The Internet Domain Name System Explained for Non-Experts

Internet Society Member Briefing #16
By Daniel Karrenberg

The Internet Domain Name System Explained for Non-Experts

Dear non-Experts,
This is for you, who always wanted or needed to know how Internet names really work. The Internet Domain Name System (DNS) is a fascinating technology; almost all Internet applications make use of it. After reading this you will not suddenly have become a DNS expert, but you will have an understanding of how the DNS works and you should be able to distinguish a real DNS expert from a pretender.

Dear Experts,
This is not for you. For the sake of explaining principles we will often generalise and do this sometimes to the point where our explanations are not strictly correct in every last detail. The relevant Internet standards are where you find the details.

Purpose of the DNS

The purpose of the DNS is to enable Internet applications and their users to *name* things that have to have a globally unique name. The obvious benefit is easily memorizable names for things like web pages and mailboxes, rather than long numbers or codes. Less obvious but equally important is the separation of the name of something from its location. Things can move to a totally different location in the network fully transparently, without changing their name. www.isoc.org can be on a computer in Virginia today and on another computer in Geneva tomorrow without anyone noticing.

In order to achieve this separation, names must be translated into other identifiers which the applications use to communicate via the appropriate Internet protocols. Let's look at what happens when you send a mail message to me at daniel.karrenberg@ripe.net. A mail server trying to deliver the message has to find out where mail for mailboxes at 'ripe.net' has to be sent. This is when the DNS comes into play. The mail server transmits this question, called a 'query' in DNS terminology to the DNS. Quickly it receives as answer:

```
ripe.net. 1800 IN MX 100 postman.ripe.net.
ripe.net. 1800 IN MX 150 postboy.ripe.net.
```

“Mail for ‘ripe.net’ should be sent first to the computer called ‘postman.ripe.net’. If ‘postman.ripe.net’ is not available, try ‘postboy.ripe.net’”.

The mail server will now make a connection with ‘postman.ripe.net’. In order to do this it needs to know the numeric Internet address for ‘postman.ripe.net’. This numeric address is sufficient to send Internet packets directly to the computer called ‘postman.ripe.net’ from anywhere in the Internet. Again the DNS is queried and it returns ‘193.0.0.199’.

postman.ripe.net. 172800 IN A 193.0.0.199

From here on, the mail server delivering your message can directly communicate with the mail server that receives and stores my mail for me to collect later on.

It is obvious that the information we just obtained in our example cannot come from a central list somewhere. This list would simply be too large, change too often and its repository would be quickly overwhelmed with queries.

So how *does* this work?

Let us follow the DNS query starting from your computer. Your computer knows the address of a nearby DNS “caching server” and will send the query there. These caching servers are usually operated by the people that provide Internet connectivity to you. This can be your Internet Service Provider (ISP) in a residential setting or your corporate IT department in an office setting. Your computer may learn the address of the available caching servers automatically when connecting to the network or have it statically configured by your network administrator.

When the query arrives at the caching server there is a good chance that this server knows the answer already because it has remembered it, “cached” in DNS terminology, from a previous transaction. So if someone using the same caching server has sent mail to someone at ‘ripe.net’ recently, all the information that is needed will already be available and all the caching server has to do is to send the cached answers to your computer. You can see how caching speeds up responses to queries for popular names considerably. Another important effect of caching is to reduce the load on the DNS as a whole, because many queries do not go beyond the caching servers.

Detail - Caching hierarchies

Caching servers can also be arranged in a hierarchy. This makes sense in cases where the network capacity is limited and/or network latency between the DNS client and the rest of the Internet is high. When connecting a laptop to the Internet through a slow dial-up connection it makes sense to run a caching server right on the laptop. This way each click on a hyperlink on the same web-site will not cause DNS related traffic over the dial-up link. Such a local caching server is often configured to send all queries for which it does not have cached answers to the ISPs caching server in turn.

Sometimes corporate networks have local caching servers that in turn send queries to a corporate caching server before they are sent out to the Internet. This way the corporate caching server can build a large cache based on queries from the whole enterprise.

Beyond the Cache

If the caching server does not find the answer to a query in its cache, it has to find another DNS server that does have the answer. In our example it will look for a server that has answers for all names that end in 'ripe.net'. In DNS terminology such a server is said to be "authoritative" for the "domain" 'ripe.net'.

In many cases our caching server already knows the address of the authoritative server for 'ripe.net'. If someone using the same caching server has recently surfed to 'www.ripe.net', the caching server needed to find the authoritative server for 'ripe.net' at that time and, being a *caching* server, naturally it cached the address of the authoritative server.

So the caching server will send the query about the mail servers for 'ripe.net' to the authoritative server for 'ripe.net', receive an answer, send that answer through to your computer and cache the answer as well.

Note that so far *only* your caching server and the authoritative server for 'ripe.net' have been involved in answering this query.

Redundancy - Many servers with the same answers

Now, what if either the local caching server or the authoritative server for 'ripe.net' are unreachable? Obviously your mail server would not get an answer and could not immediately deliver your message. It would simply queue the message and try later. However if your web browser needed to find the address of 'www.ripe.net' when you clicked on a hyperlink to that site, things are different. The browser would appear to be stuck while waiting for the name 'www.ripe.net' to be resolved to an numeric address, because it would not be able to contact the web server without knowing that numeric address. Either you would tire of waiting and do something else, or your browser would give you an error message after timing out the DNS query. This does not happen very often because there is a lot of potential for redundancy in the internet architecture, especially in the DNS.

To ensure high availability the DNS has multiple servers all with the same data. To get around the problem of the *local caching server* not being available your computer usually has a number of them configured from which it can choose. This way one can make sure that there always is a caching server available. But how about the authoritative servers?

To improve availability of *authoritative name servers* there always are a number of them for each domain. In our example of 'ripe.net' there are five of them, three of which are in Europe, one in North America and one in Australia.

```
ripe.net. 172800 IN NS ns.ripe.net.
ripe.net. 172800 IN NS ns2.nic.fr.
ripe.net. 172800 IN NS sunic.sunet.se.
ripe.net. 172800 IN NS auth03.ns.uu.net.
ripe.net. 172800 IN NS munnari.OZ.AU.
```

Our caching server does not get one place to look up authoritative information about 'ripe.net' names, but five of them, all being equally authoritative. It is entirely free to choose any one of these based on its own preference as they all hold the same information; the choice could either be made entirely randomly or based on previous response times of those servers. The important point to remember is that *any* choice is a valid one.

This mechanism provides the potential for pretty strong redundancy covering both server and connectivity problems since the servers can be located anywhere on the Internet. Of course the level of redundancy actually achieved depends on the design and placement of the servers. A rather large corporation discovered this a few years back when all of their two authoritative name servers located in the same computer room became unreachable from the Internet because of a network problem that “could not happen”.

Entering Information Into the DNS

Information is entered into the DNS by changing the databases which the authoritative name servers use to answer queries. How these databases are maintained and transmitted to the servers is a local implementation matter fully transparent to DNS clients and outside the scope of this description.

For smaller domains often a local text file is changed on one of the authoritative servers and then transferred to the others. In this case the authoritative server on which the file is maintained is called the “master” server and the other authoritative servers are called “slave” servers. However this distinction is invisible to the rest of the DNS since all of the servers are equally authoritative.

Rising in the Hierarchy

Now let us consider the situation when our caching server has just been started and has an empty cache. Consequently it neither knows the answer to your query nor does it know where the authoritative servers for ‘ripe.net’ are. However it *does* know that it is possible to ask questions for ‘ripe.net’ to an authoritative server for ‘net’ because dots separate different parts of a domain name. This knowledge is part of the DNS protocol itself which says: “In case authoritative servers for a name are not known, strip off the leftmost part of the name including the first dot and send the original query to an authoritative server for that name”.

In our example an authoritative server for ‘net’ does not know the answer to a query about ‘something.ripe.net’, because the ‘ripe.net’ servers hold that information, but it *does* know which servers are authoritative for ‘ripe.net’ queries. So instead of an answer to the query, the ‘net’ server will answer with the list of authoritative servers for ‘ripe.net’, a referral in DNS terminology. The caching server will receive the referral to the ‘ripe.net’ servers. Just as in the earlier example it now asks its question to an authoritative servers for ‘ripe.net’, obtain the answer and send it back to the client that asked for it. In addition, being a caching server, it will cache both the answer and the list of authoritative servers for ‘ripe.net’ for further use. Simple enough.

But hold on, we assumed the cache was empty in the first place, so how does our caching server know where the authoritative servers for ‘net’ are? In other words what happens once we have stripped off all parts of a domain name and still do not know where to go for an answer?

For this case there is a special set of authoritative servers, the DNS root servers. They know the addresses of all authoritative servers for names that do not have a dot in them, the Top Level Domains (TLDs) such as ‘org’, ‘com’, ‘ch’, ‘uk’.

Root servers are the only DNS servers that have to be found without any other information being cached. To solve this bootstrapping problem all caching servers have a pre-configured list of

numeric addresses for all root servers. When starting up, a caching server will send queries for the *current* list of root servers to each of these addresses in turn until it obtains an answer. Once it has obtained the current list, it knows where to send queries for names without dots. This may sound needlessly complicated but it ensures that a caching server will keep working when the addresses of root name servers change but its pre-configured list is not updated; these addresses change infrequently but they do change from time to time. A caching server will be able to use all current root servers as long as it can reach just one of the pre-configured addresses.

Domain Name

The dots divide domain names into different parts. This division in turn divides the set of all names into different *domains* where the same part of a name can be re-used, e.g. 'ripe.net' is entirely different from 'ripe.org'. More importantly those domains can be administered totally independently. The administrator of 'net' names does not need to know anything about the 'org' names. The 'org' domain is independent of the 'net' domain. Due to the hierarchical structure administrators of sub-domains within a domain can also act independently, e.g. "isoc.org" does not need to coordinate anything with "nanog.org" or even know about it.

One Last Example

So once more, here is what happens when a caching server that just started receives a query for the address of www.isoc.org. After it started, the server obtained a list of root servers and their addresses. When the query arrives it will not find the answer for 'www.isoc.org' in the cache, nor will it find the address of an authoritative server for 'isoc.org', neither the address of an authoritative server for 'org'. Having no other choice it will then ask a root server for the address of 'www.isoc.org'; the root server will answer with a referral containing the list of all authoritative servers for 'org'. Our caching server will send its query for 'www.isoc.org' to one of them and get another referral with the list of all authoritative servers for 'isoc.org'. When sending the query to one of them it will get the answer. All this typically happens in less than a second.

From here on the caching server can answer the same query again and again from the cache without asking another server. It can also send any query for 'something.isoc.org' directly to an 'isoc.org' server and send any question for another name ending in '.org' directly to a servers authoritative for 'org'. Only when the next query ends in something different from '.org' does it have to ask a root server again. Quickly the cache will contain lists of authoritative servers for all popular domains, especially for all popular TLDs; usually our caching server will not have to query for this information again for several days. This design ensures that only a tiny fraction of all queries will have to be processed by the root servers or by authoritative servers for TLDs.

Managing Change

What about the case when information at authoritative servers is updated but caching servers all over the world have still cached the old possibly incorrect information?

This is where the DNS 'Time To Live' (TTL) comes into play. Each part of DNS information that may be cached separately has a time to live associated with it. Once this time expires, the cached information must be discarded and has to be obtained from an authoritative server again if it is needed. The TTL is not configured locally in the caching server but is set in the authoritative server and passed along with the information itself. This way the administrator of a domain, can control how long it takes for any change to be known throughout the Internet. By carefully choosing values, one can set the trade off between being able to change things rapidly on the one hand and minimising computing and networking resources needed by name servers on the other hand. If particular information is not expected to change in the near future one can have a high TTL and information known to change soon can be transmitted with a low TTL. It is standard practice to reduce the TTL transmitted with information that is scheduled to change in order to make that change visible rapidly throughout the Internet; once the change has happened the TTL is increased again.

This ends our tour of the DNS for non-experts. We hope that you have enjoyed it and gained a basic understanding of one of the underpinnings of todays Internet. Maybe you are now interested in some more explanations for informed non-experts about related subjects. We are currently preparing a similar briefing on how the DNS root name server system is currently evolving.

Expanded Coverage from the Internet Society

In-depth articles, papers, links and other resources on a variety of topics are available from the ISOC site at: www.isoc.org/internet/issues

Relevant IETF RFCs

The basic RFCs describing the dns are:

- 1034 Domain names - concepts and facilities
- 1035 Domain names - implementation and specification.

They have been updated and extended by numerous other RFCs including:
RFC1101,
RFC1183,
RFC1348,
RFC1876,
RFC1982,
RFC1995,
RFC1996,
RFC2065,
RFC2136,
RFC2137,
RFC2181,
RFC2308,
RFC2535,
RFC2845, and
RFC3425.

About the Author

Daniel Karrenberg currently serves the RIPE NCC as Chief Scientist. His interests include Internet measurements, the development of the DNS and the evoloution of what others often call “Internet Governance”.

In the 1990s Daniel led the establishment of the RIPE NCC, the first of the Regional Internet Registries. He has helped to shape Internet address space distribution policy, transferring both policy development and implementation to the region. He also brought the second DNS root name server to Europe in 1997.

In the 1980s Daniel helped to build EUnet and led the effort to transition it to Internet protocols, making EUnet the first pan-European ISP and bringing Internet connections to many places in and around Europe.

Internet Society
Galerie Jean-Malbuisson 15
CH-1204 Geneva, Switzerland
Tel: +41 22 807 1444
Fax: +41 22 807 1445
http://www.isoc.org

1775 Wiehle Ave. Suite 201
Reston, VA 20190, USA
Tel: +1 703 439 2120
Fax: +1 703 326 9881
Email: info@isoc.org

Acknowledgments

The ISOC Member Briefing series is made possible through the generous assistance of ISOC’s Platinum Program Sponsors: Afilias, APNIC, ARIN, Microsoft, Qualys, Ripe NCC, and SIDA.

More information on the Platinum Sponsorship Program:
http://www isoc org/isoc/
http://www.isoc.org/isoc/membership/platinum.shtml