Automating Request-based Software Distribution

Chris Hemmerich – Indiana University

ABSTRACT

Request-based distribution of software applications over a network is a difficult problem that many organizations face. While several programs address this issue, many lack features required for more sophisticated exports, and more complex solutions usually have a very limited scope. Managing these exports by hand is usually a time consuming and error-prone task. We were in such a situation when we developed the Automated Netdist program a year ago.

Automated Netdist provides an automated mechanism for system administrators to request and receive software exports with an immediate turnaround. The system provides a simple user interface, secure authentication, and both user and machine based authentication. Each of these is configurable on a package-by-package basis for flexibility.

Netdist is a modular service. The user interface, authentication and authorization are independent of the export protocol. We are currently distributing via NFS, but adding an additional protocol is as simple as writing a script to perform the export and plugging it into Netdist.

Introduction

At Indiana University, we have a large, extended, and heterogeneous Unix workstation presence administered by many different departments and organizations. The Unix Workstation Support Group (UWSG) is charged with supporting and advising the administrators of these machines. This includes offering a variety of systems administration classes, negotiating and maintaining site licenses with various vendors, distributing a large variety of unix software, and maintaining a Unix Users Group. We also provide traditional phone, e-mail, face-to-face, and extended contact support services.

At the time Netdist was developed the UWSG consisted of six members, five full-time and one part-time, distributed across two campuses. This is a lot of ground for such a small group to cover, and we are always looking for ways to increase the efficiency of our services without sacrificing quality or our customers' satisfaction. In early 1999, I was given the task of re-working one of our least efficient and least reliable services, the request-triggered distribution of software via NFS.

Software Distribution

The UWSG distributes a large and varied collection of Unix software, to a widely varied audience. Software export laws, license agreements, pre-existing vendor-supplied distribution systems, varying security requirements, and machines with slow or non-existent network connection all work together to fragment our software distribution into the following components. These components are the environment out of which Netdist grew, and their attributes greatly influenced the development on Netdist.

- Anonymous FTP
- Secure HTTP Access
- Open NFS exports
- Request based NFS exports
- Media Checkout
- Vendor Specific Distribution Tools

Anonymous FTP

The majority of the software we distribute is via anonymous FTP. We maintain a large (150GB+) site at ftp.uwsg.iu.edu. This FTP site is our preferred method of distributing files. This archive is available to the world and contains mirrors of most of the large Linux distributions, a CPAN mirror, a mirror of the Linux kernel archives, various freeware programs and utilities, and system patches. The benefits of this system are many:

- It supports many concurrent users, and is limited only by our current hardware and the university's bandwidth.
- It is available to anyone in the world with FTP access.
- It is easy to use, and the ratio of support incidents to usage is orders of magnitude smaller than any other service we maintain.
- It is efficient to administer, requiring little maintenance.
- It is a very well known system, and administration can be shared or passed on easily.

Unfortunately, anonymous FTP is not a viable tool for all of our distribution needs. Following are several of it's limitations that have forced us to distribute packages using alternate methods.

- Access control is very limited. We maintain a distinction between Indiana University (IU) clients and non-IU clients. Anything beyond this becomes much more difficult to administer as the target audience shrinks and the complexity of administration increases.

- Security is poor. FTP has a long history of security deficiencies, vulnerabilities, and exploits. We don't make any sensitive data available via FTP, allow authenticated logins, or allow write access.
- We provide operating system media for system installations, and most operating systems aren't able to be installed over an FTP connection. Several Linux distributions support this, but we must also provide Solaris, HP-UX, IRIX, and other Unix flavors that do not.
- FTP sites cannot be mounted as local file systems. We offer several large software packages, and users might not have the free space to download a large software package, uncompress it, and then install it.

Secure HTTP Access

For software that needs to be firmly restricted to IU affiliates we use an Apache web server running SSL and an in-house mod_perl module that allows us to securely authenticate users against the University wide kerberos database. This system is useful for distributing smaller packages and we use it to distribute export-restricted security software, as well as IU-specific license codes for several software packages that we have licensed.

This service addresses the security failings of anonymous FTP, but doesn't address the concerns of installing operating systems or dealing with large software packages. In addition, the server installation and configuration is much more complex than anonymous FTP, and the overhead on the server is greater.

Open NFS Exports

We maintain open NFS exports of several Linux distributions for network installs and updates. This is a convenient way for administrators to set up new machines over the campus network. As open NFS exports lack any real security, we do not export any other software this way. Indiana University does block incoming NFS requests from outside the school network, so the exposure is relatively small.

Request Based NFS Exports

At the time I began this project, we were exporting Star Office (before it was purchased by Sun) and Sun's Workshop Compiler Suite through NFS on a per-request basis. This allowed us to control who had access to the packages, while giving our customers the convenience of installing these packages as if they were mounted locally as a CD image. Our customers were happy with this service, but we were managing it by hand which was inefficient and error-prone. I was assigned the task of replacing this service, when we learned that several new software packages would be added to it.

Media Checkout

We maintain a large collection of OS and application CDs that are available for checkout by request. This has been a very labor intensive service for us, as we must manage the physical storage of these CDs, make copies of popular requests, keep track of who has checked out what, and gently remind admins when they've kept a CD too long. A separate project to resolve the inefficiencies in this process is currently wrapping up, with wonderful results.

This process is secure, and convenient for administrators to install from. It also provides those with only a modem or no network connection a means of acquiring software. It is, however, horribly inefficient, generally requiring 15-30 minutes of work per request. We must respond to the request, obtain the physical media, schedule a pickup time, meet the customer, check their identification, register the checkout into our tracking system, and then collect the software.

Vendor Specific Distribution Tools

We use vendor-specific software distribution tools from both HP and SGI. Both of these tools use remote procedure calls for transferring software patches, updates, and new packages. These tools are convenient for administrators on campus to use, and required little administration. Both of these tools are proprietary software, and limited in the software they can serve and the clients that can access them.

We use the IRIX Network Distribution server to distribute OS and Software updates as well as new packages. In order for a machine to be able to access the server, it must be listed in the servers .rhost file. This file tends to get rather large and outdated as time progresses, and must be pruned periodically.

The HP Software Agent Suite allows us to make HP depot files available over the network to HP machines on campus. Depots are automatically available to the whole campus, but implementing more granular permissions is a non-trivial task. The system uses a graphical interface to make patching and install software painless.

Request Based NFS exports: The problem

Under the old system, an administrator that wanted StarOffice or Sun WorkShop would send an e-mail message to our group account containing the software requested and the target machine's hostname. Upon receiving the message, we would verify that all of the relevant information was included, and if necessary request any information that has been left out. Then we would edit the NFS exports file by hand to include the client machine and a comment listing when the export expired. We then re-initialized the exports and notified the user that the export would be valid for three days. Once the export had expired, we removed the client hostname manually and then re-initialized the exports to force the expiration.

While this system worked, we knew it should be better. The ease of installation from an NFS export and the wide install base of NFS clients were nice, but there were problems with the system:

- Our response time to customer requests was inconsistent. Requests were only processed during business hours, and if we were teaching a class or were otherwise busy, it could take an hour or longer to process a request.
- The process was error prone, especially while editing the NFS export file by hand. Some problems we experienced were mis-typing hostnames, corrupting the NFS export file, and forgetting to re-initialize the exports. These were all easily and quickly resolved, but increased response times and introduced short service outages.
- Handling requests interrupted other tasks. While handling a request usually only took five to ten minutes, the interruptions could be difficult to deal with. We place a high priority on response time, so even important tasks would be interrupted to handle a request.
- The system was not scalable. The time required to process exports and the chance of error both increased linearly with the number of requests. With anonymous FTP adding an extra 100 users, would not be noticeable, but the same increase in NFS requests would have required us to open another full time position.
- Removing exports was problematic. It was easy to forget about an export after three days and not remove it from the exports file. There were also occasions where a hostname was added without comments, which made it difficult to determine if an export should be removed.

Requirements in a Replacement

After cataloging the strengths and weaknesses of each of our distribution systems, we constructed a list of features required in a replacement:

- Automatically process user requests.
- Automatically clear expired requests.
- Only answer requests from IU machines
- Must not require any proprietary software on the client machine.
- Must be easy to use and to administer.
- Logging must be complete.
- Be able to authenticate the user, preferably through our existing Kerberos database
- Any existing solution must be available free of charge.

With such specific requirements, we had serious doubts about finding an existing software package that met, or even attempted to address, our needs. We had been prepared to develop the solution from the beginning, and already had several ideas about how to implement the service. Before proceeding with this implementation, we did perform a limited search for existing software that met our needs.

We searched several common search portals, such as dejanews (now deja), yahoo, and altavista. Plodding through hundreds of results from various search terms provided very few software distribution systems, and all of these were designed to face problems different from ours.

Failing to find an existing solution to our problem, we began work on an in-house solution. We wanted to maintain the benefits of a file system based export, and since only a handful of machines on campus had AFS or DFS, we decided to stay with NFS as the means of exporting the software.

We decided to leverage our experience with the secure web server interface to our kerberos database to provide secure authentication, and a web based request form. This gave us the ease of use of NFS, the security of HTTPS through user authentication, and nearly ubiquitous installation of our clients, i.e a web browser and NFS client.

With export requests and authentication handled through HTTPS, and the software exports done via NFS, we needed something to tie the pieces together. We looked at several different models before narrowing our selection down to either a system of setuid root C programs, or a client/server interface, with the web server acting as the client to a custom designed server. We implemented rough versions of both of these, and did some performance and security testing. From the results we decided to go with the client/server model.

A Brief Version History

The initial version of Netdist was designed to alleviate the immediate problem with exporting StarOffice and Sun WorkShop. We were eager to release the system, and as such were unable to implement several of the more ambitious planned features. The initial release in June, 1999, included the following features:

- Automated user authentication.
- Automated export initialization and expiration.
- Ability to include or exclude sub-networks of machines by IP number on a per-export basis.
- Variable export durations for different packages.
- Core API for implementing common methods.
- 24/7 availability, with immediate response.
- A client/server model with PGP based encryption.

The initial release went well, with positive user feedback. As it became certain that we would be adding ApplixWare and Island Office to our available software, we returned to Netdist development. In November 1999, we released a much improved Netdist with ApplixWare and Island Office added to the list of available software. The improved features included:

- Expanded API, and more encapsulated system overall.
- Ability to limit exports to specific users.
- System for managing export documentation.
- Ability to control exports from multiple machines from a single web interface.

- Ability to accept plug-ins for exports of protocols other than NFS.
- More efficient coding, faster response time on requests.

Since this revision, Netdist has been a nearly ideal service to administer. It has required intervention less than a handful of times, and the logs show continued use.

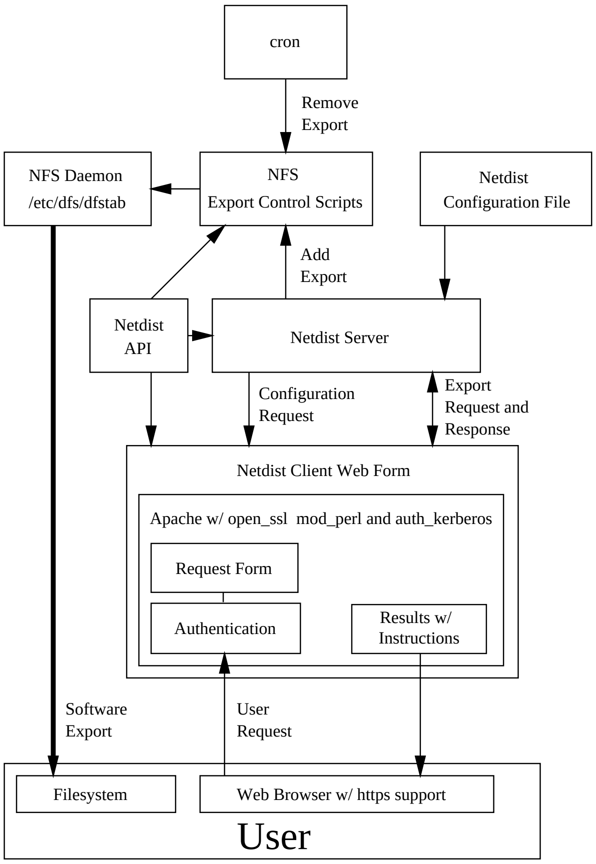

Figure 1: Information flow between Netdist components.

Architecture

The Netdist program is a system of four modular components that can each be upgraded and extended as needed with minimal effect on the other components. The components consist of an API, client, server (with configuration file), and export control scripts.

Briefly, the client is a web form that authenticates and authorizes each export request. If the request is validated, the client passes an encrypted request to the server. The server processes this request, and calls the appropriate export control script, depending on the export protocol of the software requested. The results of these exports are then returned to the client for presentation to the user. The API ties all of these sections together by providing common procedures and data structures. Upon startup, the server reads a configuration file that defines the valid export types, export restrictions, and other settings described below, and shares this information with the client. See Figure 1 for an illustration of how these different components work together to manage software exports.

The API

The API is written in object-oriented Perl, and is available to the client, server, and access control scripts. The API is composed of six modules and contains any code that should be accessed from more than one component in order to increase efficiency and to ease communication between the different components.

The API provides procedures for accessing and manipulating the data structure created by reading in the configuration file. This allows each component of netdist to manipulate this data structure. The API also manages all logging in order to guarantee comprehensive and compatible log entries from events logged in different components. API functions are also available for creating unique temporary files and performing date comparisons.

The Client

The client, or user interface to Netdist is a collection of mod_perl scripts available on an Apache Server with SSL. We use a custom Apache module to authenticate users against our kerberos database, but other Apache authentication schemes could be used. Once the user is authenticated, they are presented with a list of the exports they are authorized to request. This list comes from the data structure which the server creates from the configuration file, and is later passed to the client upon request.

The user then enters the hostname he wishes to export the software to, and selects the desired software packages. The client performs taint checking on the data before processing it to look for illegal hostnames, and attempts to compromise the web server. The client then makes sure the the hostname entered is an IU machine, performs a reverse DNS lookup to make sure the hostname and IP number match, and confirms that the selected exports are valid for that host.

If these conditions are met, the client opens a connection to the server and submits an encrypted request for the approved exports. The client waits for a success or failure response from the server and returns a message for any error that may have occurred. Otherwise, the client returns success, with links to instructions for accessing the exports, and an expiration date.

The Server

The server is a long-running Perl daemon that must run as a user privileged enough to manage the exports. In our case, NFS is used and the daemon runs as root. At startup, the daemon processes the configuration file, and then listens for connections on a UNIX or TCP port, depending on how it is configured. The server uses a PGP authentication scheme to ensure that it only processes requests from the client. There are two valid requests a client can send to the server. This first is for a copy of the configuration data structure, and the second is to export a software package. Configuration requests are responded to, and export requests are mapped to the appropriate export control scripts, and results of the export are returned.

The Export Control Scripts

Each type of software export requires its own set of export control scripts. Each set consists of one script the server calls when requesting a new export, and a second script to clean up expired exports, which should be run by cron or another scheduling utility. In the case of NFS, the exports controlled by Netdist must be specially formatted to allow the control scripts to process them. Other exports can be listed normally after the final Netdist export. On a Solaris machine the formatted entry looks like this:

```
#@work50            begin Island exports
#host1.indiana.edu  user1 2000/09/18
#host2.indiana.edu  user1 2000/09/18
#host3.indiana.edu  user2 2000/09/19
##work50            end Island exports
share -F nfs -o ro=host1.indiana.edu:
    host2.indiana.edu:host3.indiana.edu:
    filler.indiana.edu /is/netdist/ws50
```

A comment entry is added with each export containing the hostname, the username of the requester, and the date of the export. The comments not only allow the expiration scripts to determine when an export has expired, they also make the export file more informative to human readers.

Configuration

We designed the Netdist program to be highly and easily configurable. From working with many Unix programs, we felt that a simple yet powerful plaintext configuration file was the best way to achieve this. The format of the configuration file, and the API calls for processing it, were the first sections to be implemented.

We had an initial list of values to store in the configuration file, which grew as we developed the other sections. We attempted to extract every configurable value to this file, but there are a few values that we missed, and will extract in the next version of Netdist. The Netdist configuration file defines the available exports and their properties expressed as key-value pairs. The valid keys are shown in the subsections below.

export

This value is a short string that uniquely identifies the export within the system. It must be defined for each export, and is only displayed in the logs. Examples: (star51, isl60)

text

This value is a text string that identifies the export to a Netdist user. This should contain the full title of the software package, the version number, and the platforms on which the package is valid. This parameter must be defined for each export. Example: ('StarOffice 5.1 (Solaris|Linux)')

type

This value defines the type of export the package is. Currently the only valid choice is nfs, but if other export protocols are added, this key will be used to determine which scripts to call to perform the export.

host

The host value sets the host on which the export is located. The host must have a configured and running Netdist server. With this value a single web interface can manage exports from multiple servers.

port

This value defines the port on which the appropriate Netdist server will be listening. This allows Netdist to be easily introduced into environments without worrying about any given port being free.

duration

The duration value defines the number of days an export is valid before being expired.

manual

This value defines the name of the file to link to when providing instructions for acquiring and installing an export once it is approved. Instructions should be in HTML format.

users

The users value defines the list of users allowed to access this export. A value of "*" will allow all users to export the software. Alternatively a list of usernames can be provided. Only users able to export the software will see it as an option.

allow

This value defines IP ranges to which the software can be exported. Each entry is expressed as a subnet mask, and multiple entries are allowed for each export.

deny

This value defines an IP range on which the export is not permitted. This can be used to remove part of a previously allowed network, including hosts and subnets. The syntax is the same as the allow parameter, and again multiple entries are allowed.

Configuration: An Example

The first export defined in the configuration file is a default. Any keys that are unassigned in subsequent export definitions will use the default value. Below is an abridged configuration file, and an explanation of the entries.

The First line declares the beginning of the default entry. The next few entries declare that the default protocol will be NFS, and indicate the the host and port on which to look for the server on. Next the file specifies that by default all users will be able to access an export, and then gives the valid IP ranges for exporting software.

```
Default
type=nfs
host=netdist.domain.edu
port=888
users=*
allow=111.111.
allow=111.112.
export=soft1
text='Software Application \
          1.1 (Solaris, Linux)'
manual=softapp1.html
allow=111.113
allow=111.111
deny=111.111.12
export=soft2
text=''Software Application 2.1 (IRIX)'
manual=softapp2.html
deny=111.112.1
```

The export key defines the beginning of a new export. The value given becomes an internal identifier for the export and must be unique. The text and manual definitions function as described above. The two allow statements are combined, and they then replace (not expand) the default allow value for this export. The deny is then masked over the new allow value.

The second export defined is similar to the first, but different in a few key ways. First, the export value is a different string. This is essential as each export string must be unique within a netdist system. Also, since no allow key is defined, the default allow values are used and the deny value given is masked over them.

Below is a partial dump of the data structure created when the configuration file is read in. This structure is then passed on to the web client so that it can only lists the exports a user has access to and to make security checks against the requests before passing them on to the server. This dump was created with the Data::Dumper Perl module.

```
$VAR1 = bless( {
   'star52' => {
      'deny' => [],
      'users' => '*',
      'type' => 'nfs',
      'port' => 2110,
      'text' => 'StarOffice 5.2'.
                ' (Solaris|Linux)',
      'doc' => 'star52.html',
      'duration' => 3,
      'host' => 'server.indiana.edu',
      'allow' => [
         '^111.12.',
         '^111.14.',
         '^111.15.'
      ]
   },
   'default' => {
      'deny' => [],
      'users' => '*',
      'type' => 'nfs',
      'port' => 2110,
      'text' => 'default',
      'duration' => 3,
      'host' => 'server.indiana.edu',
      'allow' => [
         '^111.12.',
         '^111.14.',
         '^111.15.'
      ]
   },
   'work50' => {
      'deny' => [],
      'users' => '*',
      'type' => 'nfs',
      'port' => 2110,
      'text' => 'Sun WorkShop 5.0 (Solaris)',
      'doc' => 'workshop50.html',
      'duration' => 7,
      'host' => 'server.indiana.edu',
      'allow' => [
         '^111.12.'
      ]
   }
}, 'Configuration' );
```

Security

Security was a priority throughout the development of Netdist. We designed the architecture around several basic security tenets.

- Users need to be authenticated before accessing the service.
- Need to securely process NFS requests (which require root access), through our web server which runs under an unprivileged www account.
- Logging must be thorough and structured to make potential security problems highly visible.

Authentication

We have experience using apache with open-ssl along with with an internal mod_perl module to authenticate web connections against IU's kerberos database. These components had worked well for us in the past,allowing us to authenticate any IU user, and ensure that their username and password were safely transmitted to the server. Once authentication has been performed, Netdist works with the user through an abstract interface. Other apache authentication tools can be used, and the resulting data can be structured to work with the abstract database.

Request Processing

We required a method of responding to requests that was secure against local users as well as network based attacks. We investigated several options, including setuid programs, before deciding to use a client/server interface. They allowed us to securely make the transition from web server to root permissions, and seemed more expandable than other solutions.

After the web form processes the request, it makes a socket connection to a server running as root, and transmits the username, machine name, and export request. The server then processes and logs the request and sends the results back to the web form for display to the user. To ensure a robust service, Netdist supports both TCP and UNIX sockets for its client/server communications.

We must ensure that the server only replies to requests from authorized clients, that it encrypts any data sent over the network, and that the server handles malformed connections properly. Using UNIX sockets for communication simplifies this somewhat, as you only worry about attacks from the local machine, and can use file permissions to limit access to the socket. For this reason, UNIX sockets are preferred for Netdist, unless you need the ability to control exports on a foreign machine, in which case TCP sockets are required.

We use PGP to secure communication over the sockets. We generate a public/private key pair, assign the private key to the root account, and place the public key in the web server account's (www) keyring. The public/private nomenclature of PGP encryption isn't truly appropriate, as the public key is also private, and known only to the www account. No other copies of these keys exist, and both files can only be read by the owner. Neither of these accounts uses PGP for any other purposes, and the key pair is not used in any other way.

In effect, this configuration allows us to encrypt and sign the message with a single key pair. Since the www client is the only entity with access to the public key, any message encrypted with it has been signed by the client. Initially, we used a second public/private key pair to sign the encrypted message in the traditional PGP form. However, we eventually realized that the second key pair didn't increase security. Our www client's private key was no more secure than the server's public key, and only served to complicate the communication protocol.

This encryption/signing model relies on keeping the two keys a secret. This is a significant concern, as one of the keys is owned by the www account. Web server accounts are common targets of crackers as they often provide easy access to a machine. Not only must we worry about an attacker exploiting an insecure cgi script or server setting, it is also trivial for any user able to host cgi scripts on the web server to write a script that will print the keyfile. To reduce this risk, only members of the UWSG have accounts on the machine that hosts netdist, and only the www and root account can host web pages on the secure web server. Very few web scripts are run on this secure server, and all are closely examined, and perform strict taint checking on any entered data.

Each message sent by the client is encrypted with the public key. The server forks off a child process for each incoming message. It then goes through a series of steps to confirm the validity of the message. If any of these steps fail, the message and test failed are logged and the child process drops the connection and terminates before proceeding to the next step. These steps are completed in the following order:

1. The server confirms that the message is PGP encoded, and does not contain any additional text or data.
2. The server decodes the PGP message to ensure it was sent from the client.
3. The server performs taint checking on the command to ensure that it is a valid netdist command with no special characters, shell escapes, etc.

Once these tests are passed, the data is passed as a parameter to the appropriate export control script, and logged. At no point is code received by the server executed by perl or the shell. These precautions are in place to limit the impact of any successful attacks against the encryption/signing scheme. First and foremost, arbitrary programs can not be run as root, and the aggressive logging should allow us to identify any successful attacks and close the service until the problem is resolved.

Logging

Netdist maintains three different log files for readability. The first file logs any errors that occur during the export request process. This is the log where attacks on the system will be documented, as well as any problems that might be affecting the system. As such, it is important to closely monitor this log.

The second log file logs the time, username, supplied hostname, and requested exports of every request. This log gives a comprehensive listing of the requested exports, and can be used in conjunction with the error log to look for suspicious access patterns.

The final log records successful export requests, including the username, hostname, and timestamp of the request. This file can be be used to generate many different statistics on usage. It can be coupled with machine and user databases, to answer questions such as "How many requests were made by graduate students last semester?", or "How many requests for Star Office have been made from the Chemistry Department?".

Distributed Netdist

The Netdist program supports a distributed model, where a single web client can control software exports from multiple machines via multiple servers. This is useful for controlling multiple exports that can't be hosted on the same machine. This model requires a single master machine which contains the netdist configuration file, an instance of the Netdist server and the Netdist API. Additional machines that will export software are slaves and must contain an instance of the Netdist server, the Netdist API, and the export control scripts for each type of export is offers. The web client can be installed on any machine, and acquires its configuration from the master server. See Figure 2 for an example of how multiple machines can work together in a distributed Netdist environment.

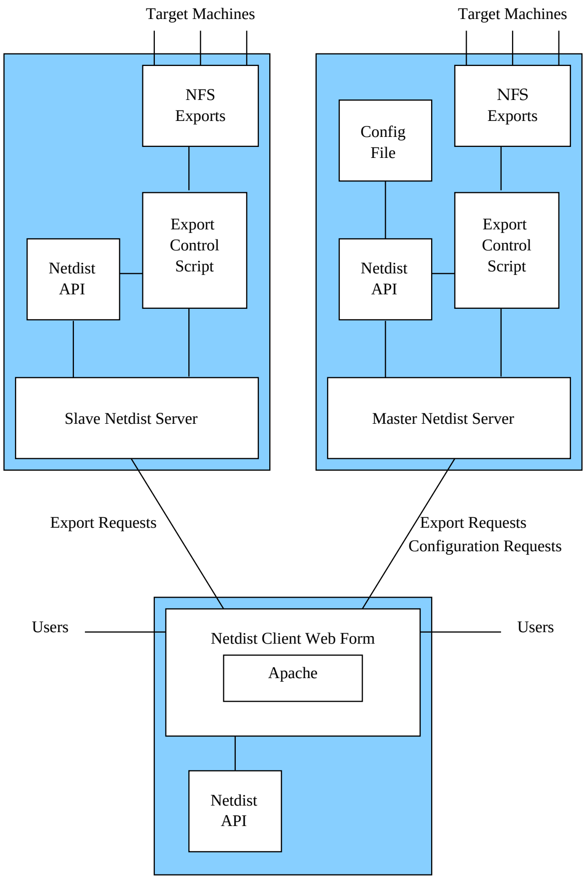

Figure 2: A sample Distributed Netdist implementation.

Portability

Currently, Netdist could be ported to other sites and platforms with a modest amount of work. It requires a host with Perl 5 and some modules from CPAN, PGP, cron (or some other scheduling routine), and an instance of Apache with at least mod_perl and preferably a module for secure transactions. The NFS export control scripts have been written for Solaris, but could be modified to work with the syntax of other Unix flavors.

The Perl modules and several scripts do have host or port specific information coded into the script, in a few places. Each of these instances is documented, and easy to update. In the next version of Netdist these values will be extracted to a configuration script to increase portability.

The Future of Netdist

The development of Netdist has slowed, as I no longer work for the UWSG, and have less time to work on it. However, the following items have been completed/are being worked on:

- Scripts to allow Netdist to control https access via .htaccess files
- Scripts to allow Netdist to control DFS and AFS exports via group membership
- Migrating from PGP to SSL for encrypting communication between the client and server. This is dependent on SSL support for perl through Net::SSLeay becoming more robust, or me re-writing the client/server in C.
- Modifying the user interface to support many more available packages via a tiered menu system.

The flexibility of Netdist is important to us, as preferred protocols and programs can change quickly. Netdist allows us to maintain a single, familiar point of contact for customers, despite changes in the way data is moved. For example there has been a push at IU for DFS and AFS in the past few years, and Netdist is ready to work with controlling access to software available on these systems.

Availability

Netdist is still pre-alpha in that we haven't done much work to ease installation, and we would like to incorporate some of the features mentioned above to increase it usefulness. As such, Netdist is not yet widely available. If you are interested in Netdist, please contact the author for the software location, and help with setting it up.

Acknowledgments

I would like to thank the people who worked with me in the UWSG at the time I was working on the Netdist project for their support, suggestions, and help with testing the system. In particular, Dick Repasky initially encouraged me to submit this work to LISA, and worked with me extensively to revise the first draft of this paper.

I would also like to thank the current members of the Data Storage Services Group for supporting my work on this paper, even when it interfered with my current job responsibilities.

Finally, I would like to thank those that helped me review and revise this paper. Their patience and willingness to repeatedly suffer my writing was instrumental in finalizing this paper, and is greatly appreciated.

Author Information

Chris Hemmerich <chemmeri@indiana.edu> works as a System Administrator at Indiana University (IU). He first became affiliated with the University as a student in 1993. Since then he has received a BS in

Biology, and is working on a Masters in CS. In 1995 he began working for IU as a lab consultant with the University Information Technology Services (UITS) division. From there he worked through various positions, including UITS Education Program Instructor/Developer, UITS Knowledge Base Programmer/Editor, and Unix Workstation Support Group Unix Systems Specialist, where he served as the primary HP-UX support contact for IU, and developed the Automated Netdist system. He currently holds a position with the Distributed Storage Services Group, where he administers IU's DFS and AFS infrastructure, while assisting with the administration of IU's HPSS installation. In addition to the e-mail address above, Chris can be reached by U.S mail at Indiana University; 2711 East 10th Street; Bloomington,IN 47408

References

[1] *CPAN documentation for Perl modules*, http://www.cpan.org .

[2] Christiansen & Torkington, *Perl Cookbook,* O'Reilly and Associates.

[3] Stein & MacEachern, *Writing Apache Modules with perl and C*, O'Reilly and Associates.

[4] P. Zimmerman, *The Official PGP User's Guide,* MIT Press.

[5] Perl man pages, especially perlipc.

[6] The Apache Website, http://www.apache.org/ .

[7] B. Wong, *Configuration and Capacity Planning for Solaris Servers*, Prentice Hall.

[8] The Open SSL Project, http://www.openssl.org /.

[9] H. Stern, *Managing NFS and NIS*, O'Reilly and Associates.

[10] *Distributed File Service Administration Guide and Reference,* IBM.

[11] The WU-FTPD Website, http://www.wuftpd.org/ .