GeoDNS—Geographically-aware, protocol-agnostic load balancing at the DNS level

John Hawley
Linux Foundation / Kernel.org
warthog9@eaglescrag.net

Abstract

The Open Source community has grown from a series of small projects hosted from anywhere that was convenient, to a globally distributed set of mirrors providing content to every region of the planet. While there has been an outpouring of support in providing mirrors to every possible facet of the Open Source community by independent entities, a problem has arisen in how to efficiently load-balance across this global infrastructure, not only for the benefit of the mirrors, but for the users as well. There are a multitude of solutions currently available to try and cope with the issues of load-balancing including physical load-balancers like squid, protocol-specific redirection like mod_geoip in Apache, and full-scale commercial content distribution like Akamai. For most situations, the commercial solutions are far outside of reach and may necessitate the removal of existing infrastructure. Things like squid require, for practicality, all of the machines to be in a single location or for a single location to provide the sum total of the bandwidth available. Lastly, the protocol-specific solutions like mod_geoip work well for their own protocol, but leave other services like rsync, ftp, git, and svn to fend for themselves—assuming that the protocol even supports redirection. Most do not.

GeoDNS is the idea of taking an incoming DNS request, doing the geographic look-up at the request time, and returning different results based on the incoming IP address. This particular approach, taken by several DNS servers including bind-geodns, powerdns, and tinydns (with patches), allows geographically diverse mirroring infrastructures like `Kernel.org`, Wikipedia, and many other sites to direct users seamlessly to an appropriate server. This helps distribute the system loads across the entirety of their mirroring infrastructure. This protocol-agnostic approach is more universal and simpler for end users to handle by making seemingly hard choices transparent to them.

1 Internet—20th Century tech solving 21st Century problems

The Internet has existed for four decades. It has evolved, grown, changed, and adapted from its humble beginnings as Arpanet to what it looks like today, with TCP/IP, IPv4, and IPv6. Though it has changed radically from where it started, the Internet's fundamental building blocks, TCP/IP and DNS, have changed very little and they continue to serve the Internet well. Though while they are the bedrock of the Internet, and are serving the needs of billions, small changes can be made to them to greatly extend their usefulness and continue to serve the Internet for several more decades.

1.1 International Growth

The Internet started as a small research project out of the Advanced Research Projects Agency (ARPA) of the United States Department of Defense. It initially spanned two nodes on the California Seaboard, and has grown to include on the order of 1.596 billion users[1] worldwide. The United States boasted, for a number of years after the inception of the Internet, the largest amount of capacity to host sites and projects connected to the Internet. This meant that a great deal of the content of the Internet was accessed by going to a single point, likely in the US, where the servers would be the best connected and provide the most good to all.

However, this single-point-of-connectivity model was unsustainable with the amount of growth outside of the United States. This growth far outstripped the capacity of intercontinental links, including satellite relays and undersea cables, as well as having significant cost and performance considerations. A growing pressure externally from a growing Internet population drove the need

[1] `http://www.internetworldstats.com/stats.htm`

for content to move from a centralized single provider to a more distributed localized content distribution, with mirrors of data forming all over the world and content distribution companies such as Akamai[2] stepping up to help fill this demand.

1.2 Mirrors, Mirrors everywhere, but not a drop to drink

As with many solutions to a problem, you may solve what is immediately needed, but it in turn creates a new problem that needs to be solved. This new-found infrastructure for mirroring and content distribution provided one such situation. Originally the Internet was intended to provide a good means of dealing with content as it existed in its centralized locations. When you went in search of content based on a URL, it was expected that the look-up would only involve a single unique system, whether this was a individual server or a group of them within close proximity to each other, providing the content. However with both the explosion of the Internet and the subsequent need for mirrors to become more localized to the users seeking the data, the need for changing this philosophy became apparent.

A simple approach initially was, and predominantly still is, employed by providing a user with a listing of the mirrors available, usually grouped by the server's country of origin. While this works, it makes the user interfaces difficult and requires that the user make an educated guess as to which mirror is likely closest to them and will provide the fastest service. These decisions can be quite difficult; it has been found for many Canadian users that it is faster to go to a mirror based in the United States as opposed to a Canadian one due to the existing backbone and routing infrastructure present in much of Canada.

Additionally, load-balancers are available to help act as a director to content. Typically, however, load-balancers act as a man in the middle (so to speak), providing a single point of entry and exit for a cluster of machines. This cluster typically resides in a single geographic location, and the load-balancer itself is a limiting factor in how much content can be distributed from these machines, as the load-balancer acts as a bottleneck to the cluster. This works particularly well for single sites, but it does not work when the machines that need to be load-balanced are geographically disjoint, though this provides high-availability with its load-balancing.

To help both high-availability and physically disjoint systems, things like round-robining in DNS can be used. This helps—in particular this alleviates a need for a user to make an explicit choice in server; however, it has a downside in that it is highly dependent on the implementation of the DNS look-up engine at the client. This dependency, unfortunately, is known to have some serious flaws in certain implementations—in extreme cases, even going so far as to sort the list of returned IP addresses in numeric order and to always use the lowest numerical address, instead of randomly cycling the list. Despite these issues and dependencies, there are a number of implementations that do the correct procedure and this particular approach does indeed help with load distribution.

With the issues inherent in round-robin DNS and the lack of geographic diversity in normal load-balancers, an additional approach is protocol-specific extensions. Protocols like HTTP have the ability to respond to clients with a redirect, pointing the client to a new server to acquire the content it is seeking. While this works, each implementation is inherently tied to a specific protocol and requires both client- and server-side support. If the option of redirection is not already available in a protocol, adding it would be difficult and would leave many clients unable to take advantage of the new feature. So while HTTP supports this option of redirection, many other protocols do not. This includes ftp and rsync, which are both heavily used in content distribution.

None of these solutions solves the problem of geographically diverse load-distribution universally; at best, these solve the problem for a particular niche. A global distribution system made up of independent entities needs to be transparent to the end users, as it's difficult for them to make good decisions as they lack information needed to make them. It needs to be versatile, able to expand and contract, resilient to changes, and most of all be protocol-agnostic so that existing and any future protocols can be easily or trivially supported.

2 GeoDNS—Knight in Dingy Armor

GeoDNS is an attempt to solve the shortcomings inherent in the predominant and existing load-balancing

[2] `http://www.akami.com`

schemes. It targets this by making a small, server-side-only change to a DNS server to allow it to respond to requests in a slightly different way, depending on the origin of the DNS request. This particular approach solves many of the issues inherent in round-robin DNS, centralized load-balancers, and protocol-specific redirection; however, it comes with its own set of quirks and issues that are equally inherent in its implementation.

2.1 Basic ideas

GeoDNS itself is a rather innocuous change to the way DNS handles requests, but gives the basic ability to do wide-scale, simple load-balancing without the need for changes to clients or custom protocols. DNS is a fundamental building block of the Internet. Every client that is attached to this global network already has a the ability to make a DNS query, converting a textual string like `example.com` into a numeric address, `208.77.188.166`. GeoDNS, like DNS views, differs slightly from a normal DNS query in that the response is altered based on additional criteria.

In the case of DNS views, it checks the incoming IP address, looking for matches in a range, and returns a different address based on each defined range. This is typically done to deal with the proliferation of Network Address Translation (NAT), where the IP address externally may be a routable IP, but is inaccessible to internal NATed machines. The DNS server returns the routable IP externally, and a non-routable IP internally based on the view criteria.

Strictly speaking, returning different data based to any query, according to RFC, would be the result of an overlapping tree and thus a "non-fatal error." However, since a client is unlikely to ever get into a situation where it would get multiple incompatible responses to an individual query, this doesn't actually cause any unexpected behavior to the client.

It does, however, break the idea of transparency across the Internet, where everything on the Internet is globally viewable and a DNS query will return the same result set no matter where you are. But this is no more broken than the idea of NAT, which currently has a huge proliferation and is arguably the reason that the IPv4 address space has not yet been completely exhausted. However, while this lack of transparency and consistency is not ideal from a DNS perspective, it is an effective means of solving a distribution problem and should be used on an as-needed basis and not be considered a standard practice for all queries.

2.2 GeoDNS: DNS View with a twist

GeoDNS comes into play to solve the same fundamental problem that a DNS view was introduced for: to return a more local resource for usage. Though where a DNS view is more likely to be solving the problem of routable vs. non-routable IP addresses, GeoDNS is more targeted at giving a user a more appropriate resource based on physical location.

GeoDNS functions very similar to a DNS view, whether it's implemented as one or not. The requesting client's IP is checked against a database, which determines a general geographic location for the requesting IP. Using this geographic identifier, a response is generated that is specific for that location, and sent back to the requesting client.

2.3 So it seems to work, but who uses it?

GeoDNS has a limited set of interested parties, which is one of the reasons that GeoDNS has not become standard in most DNS servers. Most users have their machines in a single location, on a single subnet, and not scattered across the globe. There are, however, installations that are making use of GeoDNS successfully.

For instance, this paper is focused on `kernel.org` and its usage of GeoDNS and that it works for the specific setup that `kernel.org` has. `Kernel.org` makes use of a modified set of patches that originate from caraytech. This patch set has been updated a couple of times, with the last being in 2008 by `kernel.org`.[3] However, despite `kernel.org`'s popularity within the Linux community, it is a relatively small distribution system, with only 4 locations in three countries serving the worldwide populace. Larger installations of GeoDNS servers that affect far more users exist. In particular, the largest known user of GeoDNS is Wikipedia.

Wikipedia and Wikimedia, for instance, moved to using a combination of Bind and PowerDNS[4] for their DNS

[3] It should be noted that this patch series makes use of MaxMind's GeoCountry look-up database.

[4] PowerDNS first introduced its geobackend in 2004.

servers in 2005.[5] Bind handles the primary zones themselves, while PowerDNS with the geobackend powers the distribution of users to a local Wikipedia server. A more complete description and an implementation that mirrors the Wikimedia setup can be found on the Blitzed IRC (Internet Relay Chat) network.[6]

A similar system using pgeodns (perl geo DNS server) serves `cpan.org` and `perl.org`, and has done so since 2001. CPAN is the primary locus of the Perl module development community. The pgeodns server is specifically attached to `search.cpan.org` to give to give a user a transparent local mirror for searching the CPAN archive, thus distributing the search load across multiple machines worldwide.

While `kernel.org`, Wikimedia, and CPAN are making use of the GeoDNS servers to act as load balancers, there are companies such as Server4Sale[7] that use a modified tinydns server, and are using it as a protection mechanism for their network and clients. Specifically, Server4Sale has made it available under the name Veridns, with the specific intention of using it to help deal with the problems of Denial Of Service (DOS) and Distributed Denial of Service (DDOS) attacks.

There are a number of other, primarily open source communities using GeoDNS-based services, from Apache[8], to Mozilla[9,10] to solve the same fundamental mirroring and load-balancing problem that many worldwide distribution systems are facing: how to get the data that users are seeking to them using the fastest possible system, and to load-balance these users across the world.

2.4 Where's the catch?

The art of attaching a geographic location to an IP does not come without some downsides. The databases are guaranteed to be inaccurate, require constant updates due to a shifting reality, and currently they are anything but future-proof.

Each database—whether it's being maintained by a corporate entity like MaxMind, which has a vested interest in its accuracy, or being kept up by a small open source project—faces the same challenge: the Internet is constantly shifting, and no one is required to publish geographically accurate information. So the collective maintainers are forced to go and mine public sources like Regional Internet Registries like RIPE, ARIN, and APNIC for information, but at best that gets them a broad brush-stroke of information. It does not, however, get the more subtle differences or weirdness that are sometimes used on the Internet, such as a US-based company using a subset of its IP addresses at a European facility. To become more accurate, further mining based on more data is needed, but even the best available databases are only claiming 99.8% accuracy,[11] which means that out of a potential 4.2 billion addresses, roughly 8.5 million addresses are incorrect.

These databases, while already trying to play catch-up with the ever shifting sands of the Internet, are facing another daunting and more complicated task ahead of them: creating and maintaining an additional database mapping IPv6 addresses to geographic locations. Currently all of the available databases only support IPv4, the dominant address scheme for the Internet since its first proposal in RFC 791 in September of 1981. IPv4 is running out of usable address space. (Though it has been predicted repeatedly that it should be exhausted already, usage of NATed networks has kept IPv4 dominant.) It will eventually run out of address space, and the Internet will be forced to make the long, arduous, and painful move over to IPv6—and in the process, go from a possible 4.2 billion addresses to 340 undecillion addresses. Currently most users of IPv6 are going through various tunnels and gateways, which will mask an address's actual location. Couple that with a low penetration of IPv6, and there is no current incentive to solve this particular problem by the open source community, and especially not by a for-profit entity.

So while things like GeoIP are working in their current form, they are not a perfect solution: the databases are inherently inaccurate and they currently lack support for

[5] `http://en.wikipedia.org/wiki/PowerDNS#PowerDNS_and_Wikimedia`

[6] `http://blitzed.org/DNS_balancing`

[7] `http://pub.mud.ro/wiki/Geoipdns`

[8] `http://mail-archives.apache.org/mod_mbox/www-infrastructure-dev/200904.mbox/<4239a4320904011536x5726d0eraae7065967ac1b77@mail.gmail.com>`

[9] `http://blog.mozilla.org/mrz/2008/05/28/geo-dns-getting-the-bits-closer-to-you/`

[10] `http://blog.mozilla.org/mrz/2008/06/11/geodns-one-week-later/`

[11] MaxMind's commercially available GeoIP Country Database claims an accuracy of 99.8% – `http://www.maxmind.com/app/country`

the next generation of the Internet. Despite these problems, GeoIP is quite useful. Even a database that is 80% accurate will work for a majority of the world's population as they would expect.

3 What to expect from GeoDNS

GeoDNS on the whole is an incredibly powerful and useful tool for administrators and world wide distributors. It can transparently deal with simple distribution across the entire globe and provide a simple interface for users to work with and use.

3.1 Production Zone Implementation

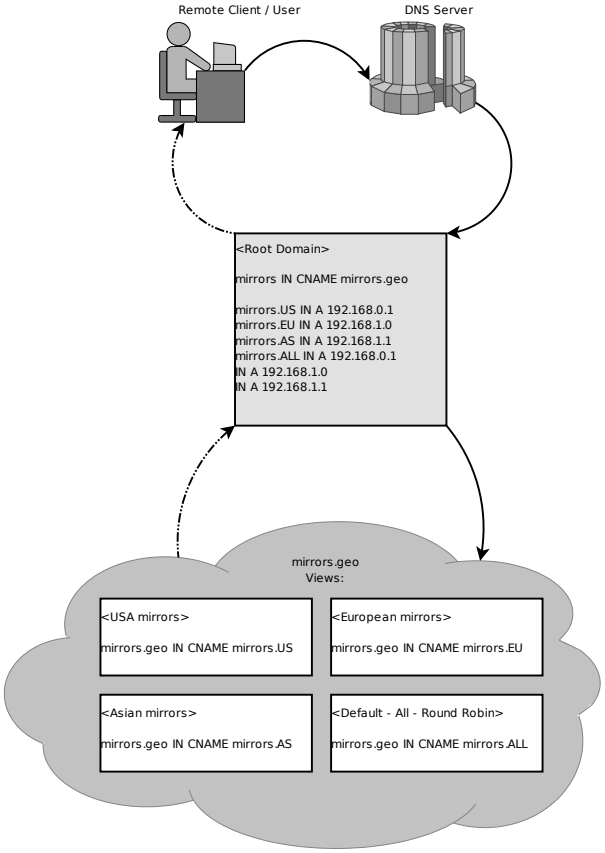


Figure 1: Suggested Flow request for a GeoDNS DNS request

GeoDNS is nothing more then a DNS answering system, and as such has a huge amount of flexibility and infrastructure that can take advantage of it. However, there are some implementation details that should be taken into account.

Since a GeoDNS-based server does not respond in the same manner as a normal DNS server, special care should be taken in choosing slaves. Slave DNS servers for your zone are going to need to be running the same GeoDNS server that your master is running, assuming that your GeoDNS server implementation supports replication at all. The best option would be to run both the master and the slaves. While this is a cumbersome proposition to some, it does solve the problem of compatibility and simplifies the fact that that zone is unexpected for most DNS servers.

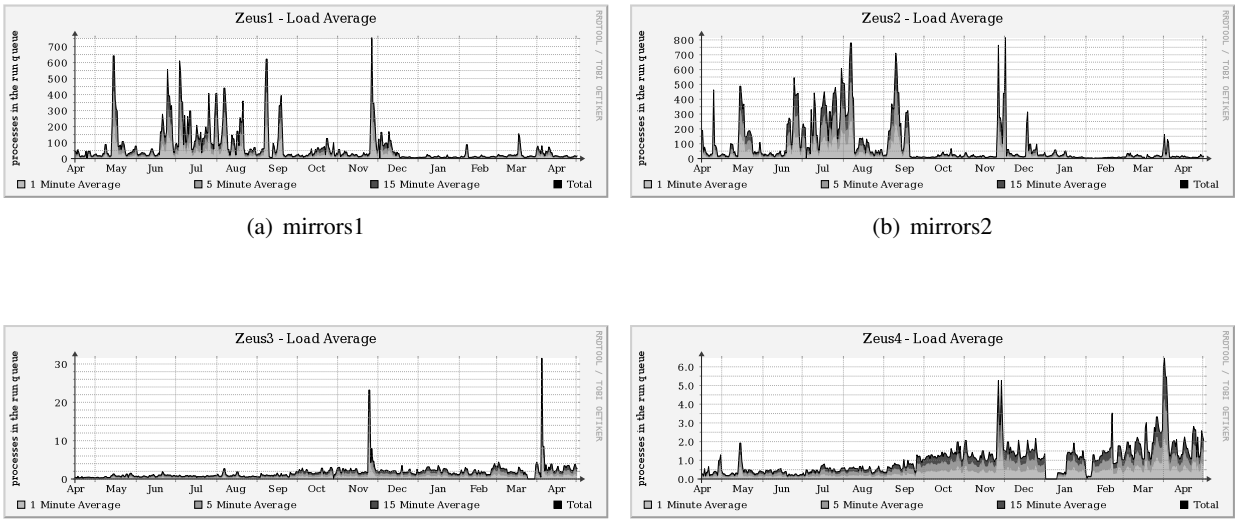
In the case of Bind + GeoIP, it has some simplifications that make it appealing. For instance it leverages the existing view infrastructure, giving it the full ability to replicate each view independently[12] and all the other advantages of views. PowerDNS with the geobackend unfortunately does not have the ability to act as the master or slave for DNS purposes,[13] however, the basic implementation of the zone structures are similar.

A basic and scalable configuration is two-fold. There will be two different servers, one authoritative for the actual IP addresses that you are using, and one that is authoritative for your GeoIP zone. While Bind can handle these both in a single instance, common pieces that would be available in all zones (like the non-geoip hosts) would need to be copied multiple times and maintained independently. This copy-and-paste structure can lead to errors and make it difficult to maintain. For convention, `example.com` will be the primary static zone with all the IP addresses in it, while `geo.example.com` will be the zone handling all of the geographic look-ups.

The request for a domain that will be served via GeoDNS should come in to a common disambiguation point, something like `mirrors.example.com`, which should be present in the primary static zone. This would be a CNAME that that would point to `mirrors.geo.example.com`, which is in the geographically aware zone. When the DNS client requests the next hop for `mirrors.geo.example.com`, GeoIP look-up would occur on the server. It would match the incoming request's IP address and select the appropriate zone's view to return. The response, as all

[12] How Can I Make A Server A Slave For Both An Internal And An External View At The Same Time? When I Tried, Both Views On The Slave Were Transferred From The Same View On The Master. `http://www.isc.org/node/282`

[13] `http://doc.powerdns.com/geo.html`



(a) mirrors1

(b) mirrors2

(c) mirrors3

(d) mirrors4

Figure 2: Load graphs of `mirrors.kernel.org`

entries in the `geo.example.com` zone, should likely be CNAMEs that point back to the primary zone of `example.com`. This final CNAME look-up would point to an IP address, or a round robin of addresses to further add distribution. Figure 1 illustrates the basic request and response structure that this would entail.

Breaking the zones up into GeoIP vs. non-GeoIP zones has several advantages. It allows you to trivially migrate between GeoDNS solutions, as the zone handling those look-ups is independent. Master and slave relationships are not limited to other GeoDNS servers in the non-GeoIP address space, meaning you can use any combination of DNS servers. Breaking this relationship up also means that requests are only being performed for only those items that require it, like a mirroring infrastructure as opposed to the mail (MX) records for a domain.

3.2 GeoDNS: Case study—kernel.org

To give some basis for a real-world example, `kernel.org` is a worldwide distribution system with equipment in three countries and five separate data centers. Its content includes the primary download location for the Linux kernel source code, and it acts as a tier-1 mirror for many of the distributions based on Linux. It boasts a user community that spans the globe, with users accessing its content on every continent, including Antarctica. With such a large user base, and a historical issue of only having servers based in the United States, `kernel.org` was looking to simplify the problem of mirror distribution without causing undue confusion to its user base.

A simple solution was first implemented using a country- or region-specific domain. New equipment was brought online in Europe, the Netherlands, and Sweden respectively, so `mirrors.eu.kernel.org` and `www.eu.kernel.org` were set up. This solved the basic problem of getting user-recognizable connectivity to the European mirrors. However, because `mirrors.kernel.org` and `www.kernel.org` have been more or less ingrained in the mindsets of users, the machines did not see quite the pick-up in usage we had hoped for, and it was clear that users in Europe were still coming back to the US mirrors to get their content.

Several different solutions were considered to more transparently direct users to a closer mirror. `Kernel.org` had a simple set of requirements:

1. The need to deal with a geographically diverse set of machines. This generally rules out things like normal load balancers.

2. Be protocol agnostic, as `kernel.org` serves data over ftp, HTTP, rsync, and git. It could not be subject to protocol-specific load balancing, so this rules out things like mod_geoip for Apache.

3. Be transparent to end users. Many of `kernel.org`'s users are automated scripts and programs, so having a disambiguation page to direct users to the closest mirror is not a solution that would work.

A DNS server based on the BGP (Border Gateway Protocol) as its determination mechanism was considered. BGP is, however, notoriously difficult to gain access to with the need to acquire an autonomous system (AS) ID from ICANN. Coupled with the difficulty in getting an AS, many larger backbone providers are uninterested in peering and sharing their routing table information with smaller entities sometimes having requirements to control upwards of seven hundred unique and routable IP addresses. Once peered, the problem becomes one of completeness of the routing table provided, with many instances of partial views of the entire routing table which would be difficult to make good decisions based on. This particular approach was abandoned for these reasons.

A GeoIP approach was then investigated, with specific emphasis on GeoIP coupled to a DNS server. It was decided to work with the Bind-based patches and to go with an organization similar to what is described in Section 3.1. The entire setup was flipped over on September 19th, 2008. Figure 2 shows the loads of the `kernel.org` machines before and after the switch to using GeoDNS. Prior to the middle of September, the two US machines, `mirrors1` and `mirrors2`, had a high and periodic load with long and consistently high loads. This is particularly evident in the summer months of 2008, with loads consistently above 50 and easily peaking above 200 in instances. It is also clear that during this same timeframe, the loads on `mirrors3` and `mirrors4`, the European machines, was virtually nonexistent. The loads were barely even high enough to register on the graphs. After September 19th, the graphs take on a much different outlook, and in fact the change was noticeable within hours of the DNS change.

Saying that the GeoDNS server's activation was a success would be an understatement. After the September 19th switchover, the graphs for the two US-based machines drop dramatically. Over the course of 7 months, they've maintained a relatively consistent load well below 100, with occasional spikes above 100. The loads on the European based machines have risen as a result of the higher traffic. Though the change is a noticeable, the machines are still coping with the new load without issue. So, in this specific circumstance, GeoDNS has been an unequivocal success and has helped to extend and better the services that `kernel.org` offers.

Proceedings of the Linux Symposium

July 13th–17th, 2009
Montreal, Quebec
Canada

Conference Organizers

Andrew J. Hutton, *Steamballoon, Inc., Linux Symposium, Thin Lines Mountaineering*

Programme Committee

Andrew J. Hutton, *Steamballoon, Inc., Linux Symposium, Thin Lines Mountaineering*
James Bottomley, *Novell*
Bdale Garbee, *HP*
Dave Jones, *Red Hat*
Dirk Hohndel, *Intel*
Gerrit Huizenga, *IBM*
Alasdair Kergon, *Red Hat*
Matthew Wilson, *rPath*

Proceedings Committee

Robyn Bergeron
Chris Dukes, *workfrog.com*
Jonas Fonseca
John 'Warthog9' Hawley

With thanks to
John W. Lockhart, *Red Hat*