TÉCNICO LISBOA

Hybrid Peer-to-Peer Domain Name System

Ricardo Miguel Brenhas Sancho

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

November 2013

Abstract

The Domain Name System (DNS) has existed for several years and as such has had it share of problems. It was not designed at the time when the Internet usage was so wide spread, however it became one of the Internet's central pillars. Having had it's share of problems in regards to exploits, a new type of problem, domain censorship, started appearing recently. Governments censoring domain names that are administered by entities that fall under their jurisdiction. We aim to develop a DNS, that addresses the problems with the current DNS, and at the same time ensure that no central authority exists, so domains names cannot simply be censored. In order to shape our idea of the system, we analysed previous research on Peer-to-Peer (P2P) network and DNS, and came to the conclusion that although implementable, it is not without it's compromises. The biggest issue was latency, but this analysis also showed that some successfully used caching to solve the problem with success. One of the issues, was keeping a connection with existing DNS, so this would facilitate the transition from one system to the other. Our final solution consists of a Hybrid P2P DNS system which borrows some ideas from the analysed systems and implements a Local DNS Nameserver to resolve user queries, while giving the user the ability to choose between the current DNS or the new Distributed approach we developed, or even used both, with the user choosing which domains resolve with which system. The new Distributed DNS is implemented using Kademlia, a Peer-to-Peer overlay network, which works as DHT. Every user acts as a node in the system and stores DNS records.

Keywords: DNS, Peer-to-Peer, P2P, censorship, resilience, distributed, domain name system

Resumo

O DNS, Sistema de nomes de domínios, existe há já vários anos e já teve alguns problemas. Não foi desenhado numa altura em que a Internet tinha um uso tão massificado, no entanto tornou-se num dos pilares centrais da Internet. Tendo tido a sua mão de cheia de problemas relacionados com ataques ao sistema nomeadamente ao seu código e arquitectura, uma nova de categoria de problemas começou a aparecer recentemente, a censura de nomes de domínio. Governos a censurarem nomes de domínio que são administrados por entidades que caiem sobre a sua jurisdição. Procuramos desenvolver um DNS que resolva os problemas com o actual DNS, e que ao mesmo tempo garanta que não existe uma autoridade central no mesmo, de forma a que nomes de domínio não possam simplesmente ser censurados. De modo a desenvolvermos a nossa ideia analisamos investigação feita previamente sobre redes P2P e DNS, e chegamos à conclusão que embora seja possível implementar, existem restrições. O maior problema foi em relação ao tempos de latência, embora a nossa análise também tenha demonstrado que existem casos em que esse problema foi resolvido com sucesso recorrendo a uma cache. Um outro problema, era manter uma ligação com o actual DNS de modo a facilitar a transição de um sistema para o outro. A nossa solução final consiste no sistema P2P DNS Híbrido, que aproveita algumas ideias dos sistemas previamente analisados e implementa um servidor local de resolução de nomes, enquanto dá ao utilizador a opção de escolher entre o sistema de DNS actual ou sistema desenvolvido por nós, ou mesmo usar ambos, sendo que o ultilizador escolhe que domínios resolvem no sistema actual e que domínios resolvem usando o sistema desenvolvido por nós. O novo DNS distribuído foi implementado recorrendo ao Kademlia, uma rede P2P, que funciona como uma Distributed Hash Table (DHT). Cada utilizador actua como um nó do sistema e guarda registos DNS.

Queria agradecer ao Prof. Doutor Ricardo Jorge Feliciano Lopes Pereira por ter aceite orientar-me neste tema e pelo apoio e orientação que me deu ao longo deste ano lectivo. Queria também agradecer ao Dr. Thomas Bocek, da Universidade de Zurique, em primeiro pelo desenvolvimento da biblioteca que serviu de base para a minha rede peer-to-peer e em segundo pela ajuda que me deu na mailing list quando precisei.

Quero agradecer ao meus colegas de faculdade, sem ordem nenhuma em especial, Duarte Barbosa e Ricardo Cunha pela ajuda prestada a experimentarem a interface gráfica e darem feedback na mesma e pelo restante feedback dado ao longo de todo o desenvolvimento.

Quero agradecer ao meus pais, Vitor Sancho e Sara Sancho, que tornaram tudo isto possível e à minha irmã, Filipa Sancho, por me estar sempre a chatear e a ajudar. À minha família e à minha segunda família por todo apoio e simplesmente por estarem presentes quando precisei. Por fim quero agradecer a todos os que me foram dando motivação ao longo deste ano de trabalho.

Contents

List of Tables

List of Figures

List of Acronyms

CDF	Cumulative Distribution Function
DDOS	Distributed Denial of Service
DHT	Distributed Hash Table
DOS	Denial of Service
DNS	Domain Name System
DNSSEC	Domain Name System Security Extensions
DZF	Distributed Zone File
FTP	File Transfer Protocol
GUI	Graphical User Interface
HP2PDNS	Hybrid Peer-to-Peer Domain Name System
HTTPS	HyperText Transfer Protocol Secure
IEEE	Institute of Electrical and Electronics Engineers
LAN	Local Area Network
P2P	Peer-to-Peer
PGP	Pretty Good Privacy
SMTP	Simple Mail Transfer Protocol
SWT	The Standard Widget Toolkit
QoS	Quality of Service

Chapter 1

Introduction

1.1 Background

Since the start of mankind there has always been a leader either elected, or self imposed. Territories have been marked and laws have been made, these laws are to be followed by everyone inside said territory. And different territories have different laws. And this introduces our problem, who has control over the Internet? Who decides what should or shouldn't be allowed on the Internet? Since the beginning of the Internet there have always been disagreements on who controls it, and to what extent can the same laws currently applied offline be applied online, where does one's jurisdiction begin and where does it end? The problem with the Internet is it has no real boundaries, people from opposite sides of the planet can talk and interact to a certain extent like if they were in the same room. This problem has lead to censoring of various content by governments or private entities, that claim to have the right to censor said content. But is the content really in their jurisdiction? Different claims over the right to censor content on the Internet have been made based on which the country the server is located, based on the country the administrator lives in or simply based on the domain the country uses. Take for example this document, say the authors are located in two different countries, let's call them A and B, the document itself is being published to the Internet on a server located in country C and the domain used access the website where the document is, is managed by an entity in country D. Who has the right to censor this document if they feel the content is not to their pleasure? Say country D decides to take over the domain, should other users from other countries be restricted of access just because those countries do not like the content, even though the content is considered legitimate in their own country? Although there are other problems to the censorship problem, domain name censorship is the one we aim to focus on and aim to solve. The DNS is one of the

Internet's central pillars, it's make the Internet easily accessible to anyone by giving users simple names to memorise instead of IP's to access websites. The Internet's freedom is at stake and a simple click from anywhere in the world can block everyone's access to information. The Internet strives to be free of political interests and economic interests. The authors feel the Internet is and should stay a place where freedom of speech is highly incentivised, and the governments should not have the power to control the flow of information as easily as they do. This means the government should not be able to take down websites simply because they want and can, when that content poses no harm to anyone, or is in anyway related to criminal activities.

This problem has been more prominent in recent years with governments and private entities overstepping their legal reach to censor domains which they deemed are not acceptable However sometimes either on purpose or simply because an error was made along the way of processing the domain seizures, completely legal domains get taken down. This might affect business owners, trying to make a move in the online market, because of such errors their credibility is affected and user might not trust the store again. There is a disastrous potential to drive a business to the ground, simply cut their profits with bad publicity[1][2]. Others may simply see their domain being sized, and as such for instance if a user maintains a blog, credibility in what they say might be affected effectively losing followers. Again we can see the discussion of jurisdiction, reach of power being bought up. Where does it end? How much, is too much?

In the US, Government agencies have even managed to cause concern with in the US government which saw the actions taken as an overreach of power[3]. However some governments point out that this is a normal course of action and others have gone as far as asking for the management of some domains to be put in the hands of a more neutral entity which might not be subject to the whims of governments trying to overstep their jurisdiction[4]. This has been going on for a couple of years now[5], how many have had their domains unjustly taken away? How much research went into identifying the infringing domains and consequently ordering their seizure?

Errors are a common thing, especially via automated systems, going on to a different type of censorship to demonstrate the degree to what these systems can be wrong, we turn to Google's take down reports. Google regularly publishes take down requests[6] it receives for it's indexing engine. However some sanity check is done by Google itself otherwise some entities would have censored their own content on the giant's search engine numerous times[7][8].

[1] http://goo.gl/XU2fjN
[2] http://www.nytimes.com/2008/03/04/us/04bar.html
[3] http://goo.gl/id9iHW
[4] http://www.wired.com/threatlevel/2012/03/feds-seize-foreign-sites/
[5] http://www.wired.com/threatlevel/2013/06/domains-seized/
[6] http://www.google.com/transparencyreport/removals/copyright/
[7] http://goo.gl/Umv4bW
[8] http://goo.gl/TjbJR3

1.1.1 DNS History

The DNS as we know was designed in 1987. However, domain names predate the DNS, and RFC 952[1] presents the specification for the Internet host table. Briefly explained this was a simply file name host.txt that was distributed across various servers using FTP. As the use of the Internet grew the bandwidth necessary to distribute the file grew as well and this solution did not scale well. So in 1987, RFC's 1034[2] and 1035[3] presented the Domain Name System we now currently use. The system has evolved over the years, exploits have been corrected and changes have been made to allow the system to scale better, but the core of the system remains the same, and so we rely on one entity to provide us with correct and updated domain names.

1.1.2 How DNS works?

This is an explanation of the DNS system and the author assumes the reader has no previous knowledge of DNS and Domain Name System Security Extensions (DNSSEC), the reader may skip this section, if he is familiar with DNS and DNSSEC. This is not however an extensive explanation, and should the reader want more detail he should refer to the approriate RFC's already mentioned above.

The DNS is a hierachy of names, however the root has no name[4]. It is composed by a set of TLD's (Top Level Domains), where there are generic names (edu, com, net, org, gov, mil, int), and country codes, which are two letter codes and come from ISO 3166[1][2]. There is a name hiearychy from those TLD's, generally it is flat, meaning that example.com will be registered to a company and the company will configure the name hiearchy as they wish. With country TLD's some use another level adding co, com, net, org and so one to their two letter code (com.pt or org.pt). Generic TLD's were created to be used by specific entities, they are categorised as follows:

- *edu* - is used for educational organizations.
- *com* - is used for comercial purposes, for companies.
- *net* - is for computers belonging to network providers NIC (Network information centers) and NOC (Network operation centers).
- *org* - is for organizations that don't fit anywhere else.
- *int* - is for organizations established by international treaties or for international databases.

[1] http://www.iso.org/iso/country_codes.htm
[2] http://www.iso.org/iso/country_names_and_code_elements

- *gov* - was intended for goverment organizations world wide but later was changed to allow only US goverment organizations.

IANA[1] is the entity responsible for the overall coordination and management of the DNS and delegation of portions of the namespace. An Internet Registry, InterNIC was selected to handle the registry of generic domains names, North American domain names and all undelegated domain names that don't fall under the previous two. RIPE NCC[2] handles the registry of European domain names, and APNIC[3] and AFRINIC[4] do the same but for Asia-Pacific and Africa respectively. These then may further delegate their zones, however when choosing a manager for a domain certains aspects should be taken in to account, such as the ability to contact their staff or if the registration of names will be fair and the rules applied to everyone equally.

The DNS is composed of root servers, 13 authorative root servers[5] to be more specific, which administer all domain names, that exist (There can be other domain names that are not controlled by these root servers however those exist on a separate network). These servers then delegate zones, such as com or pt, to other entities which will be responsible for maintaining those zones and ensuring a fair use and registration of the domain names.

The DNS main components are the "Domain Name Space and Resource Records which are specifications for a tree structured namespace and data associated with the names", Nameservers which are server programs that "hold information about the domain's tree structure and set information", and Resolvers which "extract information from Nameservers in response to a client request".

When designing the DNS the goal was for it to able to provide the same information that the previous system did, the HOST.TXT file, while allowing it to be maintained in a distributed manner, have no obvious limits in size, interoperate across various enviroments, and provide tolerable performance[5], which the HOSTS.TXT was not being able to do any more.

1.2 Proposed Solution

Looking at the Internet's current state and previous attempts to bring down various networks, the ones that stand out as the most resilient, are P2P networks. Since they have no central authority it is fairly difficult to completely shut them down. The only way to do this is to completely

[1] http://www.iana.org/
[2] http://www.ripe.net/
[3] http://www.apnic.net/
[4] http://www.afrinic.net/
[5] http://www.iana.org/domains/root/servers

shutdown all machines that use the network, this means having access to all physical machines or having the power to force the machine administrators to shutdown the machine. For this to happen the machine connected to the network or the user have to be in the country demanding such takedown. This scenario is highly unlikely as P2P networks tend to have users from various countries, with machines in various countries. For this reason, we chose to build a DNS alternative whose communication is based on P2P networks while still being able to use the current DNS. This will allow the users to choose between different name resolution services for each domain, which will allow them to circumvent any censorship that might be imposed on the current DNS. Further, every user that uses the hybrid P2P system is part of the network and will act as a server effectively keeping domain records on their machine and spreading them to others that request it. This will give the system greater resilience since no entity controls the whole network.

The system's source code will, in the spirit of free comunication and censorship, be available to anyone. So any user can make changes to the current system, this means that security aspects have to be taken in to consideration and we cannot rely on obscurity as a security measure.

This solution combines a variety of different proposed solutions along with a prototype for a fully functioning client to the end user. This prototype will include a blacklist, a key management system, the DHT networking, which is used to store the DNS data and an interface to manage the black list and keys. These components together allow us to build a working solution for the end user, which allows the user to keep using existing infrastructure while also benifiting from a censor free DNS.

1.3 Thesis Contribution

In this thesis we have provided an arquitecture for a P2P based DNS, which can at the same time run along the current DNS, as well as defined the protocols for comunication. We have designed a prototype to prove our design and have run tests using planetlab nodes to validate the prototype. Further we have written an article which talks about the first iteration of our prototype and data gathered from the tests run to validate the prototype. The paper under the name Hybrid Peer-to-Peer DNS was accepted at ICNC 2014 Technical Program and will be presented in February.

1.4 Outline

This document describes the research and work developed and it is organized as follows:

- **Chapter 1** presents the motivation, background and proposed solution.
- **Chapter 2** describes the previous work in the field.
- **Chapter 3** describes the system requirements and the architecture of GBus.
- **Chapter 4** describes the implementation of GBus and the technologies chosen.
- **Chapter 5** describes the evaluation tests performed and the corresponding results.
- **Chapter 6** summarizes the work developed and future work.

Chapter 2

Related Work

2.1 Peer-to-Peer Domain Name Systems

There have been some attempts at building a DNS system based on a P2P network. Some have opted to completely reimplement it, effectively presenting an alternative to the currently deployed system, while others opted to create a system that can coexist with the existing one, simply adding resilience to it. In this section we will present those existing solutions.

First we will look at a solution which is purely P2P. Next we will take a look at hybrid P2P solutions, this means these solutions use part of the existing DNS and add a P2P network along side the existing DNS. The use of this P2P network will be explained as we look at each of these solutions since the use for the P2P network varies, and can be of purely defensive nature, or act as a load balancer.

The authors start off by showing that many DNS related problems are due to bad configurations[6]. At least 35% of DNS traffic is never answered or receives a negative response and 18% of DNS traffic is destined to the root servers. They go on to say that having DNS served over a Chord ring would eliminate the need to have every system administrator be an expert in running name-servers, it would also provide load balancing and the root servers would be eliminated, which would provide better resilience.

Their implementation uses DHash, a Chord based distributed hash table, to store and lookup records. Lookups have a granularity of Resource Record Set (RRSet), as in the current DNS. When an RRSet1 is retrieved, the signature needs to be verified since DNSSEC is used to authenticate the RRSets. This is done by doing a lookup for another RRSet2 with the key that signed the RRSet1 and then retrieving yet another RRSet3 with the key that signed RRSet2, up until a well known key is retrieved. DHash uses consistent key hashing and caches content along the path. It also provides replication and moves data around when a node fails ensuring that there are always *n* replicas. To add or create DDNS RRSet the owner prepares the RRSet, signs it and inserts it into the DHash. The DHash will then verify the signature before accepting the data. As with current DNS servers, where the IP's of root servers are known, it is assumed that the key of the root of the hierarchy is known.

Results from this implementation showed that althought storage is well balanced by the use of DHash, some nodes still had many more records than others, however this was compensanted by DHash's caching mechanism which showed that after serving each of it's popular blocks around ten times, the records are cached in enough nodes, which means a query will likely be answered by another node other than the node responsible for the record. Other tests also showed that popular records are rapidly propagated through the network meaning that every node will have a cached copy very quickly. The authors then go on to compare the latency of lookups from their implementation against results taken from *Jung. et Al.*[7], since they did not have a big enough network to compare their results against the DNS currently deployed in the Internet. However the comparison shows a bigger latency when using a Peer-to-Peer network, the author's median was 350ms while the results they had to compare from conventional DNS stated a median time of 43ms.

They then go on to state the problems, in the current DNS, that their implementation solves. The first problem cited is administration. The author states that DDNS eliminates most of the administration problems by "automatically providing a routing infrastructure for finding name information". The author then goes on to state the problems in the current DNS outlined in the handbook for the Berkeley DNS Server, BIND, Albitz and Liu[8] that DDNS solves. DDNS solves the Slave server cannot load zone data, Loss of network connectivity, Missing subdomain delegation, Incorrect subdomain delegation, loops in name server resolution and the fact that DNS requires that domain owners manage two types of information about the domains, that is data about hosts and data about name service routing.

From this paper we gather that a Peer-to-Peer implementation of the DNS is possible, however not without compromises. For starters the median latency in lookups was around 8 times more. This means that a user on average would have to wait 8 times more to be able to retrieve an IP address for a domain when accessing it for the first time. However after that, subsequent accesses would be quicker since the record would have been cached along the way, this also introduces another important notion, caching, which seems to go a long way to make the response time in a distributed DNS acceptable. A lot of problems have been solved with this implementation, which will be a helpful study for our solution.

In the category of hybrid DNS, we first present systems that use a P2P network as a layer to protect the current DNS.

HARD-DNS is such a solution, it does not actually create a whole new DNS, rather it creates a P2P layer which complements the existing DNS[9]. Hard-DNS adds a P2P layer before the actual DNS. So the user contacts a HARD-DNS Server that then contacts the actual servers in the DNS.

HARD-DNS goal is to solve DDoS and cache poisoning attacks on the current DNS.

"HARD-DNS can be implemented as a standalone or built on top of platforms such as Content Delivery Network or Cloud Infrastructures"[9].

Again, with this implementation clients only need to change their nameservers, other than that no other change is needed on the client side. This seems to be a major concern for all the systems analysed so far. As with every new application, system, etc, it is easier to have mass adoption if changes needed to be done by the client are inexistent or very few.

HARD-DNS servers query recursively and then combine their answers using majority, this solves the cache poisoning problem. If an attacker attempted cache poisoning they would poison a HARD-DNS machine, and only that machine would be poisoned. When using HARD-DNS costumers pick a random machine, which means that the probability of picking a machine that has been poisoned is small, and even then, since answers are choosen using a majority, the client would not see this. As already mentioned HARD-DNS polls a subset of its names servers and returns a majority vote, this makes it almost impossible for an attacker to compromise all or the majority of the servers. To solve DDoS, HARD-DNS automatically does load balancing to move the load away to other servers.

Results for HARD-DNS show that it provides great resilence and is effective against cache poisoning.

Although this solution does not implement a true peer-to-peer DNS, it does show that a Peer-to-Peer Network for the DNS does help solve some of its current problems, even if this is just a layer which hides the current DNS.

This next solution, just like HARD-DNS, creates layer over the existing DNS system. This layer serves as a defense mechanism and as a routing alternative in case a certain country gets blocked from querying[10]. If this happens it will route the query to another server. That is, if a Nameserver for some reason is blocked from serving or deliberately chooses not to serve a particular DNS record, then the system will go around it.

The Peer-to-Peer network used is a DHT, namely chord. This solution simply constructs a Chord ring but when requesting a DNS record, if a node is blocked or failed it goes on to the next node in the routing table, and so on, effectively using the routing table and selecting the next node if the current node cannot resolve the DNS query throught the current DNS system. So the node receives a query and tries to resolve it through the normal DNS lookup procedure, if it can't because it is blocked, then it forwards the query on to the next node, and so on until a node is able to resolve the query. As you can see, this protects a country from DNS blocks (effectively censorship), by routing the query on to another node which can be in another country. This is done until the query can be resolved by a node, which happens when that node is not in the country where the query originated from, thus circumventing the DNS block (censorship). If however all known nodes cannot resolve the query then it is dropped.

From this analysis we can see that a good way to effectively circumvent DNS blocks is to have nodes spread all over the world to circuvemnt any blockades in place in countries. Unless there is a worldwide block then there will always be a node able to resolve the query. However in DNSs that are implemented using a peer-to-peer network this would not be an issue as nodes are normally scattered around the world. This solution creates a layer, of servers which are scattered around the world, on top of the existing DNS does to solve the problem of country specific censorship.

The next solution also creates a layer before the current DNS. However this P2P network serves all domains up to the second-level, it distributes the load with the current DNS, meaning that up to second-level domains won't be served by the DNS. So example.com, is on a peer-to-peer network and from there a query would be routed to a fixed server which manages example.com and every other subdomain. example.com or example.net is a now flatten and although it exists there is no hiearchy there for the P2P routing is this solution.

In this next solution the authors state that the goal was to "design a DNS which provides both high degree of robustness against targeted attacks and acceptable lookup efficiency"[11], which they called HDNS. The authors built a hybrid P2P based DNS model. Basically the system is composed of both a peer-to-peer overlay network and the traditional DNS model. The peer-to-peer overlay chosen was chord due to it's "simplicity, proven correctness and performance". The hierarchical tree structure of DNS and the flat structure of peer-to-peer networks where combined to create this DNS. The system is divided into two parts, the public zone and the internal zone. The public zone is the backbone network, nodes are organised in a DHT network, in this case a chord ring, and the internal zone is the edge networks, these are networks that sit outside of the DHT network, but have a direct mapping to a node in the DHT. These internal zones are organised according to traditional DNS.

To map the internal zone to the public zone, the root nodes of the internal zone are assigned unique identifiers which are then mapped to the public zone node whose identifier is equal to or follows the identifier of the root node.

Domain names are partitioned into two parts. The first part, named public zone names, includes the top-level domains names and the second-level domains names, while the second part, named internal zone names, includes all the third-level and all other levels after that. For example: example.com is the first part, while anything after that for instance test.example.com, test is the second part.

The root node public zone names are hashed into values and placed in nodes on the public zone according to distribution of keys done by the Chord DHT. Internal zone names are stored in internal zone server that act as traditional DNS Servers.

In order to query a node, a client sends the query to a node on the public zone which finds the node on the DHT responsible for the domain, from there the identifier of the internal zone responsible for the domain is found and the query is rerouted to the internal zone node, from there traditional DNS is used to find the IP and the response is returned through the same path.

The objective of the authors was to test the efficiency of the network topology and as such for their tests all caching mechanism were disabled. As expected, traditional DNS still offers the best efficiency (since there is no cache, the number of average hops is most likely always bigger in a DHT). However, when it comes to robustness chords offers the highest robustness. The important result is that the hybrid approach, offers a higher latency than traditional DNS, but still manages to maintain an acceptable latency while providing more robustness than traditional DNS. Latency in hybrid DNS grows slower than with a pure chord implementation, that is, as the number of nodes in the DHT increases the number of hops does not increase at the same pace.

Again, in this paper it is shown that acceptable results can be achieved even when there is no caching mechanism behind the DHT implementation. It also provides a great insight into the possibilty of complete delegation of a domain name to a company or entity, giving them full control of the domain name while enjoying the benefits of a peer-to-peer DNS.

This next solution is a true hybrid solution creating a DNS system, just like DHash while also allowing for the current DNS to be used. It makes use of the current DNS system and complements it. It also has the ability to run on its own. It does, however, still use root keys and the way the whole system is managed requires trusted entities to run machines 24 hours. This does not really allow for the common user to have its own choosing on which keys it should accept and does not allow for free distribution of domain names, although the same domain can be operated by different operators and those might have better prices in order to compete with each other, still the user can't simply choose to use none of those.

CoDoNS is infact a Hybrid P2P DNS in the sense that it takes the existing DNS and complements it by adding a distributed system that runs along side with it. As the authors state, CoDoNS aims to maintain backwards compatilbity with legacy DNS (current DNS) and at the same time serve as a complete replacement for it[12]. To achieve this, CoDONS uses "structured peer-to-peer overlays and analytically informed proactive caching". DHT's are used as peer-to-peer overlay. A DHT is a self organised network, which is scalable and failure resilient, however, it does have higher lookup costs when compared to local solutions. This problem is solved by the use of an analytically-driven proactive caching layer, called Beehive. Beehive auto replicates the DNS mapping in order to anticipate demand for a specific DNS record.

To use CoDoNS clients, contact a local participant in the CoDoNS network. No changes need to be done on the client side other than changing the nameservers.

In CoDoNS, domains can be added and managed by their owners. Names that have not been added are retrieved from the legacy DNS and CoDoNS maintains these mappings consistent, until the owners add the domains to CoDoNS.

Measurements in Planet Lab showed that CoDoNS substantially descreases the lookup latency, when compared to legacy DNS (current DNS), it is able to handle large flash crowds, which is when a not so popular domain suddenly becomes popular, and it can as well disseminate updates to records very quickly.

CoDoNS allows multiple operators for the same namespace, opening up the market and allowing for more competitive prices to the end client. Multiple operators are able to sign domains from the same namespace, as these are assigned from a shared coordinated pool. This ends up killing the monoply on a namespace.

"CoDoNS derives its performance characteristics from a proactive caching layer called Beehive"[12]. Beehive essentially is a proactive replication framework that enables prefix-matching DHT's, such as pastry used for CoDoNS, to achieve O(1) lookup performance. Beehive achieves this by replicating objects at all nodes with prefix i, this means that if you increse the number of prefixes at which the object is replicated, then the number of average hops needed to find a copy of the object will decrease. However, replicating every object at all nodes, would consume too much space, bandwidth and increase the update latency. The replication is optimzed according to a constant C, which represents the aggregate lookup latency that you want to achieve, and the object popularity. So in effect objects are only replicated in a number of prefixes i. Think of it as levels.

CoDoNS nodes form a peer-to-peer network. Each instituition has one or more servers, which make it act as a large cooperative shared cache. CoDoNS decouples namespace management from name resolution. Nameowners buy a certificate from one of the namespace operators and introduce it into the CoDoNS. CoDoNS allows nameowners to insert, delete, and update records. These records are then replicated on to various nodes, according to Beehive replication policy.

To allow a gradual transition, CoDoNS uses legacy DNS to resolve queries for records that have not been inserted into CoDoNS by nameowners.

As mentioned before, CoDoNS implementation uses Pastry as the peer-to-peer overlay and Beehive as the replication framework. CoDoNS uses Cryptographic delegations and self verifying records based on the DNSSEC standard[13].

Results show CoDoNS to be faster than legacy DNS, except when having to resolve records that were not introduced into CoDoNS. In such cases it has to do a lookup on the legacy DNS, which makes it slower than legacy DNS, since it combines both legacy and CoDoNS lookup latencies. This, however, only happens the first time the domain is accessed, as CoDoNS will then map it in the network automatically. Apart from this, CoDoNS also showed to adapt well to flash crowds. Due to the caching mechanism used CoDoNS demonstrated a great ability to load balance effectively, which resulted in an "even load across the system and incurs low uniform bandwidth and storage overhead at each node". Updating of replicas of records took less than one minute for 99% of the replicas.

From this paper we can again conclude that a peer-to-peer implementation of the DNS is possible and not limited to a particular DHT. We can see that proactive caching goes a long way to reduce the latency increase, when compared to legacy DNS, that we saw on the paper which introduced DDNS, as well as make the system more resilent to flash crowd effects[6]. DDNS partially addresses this problem by caching records as they were requested, however if a node were to be hit by a sudden request for a particular record, that has not yet been cached anywhere else, then it might not even be able to send out one response. Proactive caching resolves this problem by having a record already cached at a certain number of nodes, and increasing that number as more queries arrive. This means that the load will always be balanced and nodes won't become irreponsive due to the sudden increase in queries for a particular record.

Although CoDoNS seems like a particularly good solution it, does not address our main problem, censorship, as the server operators have to be trusted, this implies that one entity controls the entry of new operators into the network and, as such, can directly enforce censorship on domain names.

This next solution uses the P2P network in quite a different way, it attempts to create a P2P network where the keys for DNS records, DNSSEC keys, are distributed, while authorative nameservers are replicated[14]. Using this implementation the DNS can tolerate one third of the authorative namespaces for a zone to fail. The keys used to verify the domains are kept online and distributed using threshold cryptography. To maintain compatibility, it implements the DNSSEC interface[15]. The proposed solution achieves failure resilence against corrupted servers, and integrates the existing replication in the DNS with state-machine replication. However, it assumes the links between clients and servers are authenticated.

To use the service, the client sends a request to all replicas. The replicas agree on a sequence of requests. Suppose one replica receives request r1 from one client and then receives r2 from another client, and another replica receives these requests but in reverse order, r2 followed by r1, they will have to agree on a sequence either r1 before r2 or r2 before r1 so the state across replicas is consistent. One might think that a client would only do this for writing, however to assure that the client receives a correct answer, the client will contact all replicas and choose the answer by majority.

In order to sign responses the server generates a signature share and sends it to all other servers[14]. It then waits for t+1 valid signature shares (distinct). t is such that $t < n/2$, so no information about the zone's private key is leaked.

Because the server accepts t replicas to be corrupt the client must only accept a response if it represents t+1 replicas. So the client receives n - t reponses from distinct replicas which means the client will have a majority of responses.

However, the proposed implementation requires the clients to be modified, so the authors relaxed the solutions goals in order to come up with a solution where the client does not need to be modified. The change was to have the client only send a request to one replica, this replica then disseminates the request. The client will only do one request, but will still receive multiple reponses, now according to the client implementation it can either just accept the first response or choose the response from the gateway that it contacted. Since a response must still be signed by t+1 signature shares, then it must at some point have been valid even if it does not represent the latest values in the system (Replay attack can occur).

The prototype used BIND[1] and implemented the SINTRA replication model in Java[16]. Results showed that reads take from 50 miliseconds to serveral hundred miliseconds, while writes took 20s or more.

However, even though results show quite an increase in delay of reads and writes, the author

[1] https://www.isc.org/software/bind

states that these are reasonable, and that even though writes show quite a big increase, they happen with much less frequency and as such it doesn't have a big impact on the system's performance.

"This solution provides fault-tolerance and security guarantees to secure DNS against an attacker that compromises a fraction of the nameservers in a zone."

As we can see, in this solution although replicating the keys ensures that they can be online without being compromised, it shows that the overhead inccured is also quite a bit, which might make it infeasible for normal use.

2.1.1 Conclusions

Although many of these implementations provide us with solutions that protect against most common attacks in DNS, they fail to address our main issue, a completely censor free DNS. It is our belief that this is only possible to achieve through the use of an open network where anyone can be a node and help serve records, as well as allowing anyone to create and administer domain names.

The fact that all previous solutions relied on some kind of central authority, shows that no solution for the main problem we are trying to solve developed. As such we propose the development of a prototype that allows users to control which DNS service they choose to have their domains resolve with, being that the alternative DNS is free from any centralized decision and as such completely uncensored.

2.2 DNSSec

DNSSEC RFC clearly states that DNSSEC does not attempt to provide confidentiality for queries or responses. We also did not find any reasons to why this should change, or a use case where private DNS records would be of use, as such we keep this premise in our system design. This however does exclude the encryption of traffic between two peers. A new resource record was created to allow the association of keys with DNS names but this does not guarantee that you can request the public key for a DNS Name reliably from the resource record. In DNSSEC keys used to sign the names are either kept online or offline, offline is recommended, if no dynamic update is need, however in our case there is no root key, which means a user is free to change the keys easily whenever he wants. As such we do not put much emphasis on keys being kept offline, however this is always a good practice if it can be achieved. However if an entity signs multiple Distributed Key Files then it is not practical to have the private key offline. DNSSEC further explains that in order to read the Public Key reliably from the DNS, that key itself must be signed by a key the resolver already trusts. In our system the user must obtain the key from the entity that owns it, by accessing their website and downloading the key assuming the traditional DNS has not been compromised. In regards to TTL's, DNSSEC notes that as the TTL changes the signature is no longer valid, and propose various, solutions one being the TTL value defines a maximum TTL and a external value is the actual TTL value. In case that gets tampered with, meaning it is higher than the maximum TTL then the TTL is set to the maximum and counts down from there. To avoid this problem we suggest using end times, meaning a Distributed Zone File will stop being valid when that time expires of course having the time wrong could be a potential security flaw, but if the time on the host is changed that means the computer has been compromised and there are bigger issues to dealt with such as a complete nameserver change to a compromised nameserver. DNSSEC provides no protection for DNS requests meaning this can be tampered with. In our implementation we don't worry about DNS requests, as the potential for flaw, is a DDoS where by tampering with every request from a computer the computer keeps receiving wrong answers and never resolves the domain.

2.3 DHT

Chord, "a distributed lookup protocol", is a peer-to-peer overlay, more specifically a Distributed Hash Table (DHT), this means it works very similarly to a Hashtable, except it cannot guarantee an average lookup of O(1) due to it being distributed throughout several nodes[17]. This means that different values will map to different nodes, however you have the guarantee of consistent hashing this means that when you place a value in a node under a specific key, the next time you find the node associated with the key, you will access the node that has the value, however storage is not provided by chord and must be provided by another layer. This of course is a bit more complicated since in a peer-to-peer network nodes can join and leave without notifiying the network, so some kind of replication has to be in place to guarantee that values will not be lost. The nodes in a Chord DHT are organised as a ring, meaning that if you start at a node and keep going on to the successors, assuming the node does not go offline, you will eventualy comeback to it.

Here we will outline Chord's advantages[17]. The Chord protocol is very scalable, that is the network can grow to have a very large number of nodes and still provide good lookup efficiency of O(log n). In order to be able to efficiently route a lookup a node has to maintain a routing table with O(log n) nodes. Nodes resolve lookups with O(log n) messages. Chord provides us with one operation which given a key maps it onto a node.

Pastry is a "a scalable, distributed object location and routing substrate for wide-area peer-to-peer applications".[18]. Again Pastry offers a very scalable solution and just like Chord, Pastry's nodes form a ring. The proximity metric is left up to the application, and the authors sugest using tools such as tracerouting or comparing subnet maps, however it is really up to the application and as such a close node in Pastry does not mean that the nodes are close when looking at the physical layer. Pastry also uses different metric distances for replication, and for finding a key, which means nodes close using one metric can be far using the second metric.

Kademlia, "a peer-to-peer (key,value) storage and lookup system", uses the XOR metric to determine network proximity meaning that the node's geographic location is not relevant for this peer-to-peer network overlay. Two nodes that are in two different continents can be considered close[19]. Kademlia was specifically designed for file sharing and because of that, it takes into account file replication, and uses the same distance metric for both lookups and replication, to ensure that values mapped to the keys are always accessible, even if a node fails. It is highly scalable and nodes are organised in a tree structure according to their unique identifiers. Other than plain simply building a routing table, Kademlia perfects its routing tables, by favouring nodes which have the best uptime, with less failures.

2.4 Alternative DNS Roots

Alternative DNS Roots exist[1][20]. Most provide the same functionality as the existing root servers while extending it and adding support for more TLD's. Since these root servers provide the same functionality they are connected to the existing root servers, serving as a bridge to the commonly used root servers and serving requests for TLD's through them[21]. This seems as an attractive alternative but does not solve the censorship problem for existing TLD's, only for new ones specifically added by these alternative root servers. Even for new TLD's added by the alternative root servers, this only temporarily removes the problem of censorship since the goverment can easily shutdown these alternative root servers or force them to censor domains using the new TLD's as well.

[1] http://www.opennicproject.org/

2.5 Voting Systems

To help with spreading of keys to verify Distributed Zone Files (DZFs) we decided to build a voting system. This voting system, is nothing more than a reputation system, where users vote, on pairs domain and public key, either negatively or positively. As such we took a look at studies on reputation systems, to see what we could gather from there and use to build our own reputation system, which fits the needs of our Hybrid Peer-to-Peer Domain Name System (HP2PDNS).

We first take a look at Credence a object-based reputation system. We then go on to see what makes a reputation system.

In small networks peers tend to interact with the same group of peers and can over time build a reputation for each peer based on previous local experiences. However, P2P networks tend to span a very large number of peers, and interactions in such large networks tend to be with peers who we personally know nothing about. Credence is a robust and decentralized system which allows peers to confidently gauge object authenticity, the degree to which an object's data matches it's advertised description[22]. To achieve this, Credence uses a network-wide voting scheme with positive and negative votes. It allows clients to weight those votes according to a statistical correlation between the client and it's peers, and lastly clients are allowed to extend the scope of their correlations through selective information sharing. The system is based on object reputation. Objects are considered immutable and, as such the client's evaluation is final. On the other hand a client's evalutation of a peer can change over time.

Credence uses positive and negative votes, where any client may vote on any object. Each client possess a key which he uses to sign the vote, along with the signed vote is included a certificate, which verifies the authenticity of the key. The key and certificate are used to verify authenticity and uniquess of every vote. The authors state that in order to help avoid Sybil attacks, clients are required to download a large file at installation time, they also point out that other mechanisms are available such as solving captachas or cryptographic puzzles during installation.

These mechanisms, which the authors state help avoid Sybil attacks, are useless against open source software since the client can easily change the code to bypass this requirement.

The authors assume that honest clients will generate votes accordingly for each object and as such a client judges the authenticity of an object's based on it's reputation among the client's peers.

Votes are collected and evaluated by a client that wishes to estimate the reputation of an object. However simply listing the votes, would be easy to manipulate. As such votes are weighted based on the client's relationship with each peer. Peers having the same voting preferences, either positive or negative should in time develop a strong positive weight for each others votes. Peers with unrelated votes,should disregard each others votes.

The voting protocol works as follows, first a client issues a vote-gather to receive votes on an object. Each peer responds with a subset of matching votes, if any. This subset is composed of the votes with the heightest local weight. Using this strategy the network does not get flooded with all votes, and votes with higher local weight are dissimenated further down the network. Once the votes have been received, they are cryptographically verified and stored for later use and then their weights are calculated. Using these values the client can then estimate the authenticity of the object and decide wether to fetch it or not. With objects that have no votes the client might have to resort to estimates of popularity in the network.

This voting protocol works only when a sufficient number of votes have been made, and as such the authors believe this provides an incentive for peers to vote.

As the authors show, the system works well, as clients start voting and gathering each others votes throughout the network. They show that as the time progresses, the success rate of queries comes close to the upper bound which is the scenario where all voting data was shared over the network with each peer. At the same time, peer correlations also evolve as peers start finding peers with common interests, and as such start creating smallers groups with share same interests and use each others votes. "Credence requires no trusted entities in the network, resists attacks and effectively identifies trustworthy content and pollution."

However, some mechanisms in Credence's defense rely on the fact that the code is closed source and as such Sybil attacks are harder to pull off, because of the methods employed to mitigate such attacks. In an open sourced software, mechanisms such as captcha's before creating a new identity are ineffective since the client can simply re write the code to bypass that safety mechanism.

More, the use of a certificate signed by a "trusted" third party, solves Credence problem while still maintaining the network completely decentralized. However in our case and to make it impossible to cripple any part of the system, the system as a whole should not have reliance on any one third part. As such we take a look at what is needed to design a reputation system, and see what we can do to design a reputation system to our needs.

In order to design a reputation system the three basic components are **information gathering**, **scoring** and **ranking**, and **response**[23]. These can further be further divided. The authors, in "Taxonomy of Trust", believe these are essential for a reputation system to fulfill it's task and go on to describe possible solutions divised by the community for each mechanism.

They then go on to identify user behaviour and types of adversaries. They identify two types of adversaries, selfish peers and malicious peers. Selfish peers are those that do not want to contribute with resources to the network, an example of these are "freeriders which can be found in file-sharing networks, such as Kazaa and Gnutella." Malicious peers are peers who are willing to use any amount of resources to cause harm to either specific members of the network or to the system as a whole.

Reputation system designers normally target a specific group of adversaries. For example reputation systems with incentive schemes for sharing target freeriders, but are ineffective against malicious peers.

Malicious peers can attack using different techniques:

- Traitors - peers which behave properly to build a positive reputation and then start misbehaving.
- Collusion - when multiple peers join together to cause damage to the system.
- Front peers - peers which build a positive reputation and then provide misinformation to promote actively malicious peers.
- Whitewashers - peers that purposefully leave and rejoin the network with a new identity.
- Denial of Service (DOS) - when a peer brings large amounts of network resources to disrupt the system.

Most research the authors read does not claim to protect against all these attacks, most of them actually focus on selfish peers.

As the authors suggest, some of these system properties can only be achieved using a single point of trust meaning a centralized system. Most of the systems presented fall in between a hybrid, where some parts rely on a single point of trust while the rest of the system is decentralized. For simplicity the authors call systems that require part or all of the system to have a single point of trust as centralized.

Information gathering is the first component the authors identify for a reputation system, it is responsible for collecting information on peers, and assesing how trustworthy they are.

In order to effectively collect information on a peer, one first has to be able to identify each peer. This can be done with various identity schemes. Each scheme relies on certain properties, some of these properties, however, collide with each other and cannot be present in the same scheme. The authors focus on a restrict group of properties:

- Anonymity - The level of anonymity provided can vary, however the ip address is public and almost always used as part of the identity, other solutions are pseudonyms, which can be used in conjunction with ip addresses or on their own.
- Spoof-resistant - An identity must be spoof resistant to prevent impersonation, a common way to prevent this is using public/private keys.
- Unforgable -These can be identities provided by a central entity and as such no peer can generate new identities. This prevents against whitewashers and sybil attacks. Decentralized systems use identities that are costly to produce, but are not unforgable.

Any of these solutions, can with enough resources, be compromised.

When collecting information, the sources of information are of importance. The quality and quantity are diametrically opposed. As the information collected increases, the credibility of each piece of information normally decreases.

A Peer's reputation is based on information collected from that peer based on one or multiple sources. These sources can be based on personal experiencies of experencies from other trusted sources, which can be external trusted peers, one hop trusted peers, multi-hop trusted peers, and a global system. As we widen the sources available, we also widen the amount of information available on peers, however the credibility of the information also decreases.

One of the biggest hurdles in reputation systems is maintaing the validity of opinions. One way, that reputation system designers normally combat this, is by giving higher scores to personal opinions first, followed by opinions from close friends, and so on until we reach opinions given by complete strangers.

The second component identified by the authors is scoring and ranking. Once a peer's transaction history has been collected and properly weighted, a score is computed for that peer. Using a peer's score an agent ranks the peers that responded to a request for a specific object. For example if agent A request object B from peers C and D, after computing their score the agent will rank them with the peer with the highest score being the one of the biggest probability of providing the correct object B. There can also be a minimum threshold on the score, meaning if none of the peers achieved a minimum score, the agent will not retrieve object B from them as the risk, according to the score is not worth it.

Reputation systems, other than being used as guides to choose transactional partners, can also be used to motivate peers to contribute. These can be in the form of incentives, or punishments.

Incentives are mechanisms which give something to the agent in return for their contribution. They are an effective tool against selfishness, as they offset the cost of contributing with a benefit.

Incentives normally come in the form of improved service. For example, agents contributing with resources can be rewarded with faster download speeds, better quality, or more quantity (the amount available for the user to choose from). There is also another form of incentive which is money, distributing the cost of distributing the files over a P2P network, can result in each user making a micro payment to offset the cost.

Punishments are also another way to motivate peers, this time it motivates them to act accordingly and not misbehave. If the system can correctly identify misbehaving peers, it can take other measures to punish these users, other than just warning other peers. A common punishment is banning the peer, either for a period of time or permantely.

As the authors point out, developing a reputation system, is something that requires thinking of many separate design problems and choices. It is also important to remember that certain system properties conflict with each other, and while it would be great to build a system which can take of all problems, there is currently no known solution, however the authors believe this will in time be overcome, and a fully decentralized system with all the functionality, reliability and security of a centralized system will be possible.

This for our design posses a problem as we want to keep the system secure, reliable, while mainting it completly decentralized to avoid possible points of censorship, as such we will try to take from here all possible points for a decentralized reputation system and build a voting system that fits our needs. This system will not however attempt to be a fully decentralized system with all the functionality, reliability and security of a centralized system, but will attempt to build a voting system that is decentralized and can to some extent have reliable voting.

Chapter 3

Architecture

This chapter presents Hybrid Distribured P2P DNS (HDDNS). It is intended to be deployed on the Internet as an alternative to the current DNS, however while both still exist it provides for an easy integration with the existing DNS, allowing the user to have freedom of choice on where his queries are served from. The system is intended to be used by all, from home users to major corporations. The goal is to provide a censorship free DNS while at the same time addressing flaws in the current DNS. We aim to make this solution secure and resilient to DDoS attacks, as well as man-in-the-middle, attacks.

We will start by laying out the basic architecture design and then follow to explain each module and component in more detail. The system is composed of a DNS Dummy Server, a DHT node, a daemon which runs on the clients computer, and a DNS Translator. Effectively these all come together to form the whole system. However some of these are not essential for the system to function properly but only required for some features. We will use *utl.pt* as the domain for our use cases, and will focus our use cases on browser use. However, these could be applied to any domain access, except if explicitly mentioned otherwise.

3.1 Overview

To use the Hybrid Distributed DNS the user will download the executable, and install it. It will install the DNS Dummy Server, the DHT Node and the DNS Translator, the daemon comes as a separate component and is not required for the system to work. The DNS Dummy Server is a local server binded to localhost by default. The user's nameservers are changed to 127.0.0.1. From then on, when accessing the website utl.pt using a browser, it will lookup the record for *utl.pt*

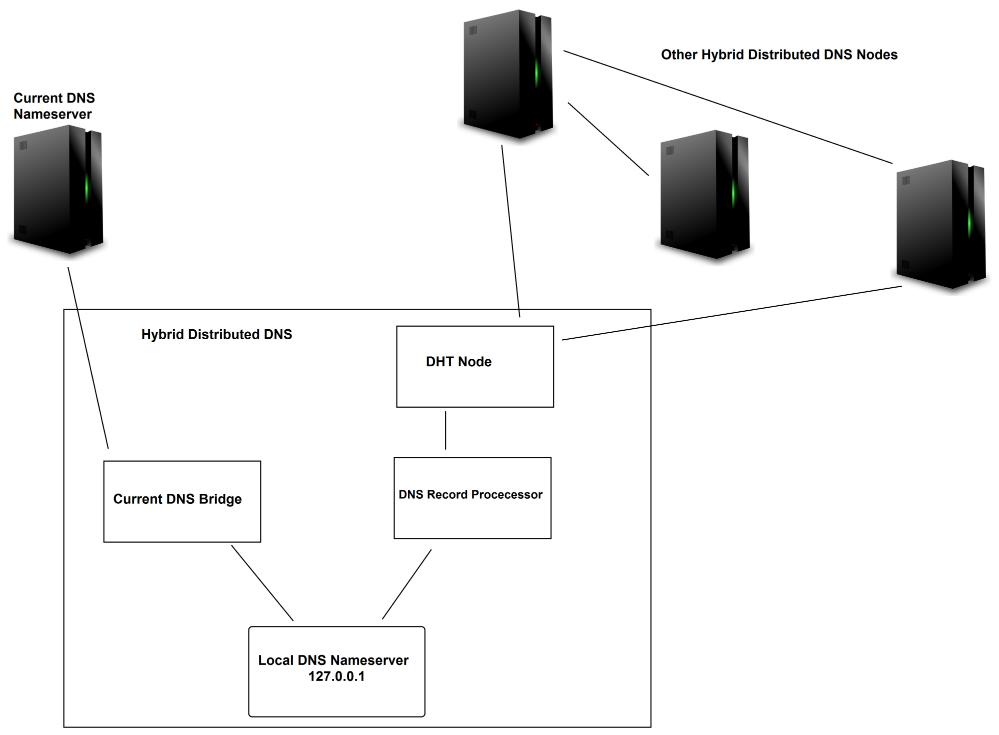


Figure 3.1: Hybrid DNS System

using the DNS Dummy Server, the DNS Dummy Server will then proceed to re-route the query to the traditional nameservers (these can be changed per user configuration) or to the DNS Translator. Depending on weather or not utl.pt is in the blacklist, as true, either the traditional nameservers will be used or the DNS Translator will be contacted. If utl.pt is not in the blacklist the query for utl.pt will be fowarded to the traditional nameservers and these are able to retrieve the record for utl.pt then the DNS Dummy Server will send the response back to the browser which will display the website. If utl.pt is blacklisted, meaning it appears in the blacklist as true, the the query is fowarded to the DNS Translator which upon receiving the query, routes it to the DHT Node. The DHT Node will then send back the Distributed Zone File if it has it. The DNS Translator forms a regular DNS Response using the Data from the Distributed Zone File and then sends it back to the DNS Dummy Server. The DHT Node upon receiving the query will look it up in it's cache, if it is present then it will send the Distributed Zone File back, if not it will proceed to request the peer-to-peer network, as soon as a response arrives or a certain time threshold is reached, a response is sent back to the DNS Translator, either a Distributed Zone File or a negative response.

Fig. 3.2 shows a high-level description of the system with it's layers, having as the point of entrance to the application the local nameserver. The DNS server although not part of the system is

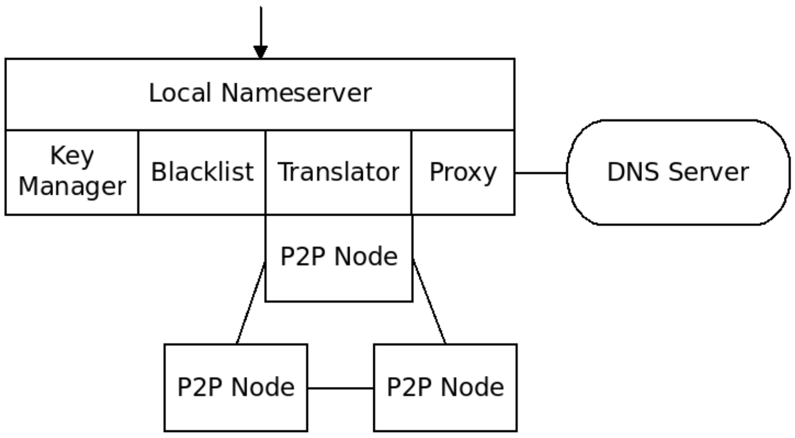


Figure 3.2: High-level description of system layers.

displayed so the reader can see how the system interacts with it. This interaction is an important part of the system since it's what effectively gives the system's hybrid capabilities, by allowing it to retrieve DNS queries from a DNS server belonging to the current DNS system.

This is one of the possible and simple use cases, in this case the user doesn't take advantage of features such as blacklisting or public keys to verify the issuer of the Distributed Zone File.

Blacklisting serves the purpose of identifying censored domains and explicitly forcing the DNS Dummy Server to only retrieve results from the peer-to-peer network. Private Public key pairs are used to sign and authenticate the owner of a DZF. The contents of a DZF is always signed using a Private Key, and the Public Key is included in the DZF. As such, a user can decide which Public Keys he recognizes as trustworthy and thus be more selective as to what Distributed Zone Files it accepts. The Public Key is included so nodes that only verify the **ac!** (**ac!**) is not corrupt and is correctly signed, can do so without having the Public Key before hand. For example the domain utl.pt is signed by Universidade Tecnica de Lisboa's Private Key, PrivK1, and using the corresponding Public Key, PubK1, users can verify that the DZF is indeed from Universidade Tecnica de Lisboa and hasn't been tampered with.

3.2 DNS Dummy Server

The DNS Dummy server acts as a proxy, and redirects the queries to other components. To the outside it appears as a fully capable DNS Server responding to queries issued to it. By default it binds to *127.0.0.1*, but can be bound to any other address. For example binding to a network address such as *192.168.1.1* might be useful for a company wishing to take advantage of the Hybrid Distributed DNS while still being able to decide which entities to trust. For example a

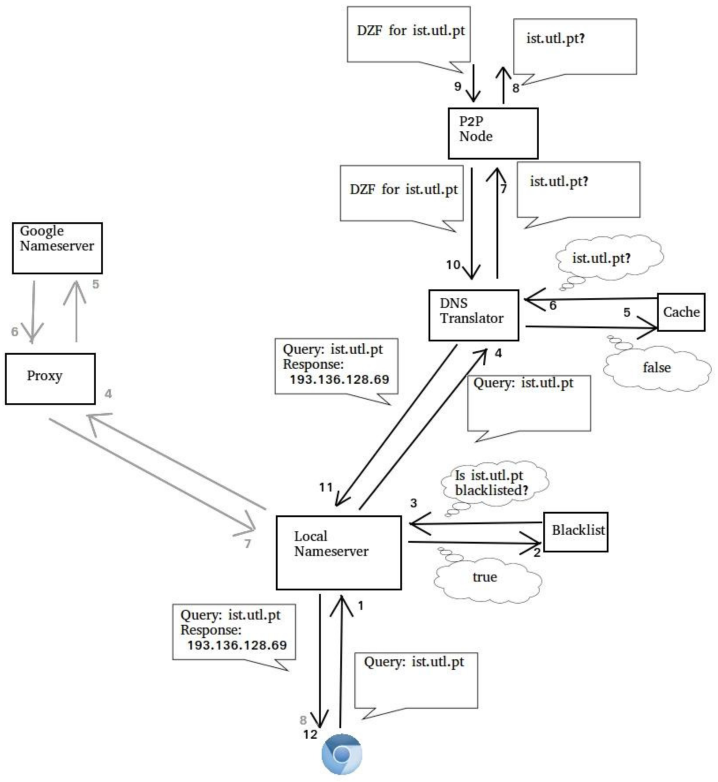


Figure 3.3: System flow of a query being routed. Black path is the path taken for a query belonging to a blacklisted domain, grey path is the path for a non blacklisted domain.

newspaper might set up the Distributed Hybrid DNS to circumvent censorship to some outside news source, while still maintaining control over the entities that it allows their employees to trust. This helps the company protect itself against any attacks from malign entities, which employees might have accepted without knowing. Received queries are sent in parallel both to a traditional nameserver and to the DNS Translator.

3.3 DNS Translator

The DNS Translator is in effect a translator from the P2P network to the traditional DNS. It receives a query from the DNS Dummy Server and asks for a trustworthy DZF that the DHT node has or will retrieve for the domain to which the query refers to. Upon receiving the the DZF, it will retrieve the appropriate record and build a valid DNS response to the query and send it to the DNS Dummy Server.

3.4 DHT Node

The DHT Node connects or is part of the P2P network where peers share the DZFs. The DHT Node is also responsible for the managing the public keys, and selecting the DZFs to send to the DNS Translator. Unless specified in the configuration, DZFs that are not signed by a recognized public key are never sent to the DNS Translator. The DHT used is based on kademlia, due to it's inherent replication mechanism and it's ability to send a query to more than one node, which speeds up the DNS record retrieval, making it a good option for our system. Fig. 3.4 shows the a query being routed to multiple nodes at the same time.

3.5 Daemon

The Daemon is used to administer the DHT Node and to receive suggestions for Public Keys when no valid Distributed Zone File is found. For administration purposes such as configuration, it has a GUI which allows the user to easily change the settings. The user can also administer the blacklist from there and install new public keys, as well as edit or remove existing keys. The other use for the daemon, and the reason for it being called a daemon, is that it connects to the System (DNS Dummy Server, DNS Translator and DHT Node) and waits for suggestions. These are suggestions of new public keys to add. They are formed by querying existing peers and asking which public keys they currently trust and from there match with our existing public keys.

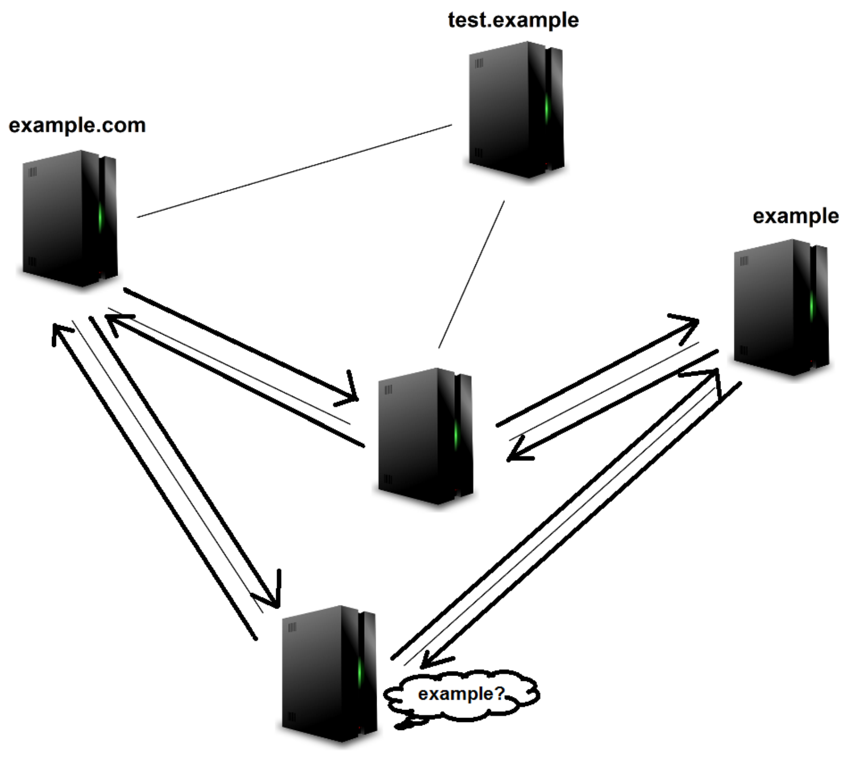


Figure 3.4: A query for DNS DRecord for domain example, with *alpha* = 2

If multiple peers trust a number of keys that we trust and all have a key that we don't, then that key will be suggested. A notification will pop up on the computer where the daemon is running and the user will be prompted to either accept or reject the key. For example, let's consider we require a match of three public keys between at least two users, user A, user B, and user C. A sends out a query to B and C asking them for the public keys they accept, A compares it's public keys against B's and C's, and finds 3 matches with B and 4 with C. This is above the minimum of 3, so A compares B and C's public keys, and finds a match for 1 key that it does not have. That key will be suggested to the user A using a pop up on the computer. If it is accepted A will now have 4 matches with B and 5 matches with C. The minimum users and the minimum key matches are defined in the configuration and a higher number will provide more security at the cost of less suggestions.

3.6 Domains

Domains are treated independently, aswell as subdomains as can be seen in Fig. 3.5. There is no enforced hierarchy between them. This means each domain and subdomain has it's own Distributed Zone File. For example *utl.pt* and *ist.utl.pt* have no connection when it comes to Distributed Zone Files and the owner of *utl.pt* may not be the owner of *ist.utl.pt*. There is only a hierarchy when managing the keys. This does however break cookies in browsers. The way cookies are currently implmented, does not work for this model, and as such would pose a security risk.

3.7 Public Keys

Public Keys are managed by the DHT Node, as previously mentioned. Adding new keys to the system can be done in two ways: one is by acquiring the public key directly from the entity that signs DZFs using the corresponding private key. The other by suggestion from the system based on users that trust the same entities. For the first case the user can simply browse the entity's website, using either traditional DNS or using an IP, and from there he downloads the Public Key and adds it to the DHT Node through a Graphical User Interface (GUI) interface available on the daemon. The other possibility is through suggestion, this happens when the user does not have a public key that will verify the domain use. When added to the system a key can be added for one domain only, or to that domain and all subdomains. Table 3.1 shows the internal representation of the key table. Extend means the key extends on to subdomains.

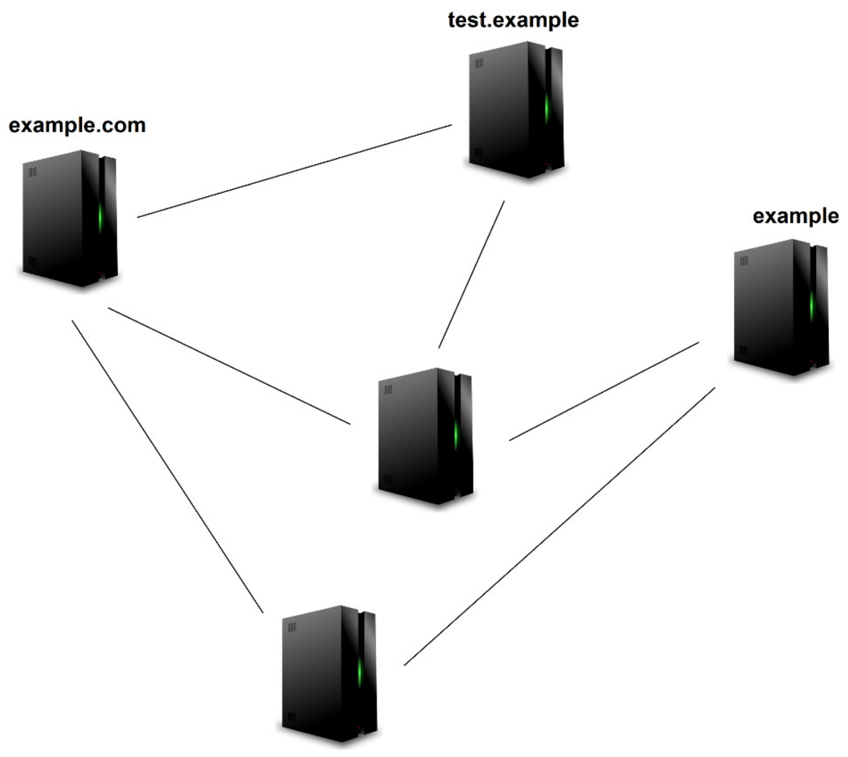


Figure 3.5: Domain Flattening

Table 3.1: Internal representation of the key table.

Domain	Key	Priority	extend
example	pk1	0	Yes
example	pk2	1	No
test.example	pk2	0	No
here.example	pk3	0	Yes

Table 3.2: Distributed Zone File Field Sizes

Field name	Field length (B)
Version	1
Domain	variable
TTL	8
Record A	5
Record AAAA	17
Record NS	variable
Record MX	variable
Record CNAME	variable
SHA-512 signature	64
Public key	remaining

Here pk1 has bigger priority than pk2 and pk1 can be used to verify subdomains. However pk2 appears as specific for test.example and as such pk1 cannot verify test.example, it can still verify, *.test.example pk3 is used to verify here.example and any other subdomains *.here.example, as such pk1 may not verify any *.here.example domains.

3.8 Distibuted Zone File

The Distributed Zone File, is a stream of bytes. This stream of bytes includes the domain to which the Distributed Zone File refers to, the Distributed Zone File version, the various records, a TTL, and a signature. The version is used to distinguish between diferent Distributed Zone Files, this is good if the signature strength needs to be updated or a new type of record needs to be added. This way, previous versions are still valid, however they should be update as soon as possible if the new version was added to strengthen the security of the signature. Each field in the Distributed Zone File has a byte which indicates the field we are reading, it is then followed by the data if it is a fixed length field, or by another byte which indicates the size of the variable length data, which may be up to 255 bytes.

3.9 Key distribution methods

A simple solution to distribute the public keys, would be to setup a site that distributes the software needed to be part of the network, along with the software would come installed a key from the entity that distributed the software. This key would be set as master root key, this means that any DNS DRecord signed by that key would be accepted by default. Of course the user can always change that behaviour, or even completly remove the key, but for these users we suggest always offering a clean version with no installed keys. Using delegation that entity could setup some kind of registration page where they would sign a certificate to delegate the domain. While this seems to give control to a certain entity losing the objective we were trying to go for, the reader must not forget that multiple entities can co-exist, and the user can choose which one to trust or build his own trust network using a package of the software with no keys installed. As the entity delegates the domains, these pass on to other entities, which will from now on be the only ones that can control the delegated domain. The entity may also choose to keep control of the domains acting like a normal registar in the current DNS. There are endless ways to administer domains and the keys, and we think that this freedom of choice is what sets this design apart from others.

3.10 Key Voting System Architecure

This system is part of the bigger architecture but is described here independently since it's functioning can be considered separate, but with access to the key database of the main system.

This system was developed as a way to enable a user to obtain new keys when he doesn't possess a valid key for a domain he tries to access. When this happens the suggestion system is triggered and if possible a valid key for that domain is suggested.

The user starts by voting on keys, either positively or negatively. These votes will then be used to correlate a user's interest with other users. Since our system allows for different keys for different domains, this means that different Distributed Zone File can be available, which might lead to different content for the same domain depending on the key the user accepts as valid. As such the user's votes are important to match the users interest and as such deduct if a new key might be suitable for the user.

Without votes from the users, the suggestion system is blind and cannot infer anything, as such suggestions will be weak or impossible depending on the user's settings.

When initiating a suggestion, the system asks all known peers for their votes. The votes are then matched against the user's own votes, giving each known peer a score. This score is used to rank peers in order of common interest. Each common vote either negative or positive has a value of one. There is a minimum score which by default is 5, however this value can be changed to whatever value the user feels is needed. A lower value will make results weaker, while also increasing the number of successful suggestion, however a higher value will make results more realiable, but will decrease the number of successful suggestions. A successful suggestion is one where the system is able to choose and key, retrieve it and present it to the user as a suggestion for installation.

After finding all peers with a score equal or more than the minimum threshold set, the system will find a key for the domain, which has a positive vote with at least 3 peers, this value can also be changed. It will not however stop with the first key it finds, and will prefer the key with most positive votes for the domain. The key will then be retrieved from the peer, and a suggestion will be made to the user, which can then opt to install the key or not.

The list of peers from which votes are requested is added by the user, along with a public key to verify the users votes. This means all users sign their votes with their private key before sending them upon request. Peers are added manually but no keys are exchanged and used to sign the votes. This design choice makes the number of suggestions smaller than if peers from the P2P network were randomly contacted, but ensures safety, and is explained in greater detail in the threat analysis section.

3.11 UI Architecure

The UI is a simple interface built using The Standard Widget Toolkit (SWT). It makes it easier to build interfaces while allowing for a number of features.

The UI presents the user with various options after connecting to the nameserver. These options include Managing the blacklist, the keys and voting on keys, check for system suggestion on keys and manage the peers used by the suggestion system, as well as an option to export it's own public key.

The main window which presents all the options as well the connection parameters, launches new windows for each of these options. There is a shared class which holds the connection to the nameserver and it is passed as a reference to each of the windows, so they can then retrieve the necessary information from the namesever to then display to the user.

All windows automatically close when the main window is closed.

3.12 DZF Creation and launching the server

In order to create a DZF there is a command line tool which allows the entity or user to create one or more DZFs for a domain. As explained before, since this a prototype only a limited number o resource records are implemented, the same happens with the DZF creation tool. This tool acts as a wizard, asking you through the whole process what the user wants to do next. Since the tool reads from the standard input this allows for an interactive wizard as well as file which contain the commands to automatically create a DZF allowing for automated creation of DZFs. This tool does not have or provide a user interface. Although this would make it easy for new comers to create their own DZF, most of the use will come from entities with their automated scripts maintain the DZFs updated. As such we did not consider making a GUI for this tool a priority. The command line wizard already provides enough assistance for most users to be able to create their own DZF.

In order to launch the nameserver all the user has to do is click on the provided jar which will automatically start the local nameserver along with all other modules that need to run, such as the P2P node, the blacklist manager and the key manager, the voting system, and the remote managment module which allows external connections to the name server in order to manage it all the external modules. The user interface however is provided as a separate component and is optional. An experienced user can manage the whole nameserver without ever touching the user interface. The user interface connects to nameserver through the remote connection module and as such if managing a local nameserver the user should enter localhost or 127.0.0.1 as the host address to connect to. When starting the nameserver if the user does not have a key pair which will be used to sign keys and votes a new one will be created and stored in a file called *identity.hddns*. Each time the nameserver is started it will check for that file, in the current directory if it cannot find the file then it will create a new key pair and save it under identity.hddns. An alternative path to the file can however be provided through the use of command line arguments when starting the nameserver. In order to provide command line arguments the user must launch the nameserver from the command line, various command line arguments are available and these allow the user to provide nameservers for the current DNS, provide the path for the file containing his key pair, as well as change the ports on which the nameserver listens to connections, as well as remote connections to manage the nameserver and provide the server with a bootstrap peer other than the default one.

Chapter 4

Implementation

4.1 Implementation Options

The objective is to get a fully working prototype of the HP2PDNS. As such speed of implementation is a must, and choosing a language the authors feel comfortable with will make the process much faster. Instead of spending time learning a new language, the authors can focus on implementing the features for the prototype. The author feels more comfortable using Java or C. Java in comparison to C also offers us more portability while sacrificing speed. Java offers us garbage collection, along with a range of libraries that facilitate networking. Looking at libraries already available to build the prototype, we find that Java already gives much of what we need with no changes or reimplementations needed. Our prototype uses TomP2P[1], a library which provides an implementation of Kademlia [19] enabling multiple values to be associated to a key (location key), using a second key, called content key. This is essential if we want to allow more than one key to sign a DZF for a specific domain. Kademlia allows for parallel asynchronous queries, enabling faster DZF lookups in the presence of slow or failed nodes, as the query finishes when the first reply is returned. We could not find a similar implementation in C, which supported multi values, as such any implementation in C would require us to either write a Kad implementation from scratch adding our needed support for multi values per key, or we would need to pick up an existing implementation, understand the code and then proceed to change it to support multi values, if possible.

Our DHT was modified to only accept DZFs which are valid, meaning the public key provided in the DZF can be used to verify the signature provided in DZF. Furthermore, any new generated

[1] `http://tomp2p.net/`, last accessed July 5th, 2013

DZF for the same domain with the same key, will only be valid if the key provided in the new DZF can be used to verify the signature of the old and new DZFs. If the key is changed, the DZF will be placed in a new content key as explained earlier in chapter 4.

We had to build a fully functioning server nameserver with a proxy relay to an existing nameserver. Our nameserver has to support the existing DNS protocol in order to be able to translate the P2P domain data, into regular DNS responses. For this we used dnsjava[1] which provides us with a DNS message parser, and an api to easily create or modify existing DNS messages. These libraries already available for use that require no modification for the purpose of our prototype, along with the reasons presented before, Java will provide us with a fast development phase possible.

For testing purposes we decided to use PlanetLab. PlanetLab is a group of computers available as a testbed for computer networking and distributed systems research. As such it provides us with servers to test our prototype.

4.2 Prototype

In order to validate our solution we decided to support only a limited number of resource records. We focused on A, AAAA, CNAME, and MX records. The first two are essential in order to use the DNS since they provide the IP address which the user should contact. The third we believe is important for redirections of domains, a common feature used, and the next is used for email, one of the oldest and most popular communications method on the Internet.

The prototype is developed in separate modules to allow different parts of the system to be easily replaced. The Nameserver class is where everything is coordinated, and all different parts of the system are created and launched there. The blacklist module, and key manager, are composed of a tree structure, to store the domain names, which are inserted. In the key manager along with the domain names, the keys are also stored in a list for each domain. This implementation is rather simple and allows to block certain domain names, or allow domain names under it. Other than the wildcard *, it does not support regex.

In order to speed up development so we can test our final system, the daemon was merged with the rest of the UI. When the UI is started so is daemon, which is not a deamon anymore but simply a thread which checks for any new suggestions in the system waiting for the user's decision and presents them to the user.

The UI is composed of five main windows. The first comes up when you launch the UI, it presents

[1] `http://www.xbill.org/dnsjava/`, last accessed July 5th, 2013

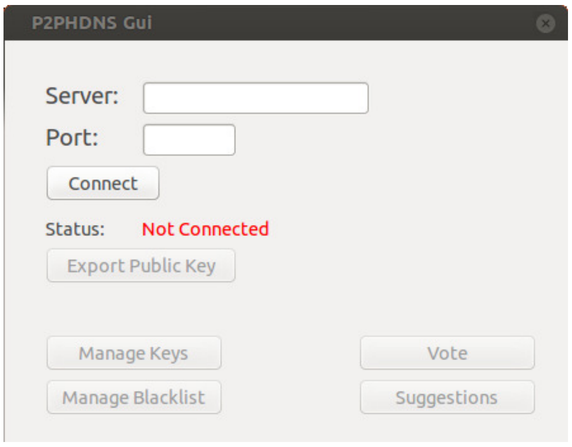


Figure 4.6: GUI Main window disconnected

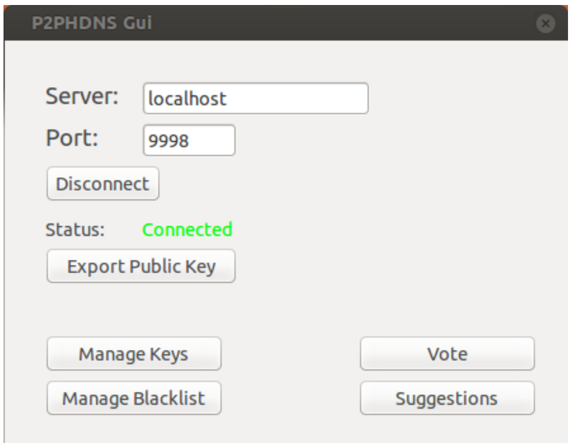


Figure 4.7: GUI Main window connected

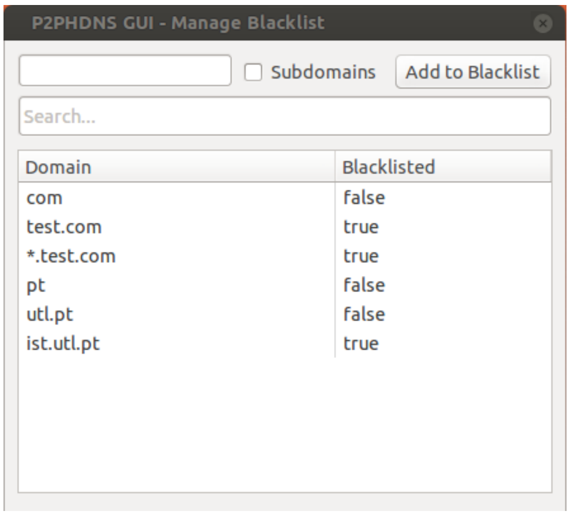
the user with two boxes to fill in the host and port where the nameserver listens for administration connections. It also displays the connection state. It also presents the user with 3 buttons to other 3 windows. A window to manage the keys, a window to manage the blacklist and a window to manage the votes. There is also an option to export his own public key.

When managing the blacklist a user can add and remove a domain from the blacklist.

When managing the keys a user can add and remove public keys for domains, he is presented with a dialog to indicate the file where the exported public key is stored.

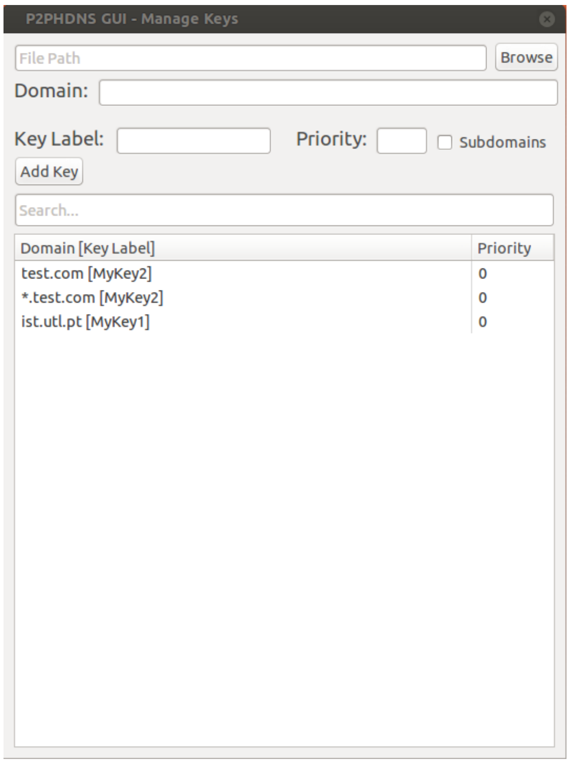
Voting is the same way the user is presented with the same list of keys and domains he has and clicking on them comes up with a dialog to vote positively or negatively for each key, as well as completely remove the vote.

Checking for suggestions can be done manually by clicking the suggestion button, and the system will also automatically make the window appear if there is a new suggestion the user has not yet



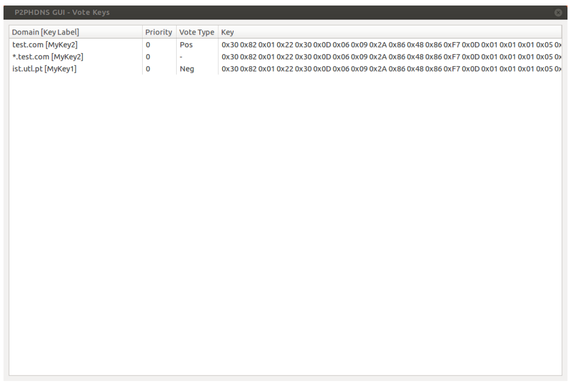
Domain	Blacklisted
com	false
test.com	true
*.test.com	true
pt	false
utl.pt	false
ist.utl.pt	true

Figure 4.8: GUI Manage Blacklist Window



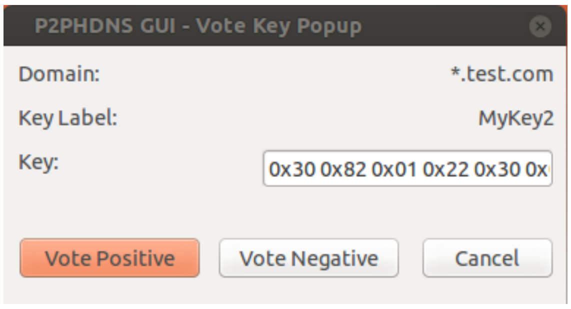
Domain [Key Label]	Priority
test.com [MyKey2]	0
*.test.com [MyKey2]	0
ist.utl.pt [MyKey1]	0

Figure 4.9: GUI Manage Keys Window



Domain [Key Label]	Priority	Vote Type	Key
test.com [MyKey2]	0	Pos	0x30 0x82 0x01 0x22 0x30 0x0D 0x06 0x09 0x2A 0x86 0x48 0x86 0xF7 0x0D 0x01 0x01 0x01 0x05 0x
*.test.com [MyKey2]	0	-	0x30 0x82 0x01 0x22 0x30 0x0D 0x06 0x09 0x2A 0x86 0x48 0x86 0xF7 0x0D 0x01 0x01 0x01 0x05 0x
ist.utl.pt [MyKey1]	0	Neg	0x30 0x82 0x01 0x22 0x30 0x0D 0x06 0x09 0x2A 0x86 0x48 0x86 0xF7 0x0D 0x01 0x01 0x01 0x05 0x

Figure 4.10: GUI Vote Keys Window



Domain: *.test.com
Key Label: MyKey2
Key: 0x30 0x82 0x01 0x22 0x30 0x

Figure 4.11: GUI Vote Keys Window

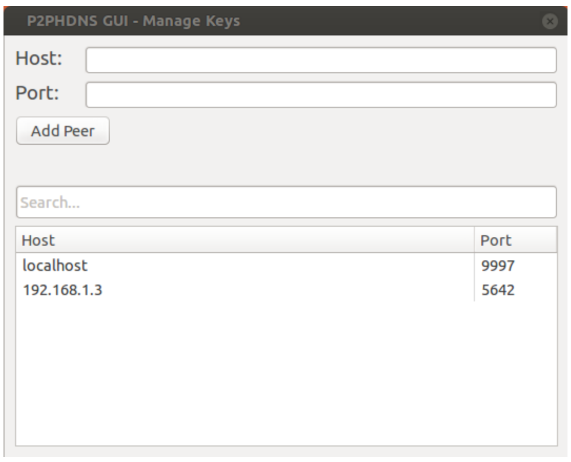
seen.

Peers belonging to the suggestion system can also be added and removed from the interface, by clicking the button manage peers in the Suggestion window.

For the implementation of the key suggestion system, the verification of the keys used to sign the votes, as well as the keys, was left out in the prototype since the test environment is known and secure and this does not affect the results.

4.3 Analysis

To the reader some design choices may seem like we are reinventing something already existent in the DNS. If you are currently in this state of mind, please be aware that our goal is to provide a decentralized DNS. With that said, we can not rely on existing infrastructure and technology to implement our decentralized service. In doing so we would be creating a centralization point which would defeat the whole purpose of the system. This is one of the reasons DNSSEC was not used in the system, as by doing so we would be placing a point of centralization and thus a point of control defeating the goal of the project. In the case of DNSSec this point of centralization



Host	Port
localhost	9997
192.168.1.3	5642

Figure 4.12: GUI Manage Peers - Voting System Window

would be the root key, without this key no record is valid. Although other properties such as better network resilience are still there, the censorship problem would not be solved. As such we avoid any existing solutions that centralize the system.

In doing that we also create another set of problems, such as phishing, or registered names takeovers. As with any non centralized system these problems are hard to solve. One way to prevent these problems is to make the user aware that by using such systems they need to be more aware of scams, and need to learn how to spot these. Giving the users the proper information goes a long way to remedy problems such as phishing. As such users should be made aware of the problems and how they can protect themselves.

Registered names, on the other hand are not solvable by informing the user. However in our system they rely on a popularity metric, meaning the entity that gets the most users to install they public key as the verifying key for that domain, will have more users using their domain, pointing to the IP address that entity desires. This itself is not realy an issue as the system allows for multiple distributed zone files per domain, each signed with a different key, meaning that if a group of users decide to use utl.pt as their domain name, the current users of utl.pt won't be affected by it. A user can install a key without even looking at it, but if this is his default behaviour then he shouldn't be using our system. The system is designed for users that want to escape a central control and have the freedom to control their domain names, and with more power comes more responsibility. As such the distributer of the system should warn new users about potential risks and the user should then make an educated decision if he/she has the knowledge and expertise to use the software the way it was designed.

Our system has a bigger learning curve for the average user, that uses it without any previous knowledge. However it provides the freedom and flexibility that our target audience wants while still providing them with the security needed, that with the proper measures on their part, reduces the risk to the same as with the current DNS.

The system provides a DNSSEC type of security with every DZF File being signed with a key, however there is no root key, that signs all other keys. Each entity which holds a DZF, will have its own key, completely independent from any other. Users are responsible for choosing which public keys they consider trustworthy.

4.4 Threat Analysis

When designing HP2PDNS, some attack vectors were identified and mitigated or solved. These ranged from threats related to user misinformation or mishandling, to P2P issues with DZFs, and finally with the Key Voting system which suggests new keys for users to install.

4.4.1 General Threats

One of the biggest hurdles we find, is user knowlegde. For starters our application is aimed at a more experienced and knowledgeable user, but even those have different degrees of understanding. The fact that users have to install the keys themselves will probably pose as a hurdle in the use of the application. In the context of the use of our application, it is imperative that the user be aware of the importance of the keys it installs to verify DZFs. A key from an malicious entity can expose the user to easy attacks such as phishing. In the context of domains, when one has control over the domain name, phishing is an extremely easy thing to pull off. Once the attacker has control over the domain name, all he has to do is put up a copy of the website and the user will think he is accessing the legitimate website, since the domain name will be correct.

Although making sure the user understand how keys work and that he is careful to choose which keys he installs is important, another big problem arrises. The key issuer, although proven to be good up until now, can at any point go bad and start abusing the trust it has gained from users. This issue is a bit more tricky to deal with as the user already trusts the key issuer based on previous experiences. In this situation the user or some third party has to notice this change of behaviour from the key issuer and take preventative measures to protect himself and other peers by warning them.

As well as the key issuer going rogue, a unique key can be compromised, in this case the key issuer itself is still trustworthy but the key it issued can not be relied upon anymore, in this case the key issuer should take measures to issue a new key and inform the users to update the keys. However our system does not currently provide mechanisms for easily updating the keys or informing users of such a change.

4.4.2 P2P Threats

For this we assume that most peers will oblige with the system rules, only a minority of the peers will try to circumvent the rules, in order to gain an advantage or simply disturb the network's operation in order to render the service partially or completely inoperable for some or all users.

In order to secure the DZFs, all DZFs are signed with a private key, the public key is included at the end of the DZF. This allows any node to verify that the DZF is correctly formed, and signed. This prevents any corrupt data from entering the network. Each P2P node will perform this verification before accepting a DZF, even when performing replication.

The location key used in the DHT, is the domain name. However, since we allow multiple values per location key, a content key is used to distinguish between the different DZFs for the same domain. The purpose of allowing multiple DZFs for the same domain is to allow a healthy competition between entities providing the DZFs. If we only had one entity that could sign the DZF per domain, that would easily allow for censorship, should that entity decide to go rogue.

To prevent removals and substitution of previously inserted DZFs into the network, upon receiving a DZF each node checks if the location key and content key chosen already have a DZF. If that is the case then the node checks if the public keys are the same and verifies both signatures, this prevents anyone, other than the owner of the key pair that signed the first introduced DZF in that specific location and content key pair, from removing or substituting it.

But there was still the problem of who occupies the location and content key pair first. Let's say for example the location and content key pair had not been used yet, and some user A who knew the public key of entity B decided to occupy that location and content key pair. When entity B decided to start distributing a DZF for that location key, it would not be able to, since the DZF that had been inserted previously was signed with a different key. The node performing the check to see if the public keys matched, would refuse the new DZF. As such a new check was put into place, since the content keys used are a SHA1 hash of the public key, then when a DZF is inserted the hash of the public key contained with the DZF is computed and checked to see if it matches the content key where the DZF is going to be inserted. If it does not then insertion fails preventing the attack described above from happening.

Another attack identified was a replay attack. Since only the validity of the DZF's are checked, then anyone with a valid DZF can insert it into the network. This by itself is no security flaw as this should be possible, otherwise replication would not work unless done by the node run by the user that owns the private public key pair used to sign and verify the DZF. However, if a malicious user possesses an old DZF, it can try to insert it into the network to substitute the current one. To prevent this attack a serial version was introduced to prevent old DZF's from being re-introduced. Whenever a DZF is inserted, if another already exists for the same domain and both public keys match, then the serial versions are compared and the serial version of the new DZF must be bigger than the serial version of the current DZF. If no DZF exists yet, the serial version is ignored.

4.4.3 Voting System Threats

A voting system especially in an open sourced software, has many issues, especially concerning the veracity of the users votes. The first issue, that can be found in any system, how do we stop a user from creating multiple identities?

This is not an easy task, in closed source systems, a challenge when creating the identity can be put in place, for example a captcha or simple logic question, *If one elephant is put in a cage with a zebra, how many animals are in the cage?* Challenge questions like this stop automated scripts from creating multiple identities. They do not however stop a user from manually creating them, but the time needed to create the number of identities needed to tamper with the system is so big, that most attackers won't even attempt it.

Since our voting system is based on comparing votes, another identified threat is vote copying. A user requests votes from it's peers, and simply copies their votes, this means that next time those peers request their votes, they will have a higher score and will be chosen over other peers to suggest a key. One way to deter this attack is to have a minimum number of peers and choose a common key between them, this makes it harder for the attacker. However if creating multiple identities, along with copying the votes the attacker could easily suggest a bad key to the user.

Another issue is votes can be tampered with if not signed. How do we know if a man in the middle attack was not performed and the votes changed, to benefit a bad key?

For these reasons we choose to limit the peers contacted, to those added manually by the user, along with public keys to verify the authenticity of the votes. Since the user manually adds the peers there is a certain level of trust in the peers contacted, and with the votes being signed by a private key and verified by a public key the user added along with the peer, the votes cannot be tampered with. Some of the attacks mentioned before are still possible if the user adds a significant number of malignant peers, since votes can still be copied. As such, the user should only add peers in which it trusts.

All these limitations have an impact on the usability of the system and the results it is able to produce. The usability becomes limited since users have to manually add other peers in whom they trust to read votes from. They have to manage these peers and their respective public keys. These keys are managed separately from keys used to verify DZFs.

The results produced by the system are limited to votes that the system is able to gather from manually added peers. Meaning that the system is not able to automatically search for more peers and compare more votes in other to choose a key to suggest. Without the user adding the peer. This means that either the user adds a lot of peers and their keys to the system , thus having to do a lot of work, in finding the peers and making sure he trusts them.

4.5 System

4.5.1 Configuration

The system allows a number of configuration settings to be defined through command line arguments, when starting the application. These settings include changing the default nameservers, the port on which dns nameserver listens, as well as the port to listen for connections for remote administration and the ip and port to bootstrap to the P2P network.

The kademlia replication parameter can is fixed and does not allow for a change by the user, this is so in theory most users will use the same replication parameter when connecting to the main network. Although they are free to change the replication parameters and even create their own separate P2P dns networks. However the ideal scenario involves one big network where all users follow the same rules.

After the system launches it can be managed through commands entered using the command line or using the GUI developed for that purpose.

4.5.2 Classes

Fig. 4.13 provides a view of the most important classes in the system, leaving out helper classes which are not relevant to explain how the system fits together, these are classes, such as utilities classes, or classes used to pass data around between different modules of the system.

As can be seen the application starts in the class Server. This class only provides a main function and instantiates the class Nameserver, passing it the command line arguments. The class nameserver set's up the configuration based on the supplied command line arguments and instantiates

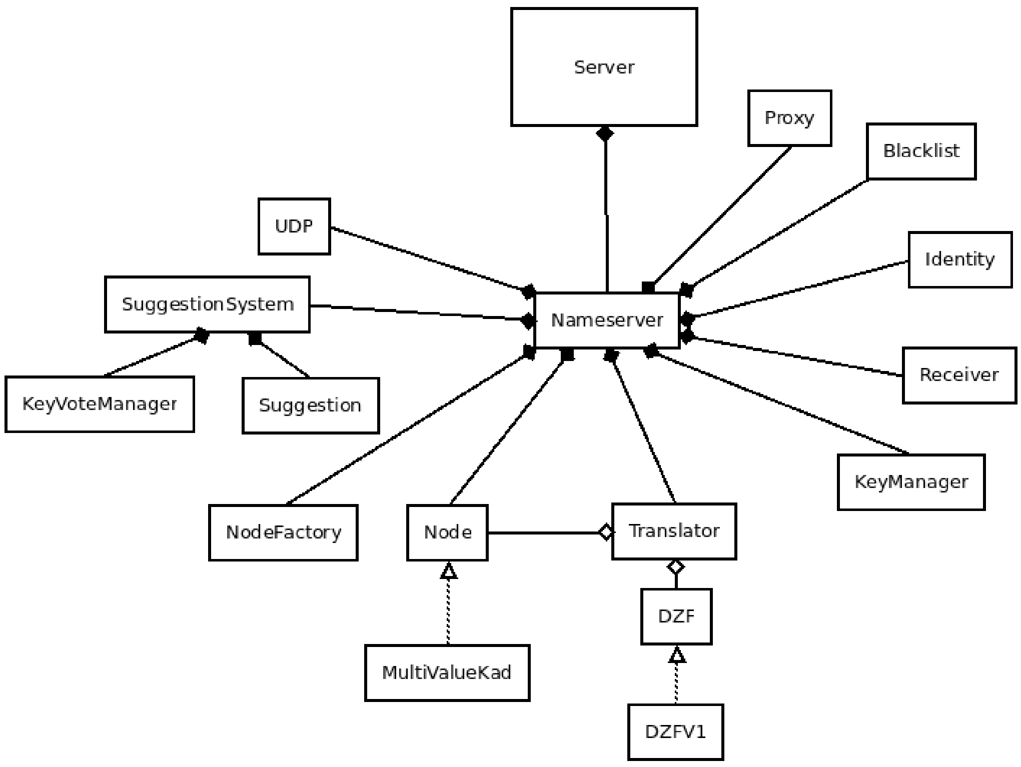


Figure 4.13: Overview of main classes

all necessary modules for the system to function. The Nameserver is aware of the whole system and everything revolves around this class.

The Identity class is responsible for loading a users public/private key from a file. If a file is not present of supplied, a new key pair is created and saved.

The class UDP is listener that is used to receive dns queries. These queries are then sent back to the Nameserver. The Nameserver then checks if the domain is in the Blacklist and based on that will pass the query on to the Translator or the Proxy. The Proxy class forwards this to a current DNS nameserver and waits for a response and hands it back to the Nameserver. The Translator class requests requests the Node for the DZF corresponding to the domain in the query.

Node is an interface, and MultiValueKad is an implementation of that Interface. An instance of Node is created by calling a static method in the class NodeFactory and giving it a string representation of the node implementation you want. This allows for a great flexibility when changing the underlying library which is responsible for the P2P network.

Upon receiving DZFs from the Node, the Translator calls upon the an instance of DZF to parse the DZF and provide a format it can read easily and build a DNS response out of it. This is an internal representation of the records and is only used during the make and parse phases. The

Translator then selects the DZF which is signed with a key, that was inserted in KeyManager. It will start looking for a DZF signed with the key that has the highest priority. Once this check is done, and the DZF is correctly signed the Translator then hands the response back to the nameserver, which passes it to the UDP class, which answers the DNS query.

The DZFV1 class is an implementation of the DZF interface, this follows a similar approach to the one used with the Node interface, except this time the Translator creates a specific version of the DZF depending on the version of the dzf it receives. This allows for various version parsers to co-exist and as such older version of a DZF will not be rendered useless when a new version is introduced.

The SuggestionSystem includes two subclasses and is called each time the system does not find a key for a domain which was supposed to be resolved using the P2P network. The KeyVoteManager class includes a user's votes and regarding the pair domain key and the Suggestion class is a thread which requests peers votes and tries to find a key to suggest. If it is able to find a key it stores the key in the system but does not add it to the KeyManager until the user accepts the suggestion. The user can reject or accept the suggestion.

Chapter 5

Evaluation

5.1 Tests Objectives

Our objective is to prove that it is possible to build application to allow users to circuvemt censorship, without any noticable difference fater the initial setup, from what they are used. In regards to the P2P network our main tests fall on time taken to answer the queries and the availability of the data, when nodes start to fail. This is relevant as the time taken to answer queries directly relates to a user's expected Quality of Service (QoS), when using the application. The availability is also related to the a user's expetected QoS and since we are dealing with a free P2P network where users join and exit as they please, the availability of the data may not always be 100%.

The application also supports the use of the current DNS and uses a proxy to relay queries to existing nameservers, as such the time taken to answer those queries should be measured and compared, against queries done to those nameservers without using the proxy.

Since the system is supposed to support multiple users querying, we launched multiple clients and measured their response times, when querying and compared it with the results obtain from our first query to the HP2PDNS.

The suggestion system is deterministic and to prove the correctness of the suggestion system we ran a number of tests, and compared the suggested key to the expected suggested key.

5.2 Tests Scenarios

Our prototype was used to evaluate HP2PDNS using a deployment of 100 nodes: a personal computer and 99 PlanetLab[1] nodes. Our experiments were conducted to verify the correct operation of HP2PDNS and measure its performance, in order to assess its suitability for daily use. The personal computer was located in Portugal, connected to the Internet with 100Mbps/8Mbps download/upload rates, a Core i5-750 and 16 DDR3 Ram. Our node also served as the nameserver and it was the node that was queried. Our experiments were conducted to verify the correct operation of HP2PDNS and measure its performance, in order to assess its suitability for daily use.

To test our prototype we downloaded the list of the 1,000,000 most visited domains on June 17th, 2013, supplied by *Alexa.com*[2] and used the first 1,000. We then proceeded to generate 1,000 DZFs, one for each domain, and added them to the network. When using our nameserver and a queried domain is not blacklisted, our nameserver serves as a proxy and as such it needs to be supplied with an existing nameserver to the current DNS, for all ours tests the supplied nameserver is Google's nameserver reachable at IP address *8.8.8.8*. All our tests were done with the cache mechanism off since the cache is local to each client and as such this would alter the results that we aim to show which is when no caching has yet been done.

All queries were performed using a dig clone written in java, supplied by dnsjava [3].

For all tests a node from planet lab was started and used as the bootstrap node, all other planet lab nodes proceed to join the network. Finally our local node joined the network and added the 1000 DZFs to the network.

For our first test the nameserver was queried 1000 times without any domains in the blacklist, this means that all queries were routed to a supplied nameserver. We then added all domains to the blacklist and queried the nameserver again, this time all queries were routed to the P2P network, the corresponding DZFs fetched and converted to regular DNS responses. The same queries were done using the Google nameserver directly without using our nameserver.

The next set of tests done were to test the network resilience in case of nodes failing. We started off with 100 nodes, queried the nameserver, after the queries were done, 10 nodes were killed, and the nameserver was queried again. This was repeated until there were only 10 nodes left.

To test if the network can handle multiple queries we repeated part of the first test this time with 4 clients. The network was setup and 4 clients, in 4 different locations queried the P2P network.

[1] `http://www.planet-lab.eu/`, last accessed July 5th, 2013
[2] `http://www.alexa.com/topsites`, last accessed July 5th, 2013
[3] `http://www.xbill.org/dnsjava/`, last accessed July 5th, 2013

In order to also test how the local nameserver can handle multiple queries, we executed multiple queries on the local node. The network was setup the same as in the first test but rather than querying each domain one after the other, domains queries were done in parallel. We started with one query at at time and then doubled, the number of parallel queries, for each run. So the domains were queried in parallel, first with two queries, then 4 queries, each run doubling the number of queries, until we reached 256 queries in parallel all from the same one node.

5.3 Test Results

Our first test intended to measure the delay introduced by the use of HP2PDNS. As name resolution is, most of the time, the first step performed when accessing an Internet service, the time it takes has a direct impact on the performance perceived by the user.

Figure 5.14 shows the time it took to retrieve the records, using the P2P network, meaning all domains were added to the blacklist, using our system as a proxy with Google's nameserver *8.8.8.8* and using Google's nameserver directly. Our tests showed us what we had seen from other studies: that the response time tends to be greater when using a P2P network. The use of the DHT increased the reply times by an order of magnitude. However, we believe that all times presented in the results are still acceptable when, the alternative is to not access the censored domain.

Using our system as a normal DNS, meaning no domains are blacklisted and all DNS queries are routed to a previously selected nameserver, in this case Google's nameserver, incurs no noticeable overhead. This was to be expected as it simply reroutes the queries and does no work other than checking if the domain is in the blacklist. The blacklist is setup as a tree, with the domain depth being the depth of the domain in the tree, meaning *utl.pt* has a depth of 2, as such the lookup on the blacklist is always in the worst case n, where n is the depth of the domain. However, using the system without using the P2P network to bypass censorship provides no real advantage. To bypass censorship the user needs to add domains to the blacklist and these will then be retrieved from the P2P network. We can see from Figure 5.14 that the times to retrieve these records from the P2P network are much higher (a tenfold increase) when compared to retrieving the records from current DNS. However, they all come in under 6s, with 90% coming in under 2s, which is quite acceptable. We believe that these times can still be improved upon and using a local cache can go a long way to improve those times, since the retrieval of the DZF from the network is the biggest overhead of the system.

P2P networks made up of voluntary nodes exhibit churn: nodes enter and leave the network at

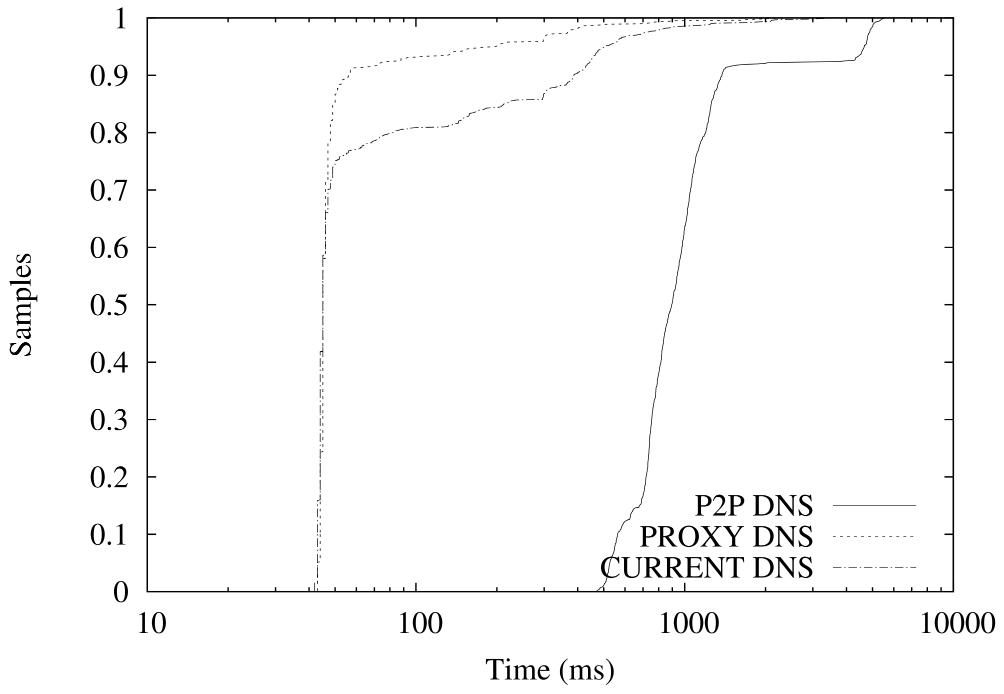


Figure 5.14: DNS Query Response Times

their convenience. Nodes can also fail or lose connectivity. The sudden departure of a large number of nodes will impact the performance of the system and may cause data to be lost.

Figure 5.15 which is a Cumulative Distribution Function (CDF) demonstrates the system's resilience, using a replication factor of 6, meaning every DZF has 6 copies on the network. The node responsible for the DZF is responsible for replicating the DZF and checking to see if there are enough replicas. We started with 100 servers and stopped 10 each time and measured the success rate and average reply time in answering queries. It must be emphasized that we did not allow the nodes to exit orderly but killed the process, preventing other nodes from being notified of the departure. This simulates the worst case: a node failure. As can be seen, the first data loses were only experienced when 40% of the nodes had failed, and the success rate never went below 60%, meaning that even when only 10% of the nodes were still running, 600 of the 1000 DZFs that were inserted into the network were still available.

We can observe that the average response time tends to increase as the number of working nodes decreases, but there are some improvements in the response time. These occur when the replication process kicks in, detects the drop in the number of replicas and creates new ones, allowing Kademlia to use its parallel search feature to benefit from faster replies from closer nodes. These performance improvements also coincide with a pause in the success rate drop. The new replicas introduced by the replication process enables the DHT to withstand the node

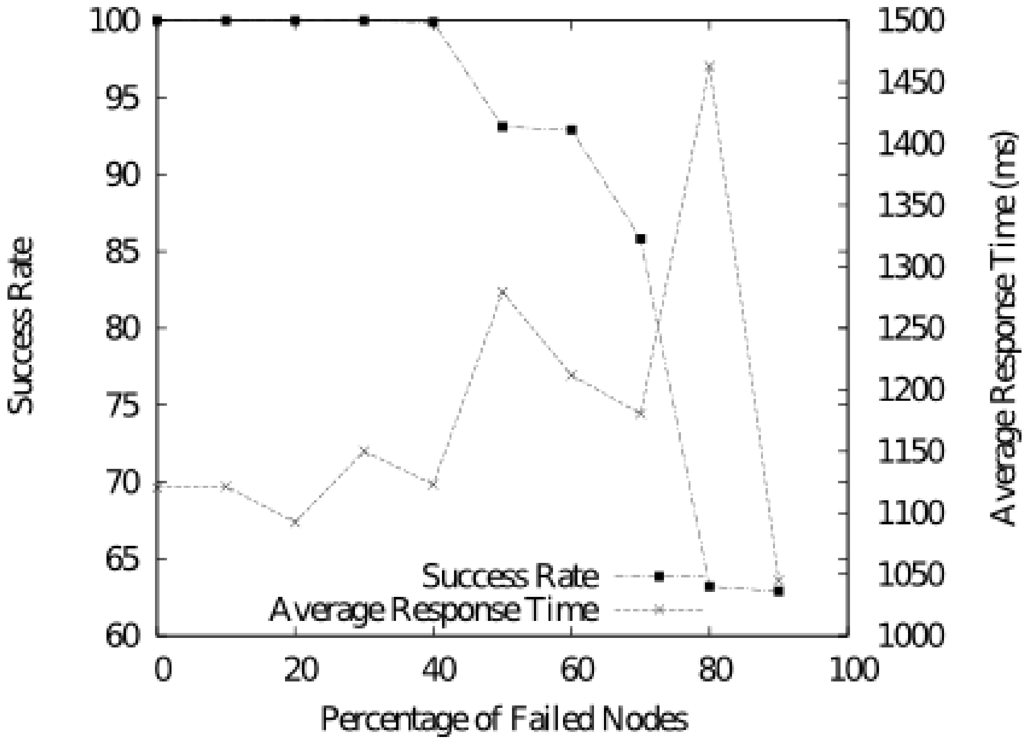


Figure 5.15: Resilience Test with replication factor of 6

failures without loss of data. This experiment shows that even with 40% of the nodes leave the network in a short period, no consequences other than a temporary increase in the reply time occur. By tuning the replication factor and the interval between runs of the replication process, it should be possible to accommodate highly dynamic networks without data loss.

The same test was performed, starting with 100 nodes but this time 50 nodes were killed at once. Afterwards, the query success rate has 93%, meaning that the replicas were reasonably well spread out.

We repeated this test case but this time with no replication. Figure 5.16 demonstrates this. It can be noted that at the begging the results are identical, this is due to the distribution of the DZFs throughout the network, however we can see where the fact that no replication was used, begins to have a big impact on the network. When 80 nodes had failed, the success rate went under 60%, a value that it did not go below when replication was active, and when 90 nodes had failed the success rate was of only 37.2.

When testing out if our system is capable of handling more than one client, we queried at the same time with four clients. Figure 5.17 and figure 5.18 demonstrate the time it took for each of the four clients to retrieve the same records. Both figures refer to two distintic runs. As can be seen the results show that all clients had average times equal to if only one client was querying

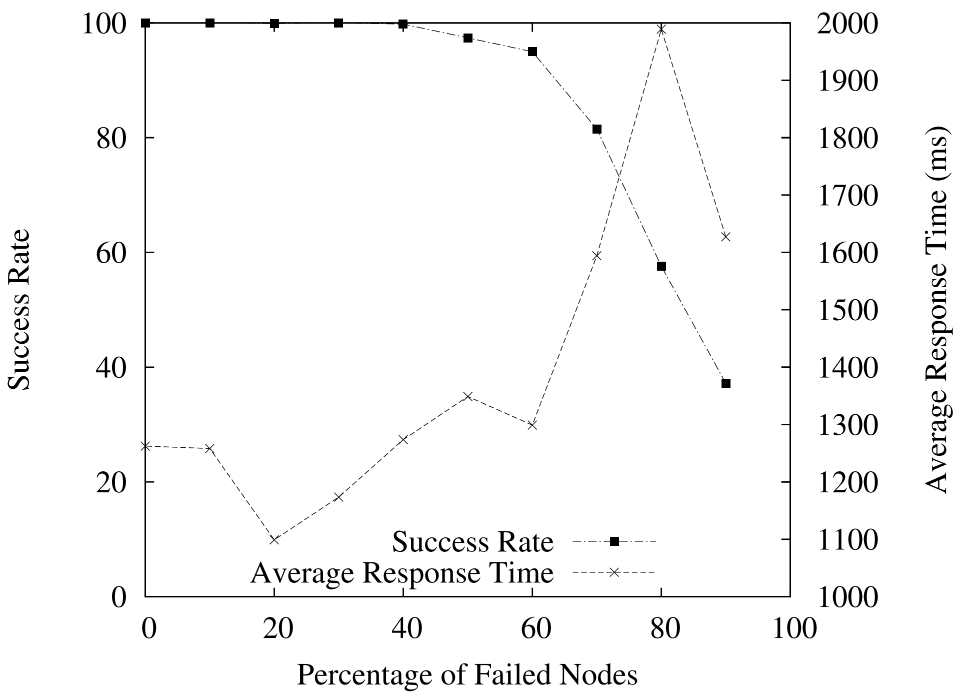


Figure 5.16: Resilience Test with no replication

the network. Between the two run the clients were consist meaning that the differences in the minimum latency that can be seen on the graphs are due to their physical location in relation to the nodes were the DZFs retrieved were stored and the network load on the node at the time. This means that the network is capable of handling multiple queries from different nodes at the same time.

Other than multiple queries to the network from different clients we also tested how the name-server itself handles multiple queries, this means answering multiple queries in parallel and sending back the responses. The average time for each of the parallel runs can be seen in figure 5.19. In this figure a sucessful query had to be answered within ten second timeout. As we can see there is an inscrease in the average time as the number of parallel queries grows. As the number of parallel query grows, the number of unanswered queries also grew.

In figure 5.20 the same test was ran this time with a timeout of ninety seconds, meaning that a successful query had to be answered with in ninety seconds. As we can see the number of failures went down. However as the number of parallel queries grew there were some unanswered queries.

The Voting system is deterministic as such we know which key should be choosen by the system as a suggestion to the user. As such the tests done were test cases with 2 or more clients, where keys where inserted into the system, votes were added and then a suggestion was made

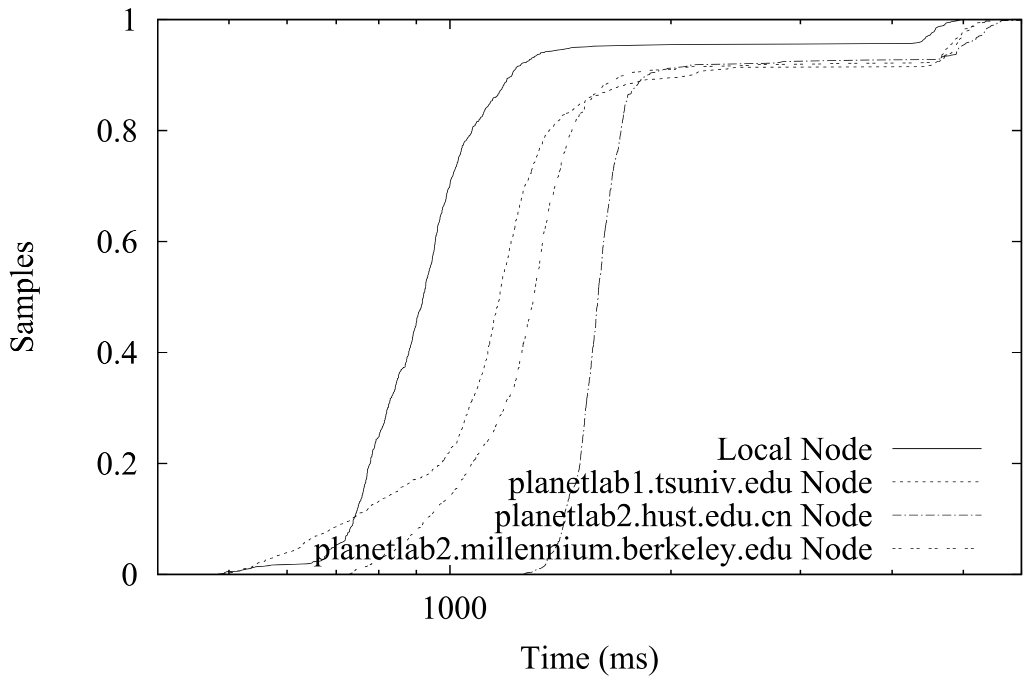


Figure 5.17: Querying the network, 4 clients at the same time

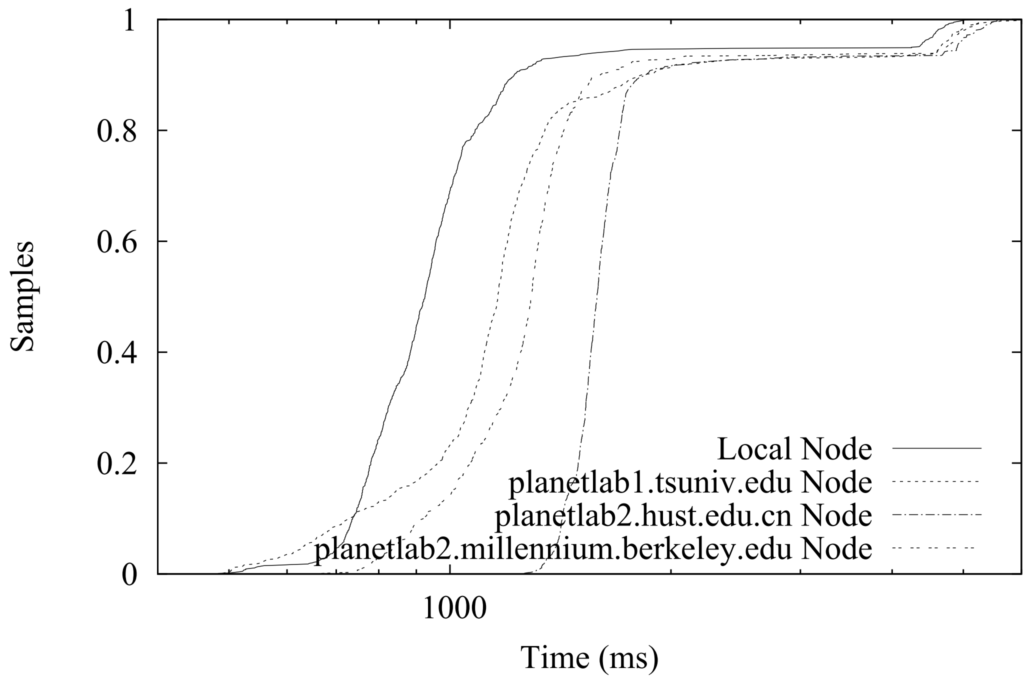


Figure 5.18: Querying the network, 4 clients at the same time

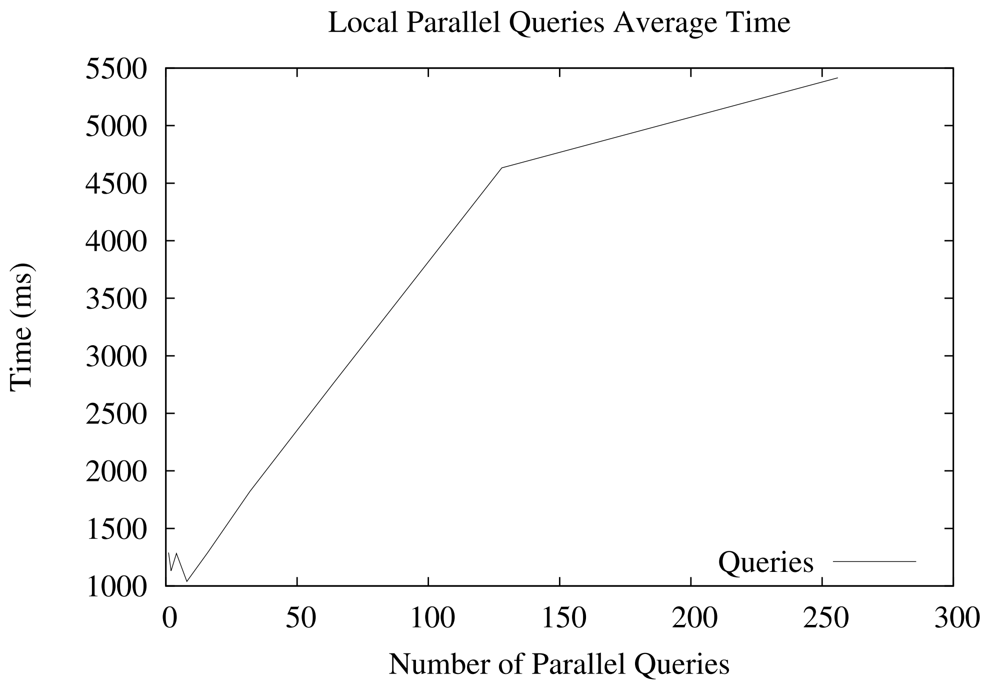


Figure 5.19: Multiple Queries from the same node (10s Timeout)

Table 5.3: Number of failed responses with multiple Queries from the same node (10s Timeout)

Number of parallel queries	**Number of unanswered queries**
1	0
2	0
4	0
8	0
16	13
32	38
64	292
128	321
256	647

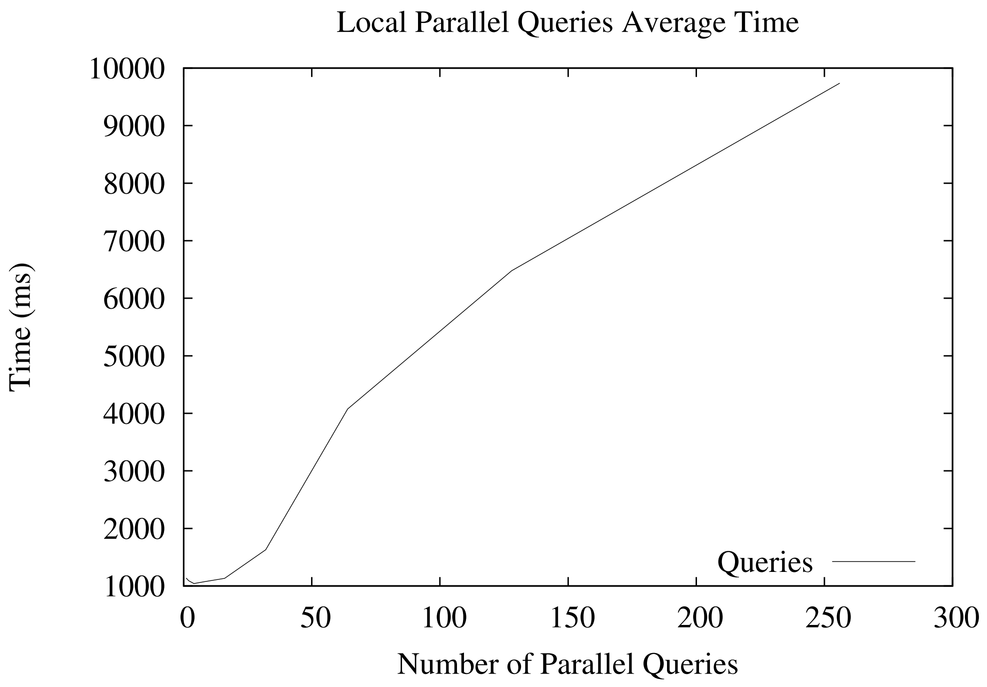


Figure 5.20: Multiple Queries from the same node (90s Timeout)

Table 5.4: Number of failed responses with multiple Queries from the same node (90s Timeout)

Number of parallel queries	**Number of unanswered queries**
1	0
2	0
4	0
8	0
16	5
32	4
64	17
128	26
256	24

by the system, to one of the clients. The suggested key was compared to the expected outcome. Although our implementation choices for the voting system prevent some attacks, most of these attacks not happening are dependent on the fact that the users added by the clients are considered trustworthy.

The minimum score needed for a peer, is set by the client. Each score point is equal to one key match, meaning both the client and the peer have the same vote for the pair domain and key. Votes are public since a client can request votes from any peer. Other than the minimum score there is a minimum number of peers that must have a score equal or higher to the minimum set, and the same vote for the pair domain and key. As such, in order to attack the user an attacker will have to create as many peers as the minimum needed by the client, and copy the client's votes to obtain the maximum score possible. However, the client's minimum number of peers needed with a minimum score is not known. After creating all the peers the attacker would have to get the client to add all these peers to the suggestion system, along with their respective public keys to verify their votes, which in this case would be a copy of the clients votes in order to obtain the maximum possible score. So this would always be a guessing game, as long as the clients change the default number of minimum peers, which is three. To make this happen, we suggest forcing the client to choose the minimum number of peers needed for the suggestion to be made, which would make the attack harder, as the attacker would have to guess how many peers he has to create and trick the user into adding them to the voting system.

To test the UI, we asked two colleagues, which study computer science as well and as such are considered experienced users. Experienced users are the target audience for our application, users who are acostumed to manage their own software and configurations and as such are the most probable audience for an application who main purpose is to change the system configuration in order to allow access to censored domains. If the user understands the concept of a domain and how it is censored, he is more likely able to use our software. As such we handed them the UI and allowed them to setup a configuration of their own, allowing them to blacklist domains and install keys to verify DZFs for certain domains. Some of the feedback given by these two users was taken in to account and changed in order to make the UI more user friendly for the final version, however their answers reflect their first contact with the UI.

Looking at user feedback we can see that when managing the keys one user clearly understood better how to user the interface. The user which classified the interface to manage keys as hard to user, giving it a 2, pointed out that the field for the file path where the public key is stored, did not have anything in it, and although there was a browse button next to it he was unsure of what to put in that field, until he used the browse button. This has already been fixed in the final release. Both users point that the interface is not very clean, and it has some cluttering of the buttons and

Table 5.5: User Feedback from the UI - Answers on a scale of 1-5 (1 Lowest, 5 Highest)

Questions	User 1	User 2
Ease of setup?	4	4
Interface cleanness?	3	5
Management of Keys?	2	5
Managing the blacklist?	4	4
Export Public key?	4	5
Overall ease of use?	3	5

fields. The setup was relatively easy for both users, although they did have to run execute a jar file, being that both are experienced users, they found that given their knowledge the setup was easy, however a simple click and run would have been even easier. Both users found the blacklist easy to manage, and pointed out that the search function was extremely helpful especially when the number of blacklisted domains becomes very big. However the low rating of the interface cleaness was what probably lead them to give it a four instead of a five. As conclusion and overall ease of use of the interface, the user that got confused in the management of keys gave it a three meaning not easy, but neither hard. The other user however gave it a five, which he mentioned was due to the fact that he understood what to do with the interface, while using and did not really find it hard to use. Since the main source of confusion for one user has been fixed, the overall interface is relatively easy to use. However a more thorough test with a wider range of users should be used to get a bigger prespective of what the users expect to see and what they understand when looking at the interface. A clear point is that although the UI serves it's purpose, it should be redesigned to make the look more clean, simple and uncluttered.

Chapter 6

Conclusion

6.1 Summary

Although P2P DNSs have been evaluated and studied, no prototype for a censorship free system has been built. At the time those studies were done, censorship was not a hot topic and as such the main concern was addressing possible attacks with the current DNS. Our main goal, was to take these studies and build a prototype for a censorship free system. However, we also decided to evaluate a different type of P2P network, based on kadmelia. Since our system is based on a P2P network it inherits the safety issues outlined in other studies in regards to DNS. We took the existing DNS and built our system on top of it. The system provides the user with a way to choose which network to use to resolve the domain names. This gives the user the speed of a centralized DNS while allowing, the user to resolve censored domains through the HP2PDNS. It allows for multiple entries per domain, with the use of public keys to differentiate DZFs for the same domain. A user will only accept a DZF for a certain domain if he has explicitly added the public key which verifies the DZFs contents. Creation of DZFs is not limited to one or limited amount of entities before hand, as such any user can have a domain free of any costs, other than those associated with running a node, and even then a user is not forced to run a node in order to be able to insert the DZF. As mentioned the system makes use of public keys to verify DZFs, this means the DZF was signed with the corresponding private key. This public private key pair is created by the entity that signs and creates the DZF. This integration of various DZFs for the same domain is smealess and does not affect normal usage, the user can even accept more than one DZF by defining a priority on the keys. Priorities define, which key a user trusts more, meaning that a DZFs signed by a key with the highest priority will always be used, unless it is not available. Only then will the system use the DZF signed by a key with the second highest priority.

In order to facilitate the distribution of keys a recomendation system was created, which although limited, allows for a faster spread of keys once the user has built his trust network, by adding peers whose votes he trusts. This recommendation system is made possible by votes from the users, in regards to the pair key domain. Meaning that a user will vote either positevly or negatively for a key in regards to a domain. Thus the user is effectively voting for the validity of the contents in the DZF.

Although HP2PDNS takes more time to resolve domains, we believe the amount of time taken is still acceptable when the alternative is to have censored domain not being resolved. This also happens only the first time the domain is resolved, since the local cache will then resolve popular domains much more quickly.

The prototype designed proves that it is possible to build a censor free DNS while also allowing the user to keep using the current system. The results show that although speed of resolution is much lower than the current system, it is by no means a prohibitive time, which would make usage impossible.

6.2 Future Work

Although our prototype provides a good foundation for a HP2PDNS, there is still more work to be done. Having the user install a certificate which will have the public key, instead of installing the key, this means that certificate could be installed in the system, and the key contained in it could be used to both initiate connections over HyperText Transfer Protocol Secure (HTTPS) and verify DZFs. This means extending the use of the public keys so they can be used in the user's web browsers to initiate HTTPS connections.

Our recommendation system is now very limited, further research should go in to providing a way for users to be able to search for keys out of their trust network while still maintaining a minimum level of vote integrity which we can assume to be safe in most cases.

Looking for better ways to improve latency is also a priority, as the system stands it is usable but still has a latency problem which might make less patient users not want to use the system.

As reported by the user feedback and althought some of the feedback was immedialtely taken into account, the user interface could use a redesign in order to make it more user friendly.

The system as a whole is geared towards tech savy users and that issue should be addressed as well, since a censorship tool should be available to all independently of their handle of technology and their ability to use advanced products.

Bibliography

[1] Harrenstien, K., Stahl, M., Feinler, E.: DoD Internet host table specification. RFC 952 (October 1985)

[2] Mockapetris, P.: Domain names - concepts and facilities. RFC 1034 (November 1987)

[3] Mockapetris, P.: Domain names - implementation and specification. RFC 1035 (November 1987)

[4] Postel, J.: Domain Name System Structure and Delegation. RFC 1591 (March 1994)

[5] Mockapetris, P., Dunlap, K.J.: Development of the domain name system. SIGCOMM Comput. Commun. Rev. **18**(4) (August 1988) 123–133

[6] Cox, R., Muthitacharoen, A., Morris, R.T.: Serving dns using a peer-to-peer lookup service (2002)

[7] Jung, J., Sit, E., Balakrishnan, H., Morris, R.: Dns performance and the effectiveness of caching. IEEE/ACM Trans. Netw. **10**(5) (October 2002) 589–603

[8] Albitz, P., Liu, C.: Dns and bind (1998)

[9] Gutierrez, C., Krishnan, R., Sundaram, R., Zhou, F.: Hard-dns: Highly-available redundantly-distributed dns (2008)

[10] Abu-Amara, M., Azzedin, F., Abdulhameed, F.A., Mahmoud, A., Sqalli, M.H.: Dynamic peer-to-peer (p2p) solution to counter malicious higher domain name system (dns) nameservers (2011)

[11] Song, Y., Koyanagi, K.: Study on a hybrid p2p based dns (2011)

[12] Ramasubramanian, V., Sirer, E.G.: The design and implementation of a next generation name service for the internet (2004)

[13] Arends, R., Austein, R., Larson, M., Massey, D., Rose, S.: DNS Security Introduction and Requirements. RFC 4033 (March 2005)

[14] Cachin, C., Samar, A.: Secure distributed dns (2008)

[15] Arends, R., Austein, R., Larson, M., Massey, D., Rose, S.: Protocol Modifications for the DNS Security Extensions. RFC 4035 (March 2005)

[16] Cachin, C., Poritz, J.A.: Secure intrusion-tolerant replication on the internet (2002)

[17] Stoica, I., Morris, R., Liben-Nowell, D., Karger, D., Kaashoek, M.F., Dabek, F., Balakrishnan, H.: Chord: A scalable peer-to-peer lookup service for internet applications. IEEE Transactions on Networking **11** (February 2003)

[18] Rowstron, A., Druschel, P.: Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In: IFIP/ACM International Conference on Distributed Systems Platforms (Middleware). (November 2001) 329–350

[19] Maymounkov, P., Mazières, D.: Kademlia: A peer-to-peer information system based on the xor metric. In: Revised Papers from the First International Workshop on Peer-to-Peer Systems. IPTPS '01, London, UK, UK, Springer-Verlag (2002) 53–65

[20] Bastick, Z.: Our Internet and Freedom of Speech 'Hobbled by History': Introducing Plural Control Structures Needed to Redress a Decade of Linear Policy. epractice.eu (March) (2012) 97–111

[21] Mueller, M.L.: Competing dns roots: Creative destruction or just plain destruction? (2001)

[22] Walsh, K., Sirer, E.G.: Fighting peer-to-peer spam and decoys with object reputation. In: Proceedings of the 2005 ACM SIGCOMM workshop on Economics of peer-to-peer systems. P2PECON '05, New York, NY, USA, ACM (2005) 138–143

[23] Marti, S., Garcia-Molina, H.: Taxonomy of trust: Categorizing p2p reputation systems. Working Paper 2005-11, Stanford InfoLab (2005)