Peer to Peer Domain Name System (P2PN-DNS)

John Grun Arjun Ohri Raj Balaji

Abstract

In this paper, we introduce the Peer to Peer Network Domain Name System (P2PN-DNS) a distributed implementation of the Domain Name System (DNS) that hopes to offer solutions to shortcomings of the current DNS such as susceptibility to outages while mitigating attempts of domain name censorship, and provide a fault tolerant name lookup service for software containers that is independent of upstream servers. Performance metrics including the time to reach consistency between nodes, service availability in lieu of loss of nodes, and average response time to DNS queries will be examined.

I Introduction

The Domain Name System (DNS) is a critical element of internet infrastructure that associates IP addresses to human readable domain names. E.g Google.com is located a IP:172.217.6.238. When DNS was introduced in 1987 with the RFC 1034 /RFC 1035 [17/18] standards, the internet consisted of relatively few computers where a simple hierarchical tree model functioned well. As the internet scaled, DNS was expanded to accommodate the exponential growth in computers. In this regard, DNS has been a tremendous success but, it is not without serious flaws, most notably respectability to attacks on or failures of the Root DNS servers. A Root DNS Server acts as the authoritative source for the entire network.

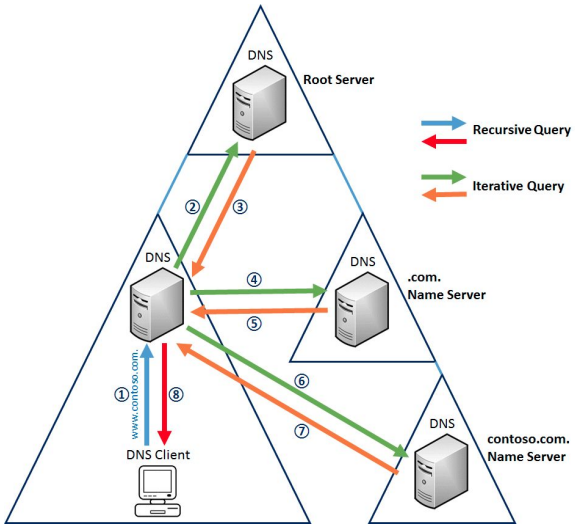

Figure 1: DNS tree structure [24]

In part due to the tree structure of DNS, there are only thirteen DNS Root Servers in the world. An attacker only needs to target a few DNS root servers to cripple IP address to domain name translation in an entire region effectively blocking internet access for most users. Attacks against DNS Root Servers have occurred and appear to be becoming more frequent from various entities, such as hackers, or state actors. It is also quite conceivable that DNS would be a target in wartime. Additionally, centrally located servers are vulnerable to regional events such as power outages or natural disasters.

Another issue that has been seen in the wild is the censorship of websites by state, corporate, or other actors via DNS poisoning. DNS poisoning blocks access to websites by disrupting the domain name lookup between a DNS resolver and the regional DNS server. E.g Google.com is located at IP:?.?.?.?. China [26] and Iran [25] have both used DNS poisoning to block access to internet domains. Corporations such as Charter, Comcast, and Optonline [3] have used DNS poisoning to reroute search results away from competitor websites or to collect advertising profits.

An additional limitation of the existing DNS system is a lack of software container support. Existing DNS solutions for software containers either rely upon a DNS repeater or upon a standard DNS server. These methods simply extend the existing DNS system. If the upstream DNS is made inoperable or poisoned, the containers will be affected as well. This dependency upon existing DNS introduces a single point of failure into what would otherwise be fault tolerant architecture.

The aforementioned problems encountered are a consequence of the centralized hierarchical architecture of DNS. A different architecture can be defined to address these problems while providing the same functionality. We propose to build a decentralized Peer to Peer Network DNS (P2PN-DNS) that will not rely on central servers. We intended to take the best practices from existing distributed robust protocols such as bitTorrent and bitcoin to accomplish our goal. Also, each P2PN-DNS node should be able to contain an internal store of the domain name IP address relations without relying on additional software or servers.

II Related and Previous Work

There have been several different methods over the years to address some or all of the aforementioned problems with DNS.

Amazon Web Services (AWS) offers a implementation of DNS known as Amazon Route 53. Route 53 is believed to be a distributed DNS system but, architectural details have not been forthcoming from Amazon.[27]

Another project DC/OS, (the Distributed Cloud Operating System) contains an internal DNS repeater. While the DC/OS DNS repeater is distributed on the internal network and consists of multiple nodes, it is not a true distributed peer to peer DNS. The DC/OS DNS is a redundant DNS system that acts as a DNS forwarder from a DNS server further up the DNS tree making it vulnerable to the same issues as DNS.[28] See https://dcos.io/

A third related work, Kad Node claims to be a small peer to peer DNS resolver or to act as a DNS proxy. Kad node stores DNS records in a distributed hash table, similar to our proposal but, it differs in that it does not appear to utilize a hash based DNS update ordering scheme. At

the writing of this paper Kad Node appears to be inactive.[29] See Kad Node at https://github.com/kadtools/kad

III Contribution

Technical Problems

The main technical concerns affecting a distributed DNS are the same as those encountered in any system where information is distributed between disparate nodes, namely data consistency, availability, and network partition tolerance. As stated by the CAP (Consistency, Availability, and Partition Tolerance) theorem, it is impossible for a distributed data store to simultaneously provide all three of the guarantees of consistency, availability, and partition tolerance at any given time. This implies compromises must be made between the three. In the case of a distributed DNS the decision as to which guarantees to support and which to neglect is relatively straight forward. In a real world environment, computers crash, network connections fail, and infrastructure can be disabled, thus network partition tolerance is a requirement of any realistic distributed DNS. As a critical service of internet infrastructure, DNS must be always available for the internet to operate properly. Thus availability is a required guarantee as well.

With partition tolerance and availability as required guarantees, continuous consistency becomes impossible to guarantee according to the CAP theorem. In the case of DNS, continuous consistency is not a requirement and eventual consistency is more than sufficient. In fact, currently a DNS record updates are eventually consistent and can take up to 24 hours to complete.

Partition tolerance and availability can both be addressed by designing each node to operate independently of all the others on the network. The independent nodes need only exchange update information to ensure eventual consistency of the DNS records. In this architecture, the more nodes added to a network, the more reliable the system will become.

In a distributed system relying upon eventual consistency to make data agree across nodes, the issue of order of updates arises, since there is the possibility of updates arriving at other nodes in a different order than they must be applied to ensure agreement between nodes. This can be addressed by writing updates to a commit log prior to writing the DNS update. A commit log is a data structure that keeps track of the operations to be performed in first in first out (FIFO) order. When a new DNS record update is received by a node, the update is first written to the commit log. While changes are in the commit log, actions can be taken to ensure the correct ordering of the updates. The ordering of the commits can be verified and corrected by comparing timestamps or by using a hashing based scheme. An update will only be removed from the commit log once the node has finished updating the DNS record.

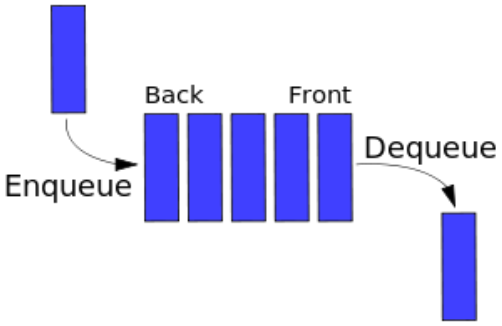

Figure 2: Commit Log Functionality [16]

There are some properties of DNS record updates that allow for some simplifications. Since the update history of each domain name in DNS is independent of all other domain names, we do not need to ensure ordering between different domain names. This property can be utilized to simplify the consistency requirement since it greatly limits the amount of possible combinations in ordering. We need only establish an update chain on a per domain name basis.

The update history can be established with timestamping. Each update is time stamped, and if every node is synced to the same clock the order of updates can be ordered correctly. If the nodes do not have a common clock source this method fails to guarantee the correct update ordering. This ordering problem is what data structures and algorithms such as Merkle trees and blockchains were developed to solve. A Blockchain or Merkle tree takes a hash of a data record, then the next data update in line is a hash of the new update combined with the previous hash. The update received by a node can be confirmed ordered correctly by hashing the previous hash with the current update and comparing the result. If the hash is the same, the DNS record can be updated. If the hash is incorrect a Quorum can be called to reach a consensus between nodes. Since a record update is only concerned about the most recent update and is not dependent upon the entire history of the update chain the ordering requirement can be relaxed to only include the last few updates. In this case, a Quorum can serve the purpose of correcting out of order entries. This situation may arise if some nodes do not receive a DNS update or network issues cause updates to arrive in the wrong order.

The next technical challenge is how to store the DNS records. Since DNS are retrieved based upon the domain name, the most logical method is a simple key value store, where the domain name is used as the key and the corresponding ip address is the value.

Another issue that arises in the real world is the need to automatically find peers. In a small scale system, manually assigning peer address is possible but, as the system scales this method quickly becomes untenable. Nodes must employ a method to locate each other without the intervention of humans. One such method that has proven to be quite effective in other distributed applications is the distributed hash table (DHT). A distributed hash table (DHT) provides a lookup service similar to a hash table: (*key*, *value*) pairs are stored in a DHT, and any participating node can efficiently retrieve the value associated with a given key. Responsibility for maintaining the mapping from keys to values

is distributed among the nodes, in such a way that a change in the set of participants causes a minimal amount of disruption. This allows a DHT to scale to an extremely large numbers of nodes while handling continual node arrivals, and departures.

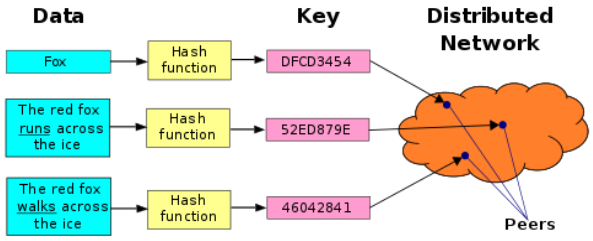

Figure 2: Distributed Hash Table Functionality [15]

Possibly the hardest technical problem to address in a distributed system is known as the "Trust Problem." The trust problem brings forth the question of which node is friendly, and which is malicious. If this implementation was utilized on the open web, the trust problem can be addressed with strategies such as proof of work as employed by technologies such as Blockchain. Nevertheless, if only running on a isolated network, methods such as trusted tokens would are viable. This problem is beyond the scope of this paper but, the authors felt it was important enough to mention and it will have to be addressed in future releases.

IV Algorithm and Implementation

The implementation of P2PN-DNS involved the union of several aforementioned key concepts in distributed computing, the key-value store, distributed hash table, and the commit log.

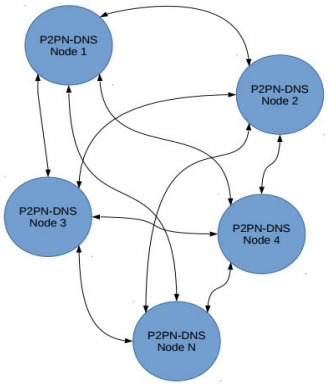

Figure 4: Implementation Block Diagram

At the most basic level DNS can be viewed as a key value store where the domain name serves as the key and the IP address serves as the value. In practice, the DNS domain name to ip address relation is called a resource record and may contain a plethora of data including IP address, CNAME, TXT, etc. For our demonstration only a simple domain name to ip address was implemented.

The main data structure enabling P2PN-DNS is the distributed hash table. In theory a distributed hash table appears simple but, in practice the implementation of the data structure and overlay network can be very labor and time intensive. There was no reason to reinvent the wheel for P2PN-DNS so, OpenDHT was used for the DHT. OpenDHT aligns well with the scope and goals of P2PN-DNS while bundling in support for other desirable features such as IPv4 and IPv6, public key cryptography, distributed shared keys, and an overall concise, and clear approach. OpenDHT can be found at https://github.com/savoirfairelinux/opendht

Since OpenDHT was originally targeted similar distributed applications, the key-value store and commit log functionality were already present and did not have to be implemented independently.

The other major facet of P2PN-DNS was the implementation of the DNS protocol. DNS protocol support was based upon a bare bones implementation of DNS called SimpleDNS. The C source code had to be heavily modified to make it compatible with the C++ elements of P2PN-DNS. Most eventenly the use of C++ 11 std::function as callbacks. Additionally, the DNS update (22) message had to be implemented. The DNS update (22) message did not become a public facing feature in the current DNS system and as such it is not included in most DNS resolvers. Systems that do implement a DNS update scheme are called Dynamic Dns or (DDNS) and have accompanying attenication to update a DNS record. (DDNS) is not directly compatible with standard DNS and relies upon an authoritative entity to make the DNS update to the rest of the Domain Name System. In the interests of time and simplicity, a minimal DNS update scheme was implemented. This shortcoming will be rectified in future releases. Please see for the original SimpleDNS source code at https://github.com/mwarning/SimpleDNS

The ordering of the updates is determined strictly based upon timestamp in this release. This timestamp method works as long as all the nodes are time synced. This is not a valid assumption in a real world environment as nodes may be operating on seperate networks with differing time sources. In future releases a hash bashed ordering scheme as mentioned in Section 3 will be investigated.

The source code for the P2PN-DNS and further documentation can be found at: https://github.com/P2PN-DNS/P2PN-DNS

V Results and Analysis

Experiment and Evaluation

There are a few metrics that matter in a real world application such as the response time to DNS queries, service availability in lieu of loss of nodes, and the average time required to sync records across nodes. For our experiments we propose to examine these metrics and compare, where applicable, to the current DNS system. Dnsmasq will serve as the comparison benchmark as it is a widely deployed DNS forwarder for DNS queries.

To measure the response time of the nodes to a DNS request, a DNS request was sent to a P2PN-DNS node via the dig utility, and the record response was recorded. The time taken from request to response was recorded by Wireshark. The same method was employed against DNS(dnsmasq). Five common domain names were chosen; google.com, Amazon.com, Nytimes.com, alibaba.com, and stackoverflow.com. Three samples were taken for each domain name on both P2PN-DNS and dnsmasq to establish a trend and to compensate

for outliers. This resulted in 15 queries per dnsmasq and 15 queries for P2PN-DNS. Dnsmasq was running on same network as P2PN-DNS. P2PN-DNS records were added via the DNS update message prior to running the experiment.

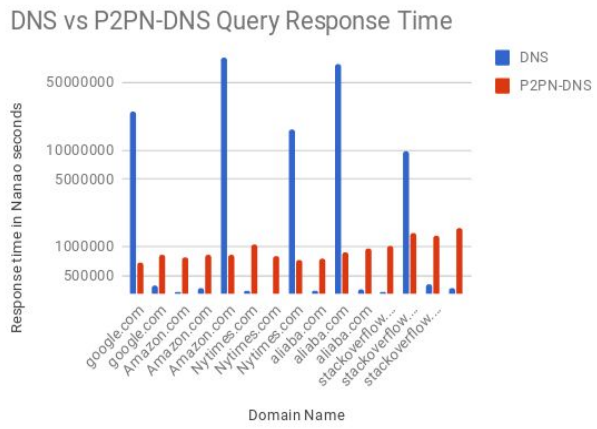

Figure 5: Query Response Times of DNS versus P2PN-DNS

The first request for each domain name to Dnsmasq was of longer duration.This was due to Dnsmasq requesting current information from a higher tier DNS server(Google DNS 8.8.8.8). The second and third requests were served from the Dnsmasq local cache, hence the much shorter duration in response time. P2PN-DNS had slightly longer response time than the Dnsmasq cached response but, less than the first non cached result. This behavior was to be expected as P2PN-DNS has to search the DHT on every lookup and currently does not have local caching enabled. Response time of P2PN-DNS could be improved by enabling local caching in future releases.

To test to the availability of P2PN-DNS in lieu of loss of nodes, is to observe the behavior of the P2PN-DNS nodes when nodes disappear from the network. To conduct this experiment, five nodes were started. A DNS record update message was sent to one of the nodes. The nodes were allowed to reach a consistent state where all nodes are informed of the updated record as observed by querying each node for the updated record. One node was be taken offline at a time. Each time a node was removed, the DNS records were checked for consistency between the remaining nodes.

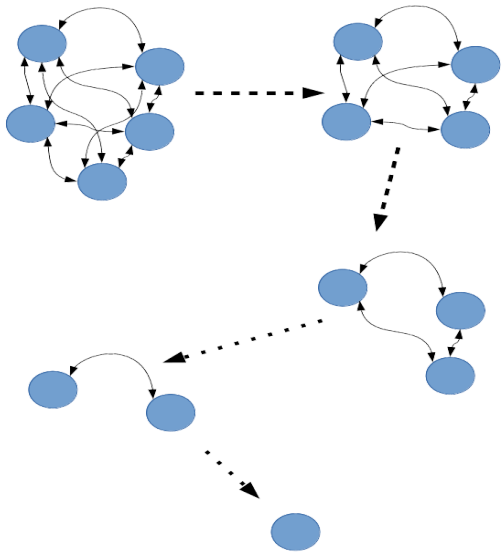

Figure 6: Progression of loss of nodes to test availability

The DNS record was returned correctly even after only one node of the original five remained. This shows that the P2PN-DNS can fulfill the availability and partition tolerance guarantees even as nodes are removed from the network.

To observe the amount of time required to sync records across nodes, the timestamp in the commit log was compared between two different nodes. The difference in the arrival timestamp between the two nodes is directly correlated to the time required to sync records. To ensure accurate timestamping, all nodes were tied to an common clock and synced with Network Time Protocol (NTP).

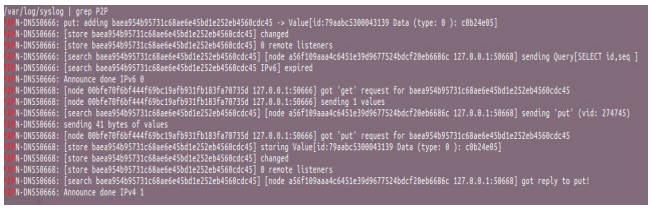

Figure 7: Syslog output from two nodes

In the figure above, the syslog output of two nodes can be observed. To produce these results, a DNS update packet was sent to one node. Syslog was monitored for logs containing information about the record update between the two nodes. The update to a record can be seen in one node and in less than one second later, the update is available on the other node. Network capabilities have an impact on the overall record sync time and can affect time jitter. The network variability(jitter) timing issue was avoided by running the nodes on the same network with a common NTP server.

VI Conclusion

With our implementation of a distributed DNS we were able to find a workable tradeoff between consistency, availability, and partition tolerance. By conducting thorough experiments and analysis, we showed that P2PN-DNS can guarantee eventually consistency, availability, and partition tolerance while also providing a comparable amount time required to process DNS queries as compared to existing DNS solutions such as DNSmasq.P2PN-DNS was shown to be available even as nodes were removed giving credence to the claim that P2PN-DNS would be less susceptible than current DNS to regional outages or purposeful attacks.

Additionally, the distributed nature of P2PN-DNS makes it applicable for software containers which would benefit from a fault tolerant solution such as P2PN-DNS.

VII References

[1] Domain Name System". En.wikipedia.org. N.p., 2017. Web. 2 November 2017.https://en.wikipedia.org/wiki/Domain_Name_System

[2] "Distributed Denial of Service Attacks on Root Nameservers ". En.wikipedia.org. N.p., 2017. Web. 2 November 2017. https://en.wikipedia.org/wiki/Distributed_denial-of-service_attacks_on_root_nameservers

[3] "I Fought My ISPS Bad Behavior and Won". Eric Helgeson. Web. 2 November 2017. https://erichelgeson.github.io/blog/2013/12/31/i-fought-my-isps-bad-behavior-and-won/

[4] Eckersley, Technical Analysis by Peter. "Widespread Hijacking of Search Traffic in the United States." Electronic Frontier Foundation, 14 Oct. 2011. Web. 2 November 2017.

https://www.eff.org/deeplinks/2011/07/widespread-search-hijacking-in-the-us
[5] “Transaction Log”. En.wikipedia.org. N.p., 2017. Web. 2 November 2017. https://en.wikipedia.org/wiki/Transaction_log
[6] “Merkle Tree”. En.wikipedia.org. N.p., 2017. Web. 2 November 2017. https://en.wikipedia.org/wiki/Merkle_tree
[7] Kangasharju, Jussi. Chapter 4: Distributed Systems: Replication and Consistency. N.p., 2017. Web. 7 November 2017. https://www.cs.helsinki.fi/webfm_send/1256
[8] “Blockchain”. En.wikipedia.org. N.p., 2017. Web. 7 November 2017. https://en.wikipedia.org/wiki/Blockchain
[9] Jacquin, Ludovic, et al. “The Trust Problem in Modern Network Infrastructures.” SpringerLink, Springer, Cham, 28 Apr. 2015. N.p., 2017. Web. 7 November 2017. https://link.springer.com/chapter/10.1007/978-3-319-25360-2_10
[10] “ACID”. En.wikipedia.org. N.p., 2017. Web. 8 November 2017. https://en.wikipedia.org/wiki/ACID
[11] DataStax Academy Follow. “A Deep Dive Into Understanding Apache Cassandra.”LinkedIn SlideShare, 25 Sept. 2013. N.p., 2017. Web. 2 December 2017. https://www.slideshare.net/planetcassandra/a-deep-dive-into-understanding-apache-cassandra
[12] “Peer-to-Peer (P2P) Systems.” SlidePlayer. N.p., 2017. Web. 2 December 2017 http://slideplayer.com/slide/4168557/
[13] “Non-Transitive Connectivity and DHTs “https://www.usenix.org/legacy/events/worlds05/tech/full_papers/freedman/freedman_html/index.html
[14] “Amazon Route 53”. En.wikipedia.org. N.p., 2017. Web. 6 December 2017. https://en.wikipedia.org/wiki/Amazon_Route_53
[15] “Distributed hash table” .En.wikipedia.org. N.p., 2017. Web. 6 December 2017. https://en.wikipedia.org/wiki/Distributed_hash_table
[16] “FIFO (computing and electronics)” .En.wikipedia.org. N.p., 2017. Web. 6 December 2017. https://en.wikipedia.org/wiki/FIFO_(computing_and_electronics)
[17] “RFC 1034 DOMAIN NAMES - CONCEPTS AND FACILITIES”, N.p., 2017. Web. 2 November 2017. https://www.ietf.org/rfc/rfc1034.txt
[18] “RFC 1035 DOMAIN NAMES - IMPLEMENTATION AND SPECIFICATION” , N.p., 2017. Web. 2 November 2017. https://www.ietf.org/rfc/rfc1035.txt
[19] “Embedded DNS server in user-defined networks” N.p., 2017. Web. 2 November 2017. https://docs.docker.com/engine/userguide/networking/configure-dns/
[20] “OpenDHT” Web. 2 November 2017. https://github.com/savoirfairelinux/opendht
[21] “SimpleDNS: Web. 2 November 2017. https://github.com/mwarning/SimpleDNS
[22] “ Dynamic Updates in the Domain Name System (DNS UPDATE)” Web. 2 November 2017. https://tools.ietf.org/html/rfc2136
[23] “ You can’t Peer to Peer the DNS” https://nohats.ca/wordpress/blog/2012/04/09/you-cant-p2p-the-dns-and-have-it-too/

[24] “DNS Server” https://gitlearning.wordpress.com/2015/06/23/dns-server/

[25] “Internet censorship in Iran” https://en.wikipedia.org/wiki/Internet_censorship_in_Iran

[26] “Great Firewall” https://en.wikipedia.org/wiki/Great_Firewall

[27] “ Amazon Route 53” https://en.wikipedia.org/wiki/Amazon_Route_53

[28] “DC/OS” https://dcos.io/

[29] “Kad Node” https://github.com/kadtools/kad