Rearchitecting DNS

Gert Pfeifer Christof Fetzer
Martin Steuer

Dresden University of Technology, 01062 Dresden, Germany

{firstname.lastname}@inf.tu-dresden.de

Abstract: The Domain Name System (DNS) has been the naming service of the Internet for more than 20 years. It is the foundation of virtually all other distributed service. The deficiencies of DNS with respect to performance, availability, and reliability are well known. We want to systematically improve DNS by (1) measuring DNS performance, (2) evaluating past proposals, and (3) try to come up with a new architecture for DNS. SEDNS is a system in which we investigate various design ideas to speed up DNS requests and improve availability, reliability, and security.

1 Introduction

The Domain Name System (DNS) is a hierarchical distributed database that can map names to some data. In most cases it is used to map a name to an IP address. The system is mature and very successful. There is hardly any distributed Internet-based application that does not use it. Unfortunately DNS is not always sufficiently dependable for companies that must depend on it. Almost always some DNS servers are not reachable. Sometimes none of the DNS servers might be available (e.g., when DNS is misconfigured or under attack). DNS responses are not always correct, which may have different reasons, e.g., cache poising. However, the most alarming problem is that DNS can be used to attack other services or a downtime of DNS can make services unusable. Many systems depend on DNS without having a fallback. So a good way to attack a distributed application is, first to make sure that itself or its clients cannot resolve names anymore.

We are designing a system, SEDNS, to cope with all these problems. SEDNS is not a replacement for DNS. Replacing DNS would be difficult since it is mature, quite well understood, popular, and such a shotgun approach often fails. So we will continue to use DNS and, just in case that it does not work according to our expectations, we provide a combination of local caches together with a peer-to-peer based fallback system. We show how to optimize the performance using local caches similar to Supercaches as introduced by [AMT03].

The rest of the article is organized as follows: Sections 2.1 and 2.2 discuss the different sources for unavailability of DNS. Section 3 discusses DNSSEC in this context. Section 4 introduces our SEDNS approach and discusses how to optimize a Peer-to-Peer overlay for this use. Section 5 concludes the paper.

2 DNS Dependability Issues

DNS is inherently unreliable, due to the fact that it is using plain-text communication via UDP as the default setting. Hence, modifications of the content can not be detected and in the case of packet loss, retries include the resolvers time-out as an extra delay penalty. TCP, which would solve this problem, is only used when the amount of data is too large to be contained in a single UDP packet. Of course, TCP introduces some overhead so that UDP seems to be the better choice if performance is the main concern.

The consequence of using UDP is that applications have to deal with requests not being answered or replies being dropped. To optimize yield, i.e., the ratio of answered requests out of the number of requests sent, we need to investigate the factors that impact the yield. We say that a failure occurred if the time-out of the resolver has expired, i.e., 5 seconds in most implementations, or that an error is returned by the server. Failures can be classified in client-side and server-side failures, like proposed by Park et al. [PPPW04].

Client-side failures are failures caused by the client's infrastructure, i.e., the stub resolver, the recursive resolver, or the LAN connection. Server-side failures are failures caused by the DNS servers involved in the processing of the request, including WAN communication failures. The following subsections describe these problems in more details.

2.1 Server-side problems

When a recursive resolver tries to find a certain resource record, it iteratively traverses the DNS name space. Starting from the top of the tree, i.e., a root server, it tries to find a DNS server that is responsible for the top level domain (TLD). From this server the search goes on from name server to name server until the request can be answered.

Since a successful DNS lookup depends on the cooperation of many remote components it is worth looking how reliable this process is. Jung et al. [JSBM01] tried to examine the effectiveness of DNS caching. In addition, their work shows the reliability of the DNS server infrastructure. They captured DNS packets at two locations, i.e., two Internet gateways. The traces show that over a third of all lookups where not successfully answered. The number of query packets is larger than the number of lookups because if a DNS server does not receive an answer for a query, it retransmits the query. This is a robustness mechanism that deals with UDP packet loss, leading to the fact that query packets for unanswered lookups account for more than 50% of the DNS query packets in the trace.

Another interesting behavior is that between 3.1% and 4.9% of the unanswered lookups formed a querying loop between two or more DNS servers. This is obviously a configuration problem. Erroneous configurations also cause many negative answers. Negative answers account for 10% to 42% of the answers in the trace. Often they are caused by querying a name that does not exist. Of cause, this might be a result of user input, such as ”ww.att.com” in the address field of a browser, but the largest cause are inverse lookups.

Usually an administrator configures at least two mappings for a name. One mapping points from the name to the IP address, another one points from the IP address to the name. This is very useful and many services are using this feature, e.g., ssh. Obviously many administrators omit configuring these inverse mapping.

Other significant causes are NS or MX records that point to names that do not exist. This might be a problem of inconsistency in a DNS zone, where old entries have not been removed or new entries have not been checked for typos.

Apart from these configuration issues there is also the problem that servers might fail completely. Reasons are software and hardware failures as well as occasional denial-of-service (DOS) attacks.

DNS is a compelling target for DOS attacks since DNS servers are well known and easy to find, a successful attack to a single server or pair of servers can cut of a whole branch of the DNS name space, and DNS outages often cause outages of other services as well. Such an attack (in this case ping flooding) hit the 13 root servers on October 23, 2002, causing some slowdown in the Internet [Nar02]. Only four root servers were able to continue service during the attack. In January 2001, Microsoft suffered from such an attack resulting in downtime for its Web presence [Thu01]. As a result, Microsoft hired Akamai Technology to ensure that such attacks do not succeed in the future. Akamai maintains geographically separated DNS servers, while all of Microsoft’s DNS servers were in the same location and thus easy to attack.

2.2 Client-side problems

Many DNS outages are not caused by global network outages or server failures, but by problems with local components, like recursive resolvers or stub resolvers. Stub resolvers are using local DNS servers (LDNS) as recursive resolvers. These servers are responsible for resolving names iteratively and also provide a cache for DNS resource records. Hence, they do the main part of resolving a name. This is necessary since stub resolvers are too limited for this task.

If these LDNS servers suffer from network problems, more or less frequent overload situations, or hardware and software failures, the DNS is unavailable. Now this seems to be a problem that can be solved by the DNS administrators. As discussed by Park et al. in [PPPW04], there might be reasons that are deterring administrators from solving this problem: Most of them optimize cost-to-benefit ratio expressed by CPU utilization. DNS is not CPU bound and it typically leaves enough CPU power to run other services on the same machine. Installing more services improves utilization but introduces a competition between services for resources. Waiting for memory or CPU, DNS might fail from time to time, i.e., let the resolvers time-out expire.

Park et al. propose to use an insurance-like service (CoDNS) where many partners share

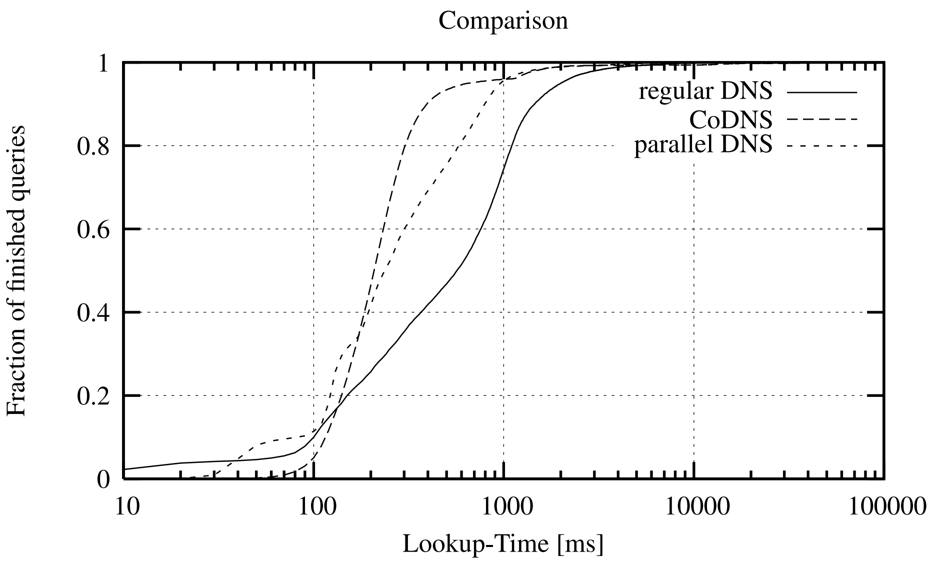

Figure 1: Comparison of average answer time without caching

their recursive resolvers. DNS requests can be forwarded, using a peer-to-peer overlay network, to other recursive resolvers. The performance of the system depends on a good selection of time-out values and knowledge about the health of other peer's LDNS servers. According to benchmarks in [PPPW04] CoDNS outperforms standard DNS by 27-82 % (latency) and increases availability by an extra 9. This can be reached by exchanging information about the health of the LDNS server, i.e., its performance and availability, with other peers in an overlay network. If the LDNS server fails, the DNS request is forwarded to a nearby overlay peer with a good LDNS server. CoDNS is deployed in Planet-Lab [Ros05], where we are using it to compare it with the front-end for SEDNS. It is configured using one extra peer for lookups and 200 ms time-out for the initial DNS request to the LDNS server. The overhead that is needed is, to monitor 10 other peers and keep track of their LDNS server's health.

SEDNS uses a modified stub resolver to query a super cache first. After a short time-out, other LDNS servers are queried in parallel requests. Using several LDNS servers masks client-side problems. In the case that this still does not obtain an answer, another local resolver is queried. It is using a peer-to-peer approach to find DNS data even in the presence of server-side problems.

Two benchmarks are presented here: (1) We measure our front-end without caching, i.e., we try to eliminate caching as good as it gets[1], and (2) with warmed-up caches. The first setting shows that the extra effort of almost doubled number of DNS messages is sensible and that performance is good in the case that the SEDNS super-cache does not produce

[1]We did not have exclusive access to the DNS servers. So our measurements have been influenced by each other and other resolvers.

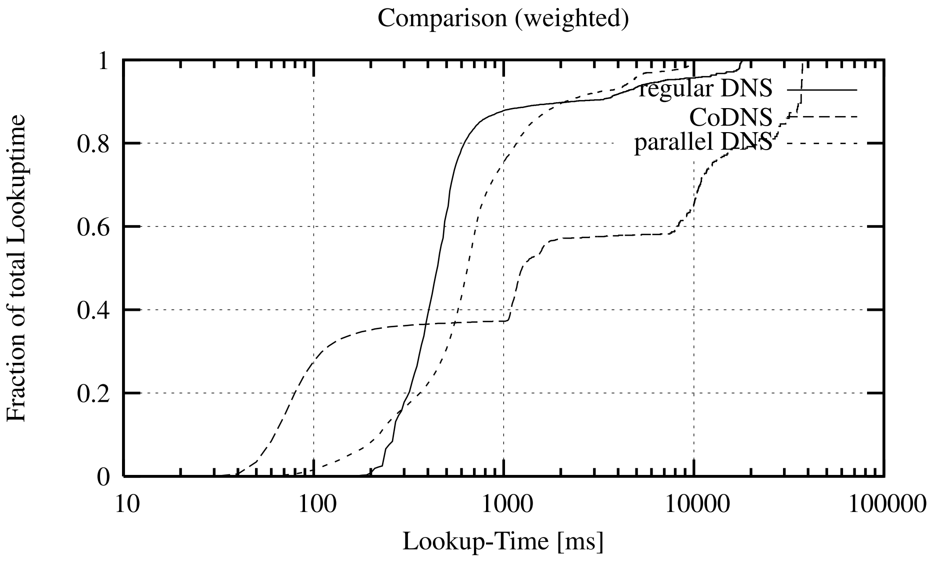

Figure 2: Comparison of weighted average answer time without caching

a hit. The second one shows the influence of LDNS and network failures on the request delay. The result is that our approach is most effective in masking client-side failures.

For using SEDNS the user must install a glibc patch for stub resolvers and the local administrator has to set up additional resolvers and a super cache in the LAN. SEDNS is usable even in the case that DNS is globally unavailable, which is not possible with CoDNS since it relies on DNS.

In comparison to CoDNS, we do not require a peer-to-peer component to be installed on each client machine, even though it would be possible, of course. Neither do we monitor the health of DNS servers.

Our approach separates different techniques to reach different goals as discussed in Sec. 4. We are still early in the design and development process, but we can already present some promising measurements.

So far we have implemented a glibc extension that uses parallel requests to the primary and secondary name server. Of course this doubles the DNS traffic in the case that both of them are doing fine, but we don't need any monitoring traffic and we keep the caches of both servers up to date. If the added traffic is of concern, one can delay the second request by a short time-out sufficient to get fast replies from the first server. In this way, one can balance the network load vs. the response time. Given the fact that most requests will be answered from cache, there is almost no additional load on the CPU, and LAN traffic is cheap, so this seems to be a practical approach.

Figure 1 shows a comparison between our resolver (parallel DNS), CoDNS and DNS. We used a list of 23771 DNS names and calculated the average of our measurements from 490

Method	lookups	total time	avg. time/lookup
DNS	23771	18074599ms	760ms
CoDNS	23771	9735546ms	409ms
parallel DNS	23771	9176794ms	386ms

Table 1: Results without caching

Planet-lab nodes[2]. We found out, that measuring the performance without caching effects is difficult. The graph shows that parallel DNS has gotten more responses within 40 and 200 ms than any of the other approaches. CoDNS has advantages for answers between 200 and 1000 ms. This is due to the ability to choose the best performing peer out of 10 neighbors while we just use our LDNS servers. However, we did not succeed in getting rid of caching effects between the different measurements, which makes evaluating the benchmark difficult. This is also caused by the fact that we had not the exclusive access to the used DNS servers. Without these effects our parallel resolver should always be at least as good as DNS.

For better comparison we provide a weighted CDF[3] in Fig.2. It shows the distribution of answer times weighted by the answer time itself. This is interesting because the extremely slow answers dominate the time, that a resolver spends for resolving queries. One can see that, e.g., our approach uses 21% of the overall lookup time for requests taking longer than 1 second. DNS spends 60% and CoDNS spends almost 50% of the overall lookup time for this fraction. This result differs from the benchmarks in [PPPW04], because these are our own measurements, where we used the CoDNS version that is available and deployed in Planet-Lab. Altogether our benchmark results show that parallel DNS outperforms the other approaches as shown in Table 1. The table shows that the additional number of DNS messages is justified by roughly 50% improvement of answer time.

Since DNS gains most of its performance from caching, we performed a measurement in which we repeated some queries to exploit caching. The main benefit of this benchmark is to isolate the delay penalty of LDNS failures. A fair measurement with caching is difficult to achieve, since the chronological order of benchmarks has an influence on the result. KyoungSoo Park proposed to show the better performance of an approach by running it first. All other approaches would find better caching behavior since the first run warmed caches. This might be unfair because the result would hardly be comparable with other benchmarks. Repeating small junks of queries solved this problem, since the 2nd junk always finds a warm cache[4]. Figure 3 and Table 2 show the results. Parallel requests always reach the best results. The reason is that LDNS or network failures are the reasons for higher delay in this case. Parallel requests perfectly hide these failures while the other approaches trigger timeouts and need retransmissions.

We also learned that asynchronous socket I/O is much better than multithreading. To simplify matters, we first used threads that parallelized calls to the glibc function *send_dg*

[2]Not all of them were available all the time. Reasons can be network problems in the Internet, nodes being re-booted after OS updates, node failures, and other problems. Usually we have been able to use half of them.

[3]as used by Park et al. [PPPW04]

[4]Of course this does not work for names that are not cacheable due to TTLs of just a few seconds. But this makes the method even more realistic.

Method	lookups	total time	avg. time/lookup
DNS	23771	5164570ms	217ms
CoDNS	23771	4169531ms	175ms
parallel DNS	23771	984413ms	41ms

Table 2: Results with caching

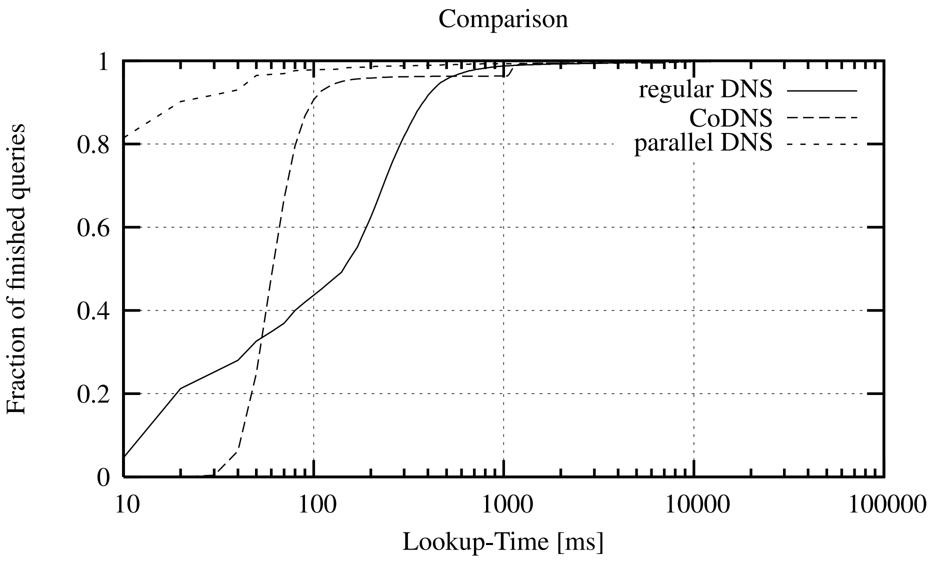

Figure 3: Comparison of average answer time with caching

that sends and receives UDP packets containing the DNS messages. When we tried to use a thread pool the performance dropped under the DNS performance in the area up to 16 ms.

CoDNS tries to pick the same peer for the same name to get some cache locality, but in the Planet-Lab configuration it uses just one peer, so this strategy might not even be used. However, CoDNS can also outperform DNS in the area between 40 and 500 ms.

Another important point for comparison is trustworthiness. Using CoDNS, the user agrees to forward DNS requests to other peers. Of course, this leaks information about the DNS names she resolves, i.e. the web sites she reads or the e-commerce services she uses. Furthermore, information about the internal structure of a company's network may also be disclosed. Using SEDNS, users can prevent this by carefully configuring their LDNS servers.

3 DNS Security Issues

Many security attacks on DNS are possible. DNS messages could be intercepted and modified on their way. This would not only be a source of confusion but would be a serious threat for many systems.

DNS Security Extensions (DNSSEC) have been designed to cope with this problem. A public key infrastructure is woven into DNS to allow using signed resource records.

With DNSSEC, the manipulation of DNS resource records is getting more difficult. An attacker can not just intercept a DNS message and change its contents. Sending a faked DNS answer directly to the stub resolver should also not be possible, since a security aware resolver, according to RFC 2535, must not trust the AD flag unless there is a secure communication channel between itself and the server, e.g., a VPN, which would prevent such attacks.

Hence, the introduction of a public key infrastructure solved many trustworthiness problems - but at the same time new problems are introduced. We have seen in Sec. 2.1 that configuration issues cause many failures. With DNSSEC the configuration of a DNS server gets even more difficult.

DNSSEC uses a hierarchical trust model that makes DNS even more fragile in the presence of configuration errors. Hence, if we want to improve the availability, DNSSEC does not seem to be the right choice. A solution that could reduce the configuration effort is a self-organizing completely decentralized peer-to-peer system. But it is already well understood, that these overlays without massive use of caching are too slow to satisfy the answer time expectations of a naming service [CMM02]. The complexity of the DNSSEC trust model can be reduced by using SSL-Certificates like described in [FPJ05]. This solutions helps to reduce DNS server dependencies, since the certification chain can be much shorter.

4 SEDNS

SEDNS tries to cope with all the dependability problems mentioned so far. We dedicate a part of our systems design to each of the problems: a local cache to improve QoS by reducing DNS latency, parallel requests to LDNS servers to mask client-side dependability problems, and a peer-to-peer approach to mask server-side dependability problems including misconfigurations and network problems. Figure 4 shows how an SEDNS resolver works in pseudocode.

Step 1 The resolver sends a query to a nearby caching-only name server. (Line 9 in Fig. 4) These servers are easy to install and the maintenance overhead is negligible. The main advantage is that the stub-resolver can learn, how long it takes to resolve a name from cache (Lines 21, 22). The time-out for this operation is dynamically adapted.

Step 2 In the case of a cache-miss, the resolver does not wait for the caching-only server to resolve the name. It proceeds to issuing parallel requests to the LDNS Servers. The earliest successful answer is delivered to the application. The number of needed DNS messages can be reduced by introducing small time-outs between transmitting the requests (Lines 10, 11). In this case the first name server should be selected randomly or using the rotate scheme in *resolv.conf* to have a better load distribution (Not shown in Fig. 4).

Step 3 In the case that no answer is obtained there must be a client-side failure or a server-side problem. The resolver now makes use of a peer-to-peer application. This service can be found using the *resolv.conf* as well[5] (Line 12, We assume that the last resolver is the peer-to-peer application). It implements the DNS protocol and is used as a gateway to the overlay network. The peer-to-peer network locates the DNS resource record and delivers it to the application, even during global DNS outages.

There have been some attempts to implement a DNS compatible service by using peer-to-peer overlay structures. Cox et al. [CMM02] simulated a Chord [SMK$^+$01] based overlay implementing a DNS compatible server using the DNS traces of Jung et al. [JSBM01]. The results show that even for a small network of 1000 Nodes the additional delay of the overlay routing leads to rather slow answers. The estimation of answer times shows that less than 30% of the DNS queries can be answered within 200ms.

Ramasubramanian et al. [RS04b] showed that proactive caching and replication can mask this problem. An rather good performance was reached using Beehive [RS04a], a framework for proactive caching. More than 50% of all DNS requests did not incur any network traffic at all. They have been served from the local cache.

Now this approach seems to be great for long-TTL entries of DNS but for entries with short TTL it would suffer from the same problems as the solution of Cox et al. [CMM02].

The common problem is that in structured peer-to-peer networks data is located using distributed hash tables (DHT). The hash sum of objects or their names is calculated and the object is located at the peer with a node ID closest to this hash sum. If an object shall be retrieved, the distance to this node ID is reduced in each routing step. This does not mean to get closer to the target, since these IDs are not linked to the location in the network. So peers having almost the same ID can still be far away from each other in terms of round trip time.

There are some approaches to optimize the routing behavior in these overlays, e.g. Castro et al. [CDHR03]. The goal to use network layer knowledge for overlay routing decisions was achieved, but the core problem remains: The location of an object is determined by a DHT. All locality information is destroyed. This might be alright as long as the object or its name does not contain location related data, but in the case of DNS this would be a wrong assumption.

[5]It has to listen at the standard DNS port to receive the query.

```
resolve(request){

 query=read(request);

 cache=readln("/etc/resolv.conf");
 for(i=0; i<resolvers_count; i++)
  resolvers[i]=readln("/etc/resolv.conf");

 scheduleQuery(cache, query, now);
 for(i=0; i<resolvers_count - 1; i++)
  scheduleQuery(resolver[0], query, cache_timeout + i*offset);
 scheduleQuery(resolver[resolvers_count - 1], query, cache_timeout + p2p_offset);

 pending = resolvers_count+1;
 while(!valid && --pending > 0){
  if ((error = select(inBuf, timeout))!=0) return error;
  response = inBuf.read();
  valid = !response.isError();
 }

 if(response.source()==cache) adapt(cache_timeout, LESS);
 else adapt(cache_timeout, MORE)
 return response.getAnswer();
}
```

Figure 4: Pseudocode of the SEDNS stub resolver

4.1 P2P overlay design

In the case, that the DNS infrastructure fails, we want to use a overlay network that efficiently locates DNS data. For this purpose we need a hash function that does not destroy any locality information.

DNS requests show locality, otherwise caching would not be as successful as it is. But the distance between two object's names or two URLs is often not determined by the round trip time between the hosts they are stored on but by the language of their content. This information is in many cases already included in the name in the shape of the country code.

While other approaches tried to exploit common keywords like Lu et al. [LHL05], we exploit the hierarchical structure of DNS. If some user looks up *www.harry-potter.uk*, it is more probable that the next lookup is *www.books.uk* instead of *www.harry-potter.fr*. So the design of our hash scheme must map this hierarchy into the hash. Therefore, we are using a prefix routing scheme, like used by Pastry [RD01] or Tapestry [HHWX01]. DNS names offer postfixes that indicate their location in the DNS name space. We directly map these postfixes to prefixes in the hash sum.

To improve the routing algorithm we exploit heterogeneity to select "Superpeers", i.e., nodes with more resources usable for routing. These Superpeers use group membership information to implement 1-hop routing.

5 Conclusion

We propose a multi-staged resolution scheme for DNS which can improve answer times by using an adaptive time-out for querying a caching-only server. Furthermore is uses different techniques to mask client-side failures and to reduce answer times at the same time. A peer-to-peer backend is only used in the case that DNS is unavailable due to client-side or server-side failures or attacks. The peer-to-peer network will be self-organizing and incrementally deployable.

The changes to the stub resolver will be implemented as glibc patch. No other software on the clients machine is necessary to install. The performance improvements will also be usable without participating in the overlay network.

References

[AMT03] Yair Amir, Daniel Massey, and Ciprian Tutu. A New Look at the Old Domain Name System, 2003.

[CDHR03] Miguel Castro, Peter Druschel, Y. Charlie Hu, and Antony Rowstron. Proximity neighbor selection in tree-based structured peer-to-peer overlays. In *Microsoft Technical report MSR-TR-2003-52*, 2003.

[CMM02] Russ Cox, Athicha Muthitacharoen, and Robert Morris. Serving DNS Using a Peer-to-Peer Lookup Service. In *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*, pages 155–165, London, UK, 2002. Springer-Verlag.

[FPJ05] Christof Fetzer, Gert Pfeifer, and Trevor Jim. Enhancing DNS Security using the SSL Trust Infrastructure. In *Proceeding of the IEEE International Workshop on Object-oriented Real-time Dependable Systems (WORDS 2005)*, 2005.

[HHWX01] Fox Harrell, Yuanfang Hu, Guilian Wang, and Huaxia Xia. Survey of Locating & Routing in Peer-to-Peer Systems, December 2001.

[JSBM01] Jaeyeon Jung, Emil Sit, Hari Balakrishnan, and Robert Morris. DNS performance and the effectiveness of caching. In *IMW '01: Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*, pages 153–167, New York, NY, USA, 2001. ACM Press.

[LHL05] Yu-En Lu, Steven Hand, and Pietro Lio. Keyword Searching in Hypercubic Manifolds. In *P2P '05: Proceedings of the Fifth IEEE International Conference on Peer-to-Peer Computing (P2P'05)*, pages 150–151, Washington, DC, USA, 2005. IEEE Computer Society.

[Nar02] Ryan Naraine. Massive DDoS Attack Hit DNS Root Servers. http://www.internetnews.com/dev-news/article.php/1486981, October 2002.

[PPPW04] KyoungSoo Park, Vivek S. Pai, Larry L. Peterson, and Zhe Wang. CoDNS: Improving DNS Performance and Reliability via Cooperative Lookups. In *OSDI*, pages 199–214, 2004.

[RD01] Antony Rowstron and Peter Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pages 329–350, November 2001.

[Ros05] Timothy Roscoe. *The PlanetLab Platform*, chapter 33. The PlanetLab Platform, pages 567 – 581. Lecture Notes in Computer Science Springer-Verlag GmbH, 2005.

[RS04a] Venugopalan Ramasubramanian and Emin Gün Sirer. Beehive: O(1) Lookup Performance for Power-Law Query Distributions in Peer-to-Peer Overlays. In *NSDI*, pages 99–112. USENIX, 2004.

[RS04b] Venugopalan Ramasubramanian and Emin Gün Sirer. The design and implementation of a next generation name service for the internet. In *SIGCOMM '04: Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 331–342, New York, NY, USA, 2004. ACM Press.

[SMK$^+$01] Ion Stoica, Robert Morris, David Karger, Frans Kaashoek, and Hari Balakrishnan. Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications. In *ACM SIGCOMM*, 2001.

[Thu01] Paul Thurrott. Microsoft Suffers Another DoS Attack. http://www.windowsitpro.com/Articles/Index.cfm?ArticleID=19770&Displ, January 2001.