Techniques for Supporting Wide Area Distributed Applications

Michael F. Schwartz

Panagiotis G. Tsirigotis

CU-CS-519-91 February 1991; Revised August 1991

Department of Computer Science
University of Colorado
Boulder, Colorado 80309-0430
(303) 492-7514
electronic mail contact: schwartz@cs.colorado.edu

Abstract

In this paper we present a number of techniques for supporting distributed applications that span many nodes across administrative boundaries in wide area internetworks. The techniques derive from experiences with research prototypes we have built in a variety of application areas. Several of the prototypes have achieved exceptionally far reaching distribution, operating in conjunction with machines in thousands of sites around the world. The techniques are applicable to a broad range of distributed applications. They concern problems with fault tolerance, administrative decentralization, scalability of operational semantics, organizing decentralized information, controlling the spread of distributed operations, and user interfaces.

1. Introduction

As wide area internetworks achieve increasingly far reaching deployment, opportunities arise for building many new types of applications that can make effective use of these networks. We focus here on the class of *wide area distributed applications* with two defining characteristics. First, these applications operate across administrative and possibly national boundaries. Second, they involve communication between many different machines, potentially involving any of thousands or millions of different machines in a wide area internet.[1]

In the past several years, a number of projects have been initiated with the goal of building some type of wide area distributed application. Some efforts focus on extending older paradigms into wider distribution, most commonly in the realm of mail, news, and file service [Gifford et al. 1985, Quarterman 1990, Spector & Kazar 1989]. Other efforts focus on interconnecting information repositories to provide easy access to libraries and other information from any point in the Internet [Evans et al. 1989, Kahle & Medlar 1991, Kahn & Cerf 1988, Lynch 1990]. Physical scientists are exploring the possibilities for supporting large scale scientific data sharing [French, Jones & Pfaltz 1990, Menke et al. 1991, NASA 1990]. U.S. government agencies have initiated efforts to support collaborative research and development [NASA 1991, Wulf 1989]. Finally, the number and variety of network accessible resources motivates efforts into directory service and resource discovery [CCITT 1988, Danzig et al. 1991, Peterson 1988, Schwartz 1991c].

Because of their scale of scope of distribution, wide area distributed applications exhibit many characteristics not found in locally distributed applications. Operations can experience additional failure modes stemming from the complexity of wide area internets, and from inconsistent implementations of component subsystems. The number of machines involved in performing operations on behalf of an application can increase the seriousness of a program operating incorrectly. The set of information and other resources that are accessed can be large and complex, making use of many subsidiary databases and networked information sources, in continuously changing combinations. This organizational complexity can be overwhelming to users. Administrative decentralization introduces a number of other problems, including security concerns and difficulty of reaching consensus about how a system will operate. Clearly, a number of other problems exist as well.

In this paper we discuss a number of techniques for supporting wide area distributed applications, based on our experiences with a number of experimental prototypes in the areas of heterogeneity, resource discovery, distributed collaboration, and network measurement. Several of the prototypes have achieved exceptionally far reaching distribution, operating in conjunction with machines in thousands of sites around the world. The scope of distribution of these prototypes provide a set of experiences relevant to many wide area distributed applications, beyond the areas of focus of the prototypes. We focus our discussions here on technical issues for supporting wide area distribution. While important, policy issues, privacy and security considerations are outside the scope of this paper. We also exclude issues that arise in supporting more general centralized and locally distributed systems, as have been covered by Lampson [Lampson 1983] and Black [Black 1985].

The remainder of this paper is organized as follows. In Section 2 we overview the experimental prototypes upon which the experiences in this paper are based. In Section 3 we present the techniques derived from our prototype experiences, and characterize the types of applications to which the techniques can be applied. The techniques we cover relate to fault tolerance, administrative decentralization, scalability, organization, controlling the spread of distributed operations, and user interfaces. In Section 4 we summarize the realm of applicability of the techniques and offer our conclusions.

2. Overview of Experimental Prototypes

In this section we overview the scale, distribution, and basic workings of the prototype wide area distributed applications and measurement studies upon which the techniques discussed in this paper are based. We describe the prototypes below, in decreasing order of their degree of contribution to the techniques in the current paper. The techniques and their rationale will be considered in more detail in the remaining sections of this paper.

---

[1] Throughout this paper we use "**i**nternet" to refer to general internetworks. We use "**I**nternet" to refer specifically to the growing collection of interconnected IP [Postel 1981b] networks that join academic, industrial, and government institutions world wide.

Internet "White Pages"

The prototype upon which the largest number of techniques in this paper are based is an Internet "white pages" directory tool called "netfind" that we developed, deployed, and measured extensively [Schwartz & Tsirigotis 1991]. Given the name of a user and a rough description of where the user works (e.g., the company name or city), netfind attempts to locate telephone and electronic mail box information about that user. Netfind uses a number of existing protocols and highly decentralized sources of relatively unstructured information, allowing it to succeed in the presence of failures or partial remote protocol support. Measurements indicate that the scope of the directory is upwards of 1,498,000 users in 2,520 sites.[2] This scope is significantly larger than other existing Internet directory services, which require that users register with an administratively centralized service (as with the SRI Network Information Center WHOIS service [Harrenstien, Stahl & Feinler 1985]), or that special servers be run at many sites around the Internet (as with the CCITT X.500 standard [CCITT 1988]). Moreover, because netfind contacts the (highly decentralized) machines on which users do their daily computing, it is able to locate very timely information about users. Netfind is in active use at approximately 70 institutions worldwide, and is being developed further commercially.

The approach taken in the design of netfind was originally motivated by practical concerns with providing a facility that could locate a usefully large number of users. To satisfy this goal, the design favored existing information sources that were difficult to use because of the scope of their distribution (the global collection of user's workstations). Addressing these difficulties led to a number of generalizable techniques, particularly because the tool's usefulness led it to receive a significant level of use.

We begin with a database of "seed" data, which provides hints of potential machines to contact when a search is requested. This database is built by gathering information from the headers of USENET [Quarterman & Hoskins 1986] news messages over time. These headers typically list the user name, organization name, city, and electronic mail box for users who post messages. When a search is requested, the seed database is consulted to locate the names of a number of machines associated with institution keywords specified in the search request. Requests use the format "*UserString InstString [InstString ...]*", where *UserString* identifies the user (typically by last name), and the conjunction of one or more *InstString*s identify the institution where the user works. For example, a search could be requested for "schwartz university colorado" or "schwartz boulder".

If the machines found in the seed database fall within more than three naming domains (an example of one domain being "cs.colorado.edu"), the user is asked to select at most three domains to search. The Domain Naming System [Mockapetris 1987] is then contacted, to locate authoritative name server hosts for each of these domains. The idea is that these hosts are often central administrative machines, with accounts and/or mail forwarding information for many users at a site. Each of these machines is then queried using the Simple Mail Transfer Protocol (SMTP) [Postel 1982], in an attempt to find mail forwarding information about the specified user. If such information is found, the located machines are then queried using the "finger" protocol [Zimmerman 1990], to reveal more detailed information about the person being sought. The results from finger searches can sometimes yield other machines to search as well. Moreover, a number of mechanisms are provided to allow searches to proceed when some of the protocols are not supported on remote hosts. Ten lightweight threads are used to allow sets of DNS/SMTP/finger lookup sequences to proceed in parallel, to increase resilience to host and network failures.

Measurement Study of Growth and Disconnection Patterns in the Global Internet

Another application upon which many experiences in this paper are based involves measuring changes in upper layer service reachability on the global TCP/IP Internet. The motivation for this study was the observation that recent highly publicized security threats could cause many sites to reduce their connectivity with the global TCP/IP Internet, by limiting what types of traffic are allowed to flow between the Internet and their internal networks. We are particularly concerned about this possibility, because for the past several years our research has become increasingly focused on the Internet as an experimental environment in which to deploy and measure wide area distributed applications. Moreover, diminished Internet connectivity will hinder or prevent the deployment of interesting new types of network services.

[2] These figures are 31% higher than those given in [Schwartz & Tsirigotis 1991]. They are based on measurements taken 6 months later than the original measurements, and reflect the rapid growth of the Internet. The measurements also underscore the ability of the techniques used by netfind to keep pace with Internet growth.

Collecting data for this study requires a series of runs of a program that attempts to connect to 12 different TCP ports on a small number of hosts in each domain in the Internet, using a set of 20 parallel threads. Each thread executes a specific measurement protocol and timeout mechanism to determine which services are available, which domains limit availability of these services in response to perceived security threats, and how service reachability limits are imposed. By running this study periodically over a period of time, we will be able to measure global changes in service level reachability.

This study evolved from an earlier study of the scope of users reachable by netfind, which attempted to connect to ports used by the netfind algorithms. By running the netfind scope measurements twice separated by a period of six months, we were able to characterize patterns in how sites that used to support Internet services stopped supporting them, as well as patterns of sites joining the Internet and supporting these services. We completed this initial version of the disconnection study in February 1991 [Schwartz 1991b]. The current study uses a methodology that avoids measurement errors associated with the fact that the Internet is growing at the same time as disconnections are occurring; reaches many more sites (nearly 13,000 domains world wide, rather than 2,400 in the original study); probes 13 services designed to characterize various aspects of participating in the global network service infrastructure, rather than the three services used by netfind; and discerns more details about the nature of the network distancing mechanisms being used [Schwartz 1991a]. We have made several measurement runs of the current study, and will continue to do so over the course of the next 6-12 months.

Measurement Study of Organizational Properties in Global Electronic Mail

Another project that contributed to the techniques discussed in the current paper involved measuring organizational properties in global electronic mail. The basic motivation for this study was the observation that hierarchical organization scales well, but is too inflexible to model complex relationships. In this study, we analyzed the organizational structure that arises naturally when people communicate, and the extent to which one can derive clues about this structure from a communication graph. We explored this structure by analyzing data collected about electronic mail among approximately 50,000 people in 3,700 different administrative domains world wide.

Our results fell into three categories. First, we provided measurements of the basic graph structure, to indicate its redundancy and interconnection density. Second, we introduced an algorithm to cluster individuals by shared interests without access to the contents of mail messages, by computing properties of the mail interconnection graph using a combination of graph reductions and traffic analysis. This algorithm has powerful potential applications, as well as privacy implications for distributed collaboration. Third, we applied the algorithm to each of a randomly sampled subset of 500 nodes within the graph, to derive measurements of how people collaborate.

Internet Resource Mapping/Discovery Project

Another project that fed into the experience base for the current paper is an effort to support resource discovery in decentralized environments of such scale that the resource space cannot be completely organized. That project focuses on mechanisms that support incremental organization of the resources, based on the efforts of widely distributed individuals, and a range of different information sources of varying degrees of quality. Our approach to this problem has two parts. The first part is to use mechanisms that "tap into" existing network protocols and information sources to provide an immediately useful tool, in a manner similar to the approach taken by netfind. The second part involves mechanisms that allow users to superimpose additional organization on the resource space in an incremental fashion. We developed a prototype that focuses on public Internet archive sites accessible via the "anonymous" File Transfer Protocol [Postel & Reynolds 1985]. This is an interesting test case, because it encompasses thousands of administratively decentralized sites containing a large collection of resources of considerable practical value.

In this prototype, three levels of information quality are supported [Schwartz et al. 1991a]. At the highest level, resources are described using a structure that allows any individual or group of users who share common interests to build a structure (called a *view*) that superimposes organization on the resource space according to their particular biases and interests. At this level, resources are described according to their conceptual roles. Below that, per-user and per-user-site caches are maintained, to record resources that have been found by individual users during their explorations. At the lowest level, the system scans USENET electronic bulletin board articles using a simple set of heuristics to recognize announcements about public archive sites, to provide a simple keyword-based index of resources throughout the Internet.

Network Visualization Project

Another project related to the experiences in the current paper involves using resource discovery techniques to support visualization of characteristics of large internets, such as topology, congestion, routing, and protocol usage [Schwartz et al. 1991b]. As with netfind and the anonymous FTP prototype, we use a number of protocols and information sources, to support discovery in the absence of global agreement on any one protocol or information source. For the visualization project, however, we use a much more extensive collection of protocols and information sources, including the Address Resolution Protocol [Plummer 1982], the Internet Control Message Protocol [Postel 1981a], the Simple Network Management Protocol [Case et al. 1989], and fifteen others. Essentially, this project is exploring the extent to which one can integrate a heterogeneous, administratively decentralized network by using resource discovery techniques.

We have built a prototype implementation that can collect network information using a small number of network protocols, and that allows a user to retrieve and display this information graphically.

Heterogeneous Naming and Remote Procedure Call

The final project related to the experiences in the current paper involved two subsystems built in conjunction with the Heterogeneous Computer Systems project at the University of Washington [Notkin et al. 1988]. The first subsystem was a Heterogeneous Name Service (HNS), designed for evolving systems that are composed of a heterogeneous collection of subsystems [Schwartz, Zahorjan & Notkin 1987]. The HNS made use of name services and associated data already existing in the individual system components. The major advantage of this design is that it allowed the underlying subsystems to evolve independently of the global name service, while still reflecting this evolutionary change to the clients of the HNS. The HNS separated the task of managing the semantics of operations on underlying name spaces from the task of accessing these name spaces. The system interposed modular procedures called Naming Semantics Managers (NSMs) between clients and the underlying name services, in such a fashion that only the HNS needed to be informed when a new NSM was placed in the system.

The second subsystem supported remote procedure calls between heterogeneous computer systems [Bershad et al. 1987]. The HRPC design involved the careful specification of interfaces between the five principal components of an RPC facility: the *stubs*, which are interposed between the client (also the server) and the runtime support; the *binding protocol*, which allows a client to locate a particular server; the *data representation*, which determines how data values are communicated; the *transport protocol*, which determines how data is carried from one host to another; and the *control protocol*, used internally by the RPC facility to track the state of a call. An RPC client (or server) and its associated stub could view each of the remaining four components as a "black box", and these black boxes could be "mixed and matched". The set of protocols to be used was determined dynamically at bind time.

3. Techniques Derived from Prototype Experiences

In this section we present the techniques derived from our prototype experiences. In some cases we refer to measurements of the prototypes, to help characterize the level of success we achieved in applying particular techniques. In other cases the experiences relate to problems we encountered in deploying a prototype, of such a nature that it was fairly obvious how well the solution would work. In other words, the important point in such cases was that the problem was encountered. In such cases we simply describe the problem and our solution in a qualitative fashion.

We divide the techniques we have developed to support wide area distributed applications into six parts. We begin by discussing techniques for improving fault tolerance, and then techniques for accommodating administrative decentralization. We then consider issues in scalability from the perspective of the operational semantics perceived by users. We then consider techniques for organizing an information space understandably. Next, we discuss techniques for controlling the spread of distributed operations. Finally, we discuss user interface techniques.

Note that these techniques were derived from experiences with a number of different research prototypes. It is unlikely that all techniques could be applied simultaneously in the context of a single application.

While the techniques we discuss were motivated by particular prototypes, the techniques are applicable in a wide range of circumstances. To aid the reader in applying these techniques to other situations, we discuss the application of each technique in several different prototypes. We also discuss the realm of applicability of each technique by indicating limitations among five dimensions. These dimensions are not meant to be a general taxonomy of impacts of distributed computations. Rather, they were derived by listing the set of limitations for all of the techniques. The first dimension is communication/information volume, which considers the volume of information that must be transmitted. This dimension considers the impact of the techniques on the shared wide area communications infrastructure (the network links, routers, name servers, etc.). The second dimension is remote processing volume. In contrast to communication/information volume, this dimension concerns the impact of the technique on the participating end systems, which in general will not be part of the shared infrastructure. The third dimension is correctness/failure semantics, which considers the impact of a technique on correctness guarantees and failure semantics. This dimension examines problems that arise when trading off exact behavior for improved performance or scalability. Dimension four is machine parsability of results, which concerns the degree to which the information generated by an application using a technique can be utilized directly by a computer, without manual human interpretation. Dimension five is human interaction, which concerns the degree to which a human will guide the progress of an application that uses these techniques. This dimension impacts applications that must run autonomously (e.g., because the volume of effort is too much for a human to guide) as well as applications that run slowly over a long period of time (making human interaction infeasible).

3.1. Fault Tolerance

Wide area distributed applications can fail in a number of ways. Like locally distributed applications, they may experience host failures, network failures, and software errors. However, because of the scale of the environment, these errors may occur in more situations, and in more complex combinations. Moreover, in addition to these failure modes, wide area distributed applications can fail because of heterogeneous administration. For example, remote nodes may run different subsets of protocols, and may support different information sources.

In this section we consider ways to improve a wide area distributed application's tolerance to host and network failures. We consider issues raised by decentralized administration in Section 3.2.

Support Redundancy Through Moderate Parallel Probabilistic Access

The primary means of improving fault tolerance in a distributed environment is through redundancy, for example by running multiple servers on independent nodes, and providing redundant network paths to these nodes. To support this redundancy, there must be some means of managing the information sources and choosing between them. Traditionally this has been accomplished by using a globally agreed upon replication scheme, such as voting [Gifford 1979] or primary copy [Oki & Liskov 1988]. As an environment becomes larger and more decentralized, however, reaching global agreement becomes difficult.

For operations that do not require a large amount of remote processing or communication, an alternative means of supporting redundancy is to use parallel probabilistic access to remote operations. Rather than relying on a controlled selection of remote servers, one can invoke a moderate number of redundant remote operations in parallel, and then choose among their results. Our experience here derives from netfind's use of parallel remote SMTP and finger queries to locate users. By using such queries, netfind achieves high availability without requiring any agreement on a replication strategy. Moreover, this technique makes the delays usually associated with remote operation timeouts less perceptible.

This is an interesting technique, because it suggests a use for parallelism other than the usual reason of speeding up computationally intensive processing. Because of the inherent latency of long distance communication, wide area distributed applications may have difficulty realizing parallel speedups. In such a case, parallelism may be more appropriately applied to improving fault tolerance, and reducing response time variance. We discovered this point when we first added parallel threads to netfind. We found that response time improved only slightly when changing from 1 to 30 threads: successful search times decreased from approximately 11 to 10 seconds on average, and unsuccessful search decreased from 38 to 31 seconds. This result was somewhat surprising, given that SMTP and finger queries each take several seconds to complete. The main advantage of parallelism turned out to be decreasing the response time for the case where a machine being queried is unavailable, since it takes a fairly long time for connection attempts to timeout in the singly-threaded case. In the multi-threaded case, the connection timeout is not perceived by the user.

Since parallelism was primarily useful for increasing fault tolerance, we observed that a small number of threads would likely achieve the desired effect. We found that reducing the number of concurrent threads from 30 to 10 increased the response time by an imperceptibly small amount, but reduced Internet loading substantially. In contrast, in our Internet growth/disconnection study we used 20 concurrent threads, because we found that timeouts occurred in the majority of name lookup and connection requests. This high rate of timeouts derived from the fact that we were attempting to connect to a ports at each domain in the Internet, rather than attempting to query a particular domain more heavily (as in the case of netfind).

Clearly, the amount of concurrency one uses will be dictated by the ratio of active network initiation time to blocked thread time. While one could construct a mathematical model to optimize the selection of concurrency level, we found that in practice it is quite easy to set the level appropriately. One need only run a small version of the application (e.g., contacting a subset of sites for our Internet growth/disconnection study), and observe the amount of concurrent Internet activity that results. We did this by monitoring packet-level traffic into and out of the machine where the experiments were being run.

Of course, one must be careful to control the spread of distributed operations when using parallel probabilistic access. We consider this issue in Section 3.5.

Minimize Points of Failure by Coalescing and Localizing Slowly Changing Information

Another way to increase the fault tolerance of a wide area distributed application is to decrease its dependence on remote information sources. This can be achieved by coalescing slowly changing application information, so that much of it can be replicated and therefore available locally. In the case of netfind, for example, the entire seed database requires only 1.4 megabytes of disk space (plus another 2.2 megabytes for the inverted index), and this database is used only for slowly changing hints. Because of these characteristics, it is quite feasible to replicate the seed database at all client sites.

By localizing the seed database, netfind determines a good deal of information about the region of machine names to query in response to a search request (i.e., the position in the domain tree plus a number of hosts at that domain) before accessing any remote information sources. Therefore, unlike hierarchical white pages services (like X.500), netfind does not depend on the proper functioning of a sequence of white pages servers. Of course, it still relies on correct service from the underlying hierarchies of domain servers and gateways, but we do not add to this complexity with an additional hierarchy of higher level servers.

This technique is also used in the Domain Naming System, since each domain server caches the location of the (small number of) root servers. That use of the technique is less significant, since the DNS only localizes the information for one node in the tree.

Besides the improvement in fault tolerance attained through this technique, the application may also exhibit better performance, since it does not consult remote servers, and it avoids incurring network overhead. The challenge in applying this technique is to achieve a balance between the scalability problems of maintaining too much information locally vs. the fault tolerance problems of relying on too much remote information.

Another example of the use of this technique is the Archie system, developed at McGill University to support resource discovery among anonymous FTP sites [Emtage 1991]. Archie maintains a list of anonymous FTP sites, and periodically connects to and retrieves a directory listing of each site. It then makes this information available to Internet users via a telnet-based interface. Because all of the information is available in a central location, it can be searched with relatively high availability, compared with contacting all sites being searched. Clearly, it is also more efficient to search information at a centralized location.

Collect Information in by Making Multiple Passes Over a Period of Time

As the number of different sites where information to be gathered by a distributed application increases, the probability that all needed information will be simultaneously available decreases. The typical approach to this problem is to provide redundant information sources. For example, one requirement of participating in the Domain Naming System is that a domain's information be provided at two different (preferably geographically distributed) sites. In practice, many sites replicate more heavily than this, particularly at higher levels in the tree (at the root, "edu" domain, "com" domain, etc.) Nonetheless, at times information about a particular domain is sometimes unavailable. Moreover, arranging for this type of replication requires a large amount of global agreement, which is difficult to provide for many different applications.

In applications that are not time critical, an alternative approach is to gather the needed information over time, in a series of passes. We use this technique in our Internet growth/disconnection study. If more than three timeouts occur while attempting connections to a particular domain, that domain is marked as temporarily unreachable, and is not probed further during the current measurement cycle. A few days later the measurements are attempted again at the sites where timeouts occurred. The measurements are repeated until the results of one measurement cycle do not differ from those in the previous cycle, indicating that all sites that have not been reached are unreachable for a relatively long period of time, and hence are probably not reachable at all for the present. A similar technique was used by Lottor in his collection of Internet host names [Lottor 1990]. Lottor's program traversed the Domain Naming System tree recursively, using zone transfers [Mockapetris 1987] to collect data about as many zones as possible on the Internet. To collect the data, he ran the program several times over a period of days.

This technique is also used in our electronic mail study. Rather than attempting to collect a list of all mail edges seen at an institution at once, the edges were collected over a period of two months. This allowed the data to contain a representative sample of peoples' electronic mail communication patterns. We found that after a few days of data collection, the number of new unique edges collected per day decreased to approximately 10% of the initial days' collections.

3.2. Accommodating Administrative Decentralization

Locally distributed applications typically assume that a site administrator can simply install a server on a set of machines, with the servers communicating using some specific network protocol and information format. In contrast, in a wide area distributed environment there is no single source of authority, and it may be difficult to reach a level of agreement sufficient to deploy a wide area distributed application in this fashion. In this section we consider techniques that reduce the required level of agreement.

Accommodate Heterogeneity by Supporting Independent Protocol Subsets

As a prelude to the techniques we discuss in this subsection, we begin with a brief discussion of the X.500 directory service, as an example of a wide area distributed application that requires a substantial level of agreement. X.500 is actually a conglomerate specification, with components defining the model (X.501), authentication framework (X.509), abstract service definition (X.511), procedures for distributed operation (X.518), protocol (X.519), selected attribute types (X.520), and selected object classes (X.521). In order to deploy a server that exports directory information, a system administrator must (a) agree with the architectural principles that went into forming the standard, (b) provide the administrative and computational resources to support the large body of software that is needed to implement the standard [Rose & Schoffstall 1989], and (c) decide what to do about problems that have not yet been resolved in the standard [Kemezis 1987].

Netfind takes a more loosely coupled approach. Instead of requiring a coordinated set of servers that each support a highly structured body of information, netfind can often provide directory service even if a remote site does not support all of the protocols it uses, or if some steps in the protocol sequence fail. For example, if remote finger service is not supported, mail forwarding information may sometimes still be found. Or, if no mail forwarding information is found, netfind attempts to finger some of the machines matched from the seed database. Similarly, netfind can proceed without information about authoritative name servers. Netfind's tolerance of partial remote protocol support allows it to locate information about a large proportion of Internet users. This scale is significantly larger than that achieved by other existing Internet directory services, including the current X.500 pilot prototype.

We used a variant of this technique in two other prototypes. Our network visualization project uses approximately twenty different protocols and information sources, cross-correlating the information when necessary to determine important characteristics, such as the location of gateways. A result of using this more extensive set of sources is that we found there are situations in which one can detect inconsistencies between the information provided by the various sources, such as different subnet masks on individual nodes within a particular IP subnet. Flagging these inconsistencies is useful to network administrators. This experience indicates that while the implementation of the protocols should be independent (so that one protocol can be used in the absence of other related protocols), it can be useful if the information provided by the protocols is overlapping.

In a slightly different character, our Internet resource mapping/discovery project uses different sources of information of different degrees, of varying degrees of quality and structure. This design is an explicit acknowledgement of the fact that it is easier to obtain unstructured information with low quality guarantees (such as resource availability information gleaned from public bulletin board announcements) but harder to make use of such information; and similarly that it is harder to obtain information that is more readily usable (such as a structured list of resource descriptions and locations).

This technique of supporting independent protocol subsets is similar to using multiple independent implementations of a function to increase reliability in the presence of unexpected situations [Siewiorek & Swarz 1982]. It also bears some relation to systems in which a client and server negotiate at run time to determine their communication protocol [Bershad et al. 1987, Postel & Reynolds 1983]. However, run time negotiation requires agreement on a negotiation protocol, which is not needed if one simply tries multiple independent protocols. Moreover, using independent protocols accommodates the case where multiple servers must be contacted, each of which supports a different protocol, but some of which do not support a negotiation mechanism. Moreover, avoiding a negotiation step may simplify the computation and therefore increase its robustness.

In general, this independent protocol subset technique is a powerful means of extending the "reach" of applications whose usefulness is proportional to the number of sites on which servers are running. For example, this technique could help extend the reach of global electronic mail systems. Mail agents capable of handline heterogeneous mail addressing syntax already exist [Allman 1985], as do gateways that can transfer mail between different subsystems [Kille 1986]. Yet, our electronic mail study indicated that a large amount of global processing effort is being expended trying multiple times to deliver messages that cannot be delivered because of problems with transmitting mail between heterogeneous subsystems [Schwartz & Wood 1991]. Building more flexible mail interconnection systems could reduce this problem.

Do Not Rely on Exact Behavior From Independent System Components

As a system becomes increasingly large and administratively decentralized, there will often be independently implemented components intended to provide the same function (e.g., NFS servers implemented by different vendors). Even when a standard exists, there are often differences in the exported interfaces, either due to implementation errors or to different interpretations applied to situations that were not clearly specified in the standard. A useful technique to apply in this situation is to avoid assuming exact behavior from independent system components.

As an example, the specification for finger and SMTP both leave open the precise format for responses to queries. In some cases we tried to make netfind parse the results of these queries (e.g., for finding other machines to search by fingering a particular machine without specifying a user name). In these cases we found it necessary to add code to handle several common response formats. We did this only when the responses were to be used as hints, so that misinterpreted responses would lead only to performance degradation, rather than to incorrect results. We avoided parsing the responses from queries in other situations (such as the results of fingering a user at a machine), to render the tool flexible to as many finger response formats as possible.

This technique assumes increased importance in an environment that contains many subsystems that need to remain operational in the presence of partial failures. For instance, the AT&T network crash of January 1990 was caused by an error that spread among network switches that were supposed to be autonomous, but which in fact assumed that other switches would only send well formed inputs. In response to a certain type of message, each switch entered an incorrect state and further distributed the message, effectively disabling a large part of the network for nine hours [Fitzgerald 1990].

One negative aspect of not having exact behavior in independent system components is that the system can fail in more complex or confusing ways. In particular, a system that uses this technique may fail to return an answer, yet this failure may not imply a negative answer. For example, because the response format from finger and SMTP can vary, in some situations netfind cannot recognize and parse their outputs. In such a case, netfind will not report a successful response. Hence, while accommodating variation in independent system components makes a system more flexible, there can be cases where some variations are not recognized. If the output of the computation is intended for a human, the results can be confusing. If the output is intended for automated processing, an error could result. These problems seem to be a natural consequence of dealing with large, complex systems whose subcomponents have not yet achieved mature, standardized implementations.

On a related note, introducing more complex failure modes into an existing subsystem can make application level behavior more confusing. For example, when the Domain Naming System was introduced, some applications had confusing failures, because the name lookup process introduced a more complex set of failure modes than had previously existed. Until that time, a name lookup failure simply meant that the name was invalid. With Domain, a failure could also happen because of problems in the name lookup mechanism. Applications thus needed to be changed to handle the additional failure modes.

As a somewhat different example of accommodating inexact system behavior, in collecting data for our electronic mail study it was important to allow a range of mail addressing formats, since the To/From lines we collected were generated by many different types of systems in many different administrative domains. The first step before the data could be analyzed was therefore to build software that could transform mail names in a variety of different formats into a single canonical format. It took several iterations of parsing data, inspecting the results, and modifying the software, before all recognizable formats could be parsed correctly. Similarly, making netfind parse various formats correctly took several iterations, as we continued to discover new formats. These experiences underscore the complexity that can arise when many administratively decentralized, heterogeneous systems are connected together.

Use Information in its Naturally Decentralized State

While it is often easier to build applications that use centralized or moderately decentralized pools of information, doing so has its costs if information is naturally highly decentralized. These costs include keeping the data consistent with the "native" data; maintaining the completeness of the database (for example by a manual administrative procedure that periodically checks for missing entries); and the problem that authority over the correctness and completeness of the data is transferred away from the native data.

As an example, consider the mechanism used by X.500 to support a directory. In X.500, each participating site runs a server that maintains directory information about that site. It is the responsibility of each site administrator or end user to keep the information up to date. Hence, while the overall directory is decentralized in the sense that the operation of individual sites is decoupled, the information about users is centralized into a pool per site.

If an individual invocation of an application does not require a large amount of information or remote processing, it may be possible to use more decentralized information. For example, netfind treats each user's information as a separate, decentralized entity, locating information about users on their workstations. Because of this, netfind can often locate more timely information about users than the current X.500 pilot (since X.500 depends upon sites registering users), and can locate more users as well. With netfind, users and site administrators need neither register information with an auxiliary database, nor remember to update this database when the information changes.

On a related note, the initial implementation of netfind used a seed database that saved user names and electronic mail addresses, in addition to the host names and organization names the current version saves. This information was then presented to the user in the case where the person being sought had posted a USENET message at some time in the past. We eventually removed this mechanism, partly to reduce the seed database size. More importantly, netfind now treats the seed database simply as hint information, and accesses user information from its naturally decentralized location.

Using information where it naturally resides is a technique carried forward from our earlier Heterogeneous Name Service work. However, netfind uses much more decentralized information than the HNS did. The HNS was essentially a framework for supporting users in specifying the semantic operations needed to incorporate new auxiliary database-style name services into a global name service. Moreover, the HNS was used for mapping named objects to data about those objects (such as the network address of a host), rather than for discovering resources.

Broaden Perspective by Decentralizing the Effort to Build Information Components

Related to using information where it naturally resides is the issue of how information components that comprise an application are compiled. In the case of a directory service, for example, one choice is to build a directory of resources through the use of a central curator who controls the information quality, and imposes a uniform organization on the information. The disadvantage of this mechanism is that the organization will reflect a single person's perspective. Alternatively, one could use contributions from a range of decentralized individuals. Doing so will increase the breadth of perspective of the information, but could lower the information quality, or muddle the clarity of organization. However, if the information is of sufficiently focused context (see Section 3.4) and is used only as a hint, breadth can be increased appreciably without an important sacrifice in information quality.

As an example, the netfind seed database is built through decentralized contributions in a narrow context: it contains {institution name, host name} pairs generated at sites where people post messages. Many different institutional keywords will lead to the same seed database records and Domain information, which usually makes it quite easy to guess keywords that will succeed for any particular search. For example, to search for someone at the Massachusetts Institute of Technology, a user could specify the institution name using "mit" or "massachusetts institute of technology"; or they could specify the location, using "ma", "mass", "massachusetts", "cambridge", "boston", or "lexington"; or they could specify a particular laboratory, using "lcs", "laboratory for computer science", "lab for computer science", "ai", "artificial intelligence lab", "artificial intelligence laboratory", "lincoln laboratory", "media lab", and many other laboratories; or they may specify machine names, such as "allspice", "theory" or "expo". In each case above, a subset or combination of keys could be specified as well (e.g., "mit media").

On the other hand, the seed database contains some spurious information because of the decentralized nature of contributions. For example, the machine isis.berkeley.edu shows up under the domain "stanford" because a user had posted a message with the organization line "working at stanford this week". Nonetheless, it is quite easy for users to ignore the small number of spurious domains in the list, and make effective use of the tool. Moreover, since the information from the seed database is used only as hints to locate people, this technique will simply increase network load by a small amount if an inappropriate domain is used from the seed database.

As a different example of the use of the technique of broadening perspective by decentralizing information building, our electronic mail study constructed communication graphs from the union of mail logs from each of 15 different sites distributed around the world. Each site logged traffic to, from, or passing through machines at that site. Frequently, mail to or from a particular individual would be logged at several sites. In that case, each site contributed to part of the profile of that user's interests. The combination of 15 sites' data produced a surprising amount of information about individuals, which can be used by our algorithm to determine how users cluster by shared interests. Running this algorithm on any particular site's data alone would produce less significant results, because of the relatively narrow observational perspective that would provide.

3.3. Scalability of Operational Semantics

A common means of enhancing scalability is to relax the semantics of operations. For example, the Grapevine name service used so-called *convergent consistency* semantics, which permitted replicated updates to be applied non-atomically [Birrell et al. 1982]. Of course, relaxing semantics may cause users to see unexpected behavior, such as mail messages not being delivered to some recipients of a recently updated mailing list [Schroeder, Birrell & Needham 1984].

This design tradeoff acknowledges the fact that it is difficult to provide guaranteed consistency/transparency semantics in a scalable way. In the current section we argue further that providing guaranteed semantics of this nature is actually the wrong approach in a large environment. David Clark of MIT made a related point when he argued against the use of transparency for large scale systems. He pointed out that it is difficult to mask failures in such systems, and that such failures may surprise users who expect full transparency [Clark 1988].

This argument becomes more compelling as the environment becomes larger, more administratively decentralized, and more dynamically changing. A wide area distributed application in such an environment may span many network links and intermediary gateways, operating under the control of a number of different administrative policies, grades of service, and support for various protocols. This type of environment offers a complex range of failure modes, from hosts and links being unavailable to denial of service at various points throughout the network.

The traditional approach to this problem is to agree on a set of global standards, and to specify a controlled replication scheme. However, systems that require all components to fit into such an agreed-upon scheme may fail in the presence of components that do not have a complete and correct protocol implementation.

Use Approximate Solutions With User Discernible Failure Symptoms

Rather than trying to provide guaranteed semantics, we believe it is more appropriate to use approximate solutions, with two characteristics. First, the user should be made explicitly aware that operations are not guaranteed to work. Second, it should be clear when an operation fails. Providing a usable application then becomes a matter of making operations succeed most of the time and making the failures discernible, rather than handling all possible failure modes.

As an example, netfind does not guarantee that it can find any user's electronic mail address, or even that it can find any user who is on the Internet. Rather, it simply promises to make a "best attempt" search. Depending on the search specified by the user, the information in the seed database, the information available through the various protocols used for searches, and failures that may occur in the Internet, netfind may fail to resolve some searches properly. Yet, it is able to succeed in a large number of cases, and most search attempts that succeed can be reproduced. Furthermore, when a search fails, netfind attempts to clarify whether the failure reflects a transient host or network failure, as opposed to the search not being able to locate the individual for other reasons (e.g., because the individual is at an institution that does not permit inter-domain SMTP or finger queries).

One place where netfind originally did not make this distinction clear was when a user attempted to search for someone whose domain is in the seed database, but that domain is not on the Internet (for example, a small company that has mail forwarded via dialup links to an Internet site). We later modified netfind to determine when this is the case, and inform the user when it happens.

In contrast to the globally specified server organization and replication mechanisms used by X.500, netfind relies on its ability to extract organizational information from the seed database and remote protocols, and on the use of multiple concurrent remote queries to provide fault tolerance. These mechanisms provide only approximate functionality, yet provide more reliable behavior than X.500 currently provides. And because users are explicitly aware that searches may fail, they are prepared to take other steps if a search does fail (e.g., using different search keys, or trying the search later). In contrast, because X.500 attempts to provide guaranteed success semantics, it has been our experience that users encountering failures give up on their searches.

3.4. Organizing an Information Space Understandably

As the space encompassed by a distributed application grows, it becomes increasingly important to support operations without forcing users to understand the full organization of the space. Doing so can improve ease of use, and can also allow the information space to be reorganized and searched more flexibly.

Database designers have recognized the importance of shielding users from organizational complexity for some time, and have successfully addressed the problem for certain types of applications by separating logical from physical organization. Unfortunately, it may be infeasible to use a database system for supporting resource discovery in a large scale, administratively decentralized environment. First, in many cases information about resources may already exist in some form. Often, this information is in a simple format (e.g., user records in the UNIX[3] /etc/passwd file), and automatically converting this information to a highly structured format is difficult. Second, even in cases where one can convert to a structured format, it may be difficult to reach agreement across administrative boundaries about the proper structure of the data. Third, to date database systems have focused primarily on problems of scale in data volume, rather than in scope of decentralization. A different set of techniques is required to support a wide area distributed database than a database that holds a large volume of information in a centralized environment. Finally, database systems typically assume strong consistency semantics. As discussed in Section 3.3, these semantics may be stronger (and more expensive) than are needed in many distributed applications.

Because of these difficulties, distributed systems designers typically do not use database systems to support information needs for systems facilities, such as name and file services. To date these systems have all achieved scalability through the use of hierarchy [Lampson 1987]. However, the use of hierarchy has problems with supporting meaningful and flexible organization.

Use Hierarchy for Delegation, Augmented with Flexible Contextual Grouping Mechanisms

As a hierarchy grows to contain an increasingly wide variety of information, maintaining a reasonable organizational structure and searching through the hierarchy become difficult, because users must understand how the (increasingly deeply) nested components are arranged. Moreover, a hierarchically organized information space tends to become convoluted and inconsistent as new types of information are encoded into the hierarchy. For example, the UNIX file name */users/faculty/schwartz/pdp/monte/asynch/init.o* contains (from left to right) information about the file's disk location, creator's role, creator, research project, research subproject, algorithm variant, contents (*init* = "initialization routines"), and file type (*.o* = "object code").[4] Reorganizing such a hierarchy is time consuming and not easily accommodated, once a user base has been established. Finally, a hierarchy is inflexible. For example, in searching for people having technical expertise in three dimensional graphics algorithms, one person might prefer the information space to be organized as "/Computers/Graphics/3D/Experts", while another might prefer an organization like "/People/Interests/Technical/Graphics/3D".

---

[3] UNIX is a trademark of AT&T Bell Laboratories.

[4] This example is a modified version of one given in [Greenspan & Smolensky 1983].

We do not mean to imply that hierarchies are bad. Rather, we believe that hierarchies are appropriate for a certain class of problems, but that they do not adequately support resource discovery. We observe that the advantages of hierarchical organization derive essentially from two characteristics. First, hierarchy supports scalability, by allowing processing and storage to be delegated in arbitrarily many parts of the tree. Second, hierarchy supports organization, because it allows related objects to be grouped together. A position in a hierarchy implies some context, and hence a full hierarchical name has more meaning than an end component name. For example, source files may be grouped together into one directory to reflect the fact that they all form a single program when compiled together. The directory name adds meaning to the individual file names.

We observe that the scalability of a hierarchy derives from its support for delegation, while its organizational inflexibility derives from its limited means for contextual grouping. We now consider ways of avoiding the disadvantages of a hierarchy by augmenting it with other techniques for contextual grouping.

Use Contextual Focus to Support Flat Rather than Level-by-Level Accesses of Structured Information

A hierarchical structure is often used for abstraction in user interfaces (e.g., presenting the abstraction of folders containing files). While such abstraction is often helpful to the user who is not familiar with the possible choices in a system, it is often an impediment to more experienced users. Experienced users usually prefer to directly select operations and resources that they know exist, rather than searching for them in a level-by-level fashion.

One way to allow experienced users to select operations directly is through the use of search paths, as used by the UNIX "csh" shell for naming executables, libraries, directories to move to, etc. Search paths work well if the set of places where needed resources exist is small and known in advance. However, in resource discovery and many other types of applications these restrictions do not apply, and other techniques are needed.

To avoid imposing a level-by-level structure on accesses, one can use information retrieval techniques to support flat searches of an information space, even if the space is large and structured. The challenge is to avoid the problems that information retrieval systems typically suffer. When searching for a particular type of resource (such as books about some topic), users of information retrieval systems typically find that their searches either match many unwanted resources (because of lack of *precision* of the specified keyword combination) or miss wanted resources (because of lack of *recall* of the specified keyword combination) [Salton 1986]. Choosing appropriate keyword combinations becomes increasingly difficult as the size of the information space increases.

These problems arise because the context of searches in information retrieval systems is typically very broad. For example, in response to a search on the keyword "window", a broad context information retrieval system might respond with some information related to houses, and other information related to computer graphics systems.

To avoid these problems, the context should somehow be narrowed. As an example, rather than forcing users to manually traverse the resource space (which in the case of Internet users is the Domain Naming hierarchy), netfind allows users to specify any of a number of different sets of keys to describe the institution where a user works. This information is then used to select parts of the Domain Naming hierarchy that might be relevant to search. The user is then presented with a brief list of these domains, and asked to prune the scope of the search to a small number of sites. Therefore, the context of searches is very narrow: the user must only select terms that specify the name of an institution, its location, or some other attribute (e.g., that it is an educational institution). This context augments the hierarchical structure of the Domain Naming System, so that a level-by-level limitation is not imposed on what parts of the space the user may specify.

If the scope of a resource discovery application is naturally focused, the context may simply be implicit. This is the case in netfind, where the resources being searched for are users at institutions on the Internet. However, for a more general resource discovery application (such as our Internet resource mapping/discovery project), the context must be set in some explicit manner. For this purpose, a promising technique is the use of private views of the global space, to allow users to superimpose added organization on an information space. This is the intent of the mechanism we implemented for our Internet resource mapping/discovery project. As an example, a person interested in graphics might build a view that organized the world according to PostScript, Tools, Window Systems, Images, and Discussions, with pointers from each of these categories to directories on archive sites around the Internet. Private naming spaces are the basis of a number of other systems as well, such as Tilde [Comer & Murtaugh 1986], Prospero [Neuman 1989], and Plan 9 [Pike et al. 1990].

In addition to providing a conceptually unified space within which to search for resources, context can also provide a means to infer semantics that can limit the scope of distributed operations. For example, because searches take place in the context of Internet mail naming, netfind can make use of Domain and SMTP information to prune the set of machines queried, by using SMTP to request mail forwarding information about a user on an authoritative name server for each domain, located by contacting the Domain Naming System. In this fashion, the set of machines to search is reduced by one to two orders of magnitude in many cases.

Retain Context of Underlying Structured Space in the Results of Flat Searches

Another problem with information retrieval techniques as they have typically been applied is that they require the user to specify precise keys in order to avoid receiving back a flood of responses. If instead the information is categorized back into the structure of the resource space, it may be possible to abstract the information far enough that the user can select among a relatively short list. In the case of netfind, the naming domains within which matching host names fall are used to provide the user with a brief list to prune. A list of 20 domains can represent hundreds of machines, allowing a user to prune the scope of the search easily. Information retrieval systems typically cannot support this technique, because the underlying space is unstructured. The only way to view information is via flat, key-based searches.

One problem with netfind's use of this recategorization technique occurs in conjunction with large domains. If the user specifies institution keys that match hosts in more than 36 domains,[5] he/she is asked to reformulate a more precise search. The original motivation for this restriction was to prevent users from specifying unreasonably broad searches (e.g., using only the key "university"). However, there are some institutions that use so many subdomains that this technique does not work well. For example, if the user tries to search for someone at Carnegie Mellon University using the current seed database, 58 different domains are matched. A stop-gap solution would be to increase the maximum domain limit, but that does not really solve the conceptual problem. A better solution would be to present the user with a one level abstracted list of matching domains, and then let the user prune the list in stages. For example, abstracting the bottom level domains from the 58 CMU domains would yield the second-level domain list consisting of cc.cmu.edu, cmu.edu, cs.cmu.edu, ece.cmu.edu, ri.cmu.edu, sei.cmu.edu, and unige.ch.[6] The user could then select cs.cmu.edu and cc.cmu.edu, and be presented with a list of subdomains under those domains. We are currently modifying netfind to do this.

Another reason to retain the context of an underlying structured space in the results of flat searches is so that users may avail themselves of the structure in progressing to more detailed searches. For example, when using a flat search mechanism to locate files in a large directory tree, it is often useful to move to the directory where a file has been found, and examine what other files are in that directory. In contrast, there is no underlying structure in many information retrieval systems.

3.5. Controlling the Spread of Distributed Operations

Distributed applications intended for use in wide area networks present the possibility of spawning processes at many nodes distributed around the network. While doing so has powerful potential for supporting loosely coupled cooperative applications [Kahn & Cerf 1988, Schwartz 1991c], it poses some dangers as well. First, there is the danger that such computations may spread uncontrollably, as in the case of the Internet Worm of November 1988 [Spafford 1989], the error that lead to the AT&T network outage of January 1990 [Fitzgerald 1990], and a number of other situations characterized by the Automatic Generation of Messages [Manber 1990]. Second, even if such computations do not spread in an uncontrolled fashion, they may potentially generate a large amount of load on both the network and the remote machines, without the user's being aware that this is happening. This problem is particularly important when users must pay for access to the network and remote resources, and when the computation may span a wide area network.

Because of these problems and the increasing attention being given to security risks [National Research Council 1991], using worm-like techniques in distributed applications has acquired a negative image. It is our perspective that the concept of a computational worm is inherently neither good nor bad; it is simply powerful. Indeed, such computations originated in the research community [Shoch & Hupp 1982].

---

[5] The number 36 comes from the use of the 10 numbers and 26 letters for specifying domains to search.

[6] The existence of unige.ch in this list happens because there is a domain called cmu.unige.ch, a point we consider in Section 3.2.

In this section we outline some techniques that, while simple, can reduce a number of the problems of uncontrolled spread of distributed operations. We offer these techniques simply as "hints" for addressing the problems. Because of the dangers of releasing an errant worm computation on a wide area network, we suggest as an open question the issue of whether a reasonable framework may be developed within which such computations can be confidently implemented and widely deployed.

We note that our techniques differ from those developed by Manber for limiting "chain reactions" in networks [Manber 1990]. Manber recommends that all messages be given identifiers, that messages generated by other messages retain the identifiers of the originating messages, and that hosts maintain a record of the outstanding messages by identifier, and refuse to generate more messages automatically in response to a message that has already been seen. While this system of limitations enforces a provable upper bound on the amount of messages that are generated, that upper bound is quite high for networks with a large number of links. Moreover, it requires a level of global cooperation in message identification and recording that may be infeasible in wide area distributed applications.

Impose Redundant Limits

A general problem with complex systems is that it is difficult to predict the range of situations to which a system will be subjected at run time. This problem is compounded by scale and heterogeneity, since the developer usually cannot test the application in all the environments within which it will eventually execute. A simple means of reducing the risk of uncontrolled spread of a worm-like distributed computation in this situation is to impose a number of redundant limits on the computation. For example, in addition to limiting in the number of hosts that will be searched, netfind limits the number of domains to be searched and the number of levels of indirection to use when acquiring new host names to search. In addition, netfind incorporates a restriction that ensures that the same host is never searched more than once. Because of these restrictions, even a user who leaves a search running in the background will never cause an unlimited amount of network activity. Indeed, measurements indicate that the most expensive search is well within an order of magnitude of the cost of the average search (700 packets vs. 137 packets, respectively). Of course, additional limits could be imposed as well, most notably a time limit on the total length of a search. In practice this turns out not to be important, since our measurements indicate that 90% of the time netfind is used interactively, with users interrupting it if no information is located within a few minutes.

As a second example of the use of this technique, in our Internet growth/disconnection study we keep track of the sites that have been probed, the machines that have been probed, the number of timeouts that have occurred at each domain being probed, and the number of connection refusals for each port at each domain being probed. The program monitors each of these variables, and reduces its probing in various ways in response to passing threshold values for each variable. The mechanisms used in this study are discussed in more detail later in this section.

A somewhat different version of this technique can be used when first deploying a wide area distributed applications, by monitoring auxiliary activity associated with the application. For example, for the first several hours of running netfind and our Internet growth/disconnection study, we used network tools to monitor the packets going to and from the application, as well as the currently open TCP connections. These tools provided a source of redundant verification that neither application was creating unduly high concurrent Internet activity.

Impose Position Independent Limits

Netfind’s task of limiting the uncontrolled spread of distributed operations is simplified by the fact that each invocation of the program has a centralized point of control. Another technique can be introduced in systems for which this is not the case. This technique is to impose limits that are detectable as having been exceeded from any position in the network, and which can be enforced even if some nodes do not function properly in limiting the spread of the computation. For example, rather than relying on the "firewall" effects of network gateways on limiting the range beyond which an application should not spread (as Morris was said to have done in his Internet Worm), the application could check at each step that it is executing on a node within a specified set of allowable networks. Even if particular nodes incorrectly identify the networks on which they are located, the computation will avoid spreading through nodes that do correctly identify themselves, and that fall beyond the specified limits. We also use this technique for limiting the range of probes in our network visualization project.

A somewhat different form of this technique was used in our electronic mail study. In that case we made the scripts that ran on each of the remote sites impose an upper bound on the amount of data that would be transmitted from that site per day. This was needed because some sites had large mail logs (e.g., one commercial site generated up to 20 megabytes of log data per day). By imposing this simple position independent limit, we limited the effect our measurement study had on the Internet.

Limit the Load a User Can Inadvertently Generate

As mentioned at the beginning of this section, wide area distributed applications may potentially generate a large amount of load on both the network and the remote sites without the user's being aware that this is happening. To reduce this problem, it is worthwhile to limit the load a user can inadvertently generate during a request, and to make expensive operations "feel" expensive, by making them somewhat more difficult to specify. As an example, the initial version of netfind simply initiated parallel searches of each of the hosts matched in the seed database. The expense of a search therefore depended on users' being careful to specify searches using keys that would only match a small number of sites. For example, rather than searching for "schwartz colorado", a more efficient search was specified by "schwartz university colorado boulder". The former would search machines from many different sites around Colorado, while the latter would search only a few machines. Yet, users are not likely to use the more verbose search specification if the shorter form seems to work as well, since the search mechanism is not evident (unless trace output is enabled). We therefore added a mechanism to force users to select at most three naming domains to search. Additionally, we added a mechanism so that if a search was to progress to a more expensive stage, the user would be consulted about whether to do so. For example, if no information is found by probing machines located by the Domain Naming System, the user is asked whether to proceed to the next stage, when individual machines matched in the seed database are queried. These changes made the expense of searches more closely match the perceived ease of specifying a search.

This principle also underlies the way the U.S. telephone system forces users to prepend long distance calls within their same area code with "1", so that they realize they are making a toll call.

A related point is in order here. If a user specifies a limit on the scope of a distributed operation, that limit should be applied when distributing the operation, rather than letting the application run in an unlimited manner, and then locally "pruning" the results that are returned. While this point may seem obvious, there are a number of directory services that limit scope in this "post pruning" manner, leading users to believe that the costs of operations are more limited than they are.

Optimize Algorithms to Limit Remote Usage at the Expense of Local Resource Usage

Building software often requires making resource usage trade offs, the most common being time vs. space. In building wide area distributed applications, it may be important to optimize use of remote resources. For example, because our Internet growth/disconnection study accesses so many remote sites, we chose to make more intensive use of local memory and CPU resources by using data structures to keep track of information about things such as which hosts had already been probed, how many probes and timeouts had occurred for each port at each site, and name lookup counts. Minimizing usage of remote resources meant that the program used more memory and expended added CPU effort traversing these data structures. For example, during the course of a single run the program's virtual memory image can grow to 15-20 megabytes, while a simpler program could have been implemented to use only 1-3 megabytes. In return, on average the program performs only 3.9 name lookups per domain, and attempts only 5.7 connections per domain. In total, we estimate that this study increases global Internet load by only .5% on the one day per month that we run it.

Another example application of this technique came from our electronic mail study. In that case, we chose to transmit data from the monitored sites in a minimally processed format (removing log entries not concerned with electronic mail) to the central data collection host. The data was then processed on that single node. While processing the data remotely would have provided a substantial parallel speedup, we chose not to do so because it would have consumed too many remote resources, under circumstances where it was difficult to predict the level of contention over those remote resources.

Clearly, this technique would not be feasible for a wide area distributed application if the amount of local storage or processing grows quickly as a function of network size. For example, the data structures used in our Internet growth/disconnection study are linear in space and logarithmic in time, as a function of the number of sites (not hosts) in the Internet. If the time or space complexity were much larger, this technique could be less applicable.

Provide a Checkpoint/Resume Mechanism for Communication Intensive, Wide Area Distributed Operations

If an application generates a large amount of network traffic to a wide area distributed set of nodes, it may be prudent to checkpoint the state of the operation periodically in such a fashion that the application can later be restarted from the place it left off. Doing so can reduce wasted effort in the case of host or network failures, or in the case where an operation is incorrectly implemented. As an example, to measure the scope of the directory provided by netfind, we ran a measurement experiment in which we sought to execute the search protocol (Domain lookup, SMTP query, and finger query) on a small number of machines in each naming domain found in the seed database.[7] As initially implemented, this experiment would proceed properly for several thousand entries into the database, and then fail and crash due to a programming error. Because of the wide area distributed nature of this experiment, debugging the problem could only reasonably be done if the problem could be repeated without each time probing machines from the initial entries in the database. By implementing a checkpoint/resume mechanism, we were able to reproduce the problem while probing only a handful of domains that had not been successfully probed before. Note that all of the usual distributed debugging problems arise when debugging wide area distributed applications, but that the scale and scope of distribution of these applications demands that extra precautions be taken to limit the consequences of an error spread over the global Internet.

This technique proved to be especially important in later developing and debugging our Internet growth/disconnection measurements, because of the added complexity of that software. The measurement program is longer (approximately 2,000 lines of C code), uses several more data structures and algorithms for minimizing network and remote host load, runs for longer (10-12 hours), and attempts many more name lookups and connections. The most difficult problem we encountered in this program turned out to be caused by an error in the implementation of Sun Microsystem's Lightweight Processes package [Kepecs 1985], which causes signals to be lost during highly concurrent network activity. Locating this problem required a large amount of debugging effort, which would have been much less feasible in the context of the global Internet without the ability to checkpoint and restart the computation.

The ability to checkpoint and restart a computation can also be useful in the case of measurement studies where the information to be computed from a set of measured data is not known completely in advance. For example, as our Internet growth/disconnection study has progressed, we have discovered additional measurements of interest. Originally we planned to compute statistics about the number of sites that discontinued service accessibility for various services over time. As we have collected and examined the data, it has become clear that there are a number of discernibly different ways that sites choose to make services unreachable. Sites can discontinue running servers, install access control lists for particular services, or run gateways that allow communication between the Internet and the internal network only to certain services. Because the program that collects data uses a checkpoint/resume mechanism that logs the low level connection attempts and results (as opposed to just overall counts of these quantities), we are able to modify the program to discern between these mechanisms by parsing the checkpointed logs more specifically. Therefore, we were able to gather the needed information without repeating the connection requests. We have extended the set of computable information in various other ways several times without rerunning the connection measurements.

Even if all of the information to be retrieved resides on one remote node, the problem of wasted distributed effort can still be important. For example, if a network failure occurs during an FTP file transfer, the user is forced to restart the entire transfer, because FTP can only retrieve entire files. On the other hand, just because the operations performed by an application span a wide area network does not mean that a checkpoint/resume mechanism is needed. For example, netfind itself (as opposed to the netfind scope measurement experiment) does not use such a mechanism, even though there is an obvious place where it could do so. Sometimes a user initiates a search that does not locate the desired user, and then follows that search with a second search with a different set of keys. Unless the user consciously avoids choosing the same naming domains to search when asked to choose at most three domains, the second search may repeat some of the effort from the first search. This may particularly be true if the two searches are separated by time, or if two different users at a site initiate the same search. Installing a shared cache would prevent this situation. Yet, the load generated by each search is fairly small (about 137 packets on average), and hence a checkpoint/resume mechanism is not critical.

---

[7] This experiment was a precursor to our Internet growth/disconnection study.

An interesting interaction occurs between techniques we have described for fault tolerance and controlling the spread of distributed operations. Two of the primary techniques we used for achieving fault tolerance were the use of parallel probabilistic access, and repeating failed operations over a number of days, to overcome temporary failures. Because parallel probabilistic access in a wide area environment risks causing operations to spread far and quickly, we introduced a number of techniques to limit the spread of operations, one important technique being the use of a checkpoint/resume mechanism. These fault tolerance and distributed control mechanisms are at tension with each other because reloading a previously checkpointed state means that timeouts that had occurred are considered to be timeouts in the current run. If the current run is being done several days later than the run from which checkpointed information was loaded, it may make sense to retry sites where timeouts originally occurred. Because of this, we added an option at measurement startup time to specify whether or not the previous timeout state was to be loaded. In cases where a run was terminated prematurely, the timeout state was loaded. In runs initiated later to reattempt missed sites, the timeout state was not loaded.

3.6. User Interface Issues

In this section we consider user interface issues for wide area distributed applications.

Make an Appropriate Role Division Between User and System

Notwithstanding the many years of effort that have gone into building artificially intelligent systems, we believe that some tasks are inherently better suited to a user than to a system, or vice versa. Systems can easily maintain a large amount of state about a problem and access this state in complex ways, while users can easily use intuition to guide them through unfamiliar territory. Systems that divide their tasks appropriately between user and system may be more successful than, for example, systems that expect users to remember many details or perform repetitive tasks accurately, or systems that try to incorporate sophisticated inferencing abilities. Making systems that can do such inferencing will probably become increasingly difficult as the universe of network accessible resources becomes increasingly large, heterogeneous, and administratively decentralized.

Netfind is an example of an appropriate user/system role division. The system deals with the details of accessing the seed database and executing concurrent remote probing protocols, but makes almost no attempt to understand the meaning of the information it retrieves. The user, in contrast, is in charge of making basic decisions about which naming domains to search (based on an intuitive understanding of what the domains mean), and when to terminate a search (because it has either completed successfully, or seems unlikely to do so).

Provide Progress Indications for Operations with High Response Time Variance

Good response time is a key focus in many distributed systems and applications. Yet, it may not be possible to provide uniformly good response time for some wide area distributed applications, because such applications may experience many different network speeds, queueing delays, and failures, as well as large variations in the degree of distribution of the resources that must be accessed. Unfortunately, high response time variance is often even more annoying to users than slow average response time.

An effective way to mediate this problem is to provide some type of trace output, so that users may observe the progress of the application. For example, netfind supports a trace option that allows users to watch each thread execute Domain, SMTP, and finger lookups. Users report that this output is interesting, and makes the search time variance much less perceptible. Because of its popularity, we changed the default to be that this output was enabled.

4. Conclusions

In this paper we have presented a number of techniques for supporting distributed applications that span many nodes across administrative boundaries in wide area internets. Such applications present particular difficulties, because of the scale and administrative decentralization of the environments within which they run. Our approaches to these problems emphasize simplicity, for example preferring a solution that usually functions well over a solution that uses deterministic algorithms that require complex solutions to hide their failure modes.

The techniques we presented are based on experiences with a number of prototypes in the areas of heterogeneity, resource discovery, distributed collaboration, and network measurement projects. Nonetheless, the techniques are applicable to a broad range of wide area distributed applications. The techniques pertain to building and deploying software systems that are complex because they are large and decentralized, independent of what those systems actually do. Table 1 summarizes the limitations on applicability of the techniques, based on the discussions given in Section 3. An "X" in an entry means that the particular characteristic may affect the applicability of the given technique. For example, the "X" for Communication/Data Volume for the "use parallel probabilistic access" technique indicates that parallel probabilistic access is less feasible in situations where each access requires a large volume of data to be transmitted. As a second example, it may not be feasible to decentralize information building if the results of a computation are expected to meet certain correctness guarantees, because decentralizing may mean that no individual is accountable for the contents of a particular piece of information.

[Section] Technique	Technique Applicability Limits For Given Application Characteristics				
	Communication/ Data Volume	Remote Processing Volume	Correctness/ Failure Semantics	Machine Parsability of Results	Human Interaction
[3.1] Use Parallel Probabilistic Access	X	X	X		X
[3.1] Coalesce/Localize Information	X				
[3.1] Collect Information in Passes					X
[3.2] Support Independent Protocol Subsets					
[3.2] Accommodate Inexact Subsystem Behavior			X	X	X
[3.2] Use Naturally Decentralized Information	X	X			
[3.2] Decentralize Information Building			X	X	
[3.3] Use Approximate Solutions			X	X	X
[3.4] Augment Hierarchy with Context					
[3.4] Use Context to Support Flat Accesses					
[3.4] Retain Context in Results					
[3.5] Impose Redundant Limits					
[3.5] Impose Position Independent Limits					
[3.5] Limit Inadvertent User Load					X
[3.5] Trade Remote for Local Cost	X				
[3.5] Use Checkpoint/Resume Mechanism					
[3.6] Make Appropriate Role Division					X
[3.6] Provide Progress Indications					X

Table 1: Summary of Technique Applicability Issues

Our prototype experiences show that it is possible to build coherent applications in the face of many complexities introduced by wide area distributed environments. Particularly significant to such environments are obstacles to tight integration, such as the need to accommodate multiple administrative domains. Our approach to these problems is to use an extremely loose style of integration. Rather than depending on any single protocol, information source, system organization, or style of specifying operations, our approach is always to accommodate multiple mechanisms in such a fashion that little agreement is needed to provide a coherent application. While standards are helpful, we believe it is difficult to specify standards that are both globally adopted and technologically current. Loose integration eases the task of building applications that must function in a wide area distributed environment, and provides a means of deploying workable systems that can be improved and evolved as problems are uncovered from real usage experiences, rather than specified in full before any experiences are gained.

Acknowledgements

We would like to thank the students who have been involved with the Networked Resource Discovery Project at various points in time. We would also like to thank Andrzej Ehrenfeucht, David Goldstein, Goetz Graefe, Ed Lazowska, and David Wood, who provided feedback on an earlier draft of this paper.

This material is based upon work supported in part by NSF cooperative agreement DCR-8420944, and by a grant from AT&T Bell Laboratories.

5. Bibliography

[Allman 1985]
E. Allman. Sendmail - An Internetwork Mail Router. *UNIX Programmer's Manual, 4.2BSD*, 2C, Comput. Sci. Division, EECS, Univ. of California, Berkeley, June 1985.

[Bershad et al. 1987]
B. N. Bershad, D. T. Ching, E. D. Lazowska, J. Sanislo and M. Schwartz. A Remote Procedure Call Facility for Interconnecting Heterogeneous Computer Systems. *IEEE Trans. Software Eng.*, SE-13(8), pp. 880-894, Aug. 1987. To be reprinted in *Distributed Processing: Concepts and Structures*, A.L. Ananda and B. Srinivasan, Editors, IEEE Computer Society Press.

[Birrell et al. 1982]
A. D. Birrell, R. Levin, R. M. Needham and M. D. Schroeder. Grapevine: An Exercise in Distributed Computing. *Commun. ACM*, 25(4), pp. 260-274, Apr. 1982. Presented at the 8th ACM Symp. Operating Syst. Prin., Pacific Grove, CA, Dec. 1981.

[Black 1985] A. P. Black. Supporting Distributed Applications: Experience with Eden. *Proc. 10th ACM Symp. Operating Syst. Prin.*, pp. 181-193, Dec. 1985.

[CCITT 1988]
CCITT. The Directory, Part 1: Overview of Concepts, Models and Services. ISO DIS 9594-1, CCITT, Gloucester, England, Dec. 1988. Draft Recommendation X.500.

[Case et al. 1989]
J. Case, M. Fedor, M. Schoffstall and C. Davin. A Simple Network Management Protocol (SNMP). Req. For Com. 1098, Apr. 1989.

[Clark 1988] D. D. Clark. Personal Communication. Commentary concerning wide area network transparency, during ACM SIGOPS European Workshop on Autonomy vs. Interdependence, Cambridge, England, Sep. 1988.

[Comer & Murtaugh 1986]
D. Comer and T. P. Murtaugh. The Tilde File Naming Scheme. *Proc. 6th IEEE Int. Conf. Distrib. Comput. Syst.*, pp. 509-514, May 1986.

[Danzig et al. 1991]
P. B. Danzig, J. Ahn, J. Noll and K. Obraczka. Distributed Indexing: A Scalable Mechanism for Distributed Information Retrieval. Jan. 1991.

[Emtage 1991]
A. Emtage. Personal Communication. McGill Univ., Montreal, Canada, Mar. 1991. Electronic bulletin board posting on comp.archives about version 2.0 of Archie anonymous FTP site directory server.

[Evans et al. 1989]
N. H. Evans, D. A. Troll, M. H. Kibbey, T. J. Michalak and W. Y. Arms. The Vision of the Electronic Library. Mercury Technical Report Series, Number 1, CMU Academic Services, 1989.

[Fitzgerald 1990]
K. Fitzgerald. Vulnerability Exposed in AT&T's 9-Hour Glitch. *The Institute*, 14(3), pp. 1-6, IEEE, Mar. 1990.

[French, Jones & Pfaltz 1990]
J. C. French, A. K. Jones and J. L. Pfaltz. Scientific Database Management. Computer Science Report No. TR-90-21, Aug. 1990. Final Workshop Report.

[Gifford 1979]
D. K. Gifford. Weighted Voting for Replicated Data. *Proc. 7th ACM Symp. Operating Syst. Prin.*, Dec. 1979.

[Gifford et al. 1985]
D. K. Gifford, R. W. Baldwin, S. T. Berlin and J. M. Lucassen. An Architecture for Large Scale Information

Systems. *Proc. 10th ACM Symp. Operating Syst. Prin.*, pp. 161-170, Dec. 1985.

[Greenspan & Smolensky 1983]
S. Greenspan and P. Smolensky. DESCRIBE: Environments for Specifying Commands and Retrieving Information by Elaboration. In *User Centered System Design, Part II: Collected Papers from the UCSD HMI Project*, Institute for Cognitive Science, Univ. of California, San Diego, Dec. 1983.

[Harrenstien, Stahl & Feinler 1985]
K. Harrenstien, M. Stahl and E. Feinler. NICName/Whois. Req. For Com. 954, Oct. 1985.

[Kahle & Medlar 1991]
B. Kahle and A. Medlar. *An Information System for Corporate Users: Wide Area Information Servers.* Thinking Machines Corporation, Apr. 1991.

[Kahn & Cerf 1988]
R. E. Kahn and V. G. Cerf. *The Digital Library Project - Volume 1: The World of Knowbots.* Corp. for National Research Initiatives, Mar. 1988.

[Kemezis 1987]
P. Kemezis. Holes in X.500 Hinder Search for Global Electronic-Mail Directory. *Data Commun. Magazine*, 16(12), pp. 62-64, Nov. 1987.

[Kepecs 1985]
J. Kepecs. Lightweight Processes for UNIX Implementation and Applications. *Proc. USENIX Summer Conf.*, pp. 299-308, June 1985.

[Kille 1986] S. E. Kille. Mapping Between X.400 and Req. For Com. 822. Req. For Com. 987, Univ. College London, June 1986.

[Lampson 1983]
B. W. Lampson. Hints for Computer System Design. *Proc. 9th ACM Symp. Operating Syst. Prin.*, pp. 33-48, Oct. 1983.

[Lampson 1987]
B. Lampson. *A Naming Service for a World-Wide Computer Network.* Digital Equipment Corporation, Syst. Research Center, Nov. 1987. Videotape Lecture from the University Video Communications Distinguished Lecture Series on Industry Leaders in Computer Science.

[Lottor 1990]
M. Lottor. Personal Communication. Discussion of Internet measurements to count number of domains and machines in the Internet. Apr. 1990.

[Lynch 1990]
C. A. Lynch. Information Retrieval as a Network Application. *Library Hi Tech*, 32(4), pp. 57-72, 1990.

[Manber 1990]
U. Manber. Chain Reactions in Networks. *IEEE Computer Magazine*, 23(10), pp. 57-63, Oct. 1990.

[Menke et al. 1991]
W. Menke, P. Friberg, A. Lerner-Lam, D. Simpson, R. Bookbinder and G. Karner. A Voluntary, Public Method for Sharing Earth Science Data Over the Internet. Tech. Rep., Lamont-Doherty Geological Observatory, Columbia Univ., Mar. 1991.

[Mockapetris 1987]
P. Mockapetris. Domain Names - Concepts and Facilities. Req. For Com. 1034, USC Information Sci. Institute, Nov. 1987.

[NASA 1990]
NASA. *Workshop Record of the Second Annual CESDIS Workshop - The Role of Computer Science in the Mission to Planet Earth.* Center of Excellence in Space Data and Information Sciences, Oct. 1990.

[NASA 1991]
NASA. *Toward a National Software Exchange.* Research Institute for Advanced Computer Science, Apr. 1991. Invitational Workshop Announcement.

[National Research Council 1991]
National Research Council. *Computers at Risk: Safe Computing in the Information Age.* National Academy Press, Washington, D.C., 1991.

[Neuman 1989]
B. C. Neuman. The Virtual System Model for Large Distributed Operating Systems. Tech. Rep. 89-01-07, Comput. Sci. Dept., Univ. Washington, Seattle, WA, Apr. 1989.

[Notkin et al. 1988]
D. Notkin, A. P. Black, E. D. Lazowska, H. M. Levy, J. Sanislo and J. Zahorjan. Interconnecting Heterogeneous Computer Systems. *Commun. ACM*, 31(3), pp. 258-273, Mar. 1988.

[Oki & Liskov 1988]
B. M. Oki and B. H. Liskov. Viewstamped Replication: A New Primary Copy Method to Support Highly-Available Distributed Systems. *Proc. 7th ACM Symp. Principles Distr. Comput.*, Aug. 1988.

[Peterson 1988]
L. L. Peterson. The Profile Naming Service. *ACM Trans. Comput. Syst.*, 6(4), pp. 341-364, Nov. 1988.

[Pike et al. 1990]
R. Pike, D. Presotto, K. Thompson and H. Trickey. Plan 9 from Bell Labs. *Proc. Summer 1990 UKUUG Conf.*, pp. 1-9, London, July 1990.

[Plummer 1982]
D. C. Plummer. An Ethernet Address Resolution Protocol -- Or -- Converting Network Protocol Addresses to 48.bit Ethernet Address for Transmission on Ethernet Hardware. Req. For Com. 826, Nov. 1982.

[Postel 1981a]
J. Postel. Internet Control Message Protocol. Req. For Com. 792, USC Information Sci. Institute, Sep. 1981.

[Postel 1981b]
J. Postel. Internet Protocol - DARPA Internet Program Protocol Specification. Req. For Com. 791, USC Information Sci. Institute, Sep. 1981.

[Postel 1982]
J. B. Postel. Simple Mail Transfer Protocol. Req. For Com. 821, USC Information Sci. Institute, Aug. 1982.

[Postel & Reynolds 1983]
J. Postel and J. Reynolds. Telnet Protocol Specification. Req. For Com. 854, USC Information Sci. Institute, May 1983.

[Postel & Reynolds 1985]
J. Postel and J. Reynolds. File Transfer Protocol (FTP). Req. For Com. 959, USC Information Sci. Institute, Oct. 1985.

[Quarterman & Hoskins 1986]
J. S. Quarterman and J. C. Hoskins. Notable Computer Networks. *Commun. ACM*, 23(10), pp. 932-971, Oct. 1986.

[Quarterman 1990]
J. Quarterman. *The Matrix - Computer Networks and Conferencing Systems Worldwide.* Digital Press, 1990.

[Rose & Schoffstall 1989]
M. T. Rose and M. L. Schoffstall. An Introduction to a NYSERNet White Pages Pilot Project. Tech. Rep., NYSERNet Inc., Dec. 1989.

[Salton 1986]
G. Salton. Another Look at Automatic Text-Retrieval Systems. *Commun. ACM*, 29(7), pp. 648-656, July 1986.

[Schroeder, Birrell & Needham 1984]
M. D. Schroeder, A. D. Birrell and R. M. Needham. Experience with Grapevine: The Growth of a Distributed System. *ACM Trans. Comput. Syst.*, 2(1), pp. 3-23, Feb. 1984.

[Schwartz, Zahorjan & Notkin 1987]
M. F. Schwartz, J. Zahorjan and D. Notkin. A Name Service for Evolving, Heterogeneous Systems. *Proc. 11th ACM Symp. Operating Syst. Prin.*, pp. 52-62, Nov. 1987.

[Schwartz et al. 1991a]
M. F. Schwartz, D. R. Hardy, W. K. Heinzman and G. Hirschowitz. Supporting Resource Discovery Among Public Internet Archives Using a Spectrum of Information Quality. *Proc. 11th IEEE Int. Conf. Distrib. Comput. Syst.*, pp. 82-89, May 1991.

[Schwartz 1991a]
M. F. Schwartz. A Measurement Study of Changes in Service-Level Reachability in the Global Internet. In preparation, 1991.

[Schwartz et al. 1991b]
M. F. Schwartz, D. H. Goldstein, R. K. Neves and D. C. M. Wood. An Architecture for Discovering and Visualizing Characteristics of Large Internets. Tech. Rep. CU-CS-520-91, Dept. Comput. Sci., Univ. Colorado, Boulder, CO, Feb. 1991. Submitted for publication.

[Schwartz 1991b]
M. F. Schwartz. The Great Disconnection? Tech. Rep. CU-CS-521-91, Dept. Comput. Sci., Univ. Colorado, Boulder, CO, Feb. 1991. Submitted for publication.

[Schwartz & Wood 1991]
M. F. Schwartz and D. C. M. Wood. A Measurement Study of Organizational Properties in the Global Electronic Mail Community. Tech. Rep. CU-CS-482-90, Dept. Comput. Sci., Univ. Colorado, Boulder, CO, Aug. 1990; Revised July 1991. Submitted for publication.

[Schwartz 1991c]
M. F. Schwartz. Resource Discovery and Related Research at the University of Colorado. *ConneXions - The Interoperability Report*, pp. 12-20, Interop, Inc., May 1991.

[Schwartz & Tsirigotis 1991]
M. F. Schwartz and P. G. Tsirigotis. Experience with a Semantically Cognizant Internet White Pages Directory Tool. *J. Internetworking: Research and Experience*, 2(1), pp. 23-50, Mar. 1991.

[Shoch & Hupp 1982]
J. F. Shoch and J. A. Hupp. The "Worm" Programs - Early Experience with a Distributed Computation. *Commun. ACM*, 25(3), pp. 172-180, Mar. 1982.

[Siewiorek & Swarz 1982]
D. P. Siewiorek and R. S. Swarz, editors. *The Theory and Practice of Reliable System Design.* Digital Press, Bedford, MA, 1982.

[Spafford 1989]
E. H. Spafford. The Internet Worm: Crisis and Aftermath. *Commun. ACM*, 32(6), pp. 678-687, June 1989.

[Spector & Kazar 1989]
A. Z. Spector and M. L. Kazar. Uniting File Systems. *Unix Review*, pp. 61-71, Mar. 1989.

[Wulf 1989] W. A. Wulf. Towards a National Collaboratory. Report of Invitational Workshop at Rockerfeller Univ. Oct. 1989.

[Zimmerman 1990]
D. Zimmerman. The Finger User Information Protocol. Req. For Com. 1194, Center for Discrete Mathematics and Theoretical Computer Science, Nov. 1990.