pDNS: a secure, private, decentralized P2P DNS

Kevin Baichoo, Tushar Dhoot, Vishal Ranjan
Stanford University

Abstract

We rethink what DNS can be, presenting pDNS, a P2P DNS. pDNS enables users to remap their view of the internet, explicitly trusting peers for particular mappings. pDNS can be used to prevent censorship and to share private mappings. Unlike DNS which adopted security as an afterthought, pDNS is built with security and privacy in mind, with features such as TLS and mutual TLS.

1 Introduction

The traditional DNS architecture was designed in the 1980s. While the current system has come a long way, we believe that there have been some growing pains. Some of the design decisions made in that context might no longer be applicable. One example is the existence of Root NameServers that hold a small 2MB file [5] and add delay to DNS resolution.

Furthermore, the traditional system is centralized, making it an easy target for censorship, and in some sense clashing with the decentralized nature of the internet. We re-imagine DNS in as a network of peer nodes, allowing users to remap domain names to different IP addresses either through direct mappings that nodes know themselves, or indirect mappings via trusted peers. Our system would be resistant to censorship as nodes are easy to spin up and users can easily swap out peers. In effect, it would be a difficult game of 'whack-a-mole'.

In section 2 of the paper, we'll discuss the backdrop to which pDNS was designed. In section 3, we present a high level system overview of pDNS. In section 4, we dig into the design details of our system. In section 5, we evaluate the requirements and performance of pDNS and propose future work.

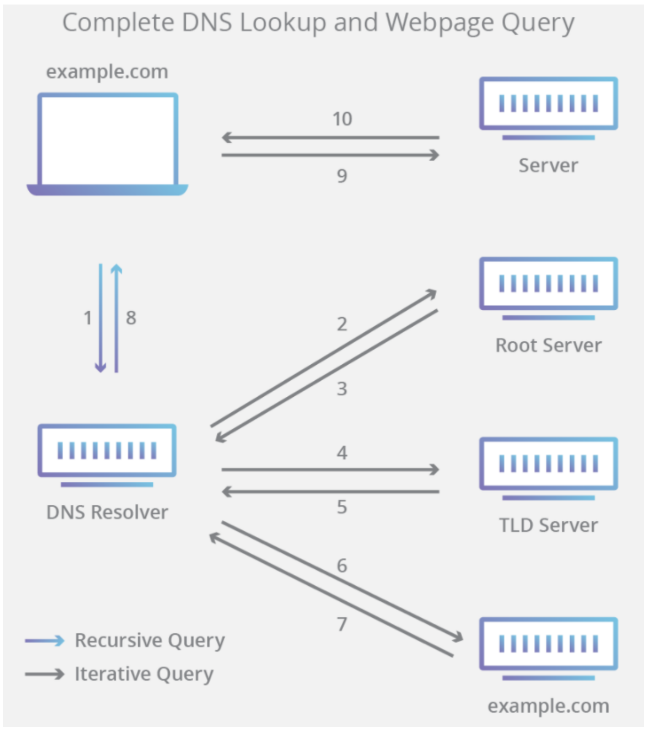


Figure 1: DNS Lookup for a example.com using the traditional DNS

2 Background

Domain Names allow us to map human readable strings to IP addresses. The Domain Name System (DNS) is the mechanism by which we perform this mapping. There are various DNS record types; the most widely used records are A for IPv4 and AAAA for IPv6.

Below are the various steps to resolve a DNS record using the current system (example depicted in Figure 1):

- (1) The browser sends a DNS request to the DNS resolver. The resolver is typically an ISP's resolver, or an open resolver such as Cloudflare's resolver.
- (2,3) The resolver will recursively resolve various parts of the domain starting from the Top Level Domain (TLD) downward. To resolve the TLD it communicates with one of 13 DNS root servers, to find the TLD server.
- (4,5) The resolver communicates with the TLD server to find the authoritative name server for the domain.
- (6,7) The resolver communicates with the authoritative name server to resolve the mapping of example.com to it's IP address.
- (8,9,10) The resolver responds to the browser's request, and the browser finally connects to example.com.

The existing DNS is centralized and hierarchical, which results in the following issues: susceptibility to censorship, squatting of valuable namespace, limited robustness, and a lack of first-class support for security and privacy.

Centralization of DNS fundamentally conflicts with the decentralized nature of the Internet and can lead to censorship. Examples include Turkey's censorship of Twitter [8], the Mirai Botnet cyberattack on DynDNS [9], and blocking of filesharing services such as The Pirate Bay in Belgium [3]. The Great Firewall of China is an example of censorship on a massive scale, involving both DNS tampering and hijacking [12]. Centralization also results in authoritative mappings for every domain name. While helpful in avoiding ambiguity, authoritative mappings has been exploited by domain squatters who take part in a "land-grab" of sorts by acquiring valuable domain names in order to sell them later at extortionate prices [10]. A single definition for a domain name also results in a lack of flexibility to local or personal preferences. For example, the *.gov* TLD is managed used exclusively by the US government regardless of the location of the Internet user.

Robustness is also impacted by the high degree of centralization and recent work has shown DNS may not be as robust as previously thought. 80% of second level domains have fewer than two named AuthServers [4]. Consequently, if those two authoritative name servers are down, all of the domains under those servers cannot be resolved. For performance, many researchers are suggesting to remove Root Name Servers [5] and instead distributing their mappings of TLDs to be maintained by nearby resolvers, creating local mappings similar to those in this paper.

Canonical DNS views security and privacy as an afterthought. DNS queries are sent in clear text, making it possible for eavesdroppers and malicious actors to observe all DNS lookups made by a user. Recently, there has been increasing adoption of DNS-over-TLS (DoT) and DNS-over-HTTPS (DoH), with browser vendors such as Firefox and Chrome leading the way. Unlike HTTPS, which is widely used, this adoption still has a long way to go. For example, Firefox's DoH update strained DNS resolver NextDNS as the load was too much for their network to handle [6]. A distributed approach, such as pDNS, enables encrypted, private name resolution without additional burden on central points of failure.

3 System Overview

pDNS relies on a chain of trust which can be different for different users based on how they want to perceive the entire domain space. While multiple conflicting mappings are supported over all server, any one server must have only one mapping for every domain name. The result of a DNS query depends entirely on the perspective of the requesting server and whom they decide to trust. This is depicted in Figure 2.

Each participating server is empowered to maintain their own mappings or delegate resolution to peer nodes, based on their preference or trust of other peer nodes. This enables multiple, independently consistent, views of DNS while decentralizing the whole system.

For simplicity, we have added support for Type A records only in the current implementation. However this can be easily extended to support AAAA and other types of DNS records as well.

4 Design

4.1 Client

The client is written in Python and operates as a "DNS Server" on localhost on the canonical DNS port 53. This is a DNS server from the point-of-view of the operating system and not to be confused with the backend

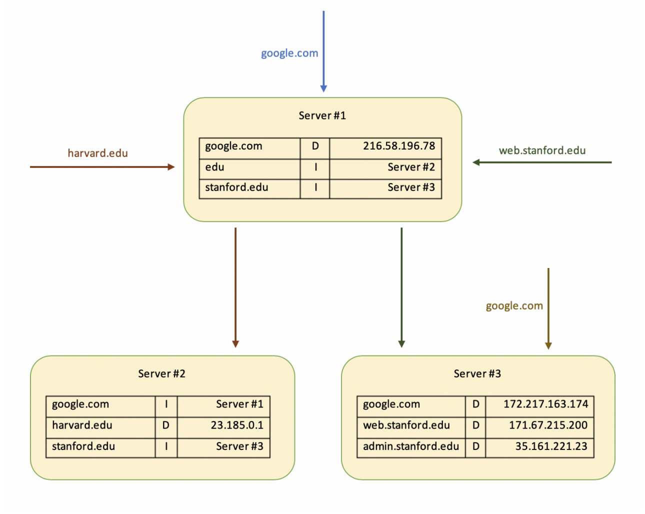

google.com
Server #1
harvard.edu
web.stanford.edu
google.com | D | 216.58.196.78
edu | I | Server #2
stanford.edu | I | Server #3
google.com
Server #2
google.com | I | Server #1
harvard.edu | D | 23.185.0.1
stanford.edu | I | Server #3
Server #3
google.com | D | 172.217.163.174
web.stanford.edu | D | 171.67.215.200
admin.stanford.edu | D | 35.161.221.23

Figure 2: Both Server 2 and Server 3 are added as trusted contact in Server 1 for ‘edu‘ and ‘stanford.edu‘ domains respectively. Domain resolution for ‘harvard.edu‘ and ‘web.stanford.edu‘ using most specific domain match. Server 3 has a different view of google.com as compared to Server 1 and Server 2.

server otherwise described in this paper. This configuration allows for simple cross-platform DNS interception as different platforms provide different hooks into name resolution, whereas this solution is supported easily across platforms. Upon receiving a DNS request, the client extracts the particular query and transforms it into a protocol buffer that is transmitted using gRPC to a backend server. gRPC is a high-performance, multi-language transmission protocol and was a natural fit for a high-throughput, low-latency system such as DNS [2]. The response from the server is then re-encapped into a DNS response and returned to the original caller (for example, the browser). The client is multi-threaded, enabling multiple concurrent DNS requests, which is necessary as a single web page might issue dozens of DNS requests.

4.2 Server

Our server is written in Java and can be run locally or remotely. We use gRPC to service DNS requests from clients and other peer nodes. Since domain resolution can be done from both local mappings and in a P2P fashion, there can be two types of connections in our case: local (client-to-server) and P2P (server-to server). We expose separate API endpoints for local and P2P connections to allow for different access control settings, although this has not been developed yet.

Before proceeding with details of the implementation, we introduce terminology that we will be using:

- **Trusted Contacts:** peer servers which are trusted by a server to resolve DNS for some specific domains/sub-domains. Peers should have static IPs, which encourages users to run their own personal server remotely so they can continue to service requests from their trusted contacts even when moving around.
- **Direct Domain Mapping:** host names which are directly resolved from the local database and do not need to contact a peer.
- **Indirect Domain Mapping:** host names for which we rely on a peer (trusted contacts) to resolve.

4.2.1 Mappings

Each server maintains a list of mappings ("domain mappings") between domain name prefixes and either an IP address or a peer. While we provide adapters so this mapping can be specified in several ways (flat files, database, etc.), we relied on a flat JSON representation for the current implementation. The mappings can optionally specify a "fallback" mapping - if no more specific mapping exists, this mapping will be used. Fallback mappings are useful to specify super-nodes, as described later.

Whenever the server receives a request (either from a peer or from its own client), it first searches for the host name in the direct domain mapping. If the mapping is found, a response is immediately sent to the client. Otherwise, a lookup is done by the domain resolver to find a most specific match (see Figure 2) for the host name based on the resource mapping using a tree-like data model for efficiency. The request is then forwarded to the matched peer node and the response is awaited. Cycles are handled through a configurable limit on number of P2P requests made for a single DNS request.

Each participating server is free to maintain any number of resource/domain mappings in any manner they chose to resolve, based on their preference or trust factor for other peer nodes. This way the DNS resolution is decentralized and users have the freedom to maintain their own set of trusted list for different types of domains/sub-domains. For example, an Indian citizen may choose to have *.gov* domains resolved by the Indian Government's peer node or a Turkish user may choose to have Twitter resolved by a trusted contact in the United States.

4.2.2 Caching

To decrease resolution latency, we have additionally implemented a least-recently used (LRU) cache to store peer-to-peer fetched DNS records. Expiration time is configurable by the user and expired entries are discarded and a fresh fetch is done. This cache has been memory optimized by storing the required data compactly when possible. Whether the cache is enabled and its capacity are both user configurable, but we default to cache enabled and a capacity of 1 million records. More on this is covered later in the Evaluation section.

4.2.3 Traditional DNS Mapping

To support bootstrapping our system, we have further implemented a special type of mapping that responds to requests by proxying them to traditional DNS. We created servers backed by this traditional DNS mapping and configured them as our fallback nodes, making it possible for us to immediately start using pDNS with no change in user-visible behavior.

4.3 Security

Unlike traditional DNS, pDNS enables a secure connection between clients and servers. The connection between the client and server is encrypted and authenticated using TLS. This prevents malicious actors from man-in-the-middling messages between the client and the server. TLS can be enabled for P2P server connections as well.

We support self-signed certificates for authentication to remove the dependency on any third party certificate authority. This aligns with our idea of having a decentralized DNS where the chain of trust is not moderated by third parties. The server's self-signed certificate need to be distributed out-of-band between users and made accessible to the client. This does not present an additional burden as trusted contact mappings must already be shared or authenticated out-of-band under pDNS.

In some cases (e.g., B2B, private servers), it is desired to have a closed group of mappings which is not exposed to a larger audience. This requires authentication of the client so requests from outside this closed group can be rejected. To enable this, we have implemented mutual TLS (mTLS) authentication through gRPC. It ensures that the traffic is secure and both client and server are authenticated with each other. In this case, the client's self-signed certificate also needs to be shared out-of-band and made accessible to the server.

mTLS can be enabled or disabled by the user, but we strongly recommend to enable it only in cases where it is desired to service requests from a closed group of networks. Two-way authentication can reduce throughput and prevents the growth of generic canonical mappings for certain websites which can be shared between peers.

4.4 Privacy

In traditional DNS, all requests are intercepted by infrastructure providers (of which there can be many) before being resolved. It is trivial to observe and associate a domain lookup request with the user making the request. In pDNS, the lookup can be decentralized using different trusted contacts for different domains/sub-domains and therefore these central observation points are eliminated. The user has the full freedom to choose their own set of (and as many) trusted contacts leading to enhanced privacy. In addition, pDNS's first-class support of TLS ensures requests cannot be read by malicious actors.

While resolving the resource for a specific domain, pDNS retrieves the DNS record one hop at a time rather than finding the end server and directly requesting a DNS resolution from it. While the chain of trusted contacts can lead to several hops on the network before the domain name is resolved, it ensures your query can only be associated to you by direct peers which you have explicitly defined as trustworthy. In this way, we have TOR-like [11] privacy in the system as the participating nodes do not know about the origin of the request or the final server which is resolving the DNS record from its local domain mapping. Furthermore, there can be more than one node which can take up the responsibility of serving specific type of domains or sub-domains. As usage of pDNS increases, it will become more and more difficult to track the origin of the request/response as the server can participate both as a local and global DNS resolver. In this case, we explicit trade the small amount of latency from extra hops for additional privacy.

4.5 Censorship Resistance

As there are no globally authoritative nodes, there is no specific node in the network which can permanently censor the resolution of a specific domain/host name. As pDNS allows for easy reconfiguration and makes it easy to stand up new nodes when one node is censored, the impact of censorship is greatly reduced. It is also possible for the user to remap domains held by squatters in traditional DNS to more useful resources. In short, the local mappings can always be modified to include any volunteer/organisation who can provide a desired way to resolve a host name.

5 Evaluation

5.1 Latency vs Hops

The latency of pDNS is $> 20x$ faster than traditional DNS for direct domain mappings and comparable for a small (< 3) number of hops to peer nodes.

We measured the latency of a worst-case scenario: five peers widely distributed across Germany, India, Japan, Brazil, and the United States and with the LRU cache entirely disabled. In each case, we requested a local domain from the region. The results are shown in Figure 3.

As predicted, the latency and variance increases linearly with the number of hops. While the vast majority of time is dominated by network latency to reach nodes in geographically distant locations, increased network latency is an acknowledged tradeoff of pDNS. In comparison, traditional DNS (represented by Google's DNS servers and Comcast's DNS server) has a fixed cost regardless of the domain name/TLD region.

In actual usage, we expect the cache to serve many queries within a few hops, which results in comparable latency to traditional DNS. In our own experience running pDNS to resolve DNS requests for web browsing there was no observable slowdown.

5.2 Concurrent Requests

We benchmarked the number of concurrent requests that could be handled by the server using Ghz, a gRPC benchmarking tool [7]. We ran the experiment on a machine with 16 cores and 42GB of memory. We ran Ghz using 200 workers, doubling the queries-per-second (QPS) of each worker from 5 to 640. Ghz was configured to use up

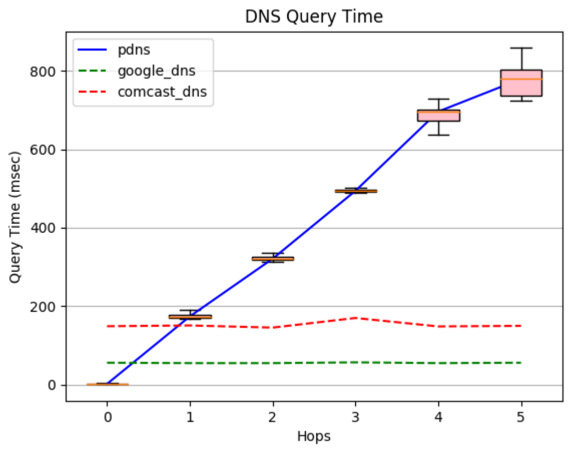


Figure 3: Latency vs number of hops for pDNS and Traditional DNS

to 12 cores, but generally used only two. In order to reduce network and CPU contention, the client connected to the backend server using insecure gRPC channels for a request that could be locally resolved. As a result, the load was able to be handled by our server using two cores. As shown in Figure 4, we are able to sustain 8K QPS on our server backend using only 2 cores in these conditions, with latency increasing as we go beyond the 8K threshold. This workload stresses the amount of gRPC connections our server could setup and respond to since the requests were short lived. We believe that with network variance and longer lived requests, our server could support a higher QPS without adding significant queuing on its end.

5.3 Memory Usage

There are two primary components of memory usage in pDNS – the domain mappings & DNS Cache. Both are expected to store millions of DNS records. We are currently supporting only Type A records with an expiry time, and all the memory calculation are based on it.

To benchmark memory usage, we used the Alexa Top Sites API [1] and found the average size of top 1M host names to be 18 chars. We conservatively assumed the average size of host names to be 50 chars for all our calculations.

The DNS cache is a memory-optimised LRU cache which uses 150MB to store 1 million DNS records. The cache stores IP addresses as **int** instead of **string**.

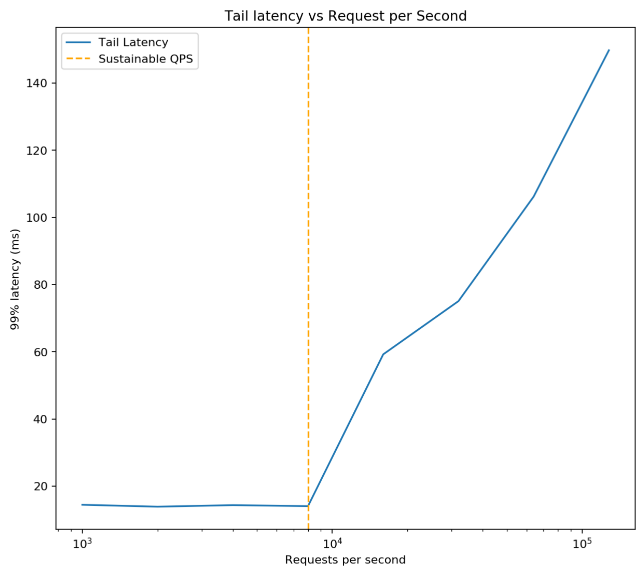


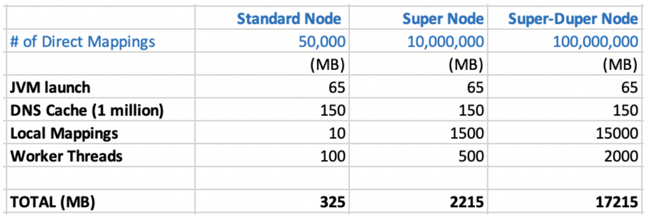
Figure 4: Tail Latency vs QPS

This leads to lower memory requirement with a minimal computational cost for the int⇔string conversion while fetching/pushing data into cache.

We also leveraged the concept of compact strings introduced in Java 9 to further reduce our memory footprint. As we are dealing with large numbers of String objects (millions of host names), we strongly recommend using Java 9 (or higher) for better performance.

The memory impact of domain mappings, resource mappings, DNS cache, the JVM, and worker threads are listed in Figure 5. The RAM/CPU requirements vary depending on the number of mappings. We have defined three broad categories of nodes based on the number of mappings to evaluate the feasibility of deployment for widespread usage:

- **Standard nodes:** up to 50k hostnames resolved directly, and depends on other nodes for DNS resolution. Nodes operated by individuals are likely to fall under this category.
- **Super nodes:** 10M hostnames resolved directly, used indirectly by Standard nodes for many mappings. Nodes operated by small or domain-specific organizations are likely to fall under this category.
- **Super-Duper nodes:** 100M hostnames resolved directly. Usage can be thought similar to TLDs in the traditional DNS, resolving the long-tail of requests. Nodes operated by large organizations (corporations, governments, non-profits) are likely to fall under this category.



	Standard Node	Super Node	Super-Duper Node
# of Direct Mappings	50,000	10,000,000	100,000,000
	(MB)	(MB)	(MB)
JVM launch	65	65	65
DNS Cache (1 million)	150	150	150
Local Mappings	10	1500	15000
Worker Threads	100	500	2000
TOTAL (MB)	**325**	**2215**	**17215**

Figure 5: Memory required by different categories of nodes

For standard nodes, the memory requirement is 325 MB which we believe is reasonable for running locally or on a small cloud virtual machine.

Even for larger nodes, we note that the cost of running such servers is not prohibitive (17.2 GB for Super-Duper nodes). Thus, pDNS Super and Super-Duper nodes can be easily stood up, making it a game changer for decentralization.

6 Future Work

The implementation of pDNS described in this paper provides a prototype that can be extended into a serious decentralization of DNS. Future work on this system should include: support for other DNS record types, making it easy to start using pDNS, through the traditional DNS fallback described earlier and improving discovery of peer nodes, supporting access control lists to encourage more control over nodes for small users, and improving CPU and memory utilization to scale to even larger nodes for organizations. With these changes, we hope pDNS can achieve critical mass and the traditional fallback can be deprecated entirely.

7 Conclusion

We present pDNS, a decentralized P2P DNS which enables a user-specified, local mapping of the domain space with security and privacy built in. We show that such a system would be resistant to censorship. Lastly we evaluate the performance of our system demonstrating comparable performance to traditional DNS and present the requirements for different node types within the system.

References

[1] Alexa - Top sites.

[2] gRPC - A high-performance, open source universal RPC framework.

[3] BAF vs Belgacom and Telenet.

[4] ALLMAN, M. Comments on DNS Robustness. In *ACM Internet Measurement Conference* (Nov. 2018).

[5] ALLMAN, M. On Eliminating Root Nameservers from the DNS. In *ACM SIGCOMM HotNets* (Nov. 2019).

[6] BRINKMANN, M. Firefox 77.0.1 will be released today to fix one issue - ghacks tech news, Jun 2020.

[7] D, B. ghz · simple grpc benchmarking and load testing tool.

[8] FARID, F. Turkey has blocked wikipedia and is censoring twitter, Apr 2017.

[9] HILTON, S. Dyn analysis summary of friday october 21 attack: Dyn blog, 2016.

[10] HOGUE, R. What is domain squatting and what can you do about it?, May 2019.

[11] MCCOY, D., BAUER, K., GRUNWALD, D., TADAYOSHI, K., AND SICKER, D. Shining Light in Dark Places: Understanding the Tor Network. University of Colorado.

[12] XU, Y. Deconstructing the great firewall of china, Apr 2019.