Mitigating DNS DoS Attacks

Hitesh Ballani
Cornell University
Ithaca, NY
hitesh@cs.cornell.edu

Paul Francis
Cornell University
Ithaca, NY
francis@cs.cornell.edu

ABSTRACT

This paper considers DoS attacks on DNS wherein attackers flood the nameservers of a zone to disrupt resolution of resource records belonging to the zone and consequently, any of its sub-zones. We propose a minor change in the caching behavior of DNS resolvers that can significantly alleviate the impact of such attacks. In our proposal, DNS resolvers do not completely evict cached records whose TTL has expired; rather, such records are stored in a separate "stale cache". If, during the resolution of a query, a resolver does not receive any response from the nameservers that are responsible for authoritatively answering the query, it can use the information stored in the stale cache to answer the query.

In effect, the stale cache is the part of the global DNS database that has been accessed by the resolver and represents an insurance policy that the resolver uses only when the relevant DNS servers are unavailable. We analyze a 65-day DNS trace to quantify the benefits of a stale cache under different attack scenarios. Further, while the proposed change to DNS resolvers also changes DNS semantics, we argue that it does not adversely impact any of the fundamental DNS characteristics such as the autonomy of zone operators and hence, is a very simple and practical candidate for mitigating the impact of DoS attacks on DNS.

Categories and Subject Descriptors: C.4 [Performance of Systems]: Reliability, Availability.

General Terms: Reliability, Security.

Keywords: DNS, Denial of Service, stale cache.

1. INTRODUCTION

In the recent past, there have been many instances of flooding attacks on the Domain Name System (DNS) aimed at preventing clients from resolving resource records belonging to the zone under attack [26-29]. While these attacks have had varying success in disrupting the resolution of names belonging to the targeted zone, the threat posed by them to DNS operation is obvious. As a matter of fact, DNS's pivotal role as a precursor to almost all Internet services implies that such attacks represent a severe threat to the Internet in general.

In response to such attacks, some of the DNS root-servers and top-level domain (TLD) servers have been replicated through IP Anycast [10]. Lately, a number of research efforts have proposed new architectures for the Internet's naming system. The key insight behind these proposals is to decouple the distribution of DNS data from the hierarchy of authority for the data [8,9]. Once this decoupling is done, several mechanisms can be used to make the data distribution infrastructure highly robust and to ensure its availability in the face of attacks. For instance, efforts arguing for centralized data distribution [8] and peer-to-peer based data distribution [7,9,22,24] represent the two extremes of the design space for such a robust distribution infrastructure.

However, we are not convinced of the need for a new DNS architecture involving a new dissemination mechanism to ensure DNS operation when nameservers are unavailable. Rather, we argue that a complementary and a much more modest tack to handle DoS attacks on DNS infrastructure is to do away with the need for 100% availability in the existing architecture. In this paper, we follow this argument and show that the need for nameserver availability in the *existing DNS framework* can be reduced simply through a minor modification in the caching behavior of DNS resolvers.

Today, DNS resolvers cache the responses they receive from nameservers to improve lookup performance and reduce lookup overhead. A resolver can use the cached responses to answer queries for a duration specified by the *time-to-live* (TTL) value associated with the response. We propose to modify the operation of resolvers such that they do not expunge cached records whose TTL value has expired. Rather, such records are evicted from the cache and stored in a separate "*stale cache*". Given a query that cannot be answered based on the cached information, resolvers today traverse down a hierarchy of DNS zones by querying the authoritative nameservers for the zone at each step. However, this resolution process fails if all the nameservers for the zone at any step of this traversal are unavailable. In such a scenario, we allow resolvers to use the information stored in their stale cache to answer the query for the unavailable zone and thus, allow the resolution process to continue.

Modifying DNS resolvers as specified above results in normal DNS operation when resolvers are able to access nameservers; only when all the nameservers for a zone do not respond to the queries from a resolver does the resolver resort to using records for the zone from its stale cache (*stale records*). This modification implies that DNS resolvers store the part of the global DNS database that has been accessed by them and use it when the relevant DNS servers are unavailable. Consequently, while attackers may be able to flood nameservers and overwhelm them, resolvers would still

CCS'08, October 27–31, 2008, Alexandria, Virginia, USA.

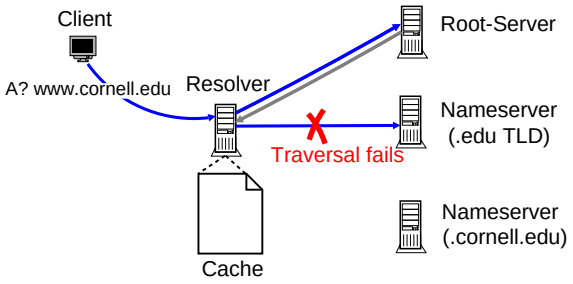


Figure 1: Traversal down the DNS hierarchy during the resolution of the A-record for *www.cornell.edu* fails if the *.edu* TLD nameservers are under attack.

have the stale records to rely upon. To this effect, this paper makes the following contributions:

- We present a simple modification in the caching behavior of DNS resolvers that would make nameserver availability less critical than it is today and hence, mitigate the impact of DoS attacks on DNS infrastructure.
- We discuss some details concerning the implementation of a stale cache in a DNS resolver. Further, our scheme has a number of practical advantages with regards to protection against flooding attacks that we discuss in section 4.1.
- We analyze a 65-day DNS trace to quantify the benefits of having a stale cache under different attack scenarios and find that the stale cache can be used to resolve a significant fraction of client queries even under severe attacks of long duration.
- Using trace-based simulation, we determine the memory footprint of the stale cache and find that maintaining even a month's worth of stale records requires a small amount of memory.
- While DNS resolvers rely on their stale cache *only* when the relevant nameservers are unavailable, the fact that the TTL-value for stale records has expired implies that it is possible that these records may not be the same as those returned by the actual nameservers (had they been available). We use the aforementioned trace to quantify this possibility and find that the probability of inaccurate records being returned in case of an attack is very small ($<$0.5%).

On the flip side, our proposal changes DNS semantics. For example, zone owners cannot expect the records served by their nameservers to be completely evicted by all resolvers within one TTL period. We analyze problems that may arise due to such semantic changes; the impact of this and other drawbacks of our scheme are discussed in section 4.2. This analysis leads us to conclude that the scheme does not adversely impact any of the fundamental DNS characteristics such as the autonomy of zone owners. Hence, we believe that the proposed resolver modification represents a very simple and practical candidate for alleviating the impact of DoS attacks on DNS.

2. A SIMPLE IDEA

2.1 DNS Resolvers Today

Clients rely on DNS primarily to map service names to the IP addresses of the corresponding servers. Typically, clients issue their queries to a local DNS resolver which maps each query to a matching resource record set (hereon simply referred to as a matching record) and returns it in the response.[1] Each record is associated with a time-to-live (TTL) value and resolvers are allowed to cache a record till its TTL expires; beyond this, the record is evicted from the cache. Given a query to resolve, a resolver executes the following actions[2]:

1. Look up the cache for a matching record. If a matching record is found, it is returned as the response.
2. If a matching record is not found in the cache, the resolver uses the DNS resolution process to obtain a matching record. This involves:
   (a) Determine the closest zone that encloses the query and has its information cached (if no such zone is cached, the enclosing zone is the root zone and the resolver resorts to contacting the DNS root-servers). For example, given an *A*-record query for the name *www.cornell.edu*, the resolver determines if records regarding the authoritative nameservers for the zones *.cornell.edu*, or *.edu* (in that order) are present in its cache.
   (b) Starting from the closest enclosing zone, traverse down the DNS zone hierarchy by querying subsequent sub-zones until the zone responsible for authoritatively answering the original query is reached or an error response from a zone's nameservers implies that the traversal cannot proceed. In either case, the resolver returns the appropriate response to the client. Also, all responses (including negative responses indicating error) during this resolution process are cached by the resolver.
3. In case the resolution process in (2.b) *fails* due to the inability of the resolver to contact all the nameservers of the relevant zone at any step of the traversal, return a response indicating the failure. Note that the term "failure" refers only to the scenario when the traversal is not completed due to the unavailability of the nameservers of a zone. Figure 1 illustrates this scenario.

2.2 DNS Flooding Attacks

We consider DoS attacks on DNS servers where attackers flood the nameservers of a zone to disrupt the resolution of records belonging to the zone and consequently, any of its sub-zones. In general, *flooding attacks* aimed at denying service to clients take advantage of the skewed distribution of functionality between clients and servers. In the case of DNS, the fact that the nameservers for a zone are completely responsible for serving the zone's records and in turn, for the operation of any sub-zones implies that their availability is critical and makes them an attractive target for flooding attacks.

2.3 Proposed Resolver Modification

We argue that changing the caching behavior of DNS resolvers so that they shoulder more of the resolution burden, especially when nameservers are unavailable, is an effective way to address DNS flooding attacks. Further, such a modification is possible within

[1]Note that the matching record may not answer the query; for example, it may reflect an error condition due to which the query cannot be answered. Hence, the term "response" includes both positive and negative responses.

[2]This is a simplification of the algorithm used by resolvers but suffices for the purpose of exposition. See [14] for a more detailed version.

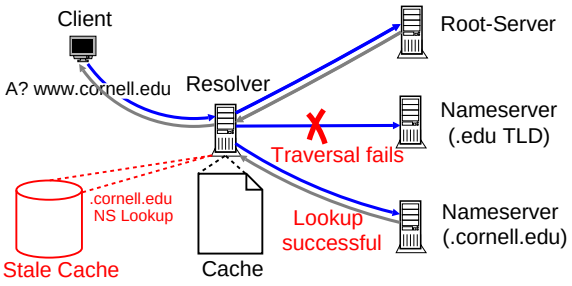


Figure 2: Resolution of the A-record for *www.cornell.edu* succeeds: a stale NS record for *.cornell.edu* allows the traversal to continue even though the *.edu* TLD nameservers are inaccessible.

the existing DNS framework. To this effect, DNS resolvers should store the responses of the queries they resolve beyond the TTL values associated with the respective responses and use stale information if all the authoritative nameservers for a zone are unavailable. Thus, the resolvers have the stale information to rely on, in case the authoritative servers for a zone are overwhelmed due to a flood of requests. More concretely, we propose the following change in the operation of DNS resolvers–

Stale Cache: Resolvers do not completely expunge cached records whose TTL value has expired. Rather, such records are evicted from the cache and stored in a separate *stale cache*. In effect, the stale cache together with the resolver cache represents the part of the global DNS database that has been accessed by the resolver.

Resolving Queries: In our proposal, the first two steps executed by a resolver when resolving a query are the same as before. Hence, given a query, the resolver attempts to respond to it based on the cached information or through the resolution process. The third step is modified as follows:

3) In case the resolution process in (2.b) fails due to the inability of the resolver to contact all the nameservers of the relevant zone at any step of the traversal, search the stale cache for the required record. If such a record is found, the resolution process in (2.b) can continue based on this stale record. Figure 2 illustrates this scenario.

This modification implies that when (and only when) the authoritative nameservers for a zone are unavailable, the resolver can resort to using responses from a previously resolved query.

Stale Cache clean-up: Existing resolvers cache the responses to the queries made during the resolution process in step (2.b). In our proposal, these responses are also used to evict the corresponding stale records from the stale cache. For example, during the resolution of the *A* record for the name *www.cornell.edu*, the resolver may query the authoritative nameservers of the zone *.edu* for the authoritative nameservers of the sub-zone *.cornell.edu*. When a response containing records regarding these nameservers is received, it is cached and is also used to evict any nameserver records for *.cornell.edu* present in the stale cache. Note that this newly cached response will be evicted to the stale cache upon expiration of its TTL value. Also note that all responses (including negative responses) are used to evict the stale cache. For example, a NXDOMAIN response from the nameserver for *.edu* indicating that the sub-zone *.cornell.edu* no longer exists will also lead to eviction of the existing nameserver record for *.cornell.edu* in the stale cache. Hence, this clean-up process ensures that a record stored in the stale cache always corresponds to the latest authoritative information that the resolver received.

2.4 Stale Cache Details

From an implementation point of view, a resolver can perform steps (2.b) and (3) of the query lookup concurrently. For instance, continuing the earlier example, while the resolver queries the zone *.edu*'s nameserver for the nameservers of the sub-zone *.cornell.edu*, it can lookup its stale cache for information regarding the nameservers for *.cornell.edu*. As mentioned earlier, the information from the stale cache is used only if the resolver is unable to contact all the nameservers for *.edu* and hence, the latency of the stale cache lookup is not critical. Consequently, the stale cache can even be maintained on the resolver's disk. However, as we show in section 3.3, even a month's worth of stale records require a small amount of storage space and hence, we envision resolvers maintaining their stale cache in memory.

3. EVALUATION

In order to evaluate the advantages of a stale cache, we collected DNS traffic at the link that connects the Cornell Computer Science department's network to the Internet. The network comprises of ≈1300 hosts. The trace was collected for a period of 65 days – from 21^{st} Nov, 2007 to 24^{th} Jan, 2008. It consists of 84,580,513 DNS queries and 53,848,115 DNS responses for a total of 4,478,731 unique names. Each collected packet was anonymized to preserve the privacy of the network's clients. This included anonymizing the source and destination IP addresses and the names and addresses in the DNS part of packet. The fact that the trace was collected at the network's border router and not at the resolvers (i.e., the caching nameservers) that reside inside the network implies that we do not see all the queries generated by clients. Specifically, client queries that can be answered based on the cached contents of the resolvers do not appear in our trace. This quirk of the collection process has important implications for the results presented here and we discuss these later in the section.

Given the trace, we can simulate the operation of a stale cache serving clients in the network under different attack scenarios. Such a simulation is governed by two key parameters:

- *Stale cache size*: A stale cache size of *x* days implies that stale records are kept in the stale cache for a maximum of *x* days. In our simulations, we vary the stale cache size from 1 to 30 days. Further, in section 3.3 we measure the actual memory footprint for a stale cache of *x* days.

- *Attack duration*: This allows us to evaluate the operation of the stale cache under attacks of varying durations. For any given type of attack, we simulate the attack lasting for a duration of 3, 6, 12 and 24 hours.

Hence, to simulate the operation of a 7-day stale cache under an attack lasting 3 hours, we populate the stale cache using the DNS queries and responses in the first 7 days of the trace. We then simulate an attack every 3 hours while ensuring that the stale cache contains trace data for the past 7 days. This allowed us to have 464 simulation runs ((65-7) days * 8 simulations per day) for a 3-hour attack while using a 7-day stale cache. Thus, we were able to simulate a number of attacks for any given stale cache size and attack duration.

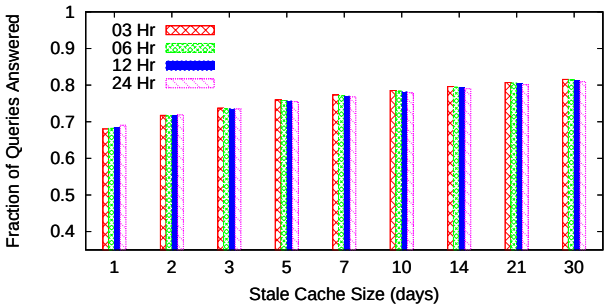


Figure 3: Fraction of Queries Answered using a stale cache of varying size during an attack wherein none of the nameservers are operational.

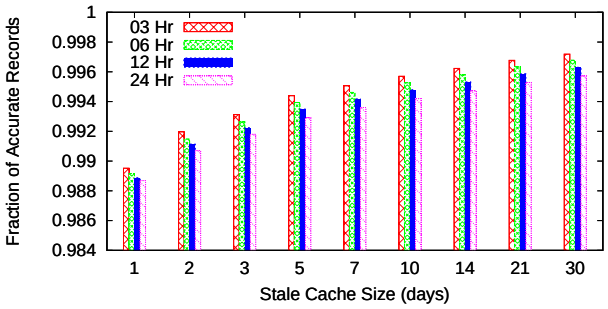


Figure 4: Fraction of Accurate Records in responses based on a stale cache of varying size during an attack wherein none of the nameservers are operational.

3.1 Is history useful?

We wanted to determine if there is any value to maintaining historical information in the form of DNS records beyond their TTL-values. To this effect, we consider an attack wherein none of the DNS nameservers are operational and hence, all queries that cannot be answered based on the information cached at the resolvers rely on the simulated stale cache. Note that this is does not represent a realistic flooding attack; instead, the objective here is to use an extreme scenario to test the limits of the value of keeping around stale DNS information.

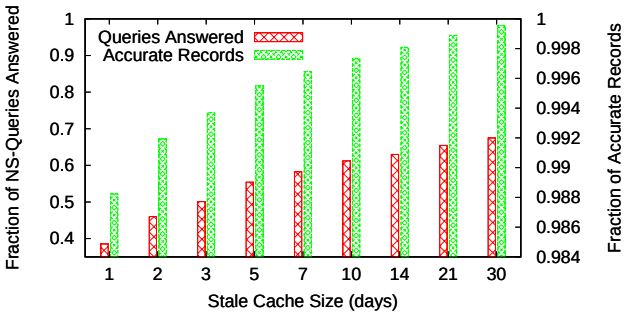
We simulated the attack scenario described above for varying attack durations and varying stale cache sizes. Here we focus on those queries that cannot be answered based on the resolver cache. Figure 3 plots the fraction of such queries that can be answered based on the stale cache. The figure shows that a 1-day stale cache can be used to answer 68% of such queries over the course of a 3-hour attack. The fraction of queries answered increases with the stale cache size; for instance, a 3-day stale cache can answer 73.7% and a 14-day stale cache can answer 79.6% of the queries. However, increasing the stale cache size beyond 14 days yields diminishing returns; for instance, a 21-day stale cache can answer 80.7% and a 30-day stale cache can answer 81.5% of the queries.[3] Past studies have found that the popularity of DNS names follows a zipf distribution [11] and the diminishing returns from increasing the stale cache size appear to be a consequence of this.

The variation of the fraction of queries answered with the attack duration for a given stale cache size is a little more complicated. For a small stale cache (<=2 days), the fraction of queries answered increases with attack duration. While non-intuitive, this can be explained based on the facts that, 1). for attacks of short-duration, many queries can be answered based on the resolver's cache and 2). the focus here is on queries that can be answered using the stale cache. Consequently, for an attack lasting 3 hours, many queries can already be answered based on the resolver's cache and the probability that a query whose answer is not cached can be answered based on the stale cache is small. For attacks of longer duration, most of the cached records have expired and hence, the stale cache is able to answer more queries.

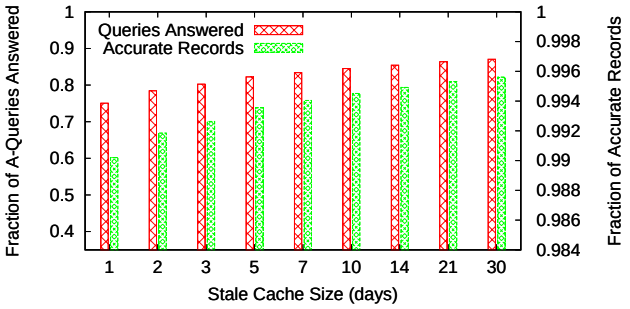
However, this effect diminishes for larger stale caches. The figure shows that for larger stale cache sizes, there is a small reduction in the fraction of queries answered as the attack duration increases. For instance, a 14-day stale cache can be used to answer 79.4% of the queries during an attack lasting 6 hours and 79% of the queries during an attack lasting 24 hours.

As mentioned earlier, an important thing to note is that the trace does not include queries that are answered based on the cached contents of the network's resolvers. This implies that the numbers regarding the fraction of queries answered (and similar numbers in the rest of this section) vastly underestimate the actual fraction of client queries that succeed in case of an attack with a stale cache in place.

However, the fact that the TTL-value for the stale records has expired implies that responses to client queries based on the stale cache may not be the same as the responses that would be received in case the actual nameservers were operational. This leads to the notion of *accurate* and *inaccurate* records. Note that using an inaccurate record in the resolution process does not necessarily imply that the name being queried is resolved to a wrong address. In-



(a) NS-queries



(b) A-queries

Figure 5: For (a) NS-queries and (b) A-queries, Fraction of Queries Answered and Accurate Records when using a stale cache during an 3-hour attack.

[3]For clarity, the X-axis in figure 3 and the figures in the rest of this section is limited to some chosen stale cache sizes.

stead, in spite of the use of an inaccurate record, a name may be resolved properly or may not be resolved at all – we discuss these possibilities and their implications in section 4.2.

Our trace-based simulation allows us to determine the accuracy of the DNS records in responses that utilize the stale cache. Specifically, for each query received during a simulated attack, we compare the response based on the stale cache and the actual response from the nameserver had it been accessible (we get this information from the trace) and all matching records are counted as accurate records. Figure 4 plots the fraction of accurate records for varying stale cache size and attack duration. The accuracy percentage increases with increasing stale cache size. This results from the fact that as the stale cache increases in size, it can answer more and more queries for NS records that tend to be more stable and hence, the increase in accuracy.[4] However, the increase tapers off beyond a stale cache size of 10-14 days; a 10-day stale cache yielded 99.6% accurate records during a 3-hour attack. Also, the fraction of accurate records reduces by a small amount as the attack duration increases.

Next, we focussed on different kinds of queries. Specifically, we studied queries for NS-records (i.e. NS-queries) and queries for A-records (i.e. A-queries) and determined the fraction of such queries that can be answered using the stale cache and the accuracy of the corresponding stale records. In case of an attack lasting 3-hours, the values for these fractions are plotted in figure 5. The figures show that while the fraction of queries answered increases with increasing stale cache size in both cases, the fraction of NS-queries answered is much less than the fraction of A-queries answered. For instance, a 14-day stale cache can answer 63% of NS-queries and 85.4% of A-queries.

This results from the fact that NS-records tend to have higher TTL values as compared to A-records (especially A-records for names not belonging to nameservers). Consequently, most of the NS queries can be answered using the resolver cache. Further, if a NS-query cannot be answered through the resolver cache, it is more likely that the corresponding NS-records weren't queried for in the past and hence, would not be present in the stale cache too. This also implies that the fraction of NS-queries answered hits the point of diminishing returns much later than the fraction of A-queries answered. The figure also shows that, as expected, the accuracy of NS-records is higher than that of A-records. In both cases, the accuracy of the stale records returned to clients increases with the stale cache size and is >99.5% with a stale cache of more than 10 days.

Overall, these results show that even in the extreme attack scenario considered here, the stale cache can answer a significant fraction of the client queries in a surprisingly accurate fashion.

3.2 Performance under different attack scenarios

We now evaluate the performance of the stale cache under three different attacks scenarios. The first attack involves the root-servers not being accessible to the clients. Today, such an attack would cause any queries for NS records corresponding to the top-level domains (TLDs) to fail.[5] However, in case of the trace-based simulation of a stale cache, all such queries succeeded. This is because, for the query and response pattern captured in our trace, the NS records for all the TLDs were present either in the cache or the stale cache at all times. Thus, the stale cache would have ensured that all

[4]We explain the increase in the fraction of NS-queries answered later in the section.

[5]This assumes that the NS records are not present in the cache of the network resolvers or have expired since the attack started.

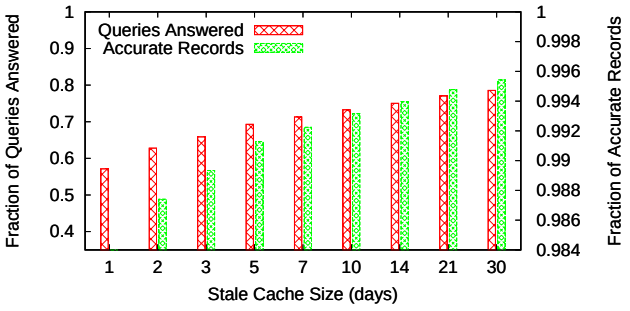


Figure 6: Fraction of Queries (for two-level names) Answered and Accurate Records when using a stale cache during an 3-hour attack on the TLD nameservers.

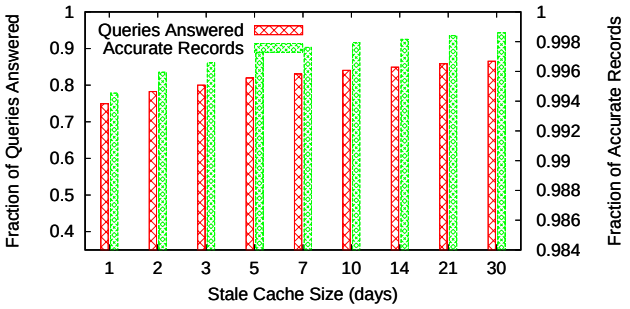


Figure 7: Fraction of Queries (for three-level names) Answered and Accurate Records when using a stale cache during an 3-hour attack on second-level nameservers.

names would still resolve and hence, would effectively shield the network from an attack on the root-servers.

The second attack involves clients not being able to access TLD nameservers. Today, this would cause queries for any records corresponding to two-level names such as *a.com* to fail. Further, any queries for longer names that rely on the resolution of two-level names would fail too. Here we restrict ourselves to the queries for two-level names that cannot be answered based on the resolver cache. Figure 6 plots both the fraction of such queries that the stale cache can answer and the fraction of records in these responses that are accurate in case of a 3-hour attack. The trends for longer duration attacks are similar. The fraction of queries answered increases with an increasing stale cache size though it tapers off for a stale cache of more than 14 days. A 14-day stale cache can answer 75% of the queries for two-level names. The reason for the lower fraction of queries answered is that clients typically access the NS records for names such as *a.com* and these records tend to have a high TTL-value. As explained earlier, this implies that most such queries are answered based on the resolver cache and if a record is not cached, there is a higher probability that it has not been queried at all and hence, is not present in the stale cache too. Of course, the fraction of total client queries that succeed when using the stale cache is much higher. The graph also shows that records from a 14-day stale cache are 99.4% accurate and accuracy increases with the stale cache size too.

Similarly, the last attack scenario involves second-level nameservers being inaccessible. This would cause queries for any records corresponding to a three-level name such as *b.a.com* to fail. We focus on such queries that are not cached by the network's re-

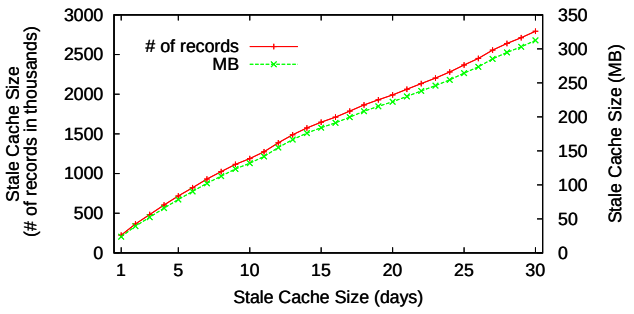


Figure 8: Stale cache memory footprint

solvers. Figure 7 plots the fraction of such queries answered using stale records during a 3-hour attack and the accuracy of these stale records. As before, the fraction of queries answered increases with an increasing stale cache size. However, in this case, the returns from increasing the stale cache size are diminished much sooner that the previous attack scenario. A 14-day stale cache can answer 85% of the queries for three-level names. Both A and NS records for names such as *b.a.com* are accessed by clients and these tend to have lower TTL-values than the records for two-level names. This explains the higher percentage of queries answered. The graph also shows that records from a 4-day stale cache are 99.8% accurate.

3.3 Memory Footprint

We now evaluate the memory requirements of the stale cache. Figure 8 plots both the number of DNS records and the actual memory used by a stale cache of size 1 to 30 days. As one would expect, the memory requirements of the stale cache increase as the number of days increase. Note that the simulated stale cache stores DNS records without any encoding and hence, there is scope for further reducing the memory required for the stale cache. More importantly, the figure shows that even for a network with 1300 hosts and a query-response rate of $\approx$25 DNS packets per second, the stale cache memory footprint is very small. For instance, maintaining stale records for a period of 30 days given the query pattern in our trace requires $<$313MB of storage space.

Of course, the stale cache memory requirements depend on the number of clients being served by the resolver and their query patterns. Also, the evaluation in the previous section shows that the gains to be obtained from stale records older than two weeks are minimal. These factors suggest that, in practice, resolvers will keep stale records only for a configurable number of days, for example stale records for the past couple of weeks. Further, the resolver will be limited to at most a certain amount of memory for the stale cache. In case the stale cache fills up, the resolver would evict records based on some criterion. For instance, the resolver could use the query pattern of clients to evict the least recently used DNS records (LRU eviction). However, given the amount of memory on modern machines, we believe that resolvers should easily be able to maintain a stale cache containing records for a couple of weeks and there shouldn't be a need for more complex eviction algorithms. In section 4.2 we discuss how placing a limit on the duration for which stale records are kept addresses some of the practical concerns arising out of the use of stale information.

4. DISCUSSION

There have been a number of "clean-slate" proposals to make the availability of specific nameservers less critical for the operation of Internet's naming system. These proposals [7-9,12,22,24] decouple the ownership of names from the task of distributing them and try to architect a robust mechanism for distributing the names. However, such an approach could increase the total DNS overhead many times over, especially in the face of the use of DNS for load balancing purposes. On a more general note, while most of us agree that DNS is afflicted by a few problems, we think that a majority of them can be attributed to misconfigurations, improper implementations, violations of best current practices, or even a lack of motivation to address them and not to major architectural flaws. For example, problems regarding high lookup latency can mostly be attributed to misconfigurations (i.e. broken and inconsistent delegations) [22] and the long timeouts used by resolvers in case of errors [19]. Consequently, despite a number of proposals arguing to the contrary, we do not see a pressing need for an architectural change. Guided by this observation, our proposal represents an exercise in showing how minor operational modifications can address DNS problems; specifically, modifying the caching behavior of DNS resolvers can reduce the impact of flooding attacks on DNS.

In the rest of this section we discuss the advantages of the proposed modification and a few possible objections to it.

4.1 Pros

DNS Robustness. The proposed modification ensures that resolvers can respond to queries for a zone even if the zone's authoritative nameservers are unavailable, assuming that the resolver has queried the zone at some point in past and the previous response is present in the resolver's stale cache. The evaluation in the previous section showed that a stale cache can indeed make DNS more robust to DoS flooding attacks. Further, while past attempts such as the anycasting of DNS nameservers provide nameserver operators with a mechanism, albeit a very expensive one, to protect the name resolution for their zones, our modification represents an insurance policy that can be adopted by the resolver operators and hence, provides some control to the client.

Simplicity. The biggest argument in favor of the stale cache as a means of increasing DNS robustness is its simplicity. The proposed scheme:

- *Does not change the basic protocol operation and infrastructure*; only the caching behavior of resolvers is modified.
- *Does not impose any load on DNS*, since it does not involve any extra queries being generated.
- *Does not impact the latency of query resolution*, since the stale cache is utilized only when the query resolution fails.

Incremental Deployment. Any single resolver can adopt the modifications proposed in this paper and achieve significant protection from attacks against the DNS servers it and its clients access. Hence, the proposal can be incrementally deployed.

Motivation for Deployment. Modifying a resolver is beneficial for the clients being served by the it since the resolver can resolve queries for zones that have been accessed by it in the past even if the nameservers for the zones are not available. Hence, there is motivation for the resolver operators to switch to the modified resolver.

4.2 Objections

DNS caching semantics and the possibility of inaccurate information being used. The biggest objection against the proposed modification is that it changes the semantics of DNS caching. With the current DNS specifications, a zone operator can expect the reco-rds served by the zone's authoritative nameservers to be completely

expunged by resolvers within TTL seconds. With our proposal, such records would be evicted to the stale cache. The problem with such an approach is best explained through an example. Let's consider a zone whose records have been updated. Also, consider a resolver that has accessed the zone but not since the update and so the zone's records in the resolver's stale cache are obsolete or inaccurate. Given this, if the resolver needs to resolve a query for the zone at a time when all the zone's authoritative nameservers are unreachable, it would resort to using the inaccurate records present in its stale cache.

The problematic scenario described above arises only when two conditions are met:

1. The DNS records for the zone in question have been updated since the last access by the resolver.

2. The nameservers for the zone are currently inaccessible.

Condition (1) can arise due to several reasons: for instance, the nameservers for a zone have been moved, the service itself has migrated or there have been address space changes, DNS based load-balancing across the nameservers or the application servers, etc. We consider these below.

First, if the nameservers have been moved, the name resolution may fail while if the service migrates, the name may be resolved to the wrong address. Both these are undesirable scenarios. However, restricting the duration for which resolvers can keep records in their stale cache helps us avoid these. Specifically, to account for this, a nameserver/service needs to be run on both its old and new address for a couple of weeks after migration. This allows for the old records to be flushed from the stale cache of resolvers. Note that zone operators anyway need to do this today since a large number of misbehaving resolvers disregard TTL values and use expired records even when the nameservers for a zone are available [32,34].

Second, if the DNS records have been changed to balance client load, the name would probably resolve properly but this might interfere with the load across the servers. In a recent study, Poole et. al. [21] found that name-to-IP mappings tend to be very stable with less than 2% of DNS names changing IP addresses more than once a week. Further, most of these names can be attributed CDNs like Akamai trying to balance client load across their servers.[6] This implies that not only is condition (1) rare, in a vast majority of cases where it does occur, using the stale records would not lead to wrong resolution. While this is far from perfect, the small possibility of load imbalance across the servers when they are under attack (in which case the load balancing isn't working anyway) seems like a small price to pay for the robustness offered by a stale cache. Also, the possibility of a resolver using inaccurate records for a zone is much less for zones that the resolver frequently accesses.

Further, resolvers may choose to apply the modified caching sch--eme to infrastructure records only. Infrastructure records, as defined by [17], refer to records used to navigate across delegations between zones and include the *NS* records (and the corresponding *A* records) for zones. Past studies show that such records change even more infrequently [9,17] than other DNS records and hence, this would further reduce the possibility of resolvers using inaccurate records while still providing a large robustness gain.

Finally, it is also possible to make changes on the client-side DNS software to make applications aware of the use of stale records. A resolver could use the RCODE field in the DNS header (a 4-bit field; values 5-16 for this field have been reserved for future use [14]) to inform the querying client that the response is based on the stale cache. Similarly, the client `gethostbyname` and the relevant `libresolv` functions could be modified to interpret the new RCODE value and inform applications of the same. With these changes in place, applications would have the flexibility of being able to account for the possibility of inaccurate records and decide whether to use stale records or not based on application semantics and/or user choice. However, most applications that need to make sure that they are accessing the right resource use application-specific authentication anyway; for instance, financial web-sites commonly use personalized site-keys for this purpose [33]. This, combined with the fact that the possibility of stale records being inaccurate (especially ones that lead to wrong resolution of names) is miniscule, implies that we don't feel that the overhead of modifying the DNS-software at all clients is justified.

Autonomy for zone operators. Another important concern is that the proposed modification would seem to move autonomy away from zone operators to resolver operators. Allowing resolvers to store records after their TTL value has expired suggests that zone operators do not control the access to their sub-zones; for instance, they could not kill off their sub-zones when they wish to.

However, this is not the case. The fact that we don't modify DNS's hierarchical resolution process implies that resolvers still need to go through the nameservers for a zone in order to access its sub-zones and hence, the autonomy of zone operators is not affected. For instance, let's assume that the operator for the zone *.com* needs to kill off the sub-zone *.rogue.com*. Typically, this would involve *.com*'s zone operator configuring the zone's authoritative nameservers to respond to any queries regarding *.rogue.edu* with a NXDOMAIN, implying that no such domain exists. Consequently, a resolver trying to resolve a query like the *A* record for *www.rogue.- -com* by traversing down the DNS zone hierarchy would receive a NXDOMAIN response from one of the *.com* nameservers and would forward this to the client that originated the query. Further, this response would be cached and eventually be evicted to the stale cache. Thus, if there are any such future queries at a time when all the *.com* nameservers are unavailable, the resolver would still return a NXDOMAIN response.

Attackers attempting to force the use of inaccurate information. Apart from the possibility of inaccurate data being used, there is also the possibility of attackers taking advantage of the stale cache maintained by resolvers to force the use of inaccurate records. Attackers may keep track of updates to the records of a zone and start flooding the authoritative nameservers for the zone as soon as some of the records are updated. If the attack overwhelms the zone's nameservers, resolvers trying to resolve the zone's records would rely on the obsolete data stored in their stale cache. In effect, attackers can now flood the nameservers for a zone in order to delay the propagation of updates to the zone's records for the duration of the attack. While we cannot imagine many cases where such an attack could be used, one scenario where it does appear to be harmful is to undermine the autonomy of zone-operators. In the example above, the owners of the *.rouge.com* zone may flood the *.com* nameservers to force the use of stale NS records for their zone and hence, prevent their zone from being killed. The bigger problem here is that there is incentive to flood the nameservers of a zone to prevent sub-zones from getting killed. This problem captures an inherent trade-off that the use of stale records exposes: when a zone's nameservers are being flooded, all sub-zones, including sub-zones that were deleted in the recent past, are accessible. While this is certainly a serious concern, it is important to note that the sub-zones will stay alive only as long as the zone's nameservers are inaccessible. Given that measures to counter flooding attacks on

[6]The fact that the actual DNS names in our trace have been anonymized implies that we cannot determine if the changed mappings observed by us can also be attributed to CDNs.

nameservers, such as filtering by ISPs, are usually applied within a day or two of the attack, the sub-zones would be able to stay alive for not too long a duration.

Privacy Concerns. With our proposal, DNS resolvers store DNS records long beyond their TTL-values. This leads to privacy concerns in case the resolver is broken into. Specifically, if a resolver were to be compromised, the attacker would gain access to all the stale cache records and hence, would have a heap of information about what the resolver's clients have been querying and in turn, their web-access patterns. However, the stale cache would not provide the attacker with information about queries from individual clients. Also, this is certainly no worse than other DoS mitigation proposals that require DNS resolvers to query entities other than a zone's authoritative nameservers to resolve the zone's records and hence, leak out private information as an integral part of their operation.

Resolution latency in the face of an attack. In our proposal, if a resolver is unable to reach the authoritative nameservers of a zone, it resorts to using the zone's records in the stale cache. Consequently, the resolver must query each of the nameservers for the zone, wait for the query to timeout (and possibly retry) before it can use the stale cache. With the current timeout values used by resolvers, this would entail a high lookup latency in the face of attacks (i.e. when the nameservers for a zone are unavailable). For example, the default configuration for the BIND8 resolver [31] involves sending queries to each nameserver for 30 seconds with an exponentially increasing period between consecutive retries. So, clients accessing a zone with two authoritative nameservers at a time when both of them are unavailable would need to wait for 60 seconds before receiving a reply. However, most resolvers allow the retry and timeout values to be configured and hence, the lookup latency problem can be solved by using aggressive values for these timers. As a matter of fact, past work has already suggested that these timer values are major contributors to the high lookup latency when errors are encountered [19].

DoS'ing the application servers. The proposed modification does not reduce the vulnerability of nameservers to DoS attacks. Consequently, attackers can still flood them so that they are unable to serve (and update) the records of the corresponding zones. Rather, the modification makes the availability of DNS nameservers less critical and hence, significantly reduces the impact of DoS attacks on DNS.

Further, the proposal does not address the general DoS problem and attackers can deny service to clients by attacking the application servers instead of the corresponding DNS nameservers. As a matter of fact, a flooding attack that chokes the network bottleneck for a zone's nameservers is also likely to hamper the availability of the zone's application servers. In such a scenario, there isn't much value to being able to resolve the names for the application servers since clients would not be able to reach them anyway.[7] In effect, this concern boils down to how common is it for application servers and their nameservers to share a network bottleneck. We intend to measure this for nameservers on the Internet as part of our future work.

Interaction with DNSSec. The proposal does not have any harmful interactions with or implications for DNSSec. In case the resolver cannot reach the nameservers of a zone and relies on the corresponding records in the stale cache, the records ought to be classified as "Undetermined" by the resolver.[8] Hence, any DNSSec policies expressed by the resolver operator for undetermined records naturally apply to the stale records.

[7]Note that there is still a lot of value to being able to access the sub-zones when a zone's nameservers are being flooded. For example, being able to access the rest of the name system when the root-servers are being flooded.

5. RELATED WORK

A number of recent efforts [7-9,22,24] have proposed new architectures for the next generation Internet naming system that address DNS's performance and robustness problems. Other proposals to change the DNS architecture include multicasting the global DNS database to specialized servers to reduce the response time for clients [12] and augmenting the DNS structure with additional pointers that can be used to access sub-zones and hence, increase DNS robustness against flooding attacks [25]. [20] argues for taking advantage of site multihoming by spreading the identity of end hosts and rate-limiting name resolution requests to mitigate DoS attacks. Balakrishnan et al. [1] propose to replace the hierarchical DNS (and URL) namespace with flat identifiers. We show that a minor operational change to resolvers in the *existing DNS framework* can significantly mitigate the impact of DoS attacks on DNS.

The use of caches and more generally, of stale data to improve system availability shows up in many aspects of computer science. Examples include using stale data to improve availability of services [13] and even shared memory multiprocessors [23]. [15] proposes and evaluates the use of stale data to reduce the measurement overhead for placement of services on the Internet. This paper evaluates the efficacy of stale data in increasing DNS availability.

Pappas et al. [17] argue for the use of long TTL values for infrastructure DNS records as a means of alleviating the impact of DoS attacks on DNS. We share with their proposal the basic notion of using records already present in the resolver cache for a longer period. While our proposal involves changing the caching behavior of resolvers, using longer TTL values for a zone's records involves a minor configuration change at the zone's nameservers and hence, does not necessitate any software update. However, using long TTL values represents a technique that can be used by nameserver operators. Also, long TTL-values make it harder for operators to update their records. In subsequent work [18], the authors augment their proposal and argue for resolvers proactively renewing the infrastructure records present in their cache as a means of mitigating attack impact. This scheme has an important advantage over the use of stale records: it does not modify DNS caching semantics. However, as shown in [18], proactive renewal of DNS records by resolvers, when used in isolation, increases DNS traffic many times over. Further, the overhead of such an approach implies that it cannot be used for non-infrastructure DNS records, a large fraction of which don't change very rapidly.

In past work [2], we discuss the use of stale DNS records as a DoS mitigation mechanism. This paper follows up on that proposal and quantifies the advantage of a stale cache and the possibility of using obsolete information through trace-based simulations. Nonamed [35] is a quasi DNS resolver that provides users the option of using stale DNS information which maybe be useful for operation when disconnected from the Internet. We argue for the use of a zone's stale records only when all nameservers for the zone are unavailable. Cohen and Kaplan [6] propose the use of stale DNS records for improving DNS performance. This involves fetching data based on the stale records and issuing a DNS query to refresh the stale record concurrently. CoDNS [19] is a cooperative DNS lookup service designed to alleviate client-side DNS problems. We share with their proposal the notion of client-side (i.e. resolver-

[8]Undetermined records correspond to records resulting from a non-DNSSec lookup [30].

side) changes to address DNS problems. While CoDNS involves resolvers co-operating amongst each other to mask resolver-side issues, we propose that resolvers use local storage to insure themselves (and their clients) against DoS attacks on DNS.

There have also been studies to determine the characteristics of the existing DNS architecture. Jung et. al. [11] use DNS traces to study client-perceived DNS performance and the effectiveness of client caching. They found name accesses to be heavy-tailed which also shows up in our measurements as the diminishing returns of increasing the stale cache size. [16] studied both the deployment patterns and the availability of DNS name servers while [4] measured the performance of the E root-server and observed instances of DoS attacks wherein the root-server was used as reflector.

6. FUTURE WORK

This paper presents a very simple modification to the caching behavior of DNS resolvers. A preliminary evaluation based on DNS-traces collected at Cornell University shows that stale records can be quite effective in mitigating the impact of DoS attacks on DNS. While the proposed modification certainly has some drawbacks, the cost-benefit ratio, especially given the frequency and the impact of DoS attacks, appears to favor the use of the stale cache. However, a few aspects of our proposal require more work. For instance, privacy concerns implied that we had to anonymize the collected DNS traces and hence, were not able to study the DNS records that would have been inaccurate had they been used as stale records in the face of an attack. Specifically, we would have liked to determine if this was due to load-balancing across nameservers and if the clients would still have been able to access the desired resource. We are in the process of obtaining the relevant part of the unanonymized trace to answer this and similar questions.

We are currently implementing the proposed modification into dbjdns [3], a popular DNS resolver. We also intend to explore the possibility of implementing this as an add-on to the CoDNS resolution service [19] running on PlanetLab [5]. Apart from clearing up the implementation issues, such an exercise would help us analyze the advantages of maintaining a stale cache in the face of actual attacks (which occur frequently enough to make this exercise worthwhile!).

Acknowledgements

We would like to thank Larry Parmelee at CFS for his help and patience with the DNS collection process. We are also grateful to Paul Vixie at ISC for helpful discussions on why this proposal should "not" be incorporated in DNS resolvers.

7. REFERENCES

[1] H. Balakrishnan, K. Lakshminarayanan, S. Ratnasamy, S. Shenker, I. Stoica, and M. Walfish, "A Layered Naming Architecture for the Internet," in *Proc. of ACM SIGCOMM*, 2004.

[2] H. Ballani and P. Francis, "A Simple Approach to DNS DoS Mitigation," in *Proc. of workshop on Hot Topics in Networks*, Nov 2006.

[3] D. J. Bernstein, "djbdns: Domain Name System Tools," Apr 2008, http://cr.yp.to/djbdns.html.

[4] N. Brownlee, k claffy, and E. Nemeth, "DNS Measurements at a Root Server," in *Proc. of Globecom*, 2001.

[5] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman, "PlanetLab: An Overlay Testbed for Broad-Coverage Services," *ACM SIGCOMM Computer Communication Review*, vol. 33, no. 3, July 2003.

[6] E. Cohen and H. Kaplan, "Proactive Caching of DNS Records: Addressing a Performance Bottleneck," in *Proc. of Symposium on Applications and the Internet*, 2001.

[7] R. Cox, A. Muthitacharoen, and R. T. Morris, "Serving DNS using a Peer-to-Peer Lookup Service," in *Proc. of IPTPS*, 2002.

[8] T. Deegan, J. Crowcroft, and A. Warfield, "The Main Name System: An Exercise in Centralized Computing," *SIGCOMM Comput. Commun. Rev.*, vol. 35, no. 5, 2005.

[9] M. Handley and A. Greenhalgh, "The Case for Pushing DNS," in *Proc. of Hotnets-IV*, 2005.

[10] T. Hardy, "RFC 3258 - Distributing Authoritative Name Servers via Shared Unicast Addresses," April 2002.

[11] J. Jung, E. Sit, H. Balakrishnan, and R. Morris, "DNS performance and the effectiveness of caching," *IEEE/ACM Trans. Netw.*, vol. 10, no. 5, 2002.

[12] J. Kangasharju and K. W. Ross, "A Replicated Architecture for the Domain Name System," in *Proc. of INFOCOM*, 2000.

[13] R. Ladin, B. Liskov, L. Shrira, and S. Ghemawat, "Providing high availability using lazy replication," *ACM Trans. Comput. Syst.*, vol. 10, no. 4, 1992.

[14] P. Mockapetris, "RFC 1035, DOMAIN NAMES - IMPLEMENTATION AND SPECIFICATION," Nov 1987.

[15] D. Oppenheimer, B. Chun, D. Patterson, A. C. Snoeren, and A. Vahdat, "Service placement in a shared wide-area platform," in *Proc. of the USENIX '06 Annual Technical Conference*, 2006.

[16] J. Pang, J. Hendricks, A. Akella, R. D. Prisco, B. Maggs, and S. Seshan, "Availability, usage, and deployment characteristics of the domain name system," in *Proc. of Internet Measurement Conference*, 2004.

[17] V. Pappas, B. Zhang, E. Osterweil, D. Massey, and L. Zhang, "Improving DNS Service Availability by Using Long TTLs," draft-pappas-dnsop-long-ttl-02, June 2006.

[18] V. Pappas, D. Massey, and L. Zhang, "Enhancing DNS Resilience against Denial of Service Attacks," in *Proc. of Conference on Dependable Systems and Networks (DSN)*, 2007.

[19] K. Park, V. Pai, L. Peterson, and Z. Wang, "CoDNS: Improving DNS Performance and Reliability via Cooperative Lookups," in *Proc. of USENIX OSDI*, 2004.

[20] D. S. Phatak, "Spread-Identity mechanisms for DOS resilience and Security," in *Proc. of SecureComm*, 2005.

[21] L. Poole and V. S. Pai, "ConfiDNS: leveraging scale and history to improve DNS security," in *Proc. of the 3rd USENIX Workshop on Real, Large Distributed Systems (WORLDS)*, 2006.

[22] V. Ramasubramanian and E. G. Sirer, "The Design and Implementation of a Next Generation Name Service for the Internet," in *Proc of ACM SIGCOMM*, 2004.

[23] D. Soring, "Using lightweight checkpoint/recovery to improve the availability and designability of shared memory multiprocessors," Ph.D. dissertation, University of Wisconsin-Madison, 2002.

[24] M. Theimer and M. B. Jones, "Overlook: Scalable Name Service on an Overlay Network," in *Proc. of ICDCS*, 2002.

[25] H. Yang, H. Luo, Y. Yang, S. Lu, and L. Zhang, "HOURS: Achieving DoS Resilience in an Open Service Hierarchy," in *Proc. of Conference on Dependable Systems and Networks (DSN)*, 2004.

[26] "Microsoft DDoS Attack, NetworkWorld," Jan 2001, http://www.networkworld.com/news/2001/0125mshacked.html.

[27] "Root Server DDoS Attack, RIPE Mail Archive," Nov 2002, https://www.ripe.net/ripe/maillists/archives/eof-list/2002/msg00009.html.

[28] "Akamai DDoS Attack, Internet Security News," Jun 2004, http://www.landfield.com/isn/mail-archive/2004/Jun/0088.html.

[29] "UltrDNS DDoS Attack, Washington Post," May 2005, http://blog.washingtonpost.com/securityfix/2006/05/blue_security_surrenders_but_s.html.

[30] "CISCO DNSSEC page," Aug 2006, http://www.cisco.com/web/about/ac123/ac147/archived_issues/ipj_7-2/dnssec.html.

[31] "Internet Systems Consortium," Aug 2006, http://www.isc.org/.

[32] "SLASHDOT: Providers Ignoring DNS TTL?" Aug 2006, http://ask.slashdot.org/article.pl?sid=05/04/18/198259&tid=95&tid=128&tid=4.

[33] "SiteKey at Bank of America," Jul 2007, http://www.bankofamerica.com/privacy/sitekey/.

[34] "DNS - What do big sites do?" Aug 2008, http://forum.powweb.com/archive/index.php/t-54961.html.

[35] "nonamed - Man page," Aug 2008, http://www.minix3.org/previous-versions/Intel-2.0.3/wwwman/man8/nonamed.8.html.