# Administration of a Unix computer network

by

**Jeffrey Lynn Detterman**

A Thesis Submitted to the

Graduate Faculty in Partial Fulfillment of the

Requirements for the Degree of

**MASTER OF SCIENCE**

Department: Electrical Engineering and Computer Engineering
Major: Computer Engineering

Signatures have been redacted for privacy

Signatures have been redacted for privacy

Iowa State University
Ames, Iowa

1994

# TABLE OF CONTENTS

# CHAPTER 1. INTRODUCTION

The recent acceptance of Unix as a mainstream operating system has generated a great deal of demand for the black magic art of Unix system administration.

When Unix was first developed it was designed to be used as a stand–alone operating system that allowed the end user the ability to run their programs on their computer. Unix was generally run on one big main frame class computer. The early day system administrator generally managed the Unix computer on a part–time basis and spent the majority of their time working in some other capacity.

As the popularity of Unix grew and the cost of computers dropped dramatically, Unix evolved into a networked operating system. The computing model no longer consisted of a single computer, and had now become an entire network of computers capable of sharing resources. The computers each acted as a stand–alone processor, capable of sharing resources with any other processor on the network.

This created a demand for full–time positions as Unix system administrators, dedicated to keeping the entire network up and running. My thesis will outline and discuss the measures taken to administer our Unix network. The opinions presented should be used as a relative guide, and not an absolute statement on how or how not to administer a Unix environment.

# CHAPTER 2. GOALS

Our goal was to provide our Engineering community with the best computing environment possible. Unix was selected as the best operating system that would help us achieve this goal.

## Goals

The goals of our team were the following:

- Provide the user with the best Unix environment possible.
- Provide the user with a consistent user interface. Provide the user with a common look and feel on all workstations
- Provide as much redundancy/fault tolerance into the Unix network as possible.
- Provide the user the ability to run any application from any workstation in the network.
- Provide the user the ability to run applications without making any modifications to their startup files.
- A scalable solution capable of managing several hundred workstations.
- Provide the user with a global login that allows him/her to login to any workstations on the network.
- Provide the user with a global file system that allows him/her to access any file on any workstation that he/she has access privileges to.
- Provide the user the ability to print to any printer from every workstation on the network.
- Provide the user with outbound Internet access.

# CHAPTER 3. HISTORY

I was hired as the Unix System Administrator for Rockwell's Collins Commercial Avionics Division in November of 1990. At that time the Unix network consisted of only three DECstation 3100s from Digital Equipment connected to the same ethernet backbone (Figure 3.1). The DECstations were used for a handful of applications by only a few engineers.

The rest of the network consisted primarily of Vaxes running VMS and PC's running Digital's Pathworks product. The main communication protocol used by the clients was DECnet. The network was essentially several backbones bridged together to form one logical network. The TCP/IP portion of the network consisted of only one class B subnet.

Today the Unix network is made up of the following types of Unix workstations:

- 220 DECstations
- 78 Sparcstations
- 5 Silicon Graphics workstations
- 9 Hewlett Packard workstations
- 23 DEC Alpha workstations
- 22 PC's running various versions of Unix
- 18 Apollo workstations

In addition to the Unix workstations, there are also 450 Vaxstations and 1200 PC's sharing the network. These hosts are capable of running both DECnet and TCP/IP.

The network is now made up of 20 TCP/IP subnets all connected via an FDDI backbone with connections to remote facilities in Florida, Washington, Wisconsin and other Rockwell facilities in Iowa.

The workstations are used for a wide range of applications for just about all aspects of Engineering which include:

- MCAD design including solid modeling
- PCB design
- ECAD layout and simulation
- VLSI design
- Software Engineering
- Systems Engineering
- Desktop Publishing

The Unix Network has become an integral part of the engineering process within our division. Employees depend upon the availability of the Unix network to perform their jobs.

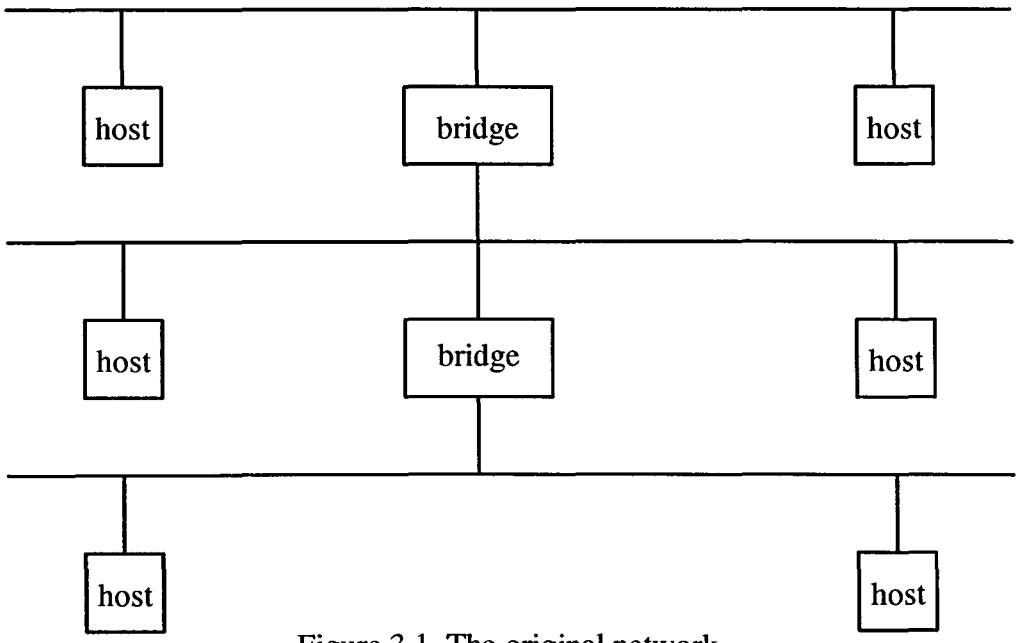


Figure 3.1. The original network.

# CHAPTER 4. THE NETWORK

## The Original Network

The original network was made up of several thinnet (10base2) segments scattered throughout the various buildings and labs within Rockwell. The segments were all connected to a thicknet backbone (10base5) via bridges. The primary protocol used for communication between the hosts was DECnet.

This configuration worked well for a limited number of TCP/IP hosts and a large number of DECnet hosts because DECnet is a bridgeable protocol. We were generally able to limit the traffic on a particular segment by adding bridges as necessary to isolate traffic.

The TCP/IP portion of the network consisted of one subnetted class B address, 131.198.200. Using 8 bits of subnet mask we were then able to connect 254 TCP/IP hosts to the network.

This configuration worked fine for a limited number of TCP/IP machines on the network, but when we started converting all our DECnet hosts to TCP/IP and added several hundred Unix clients to the network we began running into problems. For one thing, we were limited to 254 hosts with the one class B subnet. We were also seeing problems where the segments became congested with less than 30 percent network traffic. The segments needed to be separated into smaller pieces and the traffic isolated to those areas. This suggested that a routed environment would be needed.

There are two divisions at our campus, Collins Commercial Avionics (CCA) and Collins Avionics Communications Division (CACD). Each division has several different organizations that manage various collections of computers within each division. When one group would take over the office space in a particular area of the campus they would cable it to their standard net-

work cabling scheme. When that group eventually moved out of that area, another group would move in. The area would need to be rewired to the new group's standard network cabling scheme. This was an unnecessary cost, both in time and money.

Our divisions perceived this as a problem and elected to standardize on twisted pair (10baseT) as the preferred inter–office connection medium on our campus.

**The Physical Network**

Each office was then wired with several twisted pair ports. These ports were controlled by intelligent hubs located in wiring closets in each building. By using the intelligent hubs, we were able to monitor and isolate traffic on a particular segment very easily. It also helped in diagnosing problems. If a fault was identified in a segment, that segment could be logically shutoff by the software controlling the hub.

With this new wiring configuration we were able to easily reuse existing networks. When a group moved out of an office area, the new group did not need to rewire the office area. They just needed to connect the twisted pair runs into their own hubs.

This configuration also gave us the ability to provide different networks to the same office, thus allowing a great deal of flexibility. Employees in the same office could now have a connection to the local network and a remote network.

By using twisted pair with intelligent hubs we were able to break the network into small, manageable pieces. Small networks could be created that were controlled by the hub. Software on the hub then allowed us to monitor and control network traffic. An added benefit was that we could lock down ports so users could not arbitrarily hook their machines to the network. This was all possible from one or more management stations.

We chose twisted pair with the hope that a faster standard would someday evolve and we could run that protocol over the existing network. These protocols included: CDDI and ATM.

**The Logical Network**

The original network consisted of one TCP/IP subnet capable of holding 254 hosts. To accommodate the addition of the many new TCP/IP hosts, we expanded our network from one class B subnet to 20 class B subnets. Additionally, we moved from a bridged network to a routed and bridged network. With the new scheme we routed TCP/IP between each subnet, and bridged DECnet between groups of subnets also known as DECnet areas.

We standardized on subnets in the class B network 131.198 with an 8 bit subnet mask of 255.255.255.0 and a broadcast address of 255. This made assignment of IP addresses rather easy, due to the even boundaries that the subnets fell on. For example, if a host had an address of 131.198.213.10, it was very easy to determine that the network address was 131.198.213 and the host address was 10. When we switch to variable length subnets in the future we will lose this simplicity.

There was still the limitation that there could only be 254 hosts per subnet. In the Futures section I will discuss our current plans to go to a variable length subnet mask to resolve this problem.

The subnets were generally broken down along geographical boundaries. In some cases, we wanted hosts that were in one subnet to appear as though they were in a different buildings local network. To accomplish this we decided to use Fiber–Optic–Inter–Repeater–Link (FOIRL) links. This was primarily needed for the VMS hosts that were clustered together. We did not want the VMS cluster members to boot across a router, so we would string fiber from a port on their hub to the hub for the network that they wanted to be in. We did not want the VMS cluster members booting across the routers because this would increase their dependance on the router. If the router crashed, so would the VMS cluster member.

In an effort to eliminate routing problems, we standardized on all the TCP/IP clients using static routes to implement their routing. The clients would point to the routers to get their

routing information. This was primarily driven by two instances where inexperienced users had connected TCP/IP clients to the network and started the routed daemon on their machines with the wrong option, advertising to the routers that they were a route to the Internet. This caused the routers to try and force all Internet traffic to the host in question. This brought our Internet communication to it's knees. This occurred twice before we decided to not allow the clients to run the routed daemon locally. If the same situation occurs again, we will only need to flush the routing tables on the routers. With the old method, we had several hours where the routers kept getting RIP updates that were bouncing around the network.

**Connecting the Physical Networks Together**

Once we had standardized on a multi–subnet routed network using twisted pair as the physical medium, we needed to determine how to connect the intelligent hubs together. To do this, we installed an FDDI backbone between the various buildings. Additionally, we purchased three Cisco ASG+ routers to act as routing devices for the various subnets and hooked them directly to the the FDDI backbone (Figure 4.1).

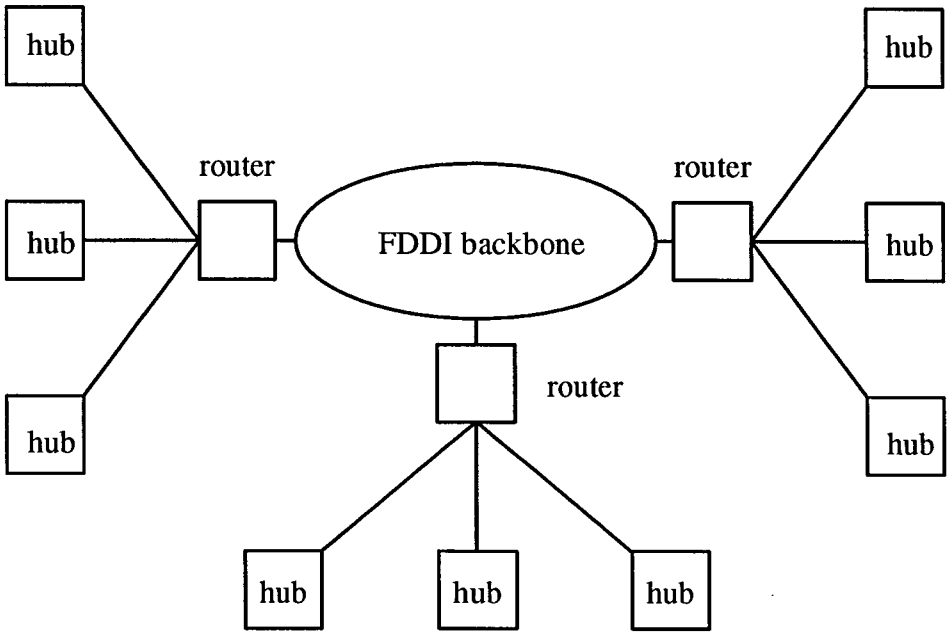


Figure 4.1 The current network.

# CHAPTER 5. FILESYSTEM LAYOUT

Once we had determined the underlying network, the next step was to determine where we were going to store the operating system for each one of the Unix clients. We looked at three possible schemes:

- Diskless clients
- Dataless clients
- Datafull clients

## Diskless Clients

The first scheme that we looked at using was diskless Unix clients. Diskless Unix clients are just that. They don't have any physical disk drives attached locally. They get all of their software from a server machine somewhere on the network. An area on the server is set aside for each client's configuration files. The clients all boot off of one copy of the operating system located on the server. We found several problems with this configuration that we did not like. First, the configuration created a great deal of network traffic and second, it was totally dependant on the network being available. If the network wasn't functioning correctly or was overloaded, the clients were rendered useless. This might work in small isolated networks where the only clients on the network were the diskless clients and the server, but we could not control this. We were forced into sharing the network with all of the other machines that were out there. So we decided against this approach.

The advantage to this approach is you don't need to buy a disk drive for each client, which makes it a cheap solution. The clients also share one copy of the operating system, so it in theory requires less time to administer. In my opinion this approach sacrifices user productivity for cheaper administration costs.

**Dataless Clients**

The second scheme that we looked at using was dataless Unix clients. Dataless Unix clients are a hybrid of diskless Unix clients. Small disk drives are bought for each client and mounted locally. The disks generally will hold the Unix root partition and act as a local page and swap device. This makes this scheme a little more robust than diskless Unix clients. It is still, however, very dependant on the network. The clients now share the Unix /usr partition from a server and store their configuration files locally. This scheme was a little more expensive due to the cost of the local disk drives but was a lot more robust. We still felt that this scheme imposed too much of an unnecessary load on the network.

**Datafull Clients**

The third and final scheme that we looked at was datafull Unix clients. Datafull Unix clients have their entire operating system loaded on their local disk drives. We chose this approach because the clients could withstand network outages better. If the network went away, the machine would wait for the network to come back. Then the user could continue doing what he/she had been doing. With the dramatic drop in SCSI disk drive prices over the last couple of years, we thought that the extra money for a local disk drive was well worth the investment. We have standardized on 1 Gb disk drives as a minimum configuration for each Unix client.

We have found this configuration to work very well. It produces very little network traffic and adds stability to the local clients. When we first looked at the various schemes, many of the vendors recommended diskless or dataless as their model for Unix networks. The reasoning was that the diskless and dataless approaches required less administration. We found that whichever method you used, it still required the administrator to maintain various configuration files for each of the clients. The location of the files would differ depending on the scheme used, but the time associated with maintaining these files would not vary a great deal. We maintain that the difference between the three schemes lies in the time spent loading the operating system. In

diskless and dataless, the operating system is loaded onto the servers and all of the clients then look to the servers. With datafull the operating system must be loaded onto each client. We feel that this extra administration time is easily recovered by a relatively small number of network outages that would lead to work stoppage in the other schemes.

To minimize the time necessary to load the operating system and to simplify the administration of the Unix clients we have taken a standardized approach. Every clients operating system load is identical, except for the configuration files. The configuration files consist of the following files:

- /etc/aliases
- /etc/auto.master
- /etc/auto.direct
- crontab file
- /etc/ethers
- /etc/exports
- /etc/fstab
- /etc/hosts
- /etc/hosts.allow
- /etc/hosts.deny
- /etc/hosts.equiv
- /etc/hosts.lpd
- /etc/inetd.conf
- /etc/mail.aliases
- /etc/motd
- /etc/netgroup
- /etc/ntp.conf
- /etc/passwd
- /etc/printcap
- protocols
- rc

- rc.local
- resolv.conf
- .rhosts
- services
- sendmail.cf
- shells
- svc.conf
- syslog.conf
- termcap
- /etc/ttys

If a client loses it's disk drive then we just need to swap the disk with a preconfigured disk drive and reload the configuration files. We are currently developing a prototype system to maintain these configuration files via Revision Control System (RCS) and rdist. I will explore this in the Futures section of this document.

To simplify this process, we have tried to standardize these configuration files whenever possible and have used NIS wherever we could. Keeping the configurations similar has simplified this process a great deal.

**User Data**

We took a very different approach than most sites when deciding where to store the users data. Instead of storing the users data on dedicated servers, we store the data locally on the users workstation. We create a directory on each machine called /usr/users. The users home directory is then loaded locally to this directory. By storing the data locally the user is generally able to continue to work even if the network is down. The application that the user is using is generally loaded in memory from an application server. When the network goes down the user is able to continue working, making modifications to his/her local disk.

We enforce this policy with a great deal of authority. All user data must be located in the directory /usr/users. We standardized on this policy for two reasons. The first reason is that we can now be sure where the users data is, and therefore can guarantee that if we backup the /usr/users directory on every node that we will be backing up all of the users data. If the user is allowed to place his/her home directory anywhere, we would have to either backup every directory on every disk throughout the network or maintain a list of which directories to backup on each client, neither of which we wanted to do. The second reason is to minimize the users confusion level. The users know that every users home directory is located in the directory /usr/user, so they can access each others accounts easily. To allow users to access one anothers directories easily, we have created a directory called /accts on each system. This directory contains a link for each user that points to the users home directory. Users can then easily access one anothers home directories by specifying /accts/username.

**Application Data**

We decided to load all application data into the directory /usr/apps. If an application is loaded locally on a client then it is installed in the directory /usr/apps/application_name. We did this for basically the same reasons that we standardized on /user/users for storing user data. One, it is simple and easy to remember, two, it makes reloading the operating system much easier. If all of the local applications are loaded in the directory /usr/apps then all you need to do is save this directory, upgrade the operating system and then restore the /usr/apps directory. If you allow the applications to be stored anywhere then it is easy to overlook applications stored in non–standard areas.

**Project Data**

We decided to store project data on the local workstations in the directory /usr/apps/projects. Project data generally consists of data that would be used by a project rather than by a single

person. An example of user data would be a printed circuit board that a single user was designing. This data is best stored in the users home directory on his/her local workstation. An example of project data would be a shared data area that contained documentation for a particular project that multiple people were working on.

# CHAPTER 6. NETWORK SERVICES

There are many different services that a Unix Administrator can configure to make the administration of a Unix network easier. I will try to address some of these services in this chapter.

## NIS

To achieve the goal of allowing users to login to every client on the network, we chose to implement Network Information System (NIS) to manage most of our configuration files on each one of the clients. NIS was formerly known as Yellow Pages (YP).

NIS is a distributed database that replaces copies of commonly replicated configuration files with a centralized management facility. In a standard Unix environment the clients configuration files are local ascii text files that must be maintained by the administrator. With NIS you have one database of configuration files located on the master NIS server. The clients then ask the server for information instead of looking it up in the ascii files. To add flexibility you can configure slave servers throughout the network to lighten the load on the master server. These slaves are updated from the master via file transfers on a regular basis.

We use NIS to share the following configuration files:

- /etc/aliases
- /etc/group
- /etc/hosts
- /etc/mail.aliases
- /etc/netgroup
- /etc/networks
- /etc/passwd
- /etc/protocols
- /etc/rpc

- /etc/services

By running NIS we are able to allow any user to login to any workstation. If we used the standard Unix approach, we would need to keep each password file up to date with each users password information, on every Unix client. Worse yet, the password information might be different from one client to another.

Another feature that NIS provided was the ability to create netgroups. Netgroups are a way to create sets of users and hosts that can then be specified in several of the local configuration files. Normally this type of information is listed statically in the local configuration files. For example, if I wanted two clients, *hosta* and *hostb*, to be able to access a particular client via NFS, I would need to statically list *hosta* and *hostb* in the /etc/exports file on the client in question. If I later decided that I wanted to add or delete from this list, I would need to modify the exports file accordingly. The concept of a netgroup lets me specify the netgroup in the exports file. Then, if I want to modify the members of the netgroup, I just need to modify the NIS netgroup map, not the local configuration file. We use this feature a great deal.

NIS allows us to control the local configuration files as well. By installing the standard configuration files that come with the operating system and then serving the files listed above via NIS, we can make changes to one central location that are seen by all of the Unix clients. Without NIS, we would again need to maintain an up to date copy on each one of the Unix clients.

There are several security issues that might cause many administrators to avoid using NIS. We determined that the security issues were out–weighed by the simplicity and convenience of using NIS. We were able to limit the security issues to local issues that we could tolerate. I will address these issues in the chapter on security.

We determined that for maximum flexibility and redundancy we needed at least two NIS slaves per subnet, because the NIS clients use broadcast messages to determine which NIS server to use. These broadcasts will not cross the routers because we have disabled broadcasts between

subnets. By having two slaves per subnet, if one server goes down then the clients will roll over to the other one. If we did not have the second slave, then the clients become useless until the NIS slave comes back on-line.

We have also added configuration management to the configuration files on the NIS master via RCS. This allows the administrators to revert back to old versions of the NIS maps. In the past, one administrator would make a change that might cripple the network and the rest of the administrators would not know what had changed. With RCS the administrators can go in and see what the administrator had changed and easily revert back to the older version. This has saved a great deal of time and confusion.

**NFS**

To allow users the ability to see their home directory from every Unix client and be able to access any file on the network, we decided to use Network File System (NFS).

NFS is a distributed filesystem. An NFS server has one or more disks that are remotely mounted by NFS clients. To the NFS clients, the remote disks look and act like local disks. Access to remote NFS servers is controlled via entries in the /etc/exports file. Each filesystem that is to be available for remote mounting is listed along with the clients that will have access to them. If no clients are listed, then all clients have access. Netgroups can be used to grant access to NFS clients. Instead of specifying a particular client or a list of clients you can use a netgroup name. This name will be looked up via NIS when the mount is requested. This allows the administrator to add new clients to the network without having to modify each clients /etc/exports file in order for the new client to be able to mount it.

**Automounter**

To make accessing remote filesystems and administration of remote mounts easier, we decided to use the automounter. The automounter is a tool that automatically mounts NFS file-

systems as they are referenced and unmounts them when they are no longer needed. We use the /net option to the automounter to simplify things even more. This option causes all NFS clients' disks that are mountable to appear as: /net/*hostname*/diskname, where hostname is the actual name of the remote client. As an example, if I wanted to access all of the disks on a client named fred, all I would need to do is to set my working directory to /net/fred. I would then see all of the disks that the NFS server fred was allowing my client to NFS mount. This allows the users to access any disk on the network in a simple and easy way.

This also simplifies the administration of the /etc/fstab file on each one of the clients. If we did not run the automounter, then we would need to create an entry in the fstab file for each remote client and its disks that a client wanted to access. This would take a great deal of time to accomplish let alone maintain.

The automounter also decreases the amount of network traffic as disks are mounted on a need only basis. Disks are only mounted when they are needed and dismounted when they are not. In a standard NFS configuration, a remote clients disks are NFS mounted at boot time and stay mounted while the client is up. There is a certain amount of handshaking that goes on between the NFS client and server while the disks are mounted. With the automounter this traffic is reduced since the disks are only mounted when they are actually needed.

**DNS**

In traditional Unix systems, the /etc/hosts file maintains the hostname to host address mapping. This becomes cumbersome as more and more hosts are added. It becomes extremely difficult to keep the table up to date. NIS can be used to share your local host map, but what about the data for remote sites? You could use NIS to do this, but it would be a large table and it would be very dynamic and probably always out of date. To solve this problem the Domain Name Server (DNS) was developed. DNS is a distributed database. It allows local control of the segments of the overall database, like NIS, yet allowing data in each segment to be available across the

entire Internet. DNS varies from NIS in that its information is available to all machines that run DNS. NIS only advertises its information to machines within its own local NIS domain.

The TCP/IP universe is broken into several domains. The root level domains are:

- com – for Corporations
- edu – for Education institutes
- gov – for Government organizations
- mil – for Military organizations

Under each of these domains are subdomains. The domains and subdomains are separated via periods. An example of a subdomain is rockwell.com. The subdomains then act as their own server. They can maintain the list of clients and their addresses that are associated with that subdomain while also providing access to subdomains other than their own.

The subdomains can be broken down even further to give greater flexibility. The name of the domain for our division is cca.rockwell.com. This allows for even greater flexibility.

DNS works in a client server fashion. There is a master for each valid subdomain that provides authority for that subdomain. It can also have secondary servers that act to lighten the load of the master and provide redundancy.

The master domain name server acts not only as a means of serving local hostname to IP address mappings, but also as a way of getting those mappings for clients outside of its own local domain. This means that the administrator need not keep the addresses of hosts outside of its own local domain in its /etc/hosts table or NIS hosts map. The administrator lets the local domain name server query the remote domain name server for this information.

When local clients request information about subdomains outside of their local domain, the domain name server for the local subdomain queries the domain name server for the requested domain. An IP address is returned for the requested host. The local domain name server

in turn, returns this information to the local host that requested it and caches the data for a given period of time to help speed up future references to that same address.

DNS is configured on the local client via the file /etc/resolv.conf. This file lists the domain name that the client is in and the address of the local domain name servers.

We have one master domain name server locally and three secondary servers at other Rockwell sites.

To control the master DNS configuration files, we use RCS and a perl program called h2n. H2n takes a standard flat host file that contains the names of all the clients in that particular domain along with their IP address and builds the necessary DNS files. This has allowed a great deal of flexibility. In the past, DNS was a black magic art to most administrators, with RCS and h2n any of the administrators can now add and delete hosts from DNS.

DNS uses a serial number scheme to determine which snapshot of information a DNS server is using. This serial number can be obtained from looking at the DNS files or by querying a DNS server via nslookup. If you issue the command nslookup –type=soa domain.name, the serial number will be returned. For example:

```
        # nslookup –type=soa cca.rockwell.com
Server:  hwking.cca.rockwell.com
Address:  131.198.213.213
cca.rockwell.com
      origin = hwking.cca.rockwell.com
      mail addr = jldetter.hwking.cca.rockwell.com
      serial = 1994101901
     ...
```

By using a serial number that incorporates the date, you can easily determine if the data that the slaves are providing is up to date information. We use the format *yyyymmddii*, where *yyyy* is the year 1994, *mm* is the month of the year (1–12), *dd* is the day of the month (1–31) and *ii* is an integer (01–99) allowing 99 modifications per day. In the example shown above 1994101901, the data is from the first revision on 10/19/1994. This has proven to be a valuable tool in debugging DNS problems.

**Time Services**

Since we were going to be doing software engineering on the Unix machines via the make utility, it was critical that all of the machines be set to the same time. To accomplish this, we use the ntp protocol. We have setup our Cisco routers to act as ntp servers. The local clients then check their time against the routers once a day and set their local time accordingly. This function is accomplished via a cron job.

**Printing**

We chose to make all of the clients spool their print jobs to one server. This server prints directly to the printers. There were several reasons for this. One reason was that it allowed all of the users to use and see a common print queue. If each client was allowed to print directly to the network printers, the users would not see each others jobs in the print queue. Secondly, some of the printers required a special protocol to submit print jobs. By forcing the clients to send their jobs to one server, we only needed to load the new protocol on that server.

All of the clients print to a fictitious host named prnsvr. Prnsvr is an alias defined within DNS. This allows us to change print servers by changing the alias entry in DNS. We generally have one other machine in the network configured so it can print directly to the printers in case we do lose the primary print server.

As a general rule of thumb, we also try and make every printer a network printer. This means they are not connected directly to a workstation. By doing this, the printers are not dependent on any one workstation being available to print and allow any host on the network easy access.

**CRON**

Cron is a daemon process that runs in the background and wakes up every minute to check for job entries in its crontab file. If it finds a job for the current time, it runs the job. We use this function to do various nightly jobs. The easiest way to do this is to create, what we call, a master_cron_script that runs at a given time every night on every client in the network. This program is stored in a globally accessible area. By adding the functions to our master cron program, we do not have to go to each client and add this functionality to its local crontab file.

We have developed a process where cron starts at 1 AM every morning on each client and executes a program locally called execute_master_cron_script. This program sleeps for a calculated period of time depending on its host address. When awakened, it executes the master_cron_script. We established this back–off algorithm to prevent all of the clients from trying to mount a particular machine at one time.

We use this function for updating some of the local configuration files and performing various daily tasks like time synchronization.

# CHAPTER 7. BACKUP

Backup is one of the most important things that a Unix administrator does. Backup is used as a way of restoring disks in the event that a catastrophic disk drive failure occurs. If we are able to restore files that a user deleted or corrupted, then we consider this an added bonus. In this chapter I will discuss how we do backup at our site.

## What to Backup

The first thing to consider when establishing a backup procedure is what data you want to backup. We decided early on that we would backup selected directories on each client. Backing up all of the disks on the network seemed redundant since one of our other goals was to make the operating system loads identical on all of the clients. It did not seem very logical to be backing up 300 + copies of the operating system when the operating system was available on Compact Disk (CD). If we lost a disk drive, it would be faster to recover the disk from the vendors CD than from the backup tapes. Since we were not going to backup all of the directories on each client it was important that we establish a standard directory structure and enforce it.

We decided to allow the users to write to basically two directories: /usr/users and /usr/apps/projects. The /usr/users directory would be used to house the users home directories. The /usr/apps/projects directory would be used for project data. The projects directory might include Interleaf bulletin boards, Cadre databases, Compass design trees, etc, ...

By default, we backed up these two directories on every client. The users were not allowed to create other directories outside of these two areas. One other area that Unix administrators were allowed to place data in was the directory /usr/apps. We store all of our locally loaded applications like Interleaf, etc, ... in this directory. We selectively backed up these directories. For example, if I wanted the Interleaf application backed up, I would need to make arrangements

with the backup administrator to ensure that the Interleaf directories were backed up on the client that Interleaf was installed on.

This policy seemed to work well as long as the general user was fully aware of this policy. To ensure this, we periodically made this information available to the users via various local news postings and gopher.

**Backup Schedule**

After determining what to backup, you need to determine the backup schedule. We decided to do incremental backups on Monday through Thursday and full backups on Fridays. The type of incremental backup that we perform saves everything that has been modified since the last full backup. In traditional incremental backups, the data is only saved if it has been modified since the last backup, whether it was a full or an incremental. This policy was chosen because many of our applications store their files in databases that are located in several different directories. For a given project, one file might be modified one night and another file in a different directory would be modified on a different night. With the traditional approach the administrator would need to do multiple restores to restore the directory. With our method, the administrator would only need to restore from the previous full and the last incremental.

**Backup Medium**

The next step is determining what type of medium to use. There are several available today: 8mm, 9 track, 4mm DAT and even WORM. The type of medium that you choose will be determined by the following criteria:

- How long you intend on keeping your data
- How much data you will be backing up
- Cost

We decided to go with 8mm because of its shelf life, low cost and capacity. We save our backup tapes for 6 months and then we put them back into a pool to be reused.

**Backup Software**

Once you have determined what to backup, when to back it up and what medium to back it up to, you will need to find a software package to do it for you. You have three options:

- Buy a commercial package
- Use a pseudo internet standard package that is freely available
- Write your own

We chose to go with a commercial package. It was more expensive, but we did not have to maintain the programs.

We purchased Networker from Legato. It was a little pricey but performs well for the money. The program comes with many GUIs and is fairly straightforward. The indexes that it creates are extremely large. Each one of the backup servers has roughly 6 Gb of disk to support 6 months worth of on-line backup data. This may seem expensive up front, but the time that it saves the administrator is well worth it. Having 6 months of backup data on–line allows the administrator to search for files to be recovered. The search shows modification times and sizes for each nights backup for all files. This makes it very easy to determine when files are changed or corrupted which allows the administrator to easily determine which backup to restore from.

We currently are using two Sparcstation IPX's as our backup servers. Each server is capable of backing up 200 clients. We use Exabyte 10i 8mm stackers that write 5 Gb worth of compressed data per tape. With the software compression that Networker provides, we get roughly 10 Gb of data per tape. The stackers automatically change tapes and pretty much runs themselves once they are setup. They do, however, require interaction from time to time to perform restores.

**Restores**

If a user deletes or corrupts a file, he/she must submit a formal backup request via the helpdesk specifying the exact name of the file and date that the file should restored from. We try and guarantee that we will have all restores completed within two working days from the time of the request. Because we store our backup tapes at an off–site facility it usually takes 24 hours for us to receive the tapes before we can begin the restore.

The tapes are stored at a remote facility in the event that out site is damaged by some type of catastrophe.

# CHAPTER 8. MAIL

Another important topic to most users and administrators is electronic mail, also called email for short. Email has become the basic communication medium for most Unix users. Email permits them to send and receive electronic messages from around the world. A great deal of administration is required to ensure reliability. In this chapter, I will address several of the methods that we have found to make email more reliable and easier to administer.

## Where to Deliver Mail

This seems to be the biggest concern to the end user. What is my email address and will it ever change? We have found the best method for handling this to be, having all email distributed to one central location. This greatly simplifies administration of email. Now you only need to worry about the mail server being up and running in order for the user to receive his/her mail. If you allow mail to be delivered to every client, you must ensure that all clients are up and running to guarantee that mail is reliable. This method creates the problem; "what happens when the mail server is down?".

To solve both of these problems we have created what is called an MX record in DNS. It is basically an alias that tells Simple Mail Transport Protocol (SMTP) clients where to deliver mail that is destined for a particular address. We have setup an MX record for a fictitious host called cca.rockwell.com.

With MX records, you can have the mail delivered to more than one host with different weighting schemes. In this particular case, we have one MX record with a low weight that points to the actual mail server. Additionally, we have two other MX records with higher weights that point to other backup mail servers across the network.

When an SMTP client tries to send email to a remote address, it looks up its name via DNS. This lookup recognizes that the destination is an MX record and tries to deliver the message to the lowest weighted MX record entry. If it is unsuccessful after a given period of time, it then tries the next highest weighted entry until it delivers the mail or fails and bounces the message back to the sender.

If the mail server goes down, the mail is queued to one of the backup mail servers. When the mail server comes back on–line, the backup servers deliver the queued mail to the mail server. This ensures that no mail is lost when the mail server is down.

In order for this to work, all of the clients are setup so that the email that they send out has a return address of *user@cca.rockwell.com*. This is accomplished by making modifications to the sendmail.cf file. This involves defining a mail relay in the configuration file.

You must do a good job of informing the user that his/her email address is *user@cca.rockwell.com*. If you do not inform the user, he/she will be inclined to give out their email address as *user@their_client.cca.rockwell.com*. We try and make this common knowledge via an email FAQ that we post monthly to one of our local news groups and is readily available via gopher or Mosaic.

The local clients are still capable of receiving mail messages directly, it is just not the supported method.

Administrators must see to it that old email addresses remain active. This means if you establish an email address for the users email to be delivered to, this address had better remain valid indefinitely. This is accomplished via MX records.

**Sendmail.cf**

We configure our SMTP clients so they deliver all of their mail to the mail server, which then delivers the mail. This includes both internal and external messages. This allows us to easily

make modifications to the mail server that affects all of the out going messages without needing to modify each clients sendmail.cf file

Making all of the out going mail pass through the mail server allows the administrator to easily gather mail statistics for internal and external email as well.

**Mail Spool Directory**

Since we have decided to deliver all of the mail to one server, we also decided to have only one mail spool directory in the network. All of the clients NFS mount this directory. By doing this, users can access their mail from any client. If each client had its own mail spool directory then the users would only see mail that had been sent directly to the client that they were logged into.

The spool directory is located on the mail server as well. If the mail spool directory was not located on the mail server, there could be a situation where the mail server was available and receiving mail messages, but the mail spool directory was not. This situation would lead to lost mail messages. To prevent this from happening we placed the mail spool directory on the mail server.

**Dangerous Email**

There are numerous problems with email that users need to aware of. A user should never write anything in an email message that they would not write in a formal document. It is very easy for an employee to create an inflammatory mail message and send it, since they are not face to face with the end user. The employee needs to be aware that the his/her message can end up being sent to many others. This is becoming a large problem. A user sends a mail message to a person, who in turn sends it to another user, who sends it to another user. Users must be aware of this problem and remember to leave company proprietary information out of email messages.

The other big problem with email is deleted mail messages. Are deleted mail messages actually deleted? If a user receives a mail message and deletes it, it usually is deleted from his mail, but it might remain on the system in a wastebasket or on a backup tape for an extended period of time. This becomes a problem if a company becomes involved in litigation and the deleted mail messages are somehow found by the prosecution. A great example of this was Oliver North and his dealings in the Iran Contra affair. Mail messages that he perceived as being deleted were found somewhere on a system and later used against him in court.

We are currently looking at ways to ensure that deleted mail messages disappear from disk and backup tapes. The easiest way to eliminate them from backup is to exclude the mail directories from the usual backup mechanisms. A separate backup of the mail system to separate tapes that are recycled on a regular basis could be done to avoid the problems outlined above.

# CHAPTER 9. SECURITY

At a conference, I heard someone say "*Security is the reciprocal of convenience*". I tend to agree with this statement. The more secure a network becomes, the more difficult it is for a user to do his/her job. The administrator must weigh the options between security and convenience and come up with a happy medium somewhere in between, so end users can use the system and still feel secure from attack.

We decided to take the security adage "*Prohibit that which is not permitted*". This means we explicitly determine what to allow and deny everything else. In this chapter I will discuss some of the security issues that we have found.

## NIS

The biggest problem with NIS is there is no authentication between the clients and the NIS server. The server will respond to any host that requests information for a particular NIS domain. If users from the outside are able to access your network and determine your NIS domainname, they are free to get all of your NIS maps. This includes your password file which contains the users login names, their passwords, as well as a list of all of your hosts. To prevent this from happening, we use filters on our routers to block NIS traffic to the outside world. The ability for a user to gather NIS data locally is still there, but is restricted to our site. We have taken the approach that we trust our own users and go with that.

## TCP/IP

There are many security problems associated with the TCP/IP protocol. The biggest problem is controlling which machines have access to your clients. You can control which users can login into a client via entries in the password file, but there are no traditional Unix features that limit which machines can try and login to your clients. We would like to limit access to our

clients to machines within our own domain. Additionally, we would like the ability to grant other remote clients access to certain clients within our network.

To control TCP/IP access to our clients we have elected to use a program called tcp_wrapper, commonly known as tcpd. This program was written by Wietse Venema, from Eindhoven University of Technology. Tcpd acts as a service between inetd and the actual TCP/IP daemons that are executed. Normally, when a user connects to a client via an application such as telnet or rlogin, the users process connects to the inetd program and executes the appropriate daemon process.

With tcpd there is an intermediate step before inetd calls the appropriate TCP/IP daemon. The connection is authenticated against a configuration file before it is allowed to execute the appropriate daemon.

There are two configuration files associated with tcpd. One is called hosts.allow and the other is called hosts.deny. The hosts.allow file defines which remote clients can connect to a client and which TCP/IP applications they are permitted to execute. The hosts.deny file defines which clients are denied services.

The configuration file is very flexible, allowing you to specify which clients can connect in a variety of ways. Connections can be specified by domain name, IP address, subnet, netgroup and/or hostname. This allows a great deal of flexibility.

We use these files to restrict access to our clients. We only allow access to our clients from machines within our two local domains. We deny access to all others, except remote clients that have been approved access by the administration team.

Another feature of tcpd that we really liked was the ability to log all TCP/IP connections. It logs the time, the origin, the type of connection and whether or not a connection succeeded.

This log file can then be reviewed in the event there is a security problem or just to determine which clients are accessing a particular client.

We have configured tcpd so connections that are refused spawn a program that finger the remote client, trying to determine who initiated the connection and from which host it originated. This information is mailed to the administrators. This alerts the administrators to any possible unauthorized access.

Tcpd can be used to eliminate the possibility of users creating their own .rhosts files in their home directory. Without tcpd, users were able to create their own .rhosts file that could possibly allow users from other systems to easily gain access to our network without specifying a password. We restrict access to the rsh and rlogin commands via tcpd and a netgroup entry in hosts.allow. Clients from outside of our NIS domain can only access our clients via telnet and ftp. All other functions are disabled. This includes: rlogin, rsh, finger and tftp.

Entries in /etc/hosts.equiv and /.rhosts can also permit unauthorized access from remote clients. These files are used to grant remote clients the ability to login without specifying a password. The hosts.equiv file specifies which remote clients general users can login from. The .rhosts file specifies which remote clients root has this capability from. Tcpd limits which clients can connect, but you still must maintain hosts.equiv and .rhosts to allow the connections to be made.

We again use a netgroup that contains all of our NIS clients. Since we are running tcpd we do not really need to keep these files up to date. We could just place a plus in the files. This would allow any host to connect and let tcpd control which clients could access a client. At our site we keep these files up to date in the event that a clients' operating system is reloaded and we forgot to load tcpd, or tcpd somehow got disabled. This way we are covered if tcpd is not working correctly.

**NFS**

There are several security problems associated with NFS. The most noteworthy, is controlling which remote clients have access to your client via NFS. This is controlled via the exports file. This file lists the various filesystems that may be mounted by remote clients and the names of the remote clients that may do so. If no clients are specified, then all remote clients are allowed access. Once again, we use a netgroup that contains a list of all our local clients to control this remote access.

Another equally important security issue is making sure users that have access rights to your files, are the only ones accessing them. This can be caused by UID and GID mismatches between the NFS client and sever. If I assign the user *"joe"* the right to read a file on my NFS server, the NFS server exports this permission as the UID of "*joe*", not the user name "*joe*". On the NFS client that is mounting my directories, the user whose UID is the same as "*joe*" on my NFS server, will have read access to the file. It is important to ensure that the filesystems that you are allowing to be mounted via NFS have their UIDs synched or the files within these directories be locked down enough to prevent unauthorized access. To prevent this situation from ever occurring we only allow clients within our NIS domain to access our clients via NFS.

**DNS**

The problem with DNS is it advertises your local clients hostnames to the rest of the Internet. This information may allow remote users to gain access to your local clients. If a user does not know that a client exists, it becomes harder for him/her to attack it. To eliminate this problem, you can run an internal name server that lists all of the local clients and can query remote name servers, but does not allow external name servers to query itself. A second name server is then ran on your firewall that does not list internal clients and is accessible from the Internet. This method hides your clients from the Internet.

A problem with this method is the email sent from your local clients must be modified so it appears to come from a client visible from the Internet, or you will not be able to receive email. We are currently working on providing this type of service via our firewall.

**Root Access**

An important issue in controlling the security of the network is limiting access to the root account to a controllable number of users. We limit this capability to the local administrators only. Having access to the root login is a problem. It allows users the ability to override any security measures that you might have put in place. Also, it makes it extremely difficult to diagnose problems when multiple people have the ability to modify configuration files on a client. If a user needs root access then we give them sudo privileges for a given command on a given client.

**Sudo**

Sudo is a program written by Cliff Spencer. It permits users to execute commands as root without having access to the root account. Access to sudo is controlled via an access list. Access to sudo is controlled by username, command and client. The administrator can grant access for a given command, to a given user, and on a given client. To execute a program as root the user specifies:

*sudo command –command_options*

The user is then asked to enter his/her password to authenticate the user. This is an added protection so users cannot execute commands as root from a terminal left unattended. The program allows the user to run an additional sudo command in the next 15 minutes without having to re–issue their password.

Sudo logs the username, the time of execution, the command and the client that it was executed on. This way, the administrators can review the list on a regular basis to monitor what

the non–administrators are doing. This is essential to ensure that the users are not making modifications that will compromise security.

It is very important to limit which commands the user has access to. If a user is permitted access to the shell commands for example, he/she can spawn a shell as root and none of the commands executed within that shell will be logged. We limit the commands that users can run to the minimum necessary for them to do their job. These usually include the ability to kill hung processes and restart license servers.

### Passwords

Users need to be careful when selecting their password. If the password is easily guessed, then hackers can easily gain access to their account. The password should not be their name, a relatives name, their dogs name etc... It should actually not even be a word in the dictionary.

We periodically run a program, written by Alec David Edward Muffett, called crack to determine if a users password is guessable. If it is, we make the user change his/her password. We do this so that if someone from the outside does manage to get our password file, it will be harder for them to gain access.

### Modem Connections

To prevent any possible security problems with modems, we do not allow them to be connected to the local clients. Users that need access to modems are forced to use our Micom system to connect to a modem pool, that allows them the ability to connect to external sites. If they need access to their systems from the outside world they are encouraged to use the external access methodology described in the next paragraph. We understand that there will be exceptions to this rule, but we have made an effort to limit these connections to only those which make business sense and/or are the only solution available.

**Access from Remote Sites**

Users periodically require access to our facility from a remote site. We allow them two mechanisms:

- Dial–in via a modem bank
- Direct Internet access via secure id cards

The dial–in method allows them to use a modem from a remote site and connect to a secure modem pool here at Rockwell. The user must provide his/her login name, a password from a secure id card and his/her local password. Once the user is authenticated, he/she is then connected to our Micom switch that allows the user access to any local client. We use Security Dynamics secure id cards. They are a credit card sized device that provides a password to the user that changes every minute. Software within the modem pool then determines if the secure id password specified is correct for the given time.

To allow direct access to our network from remote Internet sites, users must first login to our bastion host. The user is required to give his/her login name on the bastion host and a password from a secure id card. If the user authenticates correctly then the user will be allowed to connect to any local client. Once again, we will be using the secure id cards from Security Dynamics.

**External Access**

We have changed our stance on external access from restricted to full outbound access. Users can now use Mosaic to telnet and ftp files to/and from the network. Our previous policy allowed the user full access to some of the Internet news groups and restricted access to ftp and telnet. Users who needed ftp and telnet access were asked to fill out a form and have their manager sign it. They were allowed to login to one client on the network that was allowed to connect to the Internet. With the recent introduction of Mosaic, we have decided that denying the users

access to the Internet was becoming too much of a liability. There was data that the users needed to access that was being denied.

We allow the users to ftp and telnet files to/and from the Internet via proxies on the bastion host. They first must connect to our bastion host. The bastion then connects them to the location that they wish to connect to. The bastion serves as an authentication device and hides the path to the local client.

**Firewall**

Once we had decided that we were going to allow the general user full access to the Internet, we determined that we needed a firewall to ensure our safety. We looked at several different schemes and decided to use the screened subnet approach (Figure 9.1). In a screened subnet configuration, Internet traffic is allowed through the screening router between the Internet and the screened subnet, but only to the screened subnet. All traffic to the internal network is blocked. The internal network is then only allowed to talk to the screened subnet via the screening router between the internal network and the screened subnet.

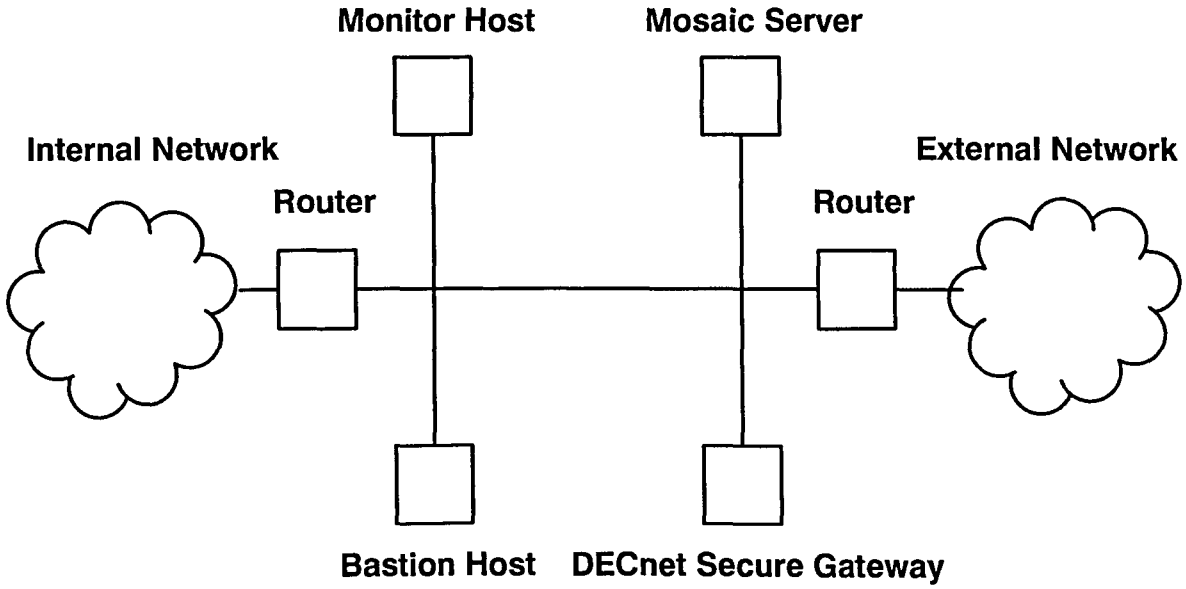


Figure 9.1 Firewall configuration.

# CHAPTER 10. USER ACCOUNT MANAGEMENT

One of the more tedious and time consuming tasks of an administrator is the user account management. This task involves registering and deleting users, changing users passwords and adding and removing users from different groups.

## Account Creation

To simplify the account management dilemma, we have one person that creates and deletes all accounts. To get an account, a user is required to fill out a form and have his/her manager sign it. We keep this form for our own records.

We have a set of scripts that is used to ensure that all users accounts are created the same. The scripts add the users entry into the NIS password map and create the account. The appropriate login files are then copied to the users account and the user is assigned a unique password.

## Account Deletion

As users leave the company their accounts need to be disabled. When this happens we pull the users registration form and determine who his/her manager is. We contact the manager to determine if any of the data in the users account will need to be saved. If so, we ask for a person to assign this data to and transfer the files and ownership to that person. The users account is then deleted from the system.

In the future, we would like to create a program that would give us updates from the company's payroll system. This would allow us to readily see changes to employees statuses. Currently we have no way of knowing that an employee has left the company other than by word of mouth. It would be nice to get an updated list on a periodic basis that could be used to determine whether or not an employee has left the company, so we could disable his/her login.

### Group Administration

When we create a group, we designate one user as being the owner of the group. We only make modifications to that group, that are approved by the owner of the group. This involves the owner sending an email message to the person that modifies the group access, telling them what modifications he/she wants made. This ensures that only the people that should have access to a particular group do. Without this handshaking, we were having users requesting to be added to groups that they were not supposed to be in.

### User Login Files

The users login files are assigned when the account is created. We do this so users start out with a default set of login files. We wanted to avoid the users login files changing drastically. Users tend to share one anothers login files and if they appear different they become confused. We try and keep them as simple as possible and avoid making modifications to the login file so that they can run particular applications. Instead, we mandate that applications be accessed via wrappers. I will go into more detail on this in the chapter on applications.

### Quotas

We have elected not to use quotas. The users home directories are usually stored on their local disk drives and we leave the administration of the size of a users home directory to the users that are co–located on the local disk. We have a program that runs out of cron that checks the amount of free space on each clients /usr/users directory. If the directory drops below a given percent free, a program runs to determine the top disk space users on that client. Then an email message with this list, is sent to the users that are located on this local disk drive. This peer pressure approach has worked rather well.

**UID Assignment**

Since we will be sharing files with the other local divisions, we determined that we should divide up the UID's. We broke the UID's into two ranges. The UID's 1000–15999 were assigned to our division and the UID's 16000–32000 were assigned to the other division. When a user needs an account he/she is assigned a UID based upon his/her division. This way when we shared files via NFS, we could ensure there would not be any UID mismatches. The UID's 0–999 were reserved for use by applications and special Unix logins within both divisions.

There have been several pushes by our corporate office to do this on a company wide basis. Until that happens we are going to stick with what we have now. An ideal solution would be for a user to be assigned a username and UID when he/she was hired at Rockwell. This would make the assignment process much easier.

**User Login**

We have standardized on a common login naming convention. Users were assigned logins based upon their name. The first letter in their login was their first initial, the second letter was their middle initial and the last six characters were the first six characters of their last name. As an example the user Joe W. Smith would have a login of jwsmith. Exceptions were handled by appending numerals to the end of the login name in lieu of characters from the last name of the user.

This provides an easy way for users to identify one another.

# CHAPTER 11. USER SUPPORT

When I was the sole administrator, it was acceptable for every user to contact me directly for problem resolution. As the number of administrators and users grew, we needed a better process of getting the problem reports to the appropriate administrator. To accomplish this we implemented a helpdesk.

## Helpdesk

We assembled a team of administrators from various groups within our organization and evaluated several commercial products. We have set a long term goal of using a product from SIO technologies called Multihelp. The application runs on the Vax and has GUI clients for the other systems. This GUI generates an email message and sends it to the application on the Vax. The product is scheduled to be released later this year.

For a temporary solution, we found a program called request, written by James Sharpe and Shawn Instenes from Lawerence Livermore Labs. It is written in Perl and is easily ported to most Unix platforms. It is a simple program that prompts the user for answers to questions and logs them into a dbm database. Each call is assigned a unique sequence number that is used to distinguish the calls. When a user opens a request, an email message containing the users call, is sent to all of the administrators. The administrators then use a menu based program to assign, update, view and close the various calls. Each time that a call is modified the entire contents of the call are emailed to the originator and all of the administrators.

Not only is this a great way of getting the appropriate problem assigned to the right administrator, but it also is a great teaching, logging and monitoring tool.

It is a training tool because every call that is updated or closed is sent to all of the administrators. The administrators can then read through the calls and learn how to perform various tasks.

It is a good monitoring tool because it saves all of the information to an ascii file that the administrators can review at a later date. This comes in particularly handy when subbing for another administrator on a subject that you are unfamiliar with or for remembering how you performed a particular task in the past.

It is a good monitoring tool because it allows the senior administrators to monitor the other administrators and make sure that the administrator is getting the correct information to the user.

The program has also given us a methodology for measuring what we do, which will help us justify more manning in the future. One of the largest problems that an administrator faces is showing what he/she does to upper management. Having the helpdesk has given us benchmarks to work off of. We can now prove that we were busier than we were last year and that calls are going unanswered.

# CHAPTER 12. LOCALLY DEVELOPED SOFTWARE

As we started to add more workstations, we began to get more and more demand for shell scripts and programs to do various tasks. These programs were either written locally or pulled down from the Internet. We decided to place these programs into a common directory that users could easily access. We decided upon the directory /usr/local.

## /usr/local

The /usr/local directory is NFS mounted by the clients from one of the /usr/local servers. We have designated several machines as /usr/local servers. One of the servers is designated as the master and all updates are made to it. A program is used to push the updates to the slaves. This program can be run manually during the day or by cron at night. We have added this function as a cron job on one of our machines to ensure that the /usr/local trees are always up to date.

In this directory, we place all of the software and shell scripts that we write locally. These include programs for both the user and the administrators. The most important directory in this tree is called /usr/local/bin. It contains most of the scripts and wrappers that a user will need to do his/her job. We try to ensure that all of the users paths are set to this directory.

# CHAPTER 13. APPLICATIONS

Applications are the driving force behind computers. Without applications, there would be no use for computers. The Unix administrator generally spends a great deal of time administering applications. The administrator is usually responsible for installing and maintaining the applications as well as answering questions and solving problems with using the application. I will cover two topics in this chapter, how to handle setting up a users environment so that the user can easily run applications and where to physically locate the applications.

## Providing Easy Access to the Applications

When we had a limited number of applications, we allowed applications to make modifications to the users login files. We eventually decided that this was a bad idea. We now mandate that applications be invoked via a wrapper program stored in /usr/local/bin.

A wrapper program is a shell script that sets up the necessary environment to run an application and then calls out the actual program. This has several benefits. One, it allows users to access applications without modifying their logins. All that is required for a user to be able to run the application is for the directory /usr/local/bin to be in their path. Secondly, it allows the administrator the ability to move software throughout the network without affecting the users. With the old setup method where the programs would modify the users login, when the program was moved, then the users login would need to be modified in order for them to run the application. With the wrapper the administrator can move the application, modify the wrapper script and the user will not see a difference.

Wrappers also reduce the time it takes for a user to login. With the old method, if a user wanted to run several programs, then his/her path would need to be modified to include all of the necessary directories to run the applications. Every time that the user logged in, irregardless

of whether he/she wanted to run a particular application, the environment would be setup and would add extra time to the login process. It makes the process longer because Unix requires that all of the directories in a users path be mounted when a user logs in. This takes time and is unnecessary.

Using wrappers has also helped us get around path length restrictions. The users search path can only be a given number of characters long. When the user modified his/her path variable in order to be able to execute multiple applications, he/she would often hit this limit. The wrapper overcomes this problem.

The one limitation to using a wrapper involves applications that have several small programs that all must be be run from the command line. In this case, the administrator would need to maintain a wrapper for each one of these programs.

**Where to Physically Locate the Applications**

Initially, we loaded applications onto selected workstations throughout the network. We would pick a workstation with ample disk space and load a particular application on to it. The clients would then look to this workstation for that particular application.

This worked on a small scale with a handful of workstations on a limited number of subnets, but as the number of clients, applications and networks grew, this scheme did not scale.

Not having ownership of the servers eventually became a big problem as well. Since we did not purchase the workstations, we could not control the amount of loading on a server. We were constantly running out of disk space on applications workstations. The local groups would fill up the disk drives and we would not be able to load new applications. We also ran into problems where one group would move offices and need to have their workstations shutdown to be moved, this would then deny other users access to the application loaded on their workstation.

Our solution to this problem was buying dedicated application servers. With the dedicated servers were able to control the loading and availability of the servers better. We no longer have to worry that a general user had reset the power on his/her workstation because they could not login.

We purchased three Alpha AXP 2100 servers from Digital Equipment. Each server is equipped with 128 MB of RAM and 20 Gb of disk. The servers are housed in our computer room so we can prevent people from having access to them, and allowing them to be on uninteruptable power.

To reduce the loading and dependence on the network we decided to multihome the servers. Multihoming involves installing multiple ethernet cards on a server and connecting it to multiple networks (Figure 13.1).

We assigned each interface on a server both a unique name and a common name in DNS. In figure 13.1 the three network connections for the server are each assigned a unique name, *server–network(A–C)*, and a common name, *server*. A client that references the server by the unique name will be returned the address of that particular interface via DNS. A client that references the server via the common name will get a name based on the following scheme. If the client is directly connected to a network that the server is also connected to, then DNS will return the address of the common network interface. If the client is not connected to a network that the server is on, then DNS will use the round robin feature in DNS to assign an address.

The round robin feature in DNS causes DNS to cyclically step through the three addresses listed for the common name. This provides a rudimentary form of load balancing.

By specifying the common names for the servers in the local clients automounter configuration files, we can force the local clients to communicate with the servers on their local net-

work. If the client is not located on network that the server has a connection to, then it will be assigned an address from one of the other servers interfaces.

By the clients communicating to the application servers via their local networks we are hoping to reduce our dependency and loading on the routers.

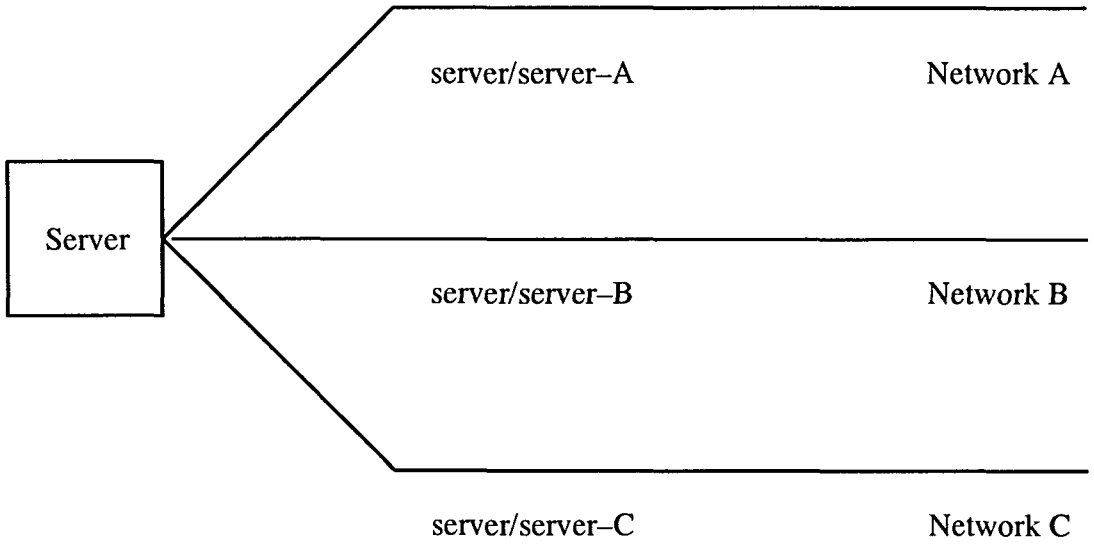


Figure 13.1 An example of a multihomed server.

# CHAPTER 14. INTERNET ACCESS

The recent explosion of information accessible via the information superhighway has lead us to rethink our internet access policy. In the past, we restricted access to the Internet to only one client in the network. Users that needed Internet access were then required to fill out a registration form signed by his/her manager to gain access to that host.

With the recent success of Mosaic, we needed to rethink this policy. Denying users full access to the Internet became a liability. Our engineers were not getting access to the same data that our competitors were. We had a great deal of demand from users that also wanted to do business with customers via the Internet.

So we decided to allow users outbound access to the Internet. This access included:

- Electronic mail (email)
- File transfer (ftp)
- Mosaic
- Internet news (nntp)
- Telnet

## Electronic Mail

Inbound email is directed through the firewall to the bastion host. A process on the bastion host then delivers the mail internally. This process hides the internal clients from the outside world and protects them from any sendmail insecurities.

## Ftp

Ftp allows the user the ability to retrieve and send files to a remote computer. We provide external ftp access via an ftp proxy server on our bastion host. Users must first connect to the

bastion host and then they can connect to an external client. This process hides the internal host from the external client and logs all transactions.

### Mosaic

We have compiled our Mosaic client to use the socks libraries to tunnel through our firewall. We are hoping to implement a proxied version as soon as it becomes available.

### Internet News

Internet news is allowed to pass through the firewall to the local news server. Users read news from the internal news server. The news administrator determines which news groups are appropriate and only allows those groups to filter through the firewall.

### Telnet

Telnet provides the user with terminal emulation to external clients. We provide external telnet via a telnet proxy server on our bastion host. Users must first connect to the bastion host and then they can connect to remote clients. The proxy server hides the internal client from the outside world and provides logging of all activities.

# CHAPTER 15. FUTURES

One of the problems associated with being a Unix system administrator is that nothing is constant. This means your job is never done. As soon as you implement something, a better solution is eventually found and you must adapt to this new technology.

There are several projects that we are currently working on:

- A mechanism to manage the configuration files on all of the clients
- Implement DCE
- Conversion to a variable length subnet mask

## Configuration File Management Mechanism

We are trying to develop a database application that will control the configuration files on the local clients. The program will allow the administrator to make changes to a central database, that will propagate changes to the local clients. We are working on a prototype system that uses RCS, the Unix make utility and the rdist program. Rdist is a program written by Michael Cooper from the University of Southern California.

## Implement DCE

We foresee the need to have non–Unix clients share resources with Unix clients. To achieve this we would like to configure the Distributed Computing Environment (DCE) on our network. DCE will provide us with the necessary tools and protocols to allow the non–Unix and Unix clients to easily and securely share resources.

## Conversion to a Variable Length Subnet Mask

In the near future we will need to migrate to a variable length subnet mask to accommodate the addition of more than 254 TCP/IP hosts to a single subnet. This will involve switching the internal routing protocol that we use. We currently use Cisco's Interior Gateway Routing Pro-

tocol (IGRP) to perform internal routing, in order to transition to variable length subnet masking we will need to migrate to either Cisco's Enhanced Interior Gateway Routing Protocol (EIGRP) or the Open Shortest Path First (OSPF) protocol. We will probably switch to EIGRP because it is an easy transition from IGRP and requires less enterprise wide planning.

# CHAPTER 16. CONCLUSIONS

I have heard several people use the adage, "The good thing about Unix is that you can do things fourteen different ways, and the bad thing about Unix is that you can do things fourteen different ways". An administrator must be sure that he/she does not fall into to this trap. Develop a plan and stick to it, do not change the way that you are doing things just because someone told you of a different way. Change is good, but continuous change leads to confusion.

There are several concepts to keep in mind:

- Keep it as simple as possible. The simpler the solution the better.
- Make sure your solutions are scalable.
- Work as a team.
- Be consistent.
- Stay current on new technologies.
- Don't reinvent the wheel if someone else has already invented it.

I cannot overemphasize the concept of simplicity and consistency enough. Make every machine look and act the same. It's easier on you and the user.

In this paper, I have presented some of the topics that I feel are essential to Unix administration. These concepts might not be the perfect solution for all Unix environments, but they have proven to be the best possible solutions for us.

## REFERENCES

[1] Albitiz, P.; Liu, C. *DNS and Bind in a Nutshell*, O'Reilly & Associates: Sebastopol,CA, 1992.

[2] Stern, H. *Managing NFS and NIS,* O'Reilly & Associates: Sebastopol,CA, 1991.

[3] Hunt, C. *TCP/IP Network Administration,* O'Reilly & Associates: Sebastopol, CA, 1991.

[4] Costales, B.; Allman, E. *Sendmai,* O'Reilly & Associates: Sebastopol, CA, 1993.

[5] Peek, J.; O'Reiley, T.; Lukides, M. *Unix Power Tools*, O'Reilly & Associates: Sebastopol, CA, 1993.

[6] Nemeth, E.; Snyder, G.; Seebass, S. *Unix System Administration Handbook*, Prentice−Hall: Englewood Cliffs, New Jersey, 1989.

## ACKNOWLEDGEMENTS

I would like to thank all of my team members at Rockwell: Connie Calderon, Tom Clark, Jennifer Dayton, Alan Hanson, Mike Irvine, Ken Kroymann, Dave Marshall, Len Struttmann and Chuck Wilson for their help on bringing the network to what it is today.

I would also like to thank my loving wife Tina, for her support, understanding and tolerance during the whole process.