CMU-ITC-87-060

Administrative Cells: Proposal for Cooperative Andrew File Systems

Edward R. Zayas
Information Technology Center
Carnegie Mellon University
25 June 1987
[412] 268-6798

Table of Contents

1. Introduction

1.1. Objectives

There are several sites currently running the Andrew distributed personal computing environment, including the ITC, the CMU Computer Science Department, the Psychology Department, SUPA, the SEI and GSIA. It would be highly beneficial for these sites to share in a single file name space composed of the union of their individual file systems. The most obvious benefit is transparent access to the remote file systems, freeing users from having to issue explicit file transfers. While such transparency is attractive, it must not result in the loss of administrative control for the individual sites or "cells" of the greater Andrew community. Each cell must retain the exclusive ability to add and delete users, manage protection and volume location databases and configure file servers to best suit its needs. Also, the management costs of this united system must be kept reasonable. Noticeable degradation of local service due to overheads in maintaining the image of a common file name space certainly outweighs the advantages of such an arrangement.

This paper proposes a set of changes to the Andrew environment to allow individual installations to participate in such a global partnership without sacrificing administrative autonomy. Affected are the *Venus* workstation cache managers, the System Control Machines (*SCMs*) and various support programs. The changes are reasonably straightforward, attempting to arrive at the needed functionality with minimal disturbance to system code and maximal utilization of existing, proven algorithms. Very few of these changes are visible at the user level, allowing Andrew users to adapt quickly to the expanded environment.

1.2. Organization

This proposal begins by describing how authentication is performed in this cellular system, with Section 2 also introducing the notions of *primary identity* and *additive authentication*. Similarly, Section 3 discusses volume location resolution in this expanded environment, including the use of *cell mount points*. Naming issues in the cooperative environment are addressed in Section 4, which presents policies and mechanisms for handling service resolution. Section 5 proposes a simple extension to the *vopcon* facility to allow global monitoring of the extended Andrew system. Some unresolved issues currently under consideration are highlighted in Section 6. Finally, Section 7 presents the suggested set of steps though which Cellular Andrew can be implemented, tested and distributed.

It is assumed that the reader is familiar with the basic Andrew structure and organization, but not necessarily to have detailed knowledge of system internals.

2. Authentication

2.1. Current Mechanisms

Each user of the current Andrew environment identifies himself to the system by engaging in an authentication protocol. This procedure is first carried out via the *login* program at the start of the user's session. Authentication may take place several more times during the same session using the *su* or *log* programs, allowing the user to change his identity. After the user supplies his name and password, *login* (for example) attempts to contact an available *AuthServer* to verify this information. A list of machines hosting *AuthServers* is kept in */etc/vstab*, the same file *Venus* uses to determine the set of *FileServers* to contact. If none of these machines responds in a reasonable amount of time, the *SCM* is used. The *AuthServer* that is eventually reached checks to see if the given password matches the name. If so, it sets up and returns *clear* and *secret tokens* which serve to identify the user in its dealings with the *FileServers*. These tokens contain not only the user's central *ViceID*[1], but also expiration information to prevent processes belonging to deleted users from accessing the file system indefinitely. *Login* proceeds to pass these tokens to *Venus*, which acts as the user's agent for file manipulation. If all of the *AuthServers* are down, or if the tokens cannot be handed to *Venus*, a "local login" is performed, where the user may still access files on his own local disk *if* his password matches the one cached in the local password file.

An Andrew user can be logged in as several different people at once. This option is due to the fact that each user process belongs to exactly one *process authentication group (PAG)*. PAGs are guaranteed to be unique for all processes on a machine until a reboot occurs. User tokens are associated with the corresponding PAG, so *Venus* can select the right set when a process asks it to interact with the *FileServers*. Thus, different processes controlled by a user can be tied to different accounts. Note, however, that a single process can only be authenticated as *one* user at any one time.

2.2. Cellular Operation

In the cellular Andrew environment, users have the option of specifying the site they wish to be authenticated in when invoking *login* and its brethren. Every workstation associates itself with the administrative cell it physically resides in, as determined by the contents of the */etc/HomeCell* file. By default, the local set of *AuthServers* will be contacted as before. If the user explicitly names a remote cell, a service database will be queried to determine the set of machines advertised as hosting *AuthServers* for that site. Discussion of the organization and management of this database is deferred until Section 4. The *AuthServer* for the chosen cell is therefore contacted directly, and returns tokens which can only be properly decoded by agents in that domain. These tokens are stored in *Venus* as before, and are presented as proof of identity when accessing files that reside in that cell. The structure of a *ViceID* is expanded to include the user's 32-bit cell ID to preserve uniqueness in the expanded environment. Since *ViceIDs* appear in tokens, cell information is automatically embedded in them.

[1] *ViceIDs* do not necessarily correspond with the local *uid*, although they currently tend to match.

Users may authenticate themselves in several cells at once, but are restricted to a single identity *in any one cell.* Tokens returned from each cell as a result of each authentication transaction accumulate in *Venus*, providing *additive authentication.* This is a very useful feature, allowing a single operation to access files in several domains. Take the example where a protected file needs to be copied from the *erz* account in the CMU CSD domain into the ITC's *erz* account. If a process is authenticated as both parties, the standard *cp* works correctly. Without multiple authentication, two separate processes are needed. The CSD process may choose to deposit the file in some intermediate location made accessible to it (say, */tmp* on the local workstation disk), whereupon the ITC *erz* process moves it to its final home. Alternatively, a pipe may be set up between the two, with the first echoing the CSD file and the second writing to the proper ITC file. In order to restrict user processes to a single identity in any one cell, *Venus* only saves the last set of tokens received for that cell. This is necessary to prevent situations where the proper identity cannot be automatically selected. To illustrate, let us assume that this restriction did not hold and that an editor process held ITC tokens for both *erz* and *mikew.* It is impossible to determine a file's correct owner if it is saved into a directory writable by both people.

As stated above, *Venus* automatically selects the proper token to present to a *FileServer* in a particular domain. If no token exists for a cell, *Venus* uses an unauthenticated connection when performing a file operation, effectively reducing the client's rights to those of the *Anonymous* user in that domain. The automatic rights reduction performed when crossing into a protection domain in which the client has not identified himself is imperative if system security and autonomy are to be maintained.

While several identities are possible at any given instant, one of them must be selected as the *primary identity*, unique across all cells. Many programs need to ask "Who am I?" in order to operate properly. The prime example of such a need is *Messages*, a facility that provides a common framework for performing such related tasks as reading electronic bulletin boards and personal mail. When it comes up, it must determine who it is operating on behalf of to select the proper set of directories on which to base its activities. If a person is authenticated in several cells, there is no way to unambiguously discover which mailbox to operate on in an automatic way. The *getuid()* family of routines are no help, since they only operate on local password files and cannot even handle the simple case of a user who simply logs in remotely and wishes to take on that identity.

Primary identities are established during authentication by the use of switches in the command lines of the *login* family of programs. *Venus* marks the given identity as primary when it stores it. Primary identities are obtainable by providing counterparts to the getuid() routines. For example, *getvuid()* contacts *Venus* and gets the stored information. This change to a global, unique naming system (that is still controlled locally) not based on password files has already been considered by groups in the ITC in other contexts than cellular operation. This project serves as the ideal catalyst to implement such a facility.

This new authentication scheme has many positive characteristics, and is much more attractive than an attempt to maintain global protection files:

1. The structure currently used for protection databases, the method by which protection information propagates within a cell and all operations in existence for user additions and deletions are completely unchanged. As far as these items are concerned, there has not been *any* change in Andrew.

2. A cell's protection database is completely independent of both the number of cells in the community and the corresponding sizes of their files. A global protection database may

result in files with unworkable sizes, and will grow with the number of cells represented. Password files in the ITC are *already* large enough to require a separate index.

3. Maintaining reasonably consistent copies of a global protection database requires much more communication among the cells. The only information that must be kept current in the proposed strategy is the list of *AuthServers* for each protection domain. These lists are not expected to change very often. In addition, a global approach requires that sensitive information be transmitted across the network.
4. An error in protection database updates by one cell could corrupt other cells if a global approach were used. To avoid this, each domain must expend energy on carrying out consistency checks on any new information received before merging it in. No such corruption is possible in the proposed system, as updates never cross protection boundaries.
5. User names in the proposed system do not need to be unique in the global community, rather only within a cell.
6. Remote authentication activity does not raise the load on local servers, since the remote agents are contacted directly.
7. Special privileges within a cell do not automatically carry over to other cells, maintaining security and exclusivity. For example, a user with *System:Administrator* rights in cell *A* cannot delete users in cell *B* unless he is *also* authenticated there as a person with those same rights. Automatic rights reductions insure that remote users who do not have accounts in another cell or have chosen not to identify themselves as that person are treated as the *Anonymous* user. However, an authenticated administrator for cell *B* can still carry out his full range of activities there even if he logs in on a workstation that belongs to cell *A*.

3. Volume Location and Management

3.1. Current Mechanisms

The Andrew *volume* concept is a central one. Volumes are containers for a hierarchy of files, and are the basic units of data moved between *FileServers*. The current Andrew file system is composed of a collection of system and user volumes, joined together at *mount points*[2] in such a way as to present the image of a single, seamless file tree. *FileServers* may be instructed to *clone* read-only volumes from the read-write versions and replicate them amongst themselves for greater availability and reduced per-server demand. Mechanisms exist where administrators may create, delete, back up, replicate and relocate volumes. Complete information on the status of all Andrew volumes is kept in the *Volume Location Data Base (VLDB)* file on the *SCM* and replicated at each *FileServer*. As volume operations take place, the individual *FileServers* keep track of the changes to the volumes they host. Periodically, the *SCM* polls each *FileServer* machine and collects these changes, merging them back into a new VLDB and redistributing it.

Volumes are identified either by name or by number, both guaranteed to be unique in the Andrew environment. Volume numbers are 32-bit values whose top 8 bits identify the *FileServer* on which it was

[2]Mount points are Andrew file system objects that have no real connection with Unix file system mount points. Instead, they indicate where a particular volume is to appear in the file system tree.

created. Volume names tend to be of the form *subclass.id*, where *subclass* identifies the general group to which it belongs. For example, *user.erz* belongs to the set of user volumes, and holds the files associated with the author's account. Similarly, volume *ibm032.bin* holds executable images specific to the IBM RT. This naming structure is not enforced by the system, but makes it easier to identify and locate volumes based on the purpose they serve.

The *Venus* workstation cache manager, in interpreting path names for its client, asks the *FileServers* for volume location information. Since all *FileServers* keep a copy of the VLDB, any of them can answer such requests. Once the site(s) a volume resides on is(are) determined, *Venus* can access the files in the volume to the full extent of the permissions held by its client. The list of hosts in the */etc/vstab* file (discussed in Section 2.1) is also used by *Venus* to choose the *FileServers* to contact for volume location information. When *Venus* first comes up, it learns the name of the root volume for the entire Andrew file system via the *RViceGetRootVolume* facility. Once informed about the root volume, *Venus* is capable of processing all path names presented by its user, even across mount points leading to different volumes. Venus caches volume information, and uses a lightweight daemon process to check its mappings between volume names and numbers every two hours.

3.2. Cellular Operation

In the cooperative Andrew environment, foreign file systems are rooted at *cell mount points*. This new construct resembles the standard mount point in most ways, and contains additional information concerning the cell hosting the represented file system. When *Venus* first encounters a cell mount point while resolving a path name, it consults the service database to determine the set of hosts providing volume location information for that cell. The root volume for the remote cell is then determined by directing an *RViceGetRootVolume* transaction to one of these remote machines. Because the 32-bit volume numbers are no longer guaranteed to be unique across cells, *Venus* indexes its cached volume information on both the cell number and the volume number. In general, volume location requests generated when crossing standard mount points are directed to the set of servers associated with the *latest* cell mount point encountered in the path. If no such construct has been crossed, then these requests are delivered to the local server set.

This scheme has the same set of advantages as the new approach to accessing authentication services. These include the basic preservation of the existing mechanisms, independence of VLDBs, preservation of workable file sizes, avoidance of intra-cell communication needed to support a global VLDB, compartmentalization of errors, direct application of remote volume location workload to remote servers and automatic rights reductions across protection boundaries. In addition, the proposed system has two favorable characteristics:

1. Local file servers are completely unaffected by the file traffic generated when its workstations access remote files. The only parties involved in inter-cell file transfers are the remote *FileServers* hosting the data and the individual workstations performing the accesses.

2. The cell mount point construct allows remote file systems to be rooted *anywhere* in the local cell's file system. This allows a cell's administrators to shape their view of the file space to suit their own purposes.

While volumes may still be shuttled back and forth as they have always been between *FileServers* in the same cell, they must not be allowed to move *between* cells. Each administrative domain has full control of its volumes, and is completely responsible for their housing and backup. The only difficulty with this restriction is in handling the natural movement of users between sites. Transferring a user from the ITC cell to the CMU CS cell can still be accomplished in spite of this limitation. A dump is taken of the user's ITC volume, and the volume itself is destroyed. The dump file is copied to the CS cell, where it is restored to a new volume belonging to that site. This operation must be carried out as a cooperative effort between the maintainers of both cells.

4. Database of Services

Sections 2 and 3 assume the existence of a service database in the Cellular Andrew system capable of providing up-to-date listings of the set of hosts in any given cell which provide certain essential services. These services presently consist of authentication and volume location. In this proposal. the ARPA Internet domain system is used as the primary mechanism to implement this database. The domain system is described at a high level, along with the changes needed to use it to manage information concerning the above services. The advantages of this facility are then examined, followed by a presentation of the fallback techniques to use in cases where the primary database is unavailable.

4.1. The ARPA Internet Domain System

This description of the ARPA Internet domain system presents its basic features and operation. Readers wishing more detailed information are referred to a set of RFCs (Request For Comments) on the subject, available from the Network Information Center (NIC): 973, 883 and 882[3]. The following description is in fact based mostly on the contents of RFC 882.

The ARPA domain system was proposed in 1983, responding to the desire to create a consistent and practical database for naming resources in the heterogeneous Internet environment. Because of the frequent updates from its rapidly growing constituency, the existing centralized database was becoming unmanageable. The chosen distributed, scalable solution for keeping such things as host-to-address mappings and mailbox locations has three major components:

1. *Domain Name Space*: This is a specification for a tree-structured name space, with each node and leaf housing information about that region. The domain name *spice.cs.cmu.edu*, for example, refers to the region allocated to the SPICE Vax (*spice*), a subdomain of the Computer Science Department (*cs*), in turn a subdomain of Carnegie Mellon University (*cmu*), again in turn a subdomain of the portion of the ARPANet dedicated to educational institutions (*edu*). Query operations specify the domain name of interest and the type of information desired. Replies typically return one or more records containing the desired information.

2. *Name Servers*: These server programs house the information held in the domain tree, which includes structure data about the tree itself. Each name server is said to cover a particular *zone*, composed of the parts of the domain tree in which it is an authority. Name servers are free to cache information about other zones and respond to queries about them, but only the authoritative name server for a zone has the complete and up-to-date picture

[3]These and other RFCs are also available at the ITC, residing on-line in directory */cmu/common/rfc*. The *rfc-index.txt* file in that directory is the index of available documents, and RFC **nnn** lives in file *rfc***nnn**.*txt*. Furthermore, files named **n***99.txt* contain short descriptions of all RFCs numbered **n***00.txt* to **n***99.txt*.

for that zone.

3. *Resolvers*: These are programs that serve as an interface between a user application and the set of domain name servers. Resolvers must be able to access at least one name server. Given a user query, a resolver attempts to satisfy the request through interaction with its known server(s), including follow-ups to responses that do not answer the query directly but rather refer to other, better-informed name servers. The records obtained by resolvers have timeouts associated with them, ranging anywhere from a second to several years. Set by the name server, this insures that cached copies of these records will eventually be flushed at the desired rate. As an implementation note, many resolvers are simply implemented as user-callable routines.

4.2. Advantages to the Domain System

The Internet domain system appears tailor-made as the primary vehicle for managing and accessing the greater Andrew community's authentication and volume location information. Among its advantages for the task are:

- *Control*: Cell administrators have complete control over their zone. Specifically, they have the exclusive right to set and reset the list of machines they wish to associate with each service, determine the granularity of updates by choosing how long information extracted from their zone is valid, and implement the name servers and resolvers to their taste. For system administrators who do not care for such hands-on responsibility in this area, any or all of these things may be transparently delegated to entities higher up in the domain tree.
- *Scalability*: As a system specifically designed to scale gracefully, the domain approach meshes well with Andrew's goal of supporting a large user population. Not only will the information demands of a large number of workstations inside of a single cell be met, but the chosen system can support large numbers of Andrew cells.
- *Service Speed*: The time needed to ascertain the needed information about authentication or volume information for a remote cell is reasonable. In a demonstration of the system provided by Craig Everhart, transactions with distant name servers consistently took on the order of a small number of seconds. Since the scheme allows for the local system to run caching servers, commonly-accessed information is often discovered locally, cutting the latency considerably. This response time is perfectly acceptable for several reasons. This type of information is only needed the *first* time a pathname crosses into a particular remote cell. *Venus* caches the names of the servers it discovers on a per-cell basis, and reuses them until their timeout periods expire. Thus, the frequency of interaction with the domain system is expected to be low, especially if reasonable timeouts on the information are provided.
- *Adaptability*: New domains may be easily created, allowing the quick incorporation of new Andrew cells. In addition, existing domains may be reorganized to suit any changes in the internal structure of a particular cell. A parent zone may spawn as many subzones as it wishes, giving each the exact set of rights and responsibilities it chooses.
- *Extensibility*: While authentication and volume location are the two services needed for the current system, there is no reason why other cell information cannot be advertised in this way. The domain approach was designed to easily add new service types, allowing the same system to store data concerning a wide variety of things.
- *Track Record*: The domain system is a working, proven facility already in wide use, and continuing to expand. As mentioned above, it is already in place here at the ITC to perform mail routing. It is the result of a lot of work by networking authorities across the nation, and provides a standardized solution to a very difficult problem. It would not be prudent to "reinvent the wheel" by designing and coding a localized scheme to tackle the same problem. In fact, the competing (and most likely incompatible) mechanism produced might

put off potential Andrew sites wishing to stay with the mainstream.

4.3. Local Fallback

There is a good deal of redundancy in the domain system. Name servers for a zone may be replicated, providing continued service as long as at least one is operational. Even if all authoritative servers are down simultaneously, other name servers and local caching servers may have accumulated a good deal of the zone's information. In this case, these agents will continue to provide (non-authoritative) service while the stricken zone repairs itself. However, it is still reasonable to provide a backup system for the services required for the cellular Andrew community.

It is proposed that a backup file, (/etc/domain.backup), be maintained both in each cell's Vice file system and on each workstation's local disk. This file, used only when the domain system cannot provide any information about a service request, contains lists of service machines for the most popular cells. These files are not expected to be completely up to date, but rather to provide hints for a last-ditch attempt to contact a remote cell. The centralized copy of the file stored in Vice may be updated manually, or by a periodic server which performs the queries needed to refresh the mappings. The workstation copy of the file can be either be updated manually or by the *package* facility, as is done now with the */etc/password* file. It is not expected that this facility will ever be heavily used, since the domain system is relatively stable and reliable. However, the low maintenance required to support this backup scheme makes it cost-effective, especially in the rare occasions when total zone failure occurs and persists long enough to time out entries cached elsewhere.

5. Monitoring

The *SCM* for a cell performs a valuable performance monitoring service. Each *FileServer* maintains statistics on its own operation, and the *SCM* periodically gathers and consolidates this information. Using *vopcon*, the status of the entire cell can easily be monitored from any location in the cell. It is simple to extend this service to allow performance monitoring to be available for the entire Andrew community. A trivial program could be written to poll the *SCM* for each cell and produce a meta-collection of timing and capacity statistics, *FileServer* availability, and so on. Once again, it is also possible to rely on the domain system to advertise per-cell monitoring services.

6. Unresolved Issues

Internal ITC evaluations along with design review meetings involving representatives of the various Carnegie Mellon University departments have exposed two unresolved issues in the current proposal. Work is now underway to arrive at reasonable solutions for mapping *uid*s returned across cell boundaries to their proper string equivalents and for allowing certain sites to sacrifice some autonomy so that other cells may perform selected services on their behalf.

6.1. Mapping *uids*

Such programs as *ls (list directory)* perform mappings from internal *uids* to string names suitable for human consumption. Although these mappings will still be performed correctly by the proposed cellular system in the local case, they will produce erroneous results for files and directories in remote cells. The problem lies in the fact that *uids* are no longer unique across the entire community and that the standardized block of information used in this operation doesn't have room to carry corresponding cell identities. Proper translation requires that the corresponding cell for each *uid* be available so that the matching database (an */etc/passwd* file or White Pages) can be used.

Several solutions (and non-solutions) are being considered. If it is decided that providing accurate tranlations of remote *uids* is not necessary or too difficult, then foreign *uids* can be tagged as such by creating a special *RemoteUser* account in all cells and having *Venus* automatically replace all remote *uids* with this value. Some "real" solutions involve using non-standard information blocks that include the necessary cell ID or using unassigned high-order bits in the *uid* itself to encode site information.

6.2. Delegation of Responsibility

Some cells may wish to trade a portion of their autonomy in return for certain services which it cannot or chooses not to provide for itself. The specific case brought to our attention was volume backup. Smaller sites may not have the proper staging and tape backup equipment necessary to save and restore its own volumes. The current proposal makes it somewhat awkward to delegate this (or any other) responsibility to a foreign cell. While remote sites may access files in other cells, they do not have the right to perform volume operations there directly. The general question of whether such piecemeal control over cellular autonomy is feasible in the proposed system must be studied. If methods are found to accomplish this goal within the established framework, they may then be applied to the stated backup problem.

7. Implementation Strategy

Implementing and testing these proposed changes is not expected to be a very complex matter. The brunt of the changes are borne by *Venus*, with the rest of the load shouldered by the programs performing authentication transactions for users. The new Andrew system capable of joining cellular relationships will be fully compatible with existing Andrew implementations - any program running on the old system will also run on the new one without changes, ignorant of the added possibilities. The following is the suggested procedure for introducing the new cellular file system structure:

1. Rewrite the software mentioned above (*Venus*), login, log, su, etc) to implement the proposed changes. This includes adapting existing name servers and resolvers to interface with *Venus* and creating the authentication and volume location service databases.
2. Create a "test cell" running the new Cellular Andrew software, and put it through the standard validation routine to make sure correct local operation has been preserved. In particular, a new *SCM* and at least one *FileServer* need to be brought up for the cell.
3. Create a second test cell and attach both entities to the proper cell mount points. Test the remote authentication and volume location resolution by executing test suites containing the different operations now possible (remote logins, additive authentication, attempts to perform illegal accesses, editing remote files directly, etc).

4. Test the backup mechanism for domain naming by simulating network failures, falling back on the */etc/domain.backup* file.
5. Write the global performance-monitoring software, and get it to run on the baby community.
6. Begin the upgrade of standard programs to use the new mechanisms. A real-world acid test for the new system is the conversion of the *Messages* program. It exercises all the cellular mechanisms, and is a good candidate to take advantage of transparent access to remote files (mail can be deposited directly into mailboxes without stripping formatting information as now required by SMTP).
7. Begin distribution to real sites, forming the basis for the cooperative Andrew community.

8. Conclusion

The proposed design for a cellular Andrew environment allows several sites to cooperate in providing a unified file name space without sacrificing administrative autonomy. The fact that each cell manages its own protection and volume databases in an exclusive fashion minimizes the network communication between cells, compartmentalizes errors and allows an arbitrary number of cells to enter the community without causing undue local disturbance. Each process in the cellular system can still be properly identified, and automatic rights reductions assure the community's integrity. It is expected that the convenience of transparent access to files across protection boundaries will greatly enhance Andrew's usefulness and attractiveness.

9. Acknowledgements

Many ideas have been incorporated into this document from a variety of people. Bob Sidebotham came up with the cell mount point concept, preventing incompatible changes from having to be made to the volume subsystem as originally contemplated. Craig Everhart introduced the author to the ARPA Internet domain world and pointed out its applicability to service resolution. Several people, including David Nichols and Mike Kazar, prompted the author to rethink the inclusion of multiple identities for access purposes as embodied in additive authentication. Nathaniel Borenstein made many good observations from the point of view of an applications programmer, leading to the discovery of the need for a primary identity. All of the ITC's File Systems Group (Mike Kazar, Sherri Menees, Bob Sidebotham and headed by Mike West) contributed in some form or other through their comments and observations on earlier drafts of this proposal.