Enhancing DNS Resilience against Denial of Service Attacks

Vasileios Pappas
T.J. Watson Center
IBM Research
vpappas@us.ibm.com

Dan Massey
Computer Science Department
Colorado State University
massey@cs.colostate.edu

Lixia Zhang
Computer Science Department
UCLA
lixia@cs.ucla.edu

Abstract

The Domain Name System (DNS) is a critical Internet infrastructure that provides name to address mapping services. In the past few years, distributed denial of service (DDoS) attacks have targeted the DNS infrastructure and threaten to disrupt this critical service. In this paper we show that the existing DNS can gain significant resilience against DDoS attacks through a simple change to the current DNS operations, by setting longer time-to-live values for a special class of DNS resource records, the infrastructure records. These records are used to navigate the DNS hierarchy and change infrequently. Furthermore, in combination with a set of simple and incrementally deployable record renewal policies, the DNS service availability can be improved by one order of magnitude. Our approach requires neither additional physical resources nor any change to the existing DNS design. We evaluate the effectiveness of our proposed enhancement by using DNS traces collected from multiple locations.

Keywords: DDoS, DNS, resilience, caching

1 Introduction

The Domain Name System (DNS) [16] provides name services for the Internet. It maps hostnames to IP addresses and also provides services for a growing number of other applications, such as mapping IP addresses to geographic locations or directory services for legacy telephony applications. Furthermore, protocols such as SMTP and SIP depend on the DNS in order to route messages through appropriate application level gateways. As a result, the availability of the DNS can affect the availability of a large number of Internet applications. Ensuring the DNS data availability is an essential part of providing a robust Internet.

Due to its hierarchical structure, the DNS availability depends on a small number of servers that serve the root and other important top level domains. A number of distributed denial of service (DDoS) attacks have been directed against these top level DNS name-servers in recent years [2, 3, 5, 7]. The impact on overall DNS availability is debatable [1, 4], but some attacks did succeed in disabling the targeted DNS servers and resulted in parts of the Internet experiencing severe name resolution problems. Overall, attacks can potentially threaten the DNS availability and effectively threaten the availability of the Internet itself.

We have developed a simple approach that can effectively enhance the DNS resilience against DDoS attacks. We identify a special class of DNS records called *infrastructure records*, which store data for DNS infrastructure components (namely the name-servers). DNS resolvers use the infrastructure records to navigate the DNS hierarchy. The presence of the infrastructure records in DNS local caches can greatly improve the resilience of the DNS in the presence of failures. In this paper we propose and evaluate two methods for caching infrastructure records for longer periods of time. First, we propose to assign a much longer TTL value for the infrastructure records than the data records. This is feasible because, generally speaking, the infrastructure records change less frequently than other DNS data records. Second, we propose a set of simple record renewal policies. Our analysis shows that these two changes can improve DNS service availability during a DDoS attack by one order of magnitude.

The main benefit of our approach is that it is operationally feasible and immediately deployable by either large or small zones. In contrast, the currently deployed solution of shared unicast addresses [14] aims at absorbing the attack load by installing a large number of name-servers. This solution is suitable for large zones, such as the root and the top level domains, that can afford the cost. Smaller zones may not be able to afford adding a large number of name-servers. Other solutions proposed by the research community [10, 21, 20, 12, 11] address the problem of DDoS attacks against DNS by introducing major protocol changes or by redesigning the whole system. Although some of them are considered incrementally deployable, their adoption is hindered by the operators' reluctance to introducing major changes in an operational system. Our approach requires no protocol changes while achieving similar levels of resilience against DDoS attacks.

The rest of the paper is structured as follows. Sections 2 and 3 review the basic DNS concepts and the threat posed by DDoS attacks. Section 4 presents our TTL guidelines and caching enhancements. Section 5 evaluates of our approach using a set of real DNS traffic traces. Section 6 discusses some issues related to other attack strategies. Section 7 reviews related work and Section 8 concludes the paper.

2 Domain Name System

In DNS parlance, the name space is divided into a large number of zones. Roughly speaking, each zone is authoritative for the names that share the same suffix with the zone's name. A zone can also delegate part of its namespace to another zone, referred as a child zone. For example, the *ucla.edu* zone has delegated the *cs.ucla.edu* namespace to create a child zone. This delegation procedure results in an inverted tree structure with each node being a zone and each edge representing a delegation point. The *root* zone resides at the top of this tree structure. Generic top-level domains (gTLD), such as *edu*, and country code top-level domains (ccTLD) appear directly below the *root*. Figure 1 displays a part of the DNS tree structure with some functional elements introduced in the next two paragraphs.

Each zone stores the resource records (RRs) associated with names under its authority. There are several different types of RRs with the most common one being the address (A) resource record used to map names to IPv4 addresses. Each RR has a time to live value (TTL) that specifies the maximum lifetime when caching the record. For example, the IP address of *www.ucla.edu* is stored in an A resource record and has a TTL value of 4 hours.

All the RRs that belong to a zone are available from a set of DNS servers called authoritative name-servers (ANs) for the zone. The ANs are identified by a special type of resource record, the name-server (NS) resource record. The NS records for a zone are stored at the zone itself and also at its parent zone. Each NS record points to the name of the authoritative name-server (rather than its IP address) and thus one needs both the NS and A records of the server in order to contact a zone. We call the set of NS and A records that are associated with the ANs *infrastructure resource records* (IRRs). IRRs are used in order to construct the DNS tree structure.

Client applications typically retrieve a desired RRs by querying a stub-resolver (SR), a DNS element which is implemented in every operating system. An SR typically forwards the query to a special type of server, called caching server (CS) and the CS obtains the desired RR. More specifically, the CS obtains RR's from zone Z by querying Z's ANs. The CS knows Z's ANs either because it has previously cached Z's IRRs or by querying Z's parent zone. The parent zone knows the Z's IRRs because it is required to have a copy of Z's IRRs. If the CS does not know the IRRs for Z or Z's parent, it repeats finds the nearest ancestor zone for which it has the IRRs. Every CS is hard-coded with the IRRs of the *root* zone and thus can always start at the root zone if no better IRRs are known. A CS caches each RR that it learns for a period of time equal to the TTL value of the record. Thus, it can reply back to a SR either with information that is locally cached or with information that is retrieved directly from an AN.

3 Threat Assessment of DDoS Attacks

A successful Distributed Denial of Service (DDoS) attack against the DNS offers potential for a high "pay-off". Almost every Internet application utilizes the DNS and an attacker can potentially achieve a DoS attack against many services and many locations by disabling the DNS. The DNS tree structure also seems to introduce critical points of failure such as DNS root or DNS top level domains. Well known DDoS attacks have been launched against the DNS *root* and other top level zones [2, 3, 5, 7]. A simplistic view suggests a successful attack against the DNS root servers could cripple the Internet.

However, both the vulnerabilities and the potential impacts are more complex than the simplistic view suggests. A more informed analysis must take the various DNS components such as redundant servers and caches into account. A DNS zone can be served by a large number of DNS name-servers. Protocol limitations have fixed the number of IPv4 root server addresses at 13, but techniques such as shared unicast addresses are being used to increase the actual number of servers. Second, even if the attack successfully disables the name-servers of a targeted zone, it may have limited effect on the DNS service given that cached records will continue to be served.

3.1 Launching a Successful Attack

This paper considers DDoS attacks that target the authoritative name-servers for a zone. We assume the attack objective is to disable the DNS resolution of all zones *below the targeted zone*. Many high level zones such as the root and top level domains (com, net, edu, uk, cn, and so forth) primarily provide referrals to other zones lower in the tree. For example, an attack against the *edu* authoritative servers is intended to prevent resolvers from reaching any of the zones below *edu*. Most of the well known large-scale DNS attacks fall under this category. Section 6 discusses other types of attacks.

Whether an attack succeeds it depends on both the resources of the attacker and the defender. DDoS attacks can easily succeed if a zone is served by a small number of servers. Currently, most zones use two or three name-servers and are thus vulnerable to relatively small attacks. Larger and more critical zones tend to deploy more servers, but their number still ranges in the order of tens or hundreds. Unfortunately, some of the "botnets" controlled by attackers include hundreds of thousands of "drone" machines [6] and can potentially be successful even against the few zones that deploy anycast enabled name-servers [14]. Overall, the situation creates an arms race between attacker and defenders with both sides seeking enough resources to overwhelm the other.

3.2 Factors Affecting Attack Impact

While it may be feasible to launch a successful DDoS attack against a zone, the attack will not necessarily have any impact on Internet applications. There are mainly three factors that affect the end-user experience of a successful

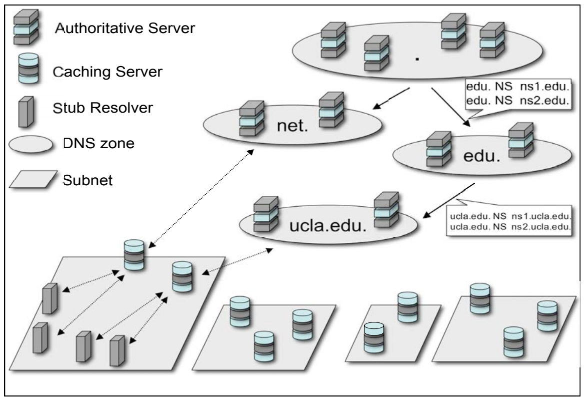

Figure 1. Overview of DNS elements.

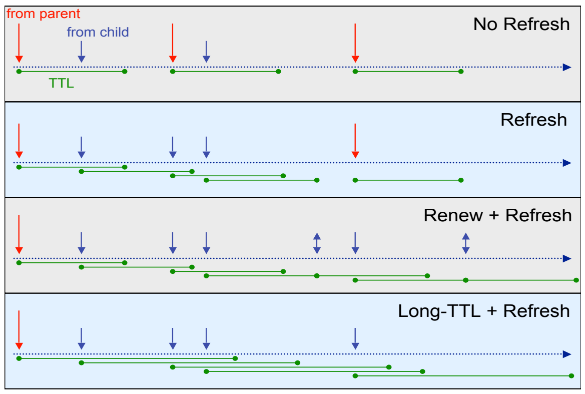

Figure 2. Proposed schemes showcase.

DDoS attack against DNS:

Position of the Target Zone If a zone is a *stub* in the DNS tree structure, i.e. it is not used in order to access the name-servers for other zones, then the attack will only affect the names defined in the targeted zone. Note that a *leaf*-zone, i.e. a zone that has no children, is not necessarily a stub-zone. In many cases leaf-zones are used in order to resolve the IP address of other zones' name-servers. In essence, the number of descendant zones that can be resolved through a zone can indicate the severity of a successful attack against that zone. If one considers only the position of a zone, then the *root* zone would be considered the most important zone given that it is needed in order to resolve all other zones.

Popularity of the Target Zone The impact of a successful attack also depends on the frequency of *referrals* provided by the target zone. The number of referrals depends partly on the the number of child zones below the parent zone. But it is also influenced by the popularity of the child zones, i.e. the number of caching servers that query them. In addition, the TTL values of the child zones IRRs can also influence the frequency of referrals. To illustrate this, consider an attack that targets the root zone. Every zone is a descendant of the root, but an attack against a popular TLD may be more catastrophic than attack against the *root*. There are only around two to three hundreds zones directly below the root, compared to millions of zones directly below the largest TLDs. Furthermore, the zones directly below the root tend to have IRR records with relatively TTL values. In contrast, many zones below the TLDs have shorter TTL values for the IRR records. As a consequence, the caching servers query TLDs more frequently than they query the *root* zone.

Resource Record Caching The impact of a successful DDoS attack is also affected by resource record caching. Even if some zones are unavailable due to a DDoS attack, the records defined at these zones may be cached in some caching servers and thus still accessible. Clearly, the higher the TTL value for a record and the more popular the record may be, the higher the probability of being cached. In essence, the use of resource record caching allows end-user application to still function even though a zone's authoritative name servers are not accessible. In a similar manner, the caching of infrastructure records can allow a caching server to access a zone's name-servers, even if an ascendant zone is not accessible due to a DDoS attack. While the caching of data records plays a role, the caching of infrastructure records plays a more prominent role in mitigating DDoS attacks. The presence of a zone's infrastructure record in the local cache allows the resolution of all the names defined inside the zone and also allows the resolution of all the descendant zones even when the parent zone (or any other ascendant zone) is unavailable.

4 Enhancing DNS Resilience

Previous efforts [10, 21, 20, 12, 11] of enhancing the DNS resilience against DDos attacks focus on reducing or eliminating critical points of failure in the DNS hierarchy. Either they introduce new ways of resolving the name-servers [21, 12] which do not coincide with the name-space tree structure, or they abandon completely the concept of name-servers [10, 20, 11], at least in the way that they are currently defined. As a consequence these previous proposals require substantial changes in the DNS infrastructure.

In contrast, our approach of enhancing the DNS resilience against DNS attacks focuses on zone popularity and caching. We introduce changes only at the caching servers and we do not require any modifications to the underlying DNS infrastructure. Our enhancements aim at forcing the caching servers (CS) to maintain for longer periods of time copies of the infrastructure resource records (IRRs) for the zones that they use the most frequently. In consequence, the number of referral, i.e queries that a CS sends at a parent zone in order to resolve names belonging to a child zone, can potentially decrease. In this way the popularity of a zone depends mainly on the number of queries generated for the names that belong to the zone and less on the number of queries generated for names belonging to a child zone.

We provide the following example in order to elaborate more on the basic idea of our approach. Let's consider a successful attack against the *edu* zone. In that case, a CS cannot resolve a zone that resides just below the *edu* if it does not have the IRRs for that zone. The probability of having the IRRs for the zone cached are increased when a zone is more popular or when the IRRs have a longer TTL.

In order to increase the probability of having the IRRs, a CS can "artificially" increase the popularity of the zone by querying it whenever the IRRs are ready to expire or the zone's administrator can increase the TTL value of the zone's IRRs. Note that in the extreme case, a CS can indefinitely query for the IRRs and the zone's administrator can unlimitedly increase the TTL value. While both of these extreme cases can lead to the best resilience against DDoS attacks, they are not desirable given that the first can introduce a considerable message overhead and the second can potentially introduce IRRs inconsistencies.

Next we present three feasible techniques that can be used in order to increase the probability of having the IRRs for a zone locally cached. Figure 2 provides their graphical representation, corresponding to the IRRs of a zone, which are cached inside a CS. The longer arrows represent referral replies from the parent zone, while the shorter arrows represent replies from the child zone. The horizontal lines represent the period of time for which the IRRs are cached. We should point out that the proposed methods *affect only the $IRRs$* and not any other record, e.g. as the various data records.

TTL Refresh In order to explain how *TTL refresh* works we first need to provide some specific details on how CSs learn and cache these records. A CS learns the IRR for a zone Z initially from Z's parent zone (P). The IRR for Z are included in the authority and the additional sections of the referral sent by P's name-servers. The CS caches these records locally and then contacts one of Z's name-servers to obtain the desired data. The reply from Z's name-servers also includes a the IRR for Z in the authority and additional sections of the reply. The CS ought to replace the cached IRR that come from the parent with the IRR that come from the child zone when they are not identical [13]. This establishes initial IRR records for zone Z.

Additional queries for names in Z can make use of the IRR data and go directly to Z's name-servers. Each query to a Z name-server will include a copy of Z's IRR data and TTL refresh uses this new data to refresh the TTL on Z's IRR. For example, a query for *www.ucla.edu* will result in the cache learning both the requested *www.ucla.edu* record and the IRR data from *ucla.edu*. If the IRR for Z has not yet expired, a later query for *ftp.ucla.edu* will go directly to the Z name-servers and the response will include both the *ftp.ucla.edu* record and another copy of the *ucla.edu* IRR data. This new copy of the IRR information could be used to refresh the cached copy of all *ucla.edu* IRR, but many popular DNS caching server implementations do not refresh the TTL value for the IRR. Note that the *www.ucla.edu* record may expire before the *ucla.edu* IRR and thus even another query for *www.ucla.edu* could be used to refresh the IRR.

This simple modification is very effective for the zones that a CS visits frequently. Assuming a CS sends some query to zone's name-servers before the IRR expires. Every query resets the TTL and the zone's IRR will be always locally cached. In contrast without *TTL refresh* the CS has to visit the parent zone when the IRR expires. The difference is shown in Figure 2. If the CS does not refresh the TTL, the IRRs expire after the first two queries and thus the third one triggers a query at the parent zone. In contrast with the refresh the CS needs to query the parent zone only at the fifth query.

TTL Renewal The limitation of the *TTL refresh* method is that it does not work well for the popular zones that the caching server queries in a less regular fashion. A CS will not have the IRRs for the zone if the time between queries to Z exceeds the TTL on Z's IRR. This is the case in Figure 2. The CS does not have the IRRs during the fifth query. The *TTL renewal* method aims at filling that gap. In essence, it allows the IRRs for the most popular zones to stay in a CS for longer periods of time, compared to the *TTL refresh* method. This is done by refetching and then renewing the TTL of the IRRs just before they are ready to expire. This is shown with the double-head arrow in Figure 2.

In order to renew the IRRs only for the most popular zones we consider four different renewal policies. The basic idea behind these policies is that each zone is assigned a certain credit c which defines the number of times the zone's IRRs can be renewed after they have expired. The assignment of credit is different for the four renewal policies but it mimics either the last recently used (LRU) or the least frequently used (LFU) cache renewal policies. More specifically we consider the following policies:

- LRU^c: This policy *sets* a zone's credit to c every time that the zone is queried. Also every time that the zone's IRRs are about to expire the credit is decreased by one and the IRRs are re-fetched. In essence, with this policy the IRRs stay in the cache for an additional period of time that is equal to $c * TTL$. It resembles an LRU replacement policy because the IRRs that haven't been recently used are the ones that expire first.
- LFU^c: This policy *adds* a credit of c to the zone's current credit, whenever the zone is queried. Again, the credit is decremented by one whenever the zone's IRRs are re-fetched. Given that for the most popular zones the credit may indefinitely increase, we consider a maximum credit M. If the current credit reaches M, then it stops increasing. This policy resembles an LFU policy for the reason that the IRRs that expire first are the ones that are not frequently used.
- $A - LRU^c$: This is an *adaptive* version of the LRU^c policy. The need for an adaptive policy arises from the fact that different zones have IRRs with different TTL values. Thus the additional time that their IRRs stay in the cache may vary. In order, to make this time equal for all the zones we consider a version of the LRU^c policy in which the credit adapts based on the TTL value. More specifically, the assigned credit is equal to $86400 * c/TTL$, where 86400 is the equivalent of one day in seconds. Thus, for example if the

$c = 3$ then the credit causes all the IRRs to stay in the cache for three days additionally.

- $A - LFU^c$: Similarly we define an *adaptive* version of the LFU^c renewal policy. Again the credit adapts to the TTL value of each zone's IRRs and it is equal to $86400 * c/TTL$. Furthermore, there still exist a maximum credit M that the current credit cannot exceed, which prevents the credit of very popular zones from increasing indefinitely. The benefit of the adaptive LFU (as well as LRU) is that zones stay in the cache for an additional period of time that is independent of their IRRs TTL values.

Finding an optimal policy [9] requires the presence of an oracle that could foresee future queries, which makes it non-practical, but our evaluation shows these simple and easily implementable policies can be very effective.

Long TTL Instead of renewing the IRRs after they expire, one can achieve the same results by simply increasing the TTL value of the IRRs. For example, assuming a current TTL value of one day, then increasing the TTL value to 3 days provides the same resilience as the LRU^2 renewal policy. Note the proposed increase of the TTL value is only for the IRRs, and not for end-host records. Thus this scheme does not effect CDN or load balancing schemes that rely on short TTL values for end-hosts. While current TTL values range from some minutes to some days, most zones have at TTL value less or equal to 12 hours. The main benefit in increasing the TTL values is that it does not require any changes at the caching servers and it can be enforced directly by the zone administrators. In addition, this modification reduces overall DNS traffic and improves DNS query response time since costly walks of the DNS tree are avoided.

But if the IRR changes at the AN's, the cached copy will be out of date. Increasing the TTL value can increase the time during which cached IRR differs from the actual IRR stored at the ANs. Fortunately, IRRs change infrequently [12]. Furthermore, DNS works as long as one name-server in the cached IRR is still valid. The penalty paid for querying an obsolete name-server is a longer resolution time. If a server fails to respond, the next server in the IRR is queried. Once any response is received for a valid server, the IRR set is updated and inconsistency is resolved. In the worst case, all servers in the old IRR fail to respond and the parent zone must be queried to reset the IRR.

Combinations The above proposed modifications can work independently as well as in parallel with each other. Clearly, by combining two methods one can complement their abilities in improving the DNS resilience against DDoS attacks. Furthermore, combining them reduces their overhead, as it is shown in a later section. Apart from the above performance benefits, there is an additional and maybe more important operational benefit. The first two methods, allow any DNS client to enhance its resilience against the DDoS attacks that target the DNS, without requiring any modifications at remote sites, i.e. infrastructure changes. The last method allows any DNS zone to improve its resilience to DDoS attacks that target any of its ascendant zones, without requiring any modifications in other zones or modifications at the DNS clients. In essence, the above methods provide the power both to the DNS clients and the DNS operators to enhance the DNS resilience against DDoS attack by introducing only local changes.

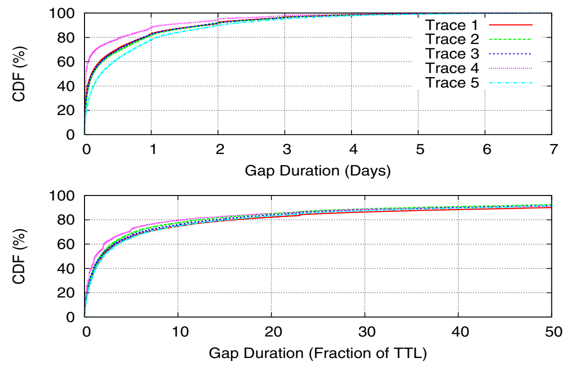

Figure 3. Time-Gap Duration (CDF)

5 Evaluation

In order to evaluate the effectiveness of these CS modifications we collected a number of DNS traces. The traces captured all the queries that were generated by stub-resolvers (SRs) and queries sent by the caching servers (CSs). The collected traces come from a number of different organizations (five US universities) and are grouped based on the caching servers (six servers). They were collected around the same period of time, and their durations ranged from one week (for a CS with very large query load) to one month. Table 1 gives some additional details for each trace, such as number of SRs (clients), the number of queries generated by SRs (requests in), the number of queries sent by the CSs to the name-servers (requests out), the number of distinct names appearing in the queries (names) and the number of distinct zones queried (zones).

First, we used these traces to measure the time duration between the expiration of a zone's IRR and the time the next query was sent to the zone. The length of this time-gap is indicative of how well the proposed schemes can work; if the time gap is long, the IRR may still expire from the cache even if it is refreshed, renewed or its TTL value is increased. Figure 3 gives the cumulative distribution function (CDF) for the time-gap duration. The upper graph gives the duration of the gaps in an absolute time (in days), while the lower graph gives the duration of the gaps as a fraction of the zone's IRR TTL value. For example a fraction of 10 it means that the gap is 10 longer compared to the TTL value. It is interesting to note that in absolute time almost all gaps are less than 5 days long, while the gaps duration varies largely when compared with the TTL values. The reason is that the IRRs TTL values vary greatly, from some minutes to some days, which leads to a greater variability in the relative gap time.

Trace	Organization	Location	Duration	Clients	Requests In	Requests Out	Names	Zones
TRC1	University	USA	7 Days	339	8480402	1930250	556809	200531
TRC2	University	USA	7 Days	486	1400490	566507	193250	45802
TRC3	University	USA	7 Days	915	3148919	1038870	306053	87893
TRC4	University	USA	7 Days	455	15061455	1989997	551617	50531
TRC5	University	USA	7 Days	291	3135620	413648	87863	44502
TRC6	University	USA	1 Month	821	3461948	1153739	117540	55632

Table 1. DNS Traces Statistics

Aside from the above simple measures, the main use for the traces is as a query workload for our simulations. The simulator also took as an input the part of the DNS tree structure that was needed in order to resolve all the zones that were captured in the traces. This part of the DNS structure was acquired in an off-line stage, by actively probing the DNS. As such the simulated DNS structure represents the real DNS structure that appeared during the period of time that we collected the traces. Furthermore, the IRR values used in the simulator are the actual TTL values for the zones during that period of time. We used the simulator in order to evaluate both the effectiveness of the proposed techniques in enhancing the resilience of the DNS against DDoS attacks and in order to gauge the overhead introduced by them.

5.1 Resilience against DDoS Attacks

In order to measure the resilience of the current DNS as well as of the proposed schemes we considered the following experiment. For the first six days we assume that all zones work normally and at the beginning of the seventh day a DDoS attack completely blocks the queries sent to the root zone and the top level domains. The attack duration ranges from 3 to 24 hours. Then we measure the percentage of queries during the attack period that fail to resolve due to the attack. We measure both the failed queries sent by the SRs to the CS, as well as the failed queries sent from the CSs to the ANs. Note the following subtle difference between queries from the SRs and queries from the CSs. The failed queries from the SRs captures the actual impact of the DDoS attack on the end-users, while the queries by the CSs captures the impact of the DDoS attack on caching servers attempting to access the DNS.

We measure the number of failed queries for the following systems: A vanilla system that captures the behavior of the current DNS, a system that implements only *TTL-refresh*, a system that implements both *TTL-refresh* and *TTL-renew* for each of the four renewal policies, a system that combines both *TTL-refresh* and *long-TTL* and a system that implements all three (with LFU as a renewal policy).

5.1.1 Vanilla DNS

Figure 4 shows the percentage of queries that fail to resolve during the time that the DoS attack takes place when simulating the current DNS. The upper graph shows the percentage of failed queries that are sent by the SRs and the lower graph shows the percentage of failed queries that are sent by the CSs. The figure provides results for the first five traces and for attacks that last from 3 to 24 hours. Clearly, when the attack duration increases then the percentage of failed queries increases for the reason that more and more records start to expire. These records include both end-host records, such as A RRs, as well as IRRs. Moreover, we see that the percentage of failed queries from CSs is higher than the percentage of failed queries from SRs. The reason is that queries from SRs can be answered locally if they are cached at the CSs, while all queries from CSs have to query the DNS infrastructure.

Furthermore the figure shows that the percentage of failed queries varies a lot for the different traces when considering queries from SRs, while it is almost the same for queries generated by the Cs. We speculate that this is due to the fact that the number of parameters that affect the success rate of queries from SRs are much larger compared to queries from CSs. For example, the query distribution, the distribution of TTLs for end-host RRs, the number of SRs that use the same CS as well as the overlap of interest between different SRs affect the success rate of queries generated by the SRs. On the other hand, the success rate of queries generated by the CSs depends only on the distribution of queries and the distribution of TTLs for IRRs. For that reason, when we compare the effectiveness of our proposed schemes we compare it against each trace separately, rather that averaging across all traces.

5.1.2 TTL Refresh

Figure 5 shows the percentage of failed queries for the same scenario of DDoS attacks that we used before, when the *TTL-refresh* method is implemented. The figure includes two graphs that show the same type of results as in Figure 4. Note also that we use the same type of figure to presents the resilience to DDoS attacks for all the consequent schemes presented in this paper. Clearly both graphs shows that by implementing the refresh of IRRs TTLs the resiliency of the DNS can greatly improve. For most cases this modification leads to a percentage of failed queries that is at least 50% lower compared to the current system.

5.1.3 TTL Refresh and Renewal

Figures 6, 7, 8 and 9 show the DDoS attack resiliency achieved when implementing the LRU^c, LFU^c, $A-LRU^c$ and $A-LFU^c$ *TTL-renewal* policies, in combination with the *TTL-refresh* method. All graphs show results for the six hours attack and contrast them to the resiliency of the current DNS for the same attack. We consider three

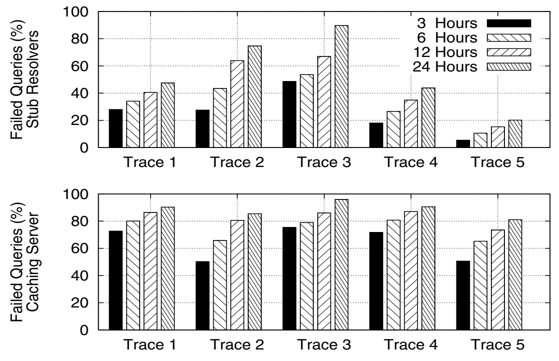

Figure 4. Vanilla DNS

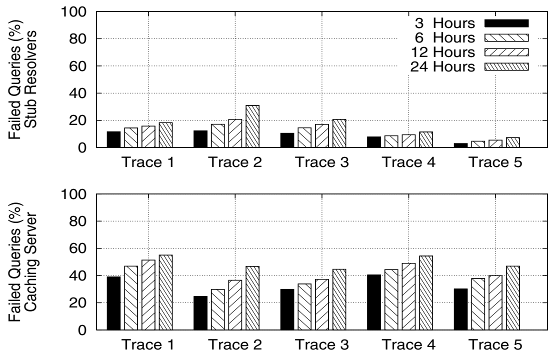

Figure 5. TTL Refresh

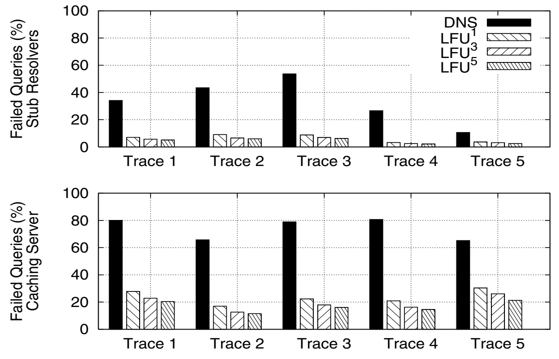

Figure 6. TTL Refresh + Renew (LRU)

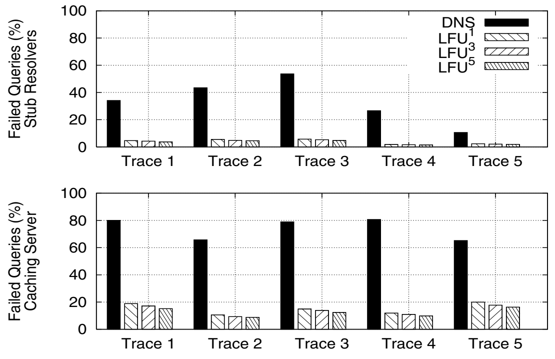

Figure 7. TTL Refresh + Renew (LFU)

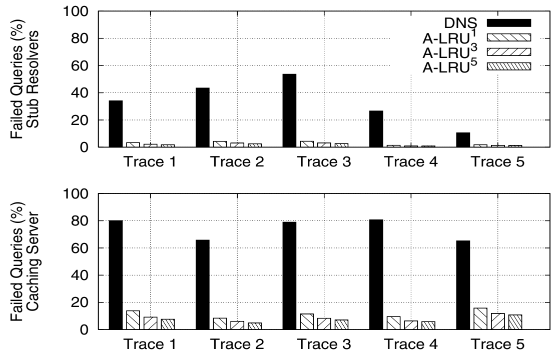

Figure 8. TTL Refresh + Renew (A-LRU)

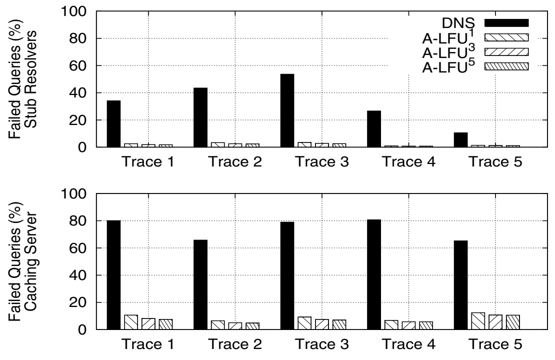

Figure 9. TTL Refresh + Renew (A-LFU)

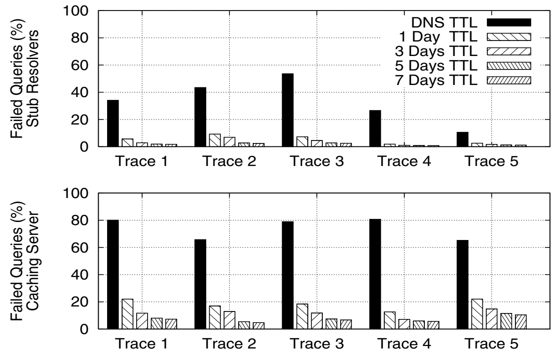

Figure 10. TTL Refresh + Long-TTL

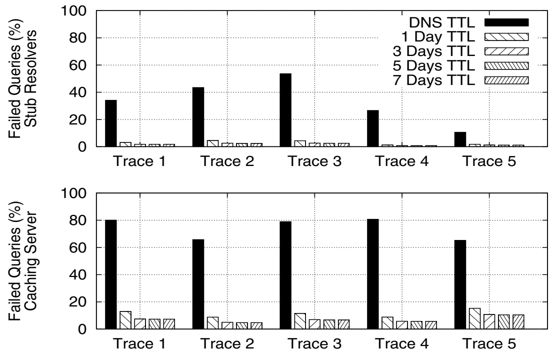

Figure 11. TTL Refresh + Renew + Long-TTL

different values for the credit c: 1, 3 and 5. The figures show that all the schemes perform almost equally with the performance becoming slightly better with the schemes in the following order: $LRU^c < LFU^c < A - LRU^c < A - LFU^c$. Intuitively the adaptive policies are better given that they are neutral to the different values of the IRRs TTLs. Furthermore, the LFU policies perform better than the LRU for the reason that they favor the most frequently used zones. In conclusion, the $A - LFU^c$ is shown to work the best: the failure rate for queries generated from the SRs is lower than 2.5%, while the failure rate for queries generated from the CS is lower than 10%. In summary, the combination of TTL *refresh* and *renew* improves the failure one order of magnitude compared to the current DNS.

5.1.4 TTL Refresh and Long-TTL

Figure 10 shows the percentage of failed queries when all zones change their IRRs TTL value to one, three, five and seven days, and the caching server implements the *TTL-refresh* method. We note that the long-TTL scheme achieves the same level of resilience as the most effective IRRs renewal policy ($A - LFU^5$). Furthermore, the figure shows that the a TTL value of five days is almost as good as a TTL value of seven days. The reason is that the time duration between the expiration of an IRR and the next time it is fetched from the zone is almost always less than five days (see Figure 3). Thus, a TTL value of seven days does not yield much more additional benefits in terms of resilience against DoS attacks.

5.1.5 TTL Refresh, Renewal and Long-TTL

Finally, we combine the cache renewal policies with the long-TTL scheme. The benefits of this hybrid approach is that it achieves the resiliency of the best renewal policy, with a much lower overhead (as show in the next section), as well as with smaller TTL values. Figure 11 shows the percentage of queries that fail to resolve when an LFU^3 renewal policy is applied to IRRs with TTL values of one, three, five and seven days. The graphs show that a TTL value of three days is good enough to achieve the maximum possible resilience to DoS attacks, given that longer TTL values do not yield any additional benefits.

5.2 Overhead

Next we consider the overhead due to the proposed modifications. We consider both the message overhead, i.e. the number of additional queries generated by a CS, and the memory overhead due to the additional zones cached at the CSs.

5.2.1 Message Overhead

Caching servers that implement one of the renewal policies can potentially increase the total DNS traffic, due to additional queries issued for re-fetching the IRRs. In contrast both the refresh and long-TTL modifications lead to a lower number of DNS messages. Indeed, Table 2 shows the increase in the number of generated DNS messages for each of the proposed schemes when compared to the current DNS. Negative values indicate a decrease in the number of generated messages.

Scheme	Trace 1	Trace 2	Trace 3	Trace 4	Trace 5
Refresh	-1.891	-0.968	-1.605	-1.044	-1.494
LRU^5	49.378	29.842	38.264	15.145	48.408
LFU^5	64.797	38.492	51.089	27.819	76.257
A-LRU^5	591.813	339.557	487.965	164.646	548.210
A-LFU^5	593.629	340.845	490.028	166.434	554.362
Long-TTL	-14.291	-7.271	-9.942	-6.131	-10.313
Combination	-9.916	-4.177	-6.226	-5.018	-5.436

Table 2. Message Overhead

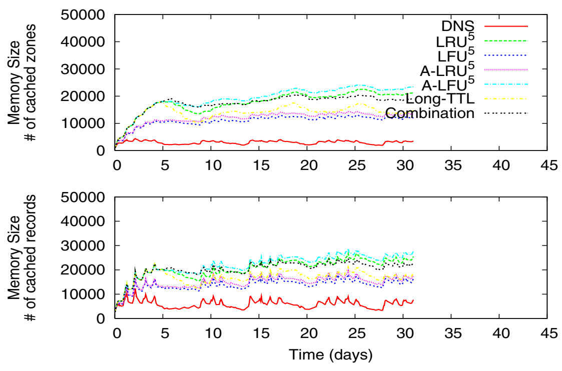

Figure 12. Memory Overhead

The table shows that the adaptive schemes incur a significant overhead, which leads in increasing the DNS traffic by five times in the worst case. That is due to the fact that there is a large number of zones that have very small TTL values (in the order of minutes) which leads to a very large number of re-fetch requests. On the other hand, the non-adaptive renewal policies come with a much smaller cost. They increase the number of generated messages by at most 76%. Given that the DNS traffic is a negligible portion of the overall Internet traffic, we believe that this increase is not significant. More importantly, the table shows that the refresh and the long-TTL schemes, with TTL set to 7 days, lead to a decrease in the DNS related generated traffic. Furthermore, the combined scheme of long-TTL with a value of three days and the LFU^3 policy leads also to a reduction in the generated messages. This in a very promising results given that the hybrid scheme can achieve the resiliency of the most effective adaptive policy, without incurring the high message overhead.

5.2.2 Memory Overhead

The three proposed modifications increase the memory requirements of the CSs, given that they require the caching of IRRs for longer periods of time. On the other hand, as it is shown in Table 1 the total number of zones that appear in a period of one week is in the order of tens to hundreds of thousands. Thus, the additional memory requirements for storing all these IRRs are in the worst case in the order of tens of Mbytes. Figure 12 shows the number of zones and records cached for any given point of time for the one month long DNS trace (TRC6). It also compares these numbers with the number of zones and records cached when using the proposed schemes. Clearly, the additional memory overhead is not an issue for the current systems, given that the proposed caching schemes increase the number of cached objects by two to three times. Note that all other traces showed similar memory overhead.

6 Discussion

In this section we elaborate more on three point that relate to our proposed solutions and that deserve an additional attention.

Deployment Issues Notably there are two practical issues that may arise when deploying our proposed modifications. A first issue is their compatibility with the DNS security extensions. The DNSSEC introduces a number of new records for authentication. Some of them can be classified as new infrastructure resource records (see [18] for more details). Thus under a DNSSEC deployment we extend the refresh, renewal and long-TTL techniques to accommodate these new IRRs. A second practical issue is the ability of parent zones to reclaim delegations. Currently, this happens automatically every time that a caching server gets a referral from the parent zone. The parent zone can point to a new set of servers in the case that the ownership of the zone changes, or it can inform the caching server that the zone does not exist anymore. Given that the goal of our techniques is to reduce the number of these referrals, caching servers may still continue querying to the old zone, as far as the old zone still functions as before. In other words, a non-cooperative owner can potentially maintain the ownership for longer periods of time by not updating the name-servers with the new set of nameservers. Apart from resolving this issue with non-technical means (i.e. legally), we can solve it by forcing the caching servers to periodically query the parent zone (for example every 7 days). In addition, current caching servers do not accept arbitrary large TTL values (more that 7 days). In this way, any new delegation can appear at the caching servers within 7 days in the worst case, i.e. when the old owner is non-cooperative.

Maximum Damage Attack In the evaluation section we considered only one case of attack, that is an attack against the root zone and all the top level domains. This attack is not necessarily the one that can cause the maximum damage (thought, we believe that is close tho the maximum one). We define the maximum damage attack as the one that maximizes the total number of failed queries across all caching servers (or stub-resolvers), for a given budget of attacked zones. Clearly, identifying the maximum damage attack is not practically feasible because it requires the traffic patterns from all stub-resolvers. Furthermore, the result is highly time dependent, meaning that the targets are not the same for different attack starting times, or for different attack durations. Even when considering the traffic pattern from only one caching server, the identification of the maximum damage attack is not straightforward. One approach is to count the number of upcoming queries, and then identify the zone whose children have the maximum number of upcoming queries. The problem with this approach is that failures can happen at any of the descendant zones, and thus it is not enough to count failures only at the children. Furthermore, failures start at a time that depends on the zones IRR TTL value as well as the time that the failure started at the parent zone. These events of cascading failures are difficult to model with known optimization techniques, such as linear or dynamic programming.

Other Types of Attacks In this paper we consider only one class of DDoS attack against the DNS, that is attacks that aim at disabling the resolution of all the descendant zones of the targeted zone. Notably, there are two other broad classes of attacks. First, attacks that aim at disabling the name resolution of the names that belong to the target zone. The goal in this attack is to disable all the services that are provided by the servers "hosted" at the targeted zone. We believe that this type of attack is defensible by adding more name-servers. Name-servers provide a stateless service (they use UDP) and thus it is much harder to overload them compared to overloading the services itself. The reason is that the most popular services are statefull, e.g. they use TCP, and thus if a DDoS attack has the ability to disable the name-servers of a zone, then it has also the ability to disable a service directly (while the reverse is not always true). The second class of attacks that we don't consider are attacks against the caching servers. These attacks are possible, but their damage is locally limited. Furthermore, the simple approach of configuring the stub-resolvers with many caching servers or more sophisticated peer-to-peer approaches [19] can address this type of attacks.

7 Related Work

He have classified the related work in three broader areas. The first two are closer to our work while the third one relates more to the DNS performance issues.

DNS Hardening In recent years there has been a number of proposal for hardening the DNS against DDoS attacks. Yang *et al* [21] have proposed to augment the DNS structure with additional pointers, that are used in order to access children zones. The pointers are stored at sibling zones and are randomly distributed across zones so as an attacker cannot identify them. Handley *et al* [12] have proposed to globally replicate the infrastructure records at every caching server by utilizing a peer-to-peer system. Both approaches assume that DNS operators are cooperative, which may not be practical given the economically competitive environment between them. Parka *et al* [19] have proposed to add a lookup peer-to-peer service between the stub-resolvers and the caching servers. This service can be used in order to defend against DDoS attack that target caching servers. On the other hand, it cannot enhance the DNS resilience against DDoS attack that target name-servers. Recently, Ballani *et al* [8] have proposed to utilize expired records. Caching servers never discard records, even if they have expired, and thus they can utilize them in the case that they cannot retrieve them from the name-servers. Unfortunately, this proposal violates the semantics of record expiration as defined for DNS, which may hinder its adoption.

DNS Redesign Apart from hardening the current system there has been a number of proposals on redesigning the DNS. Cox *et al* [10] have proposed to replace the DNS infrastructure with a peer-to-peer infrastructure implemented on top of a distributed hash table. One benefit of this approach is that all servers become equally important and thus mounting a DDoS against the system has diminishing results. The same study showed that the performance of such a peer-to-peer system, measured by the query response time, was worse than the performance of the DNS, and concluded that such as system may not be a good candidate for replacing the DNS. In a followup study, Ramasubramanian *et al* [20] improved the performance of the lookup service by replicating the most popular records across the peer-to-peer system. Following an opposite direction, Deegan *et al* [11] proposed to replace the DNS with a centralized system. While their objective was to improve many aspects of the system, such as its resilience to configuration errors [17], they argued that a centralized system could also sustain a DDoS attack. All these approaches of redesigning the DNS require a complete overhaul of the DNS structure. The concept of zones becomes relevant only at the name-space level, given that zone operators lose the ability to administer name-servers. We believe that these type of radical changes can delay the adoption of those proposals.

DNS Performance Kangasharju *et al* [15] have proposed to replace the DNS with a globally replicated database, with the goal of improving the response time of DNS queries. Cohen *et al* [9] proposed the use of proactive caching in order to address the same performance problem. It is interesting to note that both schemes can potentially improve the resilience of the DNS against DDoS attacks. On the other hand they are not designed for that purpose and thus they are not optimized for such a task. For instance both schemes deal with end-host records, while, as we argue in this paper, utilizing only infrastructure resource records is more appropriate.

8 Conclusion

Mockapetris [16], the original DNS designer, pointed out that "*The administrator defines TTL values for each RR as part of the zone definition; a low TTL is desirable in that it minimizes periods of transient inconsistency, while a high TTL minimizes traffic and allows caching to mask periods of server unavailability due to network or host problems*".

Considering DDoS attacks are simply one of the means leading to DNS server unavailability, our work reported in this paper is a realization of the above suggestion. We demonstrated not only the effectiveness of using longer TTL value in enhancing DNS resilience, but we also proposed some simple record renewal policies to be used in conjunction with a long TTL value, with a combined results of improving the availability by up to one order of magnitude. Our results can be easily generalized to any hierarchical system [21] that utilizes caching and we debunk the belief that hierarchical systems cannot provide the same level of resilience against DDoS attacks as flat peer-to-peer systems.

References

[1] Events of 21-Oct-2002. http://d.root-servers.org/october21.txt, 2002.

[2] Nameserver DoS Attack October 2002. http://www.caida.org/projects/dns-analysis/, 2002.

[3] UltraDNS DOS Attack. http://www.theregister.co.uk/2002/12/14/, 2002.

[4] DNS FAQ. http://www.cs.cornell.edu/People/egs/beehive/faq.html, 2004.

[5] DoS Attack against Akamai. http://news.com.com/2100-1038_3-5236403.html/, 2004.

[6] Million-PC botnet threatens consumers. http://www.infomaticsonline.co.uk/ vnunet/news/2167474/million-pc-botnet-threatens, 2006.

[7] ICANN Factsheet for the February 6, 2007 Root Server Attack. http://www.icann.org/announcements/factsheet-dns-attack-08mar07.pdf, 2007.

[8] H. Ballani and P. Francis. A Simple Approach to DNS DoS Defense. In *Proceedings of HotNets*, 2006.

[9] E. Cohen and H. Kaplan. Proactive Caching of DNS Records: Addressing a Performance Bottleneck. In *Proceedings of SAINT*, pages 85–94, 2001.

[10] R. Cox, A. Muthitacharoen, and R. Morris. Serving DNS Using a Peer-to-Peer Lookup Service. In *Proceedings of IPTPS*, pages 155–165, 2002.

[11] T. Deegan, J. Crowcroft, and A. Warfield. The Main Name System: An exercise in centralized computing. In *Proceedings of CCR*, pages 5–13, 2005.

[12] M. Handley and A. Greenhalgh. The Case for Pushing DNS. In *Proceedings of HotNets*, 2005.

[13] T. Hardie. Clarifications to the DNS Specification. RFC 2181, 1997.

[14] T. Hardie. Distributing Authoritative Name Servers via Shared Unicast Addresses. RFC 3258, 2002.

[15] J. Kangasharju and K. Ross. A Replicated Architecture for the Domain Name System. In *Proceedings of INFOCOM*, pages 660–669, 2000.

[16] P. Mockapetris and K. J. Dunlap. Development of the Domain Name System. *SIGCOMM CCR*, pages 123–133, 1988.

[17] V. Pappas, Z. Xu, S. Lu, D. Massey, A. Terzis, and L. Zhang. Impact of Configuration Errors on DNS Robustness. In *Proceedings of SIGCOMM*, pages 319–330, 2004.

[18] V. Pappas, B. Zhang, E. Osterweil, D. Massey, and L. Zhang. Improving DNS Service Availability by Using Long TTL Values. Internet Draft, 2006.

[19] K. Parka, V. Pai, L. Peterson, and Z. Wang. CoDNS: Improving DNS Performance and Reliability via Cooperative Lookups. In *Proceedings of OSDI*, 2004.

[20] V. Ramasubramanian and E. Sirer. The Design and Implementation of a Next Generation Name Service for the Internet. In *Proceedings of SIGCOMM*, pages 331–342, 2004.

[21] H. Yang, H. Luo, Y. Yang, S. Lu, and L. Zhang. HOURS: Achieving DoS Resilience in an Open Service Hierarchy. In *Proceedings of DSN*, pages 83–93, 2004.