

CSIS0234B Computer and Communication Networks (Class B)

Reading for Tutorial 5

Using and Managing DNS Servers

We have two ways to address computers: by IP addresses like 147.8.178.15, and by names like `study.csis.hku.hk`. Now is time to see how they are related. The Domain Name System (DNS) is the facility that provides the mapping. Conceptually, the task is that given a “domain name” like `study.csis.hku.hk`, one would like to translate it to the corresponding IP address 147.8.178.15. The question is **who** to keep the information, and **how** to get it. The DNS facility has much more feature than that, however. In this notes, we will briefly see how the whole system works.

1. Domain name abstraction

Recall that each computer can have multiple IP addresses, one for each “interface” (dialup connection, Ethernet card, etc). So DNS cannot be just a one-to-one mapping. Interestingly, each computer can have multiple DNS address as well, so it cannot be a many-to-one mapping either. Usually, this is done to give symbolic names to services. E.g., in our department, `news.csis.hku.hk` and `www.csis.hku.hk` are actually the same computer, with yet another name `webinfo.csis.hku.hk`. This is for administrative purpose: In case when there are consistently many users connecting the web service, making the news service slow, then the two can be separated to two computers, without any interference to the users (who will continue to use the names `www.csis.hku.hk` and `news.csis.hku.hk`).

DNS contains **many** many-to-many mappings. Each mapping is from domain names to “resources records”, which can contain many type of information, which is not constrained to IP addresses. For example, it can hold IP addresses for some special purpose: the standard defines a mapping for IP address for **mail exchange**, so the same domain name can resolve to different IP addresses when it is used generically and when it is used for delivering mails, allowing organizations to make sure all its mails are processed in the same computer. Another mapping stores a **cryptographic key**, to ease programs that want to communicate securely¹. It can also hold **another domain name**. This is usually done for two purposes. First, most computers with multiple addresses will want one of them to be “**canonical**”, to make it easier to know whether two domain names refer to the same computer. Second, we usually want to do “**reverse lookup**”: given an IP address, we want to be able to find its domain name. (We will see how to express an IP address as a domain name, to be used for input for DNS). Each mapping is said to be a **type**:

A	A 32-bit integer for its IP address.
SOA	“Start Of Authority”: the top-node in a “zone”.
NS	The authoritative DNS server for a domain name.
MX	The host willing to process mails for a domain name.
CNAME	The canonical name for a host.
PTR	Point to another domain name, usually for reverse lookups.
HINFO	The CPU and OS used by a host.
KEY	The cryptographic key for a domain name.

As you can see, the facility is quite generic. It is in fact even more generic: apart from allowing

¹For this to be effective, the DNS system itself must be secure. Traditionally this is not the case, but the infrastructure needed to secure down DNS is already there. It takes time for system admins to update their systems, however.

many types, it allows many **classes** for different types of networks. However, since everything we will see is in the same **IN** class (i.e., Internet), we will not further investigate in this direction.¹

Domain names have a form that everybody should know about: **labels** (e.g., `www`, `csis`, etc.) separated by the “dot” character. Previously, labels must consist of zero or more (case insensitive) letters, digits and hyphens, with the first character constrained to a letter. This constraint has been relaxed, so now it is possible to have labels of international characters. The zero-character domain is reserved as the “root” label. The labels are interpreted backwards: `www.csis.hku.hk.` means the `www` of `csis` of `hku` of `hk` of the root. Such names are said to be **absolute**, as supposed to **relative** names which does not contain the last dot. E.g., within our department, it suffices to just say `www` instead of the above full name.

2. Elements of the DNS system

The DNS system consists of several components:

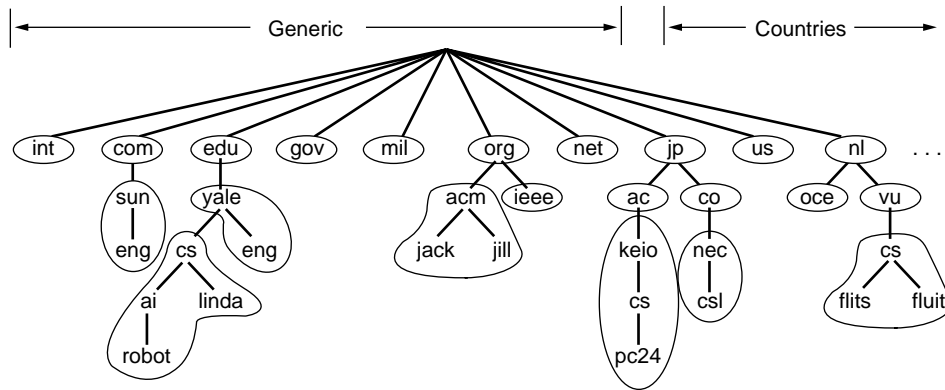
- Application programs which have a domain name (provided by the user) and want to find its mapping. It will call a “resolver function”, e.g., “`gethostbyname()`”. As we have noted before in the first tutorial, this can be done conveniently in Linux using the “`host`” command.
- The resolver is the client of the DNS, querying DNS servers to find the required mapping. The query might consist of many rounds, since the DNS server may refer the resolver to another DNS server. It has a configuration file (for a Unix system, it is usually in `/etc/host.conf` and `/etc/resolv.conf`), giving it the name of some DNS servers to query. It also has some configuration options, e.g., allow a local file (e.g., `/etc/hosts`) to take precedence over DNS queries. There are also other interfaces to make DNS queries: In C, one can call the resolver library directly (using `res_query`, etc.); and on command line, one can use `dig` or the more-or-less obsolete command `nslookup` to make direct DNS queries.
- The DNS servers, which hold partial mappings to return to resolvers. In Unix, the most (only?) popular DNS server is `bind` (Berkeley Internet Name Domain). DNS servers of most other platforms are based on it.

To the user, the DNS system is mappings from domain names to information. To the resolver, the DNS system consists of a set of many DNS servers. The picture is most interesting from the viewpoint of the DNS servers, which we will discuss next.

3. DNS and zones

The DNS system must support the name resolution for all computers in the whole world. This system won’t work if DNS is stored in only one database. Instead, individual DNS servers store only the subset of data for which they are responsible. The “division of labour” works by cutting the set of all possible names into **zones**: each DNS server needs only to cater for queries to a few zones. Zones are defined using the structure of domain names. E.g.,

¹When DNS is designed, it is envisioned that it is also used for storing information about individuals, not just hosts. However, that service is now taken up by another protocol, LDAP (Lightweight Directory Access Protocol).



In this picture, `keio.ac.jp`, `cs.keio.ac.jp` and `pc24.cs.keio.ac.jp` are all in the same zone, so their information are stored in the same DNS servers. All DNS servers know all top-level DNS servers which holds the top-level zone (i.e., the root zone). It also know the DNS server for all its **children zones**. When a DNS query is given by the user, the DNS server checks whether it is in the part of the domain name tree that it serves. If so, it replies the query by giving out the information it stores. We say the record is **authoritative**. If otherwise, the DNS server may **redirect** the client to the DNS server of another zone (usually, either the top-level zone or a child zone), so that the client knows where to continue the search.

Note that we have used the plural form when we say *their information are stored in the same DNS servers*. Information of each zone should be stored in multiple DNS servers within the zone. Such redundancy allows the system to continue working when one of the DNS server reboots. Of course, we need some way for the server to make sure the information are the same. The resource records for each zone served by a DNS server is stored in a **zone file**, with a standardized text format. They can be transferred between the DNS servers.

We have mentioned children zones. How to make one? Typically, when an organization wants a world-recognized domain name, it decides where to put the domain name (e.g., at `.com.hk`). Then it will ask the administrator of its zone to create a children (e.g., `.tmchan.com.hk`), typically for an annual charge. Once that is done, the organization must elect some IP addresses as its DNS servers, and tell the administrator of the parent zone (`.com.hk`). Then he takes charge of `.tmchan.com.hk`. He can make the whole tree under it a single zone, or he can create children zones in it, without telling the administrator of `.com.hk`. Such delegation of responsibility is what makes the Internet DNS system scalable to administer.

4. Caching and TTL

If every network connection queries the root zone servers, they will be saturated very quickly. Also, if every network connection requires several DNS queries, the system becomes exceedingly slow, and the network will be filled with many such queries. Given that many people will want information about the same domain names, it is usually beneficial to have a **cache** in the DNS server and the resolver, so that previously found results can be used as answers without going through the query process again.

Just like any other cache, it is more effective to implement the cache in the DNS server that everybody accesses, rather than for each individual in the resolver. So typically, DNS servers will provide a **recursive** service (the service we describe earlier is said to be **iterative**). With such service, it never refer the user to another DNS server. Instead, it will do all the required DNS queries by itself, and reply with the final answer. This allows the DNS server to put the answers

that it receives in its own cache, so that later queries are answered immediately. To boost the effectiveness of the cache even further, it is also possible to ask DNS servers to use **forwarders**, i.e., to forward every request to another DNS server so that the cache of the forwarded computer can also be used.

But there is a problem: **consistency**. If the resources record changes when the cache still hold old information, the old information is returned. This problem does not show up very frequently for most resources, since resources stored in DNS seldom changes. But there must be a way for the information to expire. Each DNS answer is tagged with a **TTL** (Time To Live), which tells for how long the information is good. The DNS server will only keep its cached copy for that long, after which it will go through the whole process again. Typically, for information that are relatively stable, TTL is set to 86400 (1 day); while for information that are more volatile, TTL is set much smaller, e.g., 300 (5 minutes). Cached answers are said to be **non-authoritative**, and is so marked in the reply.

5. Reverse lookup

DNS is a process taking domain names to information, usually IP addresses. Sometimes the reverse process is desired, taking IP addresses to domain names. This is done using the same system. But there is a problem: the system always use domain name as “lookup key”, so we must be able to **express the IP address as a domain name**. It is quite easy, and standardized: use the address `<reverse-of-IP-address>.in-addr.arpa`. E.g., to lookup `147.8.178.15`, one can use the domain name `15.178.8.147.in-addr.arpa.` as the key, and look for its PTR resource. This will give you an answer `postmaster.csis.hku.hk.`, the domain name of the computer holding that address. The `dig` program even has a flag `-x` to do it given an IP address.

As in normal address lookup, the information must be distributed and delegated for the system to scale. Every DNS server has B reverse zone. For example, our department has the reverse zones `176.8.147.in-addr.arpa`, `177.8.147.in-addr.arpa`, `178.8.147.in-addr.arpa` and `179.8.147.in-addr.arpa` in order to service reverse lookup requests. Each of them must **have its own zone file**.

6. Reference

The `host`, `dig` and `nslookup` commands are the primary interface to get low-level information about DNS queries. As usual, their documentation are in their own man pages. The C functions mentioned also have their man pages, in the `gethostbyname` and `resolver` man pages. The file formats of the configuration files of the resolver can be found in the man page `host.conf` and `resolv.conf`. One should refer to the `bind` manual when configuring a DNS server.

The protocol with which resolvers communicate with DNS servers are defined in RFC 1034 and 1035. The textbook also have a section (7.1) about DNS which you may find useful.

Appendix A. Configuration file of bind

The configuration file of bind consists of some global options (e.g., what directory to store master files), followed by specific options about how to serve each zone. For example, a simple configuration file might look like this:

```
options{
    directory "/var/named";
    pid-file "/var/run/named";
};
zone "." {
    type hint;
    file "named.ca";
};
zone "localhost" {
    type master;
    file "localhost.zone";
    notify no;
};
zone "0.0.127.in-addr.arpa" {
    type master;
    file "named.local";
    notify no;
}
zone "mynet.com" {
    type slave;
    file "mynet.zone"
    masters {
        192.168.1.1;
    };
};
```

There are quite a few types of zone, here we see a hint zone, a master zone and a slave zone. A hint zone defines a startup zone file for the server to find the root servers. A master zone defines the zone files for each domain that the server is authoritative. A slave zone defines a zone which is a replication of another DNS, i.e., its masters. In all cases, the file option is the location where the zone file is stored in the filesystem. The notify option of a master zone allows a master to choose whether to notify its slaves when the zone file is modified. Some other options that relates with our previous discussions include:

- `forwarders` (global/zone): A list of DNS servers to act as forwarders, so that all queries that cannot be locally satisfied are forwarded to it. Its value is a list of IP addresses, like the `masters` option above.
- `forward` (global/zone): What to do if the forwarders fails. If the option has the value `only`, it will return fail. If it has the value `first`, it will try to resolve the name on its own.
- `notify` (global/zone): Whether to tell slaves when a zone is updated. The slaves are defined as the set of all hosts referred to in the zone file, by a NS resource record.

Appendix B. Zone file format

A zone file contains the actual data stored in the DNS server. Here is a simple example:

```
$TTL      86400
$ORIGIN localhost.
@          1D IN SOA      @ root (
                                42          ; serial (d. adams)
                                3H          ; refresh
                                15M         ; retry
                                1W          ; expiry
                                1D )        ; minimum

                                1D IN NS     @
@          1D IN A       127.0.0.1
```

The first line specifies the default TTL for the resources records in this zone file. Then it specifies an ORIGIN for appending to relative domain names.

After that it is a list of resources records. The SOA record always present in all zone files. It begins with the @ symbol, which is a shorthand for the domain name of the zone as specified in the configuration file, immediately after the keyword “zone”. It also has its own TTL (1D, i.e., 1 day), and a symbol IN specifying the class. Both can be omitted. The next symbol is the @ symbol again, this time specifying where is the top of the zone. Then comes a relative name root, which expands to root.localhost. This is interpreted as an E-mail address root@localhost, specifying the administrator of the domain. Finally is the self-describing list of operating parameters. One particularly important argument is the serial number: whenever it is updated, the slave may be notified to update their entries.

After the SOA resource, there are two lines. The first line, with the resource record NS, specifies a nameserver for the zone. Note that, unlike the SOA record, it begins with no domain name. It may well be modified to a line similar to the third line. In general, when the first domain name part is omitted, it takes the value of the last resource record.

The last line is an A record, which specifies an address. The same IP address should not be given to two or more records as its A record. If an alias is needed, one should use the CNAME record, in the following form:

```
localhost-alias.  CNAME  localhost.
```

This specifies that localhost-alias. is an alias of localhost..

The nameserver line above is not very revealing, since both sides are the same. A more interesting nameserver line looks like this:

```
mysubdomain.domain.  NS  host.mysubdomain.domain.
```

It specifies that host.mysubdomain.domain. is the DNS server for the sub-domain mysubdomain.domain. If the line appears in a zone file outside mysubdomain.domain., it is usually necessary to add an A record for host.mysubdomain.domain., or else it will be impossible to contact the nameserver (since we don't know its IP address). Such an A record is usually called a glue record: logically it does not belong to the zone, but it is necessary to add it anyway to glue up with other DNS servers.