NetMapper: Hostname Resolution Based on Client Network Location

Josh Goldenhar – Cisco Systems, Inc.

ABSTRACT

Large buildings, sprawling campuses and multiple remote sites have led to an explosion in the number of IP networks for corporate computing. Workgroups are spread over multiple networks. Servers are configured with multiple network interfaces in an attempt to optimize access. When geographic or capacity issues arise, separate servers which replicate desired functionality are placed in network proximity to their clients. Some of these servers provide tool trees which have operating system (OS) and architecture specific binaries. The optimal server or server interface for a given client may or may *not* have a presence on the client's network. In such an environment, how can an administrator guarantee a network client is utilizing the desired network interface or server?

NetMapper provides a framework for resolving hostnames (real or virtual) based on the client host's location within a network hierarchy. For servers with multiple network interfaces, NetMapper chooses the best interface. For multiple servers providing replicated services via a virtual hostname, NetMapper chooses the best server. For file servers providing OS and architecture specific filesystems, NetMapper chooses the best server taking into account client OS, architecture and network attributes. In all cases, 'best' is defined by the NetMapper administrator. As a side benefit, NetMapper allows systems and network administrators to view their network hierarchy at-a-glance.

The core of NetMapper functionality is a Perl module (NetMapper.pm) and a configuration daemon (nmconfd) which serves the configuration information to NetMapper clients. NetMapper has been implemented in the Cisco engineering environment by developing a small program called localmapper which runs on all UNIX clients and generates entries in the client's local /etc/hosts file. localmapper optionally generates a small NFS automounter map for OS and architecture specific remote partitions. This fairly small collection of tools allows systems administrators to choose which servers UNIX network clients use based on geography, workgroup or any arbitrary rationale that can be defined via groups of networks.

Introduction

Imagine working for a company that has thousands of UNIX computers distributed over hundreds of networks at various sites worldwide. All workgroups demand the ability to NFS mount each other's servers. Automounter maps contain thousands of keys. Add to this picture a mixture of license, compute, revision control, NTP and various other servers which are scattered throughout these networks. Lastly, imagine a user base that demands their access to these services be based on workgroup, network and geographic topologies. In this age of explosive growth for technology companies, many systems and network administrators don't have to imagine such a situation, it is reality.

As a network environment grows, it is common to replicate services by adding additional servers. Multiple network interfaces are added to servers to increase network bandwidth or provide streamlined access for a set of clients. As soon as an additional server or interface is brought online, a question arises: How does the administrator ensure network clients are utilizing the preferred distinct server or interface?

There are several existing products and methods that aid in the optimal resolution of server hostnames. Usually, these solutions make their choices based on a measurable quantifier such as number of simultaneous connections, network load or latency. These methods are acceptable when there is no differentiation between servers (such as replicated WWW servers).

In today's corporate compute environment there are often reasons to differentiate servers based on abstract policies. What if only certain file servers contain the OS and architecture specific binaries a client needs? Geographic, political, departmental or other arbitrary conditions sometimes necessitate a group of clients connect to a certain server based on those conditions. If these conditions can be split along network divisions, NetMapper provides a solution where server hostnames can be resolved in a somewhat arbitrary nature based on the desires of the administrator.

Existing Solutions

Hostnames that resolve to multiple IP addresses usually fall into one of two categories: multiple interface hostnames or virtual hostnames. Within the scope of this paper, these terms are defined as follows:

Multiple interface hostnames are defined as real hosts which have multiple active network interfaces. The hostname itself resolves (using the name service a site chooses) to one or more of the IP addresses of its interfaces. Additionally, each interface should have a distinct interface specific hostname. In Figure 1, resolving the hostname 'masses' would return both interface IP addresses. The hostname 'masses-115' would resolve to 123.45.69.115. The hostname 'masses-156' would resolve to 123.45.68.156. Thus, the administrator has a straightforward way of specifying an individual interface without having to use dotted notation.

Virtual hostnames are those that are intended to resolve to the IP address or hostname of one of a set of real hostnames. In Figure 1, two hosts ('waiter' and 'bus-boy') serve a replicated file system called /coffee. A virtual hostname 'folgers' could resolve to the name or IP address of either 'waiter' or 'coffee'. There is no computer actually named 'folgers'.

There are several existing methods and products to aid in the optimal resolution of hostnames that resolve to multiple addresses. These methods do not make a distinction between multiple interface hostnames and virtual hostnames. Some of these existing methods are described below as well as the reasons that NetMapper was chosen over them.

Domain Name System (DNS) Solutions

DNS [1] and BIND [2] solutions are most often used to resolve a multiple address hostname. While these work well for general conditions, they are less than optimal for returning addresses based on the client's network address.

It is possible to tailor BIND and DNS to return addresses based on client network location. Often, this requires many sub-domains or relies on a questionable BIND feature which returns an appropriate response based on the requester's IP address [3,4]. The latter feature has come and gone from BIND in different versions and is not viewed as reliable over time. In general the network address of the client requesting name resolution is not considered by the name server and thus answers are not tailored to the client [5]. While DNS servers have knowledge of the requesting host's IP address, they do not necessarily have any knowledge of other network interfaces on that client. Thus, even a modified DNS server would only be able to consider the source address of the request – even if the client has another interface that might provide a better path to the server hostname in question.

There are occasions when a server interface address should not be advertised by DNS for general use. Figure 1 shows two hosts, 'waiter' and 'patron', which are linked by a point-to-point connection. 'Waiter' serves /coffee to 'patron'. Optimally, when 'patron' does a name lookup on 'waiter', the IP address of 'waiter-pp' would be returned. DNS should not return the point-to-point address of 'waiter' to any other host. Therefore, the point-to-point address can't be included in the DNS configuration as an address ('A' record) for 'waiter'.

Lastly, DNS has no knowledge of the requesting client's OS or architecture. Therefore, it can't take these elements into consideration when returning an address for a hostname.

Automounter Variables Solution

If you want to take client OS and architecture needs into consideration for NFS [6] mounting, automounter [7] variables are a suggested solution to this problem. Automounter variables can be initialized at daemon run-time to change mount paths and server names. This method suffers from lack of compatibility across platforms and the fact that it only addresses the issue of hostnames for automounted NFS services.

Commercial Solutions

One solution that is applicable for practically any IP based protocol is Cisco's LocalDirector [8,9]. This is a great product for pure load-balancing. It works

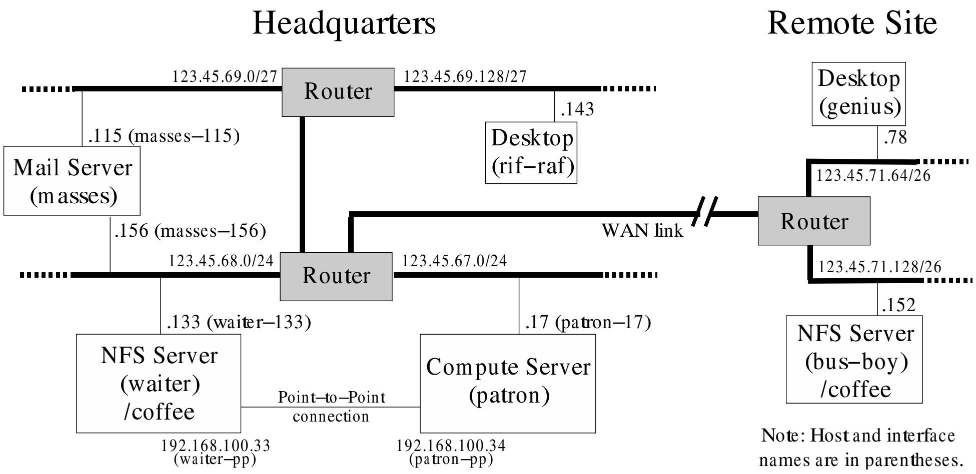

Figure 1: Example Hosts and Network Topology.

especially well to simply have network traffic automatically divided between multiple servers which all provide equal functionality. All traffic from the clients to the server(s) passes through the LocalDirector unit. This works well when the servers in the "pool" are in equal network proximity to the LocalDirector unit and the client network location is not an issue. LocalDirector can't take client OS and architecture needs into its consideration of which server to direct traffic to. While we use LocalDirector for certain applications, it can't cope with all the conditions we require.

Cisco's own DistributedDirector [10,11] overcomes LocalDirector's shortcomings in regard to network topology issues. DistributedDirector works as a DNS server. It uses knowledge of routing tables and network latency to resolve hostnames on a per-client basis. This solution is a great product for load balancing and directing traffic to hosts which provide distributed services. Because it operates as a DNS server though, it could not be used for the same reasons explained in the 'Domain Name System (DNS) Solutions' section above.

'Home-Grown' Solutions

Our previous solutions for optimally mounting NFS file systems involved a convoluted process using rdist [12], a Perl [13] script and m4 [14] to customize the auto.indirect automounter map on every client. This method worked for our largest automounter map but did nothing for other maps, processes or general name lookups. As the number of networks and hosts grew large it became obvious that this method was becoming too difficult to maintain.

After determining that no one existing solution could cope with all of our requirements, it was decided to build upon the beginnings of our own home-grown solution. NetMapper built on the groundwork of earlier attempts to solve this host/interface selection problem. The NetMapper methodology can be used as part of any number of solutions to host/interface resolution and location problems. The remainder of this paper will focus on a suite of scripts which address the problem of how a UNIX network client chooses the appropriate hostname or host specific interface when there are multiple possibilities.

Design Phase – Goals

In the NetMapper design phase there were several goals for base level functionality. The goals, and the reasoning behind them are explained in more detail below.

- A mechanism that ensures UNIX clients connect to the desired interface of a multiple interface server, or to a specific server for distributed and/or replicated services.
- Server hostnames that serve architecture or OS dependent partitions need to be selected by partition availability as well as client network location.
- Distribute as little as possible in the way of programs, source and configuration files to the clients.
- An easy to use administrative interface with basic error checking. The interface should aid in understanding the concepts behind the solution.
- The mechanism for choosing a hostname had to be well defined.

The first goal was one of the easiest to formulate, yet hardest to meet. We had to use a method to resolve hostnames that was fairly simple, yet worked on a wide variety of UNIX platforms. We did not want to implement a solution that involved replacing vendor libraries or significantly altering the operating system.

The second goal arose from the fact that in our environment we have NFS servers that provide architecture and OS specific binary tool trees. These servers are not guaranteed to contain all architecture and OS variants for these trees. When resolving a server hostname that serves such trees, it is crucial that the solution consider the client's OS and architecture. Furthermore, the solution must choose a server that has the appropriate file system available.

The third goal (distribute as little as possible) came into being from experience with our previous solutions. We were distributing automounter files, m4 definition files and Perl scripts to every UNIX client. This process took too long and was often not very reliable.

The fourth goal (ease of administration) was also based on previous experience. Our previous solutions involved multiple text configuration files which were sensitive to syntax and prone to error. We needed an interface to provide basic error checking so mistakes such as invalid hostnames did not make their way into the configuration. This interface needed to simplify the concepts behind NetMapper so that new systems administrators needed much less training than was required with previous solutions.

The final piece of the puzzle was the mechanism by which hostname and interface choices were to be made. It was decided that all decisions would be based on knowledge of the overall network hierarchy. Our sites, buildings and workgroups were all split among distinct networks or groups of networks. This would make it easy to resolve hostnames by many different conditions.

Design Phase – Decisions

After clearly defining our goals, we moved to the next phase which was deciding the best methods to achieve them.

To achieve our first goal, it was decided the easiest way to optimize hostname resolution for all services across multiple operating systems was to modify the client's local /etc/hosts file. Modern UNIX systems can be configured to use one or more methods of hostname resolution: /etc/hosts is an option to all, and is local to the client. Older UNIX systems that are not configurable default to using /etc/hosts. As long as /etc/hosts is consulted first, any program that performs a gethostbyname call will 'use' the NetMapper preferred address.

The second goal would be achieved by creating a naming convention for file systems that contain OS and architecture specific binaries. When defining a server hostname that provides such filesystems, NetMapper configuration would store which OS and architecture specific filesystems are available on that server. The client portion of NetMapper could then take this information into consideration when choosing hostnames that are defined as multiple OS and architecture file servers. This could be used to provide both hostname and path resolution for NFS mount commands.

The third goal (distribute as little as possible) would be met by making NetMapper configuration information a client/server process. The NetMapper Perl module would get its configuration information from a configuration server. If parts of NetMapper have to be installed on the client's local filesystem, they should be capable of updating themselves when necessary.

The fourth goal (ease of administration and error checking) was to be accomplished by providing a web-based administration interface. NetMapper configuration information would be manipulated via a browser to facilitate centralized administration and avoid syntax errors in the main configuration file. (NetMapper configuration would be stored in a Perl interpreted file; syntax errors could be catastrophic.) The choice of a web-based interface for configuration would allow the NetMapper network hierarchy information to be displayed in a way which would aid understanding. CGI forms and Javascript would also allow control over what information could be changed, how it would change and would assure that dependencies are honored.

The mechanism by which hostname resolution decisions would be made was to be based on a knowledge of the network hierarchy. All networks that contained potential NetMapper clients would need to be entered into the NetMapper configuration using Classless Inter-Domain Routing (CIDR) [15-18] format. Networks could be grouped together under a logical name. Logical groups would contain networks and other network groups. Each NetMapper hostname would be defined by mappings that specify which real hostnames, or interface specific hostnames should be used based on a network client's location within the network hierarchy. Architecture and OS specific virtual host definitions would contain a record of server candidates and which platforms those candidates are capable of serving. Together, these pieces would allow NetMapper to find a client's place in the network hierarchy and determine an appropriate server hostname for that network client.

Implementation

In order to achieve the goals and specifications of the design phase, NetMapper was split into parts and built on the existing infrastructure. Once all the functionality goals were finalized, the actual implementation took shape rather quickly. The design was implemented with a complete suite of Perl scripts. Details of the implementation are described below.

Enter LocalMapper

The manipulation of local /etc/hosts files is performed by localmapper. This method relies on the fact that /etc/hosts is searched from top to bottom and returns the first match it finds. Therefore, this script has a prerequisite which can have no exceptions – clients must have a static, truncated hosts file. That is, it must contain only 'localhost', its own hostname(s) and a bare minimum of other hosts.

The localmapper script appends a delimiter and all defined NetMapper hostnames to the hosts file. Thus, all localmapper modified hosts files will contain the same hostnames but possibly different IP addresses after the delimiter. If the localmapper delimiter is found in an existing hosts file, all entries after the delimiter are removed and replaced with the current NetMapper selections. This allows for overrides and other customized information to remain in the client's hosts file (above the delimiter).

The localmapper script is careful to avoid corrupting or truncating the /etc/hosts file. It uses a temporary file to construct the new hosts file. If there are any fatal errors the script aborts, leaving the existing /etc/hosts intact.

Sample /etc/hosts files based on the hosts in Figure 1 are included in Listing 1. The hostname 'folgers' is virtual – there is no actual host named 'folgers'. The hostname 'folgers' is defined in NetMapper (see Figure 2) to be either 'waiter' or 'bus-boy'. The desktop 'genius' at the remote site will be served /coffee by 'bus-boy'. Any host at headquarters will be served /coffee by 'waiter'. It should also be noted that the host 'patron' has the point-to-point IP address for 'waiter' in its /etc/hosts file, while all other hosts use the 'public' address for 'waiter'.

The localmapper script is the only program that needs to run on the client. If the localmapper script is installed on a client's local file system and is out of date, it can update itself when necessary.

OS and Architecture Specific Server Names

When resolving a virtual hostname which serves multiple architecture and OS partitions, it does no good to use the 'best' server for a network client if that server does not have the specific file system that the client needs to access.

In our compute environment we support various OS and architecture specific versions of /usr/local. There are multiple NFS servers throughout our networks which provide these OS and architecture specific file systems. On a UNIX client, /usr/local is actually a link to /auto/usrlocal. The directory /auto is an automount point defined by the auto.indirect automounter map. Thus, 'usrlocal' is an automounter key. The idea behind all this is that UNIX clients will dynamically mount /usr/local from their departmental or network closest server via a common pathname: /usr/local.

The virtual server name for the example above is simply 'usrlocalhost'. This virtual hostname must be resolved to one of several real server hostnames that serve copies of /usr/local. The problem is that not all variants of the /usr/local tree are available on every server. The NetMapper library takes this into consideration when asked to resolve such a hostname. When localmapper in invoked with the '-a' option, it produces a small indirect automounter map written to /etc/auto.netmapper. This file is overwritten if it already exists. It contains the keys, hosts and mount paths for these special file systems and servers. This allows all hosts to access these shared trees via a common name regardless of client OS or architecture.

Using the hosts in Figure 1, the /etc/auto.netmapper file for the host 'patron' is shown in Listing 2. There are various ways to make use of this map. It can be used as a separate map for a unique automount point. It could be included from another automounter map with a '+' entry. Modern UNIX systems could be configured to use this map before or after a NIS or NIS+ map.

```
# /etc/hosts file for desktop system rif-raf (Headquarters - Figure 1)
127.0.0.1        localhost
123.45.69.143    rif-raf
####### LocalMapper Auto-Generated Entries Below #######
####### Changes made below this line will be lost ######
123.45.68.133    folgers waiter
123.45.67.17     patron
123.45.68.133    waiter
123.45.69.115    masses masses-115

# /etc/hosts file for server system patron (Headquarters - Figure 1)
127.0.0.1        localhost
123.45.67.17     patron
192.168.100.34   patron-pp
####### LocalMapper Auto-Generated Entries Below #######
####### Changes made below this line will be lost ######
192.168.100.33   folgers waiter-pp
123.45.67.17     patron
192.168.100.33   waiter waiter-pp
123.45.68.156    masses masses-156

# /etc/hosts file for desktop system genius (Remote Site - Figure 1)
127.0.0.1        localhost
123.45.71.78     genius
####### LocalMapper Auto-Generated Entries Below #######
####### Changes made below this line will be lost ######
123.45.71.152    folgers bus-boy
123.45.67.17     patron
123.45.68.133    waiter
123.45.68.156    masses masses-156
```

Listing 1: Sample /etc/hosts files for three hosts in Figure 1.

```
# LocalMapper Auto-Generated automount table - changes will be lost
# key       mount options       hostname:mountpath
usrlocal    -ro,soft,noquota    waiter-pp:/export/usrlocal.sparc-sunos5
sw          -ro,soft,noquota    waiter-pp:/export/sw.sparc-sunos5
```

Listing 2: Sample auto.netmapper automount map for SunOS 5 host 'patron'.

Configuration Server

The NetMapper configuration server (nmconfd) aids in the effort to distribute as little as possible to network clients. The nmconfd script serves the Perl interpreted configuration file (netmapper.conf) to network clients at runtime. The NetMapper Perl module (NetMapper.pm) attempts to communicate with nmconfd on a hostname 'nmconfhost', port 13000.

For scalability, it is possible to have multiple NetMapper configuration servers by setting up 'nmconfhost' as a NetMapper hostname itself! The hostname 'nmconfhost' should have an entry in DNS that is the default address of the configuration server. In this manner, when localmapper runs for the first time (or has failed) a lookup of 'nmconfhost' will not result in a 'host unknown' error. (This is also the suggested way to add a default for all NetMapper hostnames.) The first time localmapper connects to nmconfd, it will use the default (DNS record) address. Once localmapper runs, later invocations will connect to the desired configuration server.

The nmconfd script can also provide the latest version of localmapper to enable self updates. The version information for localmapper is kept in netmapper.conf along with the network hierarchy, NetMapper hostnames, known architectures and OS's, interfaces to be ignored and a serial number.

Administrative Interface and Configuration Files

The information contained in netmapper.conf is stored in various Perl structures such as anonymous references to hashes and arrays. This information can be tricky to edit and is prone to syntax errors. The web administration interface was developed to greatly reduce this risk of corruption as well as presenting a more intuitive view of the network hierarchy and members (see Figure 3). The main CGI configuration script, netmapperconfig.pl, makes use of the CGI.pm, Net::DNS and Data::Dumper Perl modules for interface, name resolution and configuration file output. It was necessary to use Net::DNS for hostname resolution in case the host that was running netmapperconfig.pl was also a NetMapper client. (Calls to gethostbyname would use the /etc/hosts file for hostname resolution. If localmapper had previously modified the /etc/hosts file, the lookup of a hostname that it was trying to resolve would return whatever was

Figure 2: NetMapper configuration interface for sample host 'folgers'.

previously resolved. Thus, if a host address changed, the new address would never be known.) This interface allows the NetMapper administrators to change any information in the netmapper.conf file as well as providing some tools for testing and debugging.

Architecture/Internals

Version 1.0 of NetMapper is based on previous work along the same lines. This work was all done in Perl. The simple elegance of Perl hashes and powerful pattern matching kept Perl as the language of first choice for NetMapper. The previous work also used a Perl interpreted file to store configuration information. Availability of modules such as Data::Dumper to write Perl data structures out to a file made the decision to keep using this method a simple one.

Several data structures are used within the configuration file:

- A hash whose keys are the hostname 'aliases' which the NetMapper package is responsible for processing. The values of this top-level hash are anonymous references to other structures such as hash and array references. These other structures include such information as the default hostname or interface specific hostname, preferred hostnames based on network group names or CIDR format network addresses, available OS/architectural dependent partitions and possibly other information to be determined later.
- A hash representing the network hierarchy which consists of network IP addresses in CIDR format as well as symbolic names for groups of networks. The hash keys are either networks or network group names. The hash values are network group names of the parent of the key.
- A hash representing which OS's and machine architectures NetMapper knows about. NetMapper uses this data to produce the OS and architecture specific mount paths for indirect automounter map entries. An explanation of the OS/architecture specific partition naming is detailed in the NetMapper documentation.
- A scalar containing the version number of the NetMapper package/program (localmapper) which runs on the clients. (If the client program is not up to the current version, it can attempt to update itself via the network.)
- Various other pieces of information which would be maintained by the NetMapper administrator and should not be modified by NetMapper users (system administrators). This information might include interface names and error reporting email addresses.

NetMapper uses these structures to determine the desired hostnames for a network location in the following fashion:

1. localmapper runs on the client and initializes the NetMapper library which obtains configuration information as previously described.
2. NetMapper library functions examine network interfaces to obtain the client's active IP addresses and network lineage. (IP addresses

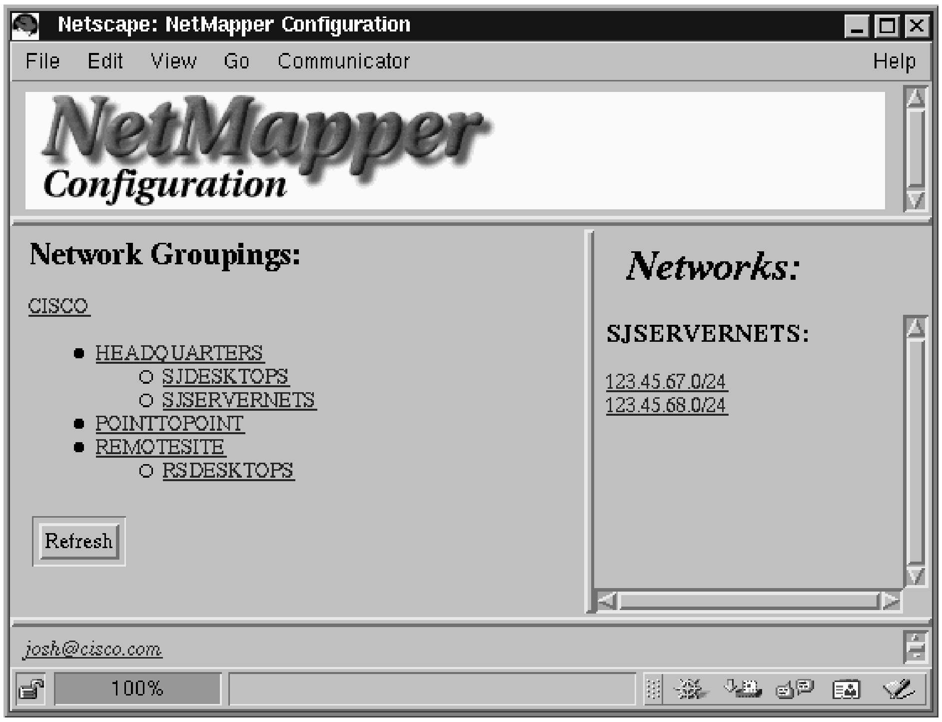

Figure 3: NetMapper network hierarchy displayed in configuration interface.

are obtained via standard UNIX commands such as netstat and ifconfig.) All CIDR format networks in NetMapper are reverse sorted by the number of significant network bits. The client's IP addresses are then compared to each element of the list of networks with a match occurring when the significant portion of the interface address and the network address are equal. The network lineage is then derived from the network hierarchy based on the matched networks. The lineage proceeds from most specific to most general network. Using the host 'rif-raf' from Figure 1 and the network hierarchy displayed in Figure 3, the lineage for 'rif-raf' would be as follows:

```
123.45.69.143/32
123.45.69.128/27
SJDESKTOPS
HEADQUARTERS
CISCO
default
```

The lineage (within the NetMapper network hierarchy) is derived via a NetMapper function (DeriveNetwork). This process yields a list of networks and network groups that this client belongs to.

3. For each hostname defined in NetMapper, a comparison is made between the lineage described above and the preferences defined in the NetMapper hostname structure. If no matches are found in the hostname definition the default will be used. If no default is found (or some other error occurs), the hostname alias will not be defined by NetMapper. (Thus hostname resolution for this entry would be determined through a NIS, NIS+ or DNS lookup.)
4. If the NetMapper hostname is determined to be an NFS server which serves architecture and OS dependent file systems, the preferred hostname will have additional match criteria applied – the preferred hostname must have the desired target file system available. If it does not, the first server candidate that does have the desired file system is selected. This fail-safe is a weak one and needs to be improved in later versions.
5. This process is recursive – If the resulting hostname is itself a NetMapper hostname, the lookup will repeat. For a distributed service, this assures selection of not only the best server for a client, but also the best interface on that server.

If part of this process fails, the NetMapper administrators are notified via email. The message contains information such as the hostname of the client and what type of error occurred. This allows NetMapper administrators to be notified of new unknown networks, machine architectures and OS's.

The NetMapper Perl module itself makes use of other modules available on the Comprehensive Perl Archive Network (CPAN) [19] such as Net::DNS and Net::Cmd. It's use and available functions can be viewed via perldoc or pod2text.

The NetMapper configuration server, nmconfd, is a very basic interactive script. It is currently written to be run via inetd. It supports a small set of commands and returns results in plain text. The command set and results are compatible with the Net::Cmd Perl module. Both reading and writing of netmapper.conf are protected by file locking to prevent reading a partially written file or simultaneous writes from the administrative interface.

The web configuration interface is not extremely complex. It makes extensive use of the CGI Perl module. It does rely on Javascript to work with frames and pop-up text lists of networks with certain mouse-overs.

Other Uses!

This paper has discussed hostname resolution in the context of modifying a client's local /etc/hosts file. It is important to point out that NetMapper is actually a Perl module. The localmapper script utilizes the NetMapper module to perform its host file manipulations, but this is only one possibility. In the following example, the NetMapper module is used to select a specific hostname in a script that will modify /etc/resolv.conf.

A systems administrator might want to populate client /etc/resolv.conf files with different nameserver addresses depending on network location. He or she could define some virtual NetMapper hostnames, perhaps something as creative as 'nameserver1' and 'nameserver2'. The new script could use the NetMapper module and call the appropriate function to return the specific real server hostnames of the appropriate 'nameserver1' and 'nameserver2' for the network(s) a client is on. The real code to lookup 'nameserver1' follows:

```
use NetMapper qw(:standard);
print
   ResolveHostname(
      'nameserver1',
      [BuildNetworkList()]
   ),
"\n";
```

That's it! Practically one line of code (plus the 'use' statement) is all it takes to make use of the NetMapper library. The script could then build a new /etc/resolv.conf based on the results. Other possibilities might be choosing a print or NTP server.

Conclusions

What makes NetMapper unique is that the administrator makes the decision about where network clients are directed via name resolution. NetMapper can be used to 'load balance' [20,21] among servers, but there are other products available that might do a better job. NetMapper works best when decisions have to be made along some very 'human' aspect of computing - something an administrator knows that the computer can't easily figure out. A great example of this is license servers. Two groups might each have their own individual pool of licenses for the same product. Each group has their own license server. If the two groups can be split into distinct networks, an administrator could create one NetMapper virtual hostname – perhaps 'dmv'. The licensed software need only be told to look to a server named 'dmv' for its license. One group (of networks) would resolve 'dmv' to one server, while the other would use the second server. Users and administrators would not have to configure environment variables or 'dotfiles'.

Availability and Compatibility

The NetMapper module and localmapper script have been tested on SunOS 4, SunOS 5, HPUX 9 and HPUX 10. It is believed to work on IRIX, AIX, HPUX 11 and Linux. The Perl code requires Perl 5 or better. The Perl module and associated scripts (and whatever else I put in there...) are available in a tar/gzip file via anonymous FTP at ftp://ftpeng.cisco.com/josh/NetMapper.tgz .

Future Directions

The future of NetMapper depends on many things. (Like the results of this paper...) Without knowing what the future holds, I can only mention some of the things I have thought about changing.

Version 1.0 is not very smart about picking a secondary host if the first selection of an architecture and OS specific host does not have the desired target file system available.

The nmconfd Perl script should probably be made into a full-fledged binary daemon. It currently is slow due to being run by inetd and then compiled.

The algorithms NetMapper uses to determine the appropriate hostname could be re-written in 'C' or some other compiled language. (I keep hoping I can put this off long enough for the Perl compiler to reach maturity.) This way they could be incorporated into a renegade DNS server (if someone wanted to do such a thing). For a majority of cases, this would remove the client portion of NetMapper.

Acknowledgments

Lanny Ripple – for writing some of the central functions within the NetMapper module that perform the network matches.

Steve Bower – for convincing me early on that CIDR style network addresses would make things better and that Perl anonymous references were a 'good thing'.

Krish Sivakumar – for walking into my cube and saying something along the lines of "...why don't you just tweak the local hosts file?"

Mark Baushke – for letting me copy the method by which we split up architecture and OS specific partitions.

References

[1] RFC 819, *The Domain Naming Convention for Internet User Applications*, Zaw-Sing Su, Jon Postel. August 1982, http://www.isi.edu/in-notes/rfc819.txt .

[2] James M. Bloom, Kevin J. Dunlap, University of California, Berkeley, "Experiences Implementing BIND, A Distributed Name Server for the DARPA Internet," *Proceedings of the Summer 1986 USENIX Conference*.

[3] comp.protocols.tcp-ip.domains FAQ, Section 5, Question 5.14, *Order of returned records*, http://www.intac.com/~cdp/cptd-faq/section5.html#order .

[4] comp.protocols.tcp-ip.domains FAQ, Section 5, Question 5.24, *Different DNS answers for same RR*, http://www.intac.com/~cdp/cptd-faq/section5.html#diferentRR .

[5] RFC 1123, *Requirements for Internet Hosts, Application and Support*, Chapter 6. R. Braden, Editor, Internet Engineering Task Force, October 1989, http://www.isi.edu/in-notes/rfc1123.txt .

[6] RFC 1094, *NFS: Network File System Protocol Specification*, Sun Microsystems, Inc., March, 1989, http://www.isi.edu/in-notes/rfc1094.txt .

[7] Brent Callaghan, Tom Lyon, Sun Microsystems, Inc., "The Automounter," *Proceedings of the Winter 1989 USENIX Conference*.

[8] Cisco Systems, Inc., *Load Balancing: A Solution for Improving Server Availability*, http://www.cisco.com/warp/public/cc/cisco/mkt/scale/locald/tech/lobal_wp.htm .

[9] Cisco Systems, Inc., *LocalDirector in the Data Center*, http://www.cisco.com/warp/public/cc/cisco/mkt/scale/locald/tech/ldir_wp.htm .

[10] Kevin Delgadilloi, *Cisco IOS Product Marketing*, "Cisco DistributedDirector," http://www.cisco.com/warp/public/cc/cisco/mkt/scale/distr/tech/dd_wp.htm .

[11] Cisco Systems, Inc., *The Effects of Distributing Load Randomly to Servers*, http://www.cisco.com/warp/public/cc/cisco/mkt/scale/distr/tech/ddran_wp.pdf .

[12] Michael A. Cooper, University of Southern California, "Overhauling Rdist for the '90s," *Proceedings of the Sixth Systems Administration Conference* (LISA '92).

[13] Tom Christiansen, "Perl 5.0 Overview," USENIX *;login* magazine, Nov/Dec 1993, Volume 18, http://www.usenix.org/publications/login/christiansen.html .

[14] Free Software Foundation, *m4 Online Manual*, http://www.fsf.org/manual/m4/index.html .

[15] RFC 1517, *Applicability Statement for the Implementation of Classless Inter-Domain Routing*

(CIDR), Internet Engineering Steering Group, R. Hinden, September, 1993, http://www.isi.edu/in-notes/rfc1517.txt .

[16] RFC 1518, *An Architecture for IP Address Allocation with CIDR*. Y. Rekhter, T. Li, September, 1993, http://www.isi.edu/in-notes/rfc1518.txt .

[17] RFC 1519, *Classless Inter-Domain Routing (CIDR): An Address Assignment and Aggregation Strategy*, V. Fuller, T. Li, J. Yu, K. Varadhan, September, 1993, http://www.isi.edu/in-notes/rfc1519.txt .

[18] RFC 1520, *Exchanging Routing Information Across Provider Boundaries in the CIDR Environment*, Y. Rekhter and C. Topolcic, September, 1993, http://www.isi.edu/in-notes/rfc1520.txt .

[19] *Comprehensive Perl Archive Network (CPAN)*, http://www.cpan.org .

[20] Internet Software Consortium, *DNS, BIND and load balancing*, http://www.isc.org/view.cgi?/products/BIND/docs/bind-load-bal.phtml .

[21] Roland J. Schemers, III, SunSoft, Inc., "lbnamed: A Load Balancing Name Server in Perl," *Proceedings of the Ninth System Administration Conference* (LISA '95).

Related Papers and Information

Cheng-Zen Yang, Chih-Chung Chen, and Yen-Jen Oyang, Department of Computer Science and Information Engineering, National Taiwan University, Taipei, "Clue Tables: A Distributed, Dynamic-Binding Naming Mechanism," *USENIX Summer 1994 Technical Conference*.

Chad Yoshikawa, Brent Chun, Paul Eastham, Amin Vahdat, Thomas Anderson, David Culler, University of California, Berkeley, "Using Smart Clients to Build Scalable Services," *Proceedings of the USENIX 1997 Annual Technical Conference*.

Giray Pultar, "Automatically Selecting a Close Mirror Based on Network Topology," *Proceedings of the 12th Systems Administration Conference*, (LISA '98).