HostDB: The Best Damn host2DNS/DHCP Script Ever Written

Thomas Limoncelli – Cibernet Corp.

ABSTRACT

HostDB is a system for generating DNS zone files, BIND configurations, and ISC DHCP server configurations. It is extremely simple yet powerful, which makes it easy to deploy. It is not so complicated that it requires an SQL database or web server. It is not a single generation script but a complete system that includes tools for deploying the files that are generated. The system is written in Perl and shell and makes heavy use of make. The zone files that it generates look hand-made, except that they are clean. Powerful tools are also provided to aid in the conversion from legacy systems. HostDB is the third generation of my DNS maintenance systems and embodies not just superior software but also best practices in system administration. HostDB uses small tools that combine to achieve large goals. Initial deployment can be done in stages to reduce risk. Deployment of updates is fully automated. It rocks.

Introduction

The goal of HostDB is to create a DNS zone generator that is full-featured without requiring complicated databases and web services. The system uses simple configuration files and automates deployment. It is easy to deploy whether you are converting a legacy system one zone at a time or setting up a new system from scratch.

Many small and medium sites (dozens or hundreds of machines) hand-edit DNS zone files. This is an error-prone practice which requires a high skill level. The smallest typo can cause big outages.

Many large sites (multiple IT teams, locations, and internet connections) use DNS management systems that provide no revision control and very little audit trail of who made what changes. Many also require external (outside the firewall, or ‘‘public’’) zone files to be maintained by hand, which adds the risk that they will become out of date with internal (inside the firewall, or ‘‘private’’) zones.

The problem with most DNS generation scripts is that they are either too simplistic to be useful or so complicated that they are intimidating to install. The simple ones are fine for doing an initial conversion from /etc/hosts to zone files that will be hand-edited in the future. Many are ‘‘point tools’’ that require a lot of integration work. The complex ones are so unwieldy that small sites find them too scary to install. Most don’t handle complicated DNS idioms such as having special MX records for mail servers, proper handling of multi-homed hosts, or templates for ranges of similarly named hosts. They don’t handle generating different zone files for inside the firewall versus outside the firewall (i.e., ‘‘host hiding’’) resulting in hand-edited external zone files that are prone to errors. Many generate zone files that are not pretty: hand-edited zone files would be more readable (if we only had the time to maintain them that way!). HostDB fixes all these issues. It does this without introducing a complicated syntax plus it generates your DHCP configuration file ‘‘for free’’! The code is written in Perl, shell, and make and is freely available.

Setup time is fast. Really fast. The system can be used to generate zone files that can be individually inserted into your legacy systems within minutes of installing the software. HostDB breaks from the Free and Open Source Software (F/OSS) tradition of providing a default configuration that is an arrogant display of the most complicated features. Instead the default configuration is a useful example that can often by used ‘‘out of the box’’ with only small changes. Documentation is included for migrating from a legacy system in small steps with plenty of testing along the way.

Audience

HostDB seeks to fill the void between simple DNS zone generation scripts and massive DNS/DHCP database-driven applications. It is perfect for a site that is starting fresh, or for one that currently hand-edits zone files and wishes to introduce automation.

The learning curve and installation curve for HostDB is very small. Thus, it is appropriate for sites that do not have the need, expertise, or resources required by systems that have a web-based front-end or use a SQL database as the backend.

HostDB is good for sysadmins that need an extremely good solution but don’t have a lot of time. Migration from a legacy system is extremely easy because the files that are generated are put into a

repository before they are copied to their final destination. Thus, comparisons between the legacy zones and the HostDB-generated zones can be done as part of acceptance testing. Tools for comparing zones are provided. Individual zone files can be put into service permitting an incremental deployment methodology (rather than a ''flash-cut''), thus reducing risk.

HostDB is not appropriate for extremely large organizations with many separate IT groups or sites with complicated dynamic DNS configurations. Their needs would be better served with other, larger, systems. It doesn't directly support a sophisticated authorization schemes, a web-based interface, or assistance for dynamic DNS updates. However, HostDB does have some hooks that make it easy for simple dynamic DNS to be used. Also, large organizations are often composed of many smaller IT organizations which would find HostDB entirely appropriate. These organizations often have one team that manages all internet connectivity and HostDB is excellent for maintaining the related DNS zones.

At this time HostDB has no support for IPv6. I'm okay with this.

What Does It Do?

I have seen many exceptional DNS/DHCP configuration management systems that provide excellent web front ends and are so powerful that implementation requires an SQL database. HostDB isn't one of those.

HostDB lets you track all DNS and DHCP information about all your hosts in a single, simple file. From that file, it generates DNS zones, BIND configuration files (named.conf) and host entries that can be included in an ISC DHCP server configuration (dhcpd.conf). Two helper configuration files guide zone creation and file deployment, but they are basically static after the system is stable.

The system generates the complete zones and configurations. No hand-editing after the fact is required. Thus, automation using make and other tools is possible.

The system handles complicated DNS idioms such as:

- Special MX records for mail servers.
- Proper DNS records for multi-homed hosts and routers.
- Templates for ranges of similarly named hosts.
- Host-hiding (generating different zones for external DNS servers).
- Clean, human-readable zone files are generated (as if they were human-generated, but neater)

HostDB includes a tool called ''mkdestinations'' which generates a Makefile that will efficiently deploy (push) individual files to local and remote hosts. It recognizes zone files and treats them specially to prevent zone-transfer storms. Non-zone files can be deployed as well, making it possible to maintain the all files related to DNS and DHCP configuration for multiple servers around a network from one central administration host.

Major tools are written in Perl, with many helper programs in BASH shell. Shell glues them together in a way that is easy for sysadmins to understand. The system tries not to re-invent the wheel, thus it uses standard UNIX tools: make, SSH, sed and awk. Sed and awk have been feeling ignored lately so we threw some work their way.

How to Configure

There are three configuration files. Each encapsulates information that would be updated by a particular audience. The format is either simple or complicated based on the audience as summarized in Table 1.

The syntax of hostdb.txt is simple because a non-DNS expert needs to be able to edit it without introducing errors. Rather than designing a syntax by and for DNS experts, much effort was spent in designing a syntax that made sense to the junior engineers that would be using the system. The filename ends with ''.txt'' to facilitate non-UNIX admins editing these files through non-UNIX tools like Notepad.

zoneconf.txt requires a little more knowledge of how DNS zones work, which is appropriate because this file should only be edited by a person with advanced DNS knowledge.

destinations.txt is extremely simple because it simply states where files are to be copied after they are generated. It permits files to be renamed as they are copied, which makes inserting a generated file into a pre-existing legacy system very easy.

Standard UNIX protections can be used to control who can edit these files. They are all plain ASCII files, appropriate for maintaining under revision control systems like RCS, SCCS, CVS or Subversion. Any system administrator maintaining key files such as these without a revision control system is taking an undue risk. Revision control means knowing what changed when. When a change is introduced that causes an outage, one can undo a change, investigate who made the change, mentor them so they do not repeat the mistake, and optionally ridicule them during staff meetings.

DNS Zone Generation

DNS zones are generated using ''mkzones''. Mkzones parses the entire hostdb.txt file, storing the important bits into various data structures. It then reads zoneconf.txt which is ''executed'' line by line, setting options and generating zones as instructed. Because of this design, an option can be set one way before a particular zone is generated and then another way before the next zone is generated. This is a powerful feature, letting one system generate zones with very different

needs. For example, if one sub-domain requires different default MX records than all the others, it is very simple to achieve that goal. Other DNS generation systems assume the same options for all zones. That makes them inappropriate for very large sites.

The power of mkzones' algorithms comes from the fact that the parsing of hostdb.txt is decoupled from the generation of the zones. This permits more sophisticated zone generation since the generation algorithms understands "the big picture," i.e., all host information is available at once.

Decoupling parsing from generation also had the benefit that the input syntax could be changed without affecting the generation code. We were able to experiment with different syntaxes over time without worrying about how this would break the generation code.

The syntax for hostdb.txt was developed with massive user input. Users were presented with options for syntax and their feedback altered the evolution of the system. Most DNS zone generation systems tend to be designed for the DNS expert. HostDB has the benefit of a host list which is in a format that is perfect for junior sysadmins and the complicated options are kept far away in a different file where only the experts can make changes.

The users quickly lined up behind a syntax that looked like /etc/hosts, not a fancy recursive syntax like ISC's BIND and DHCP configuration files. They requested a single line per IP address because when it is time to allocate an IP address (their most frequent task) the procedure they are most comfortable with is finding an unused IP address in a long list, sorted by IP address. They also didn't want to be bothered with having to keep the file perfectly indented or brackets matched.

What Makes the Generation So Cool?

To make a zone file look readable took quite a long time. After many iterations we finally realized that the most readable zone file stripped hostnames (removed the domain, when possible), were formatted with tabs (not spaces), grouped records related to a particular host, and ordered the hosts by IP address.

Therefore, when generating zone files, we output:

1. The SOA information
2. The NS records
3. Any records for the current zone name (the domain without any host)
4. All the DNS information related to a particular host (with hosts listed in IP address order)

You may notice that the examples from hostdb.txt always specify Fully Qualified Domain Names (FQDNs). That is, foo.example.com, not just simply "foo". Usually DNS systems assume a label is simply a hostname (sans domain) unless it ends with a single period. However, humans tend to forget the period. After many iterations of trying to solve this problem by being fancy, we decided it was better to require the users to always enter FQDNs and output warnings if something was not a FQDN.

The decision to group the records in the zone file by hostname (yet stay in numerical IP address order) was something of a small breakthrough discovered after many trials. To do this all DNS information has to be in memory before the first zone is generated. This way decisions can be made by checking in-memory data structures and flags that were built up during parsing. Items from the in-memory data structures are deleted as each DNS resource-record is output into the zone file, thus eliminating the possibility of outputting the same line twice.

BIND Configuration File Generation

mknamedconf uses the information in zoneconf.txt to generate ISC BIND configuration files for DNS masters and slaves inside the firewall, plus DNS masters outside the firewall. It does not generate configurations for slaves outside the firewall (external) because the author has not yet found the need for such a thing. External slaves tend to be owned and managed by separate organizations, often ISPs. We do not know enough to generate their entire bind configuration, nor can we assume they use BIND. Configuration updates are usually done via manual (emailed) requests of the owner of the slave server.

DHCP Generation

The DHCP generation is very simple. Mkzones is called with a flag that tells it to parse the hostdb.txt file as usual, but then output a ISC DHCP "host" statement for each machine with DHCP-related data. This information can be included in a ISC DHCP "dhcpd.conf" file, which must be manually created. DHCP configurations are usually too custom to warrant generating them automatically, except for the host statements.

File Name	Audience	Contents	Skills required
hostdb.txt	Anyone on your sysadmin team	Info about all hosts	Edit text files, maintain a format set up for them, understand IP addresses
zoneconf.txt	DNS engineers	How the zone files and BIND configuration files are generated	Understanding of DNS zones and terminology
destinations.txt	UNIX admins setting up BIND	Where zone files and other files are copied	Knows where zone files are stored and has root access on DNS servers

Table 1: HostDB configuration files.

The Syntax

The systems works in two phases. First, the zoneconf.txt and hostdb.txt files are used to generate all the files that are generated. Next the new files are copied to the appropriate hosts/directories based on a map described in destinations.txt.

You might think of this as:

Hostdb.txt + zoneconf.txt → generated files

Generated files + destinations.txt → deployed files.

hostdb.txt Syntax

The hostdb.txt file looks like a /etc/hosts file augmented with flags and parameters. Display 1 demonstrates the syntax of hostdb.txt:

The zone file ‘‘example.com’’ that would be generated would be a simple ‘‘A’’ record for each host plus appropriate NS and MX records as configured in zoneconf.txt. The reverse zone (1.1.10.in-addr.arpa) would be one PTR file as appropriate for each host.

Notice that each host is listed by its fully qualified domain name (FQDN). For example, spoon.example.com instead of just ‘‘spoon’’. During user testing, it was discovered that it was too difficult to come up with a syntax that permitted ‘‘short’’ hostnames to be used and still permit multiple domains to be described in the same file. Various formats were attempted. In the end, it was decided that it was cleaner to make every host be listed as a FQDN.

‘‘kettle.example.com’’ has a special option on it: ‘‘MAC=00:b0:d0:a6:cf:f1’’. This identifies its Media Access Control (MAC) address, or Ethernet, address. When a DHCP configuration file is generated, this host will have a ‘‘static assignment’’ or ‘‘permanent lease’’ for 10.1.1.2. We have found it useful to assign static assignments to all known hosts, and use ‘‘pools’’ of randomly allocated addresses only for transient hosts. This makes log files more accurate since most machines will always appear at the same IP address.

‘‘fork.example.com’’ has a CNAME of ‘‘pitchfork.example.com’’. This generates a DNS CNAME record.

In addition to CNAME=, there is also ANAME=. This generates multiple DNS A records pointing to the same IP address. HostDB generates the reverse lookup to be the first host on the line, which seems to be the most common practice.

Multiple ANAMEs and CNAMEs can be listed by separating them by colons. For example one might list ‘‘ANAME=pitchfork.example.com:branch.example.com:divide.example.com’’.

ANAMEs and CNAMEs can both appear on the same host line.

The plural ANAMES is the same as ANAME, as is CNAMES the same as CNAME. This makes the hostdb.txt file slightly more readable.

Multihomed Hosts

HostDB understands that multihomed hosts need special treatment. For example, a router has many interfaces. The best practice is to have one DNS label or name that will return all ‘‘A’’ records; one for each interface. However network engineers and others need to be able to specify a particular interface when they want. HostDB likes it both ways.

Suppose the .1 address of each network was an interface for our router, zathras. We could specify that as shown in Display 2.

The DNS zone information generated would be:

```
zathras         IN  A   10.1.10.1
                IN  A   10.1.20.1
                IN  A   10.1.30.1
                IN  A   10.1.40.1
zathras-red     IN  A   10.1.10.1
zathras-blue    IN  A   10.1.20.1
zathras-green   IN  A   10.1.30.1
zathras-purple  IN  A   10.1.40.1
```

(MX info deleted for brevity, but I assure you they are awesome.)

Someone that wanted to reach any interface would use ‘‘zathras’’. A network engineer that needed to refer to the zathras interface on the ‘‘red’’ network would specify ‘‘zathras-red.example.com.’’

The reverse-lookup zone indicates the individual interface names (zathras-red, not zathras).

HostDB also has ‘‘ISMULTIHOMED=’’ which is the same thing but for servers. Currently it is exactly the same thing as ‘‘ISROUTER=’’. The separate directives are provided should different functionality be required for routers and hosts in future versions.

Host-hiding

Host hiding is where, for either real or perceived security reasons, most sites do not want to expose the

```
10.1.1.1    pot.example.com
10.1.1.2    kettle.example.com MAC=00:b0:d0:a6:cf:f1
10.1.1.3    spoon.example.com
10.1.1.4    fork.example.com   CNAME=pitchfork.example.com
10.1.1.5    spatula.example.com
```

Display 1: Syntax of hostdb.txt .

```
10.1.10.1   zathras-red.example.com     ISROUTER=zathras.example.com
10.1.20.1   zathras-blue.example.com    ISROUTER=zathras.example.com
10.1.30.1   zathras-green.example.com   ISROUTER=zathras.example.com
10.1.40.1   zathras-purple.example.com  ISROUTER=zathras.example.com
```

Display 2: Specifying .1 addresses as a router interface.

names of their hosts to the outside world. Inside their firewall, a host may be called patentdatabase.example.com, but outside it should be simply known as h64-32-179-55.example.com. Obscuring the host name like this prevents external users from being able to make educated guesses about good attack targets. It prevents internal host names from being exposed outside the network, which might prevent embarrassment by preventing ceo-pc.example.com from appearing in the Apache log files of www.sexyteens.com.

For forward DNS lookups one generally sets up an external DNS zone file that lists only the hosts that the public internet needs access to. This is used on external DNS servers and is often managed separately from the internal of DNS zones. Any time two separate databases are used to store information about the same thing you are asking for trouble; they will get out of sync. It is my experience that even sites with very full-featured commercial DNS management systems use a hand-edited zone-file for their external DNS zones.

Reverse DNS lookups are another matter. Some sites simply do not provide any reverse-DNS records. As a result, other sites refuse to talk to them because, for real or perceived security reasons, other sites are weary of accepting connections from sites without proper reverse DNS. Other sites simply use statically generated reverse labels that encode the IP address to keep them unique. For example, at this time Optimum Online's reverse DNS for 67.82.128.6 is ool-43528006.dyn.optonline.net (the digits are the hex interpretation of the IP address, the "ool" is Optimum OnLine).

To that end, HostDB generates a different set of zone files for "internal" DNS servers (those inside the firewall) and "external" servers (those accessible to the public). HostDB generates both sets of zone files from the same hostdb.txt, thus preventing them from getting out of sync.

To make this work, HostDB assumes that a host should be hidden from outsiders unless marked otherwise. To change the default, add a "scope qualifier" to the end of the hostname. The scope qualifiers are @EXTERNAL, @EXTERNALONLY and @INBOUNDNAT. Display 3 shows a hostdb.txt that demonstrates all three.

database.example.com is a normal host. Insiders have the usual forward and reverse DNS data. Outsiders can not look up the host's IP address, and the reverse DNS information is the anonymous name of d64-32-179-55.example.com. Lookups of d64-32-179-55.example.com return a proper A record.

www.example.com is an externally exposed host. Therefore, the forward and reverse lookups work as one would expect for both internal and external users. Since "example.com" is an ANAME for this host, it behaves the same way.

vpn3000 and vpn demonstrate the @EXTERNALONLY scope. This keyword exposes a host externally but not internally. It is rarely used. In the above example we use it for a VPN (RAS) server which external people use to access internal resources. Thus, they shouldn't use it when they are inside the company. We wanted to be able to construct a client configuration that would fail when used inside the firewall, where use of the VPN should be unneeded. Thus we marked vpn.example.com as @EXTERNALONLY so that the DNS entry would appear in external zone files, but not in internal zone files. This achieved the goal. However, network administrators still needed to access the VPN server for administrative reasons. They could use the name vpn3000.example.com which resolves properly internally.

The last two rows of the table demonstrate that reverse lookups for the anonymous hostnames are properly generated. Any A record that is generated needs a proper PTR record, even when host hiding.

Not pictured in Display 3 is @INBOUNDNAT which solves a very common, but specific, problem. Suppose you have an internal host on a RFC1918 address (unrouted) network and someone has decided to poke a hole in the firewall to let outsiders access it. The firewall will do some kind of "reverse NAT" so that packets destined to a particular public address will be re-written and sent to an internal host. While the author feels this has risky security implications, at

```
64.32.179.55     database.example.com
64.32.179.56     www.example.com@EXTERNAL ANAMES=example.com@EXTERNAL
64.32.179.57     vpn3000.example.com      ANAMES=vpn.example.com@EXTERNALONLY
```

Host	Internal DNS Records		External DNS Records	
	A	PTR	A	PTR
database.example.com	64.32.179.55	database.example.com	FAILS	d64-32-179-55.example.com
www.example.com	64.32.179.56	www.example.com	64.32.179.56	www.example.com
example.com	64.32.179.56	www.example.com	64.32.179.56	www.example.com
vpn3000.example.com	64.32.179.57	vpn3000.example.com	64.32.179.57	vpn.example.com
vpn.example.com	FAILS	vpn3000.example.com	64.32.179.57	vpn.example.com
D64-32-179-55.example.com	FAILS	FAILS	64.32.179.55	d64-32-179-55.example.com
D64-32-179-56.example.com	FAILS	FAILS	64.32.179.56	vpn.example.com

Display 3: Demonstration of scopes.

least HostDB lets you get the DNS records correct when you choose to use this technique. Here's how one would do this with HostDB:

```
64.32.9.8   host.example.com@INBOUNDNAT
10.1.1.5    host.example.com
```

Scopes can pertain to each hostname listed on a line, whether it is the host, a CNAME or an ANAME. A host can have multiple ANAMES and CNAMES and each name can be normal, external, externalonly, or inboundnat independently of the others. The generation process becomes quite complicated. What is the right PTR record to generate if a host is externalonly but has an ANAME alias that is normal? (Our answer: the ANAME on the inside, the externalonly name on the outside). As new situation cropped up we had to fine-tune the algorithms.

Mail Servers

Marking a host as a mail server means the MX records should be slightly different. It should have the default MX records but also an MX record pointing to itself that is better priority than the others. You specify this in hostdb.txt with the ISMAILSERVER flag; see Display 4.

HostDB does not create duplicates, even in tricky situations like when the default MX record for a subnet is the same as the host. For example, if zoneconf. txt specifies that all hosts are to receive MX records of priority 10 for mta1.example.com and priority 20 for mta2.example.com and hostdb.txt contains the code in Display 5, then the example.com zone file would resemble Display 6.

Host "normal" receives both MX records. "msexchange" receives the MX records and the best priority MX pointing to itself. "mta1" and "mta2" have the best priority pointing to itself with a worse priority MX pointing to either mta2 or mta1 as appropriate.

DHCP Pools

HostDB provides assistance to anyone creating large DHCP pools. DHCP pools usually assign "generic" names to each IP address such as ool-24-12-12-12.comcast.net. Repetitive jobs should always be automated, thus there is a syntax for automatically generating such hostnames. A DHCP pool in the middle of a subnet might look like Display 7.

This would generate a host name called dhcp-10-1-40-4-pool.example.com for the first DHCP_POOL entry, and similar entries for the rest.

```
198.65.112.13   mta1.example.com          ISMAILSERVER
```

Display 4: Mail server specification via "ISMAILSERVER".

```
198.65.112.9    normal.example.com
198.65.112.10   msexchange.example.com   ISMAILSERVER
198.65.112.11   mta1.example.com         ISMAILSERVER
198.65.112.12   mta2.example.com         ISMAILSERVER
```

Display 5: Mail servers with varying priorities in zoneconf.txt .

```
normal1       IN A      198.65.112.9
              IN MX 10  mta1
              IN MX 20  mta2
msexchange    IN A      198.65.112.10
              IN MX 0   msexchange
              IN MX 10  mta1
              IN MX 20  mta2
mta1          IN A      198.65.112.11
              IN MX 0   mta1
              IN MX 20  mta2
mta2          IN A      198.65.112.12
              IN MX 0   mta2
              IN MX 10  mta1
```

Display 6: Resulting example.com zone file.

```
10.1.40.1   zathras-purple.example.com  ISROUTER=zathras.example.com
10.1.40.2   server2.example.com
10.1.40.3   server3.example.com
DHCP_POOL_TEMPLATE dhcp-$a-$b-$c-$d-pool.example.com
10.1.40.4   DHCP_POOL
10.1.40.5   DHCP_POOL
10.1.40.6   DHCP_POOL
10.1.40.7   DHCP_POOL
10.1.40.8   DHCP_POOL
10.1.40.9   DHCP_POOL
10.1.40.10  host10.example.com
```

Display 7: DHCP pool in the middle of a subnet.

The template takes effect from the point in the file it appears and continues until another DHCP_POOL_TEMPLATE line overrides it. For example, at one location they wanted to have slightly different hostnames for a DHCP pool for IP phones. In that address range, they put the lines shown in Display 8.

There is no coordination between the DHCP_POOL template and the DHCP configuration file that is generated. That is, if you use the DHCP_ POOL feature to generate a range of similarly-named systems, you must manually update the DHCP configuration file to include those ranges.

A future enhancement would be to have the template be a stack, permitting one to more easily switch to a new template and switch back when one is done.

While it might be useful to be able to specify a range of IP addresses (rather than to list one per line) this feature has been delayed until a clear syntax can be developed. A range would violate the ''one line per address'' rule that the sysadmins preferred. It turns out they often use their text editor's command to cursor-down 256 times to jump from subnet to subnet. Thus, any command that consolidates repetitive lines makes navigation more difficult.

However, a command-line tool is provided to generate lists of IP addresses with a template. The command was used to create the above range:

```
genrange 10.1.100.4 10.1.100.100 \
                    '$ip<tab>DHCP_POOL'
```

Zone Configuration

Zoneconf.txt stores everything required to generate the zones outside of the host information itself. Settings here change rarely and are usually the purview of senior DNS engineers. By keeping these options separate we avoid accidental modifications by junior engineers. The lines of this file are executed in order, like a sequential programming language.

The file usually starts by setting some options:

```
# When a hAAA-BBB-CCC-DDD external name
# is created, what domain is it in?
OBSCUREZONE example.com
```

OBSCUREZONE indicates the domain name to use for obscured, or anonymous, hostnames in external zone files.

SOA, MX, and ZONESERVERS sets the SOA values, default MX records, and NS records to be included in any zone file created after this point; see Display 9.

Notice that most commands are repeated once for INTERNAL and once for EXTERNAL. These settings are different for internal and external zones. There is also ''SLAVE'', which is an internal secondary. There is no support for external slave servers at this time as they tend to be at ISPs and don't let customers update their DNS server configurations directly.

When the system generates the named.conf file it will need to know some parameters to insert into any ''zone'' setting. For example, we need to specify who is allowed to do zone transfers and who is allowed to do dynamic DNS updates. We set those values as shown in Display 10.

Now the exciting part where we actually create some zone files. We do these with commands like DOMAIN and REVDOMAIN; see Display 11.

These lines direct HostDB to generate the zone file for corp.example.com, then example.com, then ex.com.

```
DHCP_POOL_TEMPLATE phone-$a-$b-$c-$d-pool.example.com
10.1.100.4  DHCP_POOL
...elided for space...
10.1.100.100     DHCP_POOL
DHCP_POOL_TEMPLATE dhcp-$a-$b-$c-$d-pool.example.com
```

Display 8: Example template for an IP phone pool.

```
SOA INTERNAL hostmaster.example.com 1H 15M 30D 60M
SOA EXTERNAL hostmaster.example.com 1H 15M 30D 60M

MX INTERNAL 10 lucy.example.com ; 20 ingw2.example.com
MX EXTERNAL 10 exgw2.example.com ; 10 exgw4.example.com

ZONESERVERS INTERNAL lucy3.example.com ingw2.example.com
ZONESERVERS EXTERNAL exgw4.example.com exgw2.example.com
```

Display 9: SOA, MX, and ZONESERVERS examples.

```
ALLOW-TRANSFER  INTERNAL        "primary_servers"
ALLOW-TRANSFER  SLAVE           "primary_servers"
ALLOW-TRANSFER  EXTERNAL        "rev65_servers"

ALLOW-UPDATE    INTERNAL        "none"
ALLOW-UPDATE    SLAVE           "none"
ALLOW-UPDATE    EXTERNAL        "none"
```

Display 10: Zone transfer and dynamic DNS update permissions.

Order is important here. After a host's DNS records are output they will not be output again. If example.com precede corp.example.com, the zone file for corp.example.com would be nearly empty, containing only the required SOA and NS records.

Notice that ex.com is only an external zonefile. That's permitted. Domains can be internal-only too. If you mix and match HostDB will do the right thing. This can be useful for subdomains that exist only inside the company.

Now we can generate the reverse lookup zone files; see Display 12. The REVDOMAIN options indicate what size in-addr.arpa zone file to create (class A, B, or C), the starting IP address, and whether this is internal only or both internal and external.

There are a few tricks one can do. For example, since the configuration is ''executed'' line by line, you can change settings between zones. For example, if you had different MX records for a particular domain, you can change along the way; see Display 13.

This would generate corp.example.com with lucy and ethel as the default MXs. Then the default MXs would changed to betty and wilma before prod.example.com is generated. Then the SOA defaults would be changed before the reverse domain 197.32.64.in-addr.arpa would be generated.

Rudimentary support for RFC 2317-style ''classless'' delegations is implemented but not complete. Eventually there will be a REVRANGE command that accepts a range like that shown in Display 14. This will generate a zone file with the specific contents required for RFC 2317.

Special Cases

While HostDB generates a file named.conf file it generates a very standard ''zone {}'' stanza for each DOMAIN command. If you need something very specialized, one can use CUSTOMDOMAIN instead. This will generate no ''zone {}'' stanza and rely on you to add one in manually using other means explained later.

This is particularly important for supporting Dynamic DNS updates. HostDB supports you by giving you enough rope to hang yourself.

Summary

That's all there is to it! You specify what zone files to create and it generates them plus the named.conf file. All the hard work is done for you. If extra lines need to be added to a zone file or configuration file, that can be done with the notation described in the next section. When the system is fully configured it should be able to generate all files without any manual intervention.

Header and Footer Files

HostDB can't be all things to all people, so there is an ''emergency escape hatch'' that lets you customize any file that is generated. You can add a header file and footer file to any file created by HostDB. For any file X generated by HostDB, the file that is actually created is made up of:

1. The contents of file $TEMPLATES/X-head (if this file exists).
2. The content generated by the HostDB system.
3. The contents of the file $TEMPLATES/X-tail (if this file exists).

```
DOMAIN corp.example.com      INTERNAL   EXTERNAL
DOMAIN example.com           INTERNAL   EXTERNAL
DOMAIN ex.com                           EXTERNAL
```

Display 11: Creating zone files using DOMAIN.

```
REVDOMAIN CLASSC 64.32.197.0   INTERNAL EXTERNAL
REVDOMAIN CLASSC 64.32.198.0   INTERNAL EXTERNAL
REVDOMAIN CLASSC 10.1.0.0      INTERNAL
REVDOMAIN CLASSC 10.1.255.0    INTERNAL
REVDOMAIN CLASSB 10.100.0.0    INTERNAL
REVDOMAIN CLASSC 10.254.0.0    INTERNAL
REVDOMAIN CLASSC 172.17.17.0   INTERNAL
```

Display 12: Creating reverse look up zone files using REVDOMAIN.

```
SOA EXTERNAL hostmaster.example.com 1H 1H 30D 60M
MX INTERNAL 10 lucy.example.com ; 20 ethel.example.com
DOMAIN corp.example.com      INTERNAL   EXTERNAL

MX INTERNAL 10 betty.example.com ; 20 wilma.example.com
DOMAIN prod.example.com      INTERNAL   EXTERNAL

SOA EXTERNAL hostmaster.example.com 2H 2H 30D 60M
REVDOMAIN CLASSC 64.32.197.0   INTERNAL EXTERNAL
```

Display 13: Specifying varied MX records for a particular domain.

```
REVRANGE 22.33.44.32 22.33.44.63 32-63.44.33.22.in-addr.arpa INTERNAL EXTERNAL
```

Display 14: Future style of RFC 2317-style range specification.

(The file name suffixes ‘‘head’’ and ‘‘tail’’ are used instead of ‘‘footer’’ and ‘‘header’’ because they sort better that way.)

This feature is useful for:

- Sneaking special records into a zonefile. If HostDB won’t generate a zone the way you want it, add the lines to a -tail file.
- Adding custom zones to a named.conf file. If there are zones not under HostDB management you can still add information about them in named.conf
- Coarsely approximating an ‘‘include file’’ mechanism. If a DHCP server configuration file doesn’t have an ‘‘include’’ mechanism, you can wrap the contents of a file around the output of HostDB’s dhcpd.conf

DHCP Generation

HostDB parses the hostdb.txt file and creates the ‘‘host’’ statements appropriate for ISC’s DHCP server. One stanza is created for each host that has a MAC= setting. No stanzas are created for hosts without MAC= settings.

A hostdb.txt line that looks like Display 15 would generate the default template like that shown in Display 16. This is appropriate for most systems. (Note that the MAC address is reformatted to be pretty by capitalizing the letters and zero-padding.)

However, if TYPE=foo is included, as alternate template is used. For example we might want a different template if we know the machine has a broken DHCP client that gets confused by extra data. A hostdb.txt line like this would select the template called ‘‘win95’’ which is intentionally minimal (see Display 17) generates the code shown in Display 18.

If the template includes a hyphen, the text after the hyphen is passed to the template as a parameter. Thus templates can be smart. Suppose there is a default netboot server and a special one for developers. Templates can be constructed that produce proper DHCP configuration for each; see Display 19.

Currently the templates are written as subroutines in Perl and are hard coded into the mkdhcp script. Future versions will move these templates to their own files, though this work has been deferred to later because (1) the need for special templates to handle broken DHCP clients is reduced over time as more vendors figure out how to write DHCP clients that aren’t broken, (2) nobody’s really complained about the system as it is.

Deployment (Phear My Aw4z0m3 Makefile Skilz!)

Generating zone files and configuration files is not enough. They must be deployed to be useful.

Rather than generate files directly where they will be used, HostDB generates all files in a directory called ‘‘out’’ (short for output). Sites with a very large legacy system for DNS/DHCP management can simply copy the newly generated files (or just the files that are useful to them) out of this directory. However, HostDB includes a tool that makes this even easier called mkdestination. It generates a Makefile that ‘‘does the right thing.’’

Mkdestination takes ‘‘destination.txt’’ for input, and generates ‘‘destination.mk’’ which can be used in the Makefile that drives your DNS/DHCP service.

Consider short example destination.txt file shown in Display 20. The format is a simple list source files, the ‘‘->’’, then destinations. Destinations

```
10.10.244.14 dell4.example.com MAC=00:5:3b:F1:21:7a
```

Display 15: Sample hostdb.txt file.

```
host dell4.example.com {
       hardware ethernet 00:05:3B:F1:21:7A;
       fixed-address 10.10.244.14;
       option host-name "dell4.example.com";
       ddns-hostname "DELL4";
}
```

Display 16: Generated default template from hostdb.txt file above.

```
10.10.244.14 dell4.example.com MAC=00:5:3b:F1:21:7a TYPE=win95
```

Display 17: Minimal template for ‘‘win95’’.

```
host dell4.example.com {
       hardware ethernet 00:05:3B:F1:21:7A;
       fixed-address 10.10.244.14;
}
```

Display 18: Code generated by template above.

```
10.10.244.13  human1.example.com MAC=00:05:3B:F1:21:7A TYPE=netboot
10.10.244.14  human2.example.com MAC=00:05:3C:E3:32:A4 TYPE=netboot-main
10.10.244.15  human3.example.com MAC=00:05:32:A2:01:12 TYPE=netboot-dev
```

Display 19: DHCP configuration templates.

can be a file name or directory on the local machine or some other. There is no macro expansion or other substitution mechanism because we want to keep the format simple.

The first line says that to deploy INTERNAL.example.com and INTERNAL.198.32.64.in-addr.arpa one should copy it into the local directory /var/named, and scp it to a /var/named on a machine called ''betty''. The next two lines specify that a file is to be copied to a particular filename. This permits one to rename files as they are copied.

Display 21 shows how one might use mkdestinations in a master Makefile.

Invoking ''make push'' would generate the destinations.mk file (if needed) then run the ''all'' recipe. The recipes in the destinations.mk file ''do the right thing'' for all files mentioned in destinations.txt. ''make push-local'' is similar but runs the ''all-local'' recipe, which only deploys files destined for the local machine. This is useful for testing purposes.

Mkdestinations is needed to overcome limitations in ''make''. ''Make'' has no awareness of remote file timestamps, nor can ''make'' handle a dynamic list of zones and files with multiple destinations.

Efficient Updates of Remote Files

Make has no idea if betty:/var/named/named.conf is up to date because it is on a different machine. Therefore, mkdestination compensates by creating local timestamp files after a remote operation is complete (that is, after a remote file is updated successfully it touches a local file whose name encapsulates the destination server and filename). Future timestamp comparisons can be done against the local file. Rather than recopying a file to a remote machine ''just in case'', the timestamps can be compared. In the case of copying a file to betty:/var/named/named.conf, the timestamp file ds/ betty__var_named_named.conf is created. (The actual filename is much more complicated to encode special characters that might cause problems.)

Mkdestination also creates a recipe called ''clean'' which deletes all timestamps. Thus, to force a ''push'' to all other hosts, simply ''make clean'' then ''make push'' and all files will be copied whether they need it or not.

Efficient Zone Serial Number Updates

The other interesting feature of mkdestination is that it is smart about zone serial numbers.

DNS zone files have an embedded serial number which is used to determine when zone transfers are to happen. If a secondary has a zone of serial number X, and is told that the primary has a higher serial number, it requests a download of that zone (a ''zone transfer''). Zone transfers are a CPU- and (possibly) network-intensive operation. We want to prevent gratuitous zone transfers.

The problem, however, is that the only way to know if a zone file will change is to generate the new one and compare it to the old one. The comparison can't be a simple ''cmp'' or ''diff'' because the serial number inserted into the zone file will be different every time. Most changes to hostdb.txt tend to affect only a few zone files.

Again, mkdestination to the rescue. mkdestination knows about serial numbers and can do comparisons that ignore the serial number line of a zonefile. The methodology is as follows.

HostDB generates all files in a directory called ''out''. Any file that needs a serial number contains the text '':serial:'' anywhere a serial number is to be placed. This makes comparison of zone files easier. Any file that did change is copied to a directory called MP and another copy is placed in a directory called MS. The MP directory contains a plain copy of the zone but the MS directory contains a copy of the file with '':serial:'' changed to the serial number used for all files generated at that time. When deciding which files need to be deployed, the timestamp of MS is used to determine if the file needs to be copied but the file that is copied comes from MP.

After a file is successfully deployed, a timestamp file is created in the DS directory marking the comple tion. In the case of copying a file to betty:/var/named named.conf, the timestamp file with a name like DS/ betty__var_named_named.conf is created. The result is

```
INTERNAL.example.com INTERNAL.198.32.64.in.addr.arpa -> \
                                    /var/named/. secondary:/var/named/.

INTERNAL.named.conf -> /etc/named/named.conf

SLAVE.named.conf -> betty:/etc/named/named.conf
```

Display 20: Sample destination.txt file.

```
destinations.mk: ../destinations.txt
      mkdestinations <../destinations.txt >destinations.mk

push: destinations.mk
      make -f destinations.mk all

push-local: destinations.mk
      make -f destinations.mk all-local
```

Display 21: Using mkdestinations in a Makefile.

that files are only copied when absolutely needed. It is optimal.

At a site with dozens of zones the DNS servers were being hit fairly hard by all the zone transfers. After mkdestination's technique was deployed the administrators were very happy to see that very few zones were actually being transferred after typical updates. It was quite impressive to see many, many zone files with an assortment of serial numbers as diverse as the history of updates had dictated. Yet, no tool more complicated than ''mkdestination'' and ''make'' was needed.

If this sounds confusing to you, just be happy in the knowledge that you don't have to understand it to benefit from the efficiencies. Files that don't need to be copied across your network aren't copied.

Deployment Examples

Here are two examples of how to deploy HostDB. First we will consider a new site starting from scratch. Then we will see how easy it is to deploy in a pre-existing environment.

We need a place to put our files. The HostDB files can be put in any directory as your site's requirements dictates (/var/hostdb or /home/adm/hostdb is typical). HostDB assumes that all executables will be found in the shell's PATH. As a result, test datasets can be placed anywhere (even /tmp). When testing new executables, one can simply put the newer versions ahead of the older ones in the shell's PATH.

In these examples we will use /var/hostdb and assume the PATH is set correctly.

Example 1: Starting from Scratch

In this example we are starting from scratch. Imagine we are setting up a new network for a new company. We have one external network (64.32.179.0/24) and one internal network (10.1.1.0/24). There will be a dhcp pool from .140 to .250 in the first network.

We begin by generating the hostdb.txt file. We generate a list of each IP address. That way when a junior engineer goes to allocate an IP address, they don't have to understand anything but (1) find an unused address on the right subnet, (2) remove the comment symbol and change the hostname as appropriate. This prevents them from introducing typos into the list of IP addresses, and prevents mistakes such as thinking they can create new IP subnets by simply editing this file. It also keeps the file neat and orderly in case you work with people that can not be trusted to keep a file in numerical order.

Display 22 shows commands that will generate a hostdb.txt that is a good starting point.

- Line 1 creates commented-out sample lines for the first 140 addresses.
- Line 2 creates addresses for our DHCP pool. Note the custom template.
- Line 3 completes the internal network's addresses.
- Line 4 uses a different template for the external hosts. It also demonstrates that if the second IP address is replaced by an integer, it is interpreted as a count of how many lines to output instead of an end address.

Display 23 shows the output.

Now edit the hostdb.txt to include the initial hosts that will be installed. To make sure that nobody

```
genrange  >hostdb.txt -d example.com 10.1.1.0   10.1.1.139
genrange >>hostdb.txt -d example.com 10.1.1.140 10.1.1.250 '$ip\tDHCP_POOL'
genrange >>hostdb.txt -d example.com 10.1.1.251 10.1.1.255
genrange >>hostdb.txt -d example.com 64.32.179.0 256 '#$ip\tex$hex.$DOMAIN\@EXTERNAL'
```

Display 22: Generating hostdb.txt from scratch.

```
#10.1.1.0       UNUSED0A010100.example.com
#10.1.1.1       UNUSED0A010101.example.com
...elided for space...
#10.1.1.138     UNUSED0A01018A.example.com
#10.1.1.139     UNUSED0A01018B.example.com
10.1.1.140      DHCP_POOL
10.1.1.141      DHCP_POOL
...elided for space...
10.1.1.249      DHCP_POOL
10.1.1.250      DHCP_POOL
#10.1.1.251     UNUSED0A0101FB.example.com
#10.1.1.252     UNUSED0A0101FC.example.com
#10.1.1.253     UNUSED0A0101FD.example.com
#10.1.1.254     UNUSED0A0101FE.example.com
#10.1.1.255     UNUSED0A0101FF.example.com
64.32.179.0     ex4020B300.example.com
64.32.179.1     ex4020B301.example.com
...elided for space...
64.32.179.254   ex4020B3FE.example.com
64.32.179.255   ex4020B3FF.example.com
```

Display 23: Output from above commands.

tries to allocate a broadcast address, we mark the ‘‘all zeros’’ address as ‘‘foo-net.example.com’’ and the ‘‘all ones’’ address as ‘‘foo-bcast.example.com’’ where ‘‘foo’’ is a unique name for each subnet. Display 24 shows the non-comment lines.

Display 25 shows our zoneconf.txt file, which is almost the same as the example provided with the software. Most sites can use this template and simple change ‘‘example.com’’ to their domain and edit their MX records and ZONESERVERS to suit their needs.

The last three lines show which zones to generate. First the zone for example.com, which has different values for the internal and external version. Next we generate the reverse lookup zone for 10.1.1.0/24, which only generates an internal zone file. Lastly we generate the 64.32.179.0/24 zone reverse lookup which has different internal and external values.

Copy the template Makefile into the main directory and we are ready for our first attempt at generating our zones. To save ourselves typing, we’ll create an alias for bash/sh/ksh or csh/tcsh (see Display 26). Then we can do ‘‘makeh’’ to generate a new set of files. If we are happy we can do ‘‘makeh push’’ to deploy the files. As a result, the ‘‘out’’ directory now contains:

```
10.1.1.0        main-net.example.com
10.1.1.1        zathras-main.example.com ISROUTER=zathras.example.com
10.1.1.2        fileserver.example.com
10.1.1.3        mailserver.example.com   ISMAILSERVER
10.1.1.4        vector.example.com
10.1.1.10       staffpc1.example.com MAC=00:b0:d0:a6:cf:f1
10.1.1.11       staffpc2.example.com MAC=00:b0:d1:a7:c0:d1
10.1.1.11       staffpc2.example.com MAC=00:b0:c2:b3:c3:d3 TYPE=freebsd
10.1.1.140      DHCP_POOL
10.1.1.141      DHCP_POOL
10.1.1.142      DHCP_POOL
...elided for space...
10.1.1.249      DHCP_POOL
10.1.1.250      DHCP_POOL
10.1.1.255      main-bcast.example.com@EXTERNAL
64.32.179.0     ext-net.example.com@EXTERNAL
64.32.179.1     isp-router.example.com@EXTERNAL
64.32.179.2     zathras-ext.example.com@EXTERNAL ISROUTER=zathras.example.com@EXTERNAL
64.32.179.3     mailqueue.example.com@EXTERNAL ISMAILSERVER
64.32.179.4     vector.example.com@INBOUNDNAT
64.32.179.5     exweb.example.com ANAME=www.example.com@EXTERNAL
64.32.179.255   ext-bcast.example.com@EXTERNAL
```

Display 24: Edited hostdb.txt file.

```
TEMPLATEDIR ..
OBSCUREZONE example.com

SOA INTERNAL hostmaster.example.com 3h 1h 1w 1h
SOA EXTERNAL hostmaster.example.com 3h 1h 1w 1h

MX EXTERNAL 10 mailserver.example.com
MX INTERNAL 10 mailqueue.example.com ; 20 s1.isp.com

ZONESERVERS INTERNAL fileserver.example.com mailserver.example.com
ZONESERVERS EXTERNAL mailqueue.example.com exweb.example.com

ALLOW-UPDATE   INTERNAL         done
ALLOW-UPDATE   INTERNAL         none
ALLOW-UPDATE   SLAVES           none
ALLOW-UPDATE   EXTERNAL         none

DOMAIN example.com                INTERNAL EXTERNAL
REVDOMAIN CLASSC 10.1.1.0         INTERNAL
REVDOMAIN CLASSC 64.32.179.0 INTERNAL EXTERNAL
```

Display 25: Example zone.txt file.

```
# bash/sh/ksh alias:
alias makeh='cd /var/hostdb/out && make -f ../Makefile'

# csh/tcsh alias
alias makeh 'cd /var/hostdb/out && make -f ../Makefile'
```

Display 26: Shell ‘makeh’ alias definitions.

```
$ ls -1
EXTERNAL.179.32.64.in-addr.arpa
EXTERNAL.example.com
EXTERNAL.named.conf
EXTERNAL.named.root
INTERNAL.1.1.10.in-addr.arpa
INTERNAL.179.32.64.in-addr.arpa
INTERNAL.example.com
INTERNAL.named.conf
SLAVE.named.conf
```

Notice that each filename is prefixed with "INTERNAL.", "EXTERNAL." or "SLAVE.". All other files are tagged as being appropriate for either the inside or the outside. EXTERNAL.named.root is the NIC's "root cache" file which the Makefile automatically retrieves via FTP.

Now check the zone files by manual inspection. Correct any errors by editing ../hostdb.txt and re-running "makeh" until you are satisfied.

Notice that in addition to zone files, ISC BIND named.conf files are generated. If you examine them, you'll find them very anemic. They just contain the "zone {}" entries which are automatically generated. To create a complete file, determine what you would put before and after the generated parts and put them in the proper -head and -tail files. Any text in ../INTERNAL.named.conf-head is prepended to the generated file, and any text in ../INTERNAL.named.conf-tail is appended to the end of the generated file. The same is true for all files generated by the system.

Now that we are happy with the files being generated, it's time to tell HostDB where to put them. To do that, we create ../destinations.txt which lists where each file is to be copied; see Display 27.

- Line 1 specifies that EXTERNAL.named.conf is to be copied to mailqueue.example.com in a directly called /etc/namedb. It is renamed to named.conf as it is copied. It is also copied to exweb.
- Line 2 specifies three files that are to be copied to two servers in their /var/named directories. The files are not renamed as they are copied. Instead, we will make sure the configuration files that refer to these files must specify the filename as they exist.
- Line 3 is like line 1 but for the internal servers.
- Line 4 is analogous to line 2.

This command will generate a makefile that will do the actual copying: makeh destinations.mk. Finally, "makeh push" will actually copy the files into place. A few rounds of debugging and the system is deployed.

Example 2: Replacing a Legacy System

In this scenario we have a working system where we hand-edit zone files and copy them using a shell script. To move from a legacy system to HostDB requires a lot of testing. HostDB includes utilities that help every step.

The first and most difficult step is to convert the zone files and turn them into hostdb.txt format. This is the reverse of what HostDB does! Sadly, we can't reverse the polarity of the flux capacitor and have everything "just work." Luckily, the HostDB package includes zone2hostdb which does 90% of the work for you. It takes a zone file as input:

```
zone2hostdb <zonefile >hostdb.txt-base
```

Since this step is automated we reduce the potential for mistakes considerably.

This gives you a first draft of what hostdb.txt should be. Now check these issues:

- Verify all mail servers are listed as ISMAILSERVER
- Verify that all routers and multihomed hosts are marked as ISROUTER or ISMULTIHOMED as appropriate
- Mark any broadcast addresses so they are not accidentally used
- Check any ANAMEs to verify that the official name is listed first on the line and that actual aliases are listed in an ANAME. This assures proper PTR records
- If any special cases are in place for PTR records, make sure the name to be used for the PTR record is always the first hostname on the line
- Set the scope to @EXTERNAL for any hosts accessible externally
- Set the scope to @INBOUNDNAT for any host that is accessed by external users by a different address than internal users
- Set the scope to @EXTERNALONLY for any hostnames only external users should access (very rare)
- Anything else "special" about your DNS zones

```
EXTERNAL.named.conf -> mailqueue:/etc/namedb/named.conf  \
                       exweb:/etc/namedb/named.conf

EXTERNAL.example.com EXTERNAL.179.32.64.in-addr.arpa \
                EXTERNAL.named.root -> \
                mailqueue:/var/named/. exweb:/var/named/.

INTERNAL.named.conf -> fileserver:/etc/namedb/named.conf \
                       mailserver:/etc/namedb/named.conf

INTERNAL.example.com INTERNAL.1.1.10.in-addr.arpa \
   INTERNAL.179.32.64.in-addr.arpa -> fileserver:/var/named/. \
   mailserver:/var/named/.
```

Display 27: File ../destinations.txt: where files are to be copied.

The next step is to use genrange to enumerate every IP address that should be listed in your hostdb.txt file, whether it is commented out or not. We use the same ‘‘genrange’’ commands as in the first example except we save the output to a file called hostdb.txt-enum.

HostDB includes a utility called ‘‘mergeiplists’’ that will merge these two lists properly even though some of the lines are commented out. It assumes the first appearance of an IP address is the authoritative line and throws the others away. Thus, we list the hostdb.txt-base first because our hand-edited file should be authoritative; hostdb.txt-enum is just to fill in the gaps; see Display 29. Now we have a pretty decent draft of our hostdb.txt file.

Before we can put it into use, we should test it extensively and then only switch to it incrementally.

Testing is made easier by a utility that is included called ‘‘canonzone’’. Comparing two zone files can be difficult because of small changes in white space, sorting, and so on. ‘‘canonzone’’ reads a zone file and outputs it in a very specific, clean, regular, format (a canonical format). If you are comparing a legacy zone file and a generated zone file, pass both thought canonzone first and the comparison will be much easier. The distribution includes a file called ‘‘Example_comparezones’’ which takes two files, passes them through canonzone, strips them down to just DNS A records, and compares them.

Here are some tips for testing your new zones:

- Use Example_comparezones as a template for your own comparison scripts.
- Review each email server and verify the MX records are as required.
- Review all other servers and verify their DNS records are as expected.
- Review each external host and verify all of its records (test from inside and outside).
- Set up a new DNS server and load the zones. Make sure they load without outputting any errors or warnings in BIND’s logfile.
- Query the DNS server to make sure you get expected results for mail servers and other important hosts. When you are satisfied, ask co-workers to do the same.
- When you are satisfied, configure yourself and a few trusted users’ machines to use this new DNS server. If anything breaks, fix it. After a few days, have your coworkers switch to this DNS server. Make sure they know to report problems to you.

When you are satisfied with the zone files, you can slowly migrate them into your legacy system. The destinations.txt file can be used to roll out just the specific zones you are confident in.

Suppose your legacy system loads your main zone from a file called /var/named/zone.example.com..zone, you can replace that file with the HostDB-generated file with a statement like that shown in Display 28. Obviously you should make good backups before deploying any new zones.

To push that zone out, ‘‘makeh push’’.

Test, test, test. Be ready to revert to the legacy zone if the problems you find are insurmountable. Follow any in-house procedures related to change windows and so on. For example, do not introduce new zones:

- Right before you leave for vacation
- The same week as the quarterly results or taxes are being prepared
- The week of a major deadline for your company
- The week raises and promotions are being decided

Lastly, you can replace your named.conf files with the ones that are generated by HostDB (though if your zone filenames change, be very careful!). You should also integrate the Makefile into your system so that it controls all DNS-related activity from one central point.

Related Work

There are many other open source systems that do some or part of what HostDB does. There are even a few that do a lot more, especially where dynamic registration is involved. However, the goal of HostDB is to maximize features without having to resort to requiring a web front-end or SQL backend. The author believes HostDB is superior to all systems that don’t require an SQL backend, and easier and faster to install than any of the systems that require SQL databases.

Here is my comparison to some popular utilities:

- h2n – O’Reilly’s ‘‘DNS and BIND’’ by Paul Albitz and Cricket Liu comes with h2n (and an enhanced version is called h2n-hp). These are fine DNS generator scripts. While it can be used for one-time conversion, there are features that let you maintain the host list as a /etc/hosts-like file. However, this is a ‘‘kit’’, not a complete system. It does not include anything like mkdestinations for deployment. No DHCP support. Source: www.oreilly.com/catalog/dns3 .
- DNSDusty – While this system does automate deployment of the files it generates, it requires a web interface (but no SQL database). It makes adding a host very easy for a junior sysadmin once it is set up. No DHCP support. Source: www.poochiereds.net/dnsdusty .

```
INTERNAL.example.com -> dnsserver:/var/named/zone.example.com..zone
```

Display 28: Replacing a legacy zone file.

```
mergeiplists hostdb.txt-base hostdb.txt-enum >hostdb.txt
```

Display 29: Merging lists.

- dnscvsutil – Maintains your DNS zone files under CVS control and automatically updates reverse zones. Helping someone do a better job of maintaining hand-edited zone files is exactly the opposite of what I want to encourage. Though, revision control is better than no revision control. No DHCP support. Source: freshmeat.net/projects/dnscvsutil .
- Sauron – Sauron is a free DNS/DHCP management system with Web and command line interfaces. It can manage multiple servers and generates complete dhcpd and named configurations from the database. This is a very complete and impressive package. It manages deployment of generated files. However, it does require a web interface and SQL database. Source: http://sauron.jyu.fi/ .
- Updatehosts – A very comprehensive system for managing DNS information. This is popular at many large commercial sites. However, the learning curve seems overly steep as it requires understanding of database relations. This is not for the neophyte or someone trying to set up a new site quickly. Requires SQL. No DHCP support. Source: ftp://ftp.tic.com/pub/updatehosts .

The Future

New features are added to HostDB on a regular basis.These projects are being considered:

- Make testing even easier: mkdestination should create a ‘‘diff’’ recipe that diffs the last distributed against the most recently generated files.
- Eliminate the need to modify the default Makefile: Move ‘‘restart commands’’ to the mkdestination system.
- Add classless delegation support: Complete support for RFC 2317-style ‘‘classless’’ delegations.
- Add IPv6 support: Add support for IPv6 records. (Those long addresses aren’t going to be fun to type. We need some human-factors help here.)
- Improve legacy conversion: Update zone2hostdb to read multiple zones, including reverse DNS zones, and use the information to do a better job. In particular, PTR records could be an indicator of which name is a canonical name and which is an ANAME.

Conclusion

HostDB generates really cool DNS zone files with all the features required by sites with complicated DNS configurations, such as aliases, ‘‘external only’’ names, ‘‘host hiding’’ and multihomed hosts. It also generates the ISC BIND configuration files required for primaries, secondaries, and external DNS servers. DHCP configuration for static leases are generated for ISC DHCP using a very flexible template-based mechanism. It is easy to configure and deploy, yet sophisticated behind the scenes. It does not offer complicated features such as real-time host authorization, web front-ends, or any feature that required sophisticated and difficult to maintain databases.

Deploying the files is made easier through the use of a simple Makefile that is generated for you. The updates are very sophisticated, doing the minimal number of updates locally and to remote machines, even being careful not to create ‘‘zone transfer storms’’ by not updating a new serial number for a zone that does not warrant it.

Availability

The software open source and is available at http://www.everythingsysadmin.com/hostdb on the web.

Acknowledgements

I would like to thank Glenn Sieb for his feedback when designing the file formats, David H. Potter for assistance developing the mkdestination algorithm, Joe Gross who read the most painful early drafts of this paper and helped me turn it into a much better paper, and Josh Simon for his editing and proofreading assistance.

References

Albitz, Paul, Cricket Liu, *DNS and BIND, Fourth Edition*, O’Reilly, April, 2001.

Cheswick, William R., Steven M. Bellovin, Aviel D. Rubin, *Firewalls and Internet Security: Repelling the Wiley Hacker, 2nd Edition*, Addison-Wesley, February, 2003.

Christiansen, Tom, Nathan Torkington, *Perl Cookbook, Second Edition*, O’Reilly, August, 2003.

Limoncelli, Thomas A. and Christine Hogan, *The Practice of System and Network Administration*, Addison-Wesley, 2002.

Wall, Larry, Tom Christiansen, Jon Orwant, *Programming Perl (third edition)*, O’Reilly, July, 2000.