Peer to Peer Domain Name System (P2PN-DNS)

Arjun Ohri John Grun 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 that hopes to offer solutions for some shortcomings of current DNS such as susceptibility to cyber attacks and censorship. It is also believed distributed domain name system like P2PN-DNS could provide a fault tolerant name lookup service for software containers. Performance metrics such as the time to reach consistency, availability in lieu of loss of nodes, and average response to queries will also be examined

I Introduction

The Domain Name System (DNS) is critical elements of internet infrastructure that exists to associate IP addresses to human readable domain names. When DNS was first introduced as the RFC 1034/RFC 1035 [17/18] standards in 1987 the internet consisted of relatively few networks where simpler hierarchical system made much more sense. As the internet scaled the same hierarchical DNS system was expanded to accommodate the exponential growth in users and networks. In this regard, the current DNS system has been a tremendous success but, it is not without serious flaws, most notably attacks on or failures of “root” DNS servers and domain name censorship. Due to the tree like hierarchical structure of DNS, there are only thirteen Domain Name System root name server clusters in the world. An attacker only needs to target these few DNS root servers in order to cripple critical internet infrastructure. In fact attacks against the DNS have occurred and appear to more frequent from various entities, such as hackers, cyber terrorists, or state actors. It is also quite conceivable that critical internet infrastructure such as the DNS would be a major target during a war. Additionally, centrally located servers are also vulnerable to regional events such as power outages or natural disasters. Another issue that has occurred is the censorship of websites via DNS injection attacks by state, corporate, and independent actors. These are intended to block access to websites by disrupting the domain name lookup between a DNS resolver and the regional DNS server. This is often done to block access to resources a government does not wish its citizens to have access to as in the case of China and Iran, or if a corporation seeks to block access to competitor websites or rerouting search results to collect advertising profits as seen with Charter, Comcast, Optonline, and other major ISPs [3]

An additional limitation of the existing DNS system is a lack of software container support. Existing DNS solutions for name address lookup in containers either rely upon a DNS server provided with a framework, or a common DNS server. Both of these methods simply extend the existing DNS system. These methods introduce a single point of failure for multiple containers that would have otherwise been mostly independent.

The aforementioned problems encountered are a consequence of the relatively centralized hierarchical architecture of DNS. a different architecture can be defined to address these problems while providing the same functionality as the existing DNS system. We propose to build a decentralized peer to peer DNS system that will not rely on a central server(s).We intended to take the best practices from distributed proven attack and tamper resistant protocols such as bitTorrent and bitcoin in order to accomplish our goals. Additionally, the software should be able to run on standard COTS computing systems and contain an internal store of the domain name IP address relations.

II Related and Previous Work

There have been several implementations of Distributed Peer to Peer DNS by different methods over the years. For example, part of its Amazon Web Services (AWS), the internet titan offers its own unique implementation of DNS known as Amazon Route 53 to developers and businesses. Both scalable and highly available, Route 53 is widely versatile as well as offering full resolution with IPv6 as well. The name Route 53 is derived from the fact that TCP or UDP utilize port 53 to address DNS server requests.This is believed to be a distributed DNS system but, architectural details are not forthcoming from Amazon.

Moreover, DC/OS, an open-source, distributed operating system based on the Apache Mesos kernel. It utilizes a proxy DNS designated for internal DNS queries within a data center but, it is not a true peer to peer DNS. It is a redundant DNS system that accepts the first DNS response to a client's request.

Another related work, Kad Node claims to be a small Peer to Peer DNS resolve which can intercept domain queries on the system level or act as an DNS proxy. Kad Node has not been adopted for for production systems due to unproven technical claims of a superior solution to existing DNS. Kad Node is also advertised as a solution to a political issue while not actually focusing on a legitimate technical issue.

Focusing on something simpler such as intercontainer communications would have proved beneficial. The aforementioned related works appear to have failed at adoption since they are trying to replace a 30 year old proven system that until now has not been critically compromised. All try to fix a larger political problem when a smaller technical problem should be addressed first such as locating

software containers in a cluster, offloading traffic from a root DNS server, or providing redundancy of DNS lookups.

III Contribution

Technical Problems

The main technical concerns affecting a distributed dns system are the same as those encountered in any system where information is distributed between disparate nodes, namely guaranteeing data consistency, availability, and network partition tolerance. As stated by the CAP (Consistency, Availability, and Partition Tolerance) it is impossible for a distributed data store to simultaneously provide more than two of the three guarantees at any given time. Thus compromises must be made between the three, thankfully the decision on which guarantees to support and which to neglect is relatively simple. In a real world deployment, computers crash, network connections fail, and infrastructure can be disabled, thus network partition tolerance is a major requirement of any realistic distributed DNS system. As a required service of the underlying 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 requirements, continuous consistency becomes impossible to guarantee according to the CAP theorem. Thankfully, eventual consistency is more than sufficient in order to accomplish the goals of a DNS service. In fact, currently a DNS record update 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 information to ensure eventual consistency of the DNS records. With this design the more nodes added to a network the more reliable the system should become.

The goal of eventual consistency can be simplified by using a commit log in each node. A commit log keeps track of the operations to be performed on the key-value store. Physically, a commit log is a file or data structure listing changes to the data. If a node goes down it can use the commit log to ensure that update transactions are not lost. When a new record update is received by a node, the update is first written to the commit log. The ordering of the commits can be verified and corrected at this point by comparing timestamps or using a hashing based scheme. The node will process the updates in the commit log in a first in first out (FIFO) order. An update will only be removed from the commit log once the node has finished completing the operation. Additionally, the update records can form the data update portion of a node to node protocol. Nodes need only send the update records to each other.

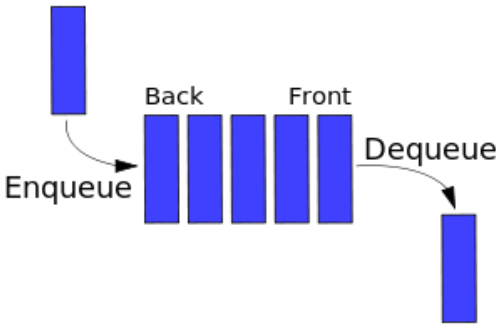

Figure 1: Commit Log Functionality [16]

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. 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 the order between different keys.This property can be utilized to simplify the consistency requirement since it greatly limits the amount of possible combinations in the ordering. We need only establish an update chain on a per domain name basis. This ordering consistency 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 data hash plus the former hash. The next update received by a node can be confirmed as correct in order by hashing the node's last record adding it to the new updates hash and hashing once again and comparing the result. If the hash is correct, the DNS record can be updated. If the hash is incorrect a Quorum can be called to reach a consensus between nodes as to the correct value of the DNS record between nodes. A Quorum can serve the purpose of correcting out of order entries since the ordering requirement can be relaxed to only include the last few updates as a record update is only concerned about the most recent information, and is not dependent upon the entire history of the data.

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 used as 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 such as Bittorrent 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 extremely large numbers of nodes and to handle continual node arrivals, departures, and failures.

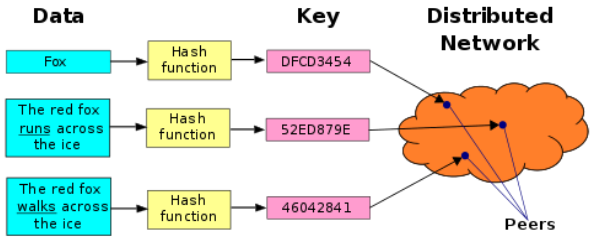

Figure 2: Distributed Hash Table Functionality [15]

Possibly the hardest problem to address is known as the “Trust Problem.” The trust problem brings forth the question of who is trustworthy. In other words, which DNS node is friendly, and which is malicious? If this implementation was utilized on the open web, this would prove to be the most difficult problem. Nevertheless, if only running on a server cloud, different methods such as tokens would be viable.

IV Algorithm and Implementation

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

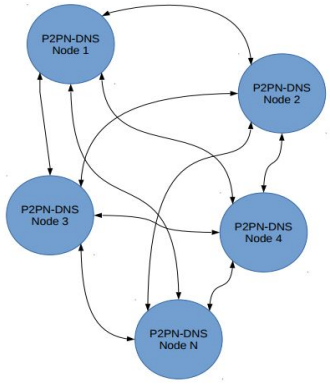

Figure 3: Implementation Block Diagram

At the most basic level the DNS system 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 value is called a resource record and may contain a plethora of data including IP address, CNAME, TXT, etc. Another critical issue is ensuring the correct ordering of the domain name ip address updates. This issue is addressed by writing the changes to a commit log prior to writing to the key-value store. While changes are in the commit log actions can be taken to ensure the correct ordering of the updates. Currently, ordering is determined strictly based upon timestamp.

The enabler of this project is the 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 so OpenDHT was used as the DHT. OpenDHT also has support for IPv4 and IPv6, public key

cryptography, distributed shared keys, and overall quick, concise, and clear approach. OpenDHT seemed to align and agree with the scope and goals of our project. Their lightweight implementation of the distributed hash table was both elegant and impressive, all cohesive in openDHT, which are elements P2PN-DNS also strives to achieve. OpenDHT can be found at https://github.com/savoirfairelinux/opendht

Since OpenDHT was originally targeted for file sharing (Bittorrent), the key-value store and commit log functionality were already present and did not have to be implemented independently.

The other major facet of this project 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 the project. Most eventenly the use of C++ 11 std::function.as callbacks. Additionally, the proposed DNS update 22 message had to be implemented. It must be noted that support for DNS updates did not become a public facing feature. Systems that implement a DNS update scheme are called Dynamic Dns or (DDNS) and have accompanying attenication to update a DNS record. 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. https://github.com/mwarning/SimpleDNS

Another feature that will have to wait for future releases is the the hashing based ordering scheme. The current implementation uses the timestamp to order the DNS updates.

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, and data consistency across 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 well established and widely deployed DNS forwarder for DNS queries for networks.

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 a standard DNS(dnsmasq). Five common domain names were chosen; google.com, Amazon.com, Nytimes.com, alibaba.com, and stackoverflow.com. Three samples were taken of 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.

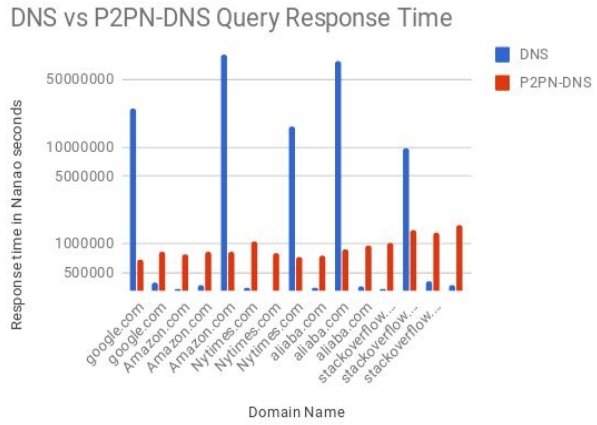

Figure 4: Query Response Times of DNS versus P2PN-DNS

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 requests following were served from 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 but, this is beyond the scope of this paper.

An important test to the availability of the system 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 will be sent to one of the nodes. The nodes will be 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 and checking consistency. One node will be taken offline at a time. Each time, the DNS records will be checked for consistency between the remaining nodes.

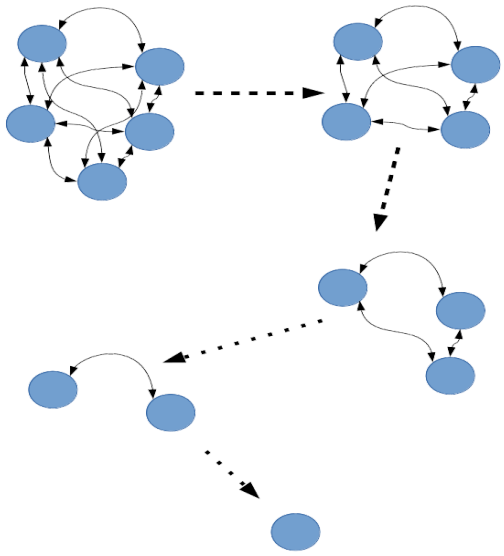

Figure 5: Progression of loss of nodes to test availability

The DNS record was returned correctly even after only one node of the original five remained. This shows that the P2PN-DNS can be available even as nodes are disabled in the network.

To observe the amount of time required to sync records across nodes, the timestamp in the

commit log can be 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 the nodes must be tied to a common accurate clock source. The common clock issue may be addressed with Network Time Protocol (NTP).

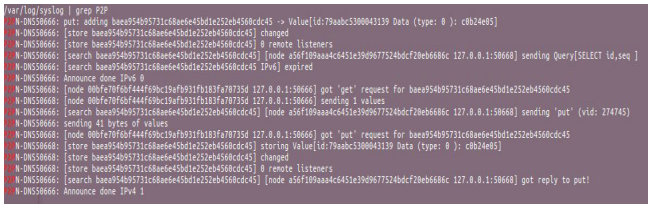

Figure 6: Syslog output from two nodes

In the figure above, the syslog output of two nodes can be observed. To yield 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 to the other node. Network capabilities have an impact on the overall record sync time. The 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 an intensive and thorough experiment and analysis, we developed a eventually consistent, available, and partition tolerant solution that addressed the problems it sought out to due to their ever present nature in the current DNS. Additionally, in respect to the scope of the project, the amount time required to process DNS queries when compared to standard DNS, specifically DNSmasq, is comparable. While resilient, current DNS does present vulnerabilities, making it a prime target for hackers, cyber terrorism, and electronic warfare, whereas in a Peer to Peer network system it is difficult to attack the innumerable amount of nodes to cause damage. DNS should iron out these cracks in its armor, as evaluated by our project, and perhaps heed the solutions and suggestions as presented as well as demonstrated. Additionally, the distributed nature of P2PN-DNS makes it applicable for software containers which would benefit from a fault tolerant solution instead of the single point of failure as seen in current implementations of 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/