

A Distributed Multicast DNS System:  
Research Paper

Daniel Johnson

June 15, 2010

# 1 Abstract

There exists a need for less dependance on the single point of failure that is DNS servers. It is quite possible for a computer to be completely connected to the internet, but run slowly or not at all in the case of a malfunctioning DNS server. My goal was to allow a group of workstations as present on a medium-size subnet to survive complete loss of a DNS server through collaboration. The syslab workstations provide an example of an appropriate size.

# 2 Background

Several efforts have been made to solve similar problems:

- The Avahi project provides name lookups for machines on the local subnet vial a .local pseudo-TLD
- DistributedDNS attempts to surpass the traditional ICANN-based name service, which is too ambitious to succeed.

My solution with work with the local link only, which will keep speed as fast as or faster than traditional nameservers, provide lookups for all hosts, not just nearby ones, and honor the authority of the root nameservers.

Another important algorithm may be DHT: Distributed Hash Tables. Since DNS records are key-value pairs, this may be perfect. It is important to note that DHT is primarily used when larger data is involved. Maintaining a cache for smaller data might have greater problems with volatility and overhead.

# 3 Development

## 3.1 Computer Language/Software

Python is being used for prototyping and network simulation.

## 3.2 Procedure

I will be using python for the initial proof-of-concept simulation and to make sure the protocol will allow for all necessary features. After the simulated

nodes can function properly, I will translate the protocol and implementation into C++, a better language for lower-level operating system functions.

The next step is to implement a NSS (Name Service Switch) module in order to allow a native linux/UNIX system to take advantage of these features. An alternative to NSS might be an intermediate nameserver, which would look like a regular nameserver to the host it was on, but would really just be a client in my system. This is similar to current caching nameservers like dnsmasq.

### **3.3 Testing/Analysis**

The simulation will test the network chatter of the protocol and the redundancy. By systematically removing nodes and wiping caches, it will be able to test how resilient the network is to changing topography.

### **3.4 Progress**

So far this project is progressing at an acceptable rate, but there have been a few difficulties and challenges along the way. First, the conversion from simulation to actual network program had some difficulties. Sending multicast messages turned out to be rather easy, but sending unicast messages (to one host at a time) has proved harder. As it turns out, regular TCP sockets are a management nightmare here, as we'd have to have one open for each and every client pair, which is way too much. After about a week of failed attempts here, I realized that, in fact, UDP would be much more efficient, as UDP listeners only have to listen on the proper port, not bother with many attempts at establishing a connection.

Another problem I had is that the simulation had each node having a unique name/identity. In the real world, these hosts have a hostname and multiple IP addresses, and when another host receives a message from you, it has to do work to recognize who sent the message, to associate them with an entry in the hostlist. Also when a message is sent referring to a host, it can be difficult to determine if the current host is the one being referred to, since the sense of identity is much weaker with real networking.

More recently, I have implemented a multi-threaded server/client, which gets the commands on one thread (from the console), and uses a locked list to communicate those requests to the other thread that is in communication with the other nodes on the network. What this does is create a

client/server model in one program: the client IS the server, it just needs to be multithreaded to do both functions at the same time.

A basic logging framework has also been developed, which is designed with shared filesystems in mind, in order to programmatically keep track of the nodes all over the network.

## 4 Results

The benefits of offloading routine and emergency duties from the nameserver has several practical benefits. First, in the event of a nameserver outage, not all systems need to fail. While non-cached entries may not be available, those that have seen high use (google.com, for example) will still be available. This helps to eliminate one instance of a single point of failure. With a sufficient number of hosts, processing queries on the main nameserver can lead to performance issues. By dividing responsibility for name lookups among hosts, the speed and scalability of lookups can be improved.

Hopefully as this concept is implemented it will be good enough to put into production in UNIX computer labs around the world. With enough effort and review, it should be possible to gain acceptance into the community, assuming the security requirements and social requirements are met.

Currently, everything is working as expected. The hybrid client-servers connect to each other, retain only the information they are interested in, and communicate these things clearly enough to help each other out. The last thing that remains is integrating with normal programs, such as firefox.

### 4.1 Next Steps

Two things need work currently, integration with system applications and a better method of visualizing the information from the hosts. Currently, the only way to look at the output of a handful of nodes is by having them all open in different tabs of a terminal, which makes viewing more than a couple of nodes worth of output difficult. A graphical view will make testing and debugging easier, as well as provide a better visualization method for showing the technology.

It is my hope that further work can be done on this to extend my proof of concept into something that can be deployed to labs in real schools and businesses.