

Local Name Resolution In Disconnected State

Samu Varjonen

Helsinki Institute for Information Technology

samu.varjonen@hiit.fi

Abstract

Domain Name System (DNS) is designed to transform human readable Internet addresses to corresponding Internet Protocol (IP) addresses. Current DNS relies on hierarchical infrastructure. It could be useful to use services provided by DNS in Local Area Networks (LAN) when there is no Internet connectivity. This can be caused by temporary network errors. There is proposals on using Distributed Hash Table (DHT) based systems to replace DNS or to give it more flexible counterpart. DNS caching is discussed in this paper, because it speeds up resolving process and makes it more reliable. In the case of local name resolving in disconnected state, this paper looks into software solutions for implementing local DNS-caches on local machines. Offline browsing is also discussed in this paper, because it solves the biggest need for local name resolving for average computer user.

KEYWORDS: DNS, PDNSD, DDNS, CoDoNS, DNSsec

1 Introduction

While the current broadband connections are more reliable than older dial-up systems used before, there still can be network problems that leave users in disconnected state. This state can be partial. For example DNS servers known to the machine might be congested or down and the user can't resolve names, while the connection to the target machine could still be made. This paper presents possibilities for home network administrators on how to manage name resolving and offline browsing in disconnected state. Background history of DNS and DNS over DHT based systems is discussed in Section 2. In the Section 2.3 we discuss about performance of the fore mentioned systems, DNS caches. In the Section 3 we will discuss about software solutions for users to cache DNS records and WWW-traffic.

2 Background

This Section discusses about the current Domain Name System and the suggested replacement systems using DHT overlay. These replacements can also be used in cooperation with the legacy DNS system, then they will act as globally distributed and cooperating DNS cache. For example bigger organizations could use these kinds of systems as a replacement for legacy systems inside their organizations.

2.1 Domain Name System

The Domain Name System constructs a tree of domain name servers that administer a certain namespace. Every namespace can be divided into smaller namespaces. One nameserver can handle multiple namespaces. Every namespace/nameserver in a tree has one or more resource records. These records contain information associated with the name. Namespaces are often divided to zones. Zone is considered to be a part of the namespace which control is delegated to some nameserver. DNS is hierarchical system, the client issues the query to its nearest DNS server. Usually a client issues a query to his/hers Internet Service Providers (ISP) server or in case of organizations, they can have their own namespace to administer and they enforce their users to use it. When DNS server receives the query it looks from its own database, does it know the address. DNS server can continue in two ways: direct or recursing. In the direct mode, DNS system gives the query to its parent and it gives the query to its parent until some node knows the address, knows that one of its children knows the address and directs the query to its child or the query reaches the root servers. When the query reaches root server the direction to a child that handles that namespace will be known and the query will be delivered to it. If there is a corresponding RRSet it will be returned, otherwise an NXDOMAIN response is returned with "Name error" (RCODE = 3, [11]), telling that the name does not exist. In both cases the message is routed back to the originator of the query. Query done this way is called an recursive query (see figure 1 arrows from 2 to 5). Iterative query works in a slightly different manner (see figure 1 arrows from 6 to 9). When a nameserver does not know the name it will return message indicating the next nameserver in the tree that might know the name. The Resource Records usually contain name, value, type and TTL. Most common types of the RRs are A for address to a name, NS for administrative name servers address, CNAME for alias for a name and MX for mail servers name for a domain. DNS protocol consists of two messages query and reply. To make it even simpler structure of the messages are identical.

2.1.1 DNS Security extensions

DNS Security extensions (DNSsec, [12]) was originally designed for protection against malicious users. For a malicious user it is possible to forge an IP address and so effectively to pretend a DNS server. It may be possible to inject forged Resource Records (RR) to the system, but this attack depends on the configuration of the targeted nameserver. It may be possible that the verification that the RRs came from

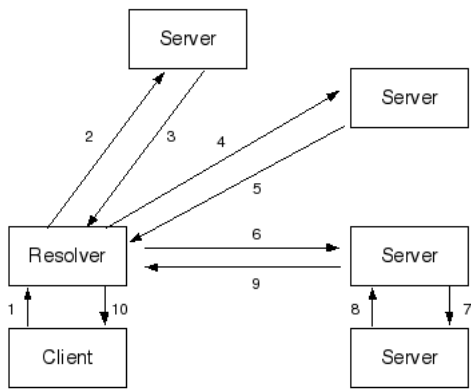


Figure 1: Name resolution iteratively (2-5) and recursively (6-9) in legacy DNS.

the correct authority is not checked properly. This attack is known as *cache poisoning*. For more details about threats against the legacy DNS that the DNSsec is solving see [13]. DNSsec uses public key cryptography and digital signatures to provide authentication of the origin and integrity protection for the DNS records. DNSsec stores these signatures to a new type of resource records called Resource Record Signature (RRSIG). DNSsec has also added a few new header bits to the legacy DNS messages. With those bits user can tell the DNS that he/she wants signatures to be returned as well.

When client wants to resolve some name to address he/she queries the name from DNS and receives address and signature then the user makes another resolution to get the signers public key and verifies the signature. While DNSsec improves the security of the legacy DNS it introduces also a new kind of attack [14]. For example malicious user can try to get DNSsec enabled server to waste its resources on cryptographic tasks. Zone enumeration or zone walking is considered to be something that only slave servers of primary master nameserver can do. This means that only a slave server can ask the whole namespace contents. While namespace information is public it is considered to be bad to let outsiders have access to the whole namespace contents [14] [10, p. 356 – 358].

2.2 Distributed Hash Table

Use of DNSsec extensions on conventional DNS systems effectively separates authentication of the data and serving the data. With this separation it is made possible to use systems like Distributed Hash Tables (DHT) to serve content and DNSsec to secure it [4].

Distributed Hash Tables (DHT) are distributed systems that provide decentralized storage service where data is saved in key-value pairs that are distributed into the system based on their key value. DHTs provide scalable and failure tolerant storage system. DNS over DHT systems also handle DoS attacks better than hierarchical legacy DNS because there is no hierarchy in the system and because the data is replicated to set of servers. Usually 6 to 8 servers depending on the underlying DHT. Replicating servers can also be

chosen in pseudo random manner. This makes it harder to a malicious user to bring down the system. A malicious user would have to bring down a large set of servers before effect of the attack is noticeable [7]. The proposed DNS over DHT systems like CoDoNS and DDNS have also their shortcomings, for example load balancing. With DNS it is very easy to provide class A RRsets in a seemingly random order every time a name is resolved or in a round robin manner, at least DDNS does not support this [4]. One of the biggest problems we think is the usage of the DNSsec. You are still required to have access to well known ISPs secret key or get them to sign your RRsets to be trusted. Ramasubramanian and Sireer propose that signature could be bought from respectable namespace owner [7]. Nothing prevents a user to sign with their own key but then the problem becomes similar than in PGP, who's signature is trustworthy. Also there is the point that the DNSsec is not that widely distributed [16]. With the DNS over DHT the local administration would not need to configure their own servers. It has been studied that most of the common problems with DNS servers are caused by incorrect configuration. Albitz and Liu list some of the most common configuration related problems in their book [10, p. 431 – 461]. In the next chapter we give a brief explanation of distributed hash tables using the Bamboo-DHT implementation. There is little differences in the exact syntax depending on the underlying DHT infrastructure, but the basics are the same. Also the handling of the Time-To-Live (TTL) differs a bit from system to system. For example in Dhash there is no actual TTL in use.

Bamboo-DHT was originally based on Tapestry, but currently is considered as a Pastry like system or completely new system. In Bamboo-DHT nodes of the overlay network are organized as a ring. In the ring nodes are assigned IDs based on their IP. Bamboo-DHT provides a simple put, put_rm, get and rm API (where rm stands for remove). Put message is defined $put(key, value, TTL)$ and get message is defined $value = get(key)$. Put_rm stands for removable put and it is defined $put(key, value, H(secret), TTL)$. If the secret parameter is left NULL then put_rm message is treated in similar manner as the regular put. The rm message is defined as $rm(key, H(value), secret, TTL)$. When a put message is delivered to a node in the overlay, it is routed to responsible node that saves the value. The network finds the responsible node by comparing the key and node IDs. The node with closest ID to the key saves the value. OpenDHT supports replication where the data is saved to responsible node and to its replicas. In case where subsequent put message has same key and value as the original put, the TTLs of the messages will be compared and larger of the TTLs is updated to the DHT. In case where only the key is same but the values are different, the value will be saved under the same key. Every key-value pair has a Time-To-Live value that is a value between 1 and 604,800 seconds (circa one week). TTL tells the system how long it should be stored in the system. Rm message should have longer TTL than the original put message. This is because the replication can cause the original put to reappear. This happens when the put is made and replicated and one of the replicating nodes goes down. If the TTL is shorter than in original put, the rm message can be removed from the system before the replicating node comes back. If

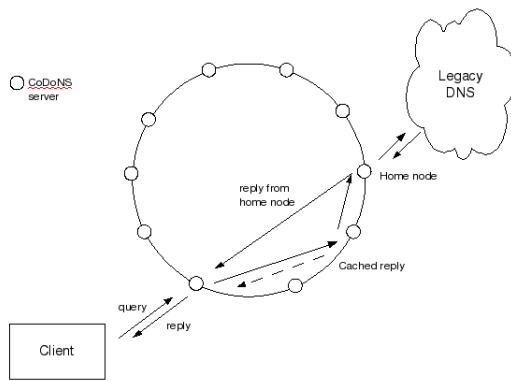


Figure 2: CoDoNS architecture. Client sends a query to CoDoNS and the query is forwarded to the home node. If home node does not know the name queried, it asks it from legacy DNS. If some node on the way finds the answer from its cache it may answer directly (dashed line).

the rm is removed from the system the replicating node starts the replication process and the key-value pair will reappear as if it was never removed. Removable put also includes hash of the secret. It is used to identify the original issuer of the put. In the remove message the originator of the put message reveals the secret to the system. Hash of the value is used to identify the correct value under one key in cases where there is several values. Because of the DHTs ability to save multiple values under one key, can get message return multiple values found under one key.

In the following Sections we discuss about two DNS over DHT systems. They both have a bit different approach to the subject. CoDoNS is more like a peer-to-peer replacement for the legacy DNS but it can cooperate and act as a distributed cache for the legacy DNS. DDNS is more like distributed cache.

2.2.1 Co-operative Domain Name System

Co-operative Domain Name System (CoDoNS) servers work on top of Pastry and Beehive [7]. CoDoNS uses the same message format and the same protocol as the legacy DNS system, making it easy for them to inter-operate. CoDoNS servers are organized as a ring and the nodes are assigned an identifier, which is used to tell which node is a *home node* for a given RRSet keys hash. CoDoNS system replicates RRSet to number of its home nodes adjacent nodes. In the case where the home node of the RRSet goes down, will the next node in the ring take its place. If answer to a query is not found the system will forward the query to legacy DNS (see figure 2). Answers from legacy system will be stored to the CoDoNS and pro-actively checked from legacy DNS. Users can also directly insert records to CoDoNS. When a user issues a query to CoDoNS system, the node contacted will reply immediately if the node is the key-value pairs home node, otherwise the message will be routed to the home node. If the home node does not have the RRSet queried the home node will forward the query to legacy DNS and from that answer the RRSet will be saved to the home node.

CoDoNS servers support *direct caching* where the users

can insert RRSet only to their own CoDoNS server and instruct their server to serve RRSet rather from their own cache than the whole system. CoDoNS system supports inverse queries where an IP address is resolved to a name rather than a name to IP address. As most of the DHT based systems have TTL. CoDoNS system does not directly support TTL for its RRSet. CoDoNS uses the RRSet TTL to identify when it needs to update RRSet from legacy DNS. Because of the load balancing done in legacy DNS the CoDoNS system will not store RRSet with low TTL. Usually RRSet that have low TTLs are used with load balancing or indirection systems. CoDoNS system can support *negative caching*, where the NXDOMAIN messages are cached to the system for short periods of time. NXDOMAIN messages are usually responses to queries that result in no answer. For example queries that contain typos that make the query to ask address for a name that does not exist. The CoDoNS system is running on planetlab and it can be easily tested. Just modify the `/etc/resolv.conf` to point to one of the running nodes in CoDoNS system. Available nodes can be found from the CoDoNS website [20].

2.2.2 DDNS

DDNS is a Domain name system that uses DHash and DNSsec (Not to be mixed up with Dynamic DNS defined in [15]) [4]. DDNS uses DHashs ability to balance load and the fact that DHash is self-healing, meaning that it does not suffer from leaving and joining nodes. This is provided by the underlying DHash. In Chord the nodes are organized in a ring like fashion. Every time an get message is issued to a node it will be routed through the ring to the node responsible for saving the key-value pair. When the response is routed back, the nodes along the way save the key-value pair to their caches. Shortening the answer time in subsequent queries. To provide the robustness DHash replicates the key-value pairs to a pseudo randomly chosen set of six servers. In DDNS they use SHA-1 hash of FQDN concatenated with resource type to store the value. For example SHA-1(www.google.com|A). For every put and update message in the DDNS, DHash verifies the signature before accepting a message. The same applies on the client side also, the client should verify the signature included in result of the query before using it. Cox et al. propose to publish keys in KEY record type for popular names, to minimize the need to do verifying along the way for every node the query traverses [4]. While DDNS seems to be efficient replacement for legacy DNS, it lacks support for load balancing features implemented in legacy DNS. Legacy DNS is capable to balance load to a set of servers by returning the addresses in random order. Some hosts like www.google.com use round robin style load balancing where addresses revolve. While DDNS is distributed, it still needs authoritative hierarchy to supply it signatures for DNSsec. When a user wants to add his/hers domain name to the system, he needs to have access to some well-known ISPs secret key. In reality the user can ask the well-known party to sign his/hers RR or buy the signature from a well-known and trusted company like Verisign. If there is no certificate authority (CA) hierarchy, PGP's web of trust approach could be adopted.

2.3 Performance

In this Section we discuss about the performance of legacy DNS and DNS over DHT systems. Performance can be divided into categories.

Availability: In legacy DNS one malfunctioning or down server can effectively separate user from the network. Separation might not be complete depending on the server that is down. User might be able to resolve only names of the local network or names in some partition of Internet. While user can not use DNS to resolve names to addresses, it is possible that the connection to the peer could be made if user has some other way to resolve names. DNS caches can be used to provide this. While caches are effective in providing resolutions, they can be problematic as discussed in 2.3.2. DNS over DHT systems can be considered to have benefits in this category. DHT systems are self-healing and they support replication of data they store. This means that the system is not affected by nodes in the ring coming and going. In overall the distributed systems can perform better in this category.

Attack resilience: This category is partly a sub-category of the availability, because of Denial of Service (DoS) attacks. Main idea in DoS is to flood the target with so much bogus work that the target for example can not reply to any valid queries. In Legacy DNS it is easy for an attacker to find attack points that have significant effect on the system performance, when the attack succeeds. In DHT based systems it is harder for an attacker to find “weak” points. DDNS can be considered better than CoDoNS in this matter, because DDNS replicates data to pseudo randomly chosen set of nodes. Rest of this category consist of attacks targeting stored data. In legacy DNS it is hard for an attacker to inject false data to the servers, but attacker could spoof answers from DNS. DNSsec is designed to prevent this as discussed in 2.1.1. DNSsec provides security in DDNS and CoDoNS.

Lookup latency: Latency is considered to be the time between sending of an query and receiving an answer from the resolving system. There are several things that affect latency. For example usage of caches and DNSsec. Caches tend to lower the latency. If user wants to be sure that the answer is completely valid, he/she has to query all the keys and signatures up to the root level and this can result in significantly longer latencies.

Failure rate: This category considers the availability of the data. Failure rate is an estimate of the amount of negative answers from the system, like the name error message discussed earlier in 2.1. This category is based on estimates made from tests performed on legacy DNS.

Load Distribution: This category considers the distribution of RRsets into the system. In DNS the Hierarchy was designed so that the data would distribute in even manner to the nameservers. It has been shown that this does not work in current DNS. In DNS over DHT systems DHT provides better load balance by hashing the key under which the value is saved.

2.3.1 DNS

DNS system is a critical component for mapping human-readable names to addresses and its performance has been

studied widely. Recently there has been more and more malicious behavior against the DNS system and also the client population who has to use DNS has increased. In recent studies it has been shown that legacy DNS is not suitable anymore. This because of the administrative needs of the DNS and its lack of fast reconfiguration [7, 4]. Attack resilience is considered as the biggest problem of the legacy DNS. For a malicious user it is possible to launch a Denial of Service (DoS) attack against the DNS. Malicious user can also spoof answers from the DNS to the client.

Ever growing client population results in growing namespaces. Growing client population affects performance and results in need to have bigger namespace. While namespace grows, it has been shown that it does not grow evenly. Ramasubramanian and Siner noticed that most of the new names go into the popular domains like *.com* [7]. This on its behalf skews the load distribution in legacy DNS. This uneven distribution causes the load of the servers to distribute unevenly. So that the servers administering popular namespaces have higher load. Pang et al. noticed that DNS servers with high load usually are more available [1]. This can be caused by constant maintenance, better hardware and redundant network connections. They also noticed that most of the users use only a small fraction of the available servers [1].

Failures in the DNS system can be caused by anything from incorrect configuration of the servers to simple typos made by users. Albitz and Liu state that most of the failed queries are caused by improperly configured servers. Their book also lists the most common configuration mistakes [10]. In the studies made at the MIT they showed that 23% of the queries get no answer. This result is affected by the retransmissions in the network. In their paper Jung, Sit and Balakrishnan found that only 13% of the queries actually result in an error [2]. In that study was noticed that those errors were caused by missing reverse mappings and incorrect NS records. In their paper they suggest a modification to the DNS that could improve the legacy DNS systems root servers’ performance. The improvement was suggested because they noticed that from 15% to 27% of all the root server traffic results in negative answer and that most of these errors are caused by typos pointing to non-existent names. These errors could be from users but also from incorrectly implemented resolvers. They suggest that intermediate servers should refuse to forward malformed queries.

Currently users have more and more mobile equipment and they need to be contacted when they move. This mobility can cause users IP address to change from network to network depending on users movement. Legacy DNS can be considered too static for this kind of usage. Because DNS was designed when equipment was more or less stationary, it was not taken to account that someday fast updates of the records could be needed. Currently for example a user could use some Voice over IP (VoIP) software on his mobile equipment. If a call is made trying to contact him the system has to have a way to resolve his current address. This can be implemented in many ways, but the idea in most of them is to use one static server. In this approach the users RRset points to this static server and the user updates its current address directly to it. These static servers are commonly called rendezvous servers (RVS) or home agents.

2.3.2 DNS Caching

DNS caches can improve the latencies of popular name queries like `www.google.com`. If the queried name is not found on the local or the nearest cache the query will be forwarded. In this case the latency of the DNS is the same or little longer than without the cache. After the query is returned from the DNS to the cache it is stored in its storage. Queries about that name are served directly from the cache resulting in shorter latencies than without the cache. Caching RRsets locally can so improve the performance of DNS but not without problems [2]. Load balancing and mobility are two issues most affected by the use of caches. If users use local caches the load-balancing features will be broken because users use the same address they were given earlier and the DNS can not issue an address to a server that has lower load. Similarly in the mobility issue the user uses the address given to it earlier and does not get the current address. Blumenthal and Clark point out in their paper [9], that there are problems also in case of content caches like Squid [19]. They state that content caches break the end-to-end nature of Internet, server administrators can not track accurately how popular their sites are and that the servers can not produce client specific actions on retrieved pages. These issues can be solved by using lower TTLs on records that belong to load balancing servers or mobile nodes. The caches could then implement a rule saying that for example RRsets with TTL lower than 30 seconds will not be cached. Negative caching is also possible. Negative caching caches NXDOMAIN messages with name errors [11]. Negative caches records should have low TTLs because the name could be taken in to use and because storing negative results for a long time could render the system usable [7]. This is caused by the fact that many of the negative answers are caused by typos and users can make unlimited number of typos or they try to reverse map some IP address that does not have reverse mapping [2].

2.3.3 DNS over DHT systems

While the presented DNS over DHT systems are similar their performance is hard to evaluate. This is caused by different test methods used and the size of the systems. Performance tests done with DDNS were results from rather small setup and limited timed interval. In CoDoNS system the performance results base to a circa 70 server setup on planetlab and the tests continue while writing of this work. CoDoNS systems writers are currently trying to invite people to use their system, so they can get more results based on normal usage of DNS systems.

Cox et al. [4] show in their results that distributed systems are more resistant to Denial of Service attacks. This is based on the fact that it is harder for the malicious user to find single weak point in the system where all the machines are as worthy as the other. So no one machine is more important than the other. Cox et al. also showed in their paper [4] that the underlying DHash implementation improves the latency for popular name queries. This is a result of the Dhash's way of caching answers on the way. Every node that routes the query to the node that saves the queried value, saves the answer for short times on their local caches.

Ramasubramanian and Sirer tell in their studies that their

system can outperform legacy DNS system. At least when considering bandwidth and storage requirements. In their paper [7] they say that CoDoNS system uses less storage base and less bandwidth. Exact reason for this result was not clear. Because the protocol is similar and uses the same message format as the legacy DNS. Similarly as in DDNS, CoDoNS is also more attack resilient than legacy DNS. In CoDoNS system the latency can be shorter than in legacy DNS, but it can also be much higher. Higher latencies occur when the value is not found from the system and the query is forwarded to the legacy DNS. Then the latency includes all the routing inside the CoDoNS system and the forwarding and routing in legacy DNS and the way back.

Load balancing in DNS over DHT is better than in legacy DNS. This is caused by the consistent hashing that the DHT systems use. In DHT systems the keys are hashed before the systems know which node should save the key value pair. This results in fairly even load balance throughout the system. In legacy systems the storing server is decided over hierarchical parameters. In other words the server that controls that part of the namespace saves the data. In DNS over DHT systems the zone data is basically removed and there is no one node handling certain part of the namespace.

2.3.4 Comparative effectiveness

It is difficult to compare legacy DNS and DNS over DHT systems. Performance information has been gathered from the legacy DNS system by multiple research groups. Performance data from the DNS over DHT systems were gathered from small systems compared to the legacy DNS. When discussing about the latency, legacy DNS outperforms DNS over DHT systems. For example studies made by Cox et al. show that DDNS system has median latency around 350 ms and the legacy DNS has median latency around 43 ms [4]. Pappas et al. show similar results in their studies too [5]. In their paper the results were similar in normal operation conditions. When they introduced high node failure to the systems, the DNS over DHT systems outperformed the legacy DNS. This is a result from the replication features in DHT systems. Pang et al. studied DHT based systems in more general manner and compared the results against the similar setups with the legacy DNS. In their studies they showed that DHT systems suffer from their maintenance loops. Over certain intervals the DHT systems fix their routing tables and check if their neighbor lists are up to date. In their studies they showed that the performance of the DHT based systems can be improved by extending the maintenance interval. It also had its downside, by extending the maintenance interval the self-healing features of the DHT suffered, making the system less available. Gupta et al. showed in their study [8] another way to improve the latencies in DHT based systems. They suggested a one-hop routing scheme. In that every node in the network would have complete information of the network. Their studies showed that one-hop routing scheme can handle systems consisting of even 2 million nodes. For systems larger than that they introduced two-hop routing scheme. These schemes introduced a pseudo hierarchy to the normal DHT ring. The ring was divided into slices and they were divided further into units. Both slices and

units had leaders, chosen from the middle of the slice or unit. The leaders of the slices multicast the routing table changes to their subordinate unit leaders and other slice leaders. The unit leaders multicast routing table changes to their subordinates. This is done to reduce the traffic caused by the routing table changes in the system. Gupta et al. also discussed about the supernode and inner ring of supernodes concepts. They suggested that there could be a formally maintained inner ring of supernodes. These supernodes would be maintained by ISPs and governments. The intention was to provide stable base system where every one could join.

3 Solutions for working in disconnected state

There is multiple solutions for enabling local name resolution in disconnected state. User could use local DNS-cache like Proxy DNS Daemon (PDNSD), user could use local installation of WWW proxy. Success of the first approach is highly dependant of the TTL used to cache DNS records locally and has he resolved the name before. This approach might allow the user to get the name resolved in disconnected state, but it still does not guarantee that the user will get any content. User can use local installation of PDNSD or Berkeley Internet Name Domain (BIND [18]) to achieve this solution. BIND could prove to be too hard to configure for a average user. If user is running a local installation of Squid or something similar, he might get the content also, but the content could still be old.

3.1 Local DNS cache

Proxy DNS Daemon (Pdnsd) is software that provides permanent caching of the name resolution process [17]. Pdnsd was originally written by Thomas Moestl, but is now maintained by Paul Rombouts. Pdnsd is used to do DNS lookups and cache them. These caches can be used to avoid long timeouts in name resolution functions such as *getaddrinfo()* that may be caused by unstable link to Internet. Pdnsd can be configured at startup or at run-time by using control program. Pdnsd has full IPv6 support. Pdnsd is also capable of reading */etc/hosts* file and import its contents as local DNS records. Pdnsd is written for Linux operating system, but it has been ported to FreeBSD and Cygwin.

Dnsmasq is another local DNS cache program. It can also act as DHCP server for small networks. Dnsmasq was intended to forward and hide DNS traffic behind one address in local area networks that use NAT. Dnsmasq is readily available to multiple platforms. Its installation is very easy. Only two packets are needed. Very little configuration is needed. Similarly than Pdnsd, Dnsmasq can import */etc/hosts* file. This makes it easy to configure all the hosts in small local area network to the Dnsmasq machines *hosts* file and modify rest of the machines to use Dnsmasq machine first. This is done simply by changing the first line in */etc/resolv.conf* to "search <IP of Dnsmasq>". Dnsmasq has only one down side, it doesn't support "permanent" caching. It does not save the cache to disk on shutdown like the Pdnsd does.

Name service cache daemon (Nscd) is a program that was

originally written for Solaris but became popular with Linux also. Nscd provides various caching options for its users, like caching of user and group information. Basically everything that can be called through libc and function like *getXbyY*. For example *gethostbyname* is one of them. One of the strongest features in Nscd is its configuration file. Nscd gives the possibility to control even the TTL of negative caching.

3.2 Local WWW content cache

Caching WWW content locally is usually called off-line browsing. Currently on Linux the major browsers support off-line browsing to some extent. There is also multiple programs available for making local mirrors of whole websites. For example *wget*, *webHTTrack*, *KHTTrack*, *HTTrack*, *SpiderFish*, *MojoServo* and *Getleft*. Their usage is in most cases very simple. User tells the program what is the URL of the site he/she wants to save locally and then the user tells the program how deep should it go. Depth in these cases is considered to be count of links from the first page. From the user side these programs introduce a new decision to the user. Should the user save the page or pages locally. This kind of programs are usually called spiders, because they only get the starting point and after that they just crawl through the links saving everything they find, until they have crawled through every link inside the depth limit.

When the user does not want to make extra decisions he/she should install program like Squid, MM3-WebAssistant, WWWOFFLE or ProxyTrack. While the first is not intended to be used locally on every machine, it could be used so. Squid is intended to work on a separate machine on the border of the local area network. Secondly Squid can be too hard to configure for average computer user. Rest of the mentioned programs are very easy to install and use. For example the MM3-WebAssistant implemented in java needs only to be unpacked to some folder and start it from command line or double clicking its jar packet. MM3-WebAssistant first starts a graphical configuration program and asks few questions and tells you how to setup your browsers proxy settings. MM3-WebAssistants configuration includes also couple of tests that the user can see that the program is up and running.

The programs (*spiders*) mentioned in the first chapter of this Section need user interaction to work. Programs (*proxies*) in the second chapter are more or less automatic in a sense that after they have been started they do not need user interaction. Both of these ways have their benefits. Proxies don't need user interaction, but some of them can use all the disk space, if user has not set the limit for the disk usage. Proxies also update their caches when the connectivity is restored. This can cause large amount of traffic from the machine after the connectivity is back. Most of the proxies can be configured to filter out content like images. Spiders on the other hand offered more control over what is downloaded and saved locally and when, but none of them did anything without user interaction.

3.3 Local Distributed DNS caches

While DNS over DHT systems are intended to be built using hundreds of servers, they can prove to be useful in smaller environments too. Programs like Dnsmasq or Pdnscd work on one machine. Their cache can be made available to other machines in the local area network, but the cache is still only on one machine. With little modifications to the replication count, it is possible to run CoDoNS system even on two machine network. In local area networks where every machine would run one node of private CoDoNS node, the cache would be identical over every machine of the network.

4 Conclusion

Both the Legacy DNS and the DNS over DHT systems have their advantages. If we consider the latency, the legacy DNS has better overall performance compared to the discussed DNS over DHT systems. DHT based systems on the other hand have better performance if we consider attack resilience, load distribution and availability.

It is possible to speed up the name resolution and make it more reliable with readily available software. Even in cases where there is no DNS connectivity softwares presented earlier offer local name resolving. All of the softwares described in this paper have their ups and downs. PDNSD has the advantage that it is a user friendly application with little or none configuration needed. Squid is the de-facto standard as a WWW cache, but it can be too hard to configure for average computer user. The simplest way to support off-line browsing is to use programs like MM3-WebAssistant. For example installing PDNSD and MM3-WebAssistant on the machine, results in cached DNS records and cached WWW content, that is updated when there is connectivity. It is also possible to use DNS over DHT solutions as distributed DNS caches in small networks.

References

- [1] Jeffrey Pang and James Hendricks and Aditya Akella and Roberto De Prisco and Bruce Maggs and Srinivasan Seshan, Availability, usage, and deployment characteristics of the domain name system, In *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, New York, NY, USA, 2004. ACM press.
- [2] Jaeyeon Jung and Emil Sit and Hari Balakrishnan and Robert Morris, DNS performance and the effectiveness of caching, In *IEEE/ACM Transactions on Networking*, pages 589 – 603, volume 10 issue 5, oct 2002. ACM press.
- [3] Hari Balakrishnan and Karthik Lakshminarayanan and Sylvia Ratnasamy and Scott Shenker and Ion Stoica and Michael Walfish, A layered naming architecture for the internet, In *Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for communications*, pages 343 – 352, sep 2004. New York, NY, USA, ACM press.
- [4] Russ Cox and Athicha Muthitacharoen and Robert Morris, Serving DNS using a Peer-to-Peer Lookup Service, In *Lecture Notes In Computer Science; Vol. 2429*, Pages: 155 – 165, Mar. 2002, UK london, Springer-Verlag.
- [5] Vasileios Pappas and Dan Massey and Andreas Terzis and Lixia Zhang, A Comparative Study of the DNS Design with DHT-based Alternatives, In *the Proceedings of IEEE INFOCOM'06*, Apr 2006.
- [6] Anupam Joshi and Sanjiva Weerawarana and Elias Houstis, On Disconnected Browsing of Distributed Information, In *Proceedings of Seventh International Workshop on Research issues in Data Engineering*, Apr 1997, Pages 101 – 107.
- [7] Venugopalan Ramasubramanian and Emin Sirer, The Design and Implementation of a Next Generation Name Service for the Internet, In *Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications SIGCOMM '04*, Volume 34 Issue 4, ACM Press.
- [8] Anjali Gupta and Barbara Liskov and Rodrigo Rodrigues, Efficient Routing for Peer-to-Peer Overlays, In *1st USENIX/ACM Symposium on networked systems design and implementation (NSDI '04)*, San Francisco, CA, 2004.
- [9] Marjory S. Blumenthal and David D. Clark Rethinking the Design of the Internet: THE End-to-End Arguments vs. the Brave New World, In *ACM Transactions on Internet Technology*, Vol. 1, No. 1, August 2001, Pages 70 – 109.
- [10] Paul Albitz and Cricket Liu, DNS and BIND, Fourth Edition, O'Reilly and Associates, 2001.
- [11] M. Andrews, Negative Caching of DNS Queries (DNS NCACHE), *RFC 2308*, March 1998.
- [12] D. Eastlake, Domain Name System Security Extensions, *RFC 2535*, March 1999.
- [13] D. Atkins and R. Austein, Threat Analysis of the Domain Name System (DNS), *RFC 3833*, August 2004.
- [14] R. Arends and R. Austein and M. Larson and D. Massey and S. Rose, DNS Security Introduction and Requirements, *RFC 4033*, March 2005.
- [15] P. Vixie and S. Thomson and Y. Rekhter and J. Bound, Dynamic Updates in the Domain Name System (DNS UPDATE), *RFC 2136*, April 1997.
- [16] Carolyn Duffy Marsan, Network World, 03/28/07, Q&A: New IAB chair mulls DNS security, unwanted Internet traffic, From <http://www.networkworld.com/news/2007/032807-iab-chair-dns-security.html>, 17.4.2007.
- [17] Thomas Moestl and Paul Rombouts, From [http://www.phys.uu.nl/~sim\\$rombouts/pdnscd.html](http://www.phys.uu.nl/~sim$rombouts/pdnscd.html), 22.1.2007.

- [18] From <http://www.bind9.net/>, 22.1.2007.
- [19] From <http://www.squid-cache.org/>,
27.2.2007
- [20] From [http://www.cs.cornell.edu/
people/egs/beehive/codons.php](http://www.cs.cornell.edu/people/egs/beehive/codons.php),
14.3.2007.