Stephen L. White
6.033 | Prof. Katrina LaCurts
R04 | Feb 24 2017

Critique of *Case Study: The Domain Name System*

Note: all references to 'the reading' refer to section 4.4.x of *Principles of System Design, Case Study: The Domain Name System (DNS)* from the course syllabus, beginning on page 175.

The internet's Domain Name System, or 'DNS' is one of the countless examples of complex system which people use everyday without even realizing it. Think about the telephone system. Wouldn't it be vastly more intuitive if you could dial "dominos" instead of having to remember a 9 digit number? And, what about if their number changes? Is there a better way? Well, think about the internet. Isn't it much more intuitive that you can just go to 'dominos.com' instead of having to remember `205.218.22.49` every time you want to order a pizza? This is the functionality that DNS offers. To imagine its implementation, you could first naively imagine that each computer has a sort of phone book which maps names to numbers. But what if someone changes their number, or there is a new one added? As it wouldn't be practical to update a list on every computer on the internet, DNS took a different *path* for their naming system.

On top of a the much more human interface of using words instead of numbers, DNS also tackles the problem of the updating records by using a *distributed directory service model*. This distributed model distributes the responsibility of maintaining up-to-date records to many different servers. When one of these servers is asked to resolve a name, it first looks through a list of names that are in its jurisdiction, so to speak. If the name is there, it resolves it, but if it is not, it begins to look through a listing of referrals to other servers. These servers each hold records for different regions of the DNS namespace. It is important to note that this lookup/referral process can be recursive, and can regularly go through a handful of iterations before resolving. This distributed and hierarchical naming system promised a fast responding, robust, and scalable system, all in the name of a better human interface.

Perhaps the main design consideration which had the greatest effect on the overall design principles has to do with its scalability. The idea of getting every computer system on the planet to use the same naming system may have seemed difficult at the time, and still does. The best way to achieve global scalability was to enforce modularity in the system using distributed hierarchical records. Keeping records on each and every machine could simply never scale, plus distribution of the records allowed for decentralization.

Going hand-in-hand with scalability is the responsivity of its service. After all, if it would be faster to memorize a few IP addresses, what is the point of a DNS? One area of friction is the

recursive structure of the system; some lookups could take many iterations. So, DNS introduced a few of optimizations to increase its response time. From the description in section 4.4.1 of the reading, page 180, DNS requires each name server to keep a cache of names that it has 'heard' from other servers, thus eliminating bad performance from repetitive and recursive calls. Although, caching like this creates duplicate copies of data on many machines. But wait, how would these duplicates be updated efficiently? Well, this is one of the downfalls to the increased responsivity. Rather than absolute correctness, DNS has a plan of "eventual consistency" meaning that each cached record has a short expiration date at which the record is deleted.

Finally, like any great system, DNS needed to be resilient. As users began integrating DNS resolvable terms instead of IP addresses into their workflows, any sort of outage could deny services. Since it was extremely important that the DNS be fault tolerant and robust on a global scale, their services needed to be distributed as aforementioned. DNS also lets *any* nameserver attempt to resolve a call rather than only the root. If a nearby nameserver heard the request and knew the resolve or had it in the cache, it can resolve it without the request needed to bubble all the way up to the root. Singular DNS records would not be dependent on any one server to resolve all requests. Even more pedantic elaborations were implemented to increase availability of the DNS service. For instance, it is discussed in section 4.4.3 of the reading that DNS requires each name service keep 2 identical replica servers. The loss of a single server is better understood when the resolution is thought of as a recursive tree; requests at the bottom keep being referred to parents as they move up the tree before resolving at some root server. The loss of a server higher in the tree, could block out all requests from nodes of which it was an ancestor.

In summary, DNS manages to provide responsive, robust, and highly scalable name services through its simplistic and pedantic configuration. It manages to keep its interface as a simple as its functionality: resolving a name. This interface is so seamless that most users do not even realize that they are using it. DNS's pedantic and forward-thinking implementers used a widely distributed system to decentralize naming, increase fault tolerance, and increase responsivity on a global scale. Along with forced backups, DNS increases scalability and fault tolerance with allowing all nameservers to cache records and resolve names without needing the root server. While this greatly increases performance, it also slightly weakens the specifications by using eventual consistency by allowing cached records to be immutable but have an expiration date. DNS's design was well thought out, and allow it to continue to play an invaluable role in the grow of the internet, some thirty years its introduction.