	0	114	.	.	.	.	.	.
GreeneKing	.	.	.	.	.	.	.	.	.	.	.	.	45	4	.	.	.	.
Merlot	.	.	.	.	498	1	.	.	0	* 4	.	.	.	.	.	.	.	.
Mission	.	.	.	.	161	0	.	.	.	.	.	.	.	.	.	.	.	.
Muscat	0	137	.	.	0	0	250	44	.	.	.	.	.	.	.	.	.	.
PinotNoir	.	.	.	.	.	.	.	.	.	.	114	0	0	0	.	.	.	.
Riesling	.	.	.	.	.	.	.	.	129	35	.	.	.	.	.	.	.	.
Semillon	0	0	.	.	.	.	.	.	103	102	.	.	.	.	.	.	.	.
Zinfandel	.	.	.	.	0	0	.	.	128	243	.	.	.	.	.	.	.	.

[a] Only the major registries for computer mail users are shown. The "p" column for each registry is the number of primary inboxes at various servers for registry individuals. The "s" column is the number of secondary inboxes. Entries containing numbers but no asterisk (*) indicate the sites of replicas of registry data. Compare the numbers in this table with Figure 1 to see the impact of network topology and the geographic location of users on the configuration of the system.

of keeping only partial configuration information in each server, that is, adding another layer to the naming hierarchy. The present resource location algorithm chooses the nearest up server from among those providing the resource in question. The time required to make this choice from up to 30 servers is acceptable if the choice is not made too often. At some point beyond that size, however, resource location by this method will become too expensive, and more complex schemes that limit choice to a precomputed, fixed-size subset will be required.

Division of the registration database into registries is the primary method for preventing scaling problems. A growing user community is met with more registries, not larger ones (see Section 5 for further discussion of this point). The number of replicas of a registry is independent of the number of servers and of

the number of users in the whole system. Thus, the amount of registration data on one registration server will not grow with the size of the system.

One aspect of Grapevine design that has not scaled properly involves distribution lists. As the system grew we were surprised to discover that the size of certain distribution lists increased as a fraction of the total user community. These lists have nothing to do with organizational structure, geographic location, job responsibilities, or projects. Rather, they reflect general interests held by a fixed fraction of the entire user community. An example is a list called Tax↑.pa used to exchange information about federal taxes. This list contains about a sixth of the entire mail user community and has over 500 members. The message delivery algorithm used by Grapevine has the accepting server determine the best inbox site for each recipient. As the user community grew, we noticed that under worst case conditions messages to such large, general interest lists began to take more than 10 minutes to process, delaying delivery of other messages. The time required to process messages to such lists will continue to increase with the size of the user community, thus placing a limit on the size of the system. Unless changes are made in the delivery algorithm, the system is already close to that limit. The frequency with which members are added or deleted for such general interest lists also tends to grow with the size of the list, as does the computational cost of adding or deleting a member.

We now believe that a more satisfactory way to handle system-wide interest lists is by adding an indirection in their interpretation. Suppose we had a registry All that we used for such lists, and that our tax distribution list was called Tax↑.All. The members would not be individuals but sublists associated with the various registries, for example Tax↑.pa, Tax↑.es, etc. We would arrange that each individual be on the sublist associated with his own registry, and ensure that this remained so by making it an automatic part of adding a name to Tax↑.All. The size of a sublist would be limited by the number of individuals in a registry. The delivery algorithm would be changed so that, when a message has recipient distribution lists from the All registry, the accepting message server would expand such lists only one level. After this first level expansion had occurred, the recipient groups and individuals would be sorted into steering lists by registry. Each steering list would be forwarded with a copy of the message to some Grapevine server whose registration server would know that registry. These message servers would finish the delivery to steering list recipients, including expanding distribution lists like Tax↑.pa, in the normal way. Thus, the delivery computation would be spread among a number of Grapevine servers that would be proportional to the number of recipients. The load on an individual message server would be tied to the size of a registry, which would not need to grow as the size of the user community increased. Grapevine handles large distribution lists if their semantics are identical to those of small distribution lists. Bulletin boards, which have rather different semantics, may offer another solution to this problem.

The proposed change to the structure of general interest distribution lists, and to the delivery algorithm, would probably allow Grapevine to expand to its specified 10,000 user size. For a larger user community, however, a related issue would begin to cause problems. As the community grew the number of messages

sent to such a list would also increase. At some point the number of messages arriving for a user would start to overwhelm both the user and the system. We do not know if this phenomenon has a natural sociological limit. In the world of paper, the problem is controlled by the distribution of information through periodicals that have editors to filter the input. An analogous filtering mechanism will be required in the world of electronic message systems before they can become universal.

A quite different scale effect depends on the size of the underlying internet. As the size of the user community increases, so does the size of the internet. The path from one server to another through the internet is through a series of links and gateways. As the number of gateways in the path increases, the probability of being able to establish a direct connection decreases. At some point, Grapevine's practice of delivering a message by direct connection from the accepting message server to the preferred inbox site becomes unsuitable. If such delivery were along a path with multiple unreliable links, and if there were other Grapevine servers along the way, then multistep forwarding could usually deliver the message faster. The unreliable links would be negotiated one at a time. With a longest path length of 11 gateways, we occasionally receive complaints from users about the message delivery delays that result.

5. CONFIGURATION DECISIONS

Running Grapevine requires the ability to make configuration decisions about how many servers to have, where to place them, and how to distribute registry replicas and inboxes among them. A number of factors influence these configuration decisions and it has been hard to develop guidelines for making such decisions (Table I and Figure 1 shows the current configuration).

A message is shared among all recipient inboxes on the same server and is stored only once. The effect of sharing is enhanced if the assignment of inboxes to message servers corresponds to the patterns of communication. A requirement for taking maximum advantage of message sharing is that the servers be large enough to match, in some sense, the unit size of the organization. An organization with very large groups of people who depend on messages to communicate among themselves would need to be provided with larger servers. In practice, the organizational structure imposed by management and social considerations provides a natural limit to the capacity needed in a single server. With Grapevine's present configuration and load the average message is shared among 4.7 inboxes: certain messages are shared among as many as 300 inboxes on a server and more than half the messages are not shared at all.

Deciding when to add new servers to the system is quite straightforward. Within a local area a server will be added when the load on the existing server gets too large. The system has grown from one to five servers in Palo Alto (Cabernet, Chardonnay, Riesling, Semillon, and Zinfandel in Figure 1) and in El Segundo (CheninBlanc, Concord, Gamay, Merlot, and Mission) as the use of the message system has expanded. When a population of users develops that is separated from the nearest server by one or more slow or unreliable network links, it usually is appropriate to add a local server. The servers in England (GreeneKing) and Toronto (deChaunac) are fairly lightly loaded, but are useful

because of the unreliable nature of the links between those areas and the rest of the internet.

In cases where there is a heavy enough local load to require multiple local servers, a configuration we have considered but not tried would be to use a pair of machines on the same local net, one of which would be just a message server and the other just a registration server. This division of function might simulate a single, larger server. Most of the disk space of the message server would be available for inboxes and the entire connection limit would be available to message reading and sending. Maximum advantage of message sharing would be realized; the registration server would realize similar benefits. In both servers, the working set of data might better fit into the disk page cache in memory, allowing more efficient operation. Of course, other servers would have to provide secondary inboxes and registry replicas as backups for the dual server.

Primary inboxes are usually assigned to the server that is closest to the workstation from which a user normally reads new messages. In local areas with more than one server, it is important to divide the assignment of primary inboxes evenly among the servers. The division should not be arbitrary or the benefit of sharing message bodies will be diminished. Our experience is that division along organizational lines reflects communication patterns and is easy to administer. It has been harder to develop a rationale for assignment of secondary inbox sites. The obvious assignment to the next nearest server can overload the secondary server when the primary one fails (see Section 9, on reliability, for further discussion of this point). Splitting secondary assignments to prevent this overloading tendency can impact the effectiveness of message sharing. If certain network links are known to be less reliable than others, it can be important to get secondary inboxes on the "other side" of the unreliable link, to prevent messages from using up resources by circulating through the pending queues on servers when the link is down. This last consideration can be particularly important for the inboxes of registration servers, which tend to have a high volume of database update messages. For registration and message servers we have adopted a policy of having the primary inbox on the server itself, the secondary inbox nearby, and a tertiary inbox at the other end of the internet. The secondary and tertiary inboxes are placed on the more lightly loaded servers.

Several factors affect the definition of registries and the distribution of registry replicas. We usually have defined registries to correspond to significant geographical areas (e.g., es for El Segundo, pa for Palo Alto, or dlos for Dallas), instead of to organizations or small geographical areas (e.g., building 35). We felt that this assignment was the most stable and the easiest to remember. Large corporations tend to restructure their organization charts and move people from one building to another reasonably frequently, and we did not want to change account names every time that happened. We now believe that the decision to use only geographical registries was mistaken. It would improve things to add organizational registries for certain distribution lists. A distribution list then could be placed in a geographic or organizational registry according to its purpose. Individuals would remain in geographical registries. For example, an organization using Grapevine currently has its people evenly split between the es and pa registries, corresponding to its two locations. The organizational distribution

lists are arbitrarily put in one of these registries, es. As a result, the entire es registry needs to be available in Palo Alto as well as El Segundo. If the organizational distribution lists were put in their own registry, then only this smaller registry would need to be available in Palo Alto.

In choosing where to put registry replicas, there are five considerations to be made: the first is making a registry's data easily accessible to the message servers containing inboxes for that registry. When a user retrieves buffered messages, authentication and inbox site location may occur interactively, so fast response is important. The second consideration is getting the contents of distribution lists close to the message servers that accept messages addressed to them. Distribution list expansion occurs after the interactive connection with the client is closed, so fast response is not an issue here. Distribution list contents are not cached by a message server, however, so expansion of a large, remote list can take a long time, delaying the processing of other messages on the input queue. The third consideration is keeping registry data available to all clients even when certain internet links breakdown. Experience indicates that some links are more vulnerable than others. We want replicas of a registry on both sides of an unreliable link. The fourth consideration is having enough replicas to make the probability of losing a registry's data because of disk catastrophes almost zero. The second, third, and fourth considerations lead to having three replicas of each registry. For large registries, the first consideration can produce several more replicas, which is another reason for not having large registries. There are seven copies of the es registry and six of the pa registry; five servers are required to contain the inboxes for the individuals in each of these registries. The fifth and final consideration for determining registry configuration is to avoid overloading any given registration server with registration data or updates.

6. TRANSPARENCY OF DISTRIBUTION AND REPLICATION

Most users and registrars treat Grapevine as if it were implemented on a single, large, reliable computer that contained all registration data and inboxes, and to which all workstations were connected through high-speed links. Our experience is that most of the time this unitary model serves users and registrars well. There are, however, a few ways in which Grapevine's distributed, replicated implementation shows through now and then to surprise users and registrars.

The most common cause for surprise is delays in propagating registration database changes. Registry replicas will contain different data while an update is propagating. Since updates are propagated by message, inconsistencies can last for many minutes. It sometimes happens that a registrar makes a change to a registry, attempts to do something else which depends on that change, and then finds that the change has apparently been lost. For example, a registrar creates a new individual and tries to add it to a distribution list; the second operation uses a different registration server, so the individual seems not to exist yet. This is not a frequent occurrence, and retrying usually solves the problem. An administrative interface program that picked one registration server for a registry during a session and stuck with it unless that server became inaccessible would reduce the incidence of this problem. A similar problem can occur when a user adds an individual to a distribution list and then immediately sends a message

to that list. If a different server accepted the message for delivery than accepted the distribution list update, then the new member might not receive the message. The sender would probably not notice that this had happened.

Certain tools that registrars would find useful are harder to provide because of the distributed implementation of Grapevine. It would be helpful for a registrar to know that a certain distribution list is no longer being used, but the distributed nature of the implementation makes it hard to record distribution list usage. The database update algorithms are intended for updates to a predominantly static collection of data, and are unsuitable for propagation of rapidly changing details. There is, further, no notion of any kind of "home server" for a registry at which its statistics should be accumulated. We would have thought more about accounting facilities if use charges had been considered.

Deleted names can cause a different management problem with a distributed implementation. Ideally the deletion of a name from a registry would remove it from all groups immediately. It is not practical to do this, because we have no means of knowing even in which registries there are groups containing the deleted name. Consequently the choice we made was to notify the owners of a distribution list when an invalid user was found in their list, that is, when the list was being used to send a message. Automatic deletion at this point is feasible.

It is necessary sometimes for registrars to have an understanding of the effects of their administrative actions on the distributed implementation. This is particularly true of changes to inbox site lists for individuals. In the original design, whenever an inbox site was removed or demoted, for example, by being made secondary rather than primary, its contents were remailed. Remailing can involve a large number of messages, especially when the user has not retrieved his buffered messages for several months, and can generate a large load on the system. We have mitigated this problem by changing the remailing policy so that messages are remailed only when a site is removed from an inbox site list. Even so, substantial remailing can and does occur. It has proved necessary to have a rule of practice that an inbox site should not be removed if there are more than a specified number of messages there; the user is forced to retrieve the messages before the inbox site is removed. We now believe that automatic remailing from removed inbox sites is a mistake. It would be better for the abandoned inbox to be marked for deletion but continue to be accessible until emptied by message retrieval. This would make it necessary to have a manual override in such cases as planned removal of a server from the system. If this change were made then registrars would no longer have to worry about this problem.

Users are sometimes surprised by Grapevine's apparent failure to eliminate a duplicate message from an inbox. Although the usual cause is a sender presenting a message for delivery twice, the distributed registration database does provide a way for real duplicates to occur. The expanded recipient list for a message can contain several instances of a single name, usually because that name appears in two distribution lists, both of which are recipients. For a message sent to one or more distribution lists, the accepting message server may discover that some distribution lists cannot be expanded because all registry replicas have become unavailable, usually due to internet links going down. In this circumstance we have chosen to complete delivery to the known recipients and leave delivery to

the recipients in unexpanded distribution lists until later. This choice can produce duplicated messages. The alternative of waiting until all expansions are completed to deliver any messages removes this source of duplicates, but can needlessly delay delivery to some recipients and is more expensive and complex to implement.

Grapevine's replicated message delivery service sometimes produces surprising results. All message servers are able to accept messages for delivery from any user to any recipients. We assume that the primary consideration is allowing a user to send a message if at all possible, independent of the availability of the local server. The resource location algorithm places delivery with the nearest responding message server without asking or informing the user. Placing delivery with a nonlocal server immediately, however, can result in a longer delivery time than placing the message with the local server a little later. The time may be very much longer if an intervening internet link fails after placement and before forwarding. Users occasionally wonder if a message has been lost if it does not arrive within a few minutes. An associated second order effect is that Grapevine lacks detailed knowledge about the properties of the internet. Resource location decisions are based on the number of internet links in a path, but cannot take into consideration the bandwidth, reliability, or congestion of those links. A server two ethernets away is usually a better choice than one that is two phone lines away, but Grapevine cannot tell them apart.

As described earlier, the message server that accepts a message for delivery is responsible for expanding all recipient distribution lists and determining the best inbox site for all recipient individuals. When large distribution lists or large numbers of individuals are from a registry whose nearest replica is far away, this design can lead to long delivery times. In such cases, improved delivery times would result if the message server forwarded the message to another that is closer to a registry replica for list expansion and inbox site location. On occasion more than an hour has been expended in moving the contents of a large distribution list and the inbox site lists for many individuals over a slow line, and in then sending essentially the same information back again with a small message tacked on.

In designing the system we assumed that users would not want to know much about its state of availability or accessibility, and accordingly they are not told about such things. This was a wrong decision. If a user cannot obtain a service he needs—typically he may find that he cannot receive his mail—he will want to know what the problem is. It might be that the server is down, or that it is restarting, or that it is inaccessible, and the experienced user will have a reasonable perception of how long these states are liable to last and in what circumstances he should bring them to someone's attention. This is an area in which we believe there is a good deal still to be found out, in particular as to how far it is worth complicating the system in order to give accurate information automatically. The state of a distributed, replicated system is much harder to determine and to describe to a user than the state of a unitary one.

7. ADJUSTING TO THE LOAD

In building Grapevine many design decisions were made based on assumptions about the nature of the expected load. Efficient operation of the system depended

on the accuracy of these predictions. Grapvine provided a new set of services to a user community that was growing quickly. While in most instances the original design decisions have stood up well, in a few cases our predictions about the detailed nature of the load proved to be wrong, which sometimes leads to bizarre performance problems. Some facilities that worked well initially broke down as the load grew. To keep the system working well we needed to adjust certain strategies and algorithms to the actual load.

In the original Grapevine design all database changes were propagated to other servers by including the entire changed entry in a message. A server receiving such an update merged it with the local entry, producing a new version of the entry that contained the latest information from both the old local entry and the update. As the system expanded, the size of distribution lists and the frequency of updates increased. The size and frequency of update messages increased correspondingly, as did the cost of performing merges. During the middle of the day some servers would get an hour or more behind in delivering messages because of the update load. When a registration server was down for a day and could not read update messages, its secondary inbox site would run out of free disk space. We responded to these problems by adding a new mechanism to propagate one sort of update. The most frequent change to the registration data is adding or removing one member from a distribution list. Most lists allow users to add and remove their own name. List owners and registrars also perform this operation frequently. For this case we made Grapevine include only the change in the update message. Thus the typical update message now contains just a list name and the added or deleted member, rather than the entire contents of the list. In addition to making a large reduction in the size of the average update message, the computation cost of the incremental update is much less than the cost of the merge. For a large list, the merge can take several minutes while the incremental update takes just a few seconds. Introduction of these "brief" updates produced enough extra capacity that the system could easily keep up with the midday load. We retain updating by complete merging of entries for all other types of database changes, because they occur infrequently and because for certain sequences of changes, (e.g., to an inbox site list), the order-preserving behavior of merges is required for correctness.

A design goal of Grapevine was to become the source of authentication and access control information for the internet. Before Grapevine was available, each file server maintained its own record of user names and passwords. It also maintained complete access control lists for each file. A user requiring access to several file servers had to be registered independently with each. After Grapevine servers had become fairly widely spread in the internet, the standard file server was changed to use Grapevine for both of these functions. The file servers no longer contain the passwords of clients. The file server checks a user's credentials with Grapevine. Any valid Grapevine individual can login to any file server. Access control lists contain the names of Grapevine individuals and groups. To determine if access to a particular file is allowed, the file server looks for the name of the logged in individual on the access control list of the file. If the name is not found then the file server asks Grapevine if the individual is a member of any of the groups on the access control list. There were three potential pitfalls in this scheme. First, it could require both the file server and a suitable Grapevine

server to be accessible for a file access to succeed. Second, it might be noticeably slower in making access control decisions than the old scheme. Third, it might overload the Grapevine servers with authentication and access control checks. To overcome these pitfalls, the results of authentication and access control checks are cached in the file server. The cache time-out is 12 hours. After that time-out an authentication is rechecked, but a timed-out authentication is believed if no Grapevine server can be contacted. Positive and negative group membership results are timed-out the same way. The cached negative results are just used as a hint to optimize the order of membership checks. The long time-out means that certain authentication and access control changes can take a long time to become effective: adding permission to reference a file takes effect quickly; the effect of revoking permission and changing passwords can be delayed, but that is consistent with our security environment.

We were somewhat surprised to discover that, even with caching, users complained that access control checks could take a very long time, especially when access to the file was denied. The problem stemmed from the way Grapevine groups are nested. Groups can contain groups, and the membership check used for access control determines membership in the closure. Some of the high-level organizational groups used for access control had 50 groups nested to three levels naming several hundred individuals. The nested groups were not all in the same registry. The membership check required finding the type of each name encountered by asking a registration server, and if it was a group then getting its membership list, possibly from another registration server. To discover that a name was not in the closure could take five minutes. These problems were overcome by two changes. First, a new type of membership check was introduced. It assumes a name is that of an individual unless the first part of the name ends with the "up-arrow" character: ↑. Since that character is used to distinguish distribution lists for the convenience of human users, it provides a syntactic method for recognizing the groups that require further expansion. With the "up-arrow" closure, it is no longer necessary to look up every name in the registration database, only the names of syntactically recognized groups. Since a particular name can never change from its original type, whether it is a group name or an individual name, it would have been possible to mark names automatically and avoid the explicit syntactic distinction. Second, for the most complex groups used for access control, a parallel flattened version is maintained. The file server uses this flattened version for access control checks. The flattened version, whether manually or automatically created, can be slightly out of date. The addition of the file server authentication and access control load to Grapevine triggered a need for additional servers, but has greatly simplified file sharing in the internet.

Grapevine is designed so that inboxes are read sequentially and the contained messages then deleted. The assumption is that clients have a place to store received messages, that is, that computer mail system users have personal workstations with local disks. Users with personal workstations, however, occasionally need access to accumulated messages when away from their workstations. To permit access from home or when traveling, we provide a service that allows message reading and sending from standard terminals. Using this service, avail-

able through a dial-in port, the messages in a user's inboxes can be read in random order without being deleted. We did not anticipate the demand that would be generated by the proliferation of personal computers within the company. Computer mail is extremely useful, and these users want access to it. Their computers generally have no special software for interfacing to Grapevine. Rather they run programs which simulate standard terminals and contact our terminal service. The result is a growing number of users who access the message system only via the terminal service. The Grapevine inboxes that are intended as buffers become semipermanent repositories for these users, loading the system in unintended ways. The disks begin to fill up with messages, the protocols intended for sequential access to all the messages in an inbox are used to inefficiently simulate random access to a subset, and connections to the servers remain open for long periods. In this case we refused to alter the system to meet the new load. The problem has been controlled by administrative limits on the numbers of such users. We conclude that any viable, commercial message system must provide well thought out facilities for users who have only terminals, or personal computers that simulate terminals, for accessing the message system. In particular, more inbox storage space, protocols for efficiently accessing the accumulated messages in random order, and higher connection limits are required.

8. OPERATION OF A DISPERSED SYSTEM

Because Grapevine is geographically dispersed, it is important for smooth and efficient operation to make monitoring, control and repair functions accessible through the internet. Two different sorts of people are involved in operating the system: operators for the server computers and system experts. Operators must be present at the site of each Grapevine server, so it is important that they be able to carry out their duties with a minimum of special knowledge about how Grapevine works. System experts are in short supply and are remote from almost all servers. They must be able to do their job without traveling to the server sites.

The responsibilities of operators include rebooting the machine, loading specific programs into it for diagnosis or repair, and getting broken hardware fixed. Grapevine software is prepared to be stopped without warning and restarted using simple procedures posted on the server. Thus operators with a minimum of training can deal with power failures, arrange for repair of broken hardware by local technicians, and handle similar problems without central coordination or advice. The replication of services in Grapevine usually allows such repairs to be made without depriving local users of any services.

Some server failures, however, require assistance from experts who know in detail how Grapevine works. The most common example is a corrupted disk. When the disk has incorrect bits stored on it due to hardware failures or software bugs then the server may crash and not be restartable. In such cases the operator runs a special program on the Grapevine computer that allows access to the disk from a remote workstation. An expert, contacted by phone or message, diagnoses and repairs the problem using an interface program on his workstation which reads and writes the Grapevine server's disk one-page-at-a-time over the internet. The expert's interface program contains facilities for interpreting and displaying the data structures on the disk. Repair of isolated disk errors is possible because

of redundant, low-level structural information. At a higher level, all registry data is replicated on other servers, allowing the content of registration database entries to be replaced. We made an engineering decision early in the Grapevine design *not* to replicate message bodies. Thus repairing a corrupted disk may require replacing a broken message with an apology, a measure that has been necessary just once. (We justified the decision not to replicate message bodies because the cost was too high for the benefit derived. Message bodies are almost never lost and, when they are, a sender frequently has a copy of an important message and will notice that it did not solicit a response from the recipient). Once the bits that were wrong are inverted, the server can be restarted from the expert's workstation. In the current system, remote disk repair has been needed once every two or three months, and usually takes less than a hour to perform. It would be straightforward to add code to the server restart sequence to make most such repairs and reduce the need for expert intervention; in a commercial system development of such software would be justified. In the case of total disk failure, the only recourse is to build a new server disk from scratch. All unretrieved messages would be lost, but registry data, except for unpropagated updates, would be replaced from the replicas on other servers. We have had only one total disk failure.

Grapevine servers include a facility, called the viticulturist's entrance, for remote monitoring and control via an interactive byte stream through the internet. A standard protocol is used that allows a remote workstation to behave like a terminal. Use of viticulturist operations is controlled with Grapevine's authentication and access control mechanisms. Facilities include displaying various operating statistics, altering variables that control certain resource management algorithms, and manually starting automatic processes. A server can be restarted with new software. Using the viticulturist's entrance with other communications protocols, such as file transfer, we have been able to operate this widely dispersed, growing system without experts ever visiting remote server sites.

One monitoring feature of the servers has been of particular value in understanding how the system works and in finding problems. Each server maintains a text log of its actions. For example, the following five items abstracted from Cabernet's log show the entries concerned with the acceptance and delivery of a message sent to a distribution list. Of the seven list members, six had local inboxes sites and one was on another server. The long number is a message-identifier.

```
15:14:02 Created 3#273@2588022842: sender Schroeder.pa
15:14:05 Received 3#273@2588022842: 1 recipients, 137 words
15:14:20 Delivered 3#273@2588022842, 6 local, 1 remote.
15:14:21 RecipientLog 3#273@2588022842: Transport↑.ms
15:14:30 Forwarded 3#273@2588022842 to Riesling.ms
```

The logs are backed up to circular buffers on file servers so that more than a week's history is always available. If malfunctions occur the build-up to failure can readily be reconstructed from these logs. It is possible to trace distributed operations through the logs of the various servers by tracking unique identifiers. When a user complains that a message did not arrive in a timely way, for example,

we can determine the details of its journey. Perhaps of even greater value is the ability for log additions to be viewed dynamically through the viticulturist's entrance. A system expert can open viticulturist's connections to several servers at once from his workstation and watch (in separate windows) the servers operate. Dynamic viewing has two advantages. First, the expert can see whether corrective action has had the desired effect, for example, by watching the free disk space increase as a result of forcing the archiving of a large inbox (see Section 9). Second, he can notice oddities. Recently, for example, we noticed that a registration server was taking so long to process an update that the connection to its inbox timed out. This meant that the update message was not deleted from the inbox, so that when the registration server reopened its connection it received the same update over again, more or less indefinitely. This was easily remedied, but it was the serendipitous aspect of log viewing that detected the problem.

Another way that Grapevine provides feedback to system experts is through the dead letter facility. Whenever a message is returned to its sender, a copy of the header of the returned message along with the reasons for delivery failure are sent to a special distribution list. This list contains the names of system experts, who thus see evidence of unusual events associated with message delivery.

Most hardware and software failures cause the server to end up in the Mesa language debugger with an uncaught signal. The version of Mesa in which Grapevine was written includes no facilities for remote debugging, so this event requires intervention by operators to note the signal name and restart the server. It would be straightforward to replace the debugger with a special program that reported the signal name and restarted the server, perhaps after requiring a remote authorization. With this change operators would almost never need to intervene.

9. RELIABILITY

A design objective of Grapevine is high reliability. A primary technique for achieving high reliability is replication of function among several servers. When one server is unavailable, others can perform the same functions for clients. The goal is that failure of a single Grapevine server not make any service unavailable to any client. Our experience in running Grapevine shows that this approach has been extremely successful. Most users do not notice when a server fails, and we sometimes have to prod local site personnel to get a server restarted.

Bug-free software, reliable hardware, and reliable communications are the foundations of a reliable system. Equally important, but more subtle, is appropriate management of resources. Lack of resources can lead to expanding paralysis of the system. The basic resources are processor cycles, memory space, communications bandwidth, and disk space. In Grapevine, processor cycle and memory space capacities are reflected by limits on the number of connections of various sorts a server will allow at one time. Communications bandwidth is a resource that Grapevine cannot manage directly; Grapevine is but one of many customers of the internet and must compete for service as best it can. Disk space is managed by having Grapevine servers dynamically reject new connections that might cause further disk space allocation when the free disk space is below 5% of the total available. No mechanisms are included for a server to abort existing

computations when disk space is low, so a server can still deadlock itself by running out. Single server disk deadlock has occurred only a few times, and is treated as a symptom of an overloaded server. It also is possible for multiserver deadlocks to develop, for example with each server refusing to accept incoming messages for local inboxes because it cannot forward outgoing messages to the others. Multiserver deadlock has occurred only when specifically provoked by an exerciser in a test system.

It is easy to overlook the implication of the fact that using redundancy to achieve reliability requires the system to have spare resources in normal operation. Such extra capacity is also important for handling peak loads gracefully. Spare capacity, however, tends to get used up, without anyone noticing, as system load grows. When a server then fails and load shifts to others, the additional load might be rejected, thus defeating the goal of reliable service. For a while the load on Grapevine was growing rapidly. The system would operate reliability, without expert intervention, for a month or two at a time and then without warning a server failure, load peak, or communication failure would set off a chain reaction of overloading. On one notable occasion, inadvertent remailing of a very large inbox caused a paralysis of disk overloading to spread in two days to all but two isolated servers. This event caused us to change the software so that a server can be restarted even when there is no free space left on the disk, and to be more careful about provoking remailing of large numbers of messages.

To some extent, we have learned how to configure the system to reserve capacity and spread the backup responsibility. The simplest way is to configure some servers to have only secondary inboxes on them (see Concord and CheninBlanc in Table I), although it is sometimes hard to convince managers to buy the required extra servers. Another technique is to split the secondary inboxes for individuals whose primary inboxes are on a certain server among several other servers, so when the primary server fails the load is spread to more than one other server. In general, system experts need to think carefully about how loads will be distributed when various servers become unavailable or when various communication links go down, and configure the registration data and inbox site assignments so that sensible behavior will result. These arrangements need to be reviewed from time to time as the pattern and size of loads change.

Grapevine is used for purposes other than delivering computer mail. In one case, Grapevine message and registration services are used to control an integrated circuit manufacturing facility [4]. An extra Grapevine server (Chardonnay) was added to the system, and configured with very little registry data and few inboxes to obtain extra reliability and performance for this application. This configuration insures that the server will not be overloaded with registration service requests or inboxes. The present design, however, includes no facilities for restricting access to its maildrop service. Its position in the internet indicates that under normal operation only local clients use it as a maildrop. But when other servers are unavailable this "extra" maildrop capacity is quickly found by the resource location algorithms, and this server can get backed up with messages to forward. We conclude that mechanisms are required in this case to limit the set of clients that can use a server's maildrop resource, so that this capacity can be reserved for its intended use.

Grapevine design follows the principle that any interactive request can be performed by one of several different servers. In one instance this principle has been violated, and the result is the expected perception by users of a less reliable service. Grapevine servers have very small disks in relation to the total size of the messages that can accumulate in an inbox. To keep the small server disk from filling up, every night all inboxes containing any messages older than seven days are archived to a nearby file server. Archiving is transparent to the client; when an inbox is emptied any archived messages are retrieved from the file server by the Grapevine server and then presented to the client. Between ten and twenty percent of the messages retrieved from Grapevine have been archived. Archiving was one of the last features added to Grapevine and it was not done very well. If the file server containing archived messages for an inbox is not accessible, then no messages may be retrieved from that inbox. Unfortunately, the inbox is still unavailable to receive new messages. In this state, then, new messages for a client will be directed to an inbox from which they cannot be retrieved. Thus availability of the message delivery path to an individual that has archived messages can depend on the availability of a particular archival file server. The rest of Grapevine's services are reliable enough that this exception is conspicuous to users. Message retrieval protocols that allow for some messages in an inbox to be inaccessible would solve this problem. A larger disk and less frequent archiving would also be desirable. For the largest community now served by a single Grapevine server, a disk of 80 Mbytes would make it possible to dispense with archiving altogether, as long as registrars were informed of unusual accumulations of messages and had tools for deleting the oldest subset of messages from an inbox.

With the present archiving arrangements it is still possible for a disk to get too full. System reliability would be improved if the age parameter that controls archiving were adjusted automatically in relation to the amount of free disk space remaining, and if inboxes that grew larger than some threshold were instantly archived. The latter mechanism would help the server defend itself against the arrival of a large amount of material for one or a few inboxes over a short period of time, as occasionally happens.

10. CONCLUDING REMARKS

Throughout this discussion the reader may have noticed that we have described potential improvements to Grapevine that have not been carried out. As time has passed, we have become more and more reluctant to change the software. This reluctance is partly due to the potential disruption that introduced bugs would have on the large user community that depends on Grapevine services to get its work done. It also is due to the fact that we are slowly forgetting the details of the implementation and thus becoming less able to predict the consequences of changes. The result is that proposed changes to the system are very carefully considered and only those with a very large payoff or low risk (usually both) are made.

Another reason why certain improvements have not been made is that Grapevine has been adopted by Xerox as the basis for the 8000 NS product message

system and clearinghouse. As a starting point for conversion to a product, the Grapevine server program was adapted to run under the Pilot operating system [15] on the 8000 NS processor ("Mission" and "Riesling" are Pilot-based 8000 NS servers). If the operational impact of an improvement would not be immediate we have tended to tell the implementors of the product system about the change rather than adding it to Grapevine. Some of the changes proposed in this paper have already been adopted in the product. The registration service has been generalized to become the product clearinghouse service. Generalizations include increasing the number of levels in the naming hierarchy to three so the product can serve a much larger user community, and allowing an expandable set of values to be associated with names in the database so the clearinghouse can answer a wider range of questions. The product message service implements inboxes in a different way so it can provide better service to users with terminals instead of workstations; a larger disk allows long-term storage of messages in inboxes, and the access protocols permit random access to the messages in an inbox. As a result of the larger disk, the need for automatic archiving is eliminated. The product system also includes a general facility for interconnecting with a variety of other message systems—Grapevine has only a special purpose facility for interconnection with the Arpanet message system. The protocols for using the product clearinghouse and message services are built on the published remote procedure call protocols [7], rather than on byte streams.

ACKNOWLEDGMENTS

Dave Redell heads the development of the 8000 NS message system. Ed Taft designed and implemented the file server facilities that use Grapevine authentication and access control. Steve Temple designed and implemented the facilities for remotely patching a Grapevine server disk. Gail Allen, Art Axelrod, Paul Brol, Carl Chilley, Marcy Congdon, Elinore Cowhig, Ron Cude, Connie Dones, Lita Germain, Chuck Hains, Robert Kierr, Joyce LaCoe, Michael Rutkaus, Lili Sanders, Connie Slawecki, Patty Smith, Ed Stone, Ron Weaver, and Bob Yost are remote site personnel who have helped us learn how to administer, expand, explain, improve, and repair the Grapevine system.

REFERENCES

1. BIRRELL, A.D., LEVIN, R., NEEDHAM, R.M., AND SCHROEDER, M.D. Grapevine: an exercise in distributed computing. *Commun. ACM 25*, 4 (April 1982), 260–274.
2. BIRRELL, A.D. AND NELSON, B.J. Communication techniques for remote procedure calls. *ACM Trans. Comput. Syst. 2*, 1 (Feb. 1984), 39–59 (this issue).
3. BOGGS, D.R., SCHOCH, J.F., TAFT, E.A., AND METCALFE, R.M. Pup: An internetwork architecture. *IEEE Trans. Commun. 28*, 4 (April 1980), 612–634.
4. BROTZ, D.K. Laurel manual. Tech. Rep. CSL-81-6, Xerox Palo Alto Research Center, Palo Alto, Calif., 1981.
5. BROTZ, D.K. IC fabrication information control via an electronic message system. In *Proc. Electrochemical Society Conference on Computer Controlled IC Processing and Monitoring*, San Francisco, May 1983.
6. COMER, D. The computer science research network: a history and status report. *Commun. ACM 26*, 10 (Oct. 1983), 747–753.
7. Courier: the remote procedure call protocol. Xerox System Integration Standard XSIS-038112, Xerox Corporation, Stamford Conn., Dec. 1981.

8. CUNNINGHAM, I., DELESTRE, D., KERR, I., MYERS, T., SEKIDO, Y., TOUILLET, D., WARE C., AND WEST, N. Emerging protocols for global message exchange. In *Proc. Compcon '82—25th IEEE Computer Society International Conference* (Sept. 1982), IEEE, New York, pp. 153–161.
9. DAWES, N.W., 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.), Elsevier-North Holland, New York, 1981, pp. 373–384.
10. Ethernet, a local area network: Data link layer and physical layer specifications version 1.0. Digital Equipment Corporation, Intel Corporation, Xerox Corporation, September 1980.
11. METCALFE, R.M., AND BOGGS, D.R. Ethernet: Distributed packet switching for local computer networks. *Commun. ACM 19*, 7 (July 1976), 395–404.
12. MITCHELL, J.G., MAYBURY, W. AND SWEET, R. Mesa language manual (Version 5.0). Tech. Rep. CSL-79-3, Xerox Palo Alto Research Center, Palo Alto, Calif., 1979.
13. MYER, T.H. AND VITTAL, J.J. Message technology in the ARPANET. In *Proc. IEEE National Telecommunications Conference '77* (Dec. 1977), IEEE, New York.
14. NOWITZ, D.A. AND LESK, M.E. A dial-up network of UNIX systems. *Unix Programmer's Manual* (7th ed.), vol. 2B, Bell Laboratories, Murray Hill, N.J., 1978.
15. REDELL, D.D. Pilot: an operating system for a personal computer. *Commun. ACM 23*, 2 (Feb. 1980), 81–92.
16. 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, 72-79, IEEE, New York.
17. THACKER, C.P., MCCREIGHT, E.M., LAMPSON, B.W., SPROULL, R.F., AND BOGGS, D. Alto: A personal computer. In *Computer Structures: Principles and Examples.* (2nd ed.), D.P. Siework, C.G. Bell, and A. Newell (Eds.), McGraw-Hill, New York, 1981.

Received March 1983; revised October 1983; accepted November 1983