Insider Tips: Resolving Names in DNS

THE NAME GAME

www.sxc.hu

The resolver is the window from your computer to the global DNS system. Simply typing a URL awakens a link to computers all over the world.

BY MARC ANDRÉ SELIG

Memory artists can memorize a list of over 300 numbers, but others might find it hard to remember even the four numbers that make up an IP address. Most people agree that a name is easier to remember than a number. The advantage of associating an object with an alphanumeric name is the real reason for the existence of the huge, globally distributed Domain Naming System (DNS), a system that links computer names and addresses around the world.

The DNS system has been the subject of countless articles and books, most of them focused on the naming system itself or on DNS servers such as Berkeley Internet Name Domain (BIND). But on the other end of the connection is an application on a simple desktop computer that needs a means of contacting the server to resolve a name to an IP address.

THE AUTHOR

Marc André Selig spends half of his time working as a scientific assistant at the University of Trier and as an ongoing medical doctor in the Schramberg hospital. If he happens to find time for it, his current preoccupation is programming web based databases on various Unix platforms.

The Resolver

The interface that provides a means for an application to access name resolution services is commonly known as the resolver. One popular approach for integrating name resolution with a Linux application is through the GNU C Library (glibc) functions. The most common functions are *gethostbyname()*, which translates a name into an IP address, and *gethostbyaddr()* for going the opposite direction. This interface, and its auxiliary functions, are increasingly being replaced by a more flexible version, which uses *getaddrinfo()* to translate hostnames into IP addresses and *getnameinfo()* for the opposite direction.

Most scripting languages provide comparable functions; for example, Perl has *gethostbyname()* and *gethostbyaddr()*. The more recent variants are not available – in Perl's case, they are part of a separate module that provides IPv6 compatibility.

Additional resolver tools are available at shell level, such as the now obsolete *nslookup* tool or the more recent *host* and *dig* tools.

The tricky thing about handling names and addresses is the fact that not all programs actually use the library functions. Some of them just do their own thing. The idea behind this roll-your-own approach is to avoid the kinds of blockages that can occur when using the glibc functions. Many shell tools were designed as debugging tools and thus provide direct network access. And for another thing, many of these tools are static builds, meaning that the resolver routines use specific (and often obsolete) library versions.

Settings

The resolver can use various data sources. The simplest data source is the */etc/hosts* file (Listing 1), which contains a list of numerical addresses and corresponding names. Each address can be mapped to one or more names.

The records in the hosts file consist of an address and one or more names, as well as blank lines and comments. The listing has various types of entries: the *localhost* 127.0.0.1 entry is common to all */etc/hosts* files, as it is required for local interprocess communications. The second address block contains important IPv6 addresses. The third block shows how addresses on the local network can be added to the local database, offering a simple way of handling domain names outside of the local domain. If the

resolver fails to find a requested hostname in */etc/hosts*, it can contact a network based DNS service such as BIND. The client can refer to its */etc/resolv.conf* file (Listing 2) for the address of the name server; the file can either be created manually or by the DHCP client.

Normally, up to three different name server entries are permitted. The second and third entries are only used if the name servers with higher-ranking entries do not respond. The *options rotate* entry in Listing 2 changes this behavior for a LAN, providing a simple means of distributing the load across multiple machines.

Sources

The resolver needs to know which of these databases are available to which Linux system and in which order they should be used. This information is stored in various configuration files.

The original configuration file for this purpose was */etc/host.conf*. The file might look like this:

```
order hosts,bind
multi on
```

The most important keyword here is *order*, followed by the order in which the methods should be applied. *hosts* represents the local address list in the */etc/hosts* file, and *bind* refers to network-based access to the DNS system. The *multi on* entry shown here means that */etc/hosts* can have multiple entries for the same name.

The file tells the resolver to check */etc/hosts* first, to see if the required name or address is listed there. If this search does not come up with the goods, the DNS service is then used; in this case, the resolver uses the configuration in */etc/resolv.conf*.

Host.conf predates glibc 2.x; that is, it corresponds to libc.so.5 or earlier. Current versions of glibc now use the name service switch from Solaris 2 and support a flexible configuration of arbitrary lookup services in a freely configurable order.

Name service switching is configured in */etc/nsswitch.conf* (Listing 3). The file contains a list of services and data sources, as well as a description of the desired behavior. The *hosts* service is relevant to name resolution; the strategy indicated here, *files dns*, is the default and means that */etc/hosts* should be used first, followed by a DNS request.

Listing 1: /etc/hosts

```
01 127.0.0.1    localhost.localdomain localhost ishi
02
03 # The following lines are desirable for IPv6
   capable hosts
04 ::1          localhost.localdomain localhost
   ip6-localhost ip6-loopback
05 fe00::0      ip6-localnet
06 ff00::0      ip6-mcastprefix
07 ff02::1      ip6-allnodes
08 ff02::2      ip6-allrouters
09 ff02::3      ip6-allhosts
10
11 172.16.45.1  natrouter
12 216.92.94.3  sedacon.pair.com
```

Listing 2: /etc/resolv.conf

```
01 nameserver 172.16.45.2
02 nameserver 172.16.45.3
03 options rotate
```

There are more configuration files. Sendmail is just one example of a program that has its own resolver library. The configuration is similar to glibc's service switch but not identical. The private configuration file for sendmail is titled */etc/mail/service.switch*. It only supports the *passwd*, *hosts*, and *aliases* services. Sendmail does not use colons to separate service names, but other than that, */etc/mail/service.switch* is very similar to */etc/nsswitch.conf*.

No Net

In some circumstances network requests may be undesirable. For example, if you have a modem or DSL, and are charged an hourly rate, avoiding DNS queries can save you a lot of money.

Admins can delete the *bind* from */etc/host.conf* and *dns* from */etc/nsswitch.conf* to stop the glibc resolver from accessing the network. However, this will also stop the client from talking to a name server on the LAN.

If you need to use a tunnel – such as a VPN – for confidential work, you will either need to avoid DNS queries or direct them through the tunnel. In many cases, all you need to do is to configure the system to dispatch email via IP addresses or add the domain name to */etc/hosts*. This leaves the browser as the most common instigator of DNS queries. A Socks-4a proxy, such as privoxy for example, will serve you well in this case because it allows you to reroute DNS lookups to the other end of the tunnel.

Considering the variety of potential resolvers, best security practices would suggest using firewall rules to prevent undesirable DNS queries.

Keeping in Line

All of the mechanisms we have discussed are implementation specific. Although each flavor of Unix has its own default resolver, the details of the implementations will differ. For example, BSD uses a different *host.conf* format, and Solaris has */etc/netconfig*, a method that replaces the service switch with a private library. ■

Listing 3: /etc/nsswitch.conf Example

```
01 passwd:        compat
02 group:         compat
03 shadow:        compat
04
05 hosts:         files dns
06 networks:      files
```

How DNS Came About and How it Still Works

The future was looking bleak. With user numbers booming, the procedures that had served ARPANET, the Internet predecessor, well in the seventies now were collapsing. And one of these procedures was the process for mapping names to addresses. The network's admins had simply maintained a large file for this purpose, and everyone on the network had just used FTP to download a copy. Anyone wanting to add a new computer to the network would simply notify the Network Information Center (NIC) by email, and NIC would update the central hosts file and put it up for grabs.

Increasing numbers of hosts led to more traffic due to the need to download the hosts file. The size of the network also posed a seemingly unsolvable problem for the network administrators: although it was possible to assign unique addresses, there was no way to avoid name conflicts. Unfortunately, duplicate names could also cause serious disruptions by making possibly more important namesakes unreachable. Additionally, information had a tendency to become obsolete on the way from the NIC to the outer reaches of the network, meaning that there was no way to ensure consistency.

Research into a solution finally led to a system which – after a few corrections and enhancements – has stood firm in the face of the Internet boom, the Domain Name System (DNS). DNS's recipe for success is decentralization. Instead of a single authority for an unimaginably large number of networked computers, the system calls for a kind of group leadership, where an entity is responsible for a group of computers known as a domain, and domains themselves can be subdivided into subdomains. The graph representing the groups and subgroups is a tree, which is very similar to the directory tree in a filesystem.

To address a specific host, you need to enter its name and domain, and any superordinate domains up to the root of the tree. This string is referred to as a fully qualified domain name (FQDN). A name server manages the list of names and the addresses they map to for a domain. The name server may assign responsibility for subdomains to other name servers.

If a client is interested in discovering the address for a specific name, it contacts the name server for its own domain. If the name server does not have the answer in its database or cache, it then contacts the name server that it considers to be closest to the target, or at least passes this server's name on to the client. In this way, the request navigates the domain hierarchy in a targeted way, until it reaches a name server that has the answer in its cache or address list and can respond to the client or the requesting name server (Figure 1).

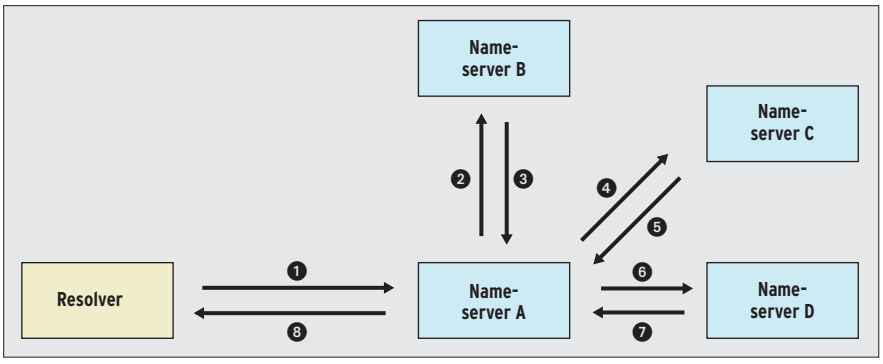

Figure 1: Name resolution sometimes requires a chain of several requests and responses: ❶ Client sends request to name server A. ❷ Name server B receives request. ❸ Server B responds with a reference to C. ❹ Server C receives the request from A. ❺ Server C responds with a reference to D. ❻ A queries D. ❼ Server D responds with address. ❽ Server A replies to client.